

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Е.Н. Эгов

РАЗРАБОТКА ПРОГРАММЫ АБСТРАКТНОГО МАГАЗИНА

Практикум по дисциплине
«Технологии программирования»
для студентов направлений подготовки бакалавриата
09.03.03 «Прикладная информатика»,
09.03.04 «Программная инженерия»

Ульяновск
УлГТУ
2020

УДК 004.42(076)
ББК 32.973.26 – 018.2я73
С90

Рецензент:

Ведущий инженер-программист ФНЦП АО «НПО «Марс», канд. тех. наук Радионова Ю. А.

*Рекомендовано научно-методической комиссией
факультета информационных систем и технологий
в качестве практикума.*

Эгов, Евгений Николаевич

С90 Разработка программы абстрактного магазина: практикум для студентов направлений подготовки бакалавриата 09.03.03 «Прикладная информатика», 09.03.04 «Программная инженерия» / Е. Н. Эгов. – Ульяновск : УлГТУ, 2020. – 201 с.

Практикум адресован студентам для выполнения и оформления лабораторных работ по дисциплине «Технологии программирования». Предоставлены варианты заданий средней и высокой сложности. Рекомендации и требования к лабораторным работам разработаны в соответствии с рабочей программой дисциплины. Предназначены для студентов, обучающихся по направлениям 09.03.03 «Прикладная информатика (профиль Прикладная информатика в экономике)» и 09.03.04 «Программная инженерия».

Работа подготовлена на кафедре «Информационные системы».

Пособие может быть использовано преподавателями и студентами других направлений подготовки (специальностей).

УДК 004.42(076)
ББК 32.973.26 – 018.2я73

© Оформление. УлГТУ, 2020
© Эгов Е.Н., 2020

СОДЕРЖАНИЕ

Лабораторная работа №0. Работа в семестре	6
Цель	6
Работа в семестре	6
Правила сдачи лабораторных	10
Правила оформления кода	12
Лабораторная работа №1. Библиотеки	13
Цель	13
Задание	13
Решение	13
Требования	50
Усложненная лабораторная (необязательно)	51
Варианты	52
Лабораторная работа №2. LINQ	56
Цель	56
Задание	56
Решение	56
Требования	64
Усложненная лабораторная (необязательно)	65
Варианты	65
Лабораторная работа №3. Работа с БД	69
Цель	69
Задание	69
Решение	69
Требования	81

Усложненная лабораторная (необязательно)	81
Варианты	81
Лабораторная работа №4. Работа с офисными пакетами	85
Цель	85
Задание	85
Решение	85
Требования	118
Усложненная лабораторная (необязательно)	119
Варианты	119
Лабораторная работа №5. Клиент-серверное приложение	123
Цель	123
Задание	123
Решение	123
Требования	144
Усложненная лабораторная (необязательно)	145
Варианты	146
Лабораторная работа №6. Многопоточность	149
Цель	149
Задание	149
Решение	149
Требования	158
Усложненная лабораторная (необязательно)	158
Варианты	158
Лабораторная работа №7. Регулярные выражения	162
Цель	162

Задание	162
Решение	162
Требования.....	183
Усложненная лабораторная (необязательно).....	183
Варианты.....	184
Лабораторная работа №8. Рефлексия.....	187
Цель	187
Задание	187
Решение	187
Требования.....	194
Усложненная лабораторная (необязательно).....	194
Варианты.....	194
Отчет по лабораторным работам.....	198
Список использованных источников	199
Приложение А	200

ЛАБОРАТОРНАЯ РАБОТА №0.

РАБОТА В СЕМЕСТРЕ

Цель

Ознакомиться с процессом выполнения работы в семестре.

Работа в семестре

В учебном плане предусмотрено 8 лабораторных работ. Работы выстроены так, что в течении семестра идет постепенное развитие одного проекта. В рамках первой лабораторной работы создается проект, в который в последующих лабораторных работах вносятся изменения, дополнения.

Каждая лабораторная работа разбита на 2 части: базовую и усложненную. **Базовую** часть нужно делать **обязательно**. Усложнённая часть не обязательна. Пример реализации для базовой части приводится в практикуме. Информацию для решения усложненной части следует искать в иных источниках. Каждая часть лабораторной работы оформляется в отдельную ветку в репозитории. За каждую лабораторную выставляются баллы, которые будут учитываться при выставлении оценки по дисциплине на экзамене. Принцип простой: кто больше работал в семестре (делал не только базовые лабораторные, но и усложненные) может претендовать на более высокую оценку.

Порядок действий будет следующим:

1. Создать репозиторию по шаблону «**Группа Фамилия И.О. Тема по варианту**».
2. Добавить к репозиторию проверяющего. Для этого в репозитории зайти в Settings, пункт Collaborators указать имя ulstuIS и добавить (результатом данной операции будет письмо для ulstuIS с предоставлением доступа в репозиторий).
3. В Visual Studio открыть Team Explorer (для этого не требуется создавать проект, просто открыть Visual Studio) и создать локальный репозиторий (читайте ниже как это делать).

4. Сделать коммит с файлами настроек git (файлы сами создадутся).
Связать локальный репозиторий с репозиторием с github (читайте ниже как это делать).
5. Перед первой лабораторной работой в репозитории создать ветку от ветки master (можно сделать на сайте, можно в Visual Studio через Team Explorer). В этой ветке потом будет храниться базовая часть первой лабораторной работы.
6. В Visual Studio открыть Team Explorer ваш репозиторий и выбрать первую ветку во вкладке веток (после выбора она станет жирной). Если во вкладке веток не отображается ветка для первой лабораторной, то перейти на вкладку «Синхронизация» и в пункте «Входящие фиксации» выбрать «Вытянуть» (рисунок 0.1).

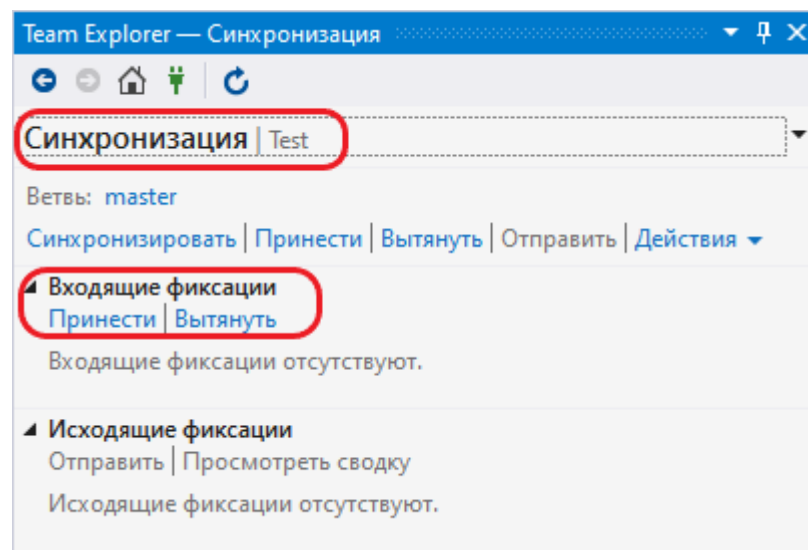


Рисунок 0.1 – Получение изменений с внешнего репозитория

7. Создать проект в папке репозитория. Имя проекта должно быть связано с вариантом задания первой лабораторной работы.
8. Во время работы и после сдачи лабораторной создаются коммиты первой лабораторной работы. После сдачи все коммиты должны быть отправлены в удаленный репозиторий.
9. На сайте github.com для ветки первой лабораторной создать pull request. Пул сравнивать с веткой master. **В названии пула должны**

обязательно фигурировать фамилия студента и номер лабораторной работы.

10. Убедиться, что во вкладке Files присутствуют все файлы лабораторной работы, которые вы делали (классы, формы и т.п.) (рисунок 0.2). Убедиться, что код оформлен верно (см. пункт **Правила оформления кода**).

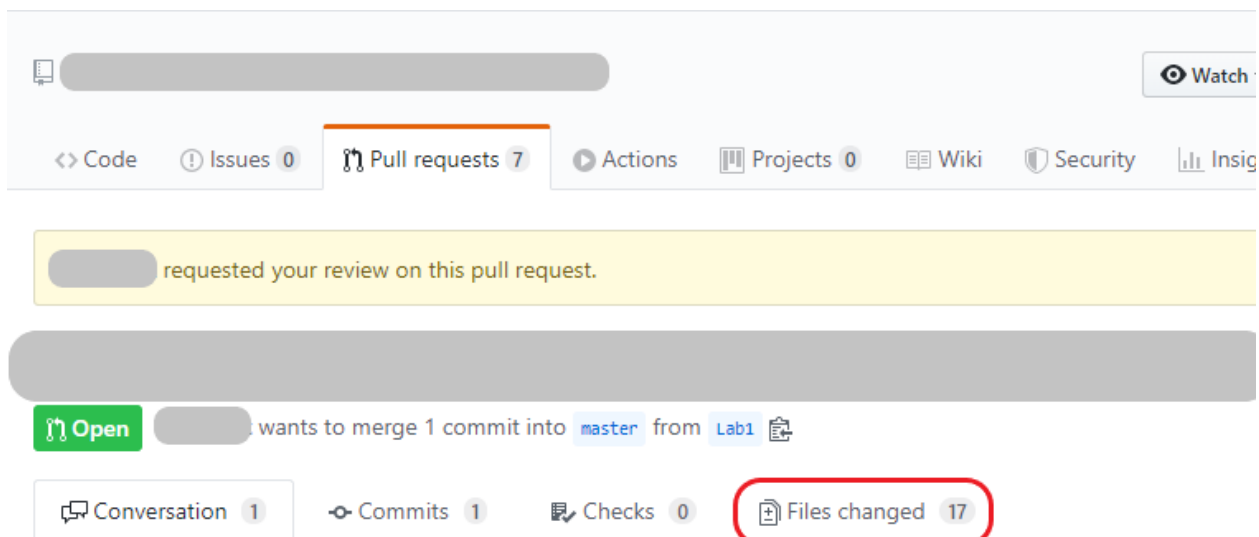


Рисунок 0.2 – Pull request

11. На вкладке Conversation справа в пункте Reviewers добавить пользователя ulstuIS (данная опция будет доступна, когда пользователь ulstuIS примет приглашение в репозиторий) (рисунок 0.3).

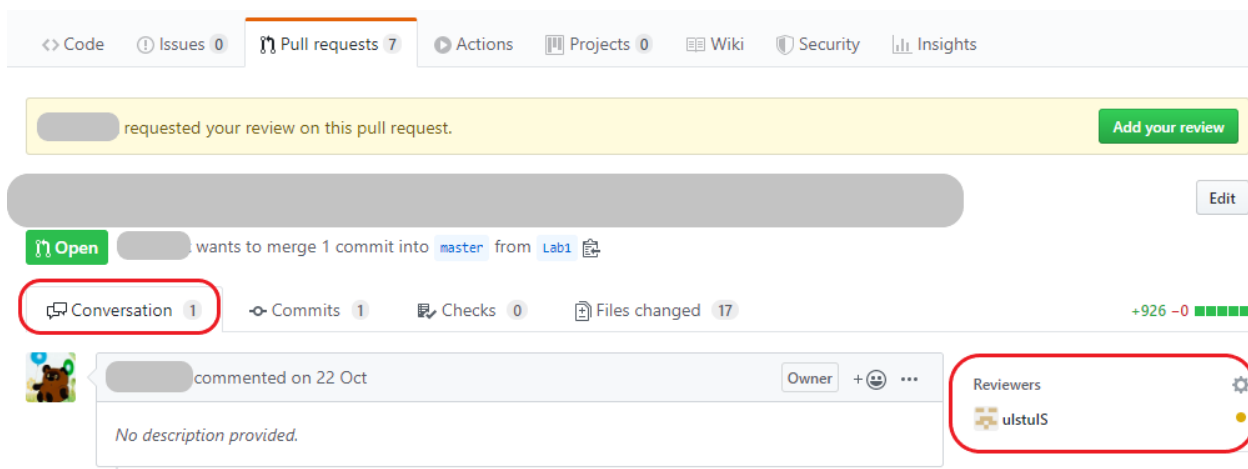


Рисунок 0.3 – Добавление проверяющего в пулл

12. После проверки на вкладке Conversation могут появиться замечания по лабораторной работе (возможно, вам на почту придут письма-оповещения). Для их устранения в Visual Studio следует вернуться на первую ветку репозитория, внести требуемые изменения, создать коммит и отправить его в репозиторий.
13. После устранения всех замечаний и повторной проверки будет выставлен комментарий «Замечаний нет». В таком случае лабораторная считается принятой. Pull request можно закрывать (лучше закрывать, а не делать мерж с веткой).
14. Для второй базовой лабораторной следует создавать ветку от первой базовой лабораторной, для третьей от второй и т.д.
15. Pull request для **второй** лабораторной создается **не в ветку master**, а в ветку **первой** лабораторной работы. Для последующих аналогично.
16. Ветки для усложненных лабораторных создаются от веток базовых лабораторных. Для второй, третьей и т.д. усложненной потребуется выполнить слияние этих веток с усложненными ветками предыдущих лабораторных. Для первой усложненной создается ветка от первой базовой. Для второй усложненной создается ветка от второй базовой и выполняется слияние с первой усложненной веткой (чтобы изменения первой усложненной оказались в ветке второй усложненной). Для третьей усложненной создается ветка от третьей базовой и идет слияние со второй усложненной и т.д.
17. **Важно!** Так как проверка пулов лабораторных происходит с некоторой задержкой по времени, то в репозитории успевают появиться несколько веток. В случае необходимости исправления замечаний по результатам проверки, могут возникать конфликты в pull request для последующих веток лабораторных работ.

Поэтому необходимо после внесения изменений в ветку, переключиться на ветку, созданную от этой ветки выполнить слияние с этой веткой, устранить конфликты, если есть, вылить изменение на внешний репозиторий. Повторить для всех остальных веток.

Правила сдачи лабораторных

- Посещение занятий **строго обязательно!** Допускаются пропуски по распоряжениям деканата (копия распоряжения), по болезни (справка из поликлиники) или по личным обстоятельствам (записка от родителей). За пропуск по неуважительной причине – минус баллы.
- К сдаче лабораторных допускаются студенты, успешно сдавшие дисциплину по технологии программирования за осенний семестр (не связано с посещением, **посещение обязательно для всех!**).
- Базовая лабораторная делается и сдается на очном занятии (по расписанию подгруппы, к которой приписан студент).
- При сдаче базовой части лабораторной могут задаваться дополнительные вопросы или дополнительные задания.
- По результатам сдачи выставляются баллы:
 - в случае сдачи лабораторной с доп. вопросами или задачами – 4 балла;
 - в случае сдачи не полной лабораторной или без ответа на доп. вопросы или без выполнения доп. задания – от 1 до 3 баллов;
 - в случае досдачи лабораторной работы на следующем занятии – ДО 2 баллов;

- Усложненная лабораторная выполняется самостоятельно вне очного занятия. Всю необходимую для выполнения задания информацию следует искать в различных источниках. Преподаватель будет **только** принимать результат. Если у вас что-то не работает, вы сами должны найти причину и разобраться что именно не работает.
- При сдаче усложненной части лабораторной могут задаваться дополнительные вопросы или дополнительные задания.
- По результатам сдачи выставляются баллы:
 - в случае сдачи лабораторной и ответе на доп. вопросы или задания – 4 балла;
 - в случае некорректного решения лабораторной, не ответа на каждый доп. вопрос, не выполнения на занятии доп. задания снимается по 1 баллу до тех пор, пока не будет сдана лабораторная.
- На занятиях (включая дополнительные) принимается не более 4-х лабораторных, из них не более 2-х усложненных лабораторных. Приоритет (особенно на доп. занятиях) отдается базовым лабораторным (они влияют на допуск к экзамену). Т.е., если у вас не сдано 3 базовых, то вам будет разрешено сдать 3 базовых и только 1 усложненную.
- На занятии (включая дополнительные) проверяется не более 4-х pull request (из них не более 2 по усложненным).
- Перед экзаменом все усложненные лабораторные, у которых не приняты pull request будут обнулены.
- **Важно!** При смене репозитория, все принятые пулы обнуляются, нужно будет выкладывать и сдавать заново.

Правила оформления кода

- имена элементов форм, имена классов, названия методов, переменных и т.п. давать логические (исходя из их предназначения);
- заголовки форм, подписи у label, кнопок должны быть оформлены на одном языке и в едином стиле;
- код должен быть отформатирован;
- в классах и реализации логики форм должны отсутствовать пустые методы;
- в файлах должны присутствовать только один из базовых элементов (класс, структура, перечисление и т.п.)
- в методах и классах недопустимо больше количество (более 3-х) пустых строк (отделяйте фрагменты комментариями или используете region для группировки кода);
- в выложенном на гите проекте должны отсутствовать файлы сборки (папка bin с исполняемым файлом и т.п., см. применение gitignore);

ЛАБОРАТОРНАЯ РАБОТА №1. БИБЛИОТЕКИ

Цель

Научиться создавать библиотеки классов. Ознакомиться с концепцией DAL.

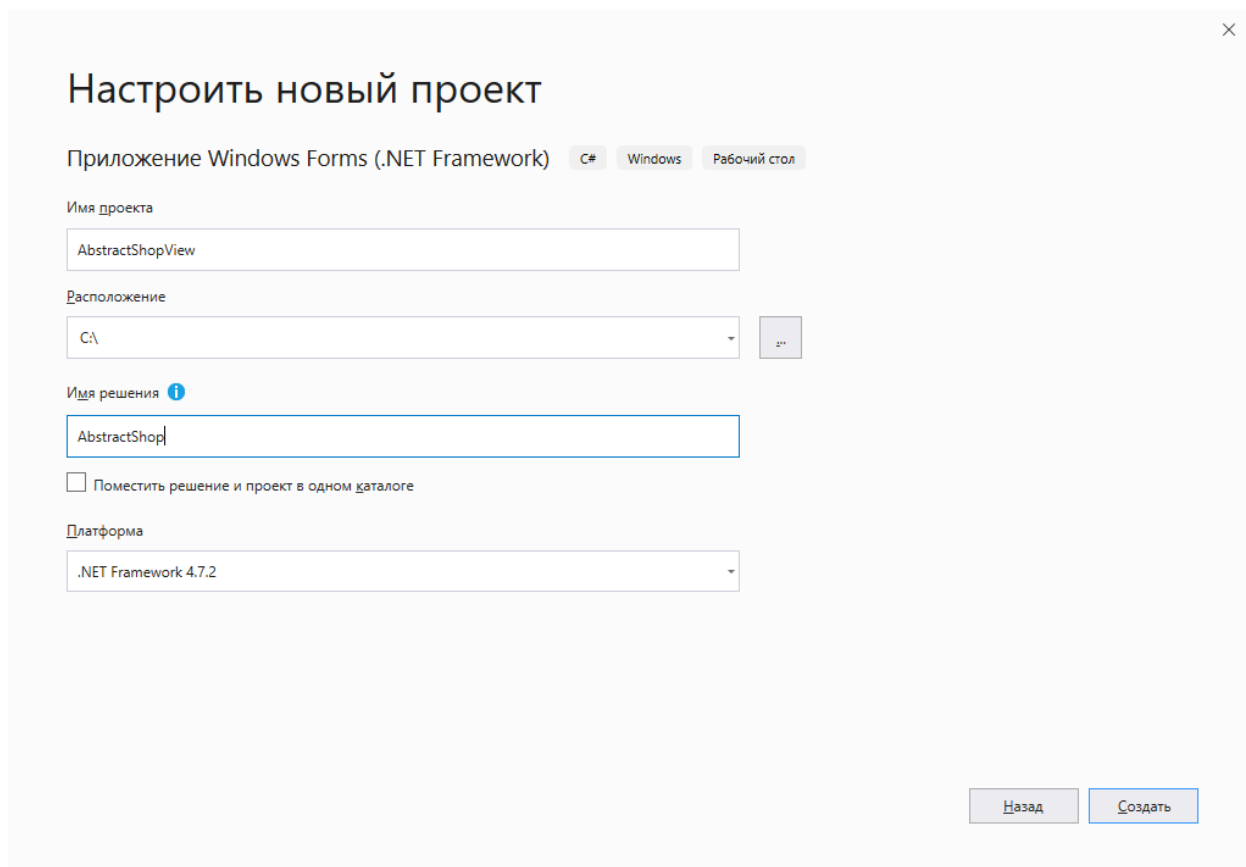
Задание

1. Создать новый проект с репозиторием. Создать ветку от главной ветки (для 1 лабораторной).
2. Создать приложение согласно заданию. Приложение реализовывать на основе архитектуры DAL. В качестве хранилища данных использовать операционную память. Данные вводить вручную при работе с программой, при ее завершении все данные будут удаляться.
3. Вылить полученный результат в созданную ветку. Убедиться, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Решение

Рассмотрим абстрактный пример магазина, в котором под заказы изготавливаются изделия, используя различные компоненты.

Создаем проект. Выберем обычное десктопное приложение. Назовем его «**AbstractShopView**». Решению дадим иное имя, потому что оно будет включать в себя несколько приложений. (рисунок 1.1). Создадим репозиторий на github.com и свяжем проект с репозиторием.



Настроить новый проект

Приложение Windows Forms (.NET Framework) C# Windows Рабочий стол

Имя проекта

AbstractShopView

Расположение

C:\

Имя решения ⓘ

AbstractShop

☐ Поместить решение и проект в одном каталоге

Платформа

.NET Framework 4.7.2

Назад Создать

Рисунок 1.1 – Создание проекта

При разработке будем использовать архитектуру DAL. Создадим проект для классов BindingModel, ViewModel, бизнес-логики и интерфейсов. Создадим под эти цели библиотеку. Чтобы библиотека была доступна как для классических приложений (.Net Framework), так и для кроссплатформенных (.Net Core), выберем тип библиотеки .Net Standard (рисунок 1.2). Назовем проект «**AbstractShopBusinessLogic**», так как в ней будет реализовываться логика проекта.

В проекте сделаем 5 папок. В первой папке поместим интерфейсы с описанием методов логики хранения данных (Interfaces). Для уменьшения связанности классов (принцип SOLID) введем еще две группы классов (т.е. 2 папки в проекте). BindingModels – для классов, в которых будут передаваться данные от интерфейса пользователя и ViewModels – для классов, в которых будет передаваться информация пользователю. Для хранения перечислений (у нас одно будет) – папку Enums. Для классов бизнес-логики сделаем папку BusinessLogics.

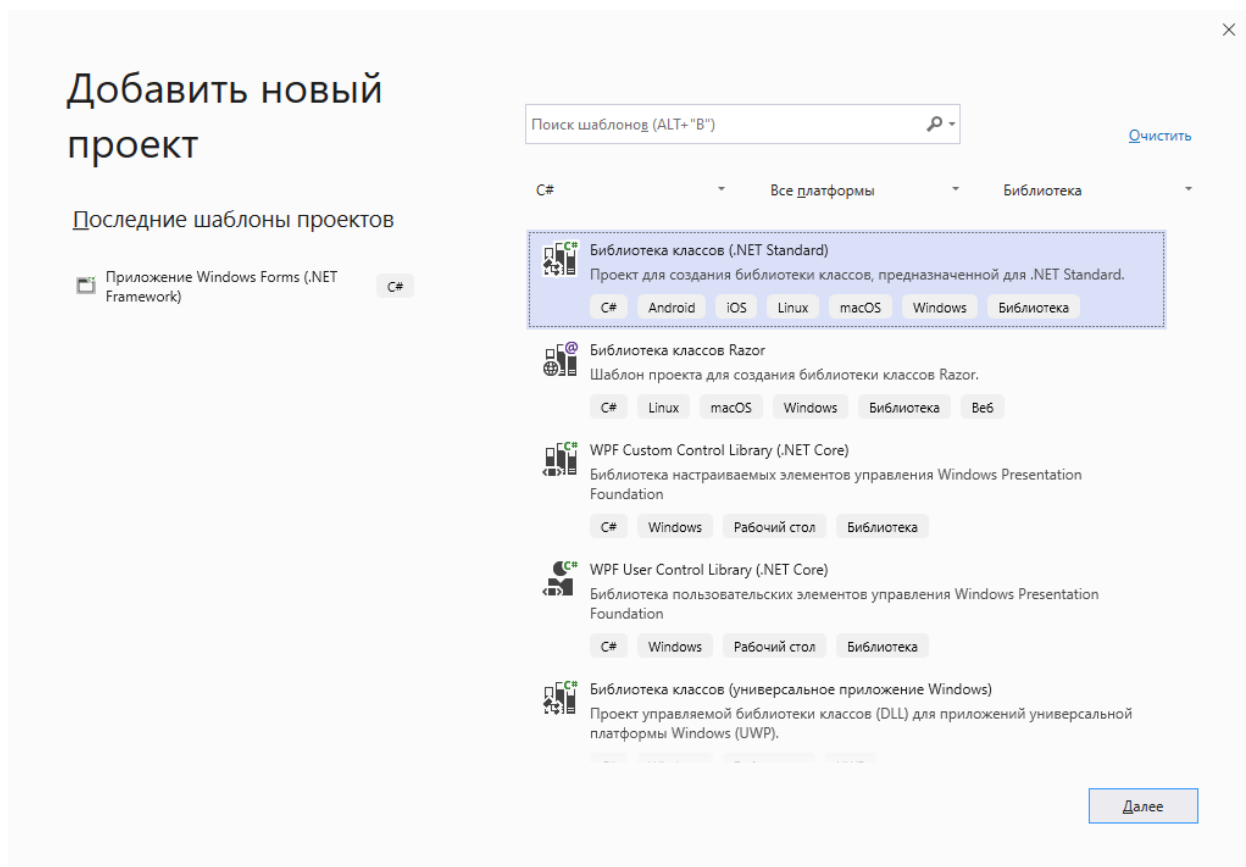


Рисунок 1.2 – Выбор типа создаваемого приложения

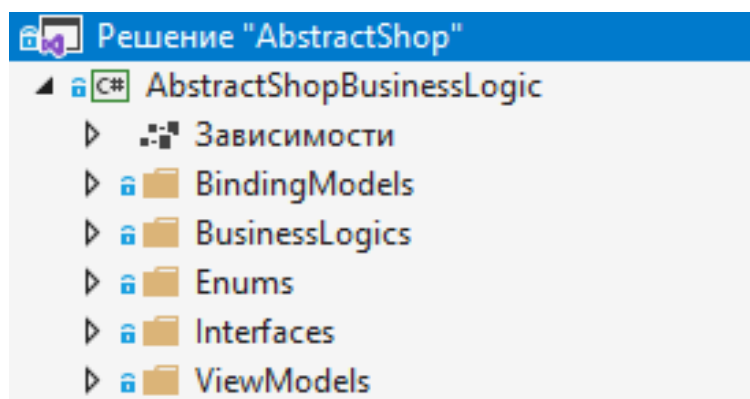


Рисунок 1.3 – Структура проекта AbstractShopBusinessLogic

Определимся какие нам нужны сущности:

- Компонент – для хранения информации по компонентам, требуемым для изготовления изделий.

- Изделие – для хранения информации по изделиям.

- Заказ – для хранения информации, о заказах на изделия.

И перечисление:

- Статус заказа (Принят, Выполняется, Готов, Оплачен).

В первую очередь создадим перечисление. Назовем его OrderStatus (листинг 1.1).

```
namespace AbstractShopBusinessLogic.Enums
{
    /// <summary>
    /// Статус заказа
    /// </summary>
    public enum OrderStatus
    {
        Принят = 0,

        Выполняется = 1,

        Готов = 2,

        Оплачен = 3
    }
}
```

Листинг 1.1 – Перечисление OrderStatus

Сделаем по сущностям классы с данными от пользователя (BindingModels). Классы будем называть по принципу «Имя сущности»BindingModel. В классах пропишем свойства, характерные для сущностей. У каждого класса укажем свойство Id для связи сущностей (компоненты с изделиями и изделия с заказами) (листинги 1.2-1.4).

```
namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Компонент, требуемый для изготовления изделия
    /// </summary>
    public class ComponentBindingModel
    {
        public int? Id { get; set; }

        public string ComponentName { get; set; }
    }
}
```

Листинг 1.2 – Класс ComponentBindingModel

```
using AbstractShopBusinessLogic.Enums;
using System;

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Заказ
    /// </summary>
    public class OrderBindingModel
    {
        public int? Id { get; set; }

        public int ProductId { get; set; }

        public int Count { get; set; }
    }
}
```



```

        public decimal Sum { get; set; }

        public OrderStatus Status { get; set; }

        public DateTime DateCreate { get; set; }

        public DateTime? DateImplement { get; set; }
    }
}

```

Листинг 1.3 – Класс OrderBindingModel

```

using System.Collections.Generic;

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Изделие, изготавливаемое в магазине
    /// </summary>
    public class ProductBindingModel
    {
        public int? Id { get; set; }

        public string ProductName { get; set; }

        public decimal Price { get; set; }

        public Dictionary<int, (string, int)> ProductComponents { get; set; }
    }
}

```

Листинг 1.4 – Класс ProductBindingModel

Также потребуется 2 класса для передачи данных от пользователя при создании заказа и при смене статусов заказов (листинги 1.5-1.6).

```

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Данные для смены статуса заказа
    /// </summary>
    public class ChangeStatusBindingModel
    {
        public int OrderId { get; set; }
    }
}

```

Листинг 1.5 – Класс ChangeStatusBindingModel

```

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Данные от клиента, для создания заказа
    /// </summary>
    public class CreateOrderBindingModel
    {
        public int ProductId { get; set; }

        public int Count { get; set; }

        public decimal Sum { get; set; }
    }
}

```

Листинг 1.6 – Класс CreateOrderBindingModel

Следующий шаг – сделать классы с данными, передаваемые для отображения пользователю (ViewModels). Классы будем называть по принципу «Имя сущности»ViewModel. Во многом они будут схожи с классами BindingModels (листинги 1.7-1.9).

```

using System.ComponentModel;

namespace AbstractShopBusinessLogic.ViewModels
{
    /// <summary>
    /// Компонент, требуемый для изготовления изделия
    /// </summary>
    public class ComponentViewModel
    {
        public int Id { get; set; }

        [DisplayName("Название компонента")]
        public string ComponentName { get; set; }
    }
}

```

Листинг 1.7 – Класс ComponentViewModel

```

using AbstractShopBusinessLogic.Enums;
using System;
using System.ComponentModel;

namespace AbstractShopBusinessLogic.ViewModels
{
    /// <summary>
    /// Заказ
    /// </summary>
    public class OrderViewModel
    {
        public int Id { get; set; }

        public int ProductId { get; set; }

        [DisplayName("Изделие")]
        public string ProductName { get; set; }

        [DisplayName("Количество")]
        public int Count { get; set; }
    }
}

```

```

        [DisplayName("Сумма")]
        public decimal Sum { get; set; }

        [DisplayName("Статус")]
        public OrderStatus Status { get; set; }

        [DisplayName("Дата создания")]
        public DateTime DateCreate { get; set; }

        [DisplayName("Дата выполнения")]
        public DateTime? DateImplement { get; set; }
    }
}

```

Листинг 1.8 – Класс OrderViewModel

```

using System.Collections.Generic;
using System.ComponentModel;

namespace AbstractShopBusinessLogic.ViewModels
{
    /// <summary>
    /// Изделие, изготавливаемое в магазине
    /// </summary>
    public class ProductViewModel
    {
        public int Id { get; set; }

        [DisplayName("Название изделия")]
        public string ProductName { get; set; }

        [DisplayName("Цена")]
        public decimal Price { get; set; }

        public Dictionary<int, (string, int)> ProductComponents { get; set; }
    }
}

```

Листинг 1.9 – Класс ProductViewModel

Значение «DateTime? DateImplement» говорит о том, что при создании записи о заказе не требуется указывать дату выполнения заказа, она может хранить в себе значения null. По логике это означает, что в момент оформления заказа мы не знаем дату, когда он будет выполнен.

У некоторых свойств классов указан атрибут DisplayName. Он будет использоваться при выводе данных на форме в табличной форме.

Следующий шаг в проекте – создание интерфейсов. Для сущностей «Компонент», «Изделие» и «Заказ» будут интерфейсы с методами для получения полного списка, фильтрованного списка и отдельного элемента, добавления, изменения и удаление записей (листинг 1.10-1.12).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

```

```

namespace AbstractShopBusinessLogic.Interfaces
{
    public interface IComponentStorage
    {
        List<ComponentViewModel> GetFullList();

        List<ComponentViewModel> GetFilteredList(ComponentBindingModel model);

        ComponentViewModel GetElement(ComponentBindingModel model);

        void Insert(ComponentBindingModel model);

        void Update(ComponentBindingModel model);

        void Delete(ComponentBindingModel model);
    }
}

```

Листинг 1.10 – Интерфейс IComponentStorage

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.Interfaces
{
    public interface IOrderStorage
    {
        List<OrderViewModel> GetFullList();

        List<OrderViewModel> GetFilteredList(OrderBindingModel model);

        OrderViewModel GetElement(OrderBindingModel model);

        void Insert(OrderBindingModel model);

        void Update(OrderBindingModel model);

        void Delete(OrderBindingModel model);
    }
}

```

Листинг 1.11 – Интерфейс IOrderStorage

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.Interfaces
{
    public interface IProductStorage
    {
        List<ProductViewModel> GetFullList();

        List<ProductViewModel> GetFilteredList(ProductBindingModel model);

        ProductViewModel GetElement(ProductBindingModel model);

        void Insert(ProductBindingModel model);

        void Update(ProductBindingModel model);

        void Delete(ProductBindingModel model);
    }
}

```

```
}  
}
```

Листинг 1.12 – Интерфейс IProductStorage

Последний шаг, сделать классы с логикой работы. Классы с логикой для изделия и компонента будут простыми реализациями принципа CRUD (листинг 1.13).

```
using AbstractShopBusinessLogic.BindingModels;  
using AbstractShopBusinessLogic.Interfaces;  
using AbstractShopBusinessLogic.ViewModels;  
using System;  
using System.Collections.Generic;  
  
namespace AbstractShopBusinessLogic.BusinessLogics  
{  
    public class ComponentLogic  
    {  
        private readonly IComponentStorage _componentStorage;  
  
        public ComponentLogic(IComponentStorage componentStorage)  
        {  
            _componentStorage = componentStorage;  
        }  
  
        public List<ComponentViewModel> Read(ComponentBindingModel model)  
        {  
            if(model == null)  
            {  
                return _componentStorage.GetFullList();  
            }  
  
            if (model.Id.HasValue)  
            {  
                return new List<ComponentViewModel> { _componentStorage.GetElement(model)  
};  
            }  
  
            return _componentStorage.GetFilteredList(model);  
        }  
  
        public void CreateOrUpdate(ComponentBindingModel model)  
        {  
            var element = _componentStorage.GetElement(new ComponentBindingModel {  
ComponentName = model.ComponentName });  
            if (element != null && element.Id != model.Id)  
            {  
                throw new Exception("Уже есть компонент с таким названием");  
            }  
  
            if (model.Id.HasValue)  
            {  
                _componentStorage.Update(model);  
            }  
            else  
            {  
                _componentStorage.Insert(model);  
            }  
        }  
  
        public void Delete(ComponentBindingModel model)
```

```

        {
            var element = _componentStorage.GetElement(new ComponentBindingModel { Id =
model.Id });
            if (element == null)
            {
                throw new Exception("Элемент не найден");
            }
            _componentStorage.Delete(model);
        }
    }
}

```

Листинг 1.13 – Класс ComponentLogic

Класс с логикой для заказов будет отвечать за получение списка заказов, создания заказа и смены его статусов (листинг 1.14).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Enums;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class OrderLogic
    {
        private readonly IOrderStorage _orderStorage;

        public OrderLogic(IOrderStorage orderStorage)
        {
            _orderStorage = orderStorage;
        }

        public List<OrderViewModel> Read(OrderBindingModel model)
        {
            if (model == null)
            {
                return _orderStorage.GetFullList();
            }

            if (model.Id.HasValue)
            {
                return new List<OrderViewModel> { _orderStorage.GetElement(model) };
            }

            return _orderStorage.GetFilteredList(model);
        }

        public void CreateOrder(CreateOrderBindingModel model)
        {
            _orderStorage.Insert(new OrderBindingModel
            {
                ProductId = model.ProductId,
                Count = model.Count,
                Sum = model.Sum,
                DateCreate = DateTime.Now,
                Status = OrderStatus.Принят
            });
        }
    }
}

```

```

        public void TakeOrderInWork(ChangeStatusBindingModel model)
        {
            var order = _orderStorage.GetElement(new OrderBindingModel { Id =
model.OrderId });
            if(order == null)
            {
                throw new Exception("Не найден заказ");
            }
            if (order.Status != OrderStatus.Принят)
            {
                throw new Exception("Заказ не в статусе \"Принят\"");
            }
            _orderStorage.Update(new OrderBindingModel
            {
                Id = order.Id,
                ProductId = order.ProductId,
                Count = order.Count,
                Sum = order.Sum,
                DateCreate = order.DateCreate,
                DateImplement = DateTime.Now,
                Status = OrderStatus.Выполняется
            });
        }

        public void FinishOrder(ChangeStatusBindingModel model)
        {
            var order = _orderStorage.GetElement(new OrderBindingModel { Id =
model.OrderId });
            if (order == null)
            {
                throw new Exception("Не найден заказ");
            }
            if (order.Status != OrderStatus.Выполняется)
            {
                throw new Exception("Заказ не в статусе \"Выполняется\"");
            }
            _orderStorage.Update(new OrderBindingModel
            {
                Id = order.Id,
                ProductId = order.ProductId,
                Count = order.Count,
                Sum = order.Sum,
                DateCreate = order.DateCreate,
                DateImplement = order.DateImplement,
                Status = OrderStatus.Готов
            });
        }

        public void PayOrder(ChangeStatusBindingModel model)
        {
            // продумать логику
        }
    }
}

```

Листинг 1.14 – Класс OrderLogic

Описав логику действий и форматы входных/выходных данных можем реализовать интерфейс пользователя.

Переходим в проект «**AbstractShopView**». Первым делом введем IoC-контейнер для установления зависимостей между интерфейсами их реализациями (реализации добавим позже) и добавим ссылку на созданный проект с интерфейсами. Открыть вкладку «Обозреватель решения», найти проект – десктопное приложение, кликнуть правой кнопкой мыши по «Ссылки» и выбрать пункт «Добавить ссылку...». Далее, перейти во вкладку «Проекты» и поставить галочку напротив проекта «**AbstractShopBusinessLogic**». Таким образом, классы, перечисление и интерфейсы, созданные в проекте **AbstractShopBusinessLogic**, будут доступны в проекте (рисунок 1.4).

IoC-контейнер добавим через NuGet пакеты. Воспользуемся IoC-контейнером Unity (рисунок 1.5).

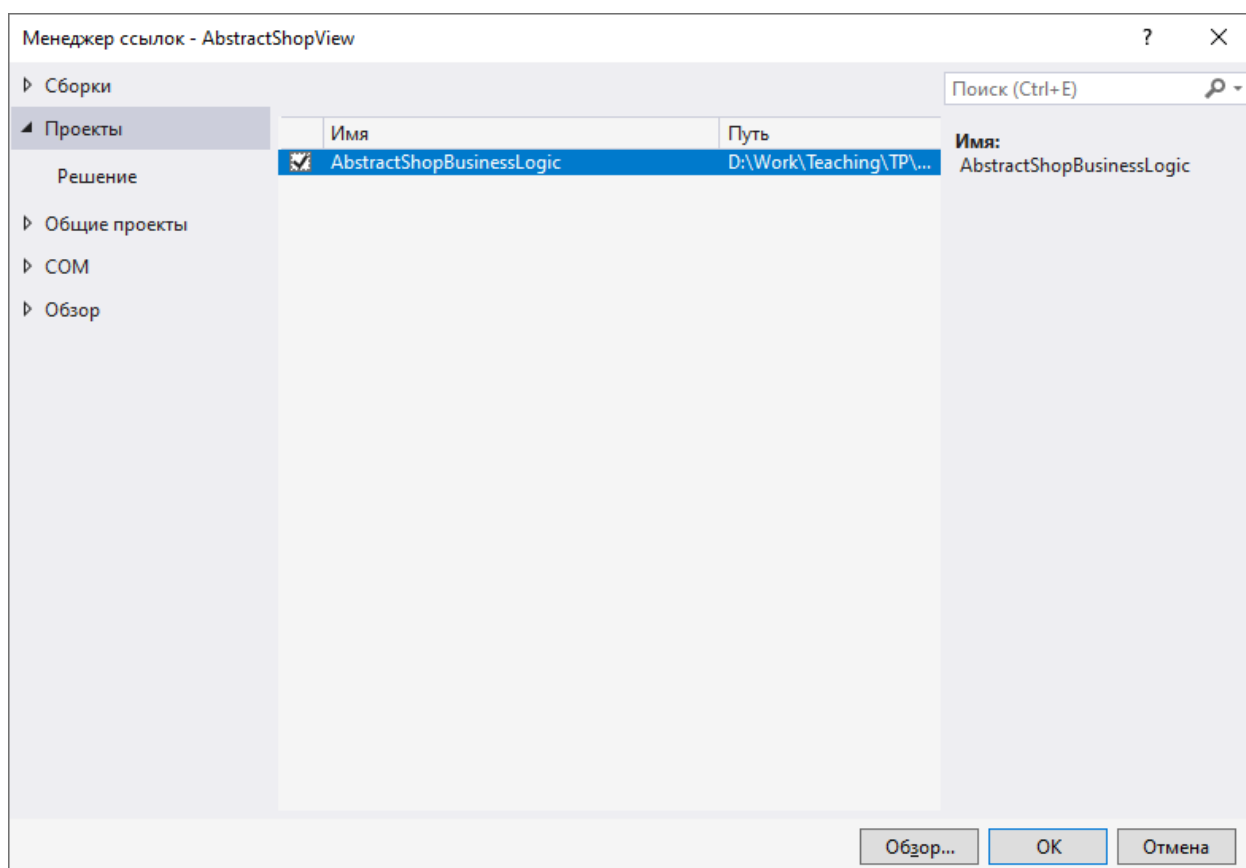


Рисунок 1.4 – Связывание проектов

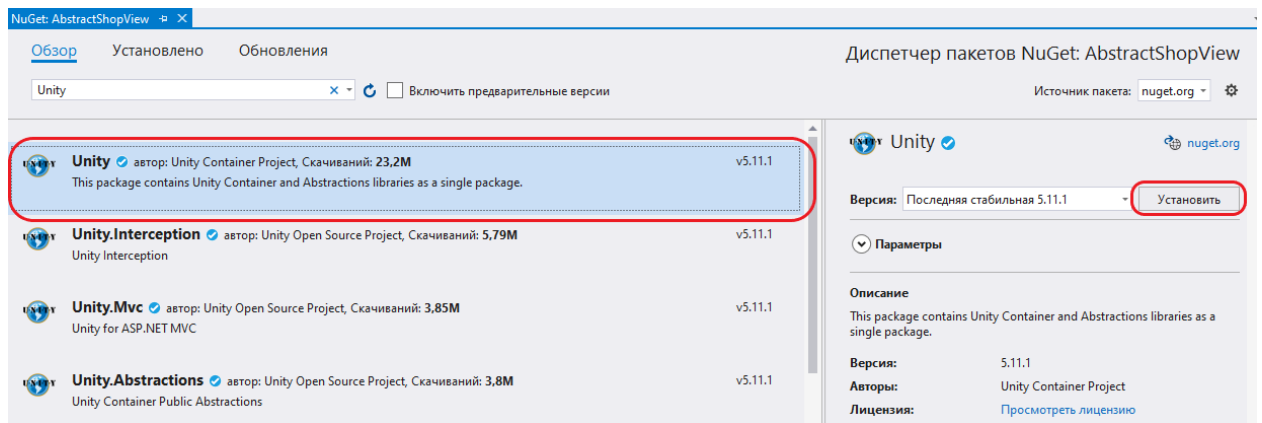


Рисунок 1.5 – Добавление IoC-контейнера Unity

Будем делать формы. Начнем с компонента. Сделаем форму для создания/изменения компонента. Там будет все просто, одно поле для ввода названия компонента (рисунок 1.6).

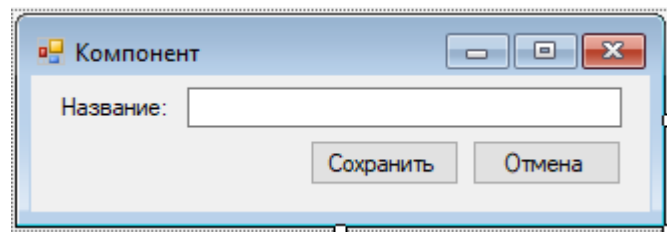


Рисунок 1.6 – Форма работы с компонентом

Логика формы будет следующей (листинг 1.15).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using System;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormComponent : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        public int Id { set { id = value; } }

        private readonly ComponentLogic logic;

        private int? id;

        public FormComponent(ComponentLogic logic)
        {
            InitializeComponent();
            this.logic = logic;
        }

        private void FormComponent_Load(object sender, EventArgs e)
        {
            if (id.HasValue)
```

```

        {
            try
            {
                var view = logic.Read(new ComponentBindingModel { Id = id })?[0];
                if (view != null)
                {
                    textBoxName.Text = view.ComponentName;
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            }
        }

        private void ButtonSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(textBoxName.Text))
            {
                MessageBox.Show("Заполните название", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
                return;
            }
            try
            {
                logic.CreateOrUpdate(new ComponentBindingModel
                {
                    Id = id,
                    ComponentName = textBoxName.Text
                });
                MessageBox.Show("Сохранение прошло успешно", "Сообщение",
                MessageBoxButtons.OK, MessageBoxIcon.Information);
                DialogResult = DialogResult.OK;
                Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            }
        }

        private void ButtonCancel_Click(object sender, EventArgs e)
        {
            DialogResult = DialogResult.Cancel;
            Close();
        }
    }
}

```

Листинг 1.15 – Логика формы FormComponent

Обязательно нужно сделать поле для Unity-контейнера, для корректной работы. В конструкторе будем передавать объект класса ComponentLogic (в его конструкторе Unity-контейнер будет подставлять реализацию интерфейса IComponentStorage). В методе загрузки будем проверять, если поле id заполнено, то пытаться получить запись и вывести ее на экран. При

сохранении, передаем данные для добавления или редактирования записи. При отмене просто закрываем форму.

Далее сделаем форму для вывода всех имеющихся компонент (рисунок 1.7).

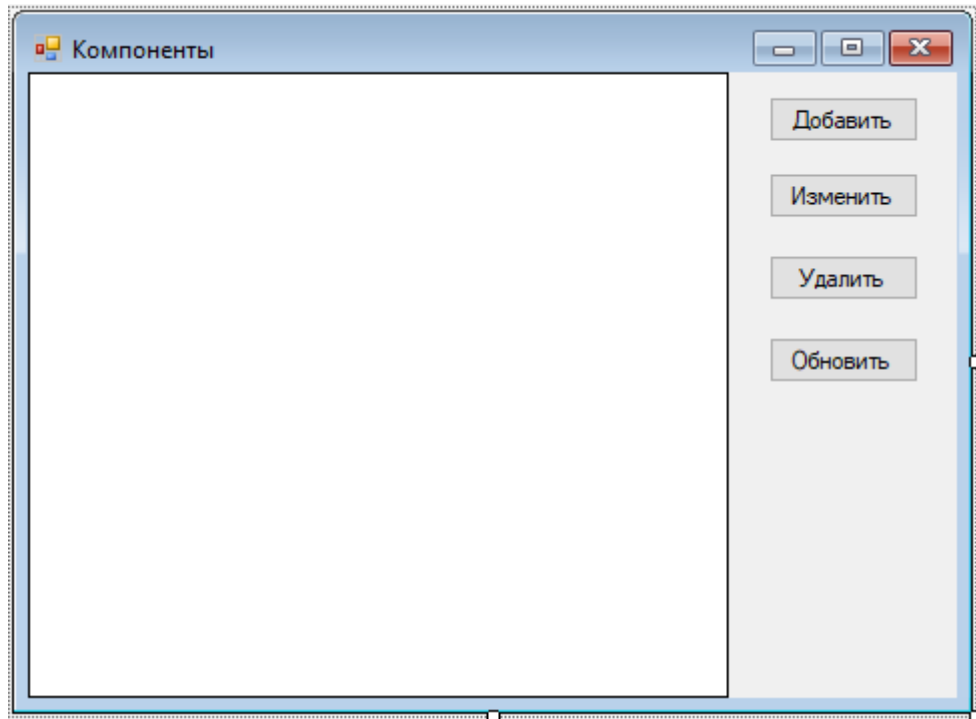


Рисунок 1.7 – Форма работы с набором компонент

Логика формы будет следующей (листинг 1.16).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using System;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormComponents : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        private readonly ComponentLogic logic;

        public FormComponents(ComponentLogic logic)
        {
            InitializeComponent();
            this.logic = logic;
        }

        private void FormComponents_Load(object sender, EventArgs e)
        {
            LoadData();
        }
    }
}
```

```

private void LoadData()
{
    try
    {
        var list = logic.Read(null);
        if (list != null)
        {
            dataGridView.DataSource = list;
            dataGridView.Columns[0].Visible = false;
            dataGridView.Columns[1].AutoSizeMode =
DataGridviewAutoSizeColumnMode.Fill;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

private void ButtonAdd_Click(object sender, EventArgs e)
{
    var form = Container.Resolve<FormComponent>();
    if (form.ShowDialog() == DialogResult.OK)
    {
        LoadData();
    }
}

private void ButtonUpd_Click(object sender, EventArgs e)
{
    if (dataGridView.SelectedRows.Count == 1)
    {
        var form = Container.Resolve<FormComponent>();
        form.Id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
        if (form.ShowDialog() == DialogResult.OK)
        {
            LoadData();
        }
    }
}

private void ButtonDel_Click(object sender, EventArgs e)
{
    if (dataGridView.SelectedRows.Count == 1)
    {
        if (MessageBox.Show("Удалить запись", "Вопрос", MessageBoxButtons.YesNo,
MessageBoxIcon.Question) == DialogResult.Yes)
        {
            int id =
Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
            try
            {
                logic.Delete(new ComponentBindingModel { Id = id });
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
            LoadData();
        }
    }
}
}

```

```
private void ButtonRef_Click(object sender, EventArgs e)
{
    LoadData();
}
}
```

Листинг 1.16 – Логика формы FormComponents

Также потребуется поле для IUnityContainer. Сделаем отдельный метод для загрузки списка компонент. При получении списка будет приходить список объектов ComponentViewModel. Список будем передавать в элемент DataGridView. Так как, каждый объект содержит 2 поля: id и название компонента, то в DataGridView создается 2 колонки под эти поля. Id нужен для внутренних нужд и его не требуется выводить пользователю. Скроем первую колонку в DataGridView. А для второй колонки настроим отображение на всю доступную ширину элемента DataGridView. В логике добавления и изменения будем вызывать форму для работы с компонентом, используя поле Container от IUnityContainer (для этого он и объявляется в классе). В логике изменения и удаления в первую очередь проверяем, что есть выбранная строка и вытаскиваем значение из первой ячейки выбранной строки. В конце логики для всех кнопок вызывается метод обновления списка.

Для изделий потребуется три формы: для списка, для отдельного изделия (создание и редактирование) и для добавления компонент в изделие. Начнем с последней формы.

Нам потребуется выбирать компонент и указывать в каком количестве он будет использоваться в изделии. Для выбора компонента на форму поместим элемент comboBox, для указания количества – textBox (рисунок 1.8).

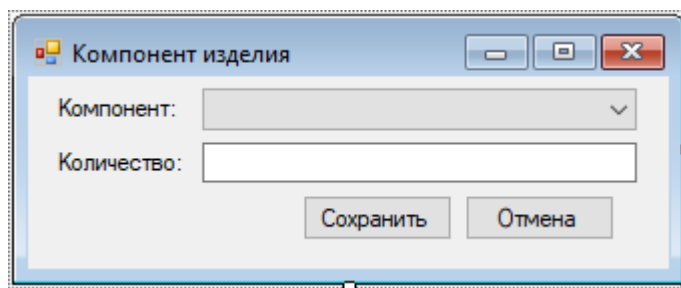


Рисунок 1.8 – Форма выбора компонента в изделие

Логика формы будет следующей (листинг 1.17).

```
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormProductComponent : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        public int Id { get { return Convert.ToInt32(comboBoxComponent.SelectedValue); }
set { comboBoxComponent.SelectedValue = value; } }

        public string ComponentName { get { return comboBoxComponent.Text; } }

        public int Count { get { return Convert.ToInt32(textBoxCount.Text); } set {
textBoxCount.Text = value.ToString(); } }

        public FormProductComponent(ComponentLogic logic)
        {
            InitializeComponent();

            List<ComponentViewModel> list = logic.Read(null);
            if (list != null)
            {
                comboBoxComponent.DisplayMember = "ComponentName";
                comboBoxComponent.ValueMember = "Id";
                comboBoxComponent.DataSource = list;
                comboBoxComponent.SelectedItem = null;
            }
        }

        private void ButtonSave_Click(object sender, EventArgs e)
        {
            if (string.IsNullOrEmpty(textBoxCount.Text))
            {
                MessageBox.Show("Заполните поле Количество", "Ошибка",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            if (comboBoxComponent.SelectedValue == null)
            {
                MessageBox.Show("Выберите компонент", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
                return;
            }
        }
    }
}
```

```

        }

        DialogResult = DialogResult.OK;
        Close();
    }

    private void ButtonCancel_Click(object sender, EventArgs e)
    {
        DialogResult = DialogResult.Cancel;
        Close();
    }
}

```

Листинг 1.17 – Логика формы FormProductComponent

В логике будут свойства для получения/возврата информации о компоненте (идентификатор и название) и количестве. В конструкторе формы получаем список компонент и заносим его в выпадающий список (так как в свойстве Id прописали работу сразу с comboBoxComponent, то он должен быть заполнен до обращения к этому свойству, а не в момент загрузки формы). При сохранении проверяем, что поля заполнены и закрываем форму.

Перейдем к форме Изделия. У изделия требуется вводить название, цену, а также список компонентов, которые в нем используются. Потому форма будет чуть сложнее, чем у компонента (рисунок 1.9). Для вывода компонент будет использовать DataGridView, у которого ручками добавим колонки (3 колонки для id компонента, которая будет скрытая, названия и количества).

Логика формы будет следующей (листинг 1.18).

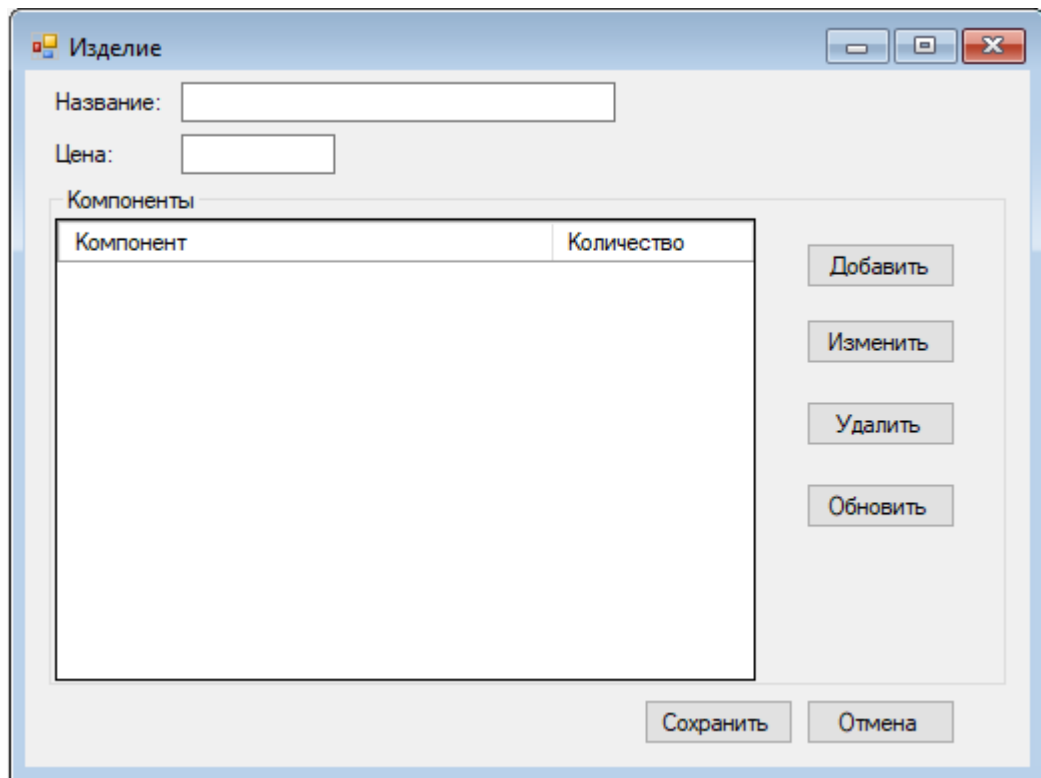


Рисунок 1.9 – Форма работы с изделием

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormProduct : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        public int Id { set { id = value; } }

        private readonly ProductLogic logic;

        private int? id;

        private Dictionary<int, (string, int)> productComponents;

        public FormProduct(ProductLogic service)
        {
            InitializeComponent();
            this.logic = service;
        }

        private void FormProduct_Load(object sender, EventArgs e)
        {
            if (id.HasValue)
            {
                try
                {

```



```

        ProductViewModel view = logic.Read(new ProductBindingModel { Id =
id.Value })?[0];
        if (view != null)
        {
            textBoxName.Text = view.ProductName;
            textBoxPrice.Text = view.Price.ToString();
            productComponents = view.ProductComponents;
            LoadData();
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}
else
{
    productComponents = new Dictionary<int, (string, int)>();
}
}

private void LoadData()
{
    try
    {
        if (productComponents != null)
        {
            dataGridView.Rows.Clear();
            foreach (var pc in productComponents)
            {
                dataGridView.Rows.Add(new object[] { pc.Key, pc.Value.Item1,
pc.Value.Item2 });
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

private void ButtonAdd_Click(object sender, EventArgs e)
{
    var form = Container.Resolve<FormProductComponent>();
    if (form.ShowDialog() == DialogResult.OK)
    {
        if (productComponents.ContainsKey(form.Id))
        {
            productComponents[form.Id] = (form.ComponentName, form.Count);
        }
        else
        {
            productComponents.Add(form.Id, (form.ComponentName, form.Count));
        }
        LoadData();
    }
}

private void ButtonUpd_Click(object sender, EventArgs e)
{
    if (dataGridView.SelectedRows.Count == 1)
    {

```

```

        var form = Container.Resolve<FormProductComponent>();
        int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
        form.Id = id;
        form.Count = productComponents[id].Item2;
        if (form.ShowDialog() == DialogResult.OK)
        {
            productComponents[form.Id] = (form.ComponentName, form.Count);
            LoadData();
        }
    }

    private void ButtonDel_Click(object sender, EventArgs e)
    {
        if (dataGridView.SelectedRows.Count == 1)
        {
            if (MessageBox.Show("Удалить запись", "Вопрос", MessageBoxButtons.YesNo,
                MessageBoxIcon.Question) == DialogResult.Yes)
            {
                try
                {
                    productComponents.Remove(Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value));
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                        MessageBoxIcon.Error);
                }
                LoadData();
            }
        }
    }

    private void ButtonRef_Click(object sender, EventArgs e)
    {
        LoadData();
    }

    private void ButtonSave_Click(object sender, EventArgs e)
    {
        if (string.IsNullOrEmpty(textBoxName.Text))
        {
            MessageBox.Show("Заполните название", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return;
        }
        if (string.IsNullOrEmpty(textBoxPrice.Text))
        {
            MessageBox.Show("Заполните цену", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return;
        }
        if (productComponents == null || productComponents.Count == 0)
        {
            MessageBox.Show("Заполните компоненты", "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            return;
        }
        try
        {
            logic.CreateOrUpdate(new ProductBindingModel
            {
                Id = id,

```

```

        ProductName = textBoxName.Text,
        Price = Convert.ToDecimal(textBoxPrice.Text),
        ProductComponents = productComponents
    });
    MessageBox.Show("Сохранение прошло успешно", "Сообщение",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
    DialogResult = DialogResult.OK;
    Close();
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
    MessageBoxIcon.Error);
}
}

private void ButtonCancel_Click(object sender, EventArgs e)
{
    DialogResult = DialogResult.Cancel;
    Close();
}
}
}

```

Листинг 1.18 – Логика формы FormProduct

Здесь будет комбинация из логики работы с элементом (изделие) и списком (компоненты изделия). В отдельном поле будем хранить список компонент изделия и выводить его в DataGridView.

Форма и логика для списка изделий будет идентична форме списка компонент.

Остается форма создания заказа и главная форма. Начнем с формы создания заказа. При создании заказа потребуется указать изделие (выпадающий список), количество и сумму (должна высчитываться из цены изделия и количества) (рисунок 1.10).

Рисунок 1.10 – Форма создания заказа

Логика формы будет следующей (листинг 1.19).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;

```

```

using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormCreateOrder : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        private readonly ProductLogic _logicP;

        private readonly OrderLogic _logicO;

        public FormCreateOrder(ProductLogic logicP, OrderLogic logicO)
        {
            InitializeComponent();
            _logicP = logicP;
            _logicO = logicO;
        }

        private void FormCreateOrder_Load(object sender, EventArgs e)
        {
            try
            {
                // продумать логику
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            }
        }

        private void CalcSum()
        {
            if (comboBoxProduct.SelectedValue != null &&
            !string.IsNullOrEmpty(textBoxCount.Text))
            {
                try
                {
                    int id = Convert.ToInt32(comboBoxProduct.SelectedValue);
                    ProductViewModel product = _logicP.Read(new ProductBindingModel { Id
= id })?[0];

                    int count = Convert.ToInt32(textBoxCount.Text);
                    textBoxSum.Text = (count * product?.Price ?? 0).ToString();
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
                }
            }
        }

        private void TextBoxCount_TextChanged(object sender, EventArgs e)
        {
            CalcSum();
        }

        private void ComboBoxProduct_SelectedIndexChanged(object sender, EventArgs e)
        {

```

```

        CalcSum();
    }

    private void ButtonSave_Click(object sender, EventArgs e)
    {
        if (string.IsNullOrEmpty(textBoxCount.Text))
        {
            MessageBox.Show("Заполните поле Количество", "Ошибка",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }
        if (comboBoxProduct.SelectedValue == null)
        {
            MessageBox.Show("Выберите изделие", "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
            return;
        }
        try
        {
            _logicO.CreateOrder(new CreateOrderBindingModel
            {
                ProductId = Convert.ToInt32(comboBoxProduct.SelectedValue),
                Count = Convert.ToInt32(textBoxCount.Text),
                Sum = Convert.ToDecimal(textBoxSum.Text)
            });
            MessageBox.Show("Сохранение прошло успешно", "Сообщение",
            MessageBoxButtons.OK, MessageBoxIcon.Information);
            DialogResult = DialogResult.OK;
            Close();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        }
    }

    private void ButtonCancel_Click(object sender, EventArgs e)
    {
        DialogResult = DialogResult.Cancel;
        Close();
    }
}

```

Листинг 1.19 – Логика формы FormCreateOrder

При загрузке формы подгружаем список изделий. Делаем отдельный метод расчета суммы и вызываем его при изменении в поле «Количество» или при выборе элемента из выпадающего списка.

И последняя форма – главная форма приложения. В ней будет отображаться список всех заказов, функционал для создания заказов, смены статусов, а также для доступа к спискам компонент и изделий (сделаем через пункты меню) (рисунок 1.11).

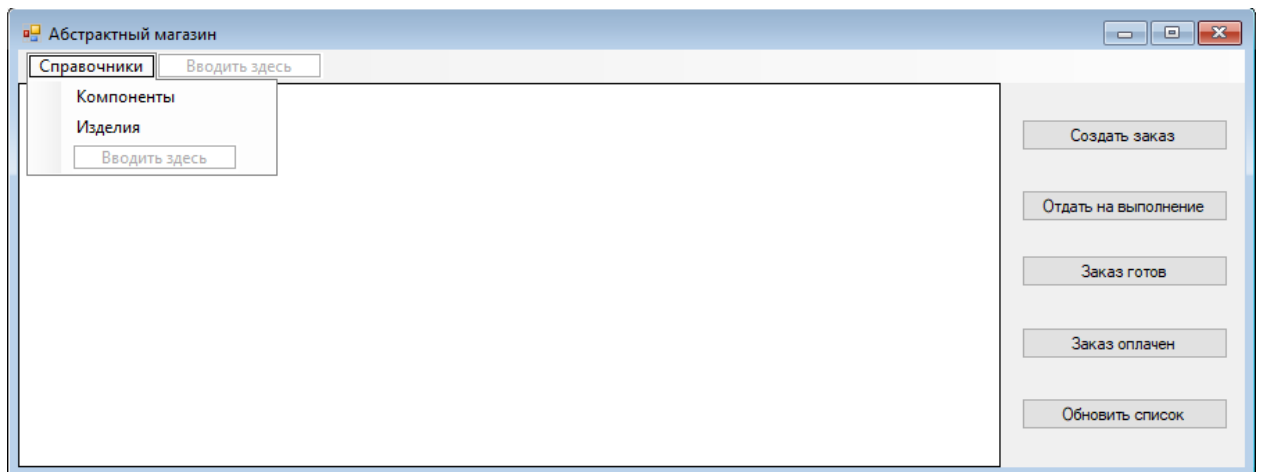


Рисунок 1.11 – Главная форма

Логика формы будет следующей (листинг 1.20).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using System;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormMain : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        private readonly OrderLogic _orderLogic;

        public FormMain(OrderLogic orderLogic)
        {
            InitializeComponent();
            this._orderLogic = orderLogic;
        }

        private void FormMain_Load(object sender, EventArgs e)
        {
            LoadData();
        }

        private void LoadData()
        {
            try
            {
                // продумать логику
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
            }
        }

        private void КомпонентыToolStripMenuItem_Click(object sender, EventArgs e)
        {
            var form = Container.Resolve<FormComponents>();
            form.ShowDialog();
        }
    }
}
```

```

    }

    private void ИзделияToolStripMenuItem_Click(object sender, EventArgs e)
    {
        var form = Container.Resolve<FormProducts>();
        form.ShowDialog();
    }

    private void ButtonCreateOrder_Click(object sender, EventArgs e)
    {
        var form = Container.Resolve<FormCreateOrder>();
        form.ShowDialog();
        LoadData();
    }

    private void ButtonTakeOrderInWork_Click(object sender, EventArgs e)
    {
        if (dataGridView.SelectedRows.Count == 1)
        {
            int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
            try
            {
                _orderLogic.TakeOrderInWork(new ChangeStatusBindingModel { OrderId =
id });
                LoadData();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
    }

    private void ButtonOrderReady_Click(object sender, EventArgs e)
    {
        if (dataGridView.SelectedRows.Count == 1)
        {
            int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
            try
            {
                _orderLogic.FinishOrder(new ChangeStatusBindingModel { OrderId = id
});
                LoadData();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
    }

    private void ButtonPayOrder_Click(object sender, EventArgs e)
    {
        if (dataGridView.SelectedRows.Count == 1)
        {
            int id = Convert.ToInt32(dataGridView.SelectedRows[0].Cells[0].Value);
            try
            {
                _orderLogic.PayOrder(new ChangeStatusBindingModel { OrderId = id });
                LoadData();
            }
            catch (Exception ex)

```

```

        {
            MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
                MessageBoxIcon.Error);
        }
    }

    private void ButtonRef_Click(object sender, EventArgs e)
    {
        LoadData();
    }
}

```

Листинг 1.20 – Логика формы FormMain

Для вызова форм списков компонент и изделий, а также формы создания заказа используется IUnityContainer. Для смены статусов логика простая, проверяем, что есть выбранная строка, извлекаем идентификатор заказа и вызываем нужный метод.

Останется только изменить вызов главной формы в классе Program. Однако, для этого требуется сперва создать реализации для интерфейсов бизнес-логики и настроить IUnityContainer. Потому перейдем к реализации.

Создадим проект «**AbstractShopListImplement**» для реализации интерфейсов хранения данных. В ссылках у него добавим проект **AbstractShopBusinessLogic**. В качестве хранилищ сущностей будут выступать списки (будем хранить только в оперативной памяти, без сохранения на жестком диске). Для того, чтобы везде был один и тот же список сущностей сделаем класс-Singleton (паттерн Singleton). Нам потребуются классы-модели сущностей (они будут слегка отличаться от классов-моделей в проекте AbstractShopBusinessLogic).

Класс-компонент будет идентичен связывающей модели (BindingModel) сущности «Компонент» (листинг 1.21).

```

namespace AbstractShopListImplement.Models
{
    /// <summary>
    /// Компонент, требуемый для изготовления изделия
    /// </summary>
    public class Component
    {
        public int Id { get; set; }

        public string ComponentName { get; set; }
    }
}

```



```
}
```

Листинг 1.21 – Класс Component

Класс изделия будет немного отличаться от модели в **AbstractShopBusinessLogic**. Список компонент будет храниться в виде пар: идентификатор компонента и количество компонента в изделии (листинг 1.22).

```
using System.Collections.Generic;

namespace AbstractShopListImplement.Models
{
    /// <summary>
    /// Изделие, изготавливаемое в магазине
    /// </summary>
    public class Product
    {
        public int Id { get; set; }

        public string ProductName { get; set; }

        public decimal Price { get; set; }

        public Dictionary<int, int> ProductComponents { get; set; }
    }
}
```

Листинг 1.22 – Класс Product

Класс заказа будет идентичен классу Binding-модели (листинг 1.23).

```
using AbstractShopBusinessLogic.Enums;
using System;

namespace AbstractShopListImplement.Models
{
    /// <summary>
    /// Заказ
    /// </summary>
    public class Order
    {
        public int Id { get; set; }

        public int ProductId { get; set; }

        public int Count { get; set; }

        public decimal Sum { get; set; }

        public OrderStatus Status { get; set; }

        public DateTime DateCreate { get; set; }

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

Листинг 1.23 – Класс Order

Теперь создадим класс для списков (листинг 1.24).

```
using AbstractShopListImplement.Models;
using System.Collections.Generic;

namespace AbstractShopListImplement
{
    public class DataListSingleton
    {
        private static DataListSingleton instance;

        public List<Component> Components { get; set; }

        public List<Order> Orders { get; set; }

        public List<Product> Products { get; set; }

        private DataListSingleton()
        {
            Components = new List<Component>();
            Orders = new List<Order>();
            Products = new List<Product>();
        }

        public static DataListSingleton GetInstance()
        {
            if (instance == null)
            {
                instance = new DataListSingleton();
            }

            return instance;
        }
    }
}
```

Листинг 1.24 – Класс DataListSingleton

Как отмечалось ранее, класс будет реализовывать паттерн Singleton. И в нем будут храниться списки от всех сущностей.

Последний шаг – реализации интерфейсов. Начнем с реализации интерфейса IComponentStorage (листинг 1.25).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopListImplement.Models;
using System;
using System.Collections.Generic;

namespace AbstractShopListImplement.Implements
{
    public class ComponentStorage : IComponentStorage
    {
        private readonly DataListSingleton source;

        public ComponentStorage()
        {
            source = DataListSingleton.GetInstance();
        }
    }
}
```

```

public List<ComponentViewModel> GetFullList()
{
    List<ComponentViewModel> result = new List<ComponentViewModel>();
    foreach (var component in source.Components)
    {
        result.Add(CreateModel(component));
    }
    return result;
}

public List<ComponentViewModel> GetFilteredList(ComponentBindingModel model)
{
    if(model == null)
    {
        return null;
    }

    List<ComponentViewModel> result = new List<ComponentViewModel>();
    foreach (var component in source.Components)
    {
        if (component.ComponentName.Contains(model.ComponentName))
        {
            result.Add(CreateModel(component));
        }
    }
    return result;
}

public ComponentViewModel GetElement(ComponentBindingModel model)
{
    if(model == null)
    {
        return null;
    }

    foreach (var component in source.Components)
    {
        if (component.Id == model.Id || component.ComponentName ==
model.ComponentName)
        {
            return CreateModel(component);
        }
    }

    return null;
}

public void Insert(ComponentBindingModel model)
{
    Component tempComponent = new Component { Id = 1 };
    foreach (var component in source.Components)
    {
        if (component.Id >= tempComponent.Id)
        {
            tempComponent.Id = component.Id + 1;
        }
    }

    source.Components.Add(CreateModel(model, tempComponent));
}

public void Update(ComponentBindingModel model)
{

```

```

        Component tempComponent = null;
        foreach (var component in source.Components)
        {
            if (component.Id == model.Id)
            {
                tempComponent = component;
            }
        }

        if (tempComponent == null)
        {
            throw new Exception("Элемент не найден");
        }

        CreateModel(model, tempComponent);
    }

    public void Delete(ComponentBindingModel model)
    {
        for (int i = 0; i < source.Components.Count; ++i)
        {
            if (source.Components[i].Id == model.Id.Value)
            {
                source.Components.RemoveAt(i);
                return;
            }
        }
        throw new Exception("Элемент не найден");
    }

    private Component CreateModel(ComponentBindingModel model, Component component)
    {
        component.ComponentName = model.ComponentName;

        return component;
    }

    private ComponentViewModel CreateModel(Component component)
    {
        return new ComponentViewModel
        {
            Id = component.Id,
            ComponentName = component.ComponentName
        };
    }
}

```

Листинг 1.25 – Класс ComponentStorage

Для работы со списком будет поле source от класса DataListSingleton (в конструкторе получаем сам объект). При создании также требуется определить новый Id для компонента (ищем максимальный Id и прибавляем 1). Отдельно сделаем методы для получения класса-модели из класса-BindingModel и класса-ViewModel из класса-модели.

Следующим реализуем интерфейс IProductStorage (листинг 1.26).

```

using AbstractShopBusinessLogic.BindingModels;

```

```

using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopListImplement.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopListImplement.Implements
{
    public class ProductStorage : IProductStorage
    {
        private readonly DataListSingleton source;

        public ProductStorage()
        {
            source = DataListSingleton.GetInstance();
        }

        public List<ProductViewModel> GetFullList()
        {
            List<ProductViewModel> result = new List<ProductViewModel>();
            foreach (var component in source.Products)
            {
                result.Add(CreateModel(component));
            }
            return result;
        }

        public List<ProductViewModel> GetFilteredList(ProductBindingModel model)
        {
            if (model == null)
            {
                return null;
            }

            List<ProductViewModel> result = new List<ProductViewModel>();
            foreach (var product in source.Products)
            {
                if (product.ProductName.Contains(model.ProductName))
                {
                    result.Add(CreateModel(product));
                }
            }
            return result;
        }

        public ProductViewModel GetElement(ProductBindingModel model)
        {
            if (model == null)
            {
                return null;
            }
            foreach (var product in source.Products)
            {
                if (product.Id == model.Id || product.ProductName ==
model.ProductName)
                {
                    return CreateModel(product);
                }
            }
            return null;
        }
    }
}

```

```

        public void Insert(ProductBindingModel model)
        {
            Product tempProduct = new Product { Id = 1, ProductComponents = new
Dictionary<int, int>() };
            foreach (var product in source.Products)
            {
                if (product.Id >= tempProduct.Id)
                {
                    tempProduct.Id = product.Id + 1;
                }
            }
            source.Products.Add(CreateModel(model, tempProduct));
        }

        public void Update(ProductBindingModel model)
        {
            Product tempProduct = null;
            foreach (var product in source.Products)
            {
                if (product.Id == model.Id)
                {
                    tempProduct = product;
                }
            }
            if (tempProduct == null)
            {
                throw new Exception("Элемент не найден");
            }

            CreateModel(model, tempProduct);
        }

        public void Delete(ProductBindingModel model)
        {
            for (int i = 0; i < source.Products.Count; ++i)
            {
                if (source.Products[i].Id == model.Id)
                {
                    source.Products.RemoveAt(i);
                    return;
                }
            }
            throw new Exception("Элемент не найден");
        }

        private Product CreateModel(ProductBindingModel model, Product product)
        {
            product.ProductName = model.ProductName;
            product.Price = model.Price;
            // удаляем убранные
            foreach (var key in product.ProductComponents.Keys.ToList())
            {
                if (!model.ProductComponents.ContainsKey(key))
                {
                    product.ProductComponents.Remove(key);
                }
            }
            // обновляем существующие и добавляем новые
            foreach (var component in model.ProductComponents)
            {
                if (product.ProductComponents.ContainsKey(component.Key))
                {
                    product.ProductComponents[component.Key] =
model.ProductComponents[component.Key].Item2;

```

```

        }
        else
        {
            product.ProductComponents.Add(component.Key,
model.ProductComponents[component.Key].Item2);
        }
    }

    return product;
}

private ProductViewModel CreateModel(Product product)
{
    // требуется дополнительно получить список компонентов для изделия с
названиями и их количество
    Dictionary<int, (string, int)> productComponents = new
Dictionary<int, (string, int)>();
    foreach (var pc in product.ProductComponents)
    {
        string componentName = string.Empty;
        foreach (var component in source.Components)
        {
            if (pc.Key == component.Id)
            {
                componentName = component.ComponentName;
                break;
            }
        }
        productComponents.Add(pc.Key, (componentName, pc.Value));
    }
    return new ProductViewModel
    {
        Id = product.Id,
        ProductName = product.ProductName,
        Price = product.Price,
        ProductComponents = productComponents
    };
}
}
}
}

```

Листинг 1.26 – Класс ProductLogic

Здесь логика будет сложнее. При получении списка и элемента потребуется делать преобразования из списка в классе-модели для получения списка компонентов изделия с их названиями. Также при создании и редактировании следует сохранять и обновлять списки компонент.

Последний шаг – в проекте **AbstractShopView** подключаем проект **AbstractShopListImplement**. В классе Program настраиваем IUnityContainer и вызываем главную форму при запуске программы (листинг 1.27).

```

using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopListImplement.Implements;
using System;
using System.Windows.Forms;
using Unity;

```

```

using Unity.Lifetime;

namespace AbstractShopView
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            var container = BuildUnityContainer();

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(container.Resolve<FormMain>());
        }

        private static IUnityContainer BuildUnityContainer()
        {
            var currentContainer = new UnityContainer();
            currentContainer.RegisterType<IComponentStorage, ComponentStorage>(new
            HierarchicalLifetimeManager());
            currentContainer.RegisterType<IOrderStorage, OrderStorage>(new
            HierarchicalLifetimeManager());
            currentContainer.RegisterType<IProductStorage, ProductStorage>(new
            HierarchicalLifetimeManager());

            currentContainer.RegisterType<ComponentLogic>(new
            HierarchicalLifetimeManager());
            currentContainer.RegisterType<OrderLogic>(new HierarchicalLifetimeManager());
            currentContainer.RegisterType<ProductLogic>(new
            HierarchicalLifetimeManager());

            return currentContainer;
        }
    }
}

```

Листинг 1.27 – Класс Program

Главная форма представлена на рисунке 1.12.



Рисунок 1.12 – Главная форма

Создание компонента представлено на рисунке 1.13.

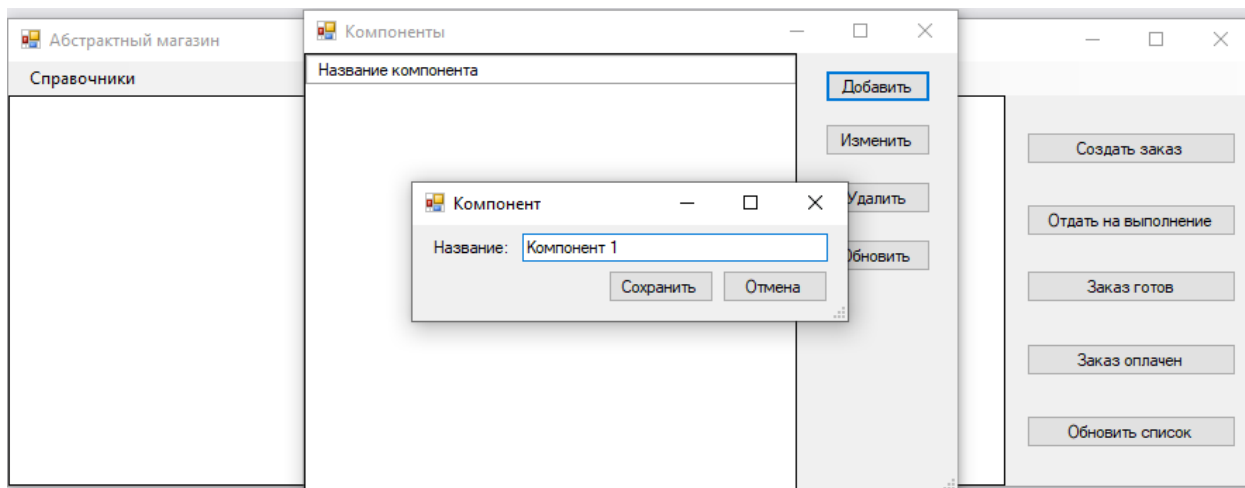


Рисунок 1.13 – Создание компонента

Создание изделия и добавление компонента к изделию представлено на рисунке 1.14.

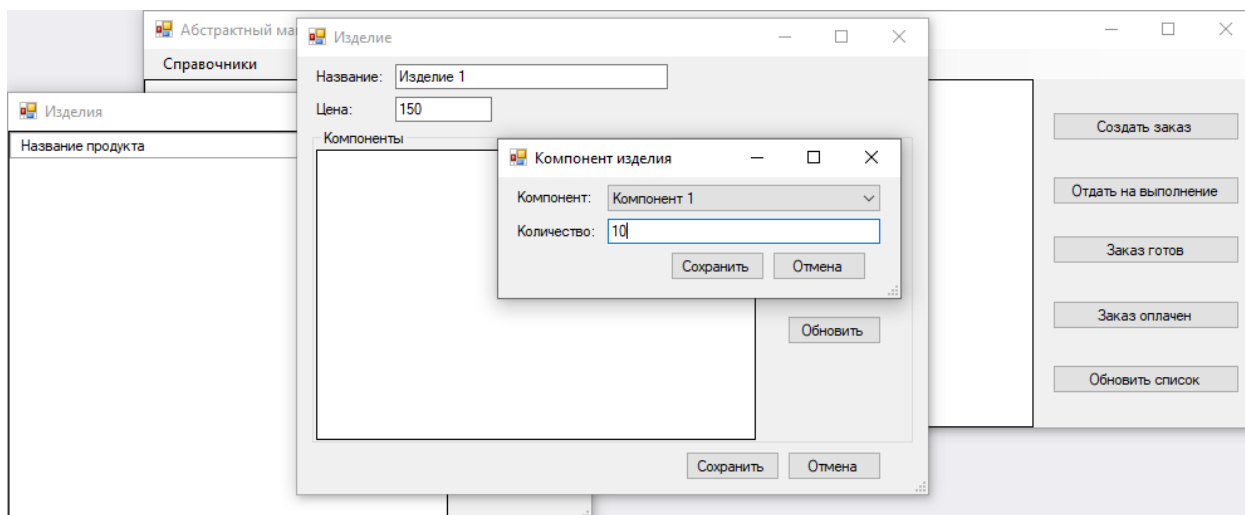


Рисунок 1.14 – Добавление компонента к изделию

Создание заказа представлено на рисунке 1.15.

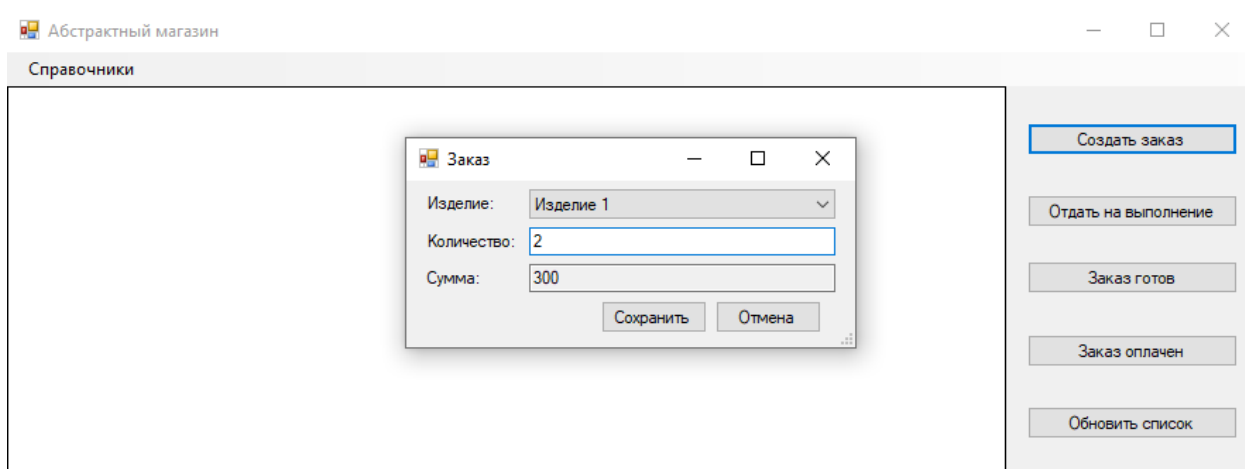


Рисунок 1.15 – Создание заказа

Проверка на неверный статус представлено на рисунке 1.16.

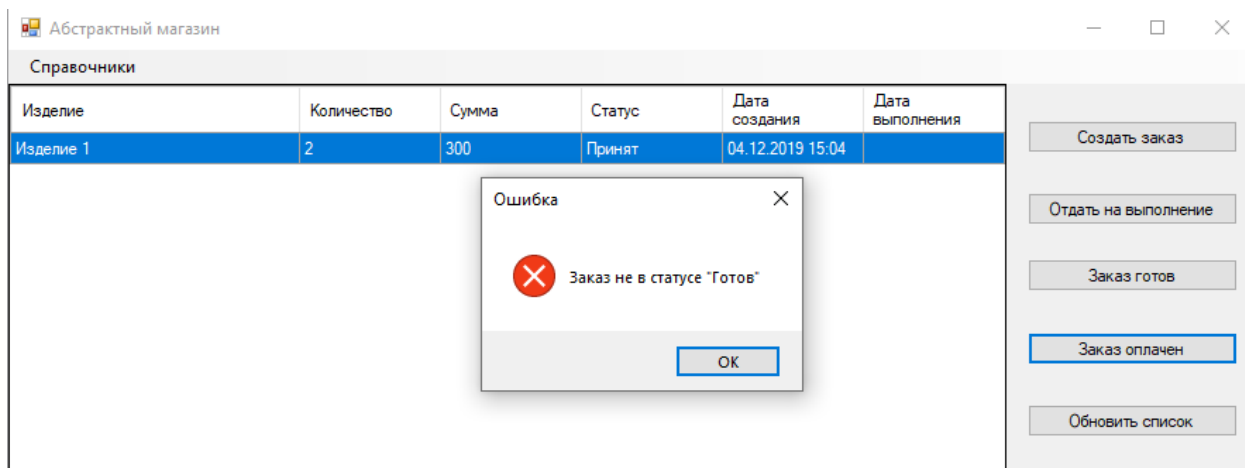


Рисунок 1.16 – Проверка на неверный статус

Правильный результат смены статуса представлен на рисунке 1.17.

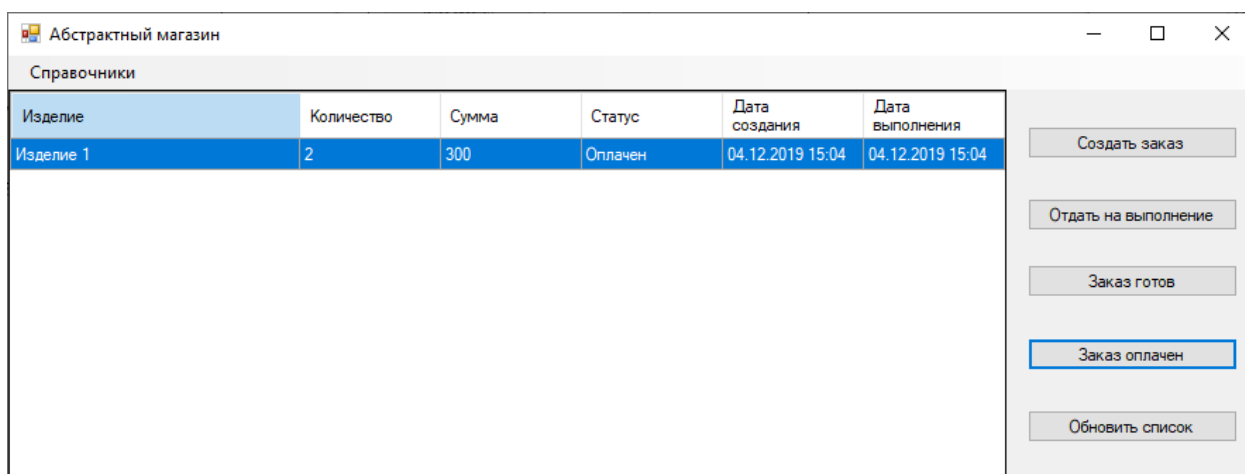


Рисунок 1.17 – Заказ со статусом «Оплачен»

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. Создать логику для изделий, по аналогии с логикой по компонентам.
6. В OrderLogic прописать логику для метода PayOrder.
7. Создать форму и логику для списка изделий по аналогии с формой и логикой для списка компонент.
8. В форме FormCreateOrder дописать логику загрузки списка изделий в выпадающий список.
9. В форме FormMain дописать логику вывода списка заказов на форму.
10. Создать реализацию для интерфейса IOrderStorage.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- создать сущность «Склад» для хранения информации по складам (не должно быть 2-х складов с одним и тем же названием) поля: название, ФИО ответственного, дата создания склада;
- предусмотреть возможность хранения любого компонента на любом складе в любом количестве;
- в логике хранения данных по складам реализовать функционал пополнения склада компонентом на определенное количество;
- в реализации интерфейса склада предусмотреть вывод складов со списком компонент, находящихся на этих складах, но не делать в логике добавления и редактирования склада работу с компонентами (т.е. получение списка и элемента как у «Изделия», а добавление и редактирование как у «Компонента», удаление как у «Изделия»);

- создать форму для отображения списка складов (аналогичную как для «Компонента» или «Изделия»);
- создать форму для работы со складом (ввод данных по складу и отображение списка компонент на этом складе, но без возможности добавления/редактирования/удаления компонент на складе);
- на главной форме в меню в пункте «Справочники» добавить пункт «Склады» и вызывать через него форму со списком складов;
- создать форму для пополнения склада (на форме предусмотреть возможность выбора склада, выбора компонента и ввода количества единиц пополнения);
- на главной форме в меню добавить пункт «Пополнение склада» и вызывать форму пополнения склада при выборе этого пункта.

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).

- 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).

ЛАБОРАТОРНАЯ РАБОТА №2.

LINQ

Цель

Научиться работать с LINQ-запросами.

Задание

1. Создать ветку от ветки первой лабораторной.
2. Создать новую реализацию интерфейсов. Использовать файлы для хранения данных. В обработке пользоваться LINQ-запросами.
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Решение

Создадим новый проект **AbstractShopFileImplement**. Модели будут идентичны моделям из проекта **AbstractShopListImplement** и мы на них не будем останавливаться. Класс со списками изменится. Он также будет реализован на основе паттерна Singleton. Однако появятся ряд новых методов. При работе с файлами возникает проблема, когда сохранять? Если на каждое действие пользователя будут производиться манипуляции с файлами (чтение/запись), то это создаст дополнительную нагрузку на жесткий диск и каналы связи, а также замедлит (для современных компьютеров незначительно) работу программы. Самое логичное – при старте программы загружать из файлов данные, а при завершении программы – выгружать их. Вопрос: как это сделать, не изменяя логики приложения? Для загрузки все просто, у нас есть класс Singleton, в котором объявлены все списки. Объект от класса создается при первой обращении к этому классу и при создании можно загружать разом данные из всех файлов. А когда выгружать данные? Тут тоже можно воспользоваться этим классом. У любого класса есть деструктор, который вызывается сборщиком мусора при очистке объекта класса из памяти (когда на него нет ни одной ссылки). Для нашего класса-Singleton есть только один экземпляр, который будет существовать все время работы программы и отчистится (вызовется

деструктор) только при завершении программы. Этим и воспользуемся (листинг 2.1).

```
using AbstractShopBusinessLogic.Enums;
using AbstractShopFileImplement.Models;
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Linq;
using System.Xml.Serialization;

namespace AbstractShopFileImplement
{
    public class FileDataListSingleton
    {
        private static FileDataListSingleton instance;

        private readonly string ComponentFileName = "Component.xml";

        private readonly string OrderFileName = "Order.xml";

        private readonly string ProductFileName = "Product.xml";

        public List<Component> Components { get; set; }

        public List<Order> Orders { get; set; }

        public List<Product> Products { get; set; }

        private FileDataListSingleton()
        {
            Components = LoadComponents();
            Orders = LoadOrders();
            Products = LoadProducts();
        }

        public static FileDataListSingleton GetInstance()
        {
            if (instance == null)
            {
                instance = new FileDataListSingleton();
            }

            return instance;
        }

        ~FileDataListSingleton()
        {
            SaveComponents();
            SaveOrders();
            SaveProducts();
        }

        private List<Component> LoadComponents()
        {
            var list = new List<Component>();

            if (File.Exists(ComponentFileName))
            {
                XDocument xDocument = XDocument.Load(ComponentFileName);

                var xElements = xDocument.Root.Elements("Component").ToList();
            }
        }
    }
}
```

```

        foreach (var elem in xElements)
        {
            list.Add(new Component
            {
                Id = Convert.ToInt32(elem.Attribute("Id").Value),
                ComponentName = elem.Element("ComponentName").Value
            });
        }
    }

    return list;
}

private List<Order> LoadOrders()
{
    // прописать логику
}

private List<Product> LoadProducts()
{
    var list = new List<Product>();

    if (File.Exists(ProductFileName))
    {
        XDocument xDocument = XDocument.Load(ProductFileName);

        var xElements = xDocument.Root.Elements("Product").ToList();

        foreach (var elem in xElements)
        {
            var prodComp = new Dictionary<int, int>();
            foreach (var component in
elem.Element("ProductComponents").Elements("ProductComponent").ToList())
            {
                prodComp.Add(Convert.ToInt32(component.Element("Key").Value),
Convert.ToInt32(component.Element("Value").Value));
            }
            list.Add(new Product
            {
                Id = Convert.ToInt32(elem.Attribute("Id").Value),
                ProductName = elem.Element("ProductName").Value,
                Price = Convert.ToDecimal(elem.Element("Price").Value),
                ProductComponents = prodComp
            });
        }
    }

    return list;
}

private void SaveComponents()
{
    if (Components != null)
    {
        var xElement = new XElement("Components");

        foreach (var component in Components)
        {
            xElement.Add(new XElement("Component",
                new XAttribute("Id", component.Id),
                new XElement("ComponentName", component.ComponentName)));
        }
    }
}

```

```

        XmlDocument xDocument = new XmlDocument(xElement);
        xDocument.Save(ComponentFileName);
    }
}

private void SaveOrders()
{
    // прописать логику
}

private void SaveProducts()
{
    if (Products != null)
    {
        var xElement = new XElement("Products");

        foreach (var product in Products)
        {
            var compElement = new XElement("ProductComponents");
            foreach (var component in product.ProductComponents)
            {
                compElement.Add(new XElement("ProductComponent",
                    new XElement("Key", component.Key),
                    new XElement("Value", component.Value)));
            }
            xElement.Add(new XElement("Product",
                new XAttribute("Id", product.Id),
                new XElement("ProductName", product.ProductName),
                new XElement("Price", product.Price),
                compElement));
        }

        XmlDocument xDocument = new XmlDocument(xElement);
        xDocument.Save(ProductFileName);
    }
}
}
}

```

Листинг 2.1 – Класс FileDataListSingleton

Остается реализовать 3 интерфейса с проекта **AbstractShopBusinessLogic**.

Реализация интерфейса IComponentStorage представлена в листинге 2.2.

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopFileImplement.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopFileImplement.Implements
{
    public class ComponentStorage : IComponentStorage
    {
        private readonly FileDataListSingleton source;
    }
}

```

```

public ComponentStorage()
{
    source = FileDataListSingleton.GetInstance();
}

public List<ComponentViewModel> GetFullList()
{
    return source.Components
        .Select(CreateModel)
        .ToList();
}

public List<ComponentViewModel> GetFilteredList(ComponentBindingModel model)
{
    if (model == null)
    {
        return null;
    }

    return source.Components
        .Where(rec => rec.ComponentName.Contains(model.ComponentName))
        .Select(CreateModel)
        .ToList();
}

public ComponentViewModel GetElement(ComponentBindingModel model)
{
    if (model == null)
    {
        return null;
    }

    var component = source.Components
        .FirstOrDefault(rec => rec.ComponentName == model.ComponentName ||
rec.Id == model.Id);

    return component != null ? CreateModel(component) : null;
}

public void Insert(ComponentBindingModel model)
{
    int maxId = source.Components.Count > 0 ? source.Components.Max(rec =>
rec.Id) : 0;
    var element = new Component { Id = maxId + 1 };
    source.Components.Add(CreateModel(model, element));
}

public void Update(ComponentBindingModel model)
{
    var element = source.Components.FirstOrDefault(rec => rec.Id == model.Id);
    if (element == null)
    {
        throw new Exception("Элемент не найден");
    }

    CreateModel(model, element);
}

public void Delete(ComponentBindingModel model)
{
    Component element = source.Components.FirstOrDefault(rec => rec.Id ==
model.Id);
    if (element != null)
    {

```

```

        source.Components.Remove(element);
    }
    else
    {
        throw new Exception("Элемент не найден");
    }
}

private Component CreateModel(ComponentBindingModel model, Component component)
{
    component.ComponentName = model.ComponentName;

    return component;
}

private ComponentViewModel CreateModel(Component component)
{
    return new ComponentViewModel
    {
        Id = component.Id,
        ComponentName = component.ComponentName
    };
}
}
}

```

Листинг 2.2 – Класс ComponentStorage

Select вернет тип `IEnumerable<ComponentViewModel>`. Так как вернуть требуется тип `List<ComponentViewModel>`, то используем дополнительно метод `ToList()`.

`FirstOrDefault` работает по следующему принципу: в выборке (в нашем случае списке) ищется элемент, согласно указанным условиям. Если в выборке есть такие элементы, то метод вернет первый встреченный из них. Если в выборке нет элементов, удовлетворяющих условию, то вернется значение `null`. Есть еще один похожий на этот метод: `SingleOrDefault`. Но он вернет ошибку, если в выборке есть 2 или более элементов, удовлетворяющих условию. Также есть методы `First` и `Single`, но они вернут ошибку, если не найдут в выборке элементы, удовлетворяющие условию (а `Single` еще, если найдет 2 или более). В методе `FirstOrDefault` мы задаем лямбда-выражение для поиска элементов с таким же идентификатором, который получили из входных параметров метода. Результат сохраняем в переменную. Далее идет проверка, если переменная не равна `null`, значит мы нашли в списке запись с таким идентификатором и можем с ней дальше работать. Иначе – выкинем ошибку.

Методы добавления, редактирования и удаления также преобразуются. Там для получения максимального значения идентификатора, для присвоения новой записи используется метод Max.

Реализация интерфейса IProductStorage представлена в листинге 2.3.

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopFileImplement.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopFileImplement.Implements
{
    public class ProductStorage : IProductStorage
    {
        private readonly FileDataListSingleton source;

        public ProductStorage()
        {
            source = FileDataListSingleton.GetInstance();
        }

        public List<ProductViewModel> GetFullList()
        {
            return source.Products
                .Select(CreateModel)
                .ToList();
        }

        public List<ProductViewModel> GetFilteredList(ProductBindingModel model)
        {
            if (model == null)
            {
                return null;
            }

            return source.Products
                .Where(rec => rec.ProductName.Contains(model.ProductName))
                .Select(CreateModel)
                .ToList();
        }

        public ProductViewModel GetElement(ProductBindingModel model)
        {
            if (model == null)
            {
                return null;
            }

            var product = source.Products
                .FirstOrDefault(rec => rec.ProductName == model.ProductName || rec.Id
                == model.Id);

            return product != null ? CreateModel(product) : null;
        }

        public void Insert(ProductBindingModel model)
        {

```

```

        int maxId = source.Products.Count > 0 ? source.Components.Max(rec => rec.Id)
: 0;
        var element = new Product { Id = maxId + 1, ProductComponents = new
Dictionary<int, int>() };
        source.Products.Add(CreateModel(model, element));
    }

    public void Update(ProductBindingModel model)
    {
        var element = source.Products.FirstOrDefault(rec => rec.Id == model.Id);
        if (element == null)
        {
            throw new Exception("Элемент не найден");
        }

        CreateModel(model, element);
    }

    public void Delete(ProductBindingModel model)
    {
        Product element = source.Products.FirstOrDefault(rec => rec.Id == model.Id);
        if (element != null)
        {
            source.Products.Remove(element);
        }
        else
        {
            throw new Exception("Элемент не найден");
        }
    }

    private Product CreateModel(ProductBindingModel model, Product product)
    {
        product.ProductName = model.ProductName;
        product.Price = model.Price;
        // удаляем убранные
        foreach (var key in product.ProductComponents.Keys.ToList())
        {
            if (!model.ProductComponents.ContainsKey(key))
            {
                product.ProductComponents.Remove(key);
            }
        }
        // обновляем существующие и добавляем новые
        foreach (var component in model.ProductComponents)
        {
            if (product.ProductComponents.ContainsKey(component.Key))
            {
                product.ProductComponents[component.Key] =
model.ProductComponents[component.Key].Item2;
            }
            else
            {
                product.ProductComponents.Add(component.Key,
model.ProductComponents[component.Key].Item2);
            }
        }

        return product;
    }

    private ProductViewModel CreateModel(Product product)
    {
        return new ProductViewModel

```

```

        {
            Id = product.Id,
            ProductName = product.ProductName,
            Price = product.Price,
            ProductComponents = product.ProductComponents
                                .ToDictionary(recPC => recPC.Key, recPC =>
                                (source.Components.FirstOrDefault(recC => recC.Id ==
recPC.Key)?.ComponentName, recPC.Value))
        };
    }
}

```

Листинг 2.3 – Класс ProductStorage

Здесь используется метод ToDictionary для преобразования списка компонент изделия в нужный формат.

Потребуется также реализовать интерфейс IOrderStorage.

Последний штрих. В проект **AbstractShopView** добавить ссылку на **AbstractShopFileImplement**. И в классе Program поменять всего одну строку.

Вместо

```
using AbstractShopListImplement.Implements;
```

Вставить

```
using AbstractShopFileImplement.Implements;
```

В результате приложение будет работать с другой реализацией и хранить данные в файлах.

После первого запуска приложения появятся 4 файла с сохранёнными данными. При повторных запусках приложения данные будут сразу доступны пользователям.

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. В классе FileDataListSingleton прописать логику сохранения и загрузки данных для «Заказа».

4. Реализовать логику для класса OrderStorage с использованием LINQ-запросов.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- сохранять в файлы информацию по складам и их загруженность;
- сделать реализацию интерфейса логики хранения данных для сущности «Склад» с сохранением в файл с использованием LINQ-запросов (не должно быть 2-х складов с одним и тем же названием);
- в логике для хранения данных «Склад» прописать метод проверки наличия на складах компонентов в нужном количестве (нужно считать суммарное количество каждого компонента на складах и сравнивать с требуемым количеством) и списания компонент со складов в требуемом количестве (возможно, потребуется списать сразу с нескольких складов компонент, если на одном нет компонента в нужном количестве);
- в реализации логики «Складов» с хранением данных в оперативной памяти метод проверки и списания оставить нереализованным;
- в бизнес-логике работы с заказами при отправке заказа в работу вызывать метод проверки и списания со складов компонент в требуемом количестве, если компонент недостаточно, то не давать переводить заказ в работу;

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
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).

ЛАБОРАТОРНАЯ РАБОТА №3.

РАБОТА С БД

Цель

Изучить работу с базами данных. Подключаться к СУБД, извлекать данные, сохранять данные, удалять.

Задание

1. Создать ветку от ветки второй лабораторной.
2. Создать новую реализацию сервисов со способом хранения данных в БД. В качестве СУБД использовать MS SQL Server. При работе с СУБД использовать *EntityFrameworkCore*.
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Решение

Для работы с БД будем использовать Entity Framework Core. Подключаем его через NuGet пакеты. Создадим новый проект **AbstractShopDatabaseImplement** для работы с базой данных. Добавим в него следующие пакеты (рисунок 3.1) (их сразу можно продублировать в проект **AbstractShopView**):

- Microsoft.EntityFrameworkCore;
- Microsoft.EntityFrameworkCore.SqlServer;
- Microsoft.EntityFrameworkCore.Tools;

Добавляем модели. Тут будут отличия по сравнению с моделями для обычных списков. Так как на основе этих моделей будут созданы таблицы в базе данных, и они должны быть связанные, то потребуется настроить связи между таблицами в классах-моделях. Также, таблицы в БД не могут хранить списки, массивы и т.п., так что придется отказаться от словаря в модели «Изделие». Его придется заменить на отдельный класс-связь «Компонента» и «Изделия».

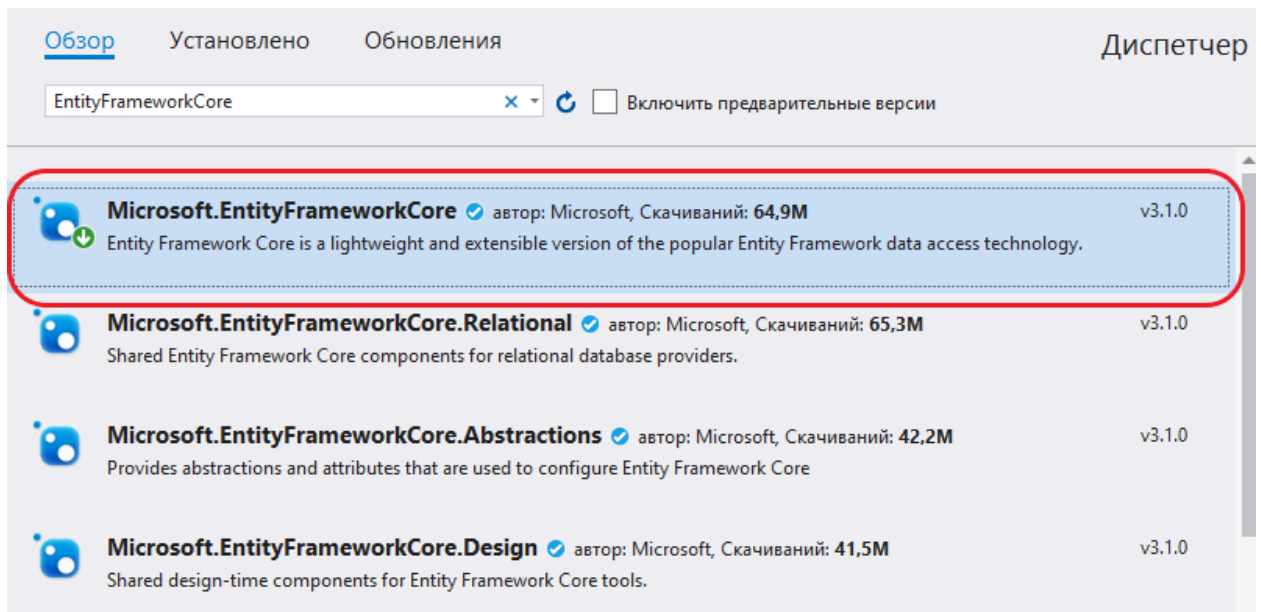


Рисунок 3.1 – Добавление EntityFrameworkCore

Сперва создадим все классы-модели на основе классов-моделей из проекта **AbstractShopFileImplement** или **AbstractShopListImplement**. Далее будем настраивать связи и ограничения.

Для компонента укажем, что поле-название обязательно к заполнению и что этот класс (таблица) будет связан с классом связи компонента и продукта типом связи один-ко-многим (по сути, у нас будет связь компонента и изделия по типу многие-ко-многим через отдельную сущность) (листинг 3.1).

```
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AbstractShopDatabaseImplement.Models
{
    /// <summary>
    /// Компонент, требуемый для изготовления изделия
    /// </summary>
    public class Component
    {
        public int Id { get; set; }

        [Required]
        public string ComponentName { get; set; }

        [ForeignKey("ComponentId")]
        public virtual List<ProductComponent> ProductComponents { get; set; }
    }
}
```

Листинг 3.1 – Класс Component

Атрибут Required указывает на то, что поле обязательно к заполнению. Также указываем в качестве списка, что у нас есть связь один ко многим к таблице связи с изделиями. У списка указываем атрибут с внешним ключом (поле в таблице связи с изделиями). У класса-связи в таком случае, должно быть поле с таким названием (листинг 3.2).

```
using System.ComponentModel.DataAnnotations;

namespace AbstractShopDatabaseImplement.Models
{
    /// <summary>
    /// Сколько компонентов, требуется при изготовлении изделия
    /// </summary>
    public class ProductComponent
    {
        public int Id { get; set; }

        public int ProductId { get; set; }

        public int ComponentId { get; set; }

        [Required]
        public int Count { get; set; }

        public virtual Component Component { get; set; }

        public virtual Product Product { get; set; }
    }
}
```

Листинг 3.2 – Класс ProductComponent

У класса-связи прописываем новое поле-ссылку на объект класса «Компонент». Таким образом, при создании таблиц в базе данных пропишется связь между этими таблицами.

У класса «Изделие» будет 2 связи: с классом-связи (компонент-изделие) и заказом (один-ко-многим). У класса обязательно должны заполняться поля «Название» и «Цена». Прописать класс самостоятельно.

Класс «Заказ» должен быть связан с «Изделием». Обязательным к заполнению у него будут поля «Количество», «Сумма», «Статус» и «Дата создания». Прописать класс самостоятельно.

Теперь создадим класс, наследник от класса DbContext, в котором пропишем наборы от наших классов (на основе этого и создается база данных с таблицами) (листинг 3.3).

```

using AbstractShopDatabaseImplement.Models;
using Microsoft.EntityFrameworkCore;

namespace AbstractShopDatabaseImplement
{
    public class AbstractShopDatabase : DbContext
    {
        protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        {
            if (optionsBuilder.IsConfigured == false)
            {
                optionsBuilder.UseSqlServer(@"Data Source=HOME\SQLEXPRESS;Initial
Catalog=AbstractShopDatabase;Integrated Security=True;MultipleActiveResultSets=True;");
            }
            base.OnConfiguring(optionsBuilder);
        }

        public virtual DbSet<Component> Components { set; get; }

        public virtual DbSet<Product> Products { set; get; }

        public virtual DbSet<ProductComponent> ProductComponents { set; get; }

        public virtual DbSet<Order> Orders { set; get; }
    }
}

```

Листинг 3.3 – Класс AbstractShopDatabase

В классе перегрузим метод OnConfiguring, где пропишем строку подключения к базе данных. Строка подключения состоит из нескольких настроек. Нас интересует 2 из них:

- Data Source – имя хоста (компьютера) и название СУБД, к которой хотим подключиться;
- Catalog – имя базы данных.

Данные настройки следует изменить для своего рабочего места и варианта задания.

Далее через DbSet прописываются таблицы, которые следует создать в базе данных.

Следующий шаг – создание базы данных. Для этого нам понадобится «Консоль диспетчера пакетов». Ее можно открыть через «Средства -> Диспетчер пакетов NuGet». В открывшейся вкладке выбираем «Проект по умолчанию» проект с файлом от DbContext (**AbstractShopDatabaseImplement**). В нем будем прописывать команду создания миграции (рисунок 3.2).

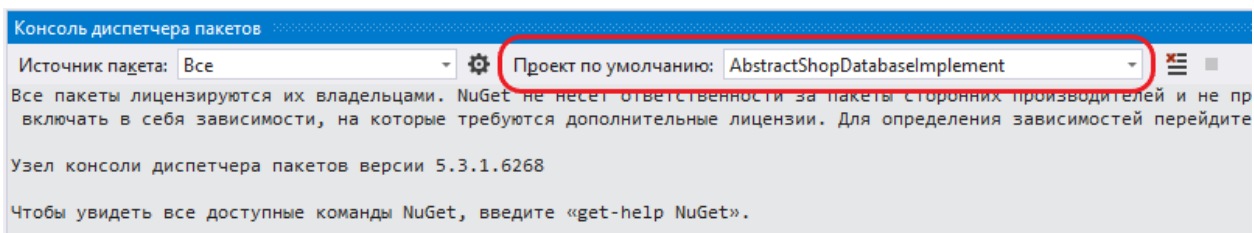


Рисунок 3.2 – Консоль диспетчера пакетов

В Консоли диспетчера пакетов прописываем команду создания миграции (листинг 3.4).

```
Add-Migration InitialCreate
```

Листинг 3.4 – Команда создания миграции

Здесь InitialCreate – название миграции (как правило для первой пишут, что это первая миграция или инициализация);

После успешного выполнения в проекте появится папка «Migrations». В ней будет храниться снимок базы данных и все миграции, создаваемые в ходе разработки (рисунок 3.3).

У нас есть снимок БД, но нет самой базы. Чтобы она появилась есть 2 варианта: запустить проект и сделать запрос к БД (если БД нет, она создастся на основе снимка), либо в консоли применить команду изменения базы данных. Первый вариант нам не подходит, так как мы еще не прописали реализации интерфейсов и не подменили их в проекте AbstractShopView. Для второго варианта открываем консоль и прописываем команду (листинг 3.5).

Если все сделано верно, то в результате будет создана (или обновлена) база данных (рисунок 3.4).

Остается прописать реализации интерфейсов. Для начала, сделаем реализацию для компонента (листинг 3.6).

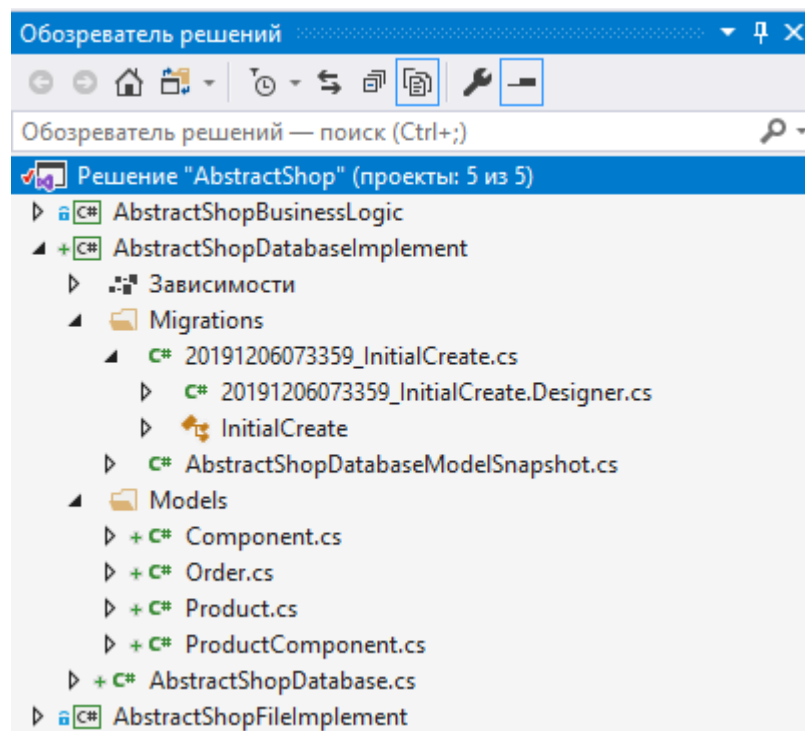


Рисунок 3.3 – Обозреватель решений

Update-Database

Листинг 3.5 – Команда применения миграции

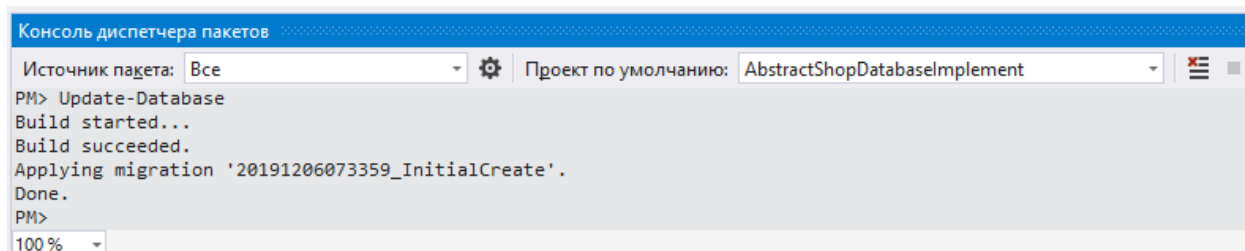


Рисунок 3.4 – Результат выполнения команды

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopDatabaseImplement.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopDatabaseImplement.Implements
{
    public class ComponentStorage : IComponentStorage
    {
        public List<ComponentViewModel> GetFullList()
        {
            using (var context = new AbstractShopDatabase())
            {
                return context.Components
                    .Select(rec => new ComponentViewModel
                    {
                        Id = rec.Id,
                        ComponentName = rec.ComponentName
                    })
                    .ToList();
            }
        }
    }
}
```

```

        })
        .ToList();
    }
}

public List<ComponentViewModel> GetFilteredList(ComponentBindingModel model)
{
    if (model == null)
    {
        return null;
    }

    using (var context = new AbstractShopDatabase())
    {
        return context.Components
            .Where(rec => rec.ComponentName.Contains(model.ComponentName))
            .Select(rec => new ComponentViewModel
            {
                Id = rec.Id,
                ComponentName = rec.ComponentName
            })
            .ToList();
    }
}

public ComponentViewModel GetElement(ComponentBindingModel model)
{
    if (model == null)
    {
        return null;
    }

    using (var context = new AbstractShopDatabase())
    {
        var component = context.Components
            .FirstOrDefault(rec => rec.ComponentName == model.ComponentName ||
rec.Id == model.Id);

        return component != null ?
            new ComponentViewModel
            {
                Id = component.Id,
                ComponentName = component.ComponentName
            } :
            null;
    }
}

public void Insert(ComponentBindingModel model)
{
    using (var context = new AbstractShopDatabase())
    {
        context.Components.Add(CreateModel(model, new Component()));
        context.SaveChanges();
    }
}

public void Update(ComponentBindingModel model)
{
    using (var context = new AbstractShopDatabase())
    {
        var element = context.Components.FirstOrDefault(rec => rec.Id ==
model.Id);
        if (element == null)

```

```

        {
            throw new Exception("Элемент не найден");
        }
        CreateModel(model, element);
        context.SaveChanges();
    }
}

public void Delete(ComponentBindingModel model)
{
    using (var context = new AbstractShopDatabase())
    {
        Component element = context.Components.FirstOrDefault(rec => rec.Id ==
model.Id);
        if (element != null)
        {
            context.Components.Remove(element);
            context.SaveChanges();
        }
        else
        {
            throw new Exception("Элемент не найден");
        }
    }
}

private Component CreateModel(ComponentBindingModel model, Component component)
{
    component.ComponentName = model.ComponentName;

    return component;
}
}

```

Листинг 3.6 – Класс ComponentStorage

Рассмотрим, что здесь происходит. В каждом методе сперва подключаемся к БД. Можно сделать, чтобы у нас был один объект от AbstractShopDatabase и мы имели постоянное соединение с базой, но это не лучшее решение. Лучше, каждый раз подключаться, выполнять нужные действия и отключаться, позволяя другим клиентам подключаться к этой базе данных (так как количество подключений ограничено). Далее идут запросы к таблице базы данных (в данном случае к таблице «Components») с получением либо набора данных, либо конкретного значения. При добавлении нам не нужно теперь вычислять идентификатор. EntityFramework распознает поля с Id и сам применяет к ним необходимые параметры, в частности, подобное поле в БД помечается как первичный ключ и, если это число, оно будет автоинкрементным, так что при добавлении можно убрать логику вычисления maxId. При добавлении/редактировании/удалении

последним действием будет вызов метода `SaveChanges` у `context` для сохранения изменений в БД (без этой команды все изменения будут сохраняться в копии БД в программе и не зафиксируются в реальной БД).

Теперь класс-реализация интерфейса `IProductStorage` (листинг 3.7).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopDatabaseImplement.Models;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopDatabaseImplement.Implements
{
    public class ProductStorage : IProductStorage
    {
        public List<ProductViewModel> GetFullList()
        {
            using (var context = new AbstractShopDatabase())
            {
                return context.Products
                    .Include(rec => rec.ProductComponents)
                    .ThenInclude(rec => rec.Component)
                    .ToList()
                    .Select(rec => new ProductViewModel
                    {
                        Id = rec.Id,
                        ProductName = rec.ProductName,
                        Price = rec.Price,
                        ProductComponents = rec.ProductComponents
                            .ToDictionary(recPC => recPC.ComponentId, recPC =>
                                (recPC.Component?.ComponentName, recPC.Count))
                    })
                    .ToList();
            }
        }

        public List<ProductViewModel> GetFilteredList(ProductBindingModel model)
        {
            if (model == null)
            {
                return null;
            }

            using (var context = new AbstractShopDatabase())
            {
                return context.Products
                    .Include(rec => rec.ProductComponents)
                    .ThenInclude(rec => rec.Component)
                    .Where(rec => rec.ProductName.Contains(model.ProductName))
                    .ToList()
                    .Select(rec => new ProductViewModel
                    {
                        Id = rec.Id,
                        ProductName = rec.ProductName,
                        Price = rec.Price,
                        ProductComponents = rec.ProductComponents
                            .ToDictionary(recPC => recPC.ComponentId, recPC =>
```

```

(recPC.Component?.ComponentName, recPC.Count))
        })
        .ToList();
    }
}

public ProductViewModel GetElement(ProductBindingModel model)
{
    if (model == null)
    {
        return null;
    }

    using (var context = new AbstractShopDatabase())
    {
        var product = context.Products
            .Include(rec => rec.ProductComponents)
            .ThenInclude(rec => rec.Component)
            .FirstOrDefault(rec => rec.ProductName == model.ProductName || rec.Id
== model.Id);

        return product != null ?
            new ProductViewModel
            {
                Id = product.Id,
                ProductName = product.ProductName,
                Price = product.Price,
                ProductComponents = product.ProductComponents
                    .ToDictionary(recPC => recPC.ComponentId, recPC =>
(recPC.Component?.ComponentName, recPC.Count))
            } :
            null;
    }
}

public void Insert(ProductBindingModel model)
{
    using (var context = new AbstractShopDatabase())
    {
        using (var transaction = context.Database.BeginTransaction())
        {
            try
            {
                context.Products.Add(CreateModel(model, new Product(), context));
                context.SaveChanges();

                transaction.Commit();
            }
            catch
            {
                transaction.Rollback();
                throw;
            }
        }
    }
}

public void Update(ProductBindingModel model)
{
    using (var context = new AbstractShopDatabase())
    {
        using (var transaction = context.Database.BeginTransaction())
        {
            try

```

```

        {
            var element = context.Products.FirstOrDefault(rec => rec.Id ==
model.Id);

            if (element == null)
            {
                throw new Exception("Элемент не найден");
            }
            CreateModel(model, element, context);
            context.SaveChanges();

            transaction.Commit();
        }
        catch
        {
            transaction.Rollback();
            throw;
        }
    }
}

public void Delete(ProductBindingModel model)
{
    using (var context = new AbstractShopDatabase())
    {
        Product element = context.Products.FirstOrDefault(rec => rec.Id ==
model.Id);

        if (element != null)
        {
            context.Products.Remove(element);
            context.SaveChanges();
        }
        else
        {
            throw new Exception("Элемент не найден");
        }
    }
}

private Product CreateModel(ProductBindingModel model, Product product,
AbstractShopDatabase context)
{
    product.ProductName = model.ProductName;
    product.Price = model.Price;

    if (model.Id.HasValue)
    {
        var productComponents = context.ProductComponents.Where(rec =>
rec.ProductId == model.Id.Value).ToList();
        // удалили те, которых нет в модели
        context.ProductComponents.RemoveRange(productComponents.Where(rec =>
!model.ProductComponents.ContainsKey(rec.ComponentId)).ToList());
        context.SaveChanges();
        // обновили количество у существующих записей
        foreach (var updateComponent in productComponents)
        {
            updateComponent.Count =
model.ProductComponents[updateComponent.ComponentId].Item2;
            model.ProductComponents.Remove(updateComponent.ComponentId);
        }
        context.SaveChanges();
    }
    // добавили новые
    foreach (var pc in model.ProductComponents)

```

```

        {
            context.ProductComponents.Add(new ProductComponent
            {
                ProductId = product.Id,
                ComponentId = pc.Key,
                Count = pc.Value.Item2
            });
            context.SaveChanges();
        }

        return product;
    }
}

```

Листинг 3.7 – Класс ProductStorage

В методах добавления/редактирования будем использовать транзакции, чтобы либо сохранялось все (и само изделие и связи с компонентами), либо был полный откат сохранения. При удалении удаляем только изделие. За счет создания связей в БД все записи в классе ProductComponent, связанные с этим изделием, удаляться автоматически. Также есть пара моментов, связанных с особенностями запросов к БД. В методе Update при обновлении связей компонентов изделий придется сначала вытащить список всех связей по изделию и только потом фильтровать их по включению в список моделей, пришедших от пользователя. Это связано с тем, что метод ContainsKey словаря не сработает при запросе к БД. Также, при чтении сначала придется выполнить запрос на получение данных (одной записи или всех) из БД (вызываем ToList после метода Where) и только потом преобразовывать через метод Select. Это связано с тем, что создание кортежей (используется в значении в словаре) невозможно при запросе к БД.

Потребуется также реализовать интерфейс IOrderStorage.

Последний штрих. В проект AbstractShopView добавить ссылку на AbstractShopDatabaseImplement. И в классе Program поменять всего одну строку. Вместо

```
using AbstractShopFileImplement.Implements;
```

Вставить

```
using AbstractShopDatabaseImplement.Implements;
```


В результате приложение будет работать с другой реализацией и хранить данные в файлах.

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. Название базы данных должно соответствовать логике вашего задания по варианту.
4. Создать модели для изделия и заказа с правильной расстановкой связей.
5. Реализовать интерфейс IOrderStorage для хранения данных в БД.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- сохранять в базу информацию по складам и их загруженность;
- сделать реализацию интерфейса логики хранения данных для сущности «Склад» с сохранением в базу данных (не должно быть 2-х складов с одним и тем же названием);
- в логике для хранения данных «Склад» метод проверки и списания компонент со складов в требуемом количестве реализовать с использованием транзакций (т.е., не должно быть проверки, а сразу идти списание с откаткой в случае нехватки компонент);

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
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).

ЛАБОРАТОРНАЯ РАБОТА №4.

РАБОТА С ОФИСНЫМИ ПАКЕТАМИ

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. Выводить в doc-файл список всех компонент. Выводить в таблицу и сохранять в excel-файл информацию о компонентах, с указанием в каких изделиях они используются. Выводить в форму отчетов и сохранять в pdf-файл список заказов за определенный период.
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Решение

Для сохранения в файлы и получения данных для отчета в проекте **AbstractShopBusinessLogic** создадим ряд классов. 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, для отчетов, не требующих даты, и для отчетов, которым необходимы даты, но можно и так. Так как вывод в doc-файл не требуется возвращать на форму какие-либо данные, то во ViewModels добавим 2 класса, первый будет возвращать данные для отчета по компонентам (листинг 4.2), второй – по заказам (листинг 4.3).

```
using System;

namespace AbstractShopBusinessLogic.BindingModels
{
    public class ReportBindingModel
    {
        public string FileName { get; set; }

        public DateTime? DateFrom { get; set; }

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

```
}
```

Листинг 4.1 – Класс ReportBindingModel

```
using System;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.ViewModels
{
    public class ReportProductComponentViewModel
    {
        public string ComponentName { get; set; }

        public int TotalCount { get; set; }

        public List<Tuple<string, int>> Products { get; set; }
    }
}
```

Листинг 4.2 – Класс ReportProductComponentViewModel

```
using AbstractShopBusinessLogic.Enums;
using System;

namespace AbstractShopBusinessLogic.ViewModels
{
    public class ReportOrdersViewModel
    {
        public DateTime DateCreate { get; set; }

        public string ProductName { get; set; }

        public int Count { get; set; }

        public decimal Sum { get; set; }

        public OrderStatus Status { get; set; }
    }
}
```

Листинг 4.3 – Класс ReportOrdersViewModel

Так как у нас уже есть классы OrderViewModel и ProductComponentViewModel, то для отчетов имя классов зададим иные, приписав им приставку Report. Создадим класс ReportLogic, где будет реализация методов получения данных для отчетов. Особенностью этого класса будет как раз то, что мы абстрагируемся от способа хранения данных, а просто пропишем логику для предоставления данных для отчетов. Также, для уменьшения объема кода в классе (а также принцип SOLID), логику выгрузки в различные форматы сделаем в отдельных классах.

Теперь нужно определиться с библиотеками, которые будем использовать для создания отчетов.

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.

Начнем с отчета в формате doc. На первом шаге требуется подключить библиотеку DocumentFormat.OpenXml через nuget-пакеты (также его нужно

будет подключить в исполняемом проекте AbstractShopView). Далее создадим класс для создания документа SaveToWord. Прежде, чем писать код создания документа, определим, как должен выглядеть документ. Для doc-файла можно вставлять абзацы с текстом, таблицы, рисунки и т.д. Остановимся на самом простом варианте – выведем простой текст. Будет заголовок и абзацы, в каждом из которых будет название одного из компонентов. Для создания отчета нам потребуется имя файла с путем, заголовок отчета и список компонент. Для этого создадим вспомогательный класс WordInfo (листинг 4.4). Также, создадим 2 класса, в одном будет передавать информацию для форматирования текста параграфа (листинг 4.5), в другом информацию по каждому параграфу (листинг 4.6). По параметрам параграфа будем хранить информацию по размеру шрифта, признак, если нужно жирным, и выравнивание текста.

```
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.HelperModels
{
    class WordInfo
    {
        public string FileName { get; set; }

        public string Title { get; set; }

        public List<ComponentViewModel> Components { get; set; }
    }
}
```

Листинг 4.4 – Класс WordInfo

```
using DocumentFormat.OpenXml.Wordprocessing;

namespace AbstractShopBusinessLogic.HelperModels
{
    class WordParagraphProperties
    {
        public string Size { get; set; }

        public bool Bold { get; set; }

        public JustificationValues JustificationValues { get; set; }
    }
}
```

Листинг 4.5 – Класс WordParagraphProperties

```
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.HelperModels
```



```

{
    class WordParagraph
    {
        public List<(string, WordTextProperties)> Texts { get; set; }

        public WordTextProperties TextProperties { get; set; }
    }
}

```

Листинг 4.6 – Класс WordParagraph

В классе SaveToWord выделим отдельные методы для создания параграфа, установки форматирования параграфа и настройки свойств страниц. Параграф может состоять из множества кусков текста, каждый со своим форматированием, но мы ограничимся только одним куском текста без отдельного форматирования (листинг 4.7).

```

using AbstractShopBusinessLogic.HelperModels;
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Wordprocessing;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    static class SaveToWord
    {
        /// <summary>
        /// Создание документа
        /// </summary>
        /// <param name="info"></param>
        public static void CreateDoc(WordInfo info)
        {
            using (WordprocessingDocument wordDocument =
                WordprocessingDocument.Create(info.FileName, WordprocessingDocumentType.Document))
            {
                MainDocumentPart mainPart = wordDocument.AddMainDocumentPart();
                mainPart.Document = new Document();
                Body docBody = mainPart.Document.AppendChild(new Body());

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

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

```

```

        Size = "24",
        JustificationValues = JustificationValues.Both
    }
    }));
}
docBody.AppendChild(CreateSectionProperties());

wordDocument.MainDocumentPart.Document.Save();
}

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

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

    properties.AppendChild(pageSize);

    return properties;
}

/// <summary>
/// Создание абзаца с текстом
/// </summary>
/// <param name="paragraph"></param>
/// <returns></returns>
private static Paragraph CreateParagraph(WordParagraph paragraph)
{
    if (paragraph != null)
    {
        Paragraph docParagraph = new Paragraph();

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

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

            RunProperties 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);
        }

        return docParagraph;
    }
}

```

```

        return null;
    }

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

            properties.AppendChild(new Justification()
            {
                Val = paragraphProperties.JustificationValues
            });

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

            properties.AppendChild(new Indentation());

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

            return properties;
        }

        return null;
    }
}

```

Листинг 4.7 – Класс SaveToWord

Отчет для Excel. Новых библиотек подключать не придется, так что перейдем сразу к созданию отчета. Отчет будет выглядеть следующим образом: заголовок, потом для каждого компонента отдельной строкой вывод его названия (будет первая колонка), в виде таблицы вывод изделий и требуемого количества компонент в нем (вторая и третья колонки) и также отдельной строкой вывод общего количества компонента на создания всех изделий (третья колонка). Для заголовка потребуется объединение ячеек (первые три в первой строке). Способ хранения данных в Excel имеет

специфику. В самих ячейках не хранится значение, а лишь порядковый номер на отдельную структуру `SharedStringTable`, в котором лежат все значения. Потому процедура вставки значения в Excel следующая:

- записать в `SharedStringTable` значение и запомнить ее позицию;
- создать ячейку, либо получить доступ к существующей;
- записать в значение ячейки сохраненный номер позиции значения.

Нам понадобится 3 вспомогательных класса: для передачи данных для отчета (листинг 4.8), для добавления значения в книгу (листинг 4.9) и для объединения ячеек (листинг 4.10).

```
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

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

        public string Title { get; set; }

        public List<ReportProductComponentViewModel> ProductComponents { get; set; }
    }
}
```

Листинг 4.8 – Класс `ExcelInfo`

```
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Spreadsheet;

namespace AbstractShopBusinessLogic.HelperModels
{
    class ExcelCellParameters
    {
        public Worksheet Worksheet { get; set; }

        public string ColumnName { get; set; }

        public uint RowIndex { get; set; }

        public UInt32Value StyleIndex { get; set; }

        public string Text { get; set; }

        public SharedStringTablePart ShareStringPart { get; set; }

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

Листинг 4.9 – Класс `ExcelCellParameters`

```

using DocumentFormat.OpenXml.Spreadsheet;

namespace AbstractShopBusinessLogic.HelperModels
{
    class ExcelMergeParameters
    {
        public Worksheet Worksheet { get; set; }

        public string CellFromName { get; set; }

        public string CellToName { get; set; }

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

```

Листинг 4.10 – Класс ExcelMergeParameters

Создадим класс SaveToExcel для создания отчета в формате Excel. Отдельно выделим методы записи значения в ячейку, объединение ячеек, а также метод создания стилей для книги (листинг 4.11).

```

using AbstractShopBusinessLogic.HelperModels;
using DocumentFormat.OpenXml;
using DocumentFormat.OpenXml.Office2010.Excel;
using DocumentFormat.OpenXml.Office2013.Excel;
using DocumentFormat.OpenXml.Packaging;
using DocumentFormat.OpenXml.Spreadsheet;
using System.Linq;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    static class SaveToExcel
    {
        public static void CreateDoc(ExcelInfo info)
        {
            using (SpreadsheetDocument spreadsheetDocument =
                SpreadsheetDocument.Create(info.FileName, SpreadsheetDocumentType.Workbook))
            {
                // Создаем книгу (в ней хранятся листы)
                WorkbookPart workbookpart = spreadsheetDocument.AddWorkbookPart();
                workbookpart.Workbook = new Workbook();

                CreateStyles(workbookpart);

                // Получаем/создаем хранилище текстов для книги
                SharedStringTablePart shareStringPart =
                    spreadsheetDocument.WorkbookPart.GetPartsOfType<SharedStringTablePart>().Count() > 0
                    ?
                    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
                WorksheetPart worksheetPart = workbookpart.AddNewPart<WorksheetPart>();
                worksheetPart.Worksheet = new Worksheet(new SheetData());
            }
        }
    }
}

```

```

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

        InsertCellInWorksheet( new ExcelCellParameters
        {
            Worksheet = worksheetPart.Worksheet,
            ShareStringPart = shareStringPart,
            ColumnName = "A",
            RowIndex = 1,
            Text = info.Title,
            StyleIndex = 2U
        });

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

        uint rowIndex = 2;
        foreach(var pc in info.ProductComponents)
        {
            InsertCellInWorksheet(new ExcelCellParameters
            {
                Worksheet = worksheetPart.Worksheet,
                ShareStringPart = shareStringPart,
                ColumnName = "A",
                RowIndex = rowIndex,
                Text = pc.ComponentName,
                StyleIndex = 0U
            });
            rowIndex++;

            foreach(var product in pc.Products)
            {
                InsertCellInWorksheet(new ExcelCellParameters
                {
                    Worksheet = worksheetPart.Worksheet,
                    ShareStringPart = shareStringPart,
                    ColumnName = "B",
                    RowIndex = rowIndex,
                    Text = product.Item1,
                    StyleIndex = 1U
                });

                InsertCellInWorksheet(new ExcelCellParameters
                {
                    Worksheet = worksheetPart.Worksheet,
                    ShareStringPart = shareStringPart,
                    ColumnName = "C",
                    RowIndex = rowIndex,
                    Text = product.Item2.ToString(),
                    StyleIndex = 1U
                });
            }
        }
    }
}

```

```

        rowIndex++;
    }

    InsertCellInWorksheet(new ExcelCellParameters
    {
        Worksheet = worksheetPart.Worksheet,
        ShareStringPart = shareStringPart,
        ColumnName = "C",
        RowIndex = rowIndex,
        Text = pc.TotalCount.ToString(),
        StyleIndex = 0U
    });
    rowIndex++;
}

workbookpart.Workbook.Save();
}
}

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

    Fonts fonts = new Fonts() { Count = (UInt32Value)2U, KnownFonts = true };

    Font fontUsual = new Font();
    fontUsual.Append(new FontSize() { Val = 12D });
    fontUsual.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Theme
= (UInt32Value)1U });
    fontUsual.Append(new FontName() { Val = "Times New Roman" });
    fontUsual.Append(new FontFamilyNumbering() { Val = 2 });
    fontUsual.Append(new FontScheme() { Val = FontSchemeValues.Minor });

    Font fontTitle = new Font();
    fontTitle.Append(new Bold());
    fontTitle.Append(new FontSize() { Val = 14D });
    fontTitle.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() { Theme
= (UInt32Value)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);

    Fills fills = new Fills() { Count = (UInt32Value)2U };

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

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

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

    Borders borders = new Borders() { Count = (UInt32Value)2U };

```

```

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

        Border borderThin = new Border();

        LeftBorder leftBorder = new LeftBorder() { Style = BorderStyleValues.Thin };
        leftBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() {
Indexed = (UInt32Value)64U });

        RightBorder rightBorder = new RightBorder() { Style = BorderStyleValues.Thin
};
        rightBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() {
Indexed = (UInt32Value)64U });

        TopBorder topBorder = new TopBorder() { Style = BorderStyleValues.Thin };
        topBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() {
Indexed = (UInt32Value)64U });

        BottomBorder bottomBorder = new BottomBorder() { Style =
BorderStyleValues.Thin };
        bottomBorder.Append(new DocumentFormat.OpenXml.Office2010.Excel.Color() {
Indexed = (UInt32Value)64U });

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

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

        CellStyleFormats cellStyleFormats = new CellStyleFormats() { Count =
(UInt32Value)1U };
        CellFormat cellFormatStyle = new CellFormat() { NumberFormatId =
(UInt32Value)0U, FontId = (UInt32Value)0U, FillId = (UInt32Value)0U, BorderId =
(UInt32Value)0U };

        cellStyleFormats.Append(cellFormatStyle);

        CellFormats cellFormats = new CellFormats() { Count = (UInt32Value)3U };
        CellFormat cellFormatFont = new CellFormat() { NumberFormatId =
(UInt32Value)0U, FontId = (UInt32Value)0U, FillId = (UInt32Value)0U, BorderId =
(UInt32Value)0U, FormatId = (UInt32Value)0U, ApplyFont = true };
        CellFormat cellFormatFontAndBorder = new CellFormat() { NumberFormatId =
(UInt32Value)0U, FontId = (UInt32Value)0U, FillId = (UInt32Value)0U, BorderId =
(UInt32Value)1U, FormatId = (UInt32Value)0U, ApplyFont = true, ApplyBorder = true };
        CellFormat cellFormatTitle = new CellFormat() { NumberFormatId =
(UInt32Value)0U, FontId = (UInt32Value)1U, FillId = (UInt32Value)0U, BorderId =
(UInt32Value)0U, FormatId = (UInt32Value)0U, Alignment = new Alignment() { Vertical =
VerticalAlignmentValues.Center, WrapText = true, Horizontal =
HorizontalAlignmentValues.Center }, ApplyFont = true };

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

        CellStyles cellStyles = new CellStyles() { Count = (UInt32Value)1U };

        cellStyles.Append(new CellStyle() { Name = "Normal", FormatId =

```



```

(UInt32Value)0U, BuiltinId = (UInt32Value)0U });

    DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats
differentialFormats = new DocumentFormat.OpenXml.Office2013.Excel.DifferentialFormats() {
Count = (UInt32Value)0U };

    TableStyles tableStyles = new TableStyles() { Count = (UInt32Value)0U,
DefaultTableStyle = "TableStyleMedium2", DefaultPivotStyle = "PivotStyleLight16" };

    StylesheetExtensionList stylesheetExtensionList = new
StylesheetExtensionList();

    StylesheetExtension 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" });

    StylesheetExtension 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>
/// Добавляем новую ячейку в лист
/// </summary>
/// <param name="cellParameters"></param>
private static void InsertCellInWorksheet(ExcelCellParameters cellParameters)
{
    SheetData sheetData = cellParameters.Worksheet.GetFirstChild<SheetData>();

    // Ищем строку, либо добавляем ее
    Row row;
    if (sheetData.Elements<Row>().Where(r => r.RowIndex ==
cellParameters.RowIndex).Count() != 0)
    {
        row = sheetData.Elements<Row>().Where(r => r.RowIndex ==
cellParameters.RowIndex).First();
    }
    else
    {
        row = new Row() { RowIndex = cellParameters.RowIndex };
        sheetData.Append(row);
    }

    // Ищем нужную ячейку
    Cell cell;

```

```

        if (row.Elements<Cell>().Where(c => c.CellReference.Value ==
cellParameters.CellReference).Count() > 0)
        {
            cell = row.Elements<Cell>().Where(c => c.CellReference.Value ==
cellParameters.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,
cellParameters.CellReference, true) > 0)
                {
                    refCell = rowCell;
                    break;
                }
            }

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

            cell = newCell;
        }

        // вставляем новый текст
        cellParameters.ShareStringPart.SharedStringTable.AppendChild(new
SharedStringItem(new Text(cellParameters.Text)));
        cellParameters.ShareStringPart.SharedStringTable.Save();

        cell.CellValue = new
CellValue((cellParameters.ShareStringPart.SharedStringTable.Elements<SharedStringItem>().
Count() - 1).ToString());
        cell.DataType = new EnumValue<CellValues>(CellValues.SharedString);
        cell.StyleIndex = cellParameters.StyleIndex;
    }

    /// <summary>
    /// Объединение ячеек
    /// </summary>
    /// <param name="mergeParameters"></param>
    private static void MergeCells(ExcelMergeParameters mergeParameters)
    {
        MergeCells mergeCells;

        if (mergeParameters.Worksheet.Elements<MergeCells>().Count() > 0)
        {
            mergeCells = mergeParameters.Worksheet.Elements<MergeCells>().First();
        }
        else
        {
            mergeCells = new MergeCells();

            if (mergeParameters.Worksheet.Elements<CustomSheetView>().Count() > 0)
            {
                mergeParameters.Worksheet.InsertAfter(mergeCells,
mergeParameters.Worksheet.Elements<CustomSheetView>().First());
            }
            else
            {
                mergeParameters.Worksheet.InsertAfter(mergeCells,

```

```

mergeParameters.Worksheet.Elements<SheetData>().First();
    }
}

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

```

Листинг 4.11 – Класс SaveToExcel

Перейдем к сохранению данных в формате pdf. Через NuGet-пакеты подключим пакет «PDFsharp-MigraDoc» в проекты AbstractShopBusinessLogic и AbstractShopView. Отчет будет выглядеть следующим образом: вверху заголовок с датами периода, за который выбираем заказы и таблица с заказами, попавшими в этот период. Нам понадобится 3 дополнительных класса: для передачи данных для отчета (листинг 4.12), для передачи информации по добавляемой строке (список строк для заполнения ячеек, каким стилем оформлять, выравнивание и т.д.) (листинг 4.13) и ячейки строки (текст, стиль, выравнивание и т.п.) (листинг 4.14).

```

using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;

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

        public string Title { get; set; }

        public DateTime DateFrom { get; set; }

        public DateTime DateTo { get; set; }

        public List<ReportOrdersViewModel> Orders { get; set; }
    }
}

```

Листинг 4.12 – Класс PdfInfo

```

using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Tables;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.HelperModels
{

```

```

class PdfRowParameters
{
    public Table Table { get; set; }

    public List<string> Texts { get; set; }

    public string Style { get; set; }

    public ParagraphAlignment ParagraphAlignment { get; set; }
}

```

Листинг 4.13 – Класс PdfRowParameters

```

using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Tables;

namespace AbstractShopBusinessLogic.HelperModels
{
    class PdfCellParameters
    {
        public Cell Cell { get; set; }

        public string Text { get; set; }

        public string Style { get; set; }

        public ParagraphAlignment ParagraphAlignment { get; set; }

        public Unit BorderWidth { get; set; }
    }
}

```

Листинг 4.14 – Класс PdfCellParameters

Создадим класс SaveToPdf для создания отчета. Выделим отдельно методы для создания отчета, создания строки для таблицы отчета и метода заполнения ячейки отчета (листинг 4.15).

```

using AbstractShopBusinessLogic.HelperModels;
using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Tables;
using MigraDoc.Rendering;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    class SaveToPdf
    {
        public static void CreateDoc(PdfInfo info)
        {
            Document document = new Document();
            DefineStyles(document);

            Section section = document.AddSection();
            Paragraph paragraph = section.AddParagraph(info.Title);
            paragraph.Format.SpaceAfter = "1cm";
            paragraph.Format.Alignment = ParagraphAlignment.Center;
            paragraph.Style = "NormalTitle";

            paragraph = section.AddParagraph($"c {info.DateFrom.ToShortDateString()} по
{info.DateTo.ToShortDateString()}");

```

```

        paragraph.Format.SpaceAfter = "1cm";
        paragraph.Format.Alignment = ParagraphAlignment.Center;
        paragraph.Style = "Normal";

        var table = document.LastSection.AddTable();

        List<string> columns = new List<string> { "3cm", "6cm", "3cm", "2cm", "3cm"
};
        foreach (var elem in columns)
        {
            table.AddColumn(elem);
        }

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

        foreach(var order in info.Orders)
        {
            CreateRow(new PdfRowParameters
            {
                Table = table,
                Texts = new List<string> { order.DateCreate.ToShortDateString(),
order.ProductName, order.Count.ToString(), order.Sum.ToString(), order.Status.ToString()
},
                Style = "Normal",
                ParagraphAlignment = ParagraphAlignment.Left
            });
        }

        PdfDocumentRenderer renderer = new PdfDocumentRenderer(true,
PdfSharp.Pdf.PdfFontEmbedding.Always)
        {
            Document = document
        };
        renderer.RenderDocument();
        renderer.PdfDocument.Save(info.FileName);
    }

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

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

    /// <summary>
    /// Создание и заполнение строки
    /// </summary>
    /// <param name="rowParameters"></param>
    private static void CreateRow(PdfRowParameters rowParameters)
    {

```

```

        Row row = rowParameters.Table.AddRow();
        for (int i = 0; i < rowParameters.Texts.Count; ++i)
        {
            FillCell(new PdfCellParameters
            {
                Cell = row.Cells[i],
                Text = rowParameters.Texts[i],
                Style = rowParameters.Style,
                BorderWidth = 0.5,
                ParagraphAlignment = rowParameters.ParagraphAlignment
            });
        }
    }

    /// <summary>
    /// Заполнение ячеек
    /// </summary>
    /// <param name="cellParameters"></param>
    private static void FillCell(PdfCellParameters cellParameters)
    {
        cellParameters.Cell.AddParagraph(cellParameters.Text);

        if (!string.IsNullOrEmpty(cellParameters.Style))
        {
            cellParameters.Cell.Style = cellParameters.Style;
        }

        cellParameters.Cell.Borders.Left.Width = cellParameters.BorderWidth;
        cellParameters.Cell.Borders.Right.Width = cellParameters.BorderWidth;
        cellParameters.Cell.Borders.Top.Width = cellParameters.BorderWidth;
        cellParameters.Cell.Borders.Bottom.Width = cellParameters.BorderWidth;

        cellParameters.Cell.Format.Alignment = cellParameters.ParagraphAlignment;
        cellParameters.Cell.VerticalAlignment = VerticalAlignment.Center;
    }
}

```

Листинг 4.15 – Класс SaveToPdf

Остается сделать только класс ReportLogic. Нам потребуется подключить 3 интерфейса для получения списка компонент, изделий и заказов. Так как вывод компонент по изделиям и заказов потребуется выводить на формы соответствующие списки, то сделаем 2 отдельных метода для получения этих списков и 3 метода для сохранения отчетов в word, excel и pdf. Так как заказы следует выбирать за период, класс OrderBindingModel потребуется расширить 2 параметрами (листинг 4.16) и поменять логику в реализациях получения списка заказов.

```

using AbstractShopBusinessLogic.Enums;
using System;

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Заказ
    /// </summary>
    public class OrderBindingModel
    {
        public int? Id { get; set; }

        public int ProductId { get; set; }

        public int Count { get; set; }

        public decimal Sum { get; set; }

        public OrderStatus Status { get; set; }

        public DateTime DateCreate { get; set; }

        public DateTime? DateImplement { get; set; }

        public DateTime? DateFrom { get; set; }

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

```

Листинг 4.16 – Дополненный класс OrderBindingModel

Класс ReportLogic будет следующим (листинг 4.17)

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.HelperModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;
using System.Linq;

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

        private readonly IProductStorage _productStorage;

        private readonly IOrderStorage _orderStorage;

        public ReportLogic(IProductStorage productStorage, IComponentStorage componentStorage, IOrderStorage orderStorage)
        {
            _productStorage = productStorage;
            _componentStorage = componentStorage;
            _orderStorage = orderStorage;
        }

        /// <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 OrderBindingModel { DateFrom =
model.DateFrom, DateTo = model.DateTo })
        .Select(x => new ReportOrdersViewModel
        {
            DateCreate = x.DateCreate,
            ProductName = x.ProductName,
            Count = x.Count,
            Sum = x.Sum,
            Status = x.Status
        })
        .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 указанием продуктов в файл-Excel
/// </summary>
/// <param name="model"></param>
public void SaveProductComponentToExcelFile(ReportBindingModel model)
{
    SaveToExcel.CreateDoc(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 4.17 – Класс ReportLogic

Переходим к проекту AbstractShopView. В классе Program в UnityContainer прописываем новый класс (чтобы ему подставлялись реализации интерфейсов) (листинг 4.18).

```

using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopDatabaseImplement.Implements;
using System;
using System.Windows.Forms;
using Unity;
using Unity.Lifetime;

namespace AbstractShopView
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            var container = BuildUnityContainer();

            Application.EnableVisualStyles();

```

```

        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(container.Resolve<FormMain>());
    }

    private static IUnityContainer BuildUnityContainer()
    {
        var currentContainer = new UnityContainer();
        currentContainer.RegisterType<IComponentStorage, ComponentStorage>(new
        HierarchicalLifetimeManager());
        currentContainer.RegisterType<IOrderStorage, OrderStorage>(new
        HierarchicalLifetimeManager());
        currentContainer.RegisterType<IProductStorage, ProductStorage>(new
        HierarchicalLifetimeManager());

        currentContainer.RegisterType<ComponentLogic>(new
        HierarchicalLifetimeManager());
        currentContainer.RegisterType<OrderLogic>(new HierarchicalLifetimeManager());
        currentContainer.RegisterType<ProductLogic>(new
        HierarchicalLifetimeManager());
        currentContainer.RegisterType<ReportLogic>(new
        HierarchicalLifetimeManager());

        return currentContainer;
    }
}

```

Листинг 4.18 – Класс Program с добавленным подключением класса
ReportLogic

На основной форме делаем пункт меню «Отчеты» и в нем 3 подпункта: «Список компонентов», «Компоненты по изделиям» и «Список заказов» (рисунок 4.1).

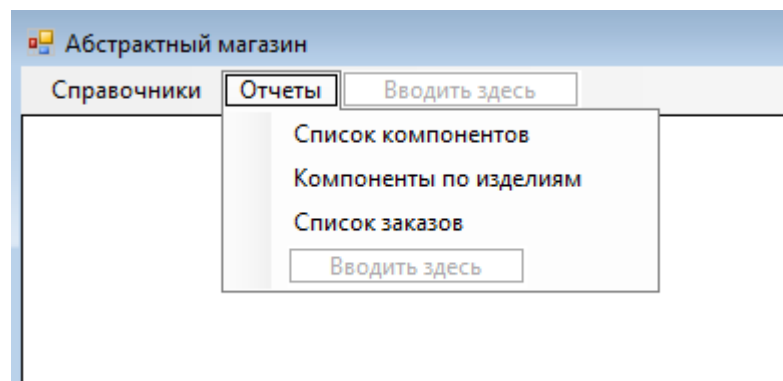


Рисунок 4.1 – Новые пункты меню на форме

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 4.2).

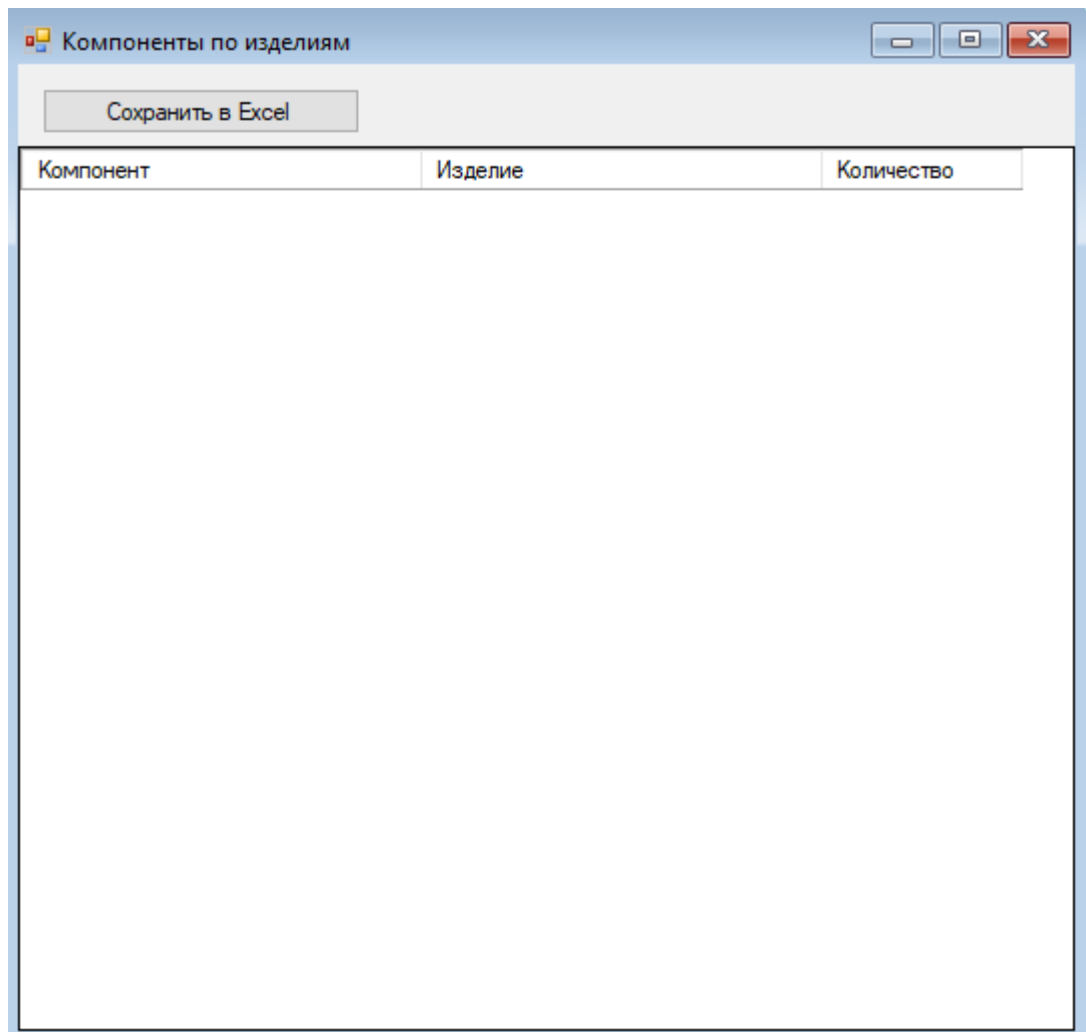


Рисунок 4.2 – Добавление колонок

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 4.19).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using System;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormReportProductComponents : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        private readonly ReportLogic logic;

        public FormReportProductComponents(ReportLogic logic)
        {
            InitializeComponent();
        }
    }
}
```

```

        this.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(new object[] { });
                }
            }
        }
        catch (Exception ex)
        {
            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
                    });
                    MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }
            }
        }
    }
}

```

Листинг 4.19 – Логика формы FormReportProductComponents

Отчет по заказам будем выводить через форму отчетов. Для этого в проект добавляем следующие пакеты: Microsoft.Report.Viewer и

Microsoft.ReportingServices.ReportViewerControl.Winforms (этот пакет подтянет за собой еще один).

Эти 2 пакета нужны для создания отчетов через шаблоны отчетов.

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). Файл создается через добавление нового элемента (рисунок 4.3).

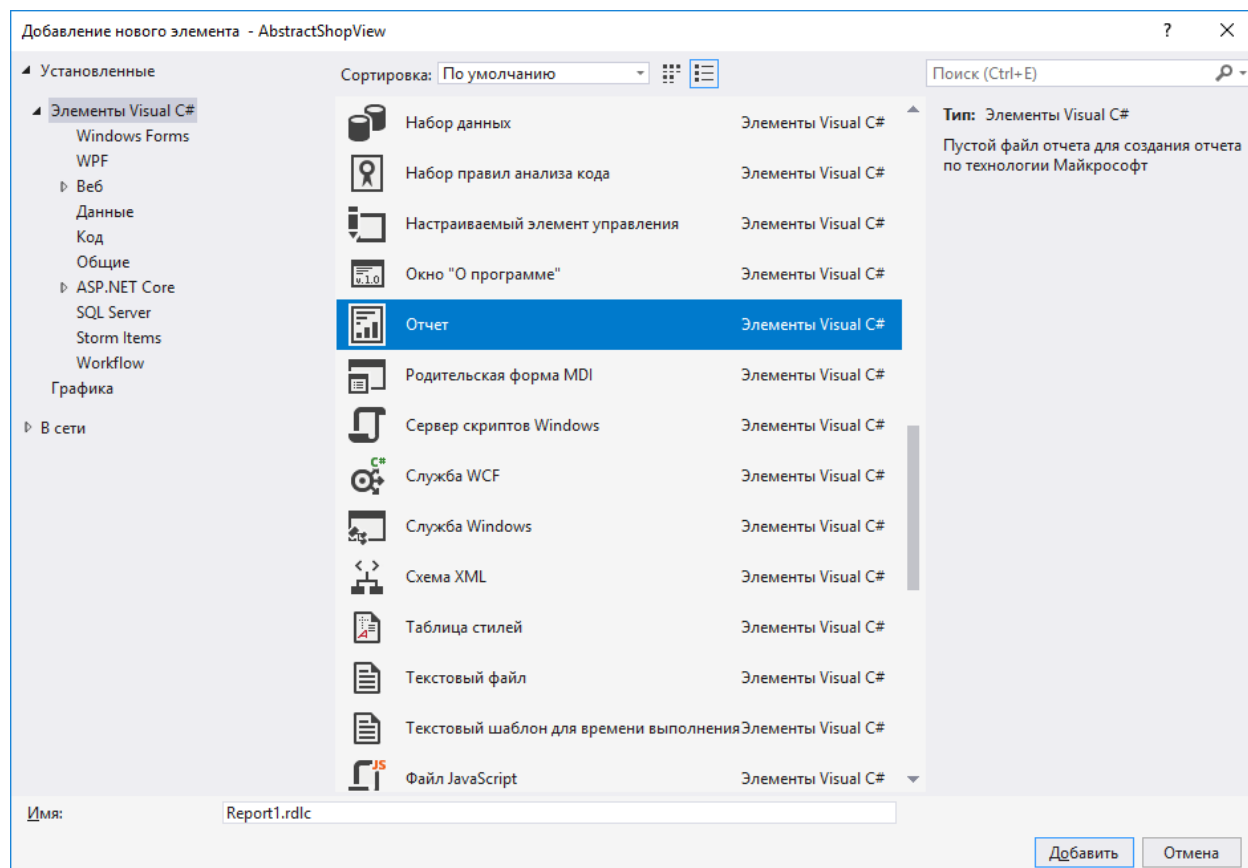


Рисунок 4.3 – Добавление нового элемента – отчета

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 (рисунок 4.4).

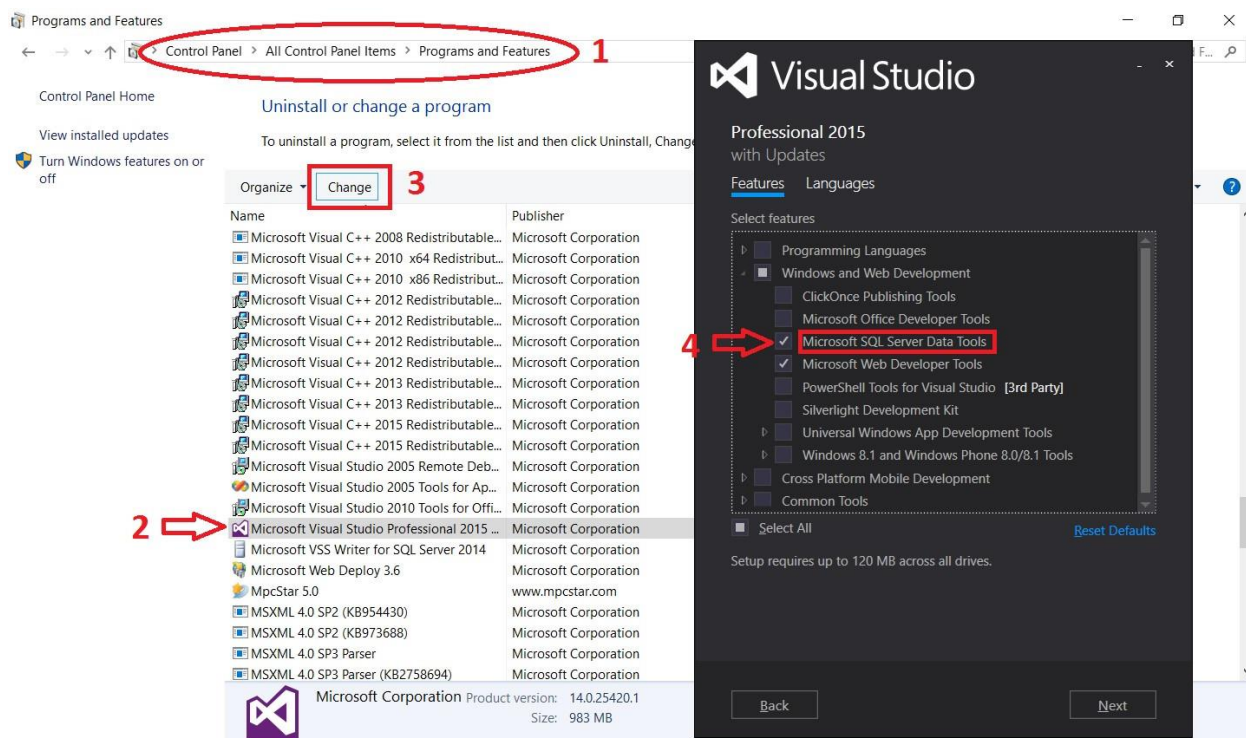


Рисунок 4.4 – Установка Microsoft SQL Server Data Tools

2. Скачать и установить 2 компонента:
 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 4.5).

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 4.6).

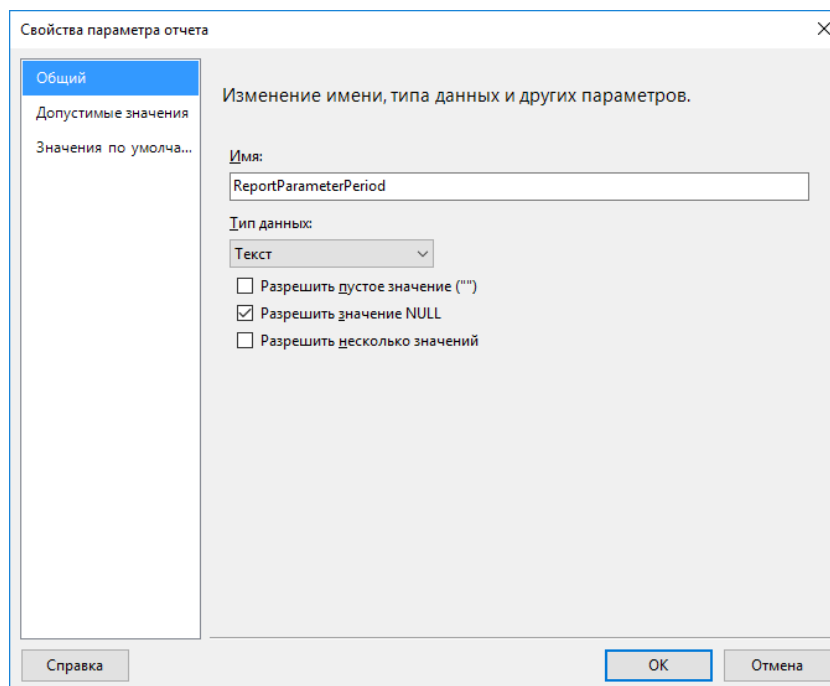


Рисунок 4.5 – Добавление параметра в отчет

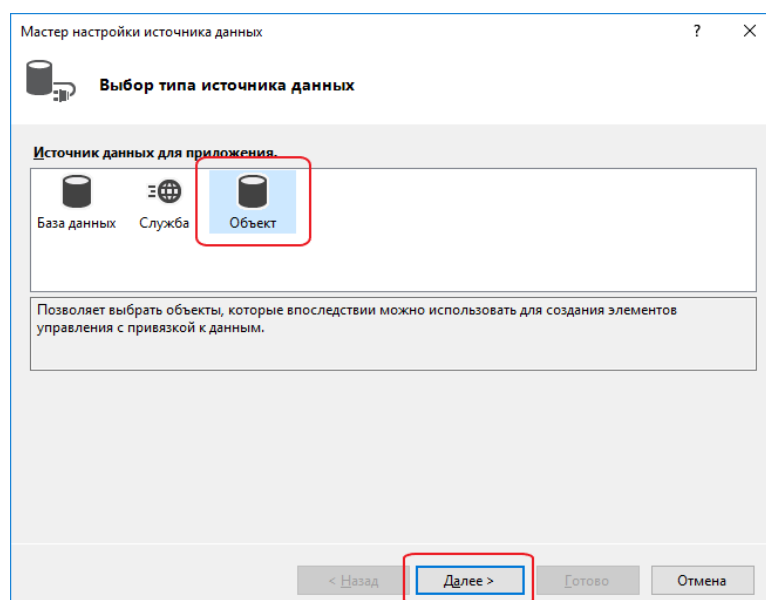


Рисунок 4.6 – Выбор типа источника данных

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 4.7).

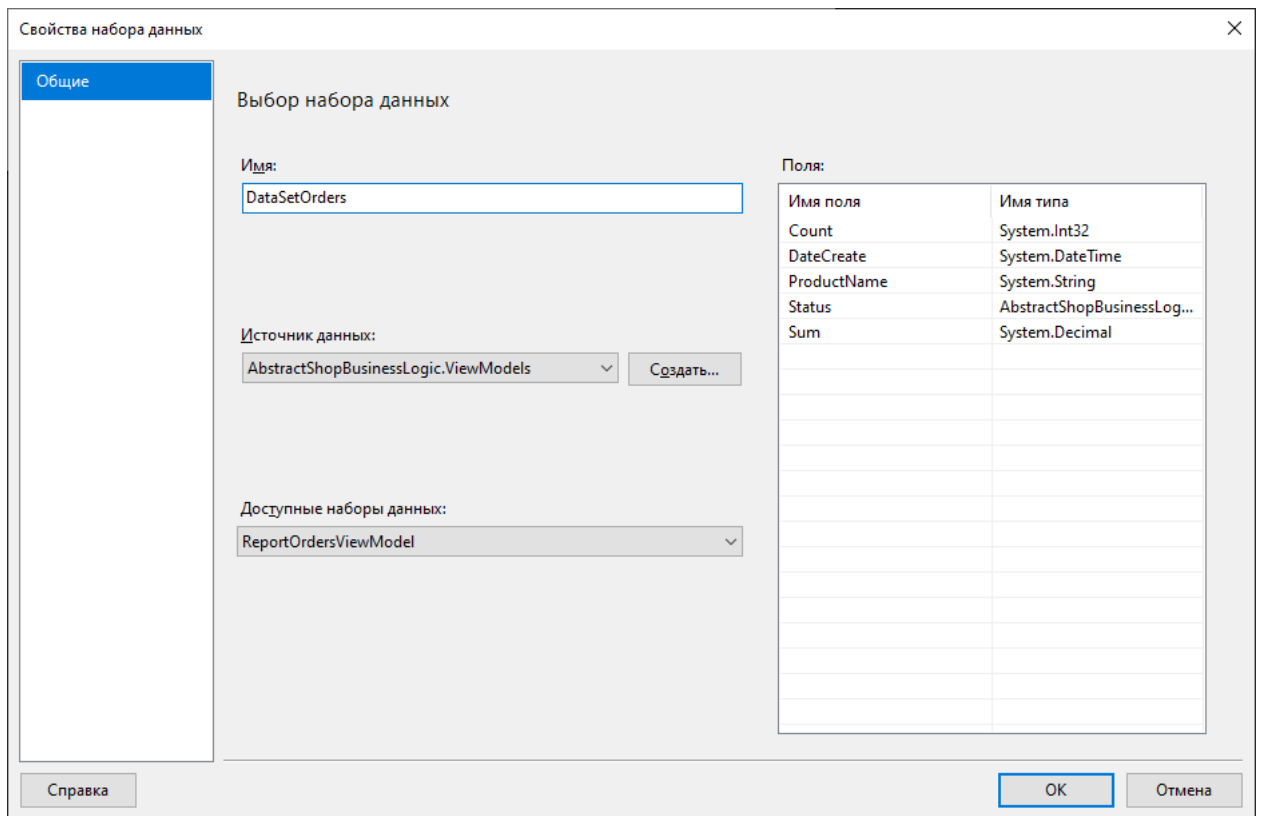


Рисунок 4.7 – Создание набора данных

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 4.8).

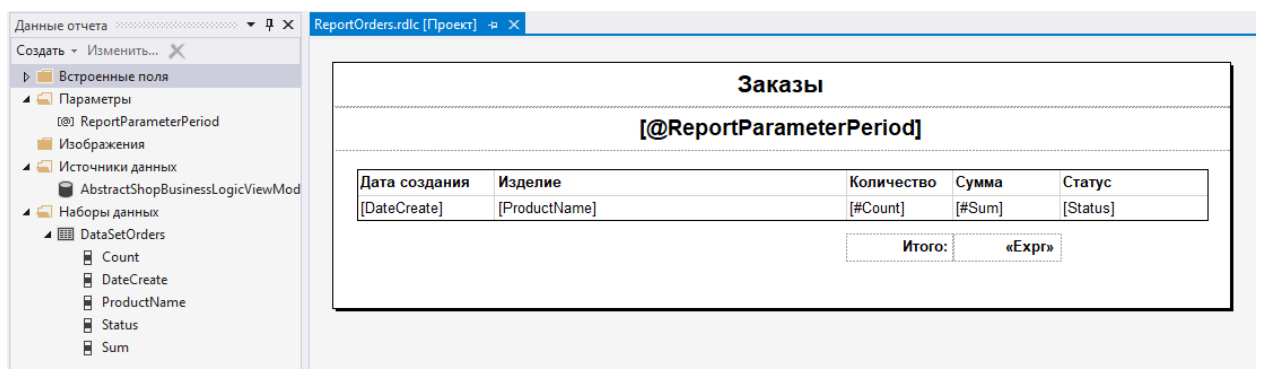


Рисунок 4.8 – Макет отчета

Создадим форму. Сверху расположим панель с двумя datePicker и парой кнопок (вывести отчет и сохранить в pdf). Остальное пространство

займет reportViewer. У reportViewer указываем какой отчет подгружать (в правом верхнем углу элемента нажимаем стрелку и выбираем из списка наш отчет) (рисунок 4.9).

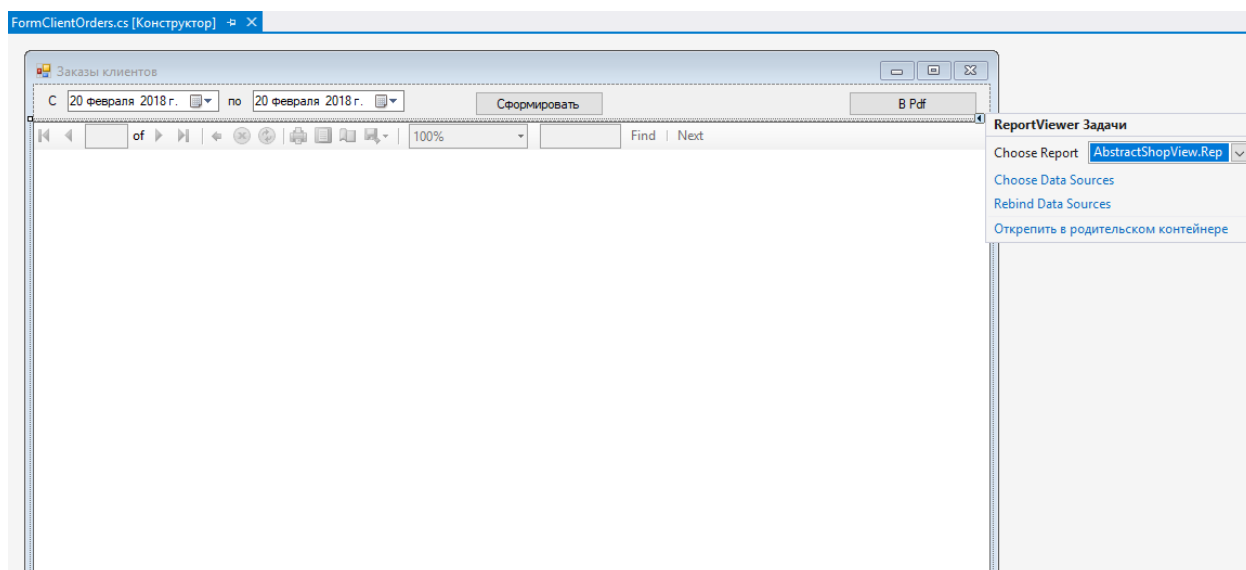


Рисунок 4.9 – Форма отчета по заказам с выбором отчета для reportViewer

Если на панели элементов (ToolBox) при работе с формой нет элемента ReportViewer, нужно сделать следующие шаги.

Щелкнуть правой кнопкой мыши на панели элементов (ToolBox) и выбрать «Выбрать элементы...» («Choose Items...»).

На вкладке «Компоненты .NET Framework» поискать элемент «ReportViewer» для WinForms. Если такого элемента нет или при попытке добавить его на форму возникают ошибки, то нажать «Обзор». В корне проекта найти папку packages, в ней – Microsoft.ReportingServices.ReportViewerControl.Winforms.xxx.xxxx.xxx, в ней папку lib/net40 и там выбрать dll Microsoft.ReportViewer.WinForms.dll.

Подгрузится dll и в списке появится новый элемент «ReportViewer». Ставим напротив него галочку, и он теперь будет отображаться на панели элементов (рисунок 4.10).

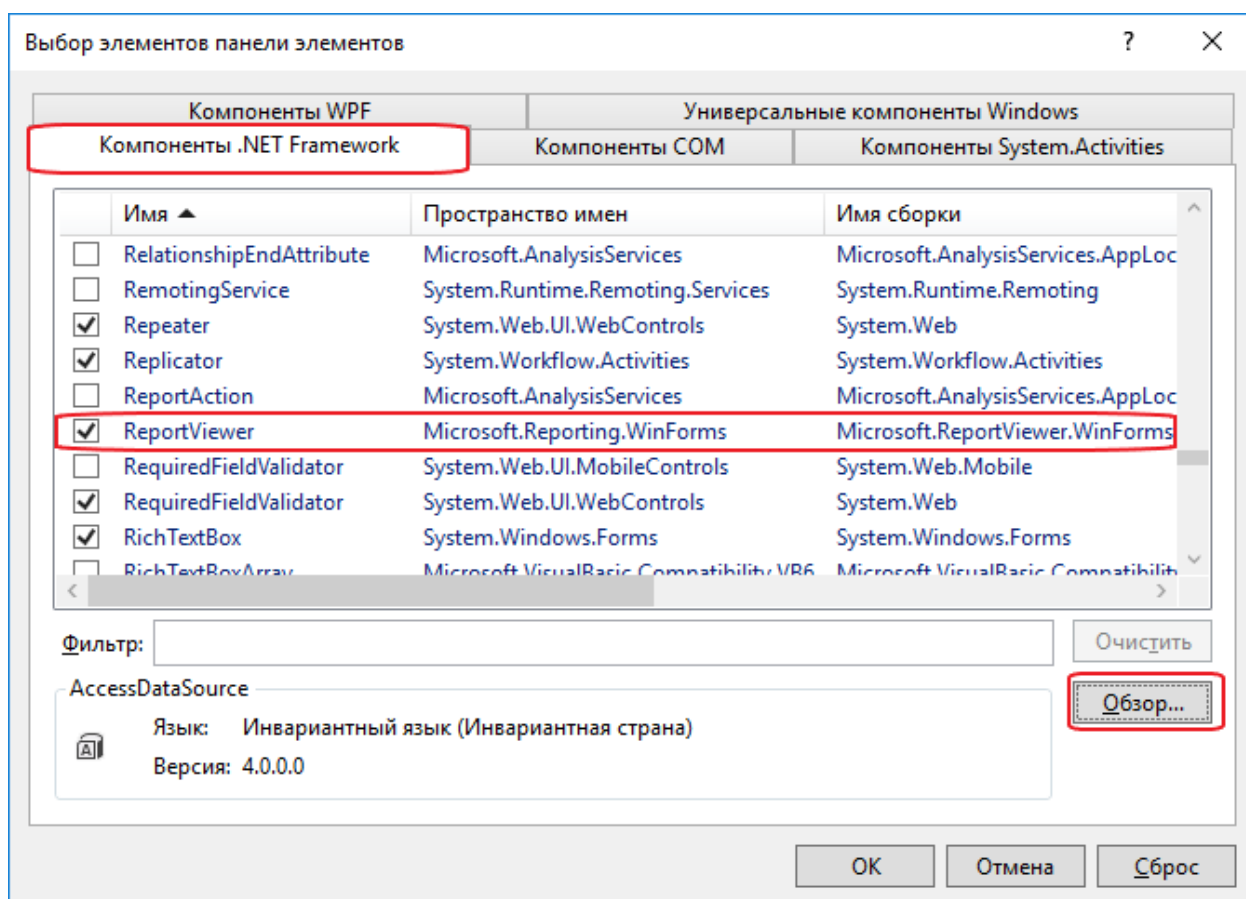


Рисунок 4.10 – Подключенные пакеты в проекте AbstractShopView

Теперь в форме добавим обработку нажатия кнопок. 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 4.20).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using Microsoft.Reporting.WinForms;
using System;
using System.Windows.Forms;
using Unity;

namespace AbstractShopView
{
    public partial class FormReportOrders : Form
    {
        [Dependency]
        public new IUnityContainer Container { get; set; }

        private readonly ReportLogic logic;

        public FormReportOrders(ReportLogic logic)
        {
            InitializeComponent();
        }
    }
}
```

```

        this.logic = logic;
    }

    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
        {
            ReportParameter parameter = new ReportParameter("ReportParameterPeriod",
"с " +
dateTimePickerFrom.Value.ToShortDateString() +
" по " +
dateTimePickerTo.Value.ToShortDateString());
            reportViewer.LocalReport.SetParameters(parameter);

            var dataSource = logic.GetOrders(new ReportBindingModel
            {
                DateFrom = dateTimePickerFrom.Value,
                DateTo = dateTimePickerTo.Value
            });
            ReportDataSource source = new ReportDataSource("DataSetOrders",
dataSource);
            reportViewer.LocalReport.DataSources.Add(source);

            reportViewer.RefreshReport();
        }
        catch (Exception ex)
        {
            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
                    });
                    MessageBox.Show("Выполнено", "Успех", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,

```

```
MessageBoxIcon.Error);
    }
}
}
}
}
```

Листинг 4.20 – Логика формы FormReportOrders

В логике главной формы прописываем вызовы форм для отчетов (листинг 4.21).

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

private void ComponentProductsToolStripMenuItem_Click(object sender, EventArgs e)
{
    var form = Container.Resolve<FormReportProductComponents>();
    form.ShowDialog();
}

private void OrdersToolStripMenuItem_Click(object sender, EventArgs e)
{
    var form = Container.Resolve<FormReportOrders>();
    form.ShowDialog();
}
```

Листинг 4.21 – Фрагмент логики формы FormMain с новыми методами

Результат вызова пункта «Список компонентов». Результат работы метода представлен на рисунке 4.11.

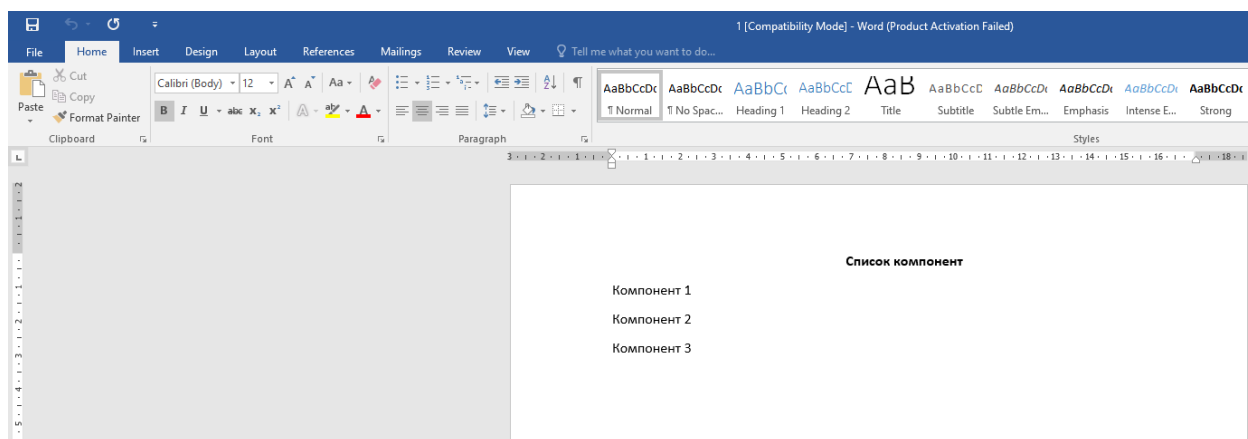


Рисунок 4.11 – Отчет по перечню компонент

Форма отчетов по компонентам в разрезе изделий представлена на рисунке 4.12.

Компоненты по изделиям

Сохранить в Excel

Компонент	Изделие	Количество
Компонент 1		
	Изделие 1	2
Итого		2
Компонент 2		
Итого		0
Компонент 3		
Итого		0

Рисунок 4.12 – Форма отчета по компонентам

Результат сохранения в Excel представлен на рисунке 4.13.

	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 4.13 – Отчет по складам

Форма отчетов по заказам за период представлена на рисунке 4.14.

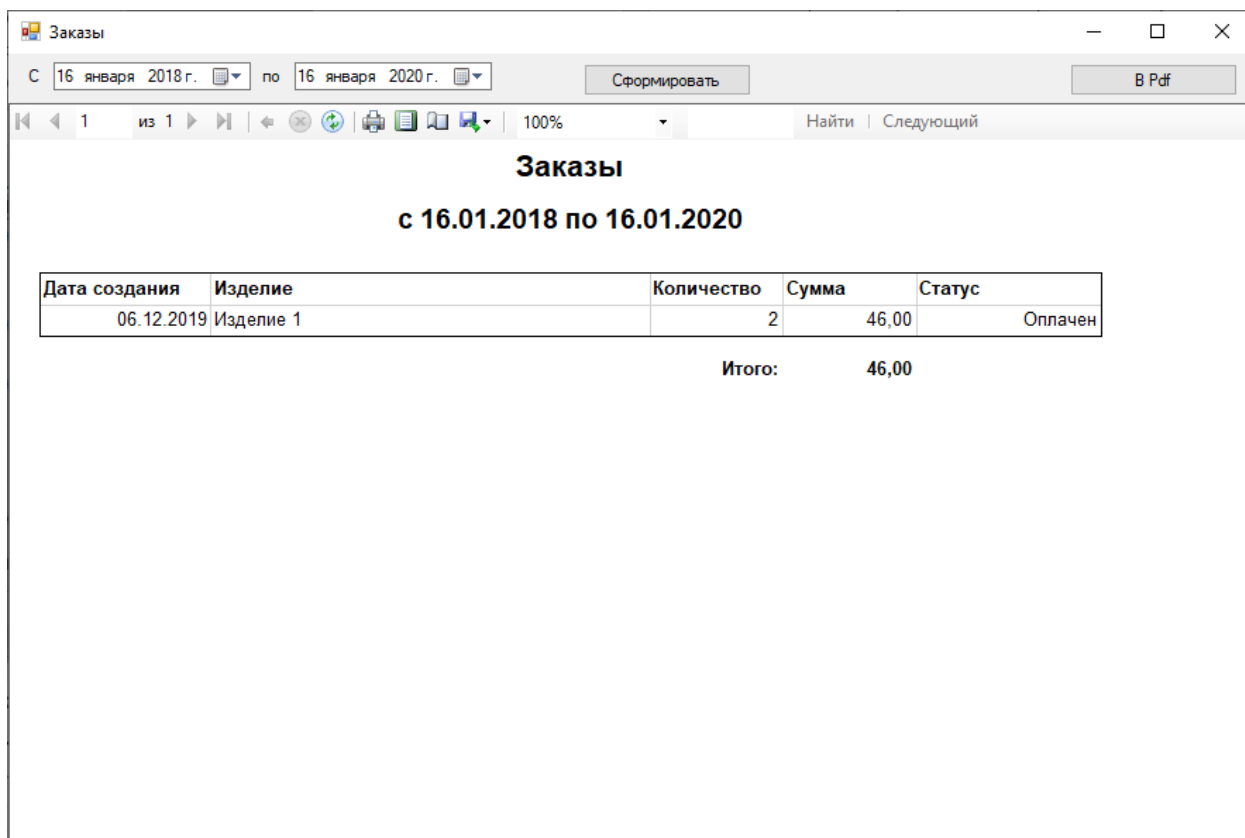


Рисунок 4.14 – Форма отчета по заказам за период
Результат сохранения в Pdf представлен на рисунке 4.15.

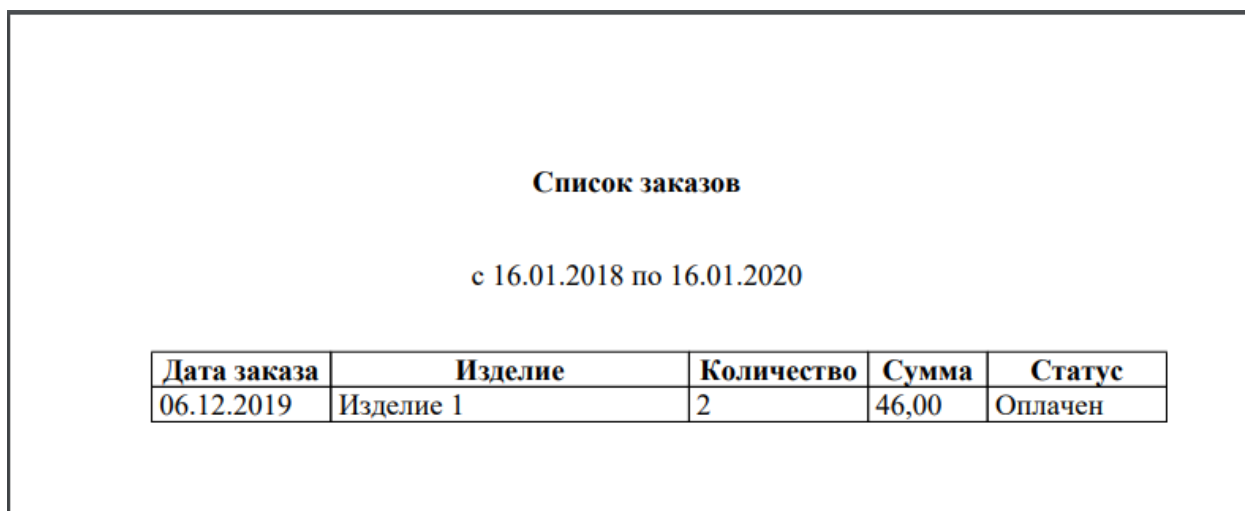


Рисунок 4.15 – Отчет по заказам за период

Требования

1. Название классов, полей, методов и свойств должно соответствовать логике вашего задания по варианту.
2. Для всех 3 реализаций логики хранения данных по сущности «Заказ» изменить логику получения списка фильтрованных заказов, чтобы можно было получать заказы за определенный период.

3. В doc-файл выводить список изделий с выделением названия жирным, а цену обычным шрифтом.
4. 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.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- выводить в doc-файл список складов в табличном виде (3 колонки: название, фио ответственного и дата создания);
- выводить в excel-файл и на форму через dataGridView информацию по загруженности складов с указанием какие компоненты и в каком количестве лежат на складе и общее количество компонент на складе;
- выводить pdf-файл и на форму через reportViewer информацию о заказах (за весь период), сгруппированных по датам (3 колонки: дата, кол-во заказов на эту дату, сумма по заказам).

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
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).

ЛАБОРАТОРНАЯ РАБОТА №5. КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ

Цель

На основе существующего приложения, создать клиент-серверное.

Задание

1. Создать ветку от ветки четвертой лабораторной.
2. *Добавить сущность «Клиент». У клиента должны храниться данные: ФИО, логин (он же электронная почта) и пароль. В заказе фиксировать какой клиент создал заказ. Создать приложение с RestAPI, которое будет предоставлять функционал для регистрации клиентов и создания ими заказов. Разработать web-приложение под клиентов, использующее приложение RestAPI для создания заказов.*
3. *Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.*

Решение

Клиент-сервер – вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между серверами и клиентами. Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, работа с базами данных).

REST (Representational state transfer) – это стиль архитектуры программного обеспечения для распределенных систем, таких как World Wide Web, который, как правило, используется для построения веб-служб. Термин REST был введен в 2000 году Роем Филдингом, одним из авторов HTTP-протокола. Системы, поддерживающие REST, называются RESTful-системами.

В общем случае REST является очень простым интерфейсом управления информацией без использования каких-то дополнительных внутренних прослоек. Каждая единица информации однозначно определяется глобальным идентификатором, таким как URL. Каждая URL в свою очередь имеет строго заданный формат.

Каждая единица информации однозначно определяется URL – это значит, что URL по сути является первичным ключом для единицы данных. Т.е., например, третья книга с книжной полки будет иметь вид /book/3, а 35 страница в этой книге – /book/3/page/35. Отсюда и получается строго заданный формат.

Как происходит управление информацией сервиса – это целиком и полностью основывается на протоколе передачи данных. Наиболее распространенный протокол конечно же HTTP. Так вот, для HTTP действие над данными задается с помощью методов: GET (получить), PUT (добавить, заменить), POST (добавить, изменить, удалить), DELETE (удалить). Таким образом, действия CRUD (Create-Read-Update-Delete) могут выполняться как со всеми 4-мя методами, так и только с помощью GET и POST.

Вот как это будет выглядеть на примере:

- GET /book/ — получить список всех книг
- GET /book/3/ — получить книгу номер 3
- PUT /book/ — добавить книгу (данные в теле запроса)
- POST /book/3 – изменить книгу (данные в теле запроса)
- DELETE /book/3 – удалить книгу

Для начала определим, какие данные нам потребуется передавать. Во-первых, данные по клиенту (он может регистрироваться, изменять свои данные). Во-вторых, данные по изделиям (просмотр списка). И в-третьих, создание заказа (просмотр своих заказов, создание заказов).

Для передачи данных нужно добавить атрибуты для Json-сериализации в классы BindingModel и ViewModel. Это будут классы ClientBindingModel, CreateOrderBindingModel, ClientViewModel, OrderViewModel и ProductViewModel. Для использования атрибутов требуется добавить ссылку на сборку System.Runtime.Serialization. В классы BindingModel и ViewModel для заказов необходимо включить информацию по клиентам (листинги 5.1-5.2).

```
using System.Runtime.Serialization;

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Данные от клиента, для создания заказа
    /// </summary>
    [DataContract]
    public class CreateOrderBindingModel
    {
        [DataMember]
        public int ClientId { get; set; }

        [DataMember]
        public int ProductId { get; set; }

        [DataMember]
        public int Count { get; set; }

        [DataMember]
        public decimal Sum { get; set; }
    }
}
```

Листинг 5.1 – Класс CreateOrderBindingModel с атрибутами сериализации

```
using AbstractShopBusinessLogic.Enums;
using System;
using System.ComponentModel;
using System.Runtime.Serialization;

namespace AbstractShopBusinessLogic.ViewModels
{
    /// <summary>
    /// Заказ
    /// </summary>
    [DataContract]
    public class OrderViewModel
    {
        [DataMember]
        public int Id { get; set; }

        [DataMember]
        public int ClientId { get; set; }

        [DataMember]
        public int ProductId { get; set; }

        [DataMember]
    }
```

```

        [DisplayName("Клиент")]
        public string ClientFIO { get; set; }

        [DataMember]
        [DisplayName("Изделие")]
        public string ProductName { get; set; }

        [DataMember]
        [DisplayName("Количество")]
        public int Count { get; set; }

        [DataMember]
        [DisplayName("Сумма")]
        public decimal Sum { get; set; }

        [DataMember]
        [DisplayName("Статус")]
        public OrderStatus Status { get; set; }

        [DataMember]
        [DisplayName("Дата создания")]
        public DateTime DateCreate { get; set; }

        [DataMember]
        [DisplayName("Дата выполнения")]
        public DateTime? DateImplement { get; set; }
    }
}

```

Листинг 5.2 – Класс OrderViewModel с атрибутами сериализации

Потребуется создать 3 реализации интерфейса клиента для списка, файлов и БД. Для всех реализаций потребуется сделать ряд однотипных шагов:

1. Добавить модель для клиента (для БД задать связь с моделью-заказом).
2. В классе-модели «Заказ» добавить поле – идентификатор клиента (для БД задать связь с моделью-клиент).
3. В классе-singleton (для БД context) добавить список от класса-модели «Клиент».
4. В логике сохранения заказа запоминать информацию о клиенте, создающим заказ.
5. В логике получения списка заказов заполнять идентификатор и ФИО клиента.

В реализации для файлов сделать загрузку и выгрузку списка клиентов, а также новое поле в списке заказов учесть.

В реализации клиента для БД потребуется:

1. Создать миграцию.
2. Обновить БД.

Самый сложный шаг – обновление базы данных, так как тут могут возникнуть сложности. В таблицу «Заказы» потребуется вставить новое поле. Если в таблице нет записей, то все пройдет нормально. Но, если записи есть, то возникнет конфликт, так как в каждую запись потребуется вставить новое поле, связанное с другой таблицей («Клиенты»), у которой нет записей еще. Тут существует несколько вариантов решения:

- Прописать значение по умолчанию и прописать код вставки клиента по умолчанию после создания таблицы «Клиенты» и до создания поля в таблице «Заказы», чтобы при установке значения по умолчанию не нарушалась связанность между таблицами.
- В классе «Заказ» указать, что поле идентификатора клиента может принимать значение NULL. В таком случае, в существующих записях заказов проставится значение «NULL».
- Удалить все записи из таблицы «Заказы».

Так как у нас проект находится на стадии разработки, то проще будет отчистить таблицу. После этого, можно обновлять схему БД.

Изменим реализации интерфейса IOrderStorage для получения выборки из списка заказов. Рассмотрим на примере реализации для файлов как изменится запрос (листинг 5.3).

```
public List<OrderViewModel> GetFilteredList(OrderBindingModel model)
{
    if (model == null)
    {
        return null;
    }

    return source.Orders
        .Where(rec => (!model.DateFrom.HasValue && !model.DateTo.HasValue &&
            rec.DateCreate.Date == model.DateCreate.Date) ||
            (model.DateFrom.HasValue && model.DateTo.HasValue && rec.DateCreate.Date
                >= model.DateFrom.Value.Date && rec.DateCreate.Date <= model.DateTo.Value.Date) ||
            (model.ClientId.HasValue && rec.ClientId == model.ClientId))
        .Select(CreateModel)
        .ToList();
}
```

```
}
```

Листинг 5.3 – Обновленный метод GetFilteredList класса OrderStorage

Логика будет следующей: если нет модели, то вернется null, если указаны даты, то будут выбираться заказы за определенный период, если указан клиент, то будут выбираться заказы, созданные определенным клиентом.

Перейдем к проекту RESTAPI. Добавим новый проект **AbstractShopRestApi** (рисунки 5.1, 5.2).

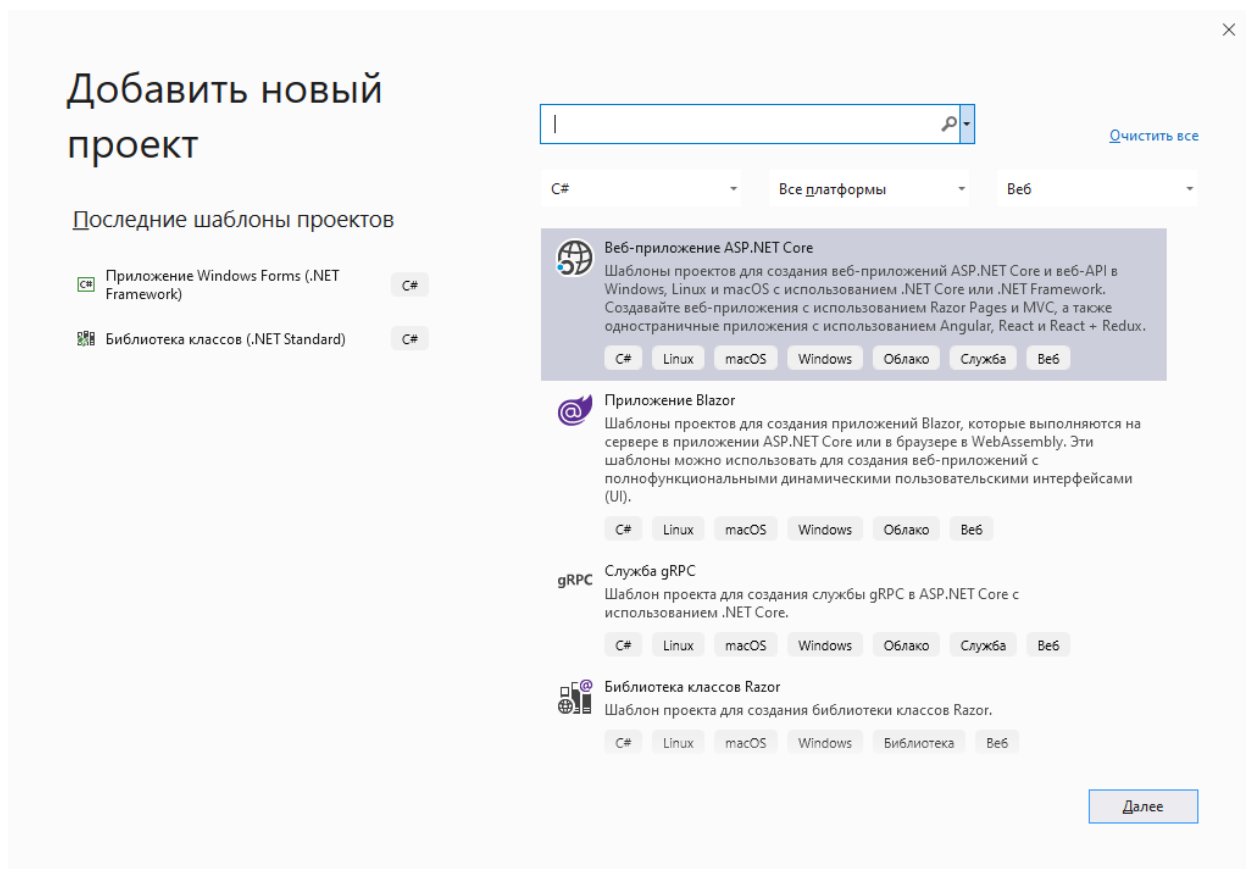


Рисунок 5.1 – Выбор типа проекта

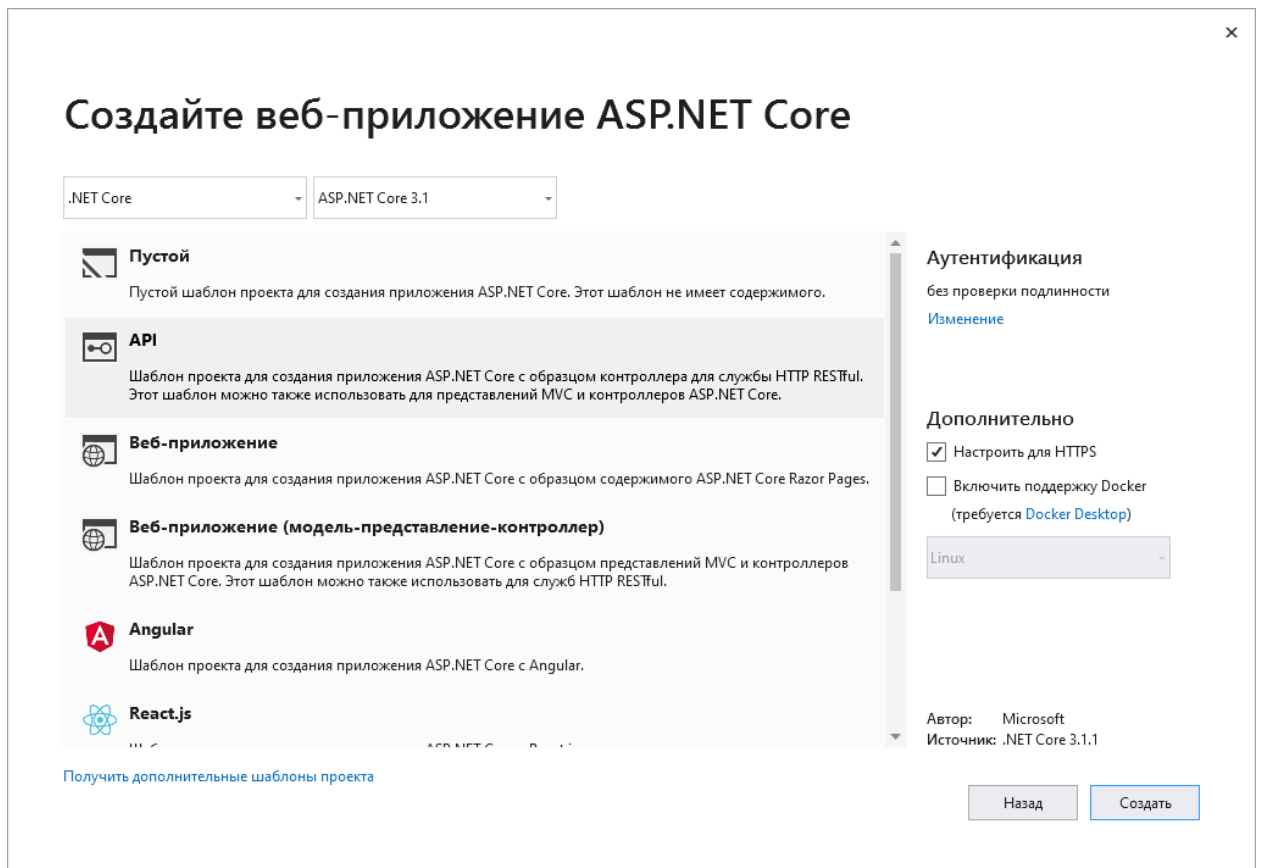


Рисунок 5.2 – Настройка проекта AbstractShopRestApi

Подключим библиотеки через Nuget-пакеты:

- Microsoft.AspNetCore.Mvc.NewtonsoftJson

Добавим ссылки на проекты:

- **AbstractShopBusinessLogic**
- **AbstractShopDatabaseImplement**

В проекте уже есть встроенный IoC-контейнер, так что не потребуется ничего дополнительно подключать. В классе Startup в методе ConfigureServices пропишем связи (листинг 5.4).

```
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopDatabaseImplement.Implements;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AbstractShopRestApi
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {

```

```

        Configuration = configuration;
    }

    public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to the
    container.
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddTransient<IClientStorage, ClientStorage>();
        services.AddTransient<IOrderStorage, OrderStorage>();
        services.AddTransient<IProductStorage, ProductStorage>();
        services.AddTransient<OrderLogic>();
        services.AddTransient<ClientLogic>();
        services.AddTransient<ProductLogic>();
        services.AddControllers().AddNewtonsoftJson();
    }

    // This method gets called by the runtime. Use this method to configure the HTTP
    request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}

```

Листинг 5.4 – Класс Startup

Сделаем 2 контроллера: один для работы с клиентами (листинг 5.5), а второй для работы с заказами (листинг 5.6).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.ViewModels;
using Microsoft.AspNetCore.Mvc;

namespace AbstractShopRestApi.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class ClientController : ControllerBase
    {
        private readonly ClientLogic _logic;

        public ClientController(ClientLogic logic)
        {
            _logic = logic;
        }
    }
}

```

```

        [HttpGet]
        public ClientViewModel Login(string login, string password) => _logic.Read(new
ClientBindingModel { Email = login, Password = password })?[0];

        [HttpPost]
        public void Register(ClientBindingModel model) => _logic.CreateOrUpdate(model);

        [HttpPost]
        public void UpdateData(ClientBindingModel model) => _logic.CreateOrUpdate(model);
    }
}

```

Листинг 5.5 – Контроллер ClientController

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.ViewModels;
using Microsoft.AspNetCore.Mvc;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopRestApi.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class MainController : ControllerBase
    {
        private readonly OrderLogic _order;

        private readonly ProductLogic _product;

        private readonly OrderLogic _main;

        public MainController(OrderLogic order, ProductLogic product, OrderLogic main)
        {
            _order = order;
            _product = product;
            _main = main;
        }

        [HttpGet]
        public List<ProductViewModel> GetProductList() => _product.Read(null)?.ToList();

        [HttpGet]
        public ProductViewModel GetProduct(int productId) => _product.Read(new
ProductBindingModel { Id = productId })?[0];

        [HttpGet]
        public List<OrderViewModel> GetOrders(int clientId) => _order.Read(new
OrderBindingModel { ClientId = clientId });

        [HttpPost]
        public void CreateOrder(CreateOrderBindingModel model) =>
            _main.CreateOrder(model);
    }
}

```

Листинг 5.6 – Контроллер MainController

Логика в методах контроллеров будет простой: получить данные, если требуется и вызывать нужный метод из нужного интерфейса. Для клиента

нам потребуется 3 действия: регистрация (добавление) новых клиентов, вход в систему и изменение данных клиента. Для заказов потребуются: получение списка заказов клиента, получение списка изделий (для выбора в заказ), получение изделия (для расчета стоимости) и создание заказа.

Протестируем разработанный сервис. Для этого запустим его. Это можно сделать 2 способами:

- назначить автозапускаемым;
- правой кнопкой щелкнуть по названию проекта, найти пункт «Отладка» и «Запустить новый экземпляр».

Выберем 2 вариант. После запуска откроется браузер, где в строке введем «<https://localhost:44344/api/main/getproductlist>» (**номер порта у вас будет иной!!!!**). Если все сделано верно, то получите список изделий (рисунок 5.3).

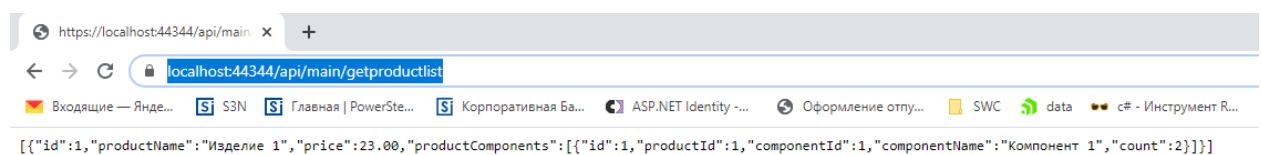


Рисунок 5.3 – Результат запроса

Последний шаг – создание приложения для клиентов. Создадим проект для клиентов (рисунок 5.4).

Создадим класс для подключения к серверу. Назовем его `APIClient`. Нам потребуется 3 метода: подключение к серверу, отправка get-запроса и отправка post-запроса (листинг 5.7).

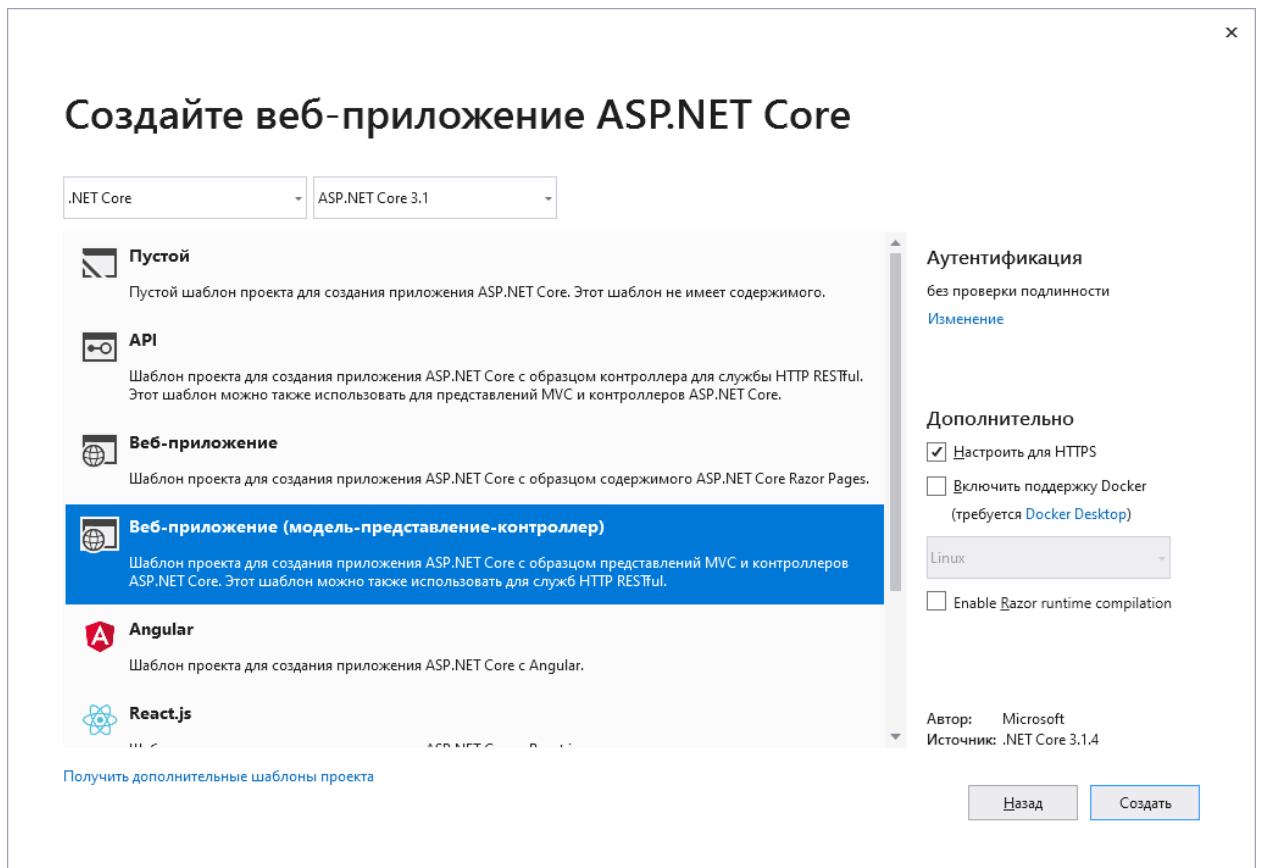


Рисунок 5.4 – Создание проекта AbstractShowClientApp

```
using Microsoft.Extensions.Configuration;
using Newtonsoft.Json;
using System;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Text;

namespace AbstractShowClientApp
{
    public static class APIClient
    {
        private static readonly HttpClient client = new HttpClient();

        public static void Connect(IConfiguration configuration)
        {
            client.BaseAddress = new Uri(configuration["IPAddress"]);
            client.DefaultRequestHeaders.Accept.Clear();
            client.DefaultRequestHeaders.Accept.Add(new
MediaTyewithQualityHeaderValue("application/json"));
        }

        public static T GetRequest<T>(string requestUrl)
        {
            var response = client.GetAsync(requestUrl);
            var result = response.Result.Content.ReadAsStringAsync().Result;
            if (response.Result.IsSuccessStatusCode)
            {
                return JsonConvert.DeserializeObject<T>(result);
            }
            else
            {
                throw new Exception(result);
            }
        }
    }
}
```

```

    }
}

public static void PostRequest<T>(string requestUrl, T model)
{
    var json = JsonConvert.SerializeObject(model);
    var data = new StringContent(json, Encoding.UTF8, "application/json");

    var response = client.PostAsync(requestUrl, data);

    var result = response.Result.Content.ReadAsStringAsync().Result;
    if (!response.Result.IsSuccessStatusCode)
    {
        throw new Exception(result);
    }
}
}
}
}

```

Листинг 5.7 – Класс APIClient

Строку подключения пропишем в настройках проекта в файле appsettings.json (листинг 5.8).

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "IPAddress": "http://localhost:62462/"
}

```

Листинг 5.8 – Файл appsettings.json

Номер порта нужно взять из настроек проекта-RestAPI (рисунок 5.5).

В проекте есть HomeController и 2 представления, Index и Privacy. Создавать еще контроллеры не будем, а вот представлений создадим (папка Views\Home). В Index будем выводить список заказов, в Privacy форму для редактирования данных клиента. Создадим представление для входа в систему (Enter.cshtml), регистрации (Register.cshtml) и создания заказа (Create.cshtml). Все формы будут частичного представления, чтобы основное меню и тело страницы было одно для всех (_Layout.cshtml).

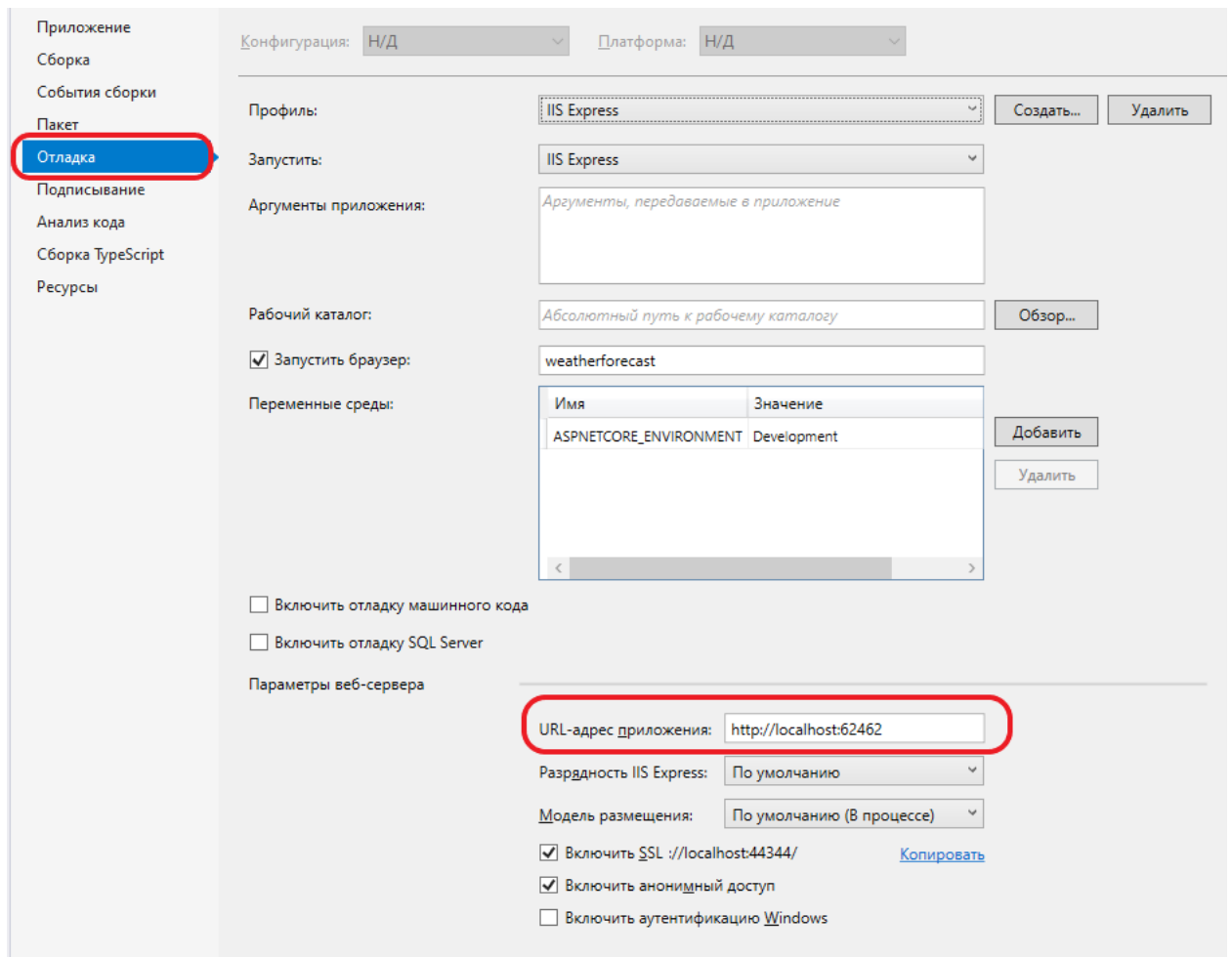


Рисунок 5.5 – Свойства проекта AbstractShopRestApi

После создания страниц перейдем в `_Layout.cshtml` и внесем туда несколько правок (листинг 5.8).

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>@ViewData["Title"] - AbstractShowClientApp</title>
  <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootstrap.min.css" />
  <link rel="stylesheet" href="~/css/site.css" />
  <script src="~/lib/jquery/dist/jquery.min.js"></script>
  <script src="~/lib/bootstrap/dist/js/bootstrap.bundle.min.js"></script>
</head>
<body>
  <header>
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-white
border-bottom box-shadow mb-3">
      <div class="container">
        <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">Абстрактный магазин</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-
target=".navbar-collapse" aria-controls="navbarSupportedContent"
          aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
        </button>
        <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-
```

```

reverse">
        <ul class="navbar-nav flex-grow-1">
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
asp-controller="Home" asp-action="Index">Заказы</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
asp-controller="Home" asp-action="Privacy">Личные данные</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
asp-controller="Home" asp-action="Enter">Вход</a>
            </li>
            <li class="nav-item">
                <a class="nav-link text-dark" asp-area=""
asp-controller="Home" asp-action="Register">Регистрация</a>
            </li>
        </ul>
    </div>
</div>
</nav>
</header>
<div class="container">
    <main role="main" class="pb-3">
        @RenderBody()
    </main>
</div>

<footer class="border-top footer text-muted">
    <div class="container">
        &copy; 2020 - Абстрактный магазин - <a asp-area="" asp-
controller="Home" asp-action="Privacy">Личные данные</a>
    </div>
</footer>
<script src="~/js/site.js" asp-append-version="true"></script>
@RenderSection("Scripts", required: false)
</body>
</html>

```

Листинг 5.8 – Представление _Layout.cshtml

Что сделали: добавили пункты меню, вписали названия на кириллице (это опционально) и подключение скриптов перенесли в header (потребуется при использовании javascript на одной из страниц).

Далее перейдем к представлениям и наполним их данными. Начнем с представления регистрации (листинг 5.9).

```

@{
    ViewData["Title"] = "Register";
}

<div class="text-center">
    <h2 class="display-4">Регистрация</h2>
</div>
<form method="post">
    <div class="row">
        <div class="col-4">Логин:</div>
        <div class="col-8"><input type="text" name="login" /></div>
    </div>

```



```

<div class="row">
    <div class="col-4">Пароль:</div>
    <div class="col-8"><input type="password" name="password" /></div>
</div>
<div class="row">
    <div class="col-4">ФИО:</div>
    <div class="col-8"><input type="text" name="fio" /></div>
</div>
<div class="row">
    <div class="col-8"></div>
    <div class="col-4"><input type="submit" value="Регистрация" class="btn btn-
primary" /></div>
</div>
</form>

```

Листинг 5.9 – Представление Register.cshtml

Тут просто форма с 3 полями (логин, пароль и ФИО) и кнопкой отправки данных на сервер.

Далее представление для входа в систему (листинг 5.10).

```

@{
    ViewData["Title"] = "Enter";
}

<div class="text-center">
    <h2 class="display-4">Вход в приложение</h2>
</div>
<form method="post">
    <div class="row">
        <div class="col-4">Логин:</div>
        <div class="col-8"><input type="text" name="login" /></div>
    </div>
    <div class="row">
        <div class="col-4">Пароль:</div>
        <div class="col-8"><input type="password" name="password" /></div>
    </div>
    <div class="row">
        <div class="col-8"></div>
        <div class="col-4"><input type="submit" value="Вход" class="btn btn-
primary" /></div>
    </div>
</form>

```

Листинг 5.10 – Представление Enter.cshtml

Представление для редактирования данных пользователя (листинг 5.11).

```

@using AbstractShopBusinessLogic.ViewModels
@model ClientViewModel

@{
    ViewData["Title"] = "Privacy Policy";
}

<div class="text-center">
    <h2 class="display-4">Личные данные</h2>
</div>
<form method="post">
    <div class="row">

```

```

        <div class="col-4">Логин:</div>
        <div class="col-8"><input type="text" name="login"
value="@Model.Email"/></div>
    </div>
    <div class="row">
        <div class="col-4">Пароль:</div>
        <div class="col-8"><input type="password" name="password"
value="@Model.Password"/></div>
    </div>
    <div class="row">
        <div class="col-4">ФИО:</div>
        <div class="col-8"><input type="text" name="fio"
value="@Model.ClientFIO"/></div>
    </div>
    <div class="row">
        <div class="col-8"></div>
        <div class="col-4"><input type="submit" value="Сохранить" class="btn btn-
primary" /></div>
    </div>
</form>

```

Листинг 5.11 – Представление Privacy.cshtml

Тут в представление будет передаваться модель и поля будут заполняться данными из модели.

Представление для создания заказа (листинг 5.12).

```

@{
    ViewData["Title"] = "Create";
}
<div class="text-center">
    <h2 class="display-4">Создание заказа</h2>
</div>
<form method="post">
    <div class="row">
        <div class="col-4">Изделие:</div>
        <div class="col-8">
            <select id="product" name="product" class="form-control" asp-
items="@((new SelectList(@ViewBag.Products,"Id", "ProductName")))"></select>
        </div>
    </div>
    <div class="row">
        <div class="col-4">Количество:</div>
        <div class="col-8"><input type="text" name="count" id="count" /></div>
    </div>
    <div class="row">
        <div class="col-4">Сумма:</div>
        <div class="col-8"><input type="text" id="sum" name="sum" readonly="true"
/></div>
    </div>
    <div class="row">
        <div class="col-8"></div>
        <div class="col-4"><input type="submit" value="Создать" class="btn btn-
primary" /></div>
    </div>
</form>

<script>
    $('#product').on('change', function () {
        check();
    });
    $('#count').on('change', function () {

```

```

        check();
    });

    function check() {
        var count = $('#count').val();
        var product = $('#product').val();
        if (count && product) {
            $.ajax({
                method: "POST",
                url: "/Home/Calc",
                data: { count: count, product: product },
                success: function (result) {
                    $("#sum").val(result);
                }
            });
        }
    }
}
</script>

```

Листинг 5.12 – Представление Create.cshtml

Здесь будет выпадающий список с изделиями, а также через скрипт посылаться запрос на получение суммы на основе выбранного изделия и введенного количества (**ВНИМАНИЕ**, метод 'change' может не срабатывать в некоторых браузерах!).

И основное представление (листинг 5.13).

```

@using AbstractShopBusinessLogic.ViewModels
@model List<OrderViewModel>

@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Заказы</h1>
</div>

<div class="text-center">
    @{
        if (Model == null)
        {
            <h3 class="display-4">Авторизируйтесь</h3>
            return;
        }

        <p>
            <a asp-action="Create">Создать заказ</a>
        </p>
        <table class="table">
            <thead>
                <tr>
                    <th>
                        Номер
                    </th>
                    <th>
                        Изделие
                    </th>
                </tr>
            </thead>
        </table>
    }

```

```

        </th>
        <th>
            Дата создания
        </th>
        <th>
            Количество
        </th>
        <th>
            Сумма
        </th>
        <th>
            Статус
        </th>
    </tr>
</thead>
<tbody>
    @foreach (var item in Model)
    {
        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Id)
            </td>
            <td>
                @Html.DisplayFor(modelItem =>
item.ProductName)
            </td>
            <td>
                @Html.DisplayFor(modelItem =>
item.DateCreate)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Count)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Sum)
            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Status)
            </td>
        </tr>
    }
</tbody>
</table>
}
</div>

```

Листинг 5.13 – Представление Index.cshtml

В представление будет передаваться список заказов и выводиться в табличном виде. Также действие на создание заказа будет не в меню, а на этой форме.

Вся логика будет сосредоточена в HomeController (листинг 5.14).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShowClientApp.Models;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Diagnostics;

```

```

namespace AbstractShowClientApp.Controllers
{
    public class HomeController : Controller
    {
        public HomeController()
        {
        }

        public IActionResult Index()
        {
            if(Program.Client == null)
            {
                return Redirect("~/Home/Enter");
            }
            return
View(APIClient.GetRequest<List<OrderViewModel>>($"api/main/getorders?clientId={Program.Cl
ient.Id}"));
        }

        [HttpGet]
        public IActionResult Privacy()
        {
            if (Program.Client == null)
            {
                return Redirect("~/Home/Enter");
            }
            return View(Program.Client);
        }

        [HttpPost]
        public void Privacy(string login, string password, string fio)
        {
            if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password)
&& !string.IsNullOrEmpty(fio))
            {
                //прописать запрос
                Program.Client.ClientFIO = fio;
                Program.Client.Email = login;
                Program.Client.Password = password;
                Response.Redirect("Index");

                return;
            }

            throw new Exception("Введите логин, пароль и ФИО");
        }

        [ResponseCache(Duration = 0, Location = ResponseCacheLocation.None, NoStore
= true)]
        public IActionResult Error()
        {
            return View(new ErrorViewModel { RequestId = Activity.Current?.Id ??
HttpContext.TraceIdentifier });
        }

        [HttpGet]
        public IActionResult Enter()
        {
            return View();
        }

        [HttpPost]
        public void Enter(string login, string password)

```

```

        {
            if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password))
            {
                Program.Client =
APIClient.GetRequest<ClientViewModel>($"api/client/login?login={login}&password={password
}");

                if (Program.Client == null)
                {
                    throw new Exception("Неверный логин/пароль");
                }

                Response.Redirect("Index");
                return;
            }
            throw new Exception("Введите логин, пароль");
        }

[HttpGet]
public IActionResult Register()
{
    return View();
}

[HttpPost]
public void Register(string login, string password, string fio)
{
    if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password)
&& !string.IsNullOrEmpty(fio))
    {
        APIClient.PostRequest("api/client/register", new
ClientBindingModel
        {
            ClientFIO = fio,
            Email = login,
            Password = password
        });
        Response.Redirect("Enter");
        return;
    }
    throw new Exception("Введите логин, пароль и ФИО");
}

[HttpGet]
public IActionResult Create()
{
    ViewBag.Products =
APIClient.GetRequest<List<ProductViewModel>>("api/main/getproductlist");
    return View();
}

[HttpPost]
public void Create(int product, int count, decimal sum)
{
    if (count == 0 || sum == 0)
    {
        return;
    }
    //прописать запрос
    Response.Redirect("Index");
}

[HttpPost]
public decimal Calc(decimal count, int product)
{

```

```

        ProductViewModel prod =
        APIClient.GetRequest<ProductViewModel>($"api/main/getproduct?productId={product}");
        return count * prod.Price;
    }
}

```

Листинг 5.14 – Класс HomeController

Для перехода между страницами используем команды Redirect. Для получение данных – запросы к RESTAPI-серверу.

Последний штрих, в классе Startup прописать вызов метода Connect класса APIClient (листинг 5.15).

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;

namespace AbstractShowClientApp
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
            APIClient.Connect(configuration);
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services
        to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllersWithViews();
        }

        // This method gets called by the runtime. Use this method to configure the
        HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Home/Error");
                // The default HSTS value is 30 days. You may want to change
                this for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }
            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

```

```

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllerRoute(
                name: "default",
                pattern: "{controller=Home}/{action=Index}/{id?}");
        });
    }
}

```

Листинг 5.15 – Класс Startup

А в классе Program добавить свойство Client от ClientViewModel (листинг 5.16).

```

using AbstractShopBusinessLogic.ViewModels;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace AbstractShowClientApp
{
    public class Program
    {
        public static ClientViewModel Client { get; set; }

        public static void Main(string[] args) =>
        CreateHostBuilder(args).Build().Run();

        public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
    }
}

```

Листинг 5.16 – Класс Program

Для корректной работы требуется запускать одновременно оба проекта, и RestAPI и web- приложение клиента.

Требования

1. Названия контроллеров и методов в них должны соответствовать логике вашего задания по варианту.
2. Прописать интерфейс для логики хранения, bindingModel и viewModel для клиента.
3. В заказе учитывать клиента, который его создает
4. Написать класс с бизнес-логикой для клиента (делать проверку на уникальное поле «логин»)

5. Написать 3 реализации для интерфейса логики хранения для сущности «Клиент».
6. В OrderStorage (3 реализации) учитывать наличие клиента при обработке заказов (отдавать заказы только клиента).
7. Для классов ClientBindingModel, OrderBindingModel, ClientViewModel, OrderViewModel, ProductComponentViewModel, ProductViewModel настроить возможность сериализации.
8. Прописать запрос на создание заказа.
9. Прописать запрос на изменение данных клиента.
10. В проекте **AbstractShopView** при создании заказа реализовать возможность выбора клиента для заказа, а также скорректировать вывод списка заказов.
11. В проекте **AbstractShopView** добавить функционал для вывода списка клиентов и возможности их удаления.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- дополнить RestAPI-приложение контроллером для работы со складами;
- в контроллере прописать методы для получения списка складов, создания склада, редактирования склада, удаление склада и пополнение склада;
- создать web-приложение для работы со складами с использованием сервиса RestAPI;
- вход в приложение осуществлять через ввод пароля (пароль задать в файле конфигурации);
- в приложении сделать представления для вывода списка складов (главная), для добавления, редактирования (отображать список компонент склада), удаления и пополнения складов.

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
- 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).

ЛАБОРАТОРНАЯ РАБОТА №6.

МНОГОПОТОЧНОСТЬ

Цель

Освоить многопоточность в представлении пользователя.

Задание

1. Создать ветку от ветки пятой лабораторной.
2. Добавить исполнителей, которые будут выполнять заказы. Реализовать функционал, имитирующий выполнение заказов (перевод в статусы «в работу» с назначением исполнителя и в статус «Готово»). Поменять интерфейс программы, убрать возможность переводить заказ в статусы «Выполняется» и «Готов» с формы, добавить функционал для вызова метода имитации работы.
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Решение

В моделях добавляем новый класс – исполнитель (листинги 6.1, 6.2)

```
namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Исполнитель, выполняющий заказы
    /// </summary>
    public class ImplementerBindingModel
    {
        public int Id { get; set; }

        public string ImplementerFIO { get; set; }

        public int WorkingTime { get; set; }

        public int PauseTime { get; set; }
    }
}
```

Листинг 6.1 – Класс ImplementerBindingModel

```
using System.ComponentModel;

namespace AbstractShopBusinessLogic.ViewModels
{
    /// <summary>
    /// Исполнитель, выполняющий заказы
    /// </summary>
    public class ImplementerViewModel
    {
        public int Id { get; set; }
    }
}
```

```

        [DisplayName("ФИО исполнителя")]
        public string ImplementerFIO { get; set; }

        [DisplayName("Время на заказ")]
        public int WorkingTime { get; set; }

        [DisplayName("Время на перерыв")]
        public int PauseTime { get; set; }
    }
}

```

Листинг 6.2 – Класс ImplementerViewModel

В класс изменения данных по заказу добавляем новое поле – идентификатор исполнителя (необязательный к заполнению) (листинг 6.3) и в BindingModel и ViewModel заказа добавляем идентификатор и ФИО исполнителя (ФИО только во ViewModel).

```

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Данные для смены статуса заказа
    /// </summary>
    public class ChangeStatusBindingModel
    {
        public int OrderId { get; set; }

        public int? ImplementerId { get; set; }
    }
}

```

Листинг 6.3 – Класс ChangeStatusBindingModel с новым полем

Добавим интерфейс для работы с исполнителями (листинг 6.4).

Также расширим перечень условий, по которым будем отбирать заказы. Введем 2 новых условия (листинг 6.5):

1. Получение заказов в статусе «Принят», чтобы отправлять их в работу.
2. Получение заказов в статусе «Выполняется» для конкретных исполнителей, чтобы они могли продолжить выполнение заказов после прерывания.

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.Interfaces
{
    public interface IImplementerStorage
    {
        List<ImplementerViewModel> GetFullList();

        List<ImplementerViewModel> GetFilteredList(ImplementerBindingModel model);

        ImplementerViewModel GetElement(ImplementerBindingModel model);

        void Insert(ImplementerBindingModel model);

        void Update(ImplementerBindingModel model);

        void Delete(ImplementerBindingModel model);
    }
}

```

Листинг 6.4 – Интерфейс ImplementerStorage

```

using AbstractShopBusinessLogic.Enums;
using System;

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Заказ
    /// </summary>
    public class OrderBindingModel
    {
        public int? Id { get; set; }

        public int? ClientId { get; set; }

        public int ProductId { get; set; }

        public int? ImplementerId { get; set; }

        public int Count { get; set; }

        public decimal Sum { get; set; }

        public OrderStatus Status { get; set; }

        public DateTime DateCreate { get; set; }

        public DateTime? DateImplement { get; set; }

        public DateTime? DateFrom { get; set; }

        public DateTime? DateTo { get; set; }

        public bool? FreeOrders { get; set; }
    }
}

```

Листинг 6.5 – Класс OrderBindingModel с новыми полями

Перейдем к реализациям. Нам потребуется:

1. Создать модель для исполнителя.
2. В модель заказа вставить идентификатор исполнителя (необязательный параметр, так как проставляться будет на этапе передачи в работу заказа).
3. В хранилище добавить список исполнителей:
 - a. для простого списка добавить новое поле – список исполнителей;
 - b. для файлового хранилища добавить новое поле – список исполнителей, методы загрузки и сохранения списка исполнителей, а также в заказе учесть новое поле – исполнитель;
 - c. для базы данных добавить новый DataSet по исполнителям, создать миграцию и применить ее в базе данных.
4. Реализовать интерфейс ImplementerStorage (для 3-х вариаций).
5. В реализациях интерфейса IOrderStorage изменить:
 - a. в методе получения фильтрованных заказов добавить 2 новых фильтра;
 - b. в методе получения заказов проставлять идентификатор исполнителя и его ФИО, если есть;
 - c. в методе создания и изменения заказа проставлять идентификатор исполнителя.
6. В классе MainLogic в методе перевода заказа в работу добавить проверку, что на работу не назначен исполнитель, и проставлять исполнителя, если все условия выполняются.

Рассмотрим, как изменится реализация на примере работы с БД (листинг 6.6).

```
public List<OrderViewModel> GetFilteredList(OrderBindingModel model)
{
    if (model == null)
    {
        return null;
    }
}
```



```

    }

    using (var context = new AbstractShopDatabase())
    {
        return context.Orders
            .Include(rec => rec.Product)
            .Include(rec => rec.Client)
            .Include(rec => rec.Implementer)
            .Where(rec => (!model.DateFrom.HasValue && !model.DateTo.HasValue &&
rec.DateCreate.Date == model.DateCreate.Date) ||
            (model.DateFrom.HasValue && model.DateTo.HasValue &&
rec.DateCreate.Date >= model.DateFrom.Value.Date && rec.DateCreate.Date <=
model.DateTo.Value.Date) ||
            (model.ClientId.HasValue && rec.ClientId == model.ClientId) ||
            (model.FreeOrders.HasValue && model.FreeOrders.Value && rec.Status ==
OrderStatus.Принят) ||
            (model.ImplementerId.HasValue && rec.ImplementerId ==
model.ImplementerId && rec.Status == OrderStatus.Выполняется))
            .Select(rec => new OrderViewModel
            {
                Id = rec.Id,
                Count = rec.Count,
                DateCreate = rec.DateCreate,
                DateImplement = rec.DateImplement,
                ProductId = rec.ProductId,
                ProductName = rec.Product.ProductName,
                ClientId = rec.ClientId,
                ClientFIO = rec.Client.ClientFIO,
                ImplementerId = rec.ImplementerId,
                ImplementerFIO = rec.ImplementerId.HasValue ?
rec.Implementer.ImplementerFIO : string.Empty,
                Status = rec.Status,
                Sum = rec.Sum
            })
            .ToList();
    }
}

private Order CreateModel(OrderBindingModel model, Order order)
{
    order.Count = model.Count;
    order.DateCreate = model.DateCreate;
    order.DateImplement = model.DateImplement;
    order.ProductId = model.ProductId;
    order.ClientId = model.ClientId.Value;
    order.ImplementerId = model.ImplementerId;
    order.Status = model.Status;
    order.Sum = model.Sum;

    return order;
}

```

Листинг 6.6 – Измененные методы класса OrderStorage

В проекте **AbstractShopBusinessLogic** создадим класс для моделирования работы исполнителей (листинг 6.7).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;
using System.Threading;

```

```

using System.Threading.Tasks;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class WorkModeling
    {
        private readonly IImplementerStorage _implementerStorage;

        private readonly IOrderStorage _orderStorage;

        private readonly OrderLogic _orderLogic;

        private readonly Random rnd;

        public WorkModeling(IImplementerStorage implementerStorage, IOrderStorage
orderStorage, OrderLogic orderLogic)
        {
            this._implementerStorage = implementerStorage;
            this._orderStorage = orderStorage;
            this._orderLogic = orderLogic;

            rnd = new Random(1000);
        }

        /// <summary>
        /// Запуск работ
        /// </summary>
        public void DoWork()
        {
            var implementers = _implementerStorage.GetFullList();

            var orders = _orderStorage.GetFilteredList(new OrderBindingModel { FreeOrders
= true });

            foreach (var implementer in implementers)
            {
                WorkerWorkAsync(implementer, orders);
            }
        }

        /// <summary>
        /// Имитация работы исполнителя
        /// </summary>
        /// <param name="implementer"></param>
        /// <param name="orders"></param>
        private async void WorkerWorkAsync(ImplementerViewModel implementer,
List<OrderViewModel> orders)
        {
            // ищем заказы, которые уже в работе (вдруг исполнителя прервали)
            var runOrders = await Task.Run(() => _orderStorage.GetFilteredList(new
OrderBindingModel { ImplementerId = implementer.Id }));
            foreach (var order in runOrders)
            {
                // делаем работу заново
                Thread.Sleep(implementer.WorkingTime * rnd.Next(1, 5) * order.Count);

                _orderLogic.FinishOrder(new ChangeStatusBindingModel { OrderId = order.Id
});

                // отдыхаем
                Thread.Sleep(implementer.PauseTime);
            }

            await Task.Run(() =>

```

```

        {
            foreach (var order in orders)
            {
                // пытаемся назначить заказ на исполнителя
                try
                {
                    _orderLogic.TakeOrderInWork(new ChangeStatusBindingModel {
OrderId = order.Id, ImplementerId = implementer.Id });

                    // делаем работу
                    Thread.Sleep(implementer.WorkingTime * rnd.Next(1, 5) *
order.Count);

                    _orderLogic.FinishOrder(new ChangeStatusBindingModel { OrderId =
order.Id });

                    // отдыхаем
                    Thread.Sleep(implementer.PauseTime);
                }
                catch (Exception) { }
            }
        }
    }
}

```

Листинг 6.7 – Класс WorkModeling

Для работы нам понадобятся 2 интерфейса, ImplementerStorage (получение списка исполнителей) и IOrderStorage (получение заказов) и класс с логикой заказов (смена статуса заказа). Главный метод будет получать список исполнителей, а также список принятых заказов. Далее этот список будет передаваться каждому исполнителю. Для каждого исполнителя будет вызываться метод (асинхронно) в нем сперва будет получаться список заказов, которые исполнитель не доделал по каким-то причинам (прервали работу имитации) и делать их заново. После завершения этих заказов исполнитель пойдет по принятым заказам, будет пытаться взять заказ в работу (если его уже кто-то успел взять, то будет вылетать ошибка, так что ставим try-catch) и выполнять его. Как это будет работать. При вызове метода DoWork будет получено 2 списка, исполнителей и заказов. Далее для каждого исполнителя будет вызван метод WorkerWorkAsync. При этом, цикл foreach по исполнителям не будет ждать, пока выполнится метод WorkerWorkAsync для первого исполнителя, а продолжит свою работу (т.е. будет вызывать метод WorkerWorkAsync для других исполнителей). Таким образом будет вызвано сразу несколько методов WorkerWorkAsync для разных исполнителей и все

они сразу смогут выполнять заказы. Внутри метода `WorkerWorkAsync` будет вызов двух задач (`Task`). Метод не будет продолжать работу, пока каждый из вызовов `Task` не завершится (за счет вызова через `await`). Так что один сотрудник не успеет взять сразу все заказы в работу, а будет постепенно выполнять заказы.

Узким местом в работе с заказами будет метод перевод заказа в работу. Если 2 исполнителя разом захотят взять заказ в работу, то может произойти накладка. Первый исполнитель получит из базы информацию по заказу, начнет ее проверку, и второй исполнитель в это же время получит информацию по этому же заказу (первый его еще не успеет забить за собой) и начнет ее проверку. Получится, что первый проверит заказ, убедится, что он свободен и возьмет в работу и второй убедится, что заказ свободен и возьмет его в работу (сменит исполнителя с первого на себя). Чтобы такого не было поставим на код перевода заказа в работу оператор `lock`, чтобы только один поток мог с ним работать за раз (листинг 6.8).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Enums;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using System;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class OrderLogic
    {
        private readonly object locker = new object();

        public void TakeOrderInWork(ChangeStatusBindingModel model)
        {
            lock (locker)
            {
                var order = _orderStorage.GetElement(new OrderBindingModel { Id =
model.OrderId });
                if (order == null)
                {
                    throw new Exception("Не найден заказ");
                }
                if (order.Status != OrderStatus.Принят)
                {
                    throw new Exception("Заказ не в статусе \"Принят\"");
                }
                if (order.ImplementerId.HasValue)
                {
                    throw new Exception("У заказа уже есть исполнитель");
                }
            }
        }
    }
}
```

```

        orderStorage.Update(new OrderBindingModel
        {
            Id = order.Id,
            ClientId = order.ClientId,
            ImplementerId = model.ImplementerId,
            ProductId = order.ProductId,
            Count = order.Count,
            Sum = order.Sum,
            DateCreate = order.DateCreate,
            DateImplement = DateTime.Now,
            Status = OrderStatus.Выполняется
        });
    }
}
}
}

```

Листинг 6.8 – Измененные элементы класса OrderLogic

В проекте **AbstractShopView** вноси ряд изменений. Во-первых, в контейнере указываем какую реализацию для интерфейса `ImplementerStorage` будем использовать. Во-вторых, добавляем формы для работы с исполнителями. В-третьих, меняем основную форму (рисунок 6.1).

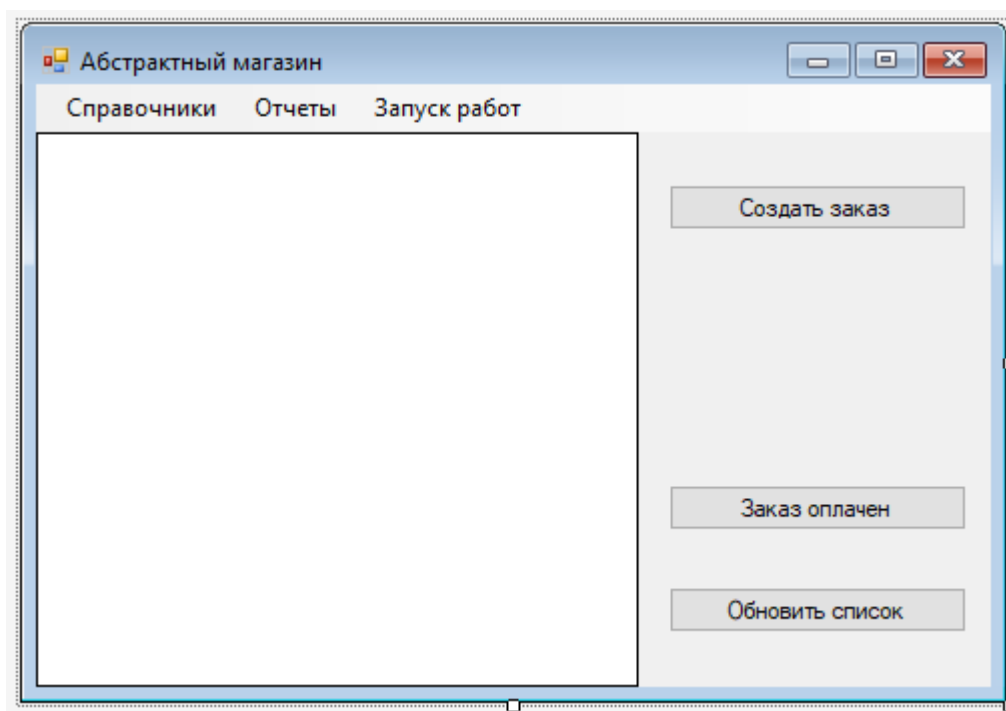


Рисунок 6.1 – Главная форма после изменений

Убираем кнопки для смены статусов заказа. Добавляем пункт меню для вызова метода имитации работы исполнителей. В пункте «Справочники» добавляем новый подпункт «Исполнители». Не забываем, что изменится порядок элементов в списке заказов!

Требования

1. Названия моделей, полей, контроллеров, форм них должны соответствовать логике вашего задания по варианту.
2. Прописать класс с бизнес-логикой для сущности «Исполнитель».
3. Прописать 3 реализации для интерфейса логики хранения для сущности «Клиент».
4. В OrderStorage (3 реализации) добавить филбтры для выбора новых заказов и заказов в работе дял конкретного исполнителя.
5. Добавить формы для работы с исполнителями.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- в случае, если для выполнения заказов не хватает материалов, заказ должен переводиться в статус «Требуется материалы» и исполнитель должен будет переходить к следующему заказу;
- для исполнителя при имитации работы поменять алгоритм работы: сперва отрабатываются заказы со статусом «Выполняются», потом заказы со статусом «Требуется материалы» (вдруг материалы подвезли) и только потом новые заказы.

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
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).

ЛАБОРАТОРНАЯ РАБОТА №7. РЕГУЛЯРНЫЕ ВЫРАЖЕНИЯ

Цель

Ознакомиться с применением регулярных выражений.

Задание

1. Создать ветку от ветки шестой лабораторной.
2. У клиента добавить проверки, что в поле логин вводится адрес электронной почты, а пароль удовлетворяет ряду условий. Делать оповещение клиента (отправка письма) при смене статусов его заказов. Проверять почту на предмет писем от клиентов.
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа .exe или .bin). Создать pull request.

Решение

Рассмотрим технологии, используемые для отправки/получения почтовых сообщений. Работа с почтой осуществляется через определенные протоколы передачи данных (наборы соглашений логического уровня об интерфейсах передачи данных). Для отправки сообщений в сетях TCP/IP предназначен сетевой протокол SMTP (англ. Simple Mail Transfer Protocol – простой протокол передачи почты). Он используется для отправки почты от пользователей к серверам и между серверами для дальнейшей пересылки к получателю.

Данный протокол получил распространение в начале 80-х годов. До этого использовался протокол UUCP, основным недостатком которого была необходимость знания полного маршрута от отправителя до получателя и явного указания этого маршрута в адресе получателя либо наличия прямого коммутируемого или постоянного соединения между компьютерами отправителя и получателя.

Сервер SMTP представляет собой конечный автомат с внутренним состоянием. Клиент передает на сервер строку вида: «команда<пробел>параметры<перевод строки>». Сервер отвечает

на каждую команду строкой, содержащей код ответа и текстовое сообщение, отделенное пробелом. Код ответа представляет собой число в диапазоне от 100 до 999, представленное в виде строки. Коды имеют следующие значения:

2XX – команда успешно выполнена

3XX – ожидаются дополнительные данные от клиента

4XX – временная ошибка, клиент должен произвести следующую попытку через некоторое время

5XX – неустраняемая ошибка

Текстовая часть ответа носит справочный характер. Общение между клиентом и сервером может осуществляться через ряд портов, открытых на почтовом сервере.

Рассмотрим пример обычной SMTP-сессии. В данном примере

C: – клиент, а S: – сервер:

S: (ожидает соединения)

C: (подключается к порту 25 сервера)

S:220 mail.MyCompany.ltd ESMTP is glad to see you!//подключение успешно

C:HELO //начать работу

S:250 domain name should be qualified //укажите отправителя

C:MAIL FROM: user1name@company.ru //адрес отправителя

S:250 user1name@company.ru sender accepted //адрес принят

C:RCPT TO:user2@company.ltd // адрес получателя

S:250 user2@company.ltd ok // адрес принят

C:RCPT TO: user3@company.ltd //еще адрес получателя

S:550 user3@company.ltd unknown user account //адреса не существует

C:DATA // передача данных письма

S:354 Enter mail, end with "." on a line by itself //ожидаются данные,

//по окончании ввода ожидается точка.

C:from: user1name@company.ru //текст письма

C:to: user2@company.ltd //текст письма

C:subject: тема //текст письма

C: //текст письма

C:Hi! //текст письма

C:. //текст письма

S:250 769947 message accepted for delivery//письмо принято для доставки

C:QUIT //клиент отключается

S:221 mail.company.tld ESMTP closing connection //сервер принял команду

S: (закрывает соединение)

В результате такой сессии письмо будет доставлено адресату user2@company.ltd, но не будет доставлено адресату user3@company.ltd, потому что такого адреса не существует.

Проверка почты. Получение писем.

Получение почтовых сообщений с сервера осуществляется посредством протокола POP3 (англ. Post Office Protocol Version 3 – протокол почтового отделения, версия 3). В данном протоколе предусмотрено 3 состояния сеанса:

- авторизация, когда клиент проходит процедуру аутентификации;
- транзакция, когда клиент получает информацию о состоянии почтового ящика, принимает и удаляет почту;
- обновление, когда сервер удаляет выбранные письма и закрывает соединение.

В таблице 1 приведены команды, поддерживаемые протоколом POP.

Таблица 1. Команды протокола POP

Команда	Описание	Аргументы	Ограничения	Возможные ответы
APOP [имя] [digest]	Команда служит для передачи серверу имени пользователя и зашифрованного пароля (digest).	[имя] – строка, указывающая имя почтового ящика. [digest] — хеш-сумма временной метки, связанной с паролем пользователя, вычисленная по алгоритму MD5. Временная метка получается при соединении с сервером.	Ее поддержка не является обязательной.	+OK maildrop has n message; -ERR password supplied for [имя] is incorrect.
USER [имя]	Передает серверу имя пользователя.	[имя] – строка, указывающая имя почтового ящика.	–	+OK name is a valid mailbox; -ERR never heard of mailbox name.
DELE [сообщение]	Сервер помечает указанное сообщение для удаления. Такие сообщения реально удаляются только после закрытия транзакции (закрытие происходит по команде QUIT или иногда по истечении времени, установленного сервером).	[сообщение] – номер сообщения.	Доступна после успешной идентификации.	+OK message deleted; -ERR no such message.
PASS [пароль]	Передает серверу пароль почтового ящика.	[пароль] – пароль для почтового ящика.	Работает после успешной передачи имени почтового ящика.	+OK maildrop locked and ready; -ERR invalid password; -ERR unable to lock maildrop.

Команда	Описание	Аргументы	Ограничения	Возможные ответы
LIST [сообщение]	Если был передан аргумент, сервер выдаёт информацию об указанном сообщении. Если аргумент не был передан, то сервер выдаёт информацию обо всех сообщениях, находящихся в почтовом ящике. Сообщения, помеченные для удаления, не перечисляются.	[сообщение] – номер сообщения (необязательный аргумент).	Доступна после успешной идентификации.	+OK scan listing follows; -ERR no such message.
NOOP	Сервер ничего не делает, всегда отвечает положительно. Команда используется для поддержки соединения с сервером при длительном бездействии.	—	Доступна после успешной идентификации.	+OK.
RETR [сообщение]	Сервер передаёт сообщение с указанным номером.	[сообщение] – номер сообщения.	Доступна после успешной идентификации.	+OK message follows; -ERR no such message.
RSET	Этой командой производится откат транзакций внутри сессии. Например, если пользователь случайно пометил на удаление какие-либо сообщения, он может убрать эти пометки, отправив эту команду.	—	Доступна после успешной идентификации.	+OK.

Команда	Описание	Аргументы	Ограничения	Возможные ответы
STAT	Сервер возвращает количество сообщений в почтовом ящике плюс размер, занимаемый этими сообщениями на почтовом ящике.	—	Доступна после успешной идентификации.	+OK [количество] [размер].
TOP [сообщение] [количество строк]	Сервер возвращает заголовки указанного сообщения, пустую строку и указанное количество первых строк тела сообщения.	[сообщение] – номер сообщения. [количество строк] – сколько строк нужно вывести.	Доступна после успешной идентификации.	+OK n octets; -ERR no such message.
QUIT	Закрытие соединения	—	—	+OK.

По аналогии с сервером SMTP рассмотрим пример сессии с сервером POP3:

```

S: <Сервер ожидает входящих соединений на порту 110>
C: <подключается к серверу>
S:  +OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
C:  APOP mrose c4c9334bac560ecc979e58001b3e22fb
S:  +OK mrose's maildrop has 2 messages (320 octets)
C:  STAT
S:  +OK 2 320
C:  LIST
S:  +OK 2 messages (320 octets)
S:  1 120
S:  2 200
S:  .
C:  RETR 1
S:  +OK 120 octets
S:  <сервер передает сообщение 1>
S:  .
C:  DELE 1

```

```

S: +OK message 1 deleted
C: RETR 2
S: +OK 200 octets
S: <сервер передает сообщение 2>
S: .
C: DELE 2
S: +OK message 2 deleted
C: QUIT
S: +OK dewey POP3 server signing off (maildrop empty)
C: <закрывает соединение>
S: <продолжает ждать входящие соединения>

```

Перейдем к нашему проекту.

Первым делом реализуем хранение писем. Заведем классы для получения письма (листинг 7.1) и вывод письма (листинг 7.2).

```

using System;
using System.Runtime.Serialization;

namespace AbstractShopBusinessLogic.BindingModels
{
    /// <summary>
    /// Сообщения, приходящие на почту
    /// </summary>
    [DataContract]
    public class MessageInfoBindingModel
    {
        [DataMember]
        public int? ClientId { get; set; }

        [DataMember]
        public string MessageId { get; set; }

        [DataMember]
        public string FromMailAddress { get; set; }

        [DataMember]
        public string Subject { get; set; }

        [DataMember]
        public string Body { get; set; }

        [DataMember]
        public DateTime DateDelivery { get; set; }
    }
}

```

Листинг 7.1 – Класс MessageInfoBindingModel

```

using System;
using System.ComponentModel;
using System.Runtime.Serialization;

```



```

namespace AbstractShopBusinessLogic.ViewModels
{
    /// <summary>
    /// Сообщения, приходящие на почту
    /// </summary>
    [DataContract]
    public class MessageInfoViewModel
    {
        [DataMember]
        public string MessageId { get; set; }

        [DisplayName("Отправитель")]
        [DataMember]
        public string SenderName { get; set; }

        [DisplayName("Дата письма")]
        [DataMember]
        public DateTime DateDelivery { get; set; }

        [DisplayName("Заголовок")]
        [DataMember]
        public string Subject { get; set; }

        [DisplayName("Текст")]
        [DataMember]
        public string Body { get; set; }
    }
}

```

Листинг 7.2 – Класс MessageInfoViewModel

Пропишем интерфейс (листинг 7.3). Он будет состоят из трех методов. Первые для получения полного и фильтрованного списков сообщений, третий – для добавления нового письма. Не будем прописывать функции редактирования и удаления писем, так как мы их будем строго из почты вытаскивать. При получении списка предусмотрим 2 варианта: получение полного списка писем (для проекта **AbstractShopView**) и получение писем конкретного пользователя (для проекта **AbstractShopClientView**). С этой целью в классе MessageInfoBindingModel пропишем необязательное поле ClientId.

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.ViewModels;
using System.Collections.Generic;

namespace AbstractShopBusinessLogic.Interfaces
{
    public interface IMessageInfoStorage
    {
        List<MessageInfoViewModel> GetFullList();

        List<MessageInfoViewModel> GetFilteredList(MessageInfoBindingModel model);

        void Insert(MessageInfoBindingModel model);
    }
}

```

```
}
```

Листинг 7.3 – Интерфейс IMessageInfoStorage

Класс с логикой для работы с письмами рассмотрим позднее.

Как обычно, нужно будет сделать 3 реализации. Они довольно просты. Рассмотрим на примере реализации для БД. Первым делом делаем модель (листинг 7.4).

```
using System;
using System.ComponentModel.DataAnnotations;

namespace AbstractShopDatabaseImplement.Models
{
    /// <summary>
    /// Сообщения, приходящие на почту
    /// </summary>
    public class MessageInfo
    {
        [Key]
        public string MessageId { get; set; }

        public int? ClientId { get; set; }

        public string SenderName { get; set; }

        public DateTime DateDelivery { get; set; }

        public string Subject { get; set; }

        public string Body { get; set; }

        public virtual Client Client { get; set; }
    }
}
```

Листинг 7.4 – Класс MessageInfo

Для реализации для БД есть ряд особенностей. Во-первых, ключом здесь будет выступать строка, так что ее надо будет пометить атрибутом Key, чтобы EntityFramework ее распознал. Во-вторых, письмо может быть связано с клиентом, так что пропишем идентификатор клиента в качестве поля (при добавлении и выводе оно не нужно, так что в классе ViewModel его не прописывали). У клиента, для связи, прописываем список писем. Добавляем в **AbstractShopDatabase** новый DbSet, создаем миграцию и применяем ее к БД. Реализация интерфейса будет следующей (листинг 7.5).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using AbstractShopDatabaseImplement.Models;
using System;
using System.Collections.Generic;
using System.Linq;

namespace AbstractShopDatabaseImplement.Implements
{
    public class MessageInfoStorage : IMessageInfoStorage
    {
        public List<MessageInfoViewModel> GetFullList()
        {
            using (var context = new AbstractShopDatabase())
            {
                return context.MessageInfoes
                    .Select(rec => new MessageInfoViewModel
                    {
                        MessageId = rec.MessageId,
                        SenderName = rec.SenderName,
                        DateDelivery = rec.DateDelivery,
                        Subject = rec.Subject,
                        Body = rec.Body
                    })
                    .ToList();
            }
        }

        public List<MessageInfoViewModel> GetFilteredList(MessageInfoBindingModel model)
        {
            if (model == null)
            {
                return null;
            }

            using (var context = new AbstractShopDatabase())
            {
                return context.MessageInfoes
                    .Where(rec => (model.ClientId.HasValue && rec.ClientId ==
model.ClientId) ||
                                (!model.ClientId.HasValue && rec.DateDelivery.Date ==
model.DateDelivery.Date))
                    .Select(rec => new MessageInfoViewModel
                    {
                        MessageId = rec.MessageId,
                        SenderName = rec.SenderName,
                        DateDelivery = rec.DateDelivery,
                        Subject = rec.Subject,
                        Body = rec.Body
                    })
                    .ToList();
            }
        }

        public void Insert(MessageInfoBindingModel model)
        {
            using (var context = new AbstractShopDatabase())
            {
                MessageInfo element = context.MessageInfoes.FirstOrDefault(rec =>
rec.MessageId == model.MessageId);
                if (element != null)
                {
                    throw new Exception("Уже есть письмо с таким идентификатором");
                }
            }
        }
    }
}

```

```

    }

    context.MessageInfoes.Add(new MessageInfo
    {
        MessageId = model.MessageId,
        ClientId = model.ClientId,
        SenderName = model.FromMailAddress,
        DateDelivery = model.DateDelivery,
        Subject = model.Subject,
        Body = model.Body
    });
    context.SaveChanges();
}
}
}
}
}

```

Листинг 7.5 – Класс MessageInfoStorage

Вернемся в проект **AbstractShopBusinessLogic**. Сделаем класс с логикой отправки писем и проверки почты. При отправке писем и проверки почты будут ряд данных (хост, порт, логин и пароль), которые будут идентичны при отправке каждого письма. Так что сделаем отдельный метод для получения этих данных. И отдельный метод для отправки письма (чтобы туда передавалось только адресат и текст письма). Для проверки почты потребуются свои данные (хост и порт), так что тоже выделим их передачу в отдельный класс. Получится три класса для передачи данных (листинг 7.6 – 7.8).

```

using AbstractShopBusinessLogic.Interfaces;

namespace AbstractShopBusinessLogic.HelperModels
{
    public class MailCheckInfo
    {
        public string PopHost { get; set; }

        public int PopPort { get; set; }

        public IMessageInfoLogic Logic { get; set; }
    }
}

```

Листинг 7.6 – Класс MailCheckInfo

```

namespace AbstractShopBusinessLogic.HelperModels
{
    public class MailConfig
    {
        public string SmtpClientHost { get; set; }

        public int SmtpClientPort { get; set; }

        public string MailLogin { get; set; }
    }
}

```

```

        public string MailPassword { get; set; }
    }
}

```

Листинг 7.7 – Класс MailConfig

```

namespace AbstractShopBusinessLogic.HelperModels
{
    public class MailSendInfo
    {
        public string MailAddress { get; set; }

        public string Subject { get; set; }

        public string Text { get; set; }
    }
}

```

Листинг 7.8 – Класс MailSendInfo

Сам класс с логикой (логика будет включать в себя получение писем, сохранение нового письма и методы для отправки писем и проверки поступивших писем) будет выглядеть следующим образом (листинг 7.9).

В классе часть методов будет статичными, чтобы не создавать много экземпляров класса и каждый раз передавать параметры для почты, а сделать это один раз и запомнить их. Для отправки писем будем использовать стандартный класс `SmtpClient` библиотеки `System.Net.Mail`. Передаем туда хост и порт почтового сервера, логин и пароль почты для отправки, а также адрес получателя, заголовок письма и его текст. Для чтения писем будем использовать библиотеку `MailKit`. Она позволяет довольно просто и быстро вытащить список всех непрочитанных сообщений и обработать их. Обе операции будем выполнять асинхронно, чтобы не задерживать основную работу программы. К нестатичным методам класс будут относиться метод получения списка писем из базы (всего списка или писем конкретного клиента) и метод сохранения нового письма. При этом, при сохранении будет попытка поиска клиента по адресу почты (в реализациях `IClientStorage` в методе `GetElement` дописать возможность поиска клиента по адресу почты, если этого не было сделано до этого).

```

using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.HelperModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopBusinessLogic.ViewModels;
using MailKit.Net.Pop3;

```

```

using MailKit.Security;
using System;
using System.Collections.Generic;
using System.Net;
using System.Net.Mail;
using System.Text;
using System.Threading.Tasks;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class MailLogic
    {
        private static string smtpClientHost;

        private static int smtpClientPort;

        private static string mailLogin;

        private static string mailPassword;

        private readonly IMessageInfoStorage _messageInfoStorage;

        private readonly IClientStorage _clientStorage;

        public MailLogic(IMessageInfoStorage messageInfoStorage, IClientStorage
clientStorage)
        {
            _messageInfoStorage = messageInfoStorage;
            _clientStorage = clientStorage;
        }

        public List<MessageInfoViewModel> Read(MessageInfoBindingModel model)
        {
            if (model == null)
            {
                return _messageInfoStorage.GetFullList();
            }

            return _messageInfoStorage.GetFilteredList(model);
        }

        public void CreateOrder(MessageInfoBindingModel model)
        {
            var client = _clientStorage.GetElement(new ClientBindingModel { Email =
model.FromMailAddress });
            model.ClientId = client?.Id;

            _messageInfoStorage.Insert(model);
        }

        public static void MailConfig(MailConfig config)
        {
            smtpClientHost = config.SmtpClientHost;
            smtpClientPort = config.SmtpClientPort;
            mailLogin = config.MailLogin;
            mailPassword = config.MailPassword;
        }

        public static async void MailSendAsync(MailSendInfo info)
        {
            if (string.IsNullOrEmpty(smtpClientHost) || smtpClientPort == 0)
            {
                return;
            }
        }
    }
}

```

```

        if (string.IsNullOrEmpty(mailLogin) || string.IsNullOrEmpty(mailPassword))
        {
            return;
        }

        if (string.IsNullOrEmpty(info.MailAddress) ||
string.IsNullOrEmpty(info.Subject) || string.IsNullOrEmpty(info.Text))
        {
            return;
        }

        using (var objMailMessage = new MailMessage())
        {
            using (var objSmtpClient = new SmtpClient(smtpClientHost,
smtpClientPort))
            {
                try
                {
                    objMailMessage.From = new MailAddress(mailLogin);
                    objMailMessage.To.Add(new MailAddress(info.MailAddress));
                    objMailMessage.Subject = info.Subject;
                    objMailMessage.Body = info.Text;
                    objMailMessage.SubjectEncoding = Encoding.UTF8;
                    objMailMessage.BodyEncoding = Encoding.UTF8;

                    objSmtpClient.UseDefaultCredentials = false;
                    objSmtpClient.EnableSsl = true;
                    objSmtpClient.DeliveryMethod = SmtpDeliveryMethod.Network;
                    objSmtpClient.Credentials = new NetworkCredential(mailLogin,
mailPassword);

                    await Task.Run(() => objSmtpClient.Send(objMailMessage));
                }
                catch (Exception)
                {
                    throw;
                }
            }
        }
    }

    public static async void MailCheck(MailCheckInfo info)
    {
        if (string.IsNullOrEmpty(info.PopHost) || info.PopPort == 0)
        {
            return;
        }

        if (string.IsNullOrEmpty(mailLogin) || string.IsNullOrEmpty(mailPassword))
        {
            return;
        }

        if (info.Storage == null)
        {
            return;
        }

        using (var client = new Pop3Client())
        {
            await Task.Run(() =>
            {
                try

```



```

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public class OrderLogic
    {
        private readonly object locker = new object();

        private readonly IOrderStorage _orderStorage;

        private readonly IClientStorage _clientStorage;

        public OrderLogic(IOrderStorage orderStorage, IClientStorage clientStorage)
        {
            _orderStorage = orderStorage;
            _clientStorage = clientStorage;
        }

        public List<OrderViewModel> Read(OrderBindingModel model)
        {
            if (model == null)
            {
                return _orderStorage.GetFullList();
            }

            if (model.Id.HasValue)
            {
                return new List<OrderViewModel> { _orderStorage.GetElement(model) };
            }

            return _orderStorage.GetFilteredList(model);
        }

        public void CreateOrder(CreateOrderBindingModel model)
        {
            _orderStorage.Insert(new OrderBindingModel
            {
                ClientId = model.ClientId,
                ProductId = model.ProductId,
                Count = model.Count,
                Sum = model.Sum,
                DateCreate = DateTime.Now,
                Status = OrderStatus.Принят
            });

            MailLogic.MailSendAsync(new MailSendInfo
            {
                MailAddress = _clientStorage.GetElement(new ClientBindingModel { Id =
model.ClientId })?.Email,
                Subject = $"Новый заказ",
                Text = $"Заказ от {DateTime.Now} на сумму {model.Sum:N2} принят."
            });
        }

        public void TakeOrderInWork(ChangeStatusBindingModel model)
        {
            lock (locker)
            {
                var order = _orderStorage.GetElement(new OrderBindingModel { Id =
model.OrderId });
                if (order == null)
                {
                    throw new Exception("Не найден заказ");
                }
            }
        }
    }
}

```

```

        if (order.Status != OrderStatus.Принят)
        {
            throw new Exception("Заказ не в статусе \"Принят\"");
        }
        if (order.ImplementerId.HasValue)
        {
            throw new Exception("У заказа уже есть исполнитель");
        }

        _orderStorage.Update(new OrderBindingModel
        {
            Id = order.Id,
            ClientId = order.ClientId,
            ImplementerId = model.ImplementerId,
            ProductId = order.ProductId,
            Count = order.Count,
            Sum = order.Sum,
            DateCreate = order.DateCreate,
            DateImplement = DateTime.Now,
            Status = OrderStatus.Выполняется
        });

        MailLogic.MailSendAsync(new MailSendInfo
        {
            MailAddress = _clientStorage.GetElement(new ClientBindingModel { Id =
order.ClientId })?.Email,
            Subject = $"Заказ №{order.Id}",
            Text = $"Заказ №{order.Id} передан в работу."
        });
    }
}

public void FinishOrder(ChangeStatusBindingModel model)
{
    var order = _orderStorage.GetElement(new OrderBindingModel { Id =
model.OrderId });
    if (order == null)
    {
        throw new Exception("Не найден заказ");
    }
    if (order.Status != OrderStatus.Выполняется)
    {
        throw new Exception("Заказ не в статусе \"Выполняется\"");
    }
    _orderStorage.Update(new OrderBindingModel
    {
        Id = order.Id,
        ClientId = order.ClientId,
        ImplementerId = order.ImplementerId,
        ProductId = order.ProductId,
        Count = order.Count,
        Sum = order.Sum,
        DateCreate = order.DateCreate,
        DateImplement = order.DateImplement,
        Status = OrderStatus.Готов
    });

    // Отправить письмо
}

public void PayOrder(ChangeStatusBindingModel model)
{
    var order = _orderStorage.GetElement(new OrderBindingModel { Id =

```

```

model.OrderId });
    if (order == null)
    {
        throw new Exception("Не найден заказ");
    }
    if (order.Status != OrderStatus.Готов)
    {
        throw new Exception("Заказ не в статусе \"Готов\"");
    }
    _orderStorage.Update(new OrderBindingModel
    {
        Id = order.Id,
        ClientId = order.ClientId,
        ImplementerId = order.ImplementerId,
        ProductId = order.ProductId,
        Count = order.Count,
        Sum = order.Sum,
        DateCreate = order.DateCreate,
        DateImplement = order.DateImplement,
        Status = OrderStatus.Оплачен
    });

    // Отправить письмо
}
}
}

```

Листинг 7.10 – Класс OrderLogic с логикой отправки писем

В проекте **AbstractShopView** делаем следующее:

1. В конфигурации (файл App.config) прописываем настройки для подключения к почтовому сервису (нам потребуется 2 пары значения хост-порт для отправки и получения писем и логин с паролем) (листинг 7.11).

```

<appSettings>
  <add key="SmtpClientHost" value="smtp.gmail.com" />
  <add key="SmtpClientPort" value="587" />
  <add key="PopHost" value="pop.gmail.com" />
  <add key="PopPort" value="995" />
  <add key="MailLogin" value="labwork15kafis@gmail.com" />
  <add key="MailPassword" value="passlab15" />
</appSettings>

```

Листинг 7.11 – Фрагмент файла App.config с настройками для почты

2. В классе Program пропишем логику для ввода настроек для отправки писем. Проверку почты будем осуществлять по таймеру (через определенное время), чтобы с одной стороны не вешать эту операцию на пользователя программы, а с другой, чтобы была возможность иметь актуальный список писем без перезапуска программы (листинг 7.12).

```

using AbstractShopBusinessLogic.BusinessLogics;

```

```

using AbstractShopBusinessLogic.HelperModels;
using AbstractShopBusinessLogic.Interfaces;
using AbstractShopDatabaseImplement.Implements;
using System;
using System.Configuration;
using System.Threading;
using System.Windows.Forms;
using Unity;
using Unity.Lifetime;

namespace AbstractShopView
{
    static class Program
    {
        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            var container = BuildUnityContainer();

            MailLogic.MailConfig(new MailConfig
            {
                SmtpClientHost = ConfigurationManager.AppSettings["SmtpClientHost"],
                SmtpClientPort =
Convert.ToInt32(ConfigurationManager.AppSettings["SmtpClientPort"]),
                MailLogin = ConfigurationManager.AppSettings["MailLogin"],
                MailPassword = ConfigurationManager.AppSettings["MailPassword"],
            });

            // создаем таймер
            var timer = new System.Threading.Timer(new TimerCallback(MailCheck), new
MailCheckInfo
            {
                PopHost = ConfigurationManager.AppSettings["PopHost"],
                PopPort = Convert.ToInt32(ConfigurationManager.AppSettings["PopPort"]),
                Storage = container.Resolve<IMessageInfoStorage>()
            }, 0, 100000);

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(container.Resolve<FormMain>());
        }

        private static IUnityContainer BuildUnityContainer()
        {
            var currentContainer = new UnityContainer();
            currentContainer.RegisterType<IClientStorage, ClientStorage>(new
HierarchicalLifetimeManager());
            currentContainer.RegisterType<IComponentStorage, ComponentStorage>(new
HierarchicalLifetimeManager());
            currentContainer.RegisterType<IImplementerStorage, ImplementerStorage>(new
HierarchicalLifetimeManager());
            currentContainer.RegisterType<IOrderStorage, OrderStorage>(new
HierarchicalLifetimeManager());
            currentContainer.RegisterType<IProductStorage, ProductStorage>(new
HierarchicalLifetimeManager());
            currentContainer.RegisterType<IMessageInfoStorage, MessageInfoStorage>(new
HierarchicalLifetimeManager());

            currentContainer.RegisterType<ClientLogic>(new
HierarchicalLifetimeManager());
            currentContainer.RegisterType<ComponentLogic>(new

```

```

HierarchicalLifetimeManager();
    currentContainer.RegisterType<ImplementerLogic>(new
HierarchicalLifetimeManager());
    currentContainer.RegisterType<OrderLogic>(new HierarchicalLifetimeManager());
    currentContainer.RegisterType<ProductLogic>(new
HierarchicalLifetimeManager());
    currentContainer.RegisterType<ReportLogic>(new
HierarchicalLifetimeManager());
    currentContainer.RegisterType<WorkModeling>(new
HierarchicalLifetimeManager());
    currentContainer.RegisterType<MailLogic>(new HierarchicalLifetimeManager());

    return currentContainer;
}

private static void MailCheck(object obj)
{
    MailLogic.MailCheck((MailCheckInfo)obj);
}
}
}

```

Листинг 7.12 – Класс Program

В методе BuildUnityContainer не забудем прописать новый интерфейс, его реализацию и класс с логикой. Для таймера сделаем метод, который он будет вызывать (проверка почты) и настроим периодичность вызова с интервалом в 100000 миллисекунд. Метод будет работать асинхронно, так что не скажется на работе основной программы.

3. Добавим форму для отображения всех писем. Там будет только dataGridView для вывода писем, без какого-либо дополнительного функционала. На главной форме сделать пункт меню для вызова новой формы.

В проекте **AbstractShopRestApi** в контроллере ClientController потребуется метод для получения списка писем клиента. В проекте **AbstractShowClientApp** создать представление для вывода писем клиента.

Остается сделать проверку данных при регистрации новых клиентов, либо смене их данных. Проверку можно сделать как на стороне клиента, либо в контроллере, либо в методах реализаций логики работы с клиентом. Третий вариант уже нам знаком по добавлению функционала отправки писем. Но здесь уже есть класс-обертка между клиентом и реализацией – это контроллер. Правильнее будет именно в нем настроить логику проверки. Так

как, при желании, можно отправить запрос в контроллер не используя интерфейс пользователя и обойти проверку, если она будет реализована на стороне клиента. В контроллере `ClientController` пропишем метод проверки передаваемых данных и будем его вызывать в методе регистрации клиента и смене им данных. Для проверки будем использовать регулярные выражения. Будет 2 проверки:

- в качестве логина клиент ввел адрес электронной почты;
- пароль имеет длину от 10 до 50 символов и содержит буквы, цифры и небуквенные символы.

Если данные не удовлетворяют хотя бы одному условию, то будет выкидываться ошибка (листинг 7.13).

```
using AbstractShopBusinessLogic.BindingModels;
using AbstractShopBusinessLogic.BusinessLogics;
using AbstractShopBusinessLogic.ViewModels;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;

namespace AbstractShopRestApi.Controllers
{
    [Route("api/[controller]/[action]")]
    [ApiController]
    public class ClientController : ControllerBase
    {
        private readonly ClientLogic _logic;

        private readonly MailLogic _mailLogic;

        private readonly int _passwordMaxLength = 50;

        private readonly int _passwordMinLength = 10;

        public ClientController(ClientLogic logic, MailLogic mailLogic)
        {
            _logic = logic;
            _mailLogic = mailLogic;
        }

        [HttpGet]
        public ClientViewModel Login(string login, string password) => _logic.Read(new
        ClientBindingModel { Email = login, Password = password })?[0];

        [HttpGet]
        public List<MessageInfoViewModel> GetMessages(int clientId) =>
        _mailLogic.Read(new MessageInfoBindingModel { ClientId = clientId });

        [HttpPost]
        public void Register(ClientBindingModel model)
        {
            CheckData(model);
        }
    }
}
```

```

        _logic.CreateOrUpdate(model);
    }

    [HttpPost]
    public void UpdateData(ClientBindingModel model)
    {
        CheckData(model);
        _logic.CreateOrUpdate(model);
    }

    private void CheckData(ClientBindingModel model)
    {
        if (!Regex.IsMatch(model.Email, @"регулярное выражение"))
        {
            throw new Exception("В качестве логина почта указана должна быть");
        }
        if (model.Password.Length > _passwordMaxLength || model.Password.Length <
            _passwordMinLength || !Regex.IsMatch(model.Password,
            @"^((\w+\d+\w+)|(\w+\W+\d+)|(\d+\w+\W+)|(\d+\W+\w+)|(\W+\w+\d+)|(\W+\d+\w+))[\w\d\W]*$"))
        {
            throw new Exception($"Пароль длиной от {_passwordMinLength} до
            {_passwordMaxLength} должен быть и из цифр, букв и небуквенных символов должен
            состоять");
        }
    }
}

```

Листинг 7.13 – Контроллер ClientController с проверкой данных

Для проверки работоспособности требуется клиент с почтой, к которой у вас есть доступ. На нее должны приходить сообщения, об изменениях. Также должна быть проверка, что письма от клиентов приходят и распознаются.

Требования

1. Названия моделей, полей, контроллеров, форм должны соответствовать логике вашего задания по варианту.
2. Прописать реализацию интерфейсов для писем.
3. В логике работы с заказами прописать логику для отправки писем.
4. Сделать вывод писем клиента в приложении **AbstractShowClientApp**.
5. Сделать вывод писем клиента и отправку письма клиенту при создании заказа в приложении **AbstractShopView**.
6. Вписать регулярное выражения для проверки почты.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- реализовать пагинацию для писем в проекте **AbstractShopView**;
- реализовать пагинацию для писем в проекте **AbstractShowClientApp**.

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
- 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).

ЛАБОРАТОРНАЯ РАБОТА №8.

РЕФЛЕКСИЯ

Цель

Ознакомиться с возможностями разбора классов и их методов.

Задание

1. Создать ветку от ветки седьмой лабораторной.
2. Разработать функционал для создания архива с бекапом базы данных. Разработать функционал для настройки табличного вывода данных (вывод в *dataGridView*) с помощью пользовательских атрибутов для классов *ViewModels*.
3. Вылить полученный результат в созданную ветку. Убедится, что там нет лишних файлов (типа *.exe* или *.bin*). Создать *pull request*.

Решение

Для начала займемся бекапом. Сделаем более универсальный функционал создания архива (сделаем в проекте **AbstractShopBusinessLogic**), и реализацию для создания архива с БД (в проекте **AbstractShopDatabaseImplement**). Алгоритм будет следующим: вытащить списки с данными, сохранить их в json-файлах (будем использовать сериализацию) в отдельной папке и создать архив этой папки. При сохранении будем активно использовать рефлексия. Это позволит нам избежать проблемы, если в будущем мы введем новые модели-классы (таблицы БД). За счет рефлексии просто будем вытаскивать все нужные типы из сборки и вызывать метод создания json-файла с данными. Выделим 3 метода, которые будут специфичны для реализации:

- получение сборки;
- получение списка коллекция классов-моделей;
- получение списка записей для каждого класса-модели.

Эти методы сделаем абстрактными. И сделаем 2 метода с основной логикой (листинг 8.1).

```

using System;
using System.Collections.Generic;
using System.IO;
using System.IO.Compression;
using System.Reflection;
using System.Runtime.Serialization.Json;

namespace AbstractShopBusinessLogic.BusinessLogics
{
    public abstract class BackUpAbstractLogic
    {
        public void CreateArchive(string folderName)
        {
            try
            {
                DirectoryInfo dirInfo = new DirectoryInfo(folderName);
                if (dirInfo.Exists)
                {
                    foreach (FileInfo file in dirInfo.GetFiles())
                    {
                        file.Delete();
                    }
                }

                string fileName = $"{folderName}.zip";
                if (File.Exists(fileName))
                {
                    File.Delete(fileName);
                }

                // берем сборку, чтобы от нее создавать объекты
                Assembly assem = GetAssembly();
                // вытаскиваем список классов для сохранения
                var dbsets = GetFullList();
                // берем метод для сохранения (из базового абстрактного класса)
                MethodInfo method =
                    GetType().BaseType.GetTypeInfo().GetDeclaredMethod("SaveToFile");
                foreach (var set in dbsets)
                {
                    // создаем объект из класса для сохранения
                    var elem =
                        assem.CreateInstance(set.PropertyType.GenericTypeArguments[0].FullName);
                    // генерируем метод, исходя из класса
                    MethodInfo generic = method.MakeGenericMethod(elem.GetType());
                    // вызываем метод на выполнение
                    generic.Invoke(this, new object[] { folderName });
                }

                // архивируем
                ZipFile.CreateFromDirectory(folderName, fileName);

                // удаляем папку
                dirInfo.Delete(true);
            }
            catch (Exception)
            {
                // делаем проброс
                throw;
            }
        }

        private void SaveToFile<T>(string folderName) where T : class, new()
        {
            var records = GetList<T>();

```

```

        T obj = new T();
        DataContractJsonSerializer jsonFormatter = new
DataContractJsonSerializer(typeof(List<T>));

        using (FileStream fs = new FileStream(string.Format("{0}/{1}.json",
folderName, obj.GetType().Name), FileMode.OpenOrCreate))
        {
            jsonFormatter.WriteObject(fs, records);
        }
    }

    protected abstract Assembly GetAssembly();

    protected abstract List<PropertyInfo> GetFullList();

    protected abstract List<T> GetList<T>() where T : class, new();
}

```

Листинг 8.1 – Абстрактный класс BackupAbstractLogic

Создадим наследника от класса BackupAbstractLogic в проекте **AbstractShopDatabaseImplement** (листинг 8.2).

В проекте **AbstractShopView** в классе Program в UnityContainer прописываем связку BackupAbstractLogic и BackupLogic. На главной форме в меню добавляем пункт «Создать бекап». В логике создаем диалог для выбора папки для сохранения и вызываем метод создания бекапа (листинг 8.3).

```

using AbstractShopBusinessLogic.BusinessLogics;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Reflection;

namespace AbstractShopDatabaseImplement.Implements
{
    public class BackupLogic : BackupAbstractLogic
    {
        protected override Assembly GetAssembly()
        {
            return typeof(BackupLogic).Assembly;
        }

        protected override List<PropertyInfo> GetFullList()
        {
            using (var context = new AbstractShopDatabase())
            {
                Type type = context.GetType();
                return type.GetProperties().Where(x =>
x.PropertyType.FullName.StartsWith("Microsoft.EntityFrameworkCore.DbSet")).ToList();
            }
        }

        protected override List<T> GetList<T>()
        {
            using (var context = new AbstractShopDatabase())

```

```

        {
            return context.Set<T>().ToList();
        }
    }
}

```

Листинг 8.2 – Класс BackUpLogic

```

private void СоздатьБекапToolStripMenuItem_Click(object sender, EventArgs e)
{
    try
    {
        if (_backUpAbstractLogic != null)
        {
            var fbd = new FolderBrowserDialog();
            if (fbd.ShowDialog() == DialogResult.OK)
            {
                _backUpAbstractLogic.CreateArchive(fbd.SelectedPath);
                MessageBox.Show("Бекап создан", "Сообщение",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
    }
}

```

Листинг 8.3 – Логика метода создатьБекапToolStripMenuItem_Click

Для конфигурации табличного вывода в проекте **AbstractShopBusinessLogic** создадим класс-атрибут для конфигурации колонки. Выделим интересующие нас данные:

- отображать или скрывать колонку;
- заголовок колонки;
- ширина колонки;
- авторазмер колонки (делать по ширине).

Для авторазмера потребуется перечисление. Создадим перечисление GridViewAutoSize. Для заполнения поступим просто, откроем перечисление DataGridViewAutoSizeColumnMode и скопируем оттуда значения (листинг 8.4).

```

namespace AbstractShopBusinessLogic.Attributes
{
    public enum GridViewAutoSize
    {
        NotSet = 0,

        None = 1,
    }
}

```

```

        ColumnHeader = 2,

        AllCellsExceptHeader = 4,

        AllCells = 6,

        DisplayedCellsExceptHeader = 8,

        DisplayedCells = 10,

        Fill = 16
    }
}

```

Листинг 8.4 – Перечисление GridViewAutoSize

Создадим класс и унаследуем его от класса Attribute (листинг 8.5).

```

using System;

namespace AbstractShopBusinessLogic.Attributes
{
    public class ColumnAttribute : Attribute
    {
        public ColumnAttribute(string title = "", bool visible = true, int width = 0,
GridViewAutoSize gridViewAutoSize = GridViewAutoSize.None)
        {
            Title = title;
            Visible = visible;
            Width = width;
            GridViewAutoSize = gridViewAutoSize;
        }

        public string Title { get; private set; }

        public bool Visible { get; private set; }

        public int Width { get; private set; }

        public GridViewAutoSize GridViewAutoSize { get; private set; }
    }
}

```

Листинг 8.5 – Класс ColumnAttribute

Так как помимо настроек самих колонок через свойства класса может быть еще важен порядок вывода свойств в таблице. Для этого у каждого класса нужно будет прописать метод возврата списка свойств.

Для всех основных view-моделей (за исключением тех, что используются для отчетов) зададим для требуемых полей атрибуты (листинг 8.6).

```

using AbstractShopBusinessLogic.Attributes;
using AbstractShopBusinessLogic.Enums;
using System;
using System.Runtime.Serialization;

namespace AbstractShopBusinessLogic.ViewModels
{

```

```

/// <summary>
/// Заказ
/// </summary>
[DataContract]
public class OrderViewModel
{
    [Column(title: "Номер", width: 100)]
    [DataMember]
    public int Id { get; set; }

    [DataMember]
    public int ClientId { get; set; }

    [DataMember]
    public int ProductId { get; set; }

    [DataMember]
    public int? ImplementerId { get; set; }

    [Column(title: "Клиент", width: 150)]
    [DataMember]
    public string ClientFIO { get; set; }

    [Column(title: "Изделие", gridViewAutoSize: GridViewAutoSize.Fill)]
    [DataMember]
    public string ProductName { get; set; }

    [Column(title: "Исполнитель", width: 150)]
    [DataMember]
    public string ImplementerFIO { get; set; }

    [Column(title: "Количество", width: 100)]
    [DataMember]
    public int Count { get; set; }

    [Column(title: "Сумма", width: 50)]
    [DataMember]
    public decimal Sum { get; set; }

    [Column(title: "Статус", width: 100)]
    [DataMember]
    public OrderStatus Status { get; set; }

    [Column(title: "Дата создания", width: 100)]
    [DataMember]
    public DateTime DateCreate { get; set; }

    [Column(title: "Дата выполнения", width: 100)]
    [DataMember]
    public DateTime? DateImplement { get; set; }
}
}

```

Листинг 8.6 – Обновленный класс OrderViewModel

Нужно определиться с местом, где прописывать логику настройки dataGridView на основе атрибутов модели. Правильнее будет создать отдельный класс, где прописать метод. Но для простоты пропишем его в классе Program. Метод будет состоять из 2-х частей. В первой части будем

настраивать колонки грида, а во второй заполнять его данными. На вход будем передавать список данных и грид, который нужно настроить и заполнить (листинг 8.7).

```
public static void ConfigGrid<T>(List<T> data, DataGridView grid)
{
    var type = typeof(T);

    var config = new List<string>();
    grid.Columns.Clear();
    foreach (var prop in type.GetProperties())
    {
        // получаем список атрибутов
        var attributes =
prop.GetCustomAttributes(typeof(ColumnAttribute), true);
        if (attributes != null && attributes.Length > 0)
        {
            foreach (var attr in attributes)
            {
                // ищем нужный нам атрибут
                if (attr is ColumnAttribute columnAttr)
                {
                    config.Add(prop.Name);
                    var column = new DataGridViewTextBoxColumn
                    {
                        Name = prop.Name,
                        ReadOnly = true,
                        HeaderText = columnAttr.Title,
                        Visible = columnAttr.Visible,
                        Width = columnAttr.Width
                    };
                    if (columnAttr.GridViewAutoSize !=
GridViewAutoSize.None)
                    {
                        column.AutoSizeMode =
(DataGridViewAutoSizeColumnMode)Enum.Parse(typeof(DataGridViewAutoSizeColumnMode),
columnAttr.GridViewAutoSize.ToString());
                    }
                    grid.Columns.Add(column);
                }
            }
        }

        // добавляем строки
        foreach (var elem in data)
        {
            List<object> objs = new List<object>();
            foreach (var conf in config)
            {
                var value =
elem.GetType().GetProperty(conf).GetValue(elem);

                objs.Add(value);
            }

            grid.Rows.Add(objs.ToArray());
        }
    }
}
```

Листинг 8.7 – Метод ConfigGrid класса Program

Остается переделать вывод табличных данных в логиках форм (листинг 8.8).

```
private void LoadData()
{
    try
    {
        Program.ConfigGrid(logic.Read(null), dataGridView);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Ошибка", MessageBoxButtons.OK,
        MessageBoxIcon.Error);
    }
}
```

Листинг 8.8 – Обновленный метод для вывода списка клиентов

Требования

1. Названия моделей, полей, контроллеров, форм должны соответствовать логике вашего задания по варианту.
2. Бекап сохранять в формате xml.
3. Во view-моделях прописать настройки для таблиц.
4. Переделать вывод всех табличных форм в проекте **AbstractShopView**.

Усложненная лабораторная (необязательно)

Для усложненного варианта требуется:

- сделать вызов методов класса ReportLogic через рефлексия;
- дополнить настройки колонок, чтобы был форматированный вывод данных (например, для дат).

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 мороженого и добавки (орехи, шоколад и т.п.). Изделия – мороженное (icescream).
- 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).

Отчет по лабораторным работам

Отчет включает в себя:

- Титульный лист (см приложение А).
- Описание основного приложения:
 - назначение (тут ваше творчество, исходя из варианта)
 - как работать (добавление, редактирование, удаление) с каждым из справочников (какие пол зачем нужны, со скринами);
 - путь заказа (от создания до «Оплачен»).
- Описание клиентского приложения.
 - как регистрироваться в системе и входить в систему;
 - как менять учетные данные;
 - как работать с заказами (создание, обновление, письма на почте).
- Опционально:
 - в основном приложение указать про склады (добавление, редактирование, удаление), если реализовывали;
 - при работе с заказами указать возможность нехватки материалов и как это решается (пополнение складов), если реализовывали;
 - описать складское приложение, если реализовывали.
- Листинг кода. 8 шрифт 2 колонки

Список использованных источников

1. Pro Git. 2nd Edition [Электронный ресурс] / Режим доступа: <https://git-scm.com/book/ru/v2>.
2. METANIT.COM. Сайт о программировании [Электронный ресурс] / Режим доступа: <https://metanit.com/sharp/>. – Загл. с экрана.
3. ProfessorWeb. .Net & Web Programming [Электронный ресурс] / Режим доступа: <https://professorweb.ru/>. – Загл. с экрана.
4. Tiberiu Covaci, Rod Stephens, Vincent Varallo, Gerry O'Brien. MCSD Certification Toolkit (Exam 70-483) // Published by John Wiley & Sons, Inc. – 2013. – 656p.
5. MCTS Self-Paced Training Kit (Exam 70-536): Microsoft .NET Framework–Application Development Foundation, Second Edition eBook // Published by Microsoft Press. – 2009. – 829 p.
6. Браузер API .NET [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/api/>. – Загл. с экрана.

Приложение А

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ИСТ
Кафедра «Информационные системы»
Дисциплина «Технологии программирования»

ОТЧЕТ ПО ЛАБОРАТОРНЫМ РАБОТАМ «АБСТРАКТНЫЙ МАГАЗИН (Указать по своему варианту)»

выполнил(а): студент(ка)
гр. ИСЭ(ПИ)бд-21
И.О. Фамилия

проверил: ст. преподаватель
Е.Н. Эгов

Ульяновск
2000 г.

Учебное издание

ЭГОВ Евгений Николаевич

РАЗРАБОТКА ПРОГРАММЫ АБСТРАКТНОГО МАГАЗИНА

практикум по дисциплине
«Технологии программирования»

Редактор Н.А. Евдокимова

Подписано в печать 00.00.2019 Формат 60×84/16

Усл. печ. л. 12,5 Заказ № 0000

Ульяновский государственный технический университет

432027, г. Ульяновск, ул. Сев. Венец, д. 32.

ИПК «Венец» УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, д. 32.