







Лабораторная работа №3

Содержание

1	ЗАДАНИЕ 	3
1.1	ВАРИАНТЫ ЗАДАНИЙ 	4
1.2	ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ 	8
2	ШАГИ ПО СОЗДАНИЮ ПРИЛОЖЕНИЯ 	9
2.1	ШАГ 1: СОЗДАНИЕ ПРОЕКТА	9
2.2	ШАГ 2: ДОБАВЛЕНИЕ ФАЙЛОВ	11
2.2.1	СОЗДАНИЕ ПАПКИ Models	11
2.2.2	КЛАССЫ: PRODUCT.CS, ORDER.CS, ORDERITEM.CS	11
2.2.3	ДОБАВЛЕНИЕ КОНВЕРТЕРА	12
2.3	ШАГ 3: РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА	14
2.4	ШАГ 4: РЕАЛИЗАЦИЯ ЛОГИКИ	15
2.4.1	КОНСТРУКТОР	16
2.4.2	РАБОТА С ТОВАРАМИ (ДОБАВЛЕНИЕ, ОБНОВЛЕНИЯ, УДАЛЕНИЕ)	17
2.4.3	РАБОТА С ЗАКАЗАМИ (ДОБАВЛЕНИЕ, ОБНОВЛЕНИЕ, УДАЛЕНИЕ)	21
2.4.4	ИЗМЕНЕНИЕ КОЛИЧЕСТВА ТОВАРОВ (КНОПКИ <<+>> И <<->>)	26
2.4.5	СБРОС ВЫДЕЛЕНИЯ В СПИСКАХ	27
3	ДЕМОНСТРАЦИЯ РАБОТЫ 	29
3.1	ДОБАВЛЕНИЕ ТОВАРОВ	29
3.2	ДОБАВЛЕНИЕ ТОВАРОВ В ЗАКАЗ	30
3.3	СОЗДАНИЕ ЗАКАЗА	31
3.4	ПРОСМОТР СОДЕРЖИМОГО ЗАКАЗА	32
3.5	УДАЛЕНИЕ ТОВАРОВ ИЗ ЗАКАЗА	33
3.6	ОБНОВЛЕНИЕ И УДАЛЕНИЕ ЗАКАЗОВ	34
4	ВОПРОСЫ ДЛЯ ЗАЩИТЫ 	39

Листинг


1	Класс Order.cs	11
2	Класс Order.cs	12
3	Класс Product.cs	12
4	Класс BooleanToVisibilityConverter	13
5	Пример разметки главного окна MainWindow	14
6	Пример кода для класса MainWindow	15
7	Пример кода для конструктора класса MainWindow	16
8	Методы обновления данных	17
9	Метод добавления нового товара	18
10	Метод обновления товара	19
11	Метод удаления товара	20
12	Методы редактирования товара и добавления в список выбранных товаров для заказа	21
13	Метод добавления заказа	22
14	Метод обновления заказа	23
15	Метод удаления заказа	24
16	Метод удаления товара из заказа	25
17	Методы управления количеством товара в заказе	26
18	Метод изменения выделения в списке заказов OrdersList_SelectionChanged	27
19	Методы очистки полей и сброса выделения товара	28

Лабораторная работа №3


1 ЗАДАНИЕ

Система управления заказами для онлайн-магазина:

1. Реализовать CRUD-приложение (WPF)¹.
2. Реализовать визуальное отображение списка товаров и списка заказов.
3. Реализовать функционал добавления, редактирования, удаления конкретных товаров, а также учёт их количества. Реализовать функции уменьшения количества товаров при добавлении их в заказ.
4. Реализовать функционал добавления, редактирования, удаления заказов, а также возможность редактирования количества товаров, добавленных в заказ.
5. В списке заказов отобразить информацию о названии, дате (времени) заказа, общей сумме заказа и список входящих в него товаров и их количества.
6. Реализовать функцию расчёта остатков товара (при уменьшении количества товара в заказе, должно увеличиваться количество товара на складе (возвращаться). При удалении товара из заказа также количество остатков должно увеличиваться (если было куплено 6 единиц товара и заказ удален (отменен), на складе должно стать на 6 единиц товара больше (вернуться))).

Весь код доступен в репозитории на GitHub ²

¹CRUD-приложение – это программа, которая позволяет вам выполнять четыре основных действия с данными (например, товарами или заказами): Создавать новые записи, Читать (просматривать) существующие, Обновлять (редактировать) их и Удалять ненужные.

²Ссылки, выделенные голубым цветом, ведут на внешние ресурсы; ссылки, выделенные светло-синим цветом, указывают на исходный код в репозитории GitHub .

Лабораторная работа №3

1.1 ВАРИАНТЫ ЗАДАНИЙ

1. Музыкальный магазин

- Товары: гитары, синтезаторы, барабаны (поля: Name, Price, Stock, Id)
- Заказы: имя клиента, дата, список инструментов
- Создать раздел «Инструменты», добавить поле «Тип» (например, струнные, клавишные)

2. Книжный магазин

- Товары: книги (поля: Name – название, Price, Stock, Id)
- Заказы: имя клиента, дата, список книг
- Создать раздел «Книги», добавить поле «Автор» в Product

3. Магазин электроники

- Товары: смартфоны, ноутбуки, наушники
- Заказы: имя клиента, дата, список гаджетов
- Создать раздел «Гаджеты», добавить поле «Бренд» (например, Apple, Samsung)

4. Спортивный магазин

- Товары: кроссовки, тренажёры, мячи
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Спортивный инвентарь», добавить поле «Категория» (обувь, оборудование)

5. Магазин одежды

- Товары: футболки, джинсы, куртки
- Заказы: имя клиента, дата, список одежды
- Создать раздел «Одежда», добавить поле «Размер» (S, M, L)

6. Цветочный магазин

- Товары: розы, тюльпаны, орхидеи.
- Заказы: имя клиента, дата, список букетов
- Создать раздел «Цветы», добавить поле «Тип» (букет, горшок)

7. Магазин игрушек

- Товары: конструкторы, куклы, машинки.
- Заказы: имя клиента, дата, список игрушек.
- Создать раздел «Игрушки», добавить поле «Возраст».

8. Магазин бытовой техники

- Товары: холодильники, пылесосы, микроволновки.
- Заказы: имя клиента, дата, список техники.
- Создать раздел «Техника», добавить поле «Мощность»

9. Магазин косметики

- Товары: помады, кремы, духи
- Заказы: имя клиента, дата, список косметики
- Создать раздел «Косметика», добавить поле «Тип» (уход, макияж)

10. Магазин автозапчастей

- Товары: фильтры, шины, аккумуляторы
- Заказы: имя клиента, дата, список запчастей
- Создать раздел «Запчасти», добавить поле «Марка» (Toyota, BMW)

11. Магазин мебели

- Товары: диваны, столы, шкафы
- Заказы: имя клиента, дата, список мебели
- Создать раздел «Мебель», добавить поле «Материал» (дерево, металл)

12. Магазин канцелярии

- Товары: ручки, тетради, маркеры
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Канцелярия», добавить поле «Тип» (письмо, рисование)

13. Магазин ювелирных изделий

- Товары: кольца, серьги, браслеты
- Заказы: имя клиента, дата, список украшений
- Создать раздел «Украшения», добавить поле «Материал» (золото, серебро)

14. Магазин зоотоваров

- Товары: корма, игрушки, клетки
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Зоотовары», добавить поле «Животное» (кошка, собака)

15. Магазин видеоигр

- Товары: игры для ПК, консолей
- Заказы: имя клиента, дата, список игр
- Создать раздел «Игры», добавить поле «Платформа» (PC, PS5)

16. Магазин садовых товаров

- Товары: семена, инструменты, горшки
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Садовые товары», добавить поле «Тип» (растения, инструменты)

17. Магазин часов

- Товары: наручные часы, настенные часы
- Заказы: имя клиента, дата, список часов
- Создать раздел «Часы», добавить поле «Механизм» (кварцевый, механический)

18. Магазин обуви

- Товары: кроссовки, ботинки, туфли
- Заказы: имя клиента, дата, список обуви
- Создать раздел «Обувь», добавить поле «Размер» (36, 42)

19. Магазин настольных игр

- Товары: шахматы, монополия, карточные игры
- Заказы: имя клиента, дата, список игр
- Создать раздел «Игры», добавить поле «Игроков» (2-4, 4-8)

20. Магазин сувениров

- Товары: магниты, статуэтки, открытки
- Заказы: имя клиента, дата, список сувениров
- Создать раздел «Сувениры», добавить поле «Страна» (Россия, Италия)

21. Магазин парфюмерии

- Товары: духи, туалетная вода
- Заказы: имя клиента, дата, список парфюма
- Изменения: переименовать «Товары» в «Парфюм», добавить поле «Объём» (50 мл, 100 мл)

22. Магазин строительных материалов

- Товары: краска, гвозди, доски
- Заказы: имя клиента, дата, список материалов
- Создать раздел «Материалы», добавить поле «Единица» (литры, кг)

23. Магазин для художников

- Товары: краски, кисти, холсты
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Арт-товары», добавить поле «Тип» (масло, акварель)

24. Магазин чая и кофе

- Товары: чай, кофе, сиропы
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Напитки», добавить поле «Вес» (100 г, 500 г)

25. Магазин для рыбалки

- Товары: удочки, приманки, катушки
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Снаряжение», добавить поле «Тип» (спиннинг, фидер)

26. Магазин для кемпинга

- Товары: палатки, спальники, горелки
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Снаряжение», добавить поле «Вес» (кг)

27. Магазин медицинских товаров

- Товары: тонометры, бинты, маски
- Заказы: имя клиента, дата, список товаров
- Создать раздел «Медтовары», добавить поле «Назначение» (диагностика, уход)

28. Магазин фильмов

- Товары: DVD, Blu-ray диски
- Заказы: имя клиента, дата, список фильмов
- Создать раздел «Фильмы», добавить поле «Жанр» (драма, комедия)

29. Магазин горнолыжного оборудования

- Товары: сноуборд, горнолыжные костюмы, шлемы
- Заказы: имя клиента, дата, список оборудования
- Создать раздел «Снаряжение», добавить поле «Вид» (сноуборд, лыжи)

1.2 ДОПОЛНИТЕЛЬНОЕ ЗАДАНИЕ ³



Реализовать удаление единственного товара в заказе. Если в заказе присутствует только один товар, то при его удалении должен удаляться и сам заказ. Количество данного товара должно возвращаться в остаток.



Добавить функционал снятия выделения товара (в списке выбранных для добавления в заказ) и снятие выделения с заказа при клике по пустому полю (внутри listBox).

³Дополнительные задания выполняются по желанию и не подлежат проверке на зачёте.

2 ШАГИ ПО СОЗДАНИЮ ПРИЛОЖЕНИЯ

2.1 ШАГ 1: СОЗДАНИЕ ПРОЕКТА

Для создания проекта необходимо выбрать:

1. Создать проект
2. Приложение WPF (Майкрософт) (Рисунок 1)

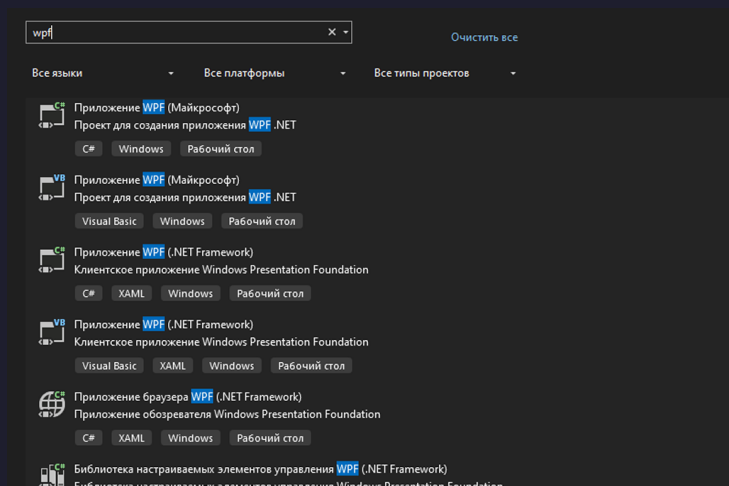


Рис. 1: Создание проекта

3. Далее задать имя проекта: StoreManager (Рисунок 2)

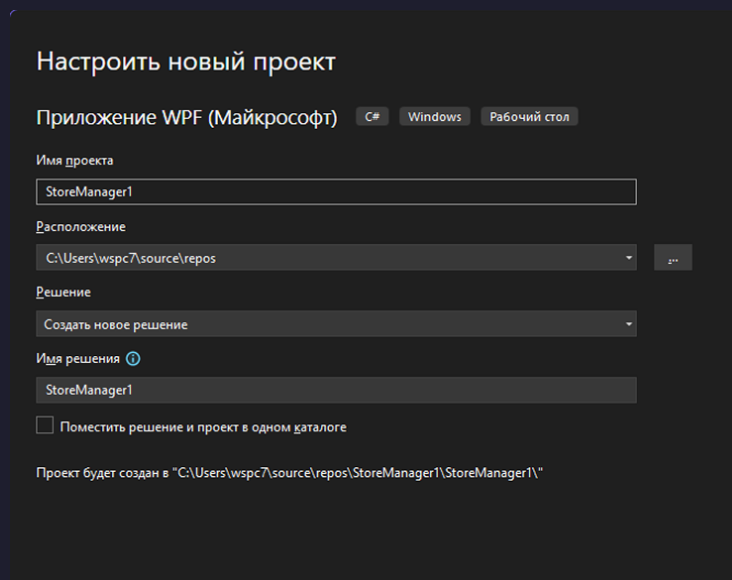


Рис. 2: Задание имени проекта

4. Выбрать платформу (пример выполнен на .net 8.0) (Рисунок 3)

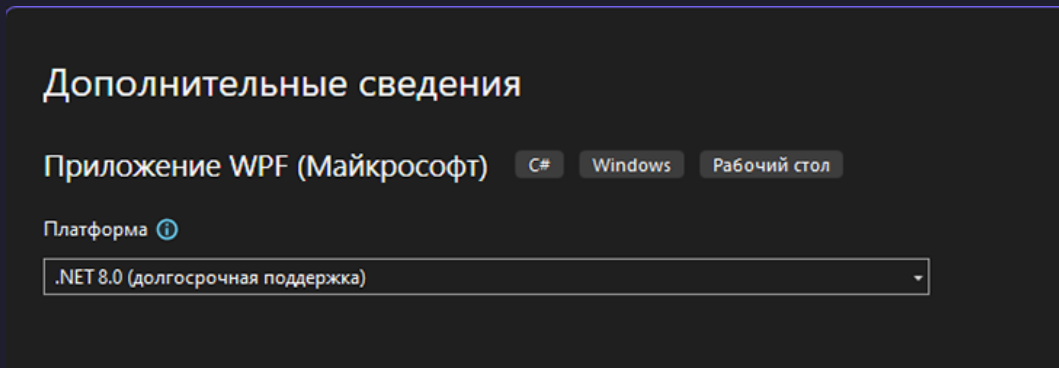


Рис. 3: Выбор платформы.

Структура проекта состоит из следующих файлов:

- `App.xaml` – конфигурация приложения. Определяет основной класс приложения, наследуемый от `Application`. В нём задаются стили, шаблоны приложения и стартовая точка⁴.
- `App.xaml.cs` – логика приложения. Содержит класс `App`, который может переопределять методы, такие как `OnStartup`, для настройки поведения при запуске, или обрабатывать события приложения.
- `MainWindow.xaml` – разметка и структура UI (интерфейса) главного окна.
- `MainWindow.xaml.cs` – логика и обработчики событий окна.

⁴Обычно указывается начальное окно в свойстве `StartupUri`

2.2 ШАГ 2: ДОБАВЛЕНИЕ ФАЙЛОВ

Необходимо добавить следующие файлы и папки:

- В корне проекта создать папку `Models`
- В папке `Models` добавить два новых файла классов: `Product.cs` и `Order.cs`.

Код для файла `Order.cs` приведён в листинге 1

```
1 namespace StoreManager.Models
2 {
3     // Модель заказа, содержащая информацию о клиенте и товарах
4     public class Order
5     {
6         // Уникальный идентификатор заказа
7         public int Id { get; set; }
8
9         // Список элементов заказа (товар + количество)
10        public List<OrderItem> Items { get; set; }
11
12        // Имя клиента
13        public string CustomerName { get; set; }
14
15        // Дата создания заказа
16        public DateTime OrderDate { get; set; }
17
18        // Общая сумма заказа, вычисляемая как сумма цен товаров умноженная на их количество
19        public decimal TotalPrice => Items.Sum(item => item.Product.Price * item.Quantity);
20
21        // Конструктор, инициализирующий пустой список товаров
22        public Order()
23        {
24            Items = new List<OrderItem>();
25        }
26    }
27 }
```

Листинг 1: Класс `Order.cs`

Данный файл отвечает за создание нового объекта заказа. Может содержать уникальные поля, не такие как в примере (в соответствии с вариантом).

Также необходим класс, позволяющий связать товар и его количество в заказе. Пример созданного класса `OrderItem` и объявления его свойств приведён в листинге 2

```
1 namespace StoreManager.Models
2 {
3     // Модель элемента заказа, связывающая товар и его количество
4     public class OrderItem
5     {
6         // Ссылка на товар
7         public Product Product { get; set; }
8
9         // Количество единиц товара в заказе
10        public int Quantity { get; set; }
11    }
12 }
```

Листинг 2: Класс Order.cs

Также необходимо создать файл `Product.cs`. Данный файл отвечает за создание нового объекта товара. Пример класса `Product` приведён в листинге 3.

```
1 namespace StoreManager.Models
2 {
3     // Модель товара, представляющая продукт в магазине
4     public class Product
5     {
6         // Уникальный идентификатор товара
7         public int Id { get; set; }
8
9         // Название товара
10        public string Name { get; set; }
11
12        // Цена товара
13        public decimal Price { get; set; }
14
15        // Количество товара на складе
16        public int Stock { get; set; }
17    }
18 }
```

Листинг 3: Класс Product.cs

Также необходимо создать папку `Converters`. Она будет нужна для создания конвертера. В данной папке нужно создать класс `BooleanToVisibilityConverter`. Пример кода для данного класса приведён в листинге 4.

```
1 using System;
2 using System.Globalization;
3 using System.Windows;
4 using System.Windows.Data;
5
6 namespace StoreManager.Converters
7 {
8     // Конвертер для отображения placeholder'ов в TextBox
9     public class BooleanToVisibilityConverter : IValueConverter
10     {
11         // Преобразует boolean (IsEmpty) в Visibility для TextBox
12         public object Convert(object value, Type targetType, object parameter, CultureInfo culture)
13         {
14             return value is bool isEmpty && isEmpty ? Visibility.Visible : Visibility.Collapsed;
15         }
16
17         // Обратное преобразование не используется
18         public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)
19         {
20             throw new NotImplementedException();
21         }
22     }
23 }
```

Листинг 4: Класс BooleanToVisibilityConverter

Этот класс реализует интерфейс `IValueConverter`, который используется для преобразования данных при привязке (`Data Binding`)

- Метод `Convert` преобразует значение типа `bool` в `Visibility` (перечисление WPF для управления видимостью элементов)
- `true` → `Visible` (элемент виден)
- `false` → `Collapsed` (элемент скрыт, не занимает место)
- Метод `ConvertBack` не реализован, так как обратное преобразование не требуется.

Конвертеры в WPF используются для преобразования данных между источником (моделью) и целью (элементом UI). Например, конвертер позволяет адаптировать значение свойства модели к формату, подходящему для отображения. В данном случае конвертер применяется для реализации эффекта `placeholder` (подсказки в текстовых полях), показывая текст, когда поле пустое (`Text.IsEmpty`).

Свойство `Visibility` в WPF имеет три значения:

- `Visible` – элемент отображается
- `Collapsed` – элемент скрыт и не занимает места
- `Hidden` – элемент скрыт, но занимает место в макете.

2.3 ШАГ 3: РЕАЛИЗАЦИЯ ИНТЕРФЕЙСА

Для того, чтобы реализовать внешний вид приложения, необходимо добавить соответствующую разметку в файл `MainWindow.xaml`. Пример разметки для данного файла представлен ниже в листинге 5

```
8      <!-- Ресурсы окна: конвертеры и стили -->
9      <Window.Resources>
10         <!-- Конвертер для преобразования bool в Visibility -->
11         <converters:BooleanToVisibilityConverter x:Key="BooleanToVisibilityConverter"/>
12
13         <!-- Общий стиль кнопок -->
14         <Style TargetType="Button">
15             <Setter Property="Background" Value="#FF6200EE"/>
16             <Setter Property="Foreground" Value="White"/>
17             <Setter Property="FontSize" Value="14"/>
18             <Setter Property="Padding" Value="10,5"/>
19             <Setter Property="Margin" Value="5"/>
20             <Setter Property="BorderThickness" Value="0"/>
21             <Setter Property="Cursor" Value="Hand"/>
22         </Style>
23
24         <!-- Специальный стиль кнопок изменения количества -->
25         <Style x:Key="QuantityButtonStyle" TargetType="Button">
26             <Setter Property="Background" Value="#FF6200EE"/>
27             <Setter Property="Foreground" Value="White"/>
28             <Setter Property="FontSize" Value="12"/>
29             <Setter Property="Width" Value="25"/>
30             <Setter Property="Height" Value="25"/>
31             <Setter Property="Margin" Value="5,0"/>
32             <Setter Property="BorderThickness" Value="0"/>
33             <Setter Property="Cursor" Value="Hand"/>
34         </Style>
35
36         <!-- Стиль для полей ввода -->
37         <Style TargetType="TextBox">
38             <Setter Property="FontSize" Value="14"/>
39             <Setter Property="Padding" Value="5"/>
40             <Setter Property="Margin" Value="5"/>
41             <Setter Property="BorderBrush" Value="#FFCCCCC"/>
42         </Style>
43
44         <!-- Стиль текста-заглушки (placeholder) -->
45         <Style TargetType="TextBlock" x:Key="PlaceholderStyle">
46             <Setter Property="Foreground" Value="Gray"/>
47             <Setter Property="FontStyle" Value="Italic"/>
48             <Setter Property="Margin" Value="8,5,0,0"/>
49             <Setter Property="IsHitTestVisible" Value="False"/>
50         </Style>
51     </Window.Resources>
```

Листинг 5: Пример разметки главного окна `MainWindow`

2.4 ШАГ 4: РЕАЛИЗАЦИЯ ЛОГИКИ

Файл `MainWindow.xaml.cs` – это `code-behind` для главного окна приложения. Он содержит логику взаимодействия пользовательского интерфейса (UI) с данными, обрабатывает события (например, нажатия кнопок) и управляет состоянием приложения. Приложение представляет собой систему управления магазином, где пользователь может.

Структура кода состоит из следующих элементов:

1. Объявления класса и полей – определение данных, используемых в приложении.
2. Конструктора – инициализация окна и начальных данных.
3. Методов обновления UI – синхронизация списков товаров и заказов с UI.
4. Обработчиков событий – реакция на действия пользователя (нажатия кнопок, выбор элементов).
5. Вспомогательных методов – очистка полей ввода.

В листинге 6 представлен пример кода для класса `MainWindow`.

```
1 using StoreManager.Models;
2 using System.Windows;
3 using System.Windows.Controls;
4 using System.Windows.Input;
5 using System.Windows.Media;
6
7 namespace StoreManager
8 {
9     // Главное окно приложения для управления товарами и заказами
10    public partial class MainWindow : Window
11    {
12        // Список всех товаров
13        private List<Product> products = new List<Product>();
14        // Список всех заказов
15        private List<Order> orders = new List<Order>();
16        // Следующий ID для нового товара
17        private int nextProductId = 1;
18        // Следующий ID для нового заказа
19        private int nextOrderId = 1;
20        // Список выбранных товаров для текущего заказа
21        private List<OrderItem> selectedProductsForOrder = new List<OrderItem>();
```

Листинг 6: Пример кода для класса `MainWindow`

Класс `MainWindow` наследуется от `Window`, так как это главное окно WPF-приложения. Ключевое слово `partial` указывает, что класс разделён между `MainWindow.xaml.cs` (логика) и `MainWindow.xaml` (разметка, сгенерированная часть).

Поля:

- `products`: Хранит список всех товаров (экземпляры класса `Product`). Это основная коллекция для управления ассортиментом магазина.
- `orders`: Хранит список заказов (экземпляры класса `Order`). Каждый заказ содержит имя клиента, дату и список товаров.
- `nextProductId`: Счётчик для генерации уникальных идентификаторов товаров.

Начинается с 1 и увеличивается при добавлении нового товара:

- `nextOrderId`: Аналогичный счётчик для заказов.
- `selectedProductsForOrder`: Временное хранилище товаров (`OrderItem`), которые пользователь выбрал при формировании нового заказа.

Эти поля представляют состояние приложения. Они хранят данные в памяти (вместо базы данных) и используются для отображения в UI и обработки пользовательских действий. Использование `List<T>` означает, что данные не обновляют UI автоматически (в отличие от `ObservableCollection`), поэтому код вручную обновляет списки.

Далее необходимо объявить основной метод (конструктор) `MainWindow`. Пример кода для данного метода представлен в листинге 7

```
24     public MainWindow()  
25     {  
26         InitializeComponent(); // Инициализация UI  
27         UpdateProductList();   // Обновление списка товаров  
28         UpdateOrderList();     // Обновление списка заказов  
29         UpdateSelectedProductsList(); // Обновление списка выбранных товаров  
30     }
```

Листинг 7: Пример кода для конструктора класса `MainWindow`

`InitializeComponent()` – метод, автоматически сгенерированный WPF, который загружает разметку из `MainWindow.xaml`, инициализирует элементы интерфейса (такие как кнопки, списки и текстовые поля) и устанавливает необходимые привязки. Без его вызова пользовательский интерфейс не будет отображён.

Вызовы методов обновления:

- `UpdateProductList()` – устанавливает `ItemsSource` для `ListBox` с товарами, чтобы отобразить начальный список (пустой на старте).
- `UpdateOrderList()` – аналогично предыдущему методу, только для списка заказов (`OrdersList`).
- `UpdateSelectedProductsList()` – обновляет список выбранных товаров для заказа (`SelectedProductsList`).

Назначение:

- Конструктор подготавливает приложение к работе, загружая UI и синхронизируя данные с элементами управления.
- Поскольку списки изначально пусты, вызовы обновления предотвращают ошибки отображения.

Методы для обновления списков товаров, заказов и выбранных товаров для заказа представлены в листинге 8

```
32     // Обновляет отображение списка товаров
33     private void UpdateProductList()
34     {
35         ProductsList.ItemsSource = null;
36         ProductsList.ItemsSource = products;
37     }
38
39     // Обновляет отображение списка заказов
40     private void UpdateOrderList()
41     {
42         OrdersList.ItemsSource = null;
43         OrdersList.ItemsSource = orders;
44     }
45
46     // Обновляет отображение списка выбранных товаров
47     private void UpdateSelectedProductsList()
48     {
49         SelectedProductsList.ItemsSource = null;
50         SelectedProductsList.ItemsSource = selectedProductsForOrder;
51     }
```

Листинг 8: Методы обновления данных

Метод `UpdateProductList` сбрасывает `ItemsSource` в `null`, чтобы очистить привязку. Устанавливает `ItemsSource` в `products`, чтобы `ListBox` (`ProductsList`) отобразил текущий список товаров. Остальные методы работают аналогично. `Null` используется для предотвращения проблемы с кэшированием данных в WPF, обеспечивая корректное обновление интерфейса.

Далее рассмотрим методы для работы с конкретными товарами. Эти методы реагируют на действия пользователя с товарами (кнопки «Добавить», «Обновить», «Удалить», выбор в списке). Метод добавления нового товара представлен в листинге 9

```
53 // Обработчик нажатия кнопки "Добавить товар"
54 private void AddProduct_Click(object sender, RoutedEventArgs e)
55 {
56     // Проверка корректности введенных данных
57     if (string.IsNullOrEmpty(ProductName.Text) ||
58         !decimal.TryParse(ProductPrice.Text, out decimal price) ||
59         !int.TryParse(ProductStock.Text, out int stock) ||
60         stock < 0)
61     {
62         MessageBox.Show("Пожалуйста, введите корректные данные о товаре. Остаток не может быть отрицательным.",
63             "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
64
65         return;
66     }
67
68     // Создание нового товара
69     var product = new Product
70     {
71         Id = nextProductId++,
72         Name = ProductName.Text,
73         Price = price,
74         Stock = stock
75     };
76
77     products.Add(product); // Добавление товара в список
78     UpdateProductList();   // Обновление UI
79     ClearProductFields();  // Очистка полей ввода
80 }
```

Листинг 9: Метод добавления нового товара

Данный метод проверяет валидность ввода:

- `ProductName.Text` не пустое.
- Проверяет `ProductPrice.Text` (что цена корректна (`decimal`)), количество – неотрицательное число (`int`).
- `ProductStock.Text` преобразуется в `int`.

Если валидация не пройдена, показывает сообщение об ошибке. Создает новый объект `Product` с уникальным `Id` (из `nextProductId`), именем, ценой и запасом из полей ввода. Добавляет товар в список `products`, обновляет UI (`UpdateProductList`) и очищает поля ввода (`ClearProductFields`).

Следующий метод позволяет обновить (редактировать) информацию о конкретном товаре. Реализация метода обновления представлена в листинге 10

```
82 // Обработчик нажатия кнопки "Обновить товар"
83 private void UpdateProduct_Click(object sender, RoutedEventArgs e)
84 {
85     if (ProductsList.SelectedItem is Product selectedProduct)
86     {
87         // Проверка корректности введенных данных
88         if (string.IsNullOrWhiteSpace(ProductName.Text) ||
89             !decimal.TryParse(ProductPrice.Text, out decimal price) ||
90             !int.TryParse(ProductStock.Text, out int stock) ||
91             stock < 0)
92         {
93             MessageBox.Show("Пожалуйста, введите корректные данные о товаре. Остаток не может быть отрицательным.",
94                             "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
95             return;
96         }
97
98         // Обновление свойств выбранного товара
99         selectedProduct.Name = ProductName.Text;
100        selectedProduct.Price = price;
101        selectedProduct.Stock = stock;
102
103        UpdateProductList(); // Обновление UI
104        ClearProductFields(); // Очистка полей ввода
105    }
106    else
107    {
108        MessageBox.Show("Пожалуйста, выберите товар для обновления.", "Ошибка", MessageBoxButton.OK,
109                        ↪ MessageBoxImage.Warning);
110    }
111 }
```

Листинг 10: Метод обновления товара

Данный метод проверяет, выбран ли товар в списке `ProductsList`.

- Проводит валидацию ввода (аналогично методу `AddProduct_Click`).
- Если товар выбран и данные валидны, обновляет свойства выбранного объекта `Product` (`Name`, `Price`, `Stock`).
- Обновляет UI и очищает поля.
- Если товар не выбран, показывает предупреждение.

Для того, чтобы исключить конкретный товар из общего списка, используется метод `DeleteProduct` из листинга 11.

```
112 // Обработчик нажатия кнопки "Удалить товар"
113 private void DeleteProduct_Click(object sender, RoutedEventArgs e)
114 {
115     if (ProductsList.SelectedItem is Product selectedProduct)
116     {
117         // Проверка, используется ли товар в заказах
118         if (orders.Any(order => order.Items.Any(item => item.Product.Id == selectedProduct.Id)))
119         {
120             MessageBox.Show("Нельзя удалить товар, используемый в заказах.", "Ошибка", MessageBoxButton.OK,
121                 ↪ MessageBoxImage.Warning);
122             return;
123         }
124         // Подтверждение удаления
125         var result = MessageBox.Show($"Вы уверены, что хотите удалить товар '{selectedProduct.Name}'?",
126             "Подтверждение удаления", MessageBoxButton.YesNo, MessageBoxImage.Question);
127
128         if (result == MessageBoxResult.Yes)
129         {
130             selectedProductsForOrder.RemoveAll(item => item.Product.Id == selectedProduct.Id); // Удаление из
131                 ↪ текущего заказа
132             products.Remove(selectedProduct); // Удаление из списка товаров
133             UpdateProductList();
134             UpdateSelectedProductsList();
135             ClearProductFields();
136         }
137     }
138     else
139     {
140         MessageBox.Show("Пожалуйста, выберите товар для удаления.", "Ошибка", MessageBoxButton.OK,
141             ↪ MessageBoxImage.Warning);
142     }
143 }
```

Листинг 11: Метод удаления товара

Данный метод выполняет следующие функции:

- Удаляет товар из списка, если он не используется в заказах.
- Если товар выбран и не связан с заказами, запрашивает подтверждение и удаляет его.
- Также удаляет его из текущего заказа (если он там есть).
- Если товар используется в заказах – показывает предупреждение.
- Обновляет UI и очищает поля.

```
143 // Обработчик изменения выделения в списке товаров
144 private void ProductsList_SelectionChanged(object sender, SelectionChangedEventArgs e)
145 {
146     if (ProductsList.SelectedItem is Product selectedProduct)
147     {
148         // Заполнение полей ввода данными выбранного товара
149         ProductName.Text = selectedProduct.Name;
150         ProductPrice.Text = selectedProduct.Price.ToString();
151         ProductStock.Text = selectedProduct.Stock.ToString();
152     }
153 }
154
155 private void AddToOrder_Click(object sender, RoutedEventArgs e)
156 {
157     if (ProductsList.SelectedItem is Product selectedProduct)
158     {
159         if (selectedProduct.Stock <= 0)
160         {
161             MessageBox.Show("Товара больше нет на складе.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);
162             return;
163         }
164
165         var existingItem = selectedProductsForOrder.FirstOrDefault(item => item.Product.Id == selectedProduct.Id);
166
167         if (existingItem != null)
168         {
169             existingItem.Quantity++;
170         }
171         else
172         {
173             selectedProductsForOrder.Add(new OrderItem { Product = selectedProduct, Quantity = 1 });
174         }
175
176         selectedProduct.Stock--; // <=== уменьшение запаса
177         UpdateProductList();
178         UpdateSelectedProductsList();
179     }
180     else
181     {
182         MessageBox.Show("Пожалуйста, выберите товар для добавления в заказ.", "Ошибка", MessageBoxButton.OK,
183             ↪ MessageBoxImage.Warning);
184     }
185 }
```

Листинг 12: Методы редактирования товара и добавления в список выбранных товаров для заказа

Далее рассмотрим методы `ProductsList_SelectionChanged` и `AddToOrder`. Когда пользователь выбирает товар в списке `ProductsList`, метод `ProductsList_SelectionChanged` заполняет поля `ProductName`, `ProductPrice`, `ProductStock` данными выбранного товара. Используется также для редактирования информации о товаре (листинг 12).

Метод `AddToOrder` добавляет выбранный товар из списка в текущий заказ. Если товар уже добавлен – увеличивает его количество. Если нет – создаёт новую позицию. Также уменьшает остаток товара на складе. Обновляет UI. Если товар не выбран или нет в наличии – показывает предупреждение.

Далее рассмотрим метод создания нового заказа. Пример кода для данного метода представлен

в листинге 13.

```
186 // Обработчик нажатия кнопки "Создать заказ"
187 private void AddOrder_Click(object sender, RoutedEventArgs e)
188 {
189     // Проверка имени клиента
190     if (string.IsNullOrWhiteSpace(CustomerName.Text))
191     {
192         MessageBox.Show("Пожалуйста, введите имя клиента.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
193         return;
194     }
195
196     // Проверка наличия товаров в заказе
197     if (!selectedProductsForOrder.Any())
198     {
199         MessageBox.Show("Пожалуйста, выберите хотя бы один товар для заказа.", "Ошибка", MessageBoxButton.OK,
200             ↪ MessageBoxImage.Warning);
201         return;
202     }
203
204     // Создание нового заказа
205     var order = new Order
206     {
207         Id = nextOrderId++,
208         CustomerName = CustomerName.Text,
209         OrderDate = DateTime.Now,
210         Items = new List<OrderItem>(selectedProductsForOrder)
211     };
212
213     orders.Add(order); // Добавление заказа в список
214     UpdateProductList();
215     UpdateOrderList();
216     ClearOrderFields(); // Очистка полей заказа
217 }
```

Листинг 13: Метод добавления заказа

Данный метод выполняет следующие функции:

- Создает новый заказ.
- Проверяет, введено ли имя клиента и есть ли хотя бы один товар в заказе.
- Если всё в порядке – формирует объект `Order`, копирует список товаров и добавляет заказ в список `orders`.
- Обновляет UI и очищает поля.
- Если что-то не заполнено – показывает ошибку.

Следующий метод позволяет обновить информацию о заказе. Пример кода для данного метода представлен в листинге 14.

```
218 // Обработчик нажатия кнопки "Обновить заказ"
219 private void UpdateOrder_Click(object sender, RoutedEventArgs e)
220 {
221     if (OrdersList.SelectedItem is Order selectedOrder)
222     {
223         // Проверка имени клиента
224         if (string.IsNullOrWhiteSpace(CustomerName.Text))
225         {
226             MessageBox.Show("Пожалуйста, введите имя клиента.", "Ошибка", MessageBoxButton.OK,
227                 ↪ MessageBoxImage.Error);
228             return;
229         }
230         // Проверка наличия товаров
231         if (!selectedProductsForOrder.Any())
232         {
233             MessageBox.Show("Пожалуйста, выберите хотя бы один товар для заказа.", "Ошибка", MessageBoxButton.OK,
234                 ↪ MessageBoxImage.Warning);
235             return;
236         }
237         // Обновление заказа
238         selectedOrder.CustomerName = CustomerName.Text;
239         selectedOrder.OrderDate = DateTime.Now;
240         selectedOrder.Items.Clear();
241         selectedOrder.Items.AddRange(selectedProductsForOrder);
242
243         // Проверка доступности перед применением изменений
244         foreach (var item in selectedProductsForOrder)
245         {
246             int availableStock = item.Product.Stock +
247                 selectedOrder.Items.Where(i => i.Product.Id == item.Product.Id).Sum(i => i.Quantity); // учитываем
248                 ↪ старый заказ
249
250             if (item.Quantity > availableStock)
251             {
252                 MessageBox.Show($"Недостаточно товара '{item.Product.Name}' на складе для обновления заказа.",
253                     "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);
254                 return;
255             }
256         }
257
258         UpdateProductList();
259         UpdateOrderList();
260         ClearOrderFields();
261     }
262     else
263     {
264         MessageBox.Show("Пожалуйста, выберите заказ для обновления.", "Ошибка", MessageBoxButton.OK,
265             ↪ MessageBoxImage.Warning);
266     }
267 }
```

Листинг 14: Метод обновления заказа

- Обновляет выбранный заказ.
- Проверяет имя клиента и наличие товаров в заказе.
- Если данные валидны – проверяет, есть ли достаточный остаток на складе для всех добавляе-

мых/увеличиваемых товаров.

- Если проверки пройдены, обновляет имя клиента, дату и товары в заказе, корректируя запасы на складе (возвращая старые и забирая новые).
- Обновляет UI и очищает поля.
- Если что-то не так – показывает сообщение об ошибке.

Следующий метод позволяет удалить выбранный заказ. Пример кода для данного метода представлен в листинге 15.

```
268 // Обработчик нажатия кнопки "Удалить заказ"
269 private void DeleteOrder_Click(object sender, RoutedEventArgs e)
270 {
271     if (OrdersList.SelectedItem is Order selectedOrder)
272     {
273         // Подтверждение удаления
274         var result = MessageBox.Show($"Вы уверены, что хотите удалить заказ для клиента
↪ '{selectedOrder.CustomerName}'?",
                "Подтверждение удаления", MessageBoxButton.YesNo, MessageBoxImage.Question);
275
276         if (result == MessageBoxResult.Yes)
277         {
278             // Возврат запасов
279             foreach (var item in selectedOrder.Items)
280             {
281                 item.Product.Stock += item.Quantity;
282             }
283
284             orders.Remove(selectedOrder); // Удаление заказа
285             UpdateProductList();
286             UpdateOrderList();
287             ClearOrderFields();
288         }
289     }
290     else
291     {
292         MessageBox.Show("Пожалуйста, выберите заказ для удаления.", "Ошибка", MessageBoxButton.OK,
↪ MessageBoxImage.Warning);
293     }
294 }
295 }
```

Листинг 15: Метод удаления заказа

Следующий метод нужен для удаления товара из заказа. Код для данного метода представлен в листинге 16.


```
297 // Обработчик нажатия кнопки "Удалить товар из заказа"
298 private void RemoveProductFromOrder_Click(object sender, RoutedEventArgs e)
299 {
300     if (SelectedProductsList.SelectedItem is OrderItem selectedItem)
301     {
302         // Проверка: это последний товар в заказе?
303         if (selectedProductsForOrder.Count == 1 && OrdersList.SelectedItem is Order selectedOrder)
304         {
305             var result = MessageBox.Show(
306                 $"Удаление последнего товара приведёт к удалению заказа клиента '{selectedOrder.CustomerName}'.",
307                 "Подтверждение удаления",
308                 MessageBoxButton.YesNo,
309                 MessageBoxImage.Warning);
310
311             if (result != MessageBoxResult.Yes)
312                 return;
313
314             // Удаляем товар и возвращаем остаток
315             selectedProductsForOrder.Remove(selectedItem);
316             selectedItem.Product.Stock += selectedItem.Quantity;
317
318             // Удаляем сам заказ
319             orders.Remove(selectedOrder);
320             UpdateProductList();
321             UpdateOrderList();
322             ClearOrderFields();
323         }
324         else
325         {
326             // Обычное удаление товара
327             selectedProductsForOrder.Remove(selectedItem);
328             selectedItem.Product.Stock += selectedItem.Quantity;
329
330             UpdateProductList();
331             UpdateSelectedProductsList();
332         }
333     }
334     else
335     {
336         MessageBox.Show("Пожалуйста, выберите товар для удаления из заказа.", "Ошибка", MessageBoxButton.OK,
337             "↪ MessageBoxImage.Warning");
338     }
339 }
```

Листинг 16: Метод удаления товара из заказа

- Метод `RemoveProductFromOrder` удаляет товар из текущего заказа (`selectedProductsForOrder`).
- Если количество выбранного `OrderItem` больше 1 – уменьшает его.
- Если товар выбран – полностью удаляет позицию (`OrderItem`) из списка `selectedProductsForOrder`.
- Возвращает количество товара на склад.
- Если это был единственный товар в редактируемом заказе (`OrdersList.SelectedItem`), запрашивает подтверждение и удаляет сам заказ.

- Обновляет UI.
- Если товар не выбран – показывает предупреждение.

Далее рассмотрим методы, отвечающие за увеличение и уменьшение количества товара в заказе (кнопки «+» и «-»). Пример этих методов представлен в листинге 17.

```
342 // Обработчик нажатия кнопки "+" для увеличения количества
343 private void IncreaseQuantity_Click(object sender, RoutedEventArgs e)
344 {
345     if (sender is Button button && button.Tag is OrderItem item)
346     {
347         if (item.Product.Stock < 1)
348         {
349             MessageBox.Show("Товара больше нет на складе.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Warning);
350             return;
351         }
352
353         item.Quantity++;
354         item.Product.Stock--; // резерв
355         UpdateProductList();
356         UpdateSelectedProductsList();
357     }
358 }
359
360 // Обработчик нажатия кнопки "-" для уменьшения количества
361 private void DecreaseQuantity_Click(object sender, RoutedEventArgs e)
362 {
363     if (sender is Button button && button.Tag is OrderItem item)
364     {
365         if (item.Quantity > 1)
366         {
367             item.Quantity--; // Уменьшение количества
368         }
369         else
370         {
371             selectedProductsForOrder.Remove(item); // Удаление товара
372         }
373         item.Product.Stock++;
374         UpdateProductList();
375         UpdateSelectedProductsList();
376     }
377 }
```

Листинг 17: Методы управления количеством товара в заказе

Следующий метод позволяет выделить определённый заказ и изменить имя заказчика. Реализация метода представлена в листинге 18.

```
379 // Обработчик изменения выделения в списке заказов
380 private void OrdersList_SelectionChanged(object sender, SelectionChangedEventArgs e)
381 {
382     if (OrdersList.SelectedItem is Order selectedOrder)
383     {
384         // Заполнение полей данными выбранного заказа
385         CustomerName.Text = selectedOrder.CustomerName;
386         selectedProductsForOrder.Clear();
387         selectedProductsForOrder.AddRange(selectedOrder.Items);
388         UpdateSelectedProductsList();
389     }
390 }
```

Листинг 18: Метод изменения выделения в списке заказов `OrdersList_SelectionChanged`

- Срабатывает при выборе заказа из списка.
- Заполняет поле имени клиента.
- Загружает список товаров из заказа в текущий список `selectedProductsForOrder`.
- Обновляет UI.

Далее рассмотрим методы очищения полей `ClearProductFields` и `ClearOrderFields`. Они позволяют очистить текстовые поля для добавления и редактирования товара и снять выделение в списке товаров. `ClearOrderFields` очищает поле имени клиента и список выбранных товаров, снимает выделение в списке заказов, при этом не трогает список товаров (оставляя выделение). Листинг этих методов приведён в листинге 19.

```
379 // Обработчик изменения выделения в списке заказов
380 private void OrdersList_SelectionChanged(object sender, SelectionChangedEventArgs e)
381 {
382     if (OrdersList.SelectedItem is Order selectedOrder)
383     {
384         // Заполнение полей данными выбранного заказа
385         CustomerName.Text = selectedOrder.CustomerName;
386         selectedProductsForOrder.Clear();
387         selectedProductsForOrder.AddRange(selectedOrder.Items);
388         UpdateSelectedProductsList();
389     }
390 }
391
392 // Очистка полей ввода для товаров
393 private void ClearProductFields()
394 {
395     ProductName.Text = "";
396     ProductPrice.Text = "";
397     ProductStock.Text = "";
398     ProductsList.SelectedItem = null; // Сброс выделения
399 }
400
401 // Очистка полей ввода для заказов
402 private void ClearOrderFields()
403 {
404     CustomerName.Text = "";
405     selectedProductsForOrder.Clear();
406     UpdateSelectedProductsList();
407     OrdersList.SelectedItem = null; // Сброс выделения
408     // Выделение в ProductsList не сбрасывается
409 }
410
411 // Обработчик клика мышью по списку товаров
412 private void ProductsList_MouseLeftButtonDown(object sender, MouseButtonEventArgs e)
413 {
414     var listBox = sender as ListBox;
415     var hitTestResult = VisualTreeHelper.HitTest(listBox, e.GetPosition(listBox));
416     if (hitTestResult != null)
417     {
418         // Проверка, попал ли клик на ListBoxItem
419         var element = hitTestResult.VisualHit;
420         while (element != null && !(element is ListBoxItem))
421         {
422             element = VisualTreeHelper.GetParent(element);
423         }
424
425         if (element == null)
426         {
427             listBox.SelectedItem = null; // Сброс выделения при клике по пустому месту
428         }
429     }
430 }
431 }
432 }
```

Листинг 19: Методы очистки полей и сброса выделения товара

Метод `ProductsList_MouseLeftButtonDown` используется, если пользователь кликнул мышью по пустому месту в списке товаров. В этом случае метод сбрасывает выделение. Применяется для удобства работы – чтобы можно было ”отменить”выбор.

3 ДЕМОНСТРАЦИЯ РАБОТЫ 63

3.1 ДОБАВЛЕНИЕ ТОВАРОВ

В итоговом варианте внешний вид приложения представлен на Рисунке 4

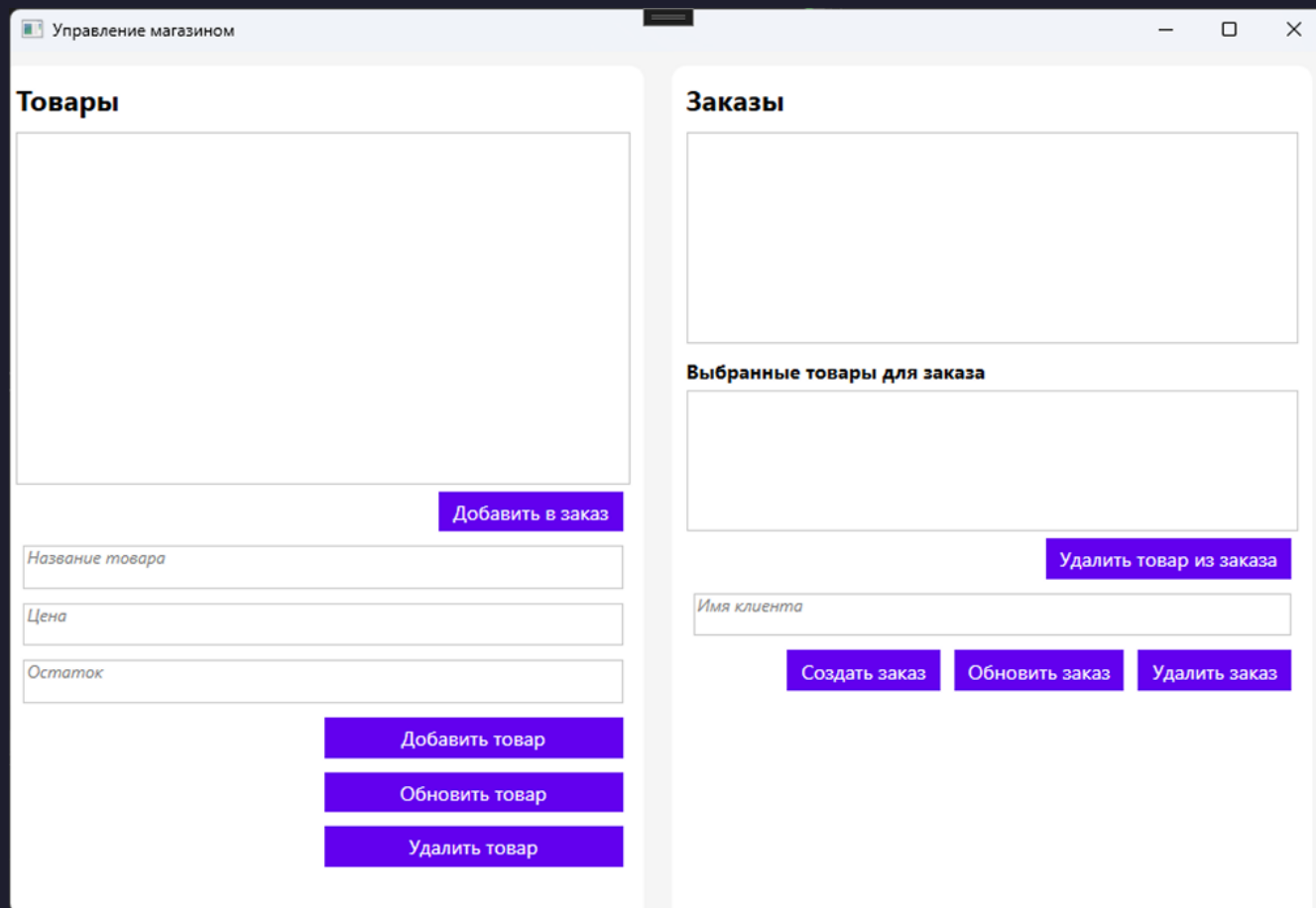


Рис. 4: Интерфейс приложения

3.2 ДОБАВЛЕНИЕ ТОВАРОВ В ЗАКАЗ

В качестве примера отображения информации в приложении, введены данные о товарах и заказе. Пример интерфейса приложения с введенными данными приведен на Рисунке 5

Управление магазином

Товары

Добавить в заказ

Хлеб

100

20

Добавить товар

Обновить товар

Удалить товар

Заказы

Удалить товар из заказа

Имя клиента

Создать заказ

Обновить заказ

Удалить заказ

Рис. 5: Пример заполнения информации в приложении

3.3 СОЗДАНИЕ ЗАКАЗА

Далее по нажатию кнопки ”Добавить товар” товар добавляется в общий список. Таким образом можно добавить группу товаров (Рисунок 6).

Управление магазином

Товары

Хлеб \$5.00 (Остаток: 80)
 Торт \$50.00 (Остаток: 40)
 Морковь \$1.50 (Остаток: 200)
 Яблоки \$0.40 (Остаток: 100)
 Творог \$2.00 (Остаток: 120)
 Пицца \$45.00 (Остаток: 30)

Добавить в заказ

Название товара

Цена

Остаток

Добавить товар

Обновить товар

Удалить товар

Заказы

Выбранные товары для заказа

Имя клиента

Создать заказ

Обновить заказ

Удалить заказ

Рис. 6: Пример добавления группы товаров

3.4 ПРОСМОТР СОДЕРЖИМОГО ЗАКАЗА

Далее необходимо выбрать конкретный товар из списка, по нажатию кнопки добавить его в список выбранных товаров. В данном списке с помощью кнопки «+» и «-» можно отредактировать количество товаров для добавления в заказ (Рисунок 7).

Управление магазином

Товары

Хлеб \$5.00 (Остаток: 79)
 Торт \$50.00 (Остаток: 39)
 Морковь \$1.50 (Остаток: 200)
 Яблоки \$0.40 (Остаток: 99)
 Творог \$2.00 (Остаток: 119)
 Пицца \$45.00 (Остаток: 30)

Добавить в заказ

Яблоки

0,4

100

Добавить товар
 Обновить товар
 Удалить товар

Заказы

Выбранные товары для заказа

Хлеб	\$5.00 x 1	+	-
Торт	\$50.00 x 1	+	-
Творог	\$2.00 x 1	+	-

Удалить товар из заказа

Имя клиента

Создать заказ Обновить заказ Удалить заказ

Рис. 7: Пример добавления товаров в заказ

3.5 УДАЛЕНИЕ ТОВАРОВ ИЗ ЗАКАЗА

Далее необходимо ввести имя клиента (название заказа) и нажать кнопку «Создать заказ» для добавления нового заказа в список заказов (Рисунок 8).

Управление магазином

Товары

Хлеб \$5.00 (Остаток: 79)
 Торт \$50.00 (Остаток: 38)
 Морковь \$1.50 (Остаток: 200)
 Яблоки \$0.40 (Остаток: 88)
 Творог \$2.00 (Остаток: 114)
 Пицца \$45.00 (Остаток: 30)

Заказы

Иван
 16.04.2025 15:15
 Итого: \$121.80
 Хлеб \$5.00 x 1
 Торт \$50.00 x 2
 Творог \$2.00 x 6
 Яблоки \$0.40 x 12

Выбранные товары для заказа

Яблоки

0,4

100

Добавить в заказ

Удалить товар из заказа

Имя клиента

Создать заказ **Обновить заказ** **Удалить заказ**

Добавить товар
Обновить товар
Удалить товар

Рис. 8: Пример создания заказа

3.6 ОБНОВЛЕНИЕ И УДАЛЕНИЕ ЗАКАЗОВ

При нажатии на конкретный заказ можно вывести список товаров (Рисунок 9).

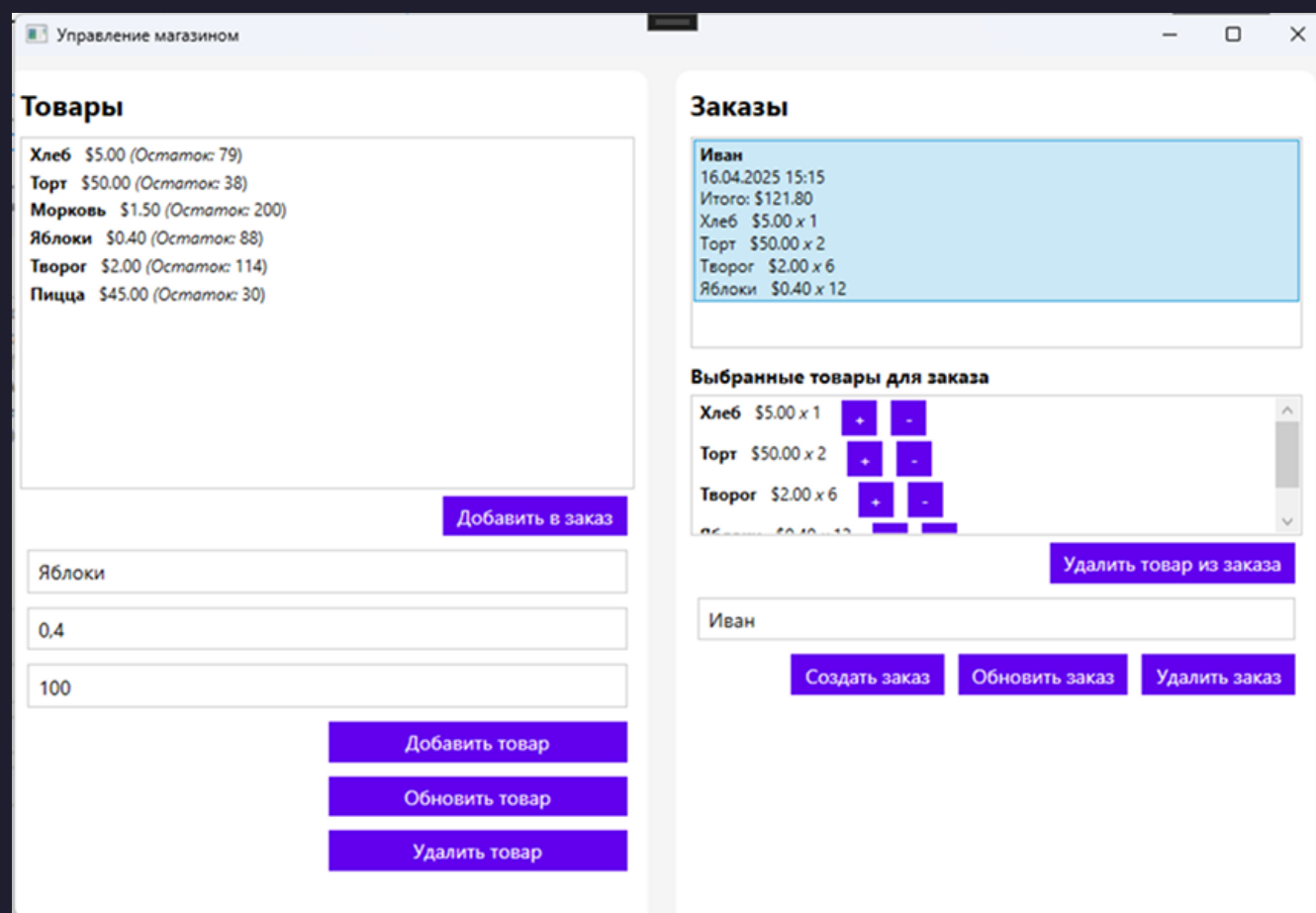


Рис. 9: Вывод списка товаров в заказе

При нажатии кнопки «Удалить товар из заказа» можно удалить конкретный товар, добавленный в заказ (Рисунок 10).

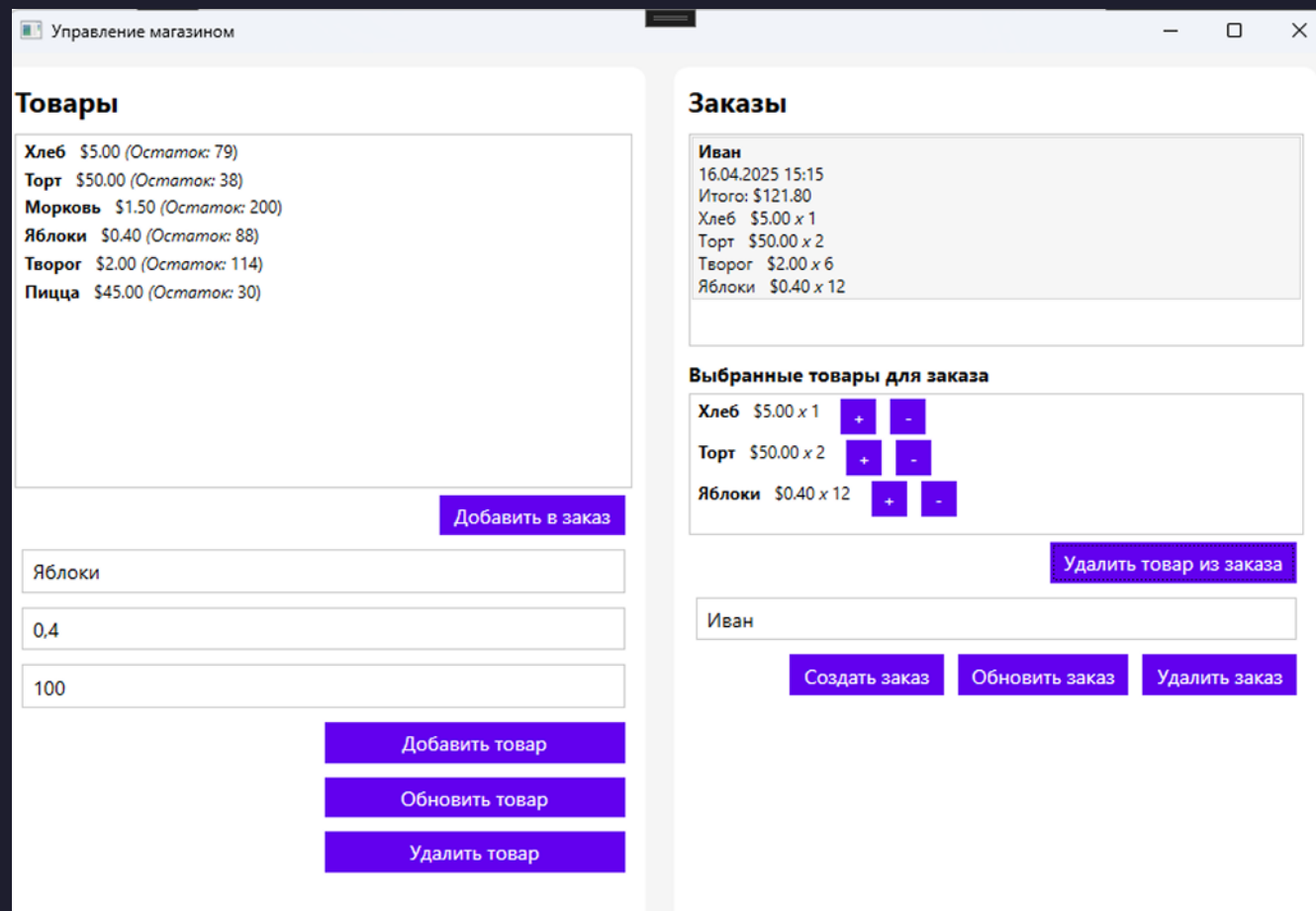


Рис. 10: Удаление товара из заказа

Кнопка «Обновить заказ» позволяет обновить информацию о заказе и удалить конкретный товар, добавленный в заказ (Рисунок 11).

Управление магазином

Товары

Хлеб	\$5.00	(Остаток: 79)
Торт	\$50.00	(Остаток: 38)
Морковь	\$1.50	(Остаток: 200)
Яблоки	\$0.40	(Остаток: 88)
Творог	\$2.00	(Остаток: 114)
Пицца	\$45.00	(Остаток: 30)

Добавить в заказ

Яблоки

0,4

100

Добавить товар

Обновить товар

Удалить товар

Заказы

Иван
16.04.2025 15:20
Итого: \$109.80
Хлеб \$5.00 x 1
Торт \$50.00 x 2
Яблоки \$0.40 x 12

Выбранные товары для заказа

Удалить товар из заказа

Имя клиента

Создать заказ Обновить заказ Удалить заказ

Рис. 11: Обновление списка товаров в заказе

При нажатии кнопки «Удалить заказ» можно удалить конкретный заказ из списка (Рисунок 12). При удалении на экран выводится сообщение с подтверждением. После удаления количество остатков товара увеличивается в соответствии с тем, сколько товара было в заказе.

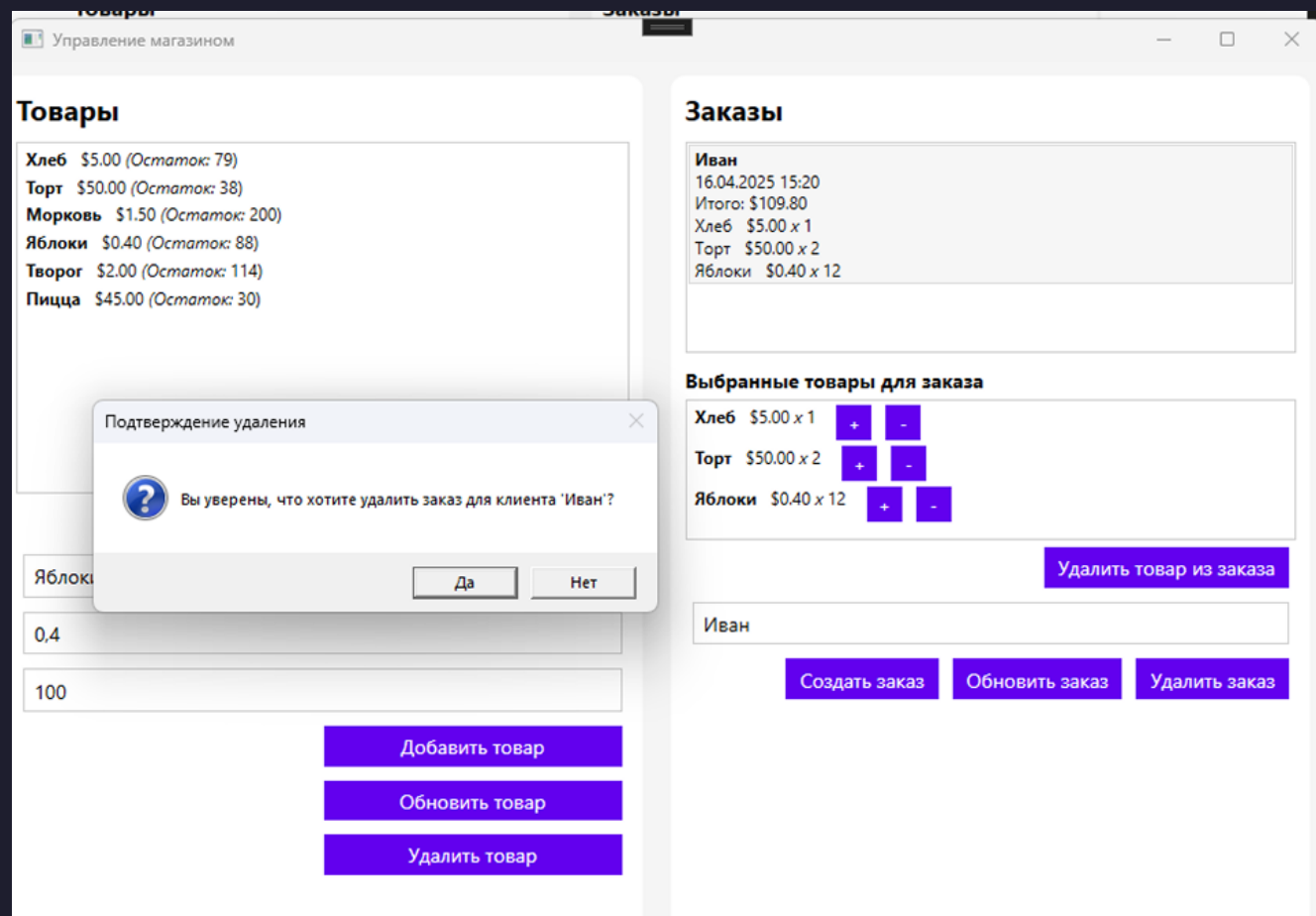


Рис. 12: Удаление заказа

При нажатии кнопки «Удалить товар из заказа» можно удалить конкретный товар, добавленный в заказ (Рисунок 13).

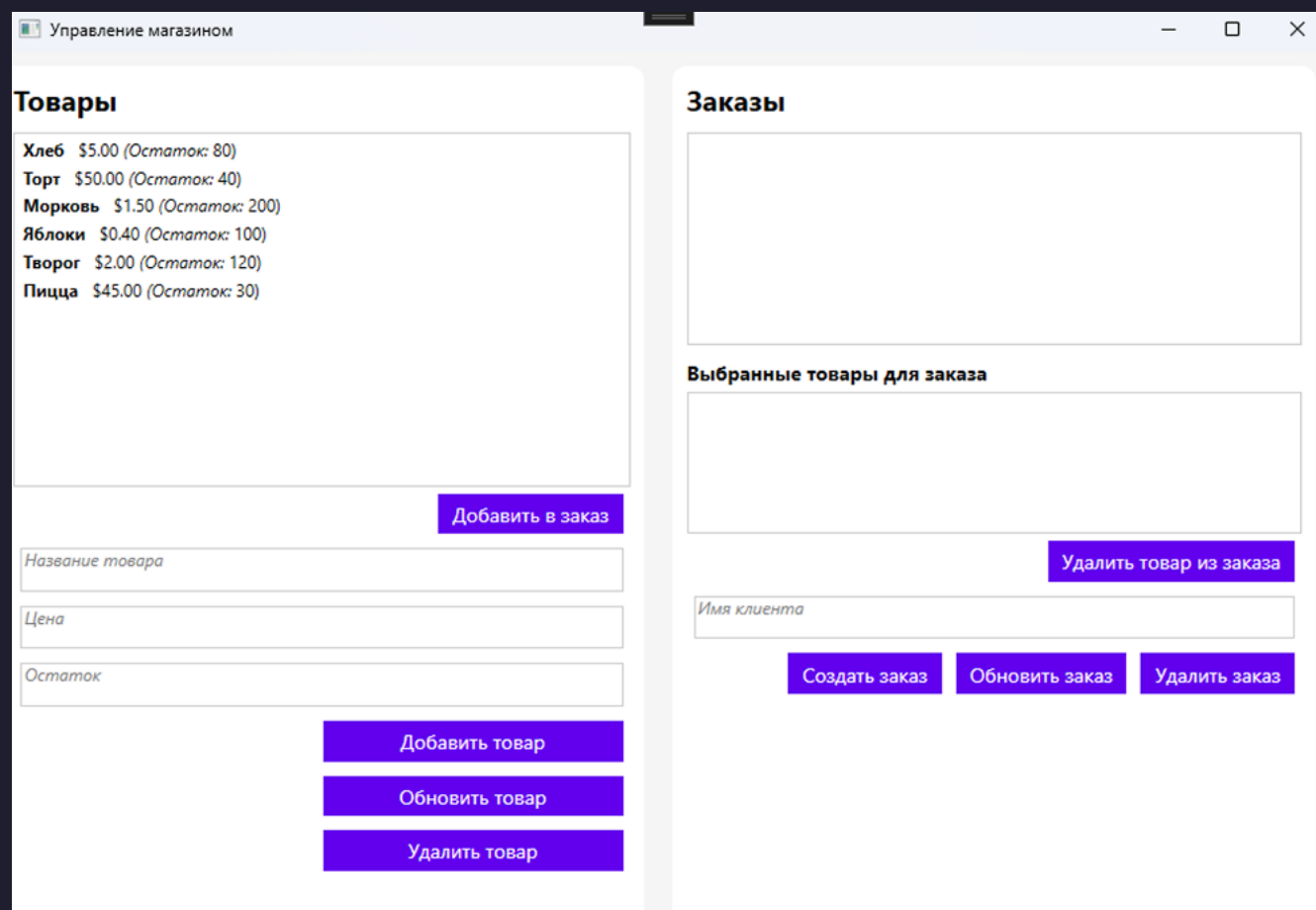


Рис. 13: Удаление товара из списка выбранных товаров для заказа

4 ВОПРОСЫ ДЛЯ ЗАЩИТЫ

1. Вопрос
2. Вопрос
3. Вопрос
4. Вопрос
5. Вопрос
6. Вопрос
7. Вопрос
8. Вопрос
9. Вопрос
10. Вопрос