МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Математическое обеспечение и применение ЭВМ»

Курсовая работа

по дисциплине «Объектное-ориентированное программирование»

на тему «Разработка программы с использованием объектно-ориентированного подхода. ИС «Склад»»

ПГУ 09.03.04 – 04КР232.11 ПЗ

Направление подготовки – 09.03.04 Программная инженерия
Профиль подготовки – Программная инженерия

Выполнил студент _____ Каневский Г.М.
Группа 23ВП2

Руководитель
к.т.н., доцент _____ Афонин А.Ю.

Работа защищена с оценкой _____

Преподаватели _____

_____

Дата защиты _____

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Пензенский государственный университет»
(ФГБОУ ВО «Пензенский государственный университет»)

Кафедра «Математическое обеспечение и применение ЭВМ»

«УТВЕРЖДАЮ»

Зав. кафедрой _____ ***Козлов А.Ю.***

"___" _____ 2025 г.

ЗАДАНИЕ
на курсовое проектирование по дисциплине
«Объектно-ориентированное программирование»

Студенту **Каневскому Глебу Марковичу** **Группа** ***23ВП2***

Тема проекта: Разработка программы с использованием объектно-ориентированного подхода. ИС «Склад».

Исходные данные (технические требования) на проектирование

1. Разработать программу БД «Склад».
2. Данные по базе включают: а) название товара, б) код товара, в) объем занимаемого места, г) масса.
3. Программа должна выполнять следующие действия: создание базы данных, удаление базы данных, сохранение текущей базы данных в файл, добавление записей, удаление записей, редактирование записей, поиск и сортировка записей, фильтрация записей по определенному критерию.
4. Обязательные требования к программе: многомодульность, использование сложных типов данных, использование коллекции для организации базы данных. Старт приложения должен сопровождаться всплывающим окном с информацией об авторе и темой проекта.
5. Визуальный интерфейс приложения реализовать с использованием графических библиотек (WinForms).
6. Среда разработки ПО: Microsoft Visual Studio 2019/2022, VSCode
7. Язык программирования: C#
8. Программное обеспечение должно быть полностью отлажено и протестировано, и должно функционировать под управлением ОС Windows 10 и выше.

Объём работ по курсу

1. Расчётная часть

1) Анализ требований.
2) Выбор и освоение инструментальных средств анализа и проектирования
3) Определение структуры и функций приложения
4) Разработка диаграмм описания приложения
5) Реализация приложения на языке C# в среде Microsoft Visual Studio
6) Отладка и тестирование приложения

2. Графическая часть

1) Схема данных
2) Диаграмма вариантов использования
3) Диаграмма классов

3. Экспериментальная часть

1) Отладка компонентов приложения и их взаимодействия
2) Функциональное тестирование приложения

Срок выполнения проекта по разделам

1	Анализ требований	к	18 февраля	2025 г.
2	Определение структуры и функций приложения	к	1 марта	2025 г.
3	Разработка UML диаграмм приложения	к	15 марта	2025 г.
4	Реализация приложения	к	12 апреля	2025 г.
5	Отладка и тестирование приложения	к	29 апреля	2025 г.
6	Оформление пояснительной записки проекта	к	17 мая	2025 г.
7	Защита курсовой работы	к	24 мая	2025 г.

Дата выдачи задания « 4 » *февраля* *2025 г.*
Дата защиты проекта « » *202 г.*

Руководитель _____ /к.т.н. А.Ю. Афонин/

Задание получил "4" *февраля* 2025 г.
Студент _____ /Г.М. Каневский/

Оглавление

Введение 5

1 Постановка задачи и предметной области 6

2 Выбор решения 8

2.1 Определение необходимых модулей программы 9

2.2 Определение структуры файла базы данных 11

2.3 Модель интерфейса 11

3 Описание разработки программы 13

3.1 Кодирование 13

3.2 Диаграмма компонентов 14

4. Отладка и тестирование 15

4.1 Функциональное тестирование 15

4.2 Тестирование приложения 16

5 Описание программы 23

5.1 Разработка приложения OOP_Curs_Kanevskii_23VP2 23

Список использованных источников 24

Приложение А 26

Приложение Б 48

Введение

Согласно заданию курсовой работы, необходимо разработать информационную систему под названием «Склад», используя принципы объектно-ориентированного программирования (ООП). Основной целью проекта является создание базы данных для хранения сведений о товарах с реализацией основных операций: добавление, удаление, редактирование, поиск, сортировка и фильтрация записей.

Актуальность разработки такой системы определяется активным внедрением информационных технологий в процессы управления и учёта товарных позиций. Автоматизация обработки и хранения данных способствует повышению производительности предприятий, снижению количества ошибок и упрощению доступа к необходимой информации.

Разработка приложения предполагает использование современных инструментов, включая язык программирования C# и среду Microsoft Visual Studio. В рамках проекта будет реализована работа с коллекциями для формирования базы данных, а также создание графического пользовательского интерфейса с использованием библиотек, таких как Windows Forms.

Для успешного выполнения проекта предстоит решить задачи:

1. Проведение анализа требований и проектирование архитектуры приложения.
2. Создание UML-диаграмм.
3. Разработка приложения с применением объектно-ориентированного подхода.
4. Проведение отладки и тестирования основных функций.
5. Подготовка документации и организация защиты проекта.

В результате реализации проекта будет создана информационная система, соответствующая заявленным требованиям и предоставляющая удобный, интуитивно понятный интерфейс для работы с товарными данными.

1 Постановка задачи и предметной области

Требуется разработать программное обеспечение под названием «Склад». Информация о товарах, хранящихся на складе, должна включать следующие характеристики: идентификатор (id), наименование, масса одной единицы (в килограммах), стоимость за единицу (в рублях), общее количество (штук/единиц). Для реализации этого раздела необходимо определить структуру базы данных. **База данных (БД) [1]** — это организованная совокупность взаимосвязанных данных, предназначенная для их хранения, обновления и обработки с помощью системы управления базами данных (СУБД). **Система управления базами данных (СУБД) [1]** — это программное обеспечение, обеспечивающее создание, ведение и совместное использование базы данных, а также контроль над доступом к данным и их целостностью. Структура базы данных позволит выявить возможные ограничения системы и минимизировать их влияние на работу приложения.

Программа должна обеспечивать создание файла базы данных, возможность добавления новых записей (**Запись** — это структурированный набор данных, объединённых в одну строку таблицы, каждая из которых соответствует отдельному экземпляру объекта [2]), редактирования и удаления уже существующих данных, а также реализацию функций поиска (**Поиск в БД** — это процесс извлечения информации, соответствующей определённым критериям, с целью получения нужных записей из таблиц [3]), сортировки (**Сортировка в БД** — это операция упорядочивания записей в таблице по значениям одного или нескольких полей, по возрастанию или убыванию [4]) и фильтрации (**Фильтрация в БД** — это операция отбора записей, удовлетворяющих определённому условию, с целью отображения только нужных данных из общего набора [5]) по заданным параметрам. Помимо этого, необходимо предусмотреть возможность сохранения текущей базы данных в файл, экспорт (**Экспорт базы данных** — это процесс переноса данных из одной системы управления базами данных (СУБД) в другой формат

или в другую СУБД. Этот процесс включает извлечение, преобразование и сохранение данных в формате, который может быть использован для резервного копирования, миграции или анализа. Экспорт может быть выполнен с использованием различных инструментов и форматов, таких как SQL, CSV, JSON, XML и другие [6]) данных в PDF-формат, полное удаление содержимого базы, а также вывод информации в удобочитаемом виде. Все эти возможности продемонстрированы на диаграмме вариантов использования (рисунок 1).

Пользовательский интерфейс (**Пользовательский интерфейс (UI)** в C# — это совокупность визуальных элементов, с которыми взаимодействует пользователь при работе с программой. В C# пользовательские интерфейсы обычно создаются с использованием технологий, таких как **Windows Forms**, **WPF (Windows Presentation Foundation)** или **UWP (Universal Windows Platform)**, и включают элементы управления, такие как кнопки, текстовые поля, метки, выпадающие списки и другие компоненты [7]) должен быть интуитивно понятным, с чётким разделением элементов меню и таблицы, отображающей сведения о товарах. Для полноценной реализации функционала требуется изучить принципы работы с файлами и организовать эффективное управление данными внутри программы.

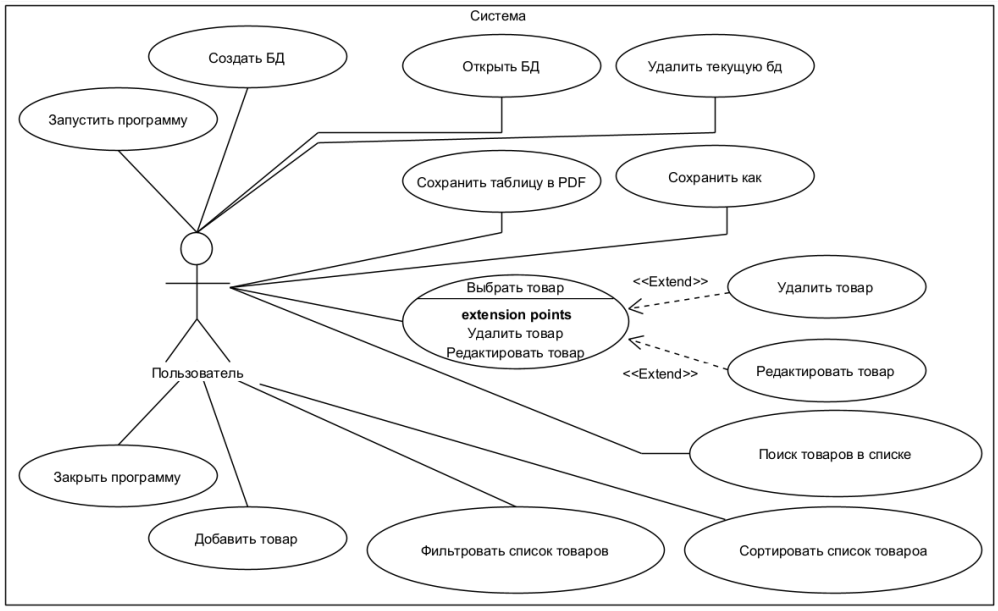

Рисунок 1 – Диаграмма вариантов использования

2 Выбор решения

Для реализации поставленной задачи было принято решение о разделении проекта на следующие программные модули (**Программный модуль в C#** — это логически обособленная часть программы, реализующая определённую функциональность и предназначенная для повторного использования. В языке C# программные модули чаще всего реализуются в виде **классов**, **пространств имён (namespaces)**, **сборок (assemblies)** или **библиотек (DLL)**. Модули способствуют структурированию кода, упрощают сопровождение и повторное использование программных компонентов [8]):

1. Product — модель, описывающая структуру данных для представления товара;

2. DataBaseContext — модуль, выполняющий функции репозитория;

3. MainForm — компонент, отвечающий за пользовательский интерфейс;

4. HelloForm — модуль, реализующий окно приветствия пользователя;

5. Program — основной модуль, содержащий точку входа в приложение. Взаимодействие между классами программы схематически показано на диаграмме классов (рисунок 2).

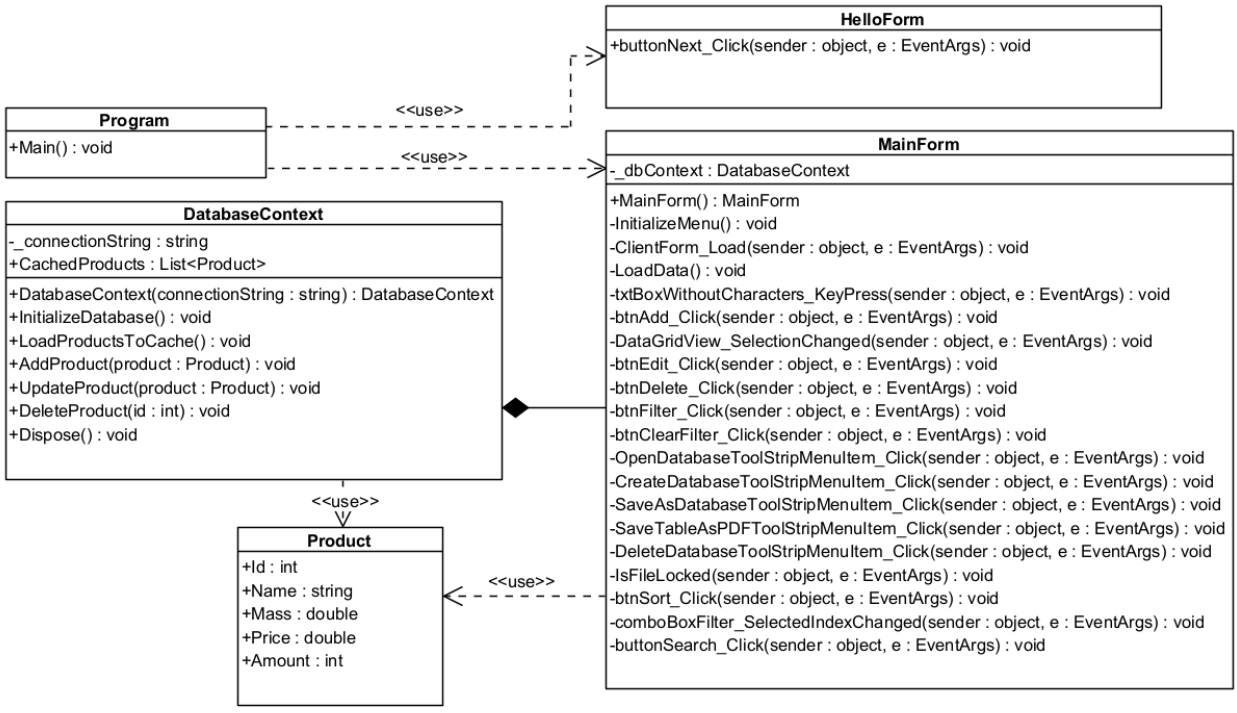

Рисунок 2 – Диаграмма классов

2.1 Определение необходимых модулей программы

Модуль DatabaseContext представляет собой репозиторий, осуществляющий работу с базой данных SQLite (**SQLite** — это легковесная, встраиваемая реляционная система управления базами данных (СУБД), реализованная в виде библиотеки. Она не требует отдельного сервера для работы, а все данные хранятся в одном файле на диске. SQLite поддерживает стандарт SQL, является кроссплатформенной, и широко используется в мобильных приложениях, встроенных системах и небольших проектах [9]), хранящей записи в табличной форме об объектах класса Furniture. Содержит методы для добавления, удаления объектов. Код данного модуля приведён на рисунке 3.

```
public class DatabaseContext : IDisposable
{
    private readonly string _connectionString;
    Ссылок: 10
    public List<Product> CachedProducts { get; private set; } = new List<Product>();

    Ссылок: 5
    public DatabaseContext(string connectionString)...

    Ссылок: 4
    public void InitializeDatabase()...

    Ссылок: 4
    public void LoadProductsToCache()...

    Ссылок: 1
    public void AddProduct(Product product)...

    Ссылок: 1
    public void UpdateProduct(Product product)...

    Ссылок: 1
    public void DeleteProduct(int id)...

    Ссылок: 2
    public void Dispose()...
}
```

Рисунок 3 – Модуль репозитория

Модуль MainForm представляет пользовательский интерфейс, позволяющий пользователю осуществлять весь задуманный функционал программы. Код данного модуля приведён на рисунке 4.

```
public partial class MainForm : Form
{
    private DatabaseContext _dbContext;

    Ссылок: 1
    public MainForm()...
    Ссылок: 1
    private void InitializeMenu()...

    Ссылок: 1
    private void ClientForm_Load(object sender, EventArgs e)...

    Ссылок: 10
    private void LoadData()...

    Ссылок: 1
    private void txtWeight_KeyPress(object sender, KeyPressEventArgs e)...

    Ссылок: 1
    private void txtPrice_KeyPress(object sender, KeyPressEventArgs e)...

    Ссылок: 1
    private void btnAdd_Click(object sender, EventArgs e)...
    Ссылок: 1
    private void DataGridView_SelectionChanged(object sender, EventArgs e)...

    Ссылок: 1
    private void btnEdit_Click(object sender, EventArgs e)...

    Ссылок: 1
    private void btnDelete_Click(object sender, EventArgs e)...

    Ссылок: 1
    private void btnFilter_Click(object sender, EventArgs e)...

    Ссылок: 1
    private void btnClearFilter_Click(object sender, EventArgs e)
```

Рисунок 4 – Модуль клиентской формы

Модуль Product отвечает за представление данных, хранящихся в базе. Содержит поля, характеризующие товар. Представлен на рисунке 5.

```
public class Product
{
    Ссылок: 9
    public int Id { get; set; }
    Ссылок: 11
    public string Name { get; set; }
    Ссылок: 11
    public double Mass { get; set; }
    Ссылок: 11
    public double Price { get; set; }

    Ссылок: 11
    public int Amount { get; set; }
}
```

Рисунок 5 – Модуль данных

2.2 Определение структуры файла базы данных

Для хранения данных выбрана опосредованная бинарная сериализация в sqlite файл через SQL-команды (**Бинарная сериализация** — это процесс преобразования объекта или структуры данных в последовательность байтов для хранения или передачи. В контексте **SQLite** бинарная сериализация может использоваться для сохранения объектов в поле типа BLOB (Binary Large Object) [10]). Внутри файла формируется таблица, содержащая все объекта типа «товар» с их характеристиками. Структура файла базы данных представлена на рисунке 6.

```
CREATE TABLE IF NOT EXISTS Product (
    Id INTEGER PRIMARY KEY AUTOINCREMENT,
    Name TEXT NOT NULL,
    Mass REAL NOT NULL,
    Price REAL NOT NULL,
    Amount INTEGER NOT NULL
)
```

Рисунок 6 – Структура файла базы данных

2.3 Модель интерфейса

Модель интерфейса разрабатываемого приложения представлена на рисунке 7.

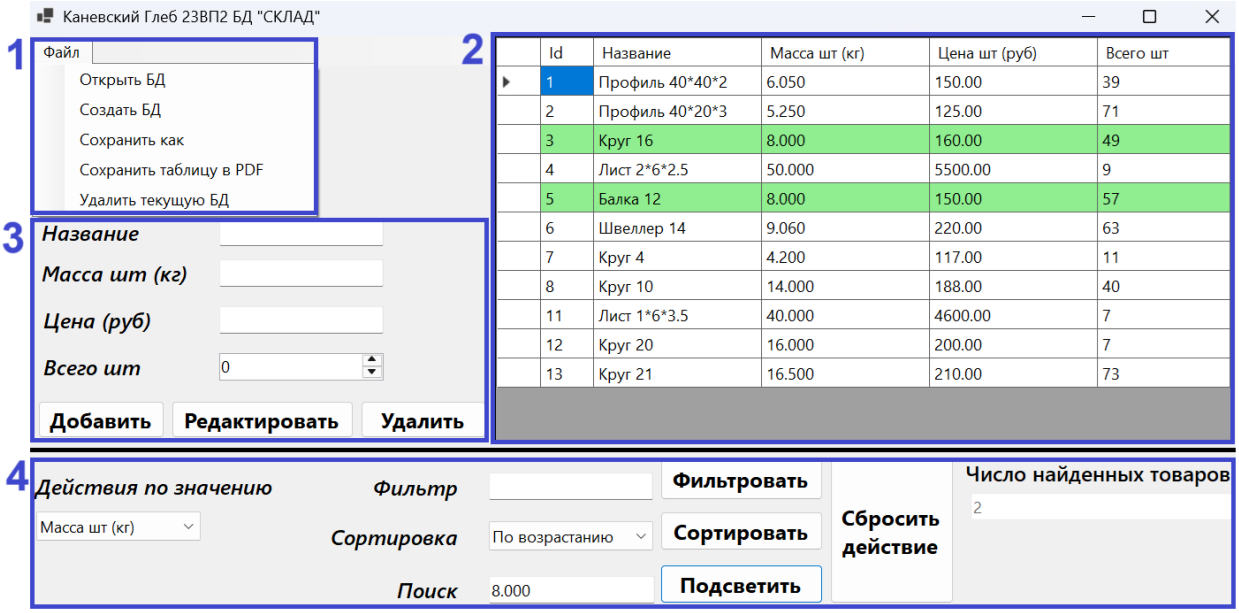

Рисунок 7 – Модель интерфейса приложения (MainForm)

Под соответствующими обозначениями на форме:

1 – Блок для работы с файлами.

2 – Таблица, отображающая товары в файле.

3 – Блок для создания/редактирования/удаления конкретного товара.

4 – Блок действия с таблицей (фильтрация, сортировка, поиск).

3 Описание разработки программы

3.1 Кодирование

В ходе выполнения курсового проекта было разработано приложение «OOP_Curs_Kanevskii_23VP2» (код приведён в приложении А).

Для алгоритма открытия базы данных была разработана диаграмма деятельности, представленная на рисунке 8.

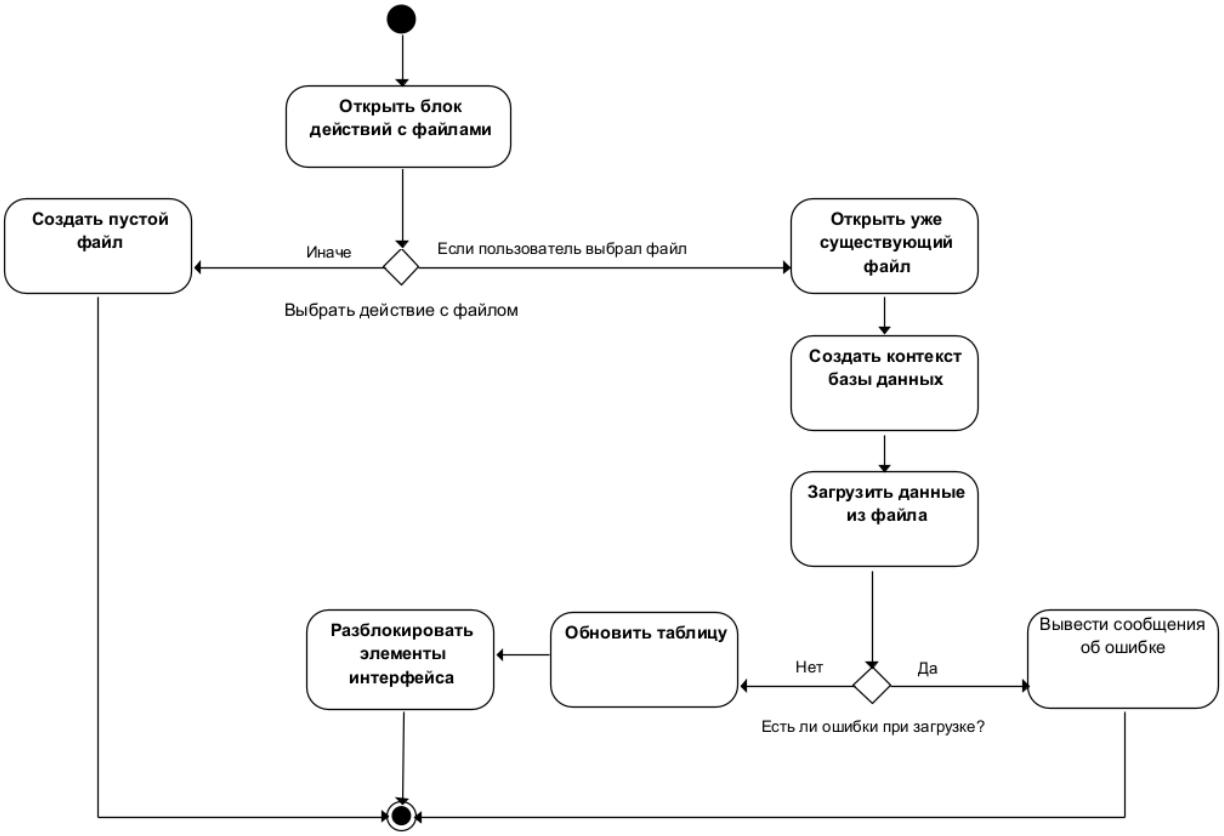

Рисунок 8 – Диаграмма деятельности для действия «окрыть базу данных»

Представленный на диаграмме деятельности алгоритм описывает последовательность действий при выборе файла пользователем. При успешном выборе создается подключение к SQLite, проверяется структура БД и загружаются данные. Разблокируются элементы управления интерфейса для работы с данными. В случае ошибки, та выводится пользователю, подключение сбрасывается.

3.2 Диаграмма компонентов

В процессе выполнения курсового проекта была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты (**Структурные компоненты** — это ключевые части (элементы) системы, программы или архитектуры, которые определяют её организацию и взаимодействие между частями. В программировании под структурными компонентами обычно понимаются модули, классы, функции, интерфейсы, блоки данных и другие элементы, из которых строится программная система. Они обеспечивают разбиение сложной системы на логически обоснованные, независимые части, что облегчает разработку, сопровождение и масштабирование программного обеспечения [11]) и связи между компонентами (рисунок 9).

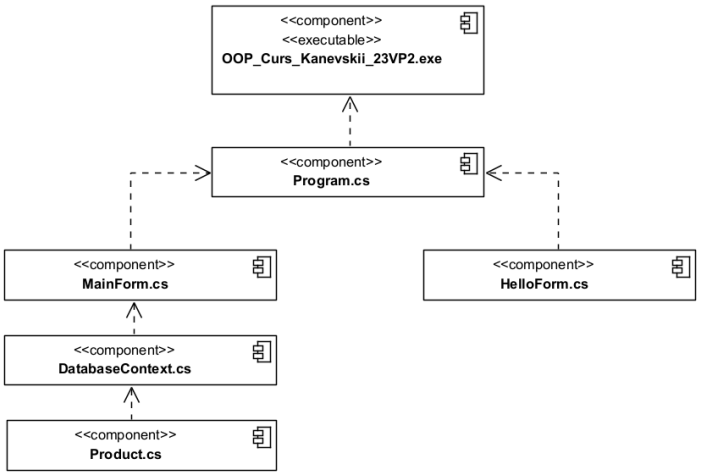

Рисунок 9 – Диаграмма компонентов

Описание компонентов приведено в таблице 1.

Таблица 1 – Компоненты программы

Название компонента	Назначение
OOP_Curs_Kanevskii_23VP2.exe	Исполняемый файл
Program.cs	Точка входа в программу
HelloForms.cs	Приветственная форма при запуске
MainForm.cs	Форма пользовательского интерфейса
DatabaseContext.cs	Файл с классом репозитория
Product.cs	Файл модели данных

4. Отладка и тестирование

4.1 Функциональное тестирование

Функциональное тестирование — один из ключевых видов тестирования программного обеспечения, направленный на проверку соответствия системы заявленным требованиям и корректности работы её функций [12]. В рамках курсового проекта оно позволяет убедиться, что разработанное приложение выполняет задачи, предусмотренные техническим заданием, а также корректно обрабатывает различные сценарии использования, включая ошибочные и граничные случаи.

Основная цель функционального тестирования — проверить, работает ли система так, как ожидается, и соответствует ли её поведение требованиям пользователя [13]. Для этого тестировщики имитируют действия реальных пользователей, проверяя:

1. Основной функционал — выполняет ли приложение заявленные операции (например, регистрация пользователей, обработка данных, формирование отчётов);

2. Интерфейс и удобство использования — насколько интуитивно понятны элементы управления, нет ли противоречий в логике взаимодействия;

3. Обработку исключительных ситуаций — как система реагирует на некорректный ввод, потерю соединения, нехватку ресурсов и другие нештатные ситуации;

4. Интеграцию компонентов — корректно ли взаимодействуют отдельные модули приложения, правильно ли передаются и обрабатываются данные между ними.

Особое внимание уделяется граничным условиям и неочевидным сценариям, которые могут привести к сбоям. Например, проверяется

поведение системы при вводе максимально допустимых значений, пустых полях или попытке выполнения запрещённых операций.

Ещё одним важным аспектом является отказоустойчивость и восстановление после ошибок. Тестирование помогает оценить, насколько система сохраняет стабильность при возникновении сбоев, предоставляет ли пользователю понятные сообщения об ошибках и может ли продолжить работу после их устранения.

Проведение функционального тестирования в курсовом проекте не только подтверждает работоспособность приложения, но и помогает выявить потенциальные уязвимости и улучшить качество продукта перед его демонстрацией. Результаты тестирования позволяют доработать логику приложения, исправить ошибки и обеспечить его надёжность в реальных условиях эксплуатации.

4.2 Тестирование приложения

В курсовой работе было выполнено функциональное тестирование разработанного программного обеспечения. Результаты тестирования приведены в таблице 2.

Таблица 2 – Проведённые тесты

Состав теста	Ожидаемый результат	Наблюдаемый результат
1. Создание пустого файла базы данных.	Система должна создать пустой файл базы данных и открыть его в таблице.	Создан файл базы данных, инструменты для работы с файлом разблокированы. (Рисунок 10)
2. Попытка добавить запись в базу данных без заполнения всех полей.	Система должна распознать пустые поля и вывести пользователю оповещение о необходимости заполнить поля.	Запись не добавлена, так как система распознала пустые поля и вывела пользователю сообщение о том, что необходимо заполнить поля. (Рисунок 11)
3. Добавление нового товара в базу данных.	Система должна проверить введённые данные из полей и добавить запись.	Введённые данные проверены, запись добавлена в базу данных и выведена в таблицу. (Рисунок 12)
4. Фильтрация по полю «Название».	Система должна распознать категорию фильтрации,	Данные считаны и проведена фильтрация по

	считать введённое значение, затем провести фильтрацию (т.е. отформатировать таблицу).	полю «Название». (Рисунок 13)
5. Сортировка товаров по массе (выбор порядка сортировки - по возрастанию).	Система должна распознать категорию сортировки, распознать порядок сортировка (по возрастанию), затем провести форматирование таблицы	Данные считаны и проведена сортировка товаров по массе (выбор порядка сортировки - по возрастанию). Отсортированная таблица выведена на экран. (Рисунок 14)
6. Поиск по полю «цена» введенного значения пользователем.	Система должна считать выбранную категорию поиска и введённое значение, затем провести поиск (подсветить совпадения).	Данные считаны и проведён поиск по полю «цена» введенного значения пользователем. (Рисунок 15)
7. Удаление выбранного товара (Круг 21).	Система должна считать выделенную строку и удалить объект из базы данных и таблицы.	Строка считана верно и удалена из базы данных, таблица обновлена. (Рисунок 16-17)
8. Редактирование товара (Профиль 40*40*2). Уменьшено общее количество на 1шт.	Система должна распознать выбранную строку, вывести поля товара пользователю для редактирования, считать новые данные и внести изменения.	Данные изменены и отображены в таблице (Рисунок 18-19)
9. Сохранение текущей таблицы в PDF формат.	Система должна проверить имя PDF файла, создать PDF файл и перенести данные из базы в него.	Создан PDF файл с указанным названием по выбранной базе данных. (Рисунок 20)

Ниже на рисунках 10-20 представлены скриншоты проведённых тестов.

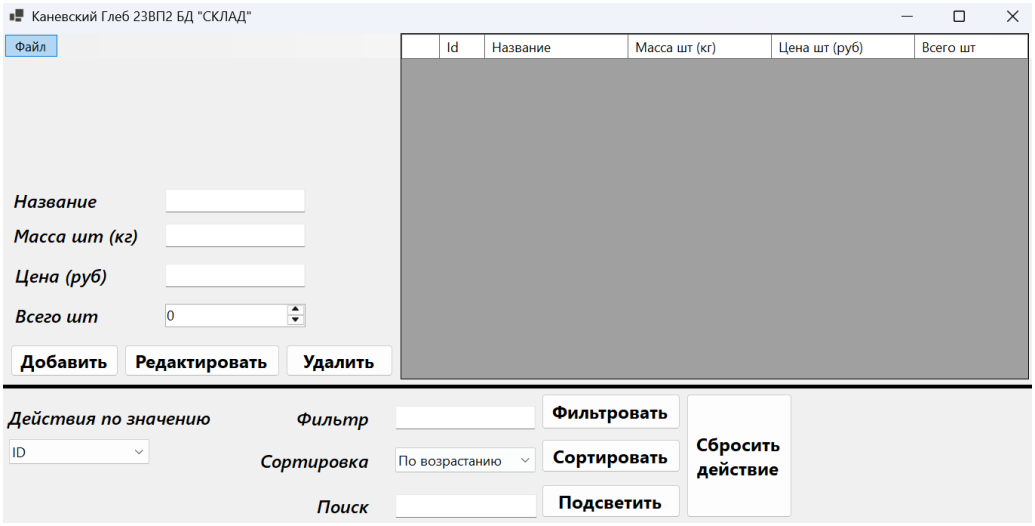

Рисунок 10 - Создание пустого файла базы данных

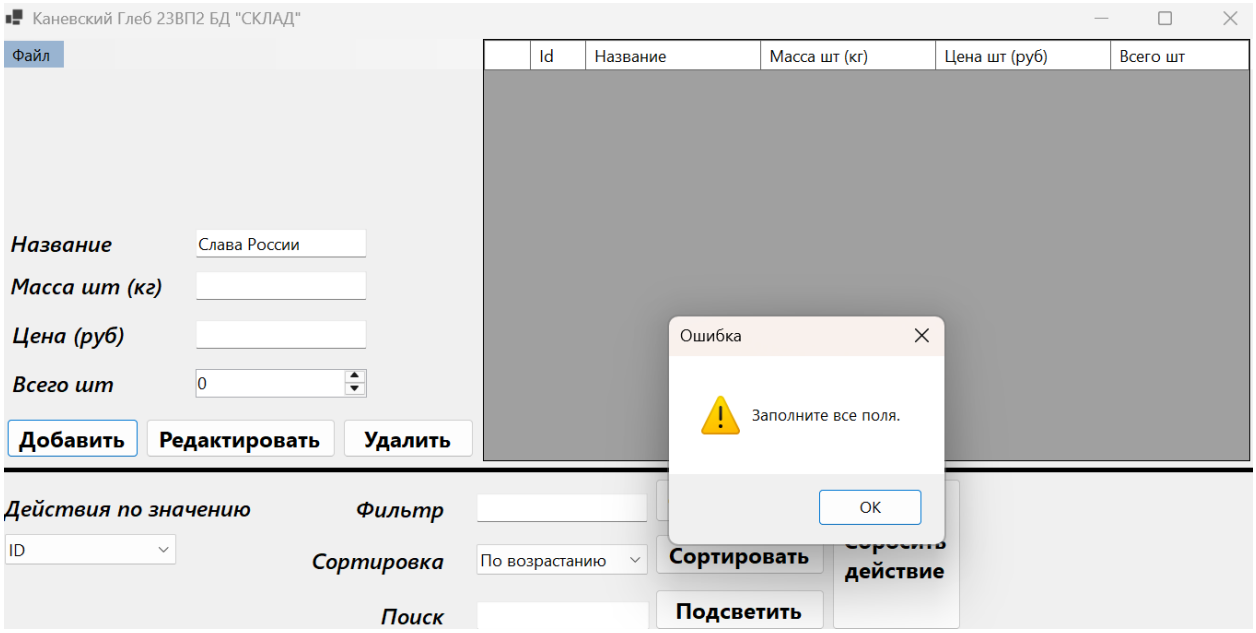

Рисунок 11 - Попытка добавить запись в базу данных без заполнения всех полей

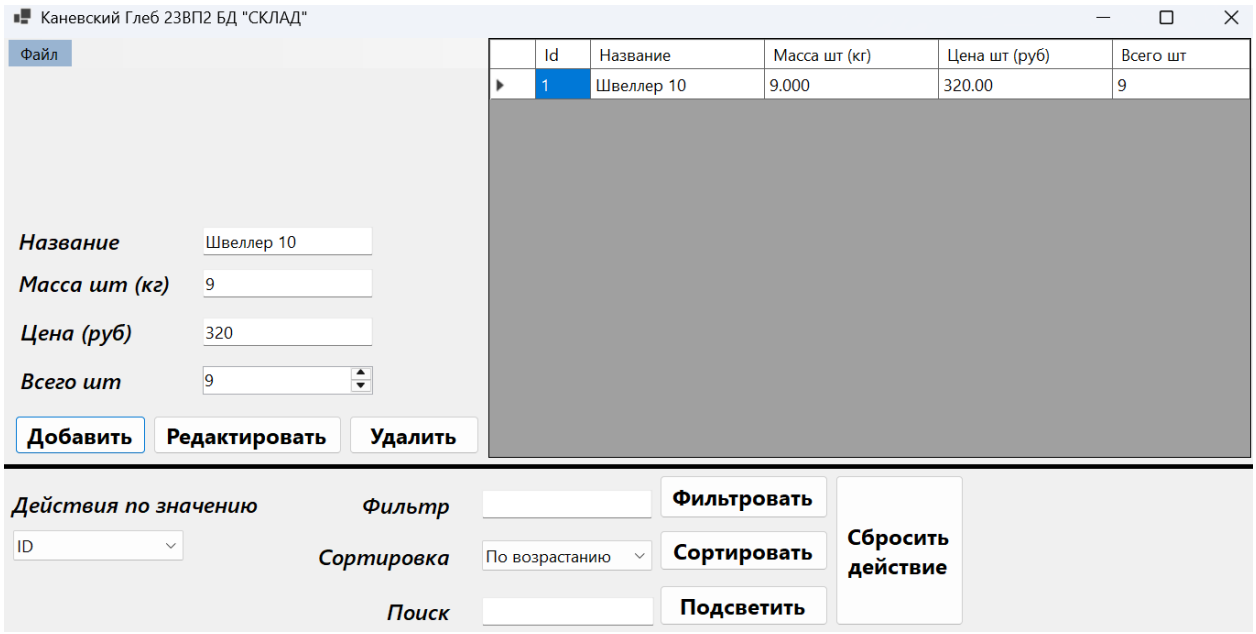

Рисунок 12 - Добавление нового товара в базу данных

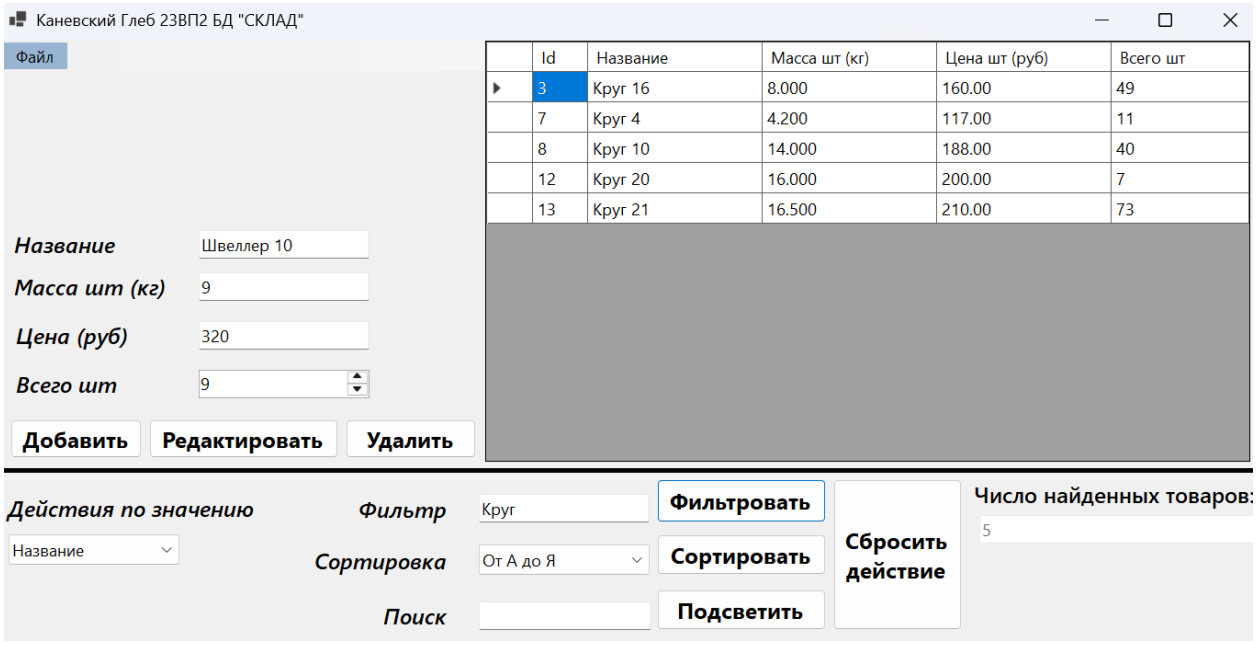

Рисунок 13 - Фильтрация по полю «Название»

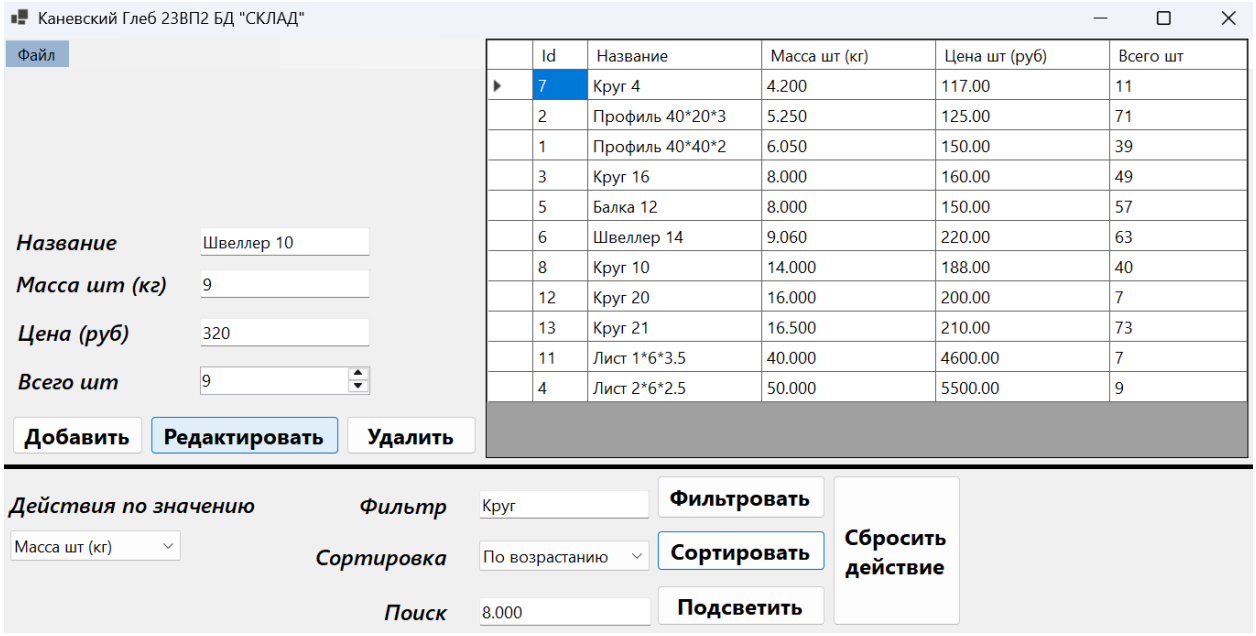

Рисунок 14 - Сортировка товаров по массе (выбор порядка сортировки - по возрастанию)

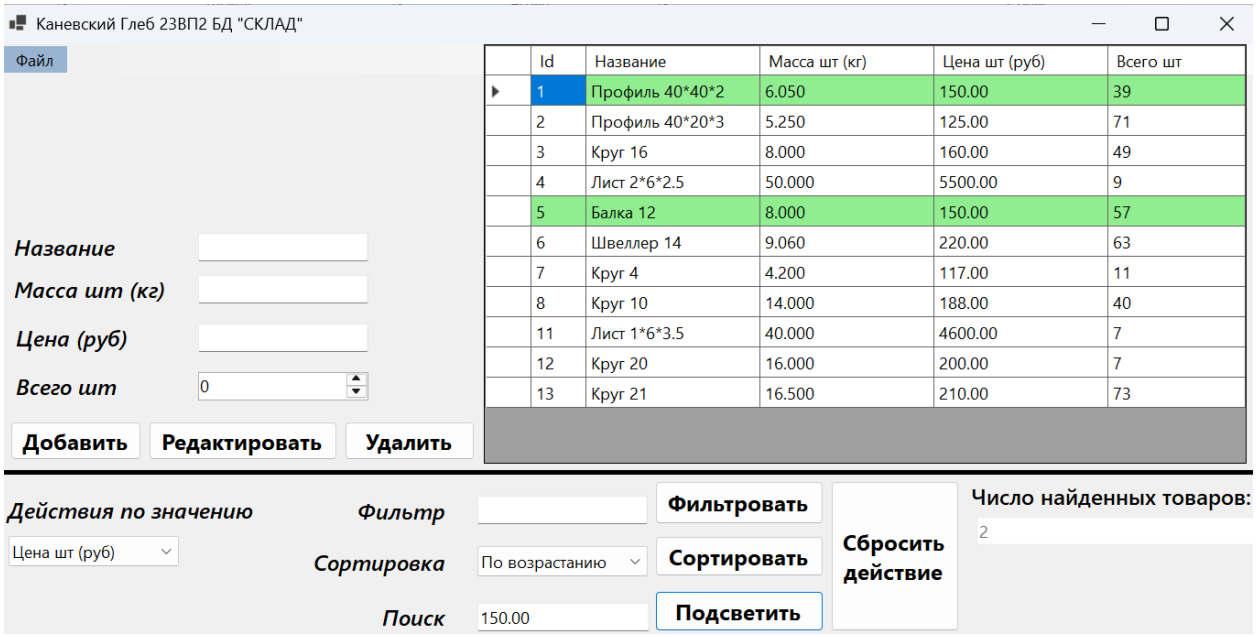

Рисунок 15 - Поиск по полю «цена» введенного значения пользователем

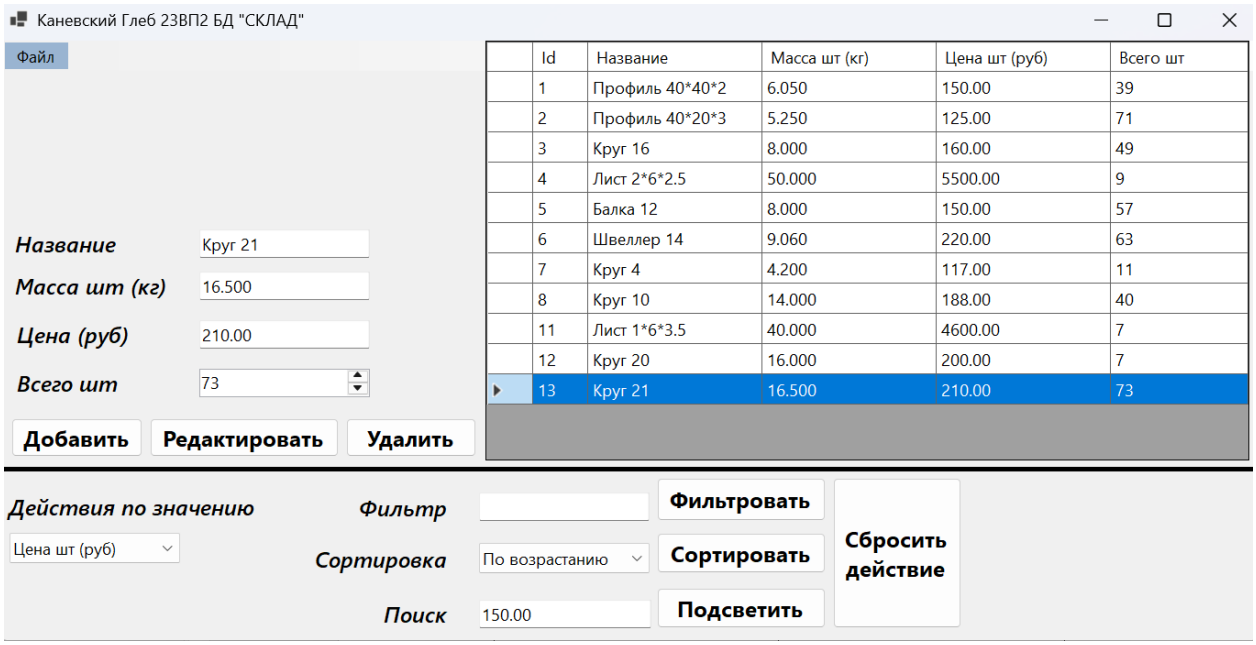

Рисунок 16 - Удаление выбранного товара (Круг 21) До

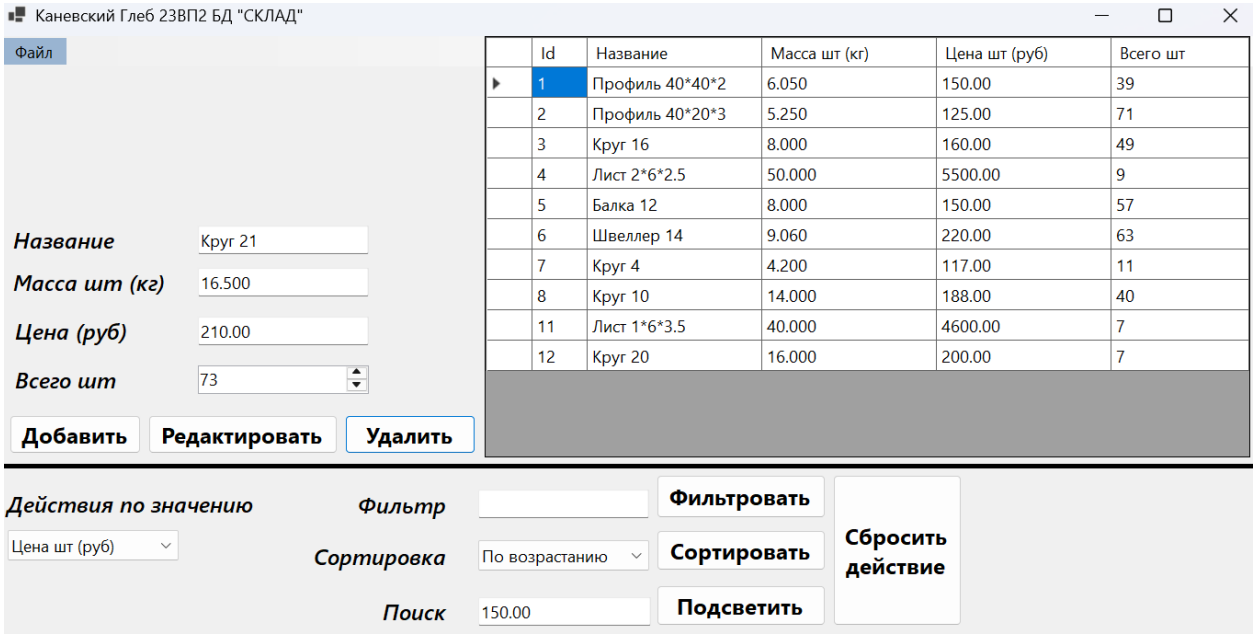

Рисунок 17 - Удаление выбранного товара (Круг 21) После

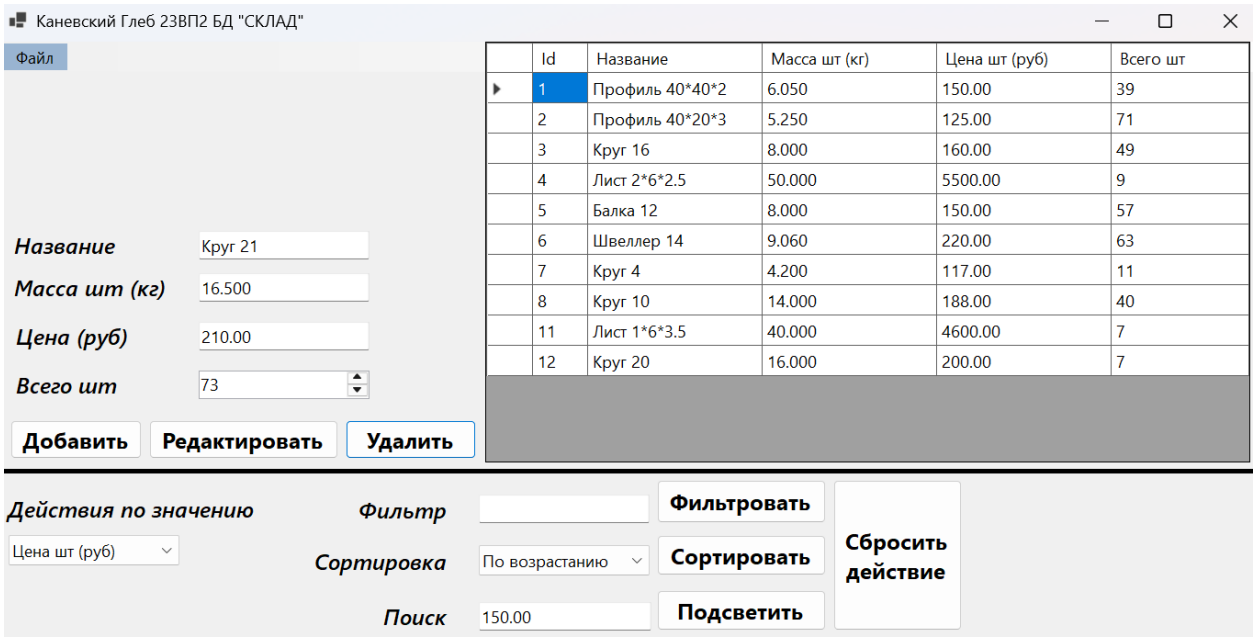

Рисунок 18 - Редактирование товара (Профиль 40*40*2). Уменьшено общее количество на 1шт До

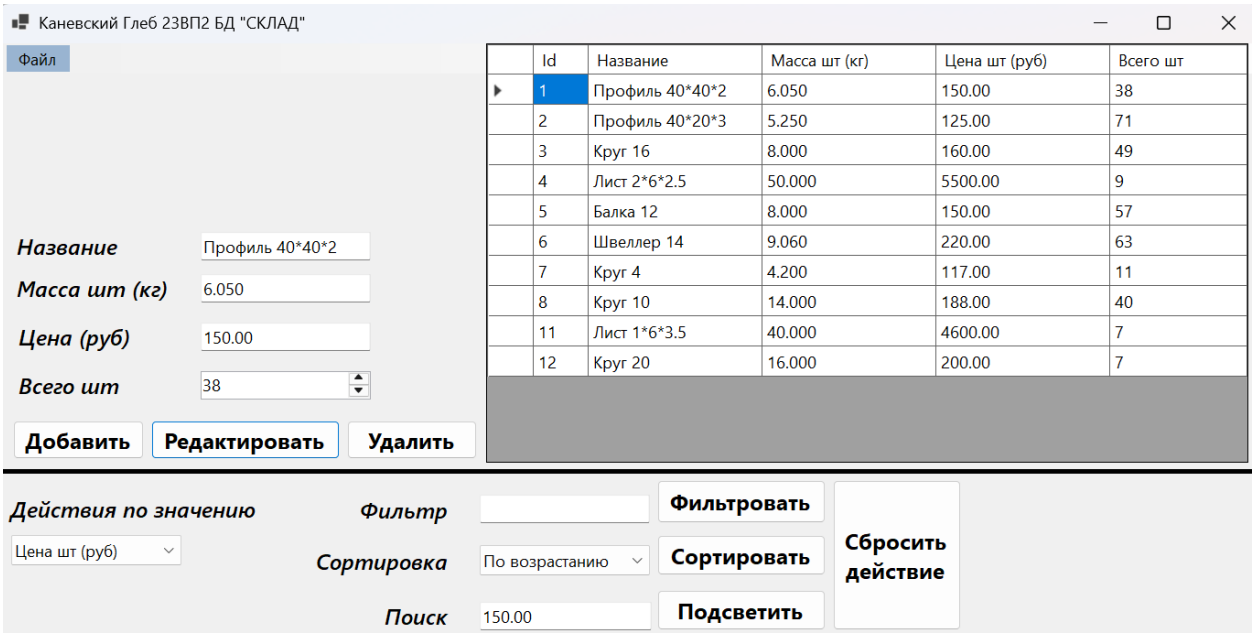

Рисунок 19 - Редактирование товара (Профиль 40*40*2). Уменьшено общее количество на 1шт После

Таблица данных

Id	Название	Масса шт (кг)	Цена шт (руб)	Всего шт
1	Профиль 40*40*2	6.050	150.00	38
2	Профиль 40*20*3	5.250	125.00	71
3	Круг 16	8.000	160.00	49
4	Лист 2*6*2.5	50.000	5500.00	9
5	Балка 12	8.000	150.00	57
6	Швеллер 14	9.060	220.00	63
7	Круг 4	4.200	117.00	11
8	Круг 10	14.000	188.00	40
11	Лист 1*6*3.5	40.000	4600.00	7
12	Круг 20	16.000	200.00	7

Рисунок 20 - Сохранение текущей таблицы в PDF формат

В ходе выполнения тестирования несовпадения и неточности обнаружены не были. Следовательно, можно сделать вывод, что программа работает корректно. Это подтверждает эффективность применённого подхода к функциональному тестированию, при котором акцент сделан на проверку соответствия поведения системы требованиям пользователя и обработку типовых и граничных сценариев.

5 Описание программы

5.1 Разработка приложения OOP_Curs_Kanevskii_23VP2

Приложение "OOP_Curs_Kanevskii_23VP2.exe" представляет собой исполняемый модуль программы, разработанной в рамках курсового проекта. Его работа начинается с отображения приветственной формы (HelloForm), содержащей информацию об авторе, названии приложения и его назначении. Пользователь может нажать кнопку "Перейти к программе", чтобы непосредственно перейти к программе. После загрузки открывается основная форма приложения (MainForm), однако её функционал изначально ограничен – большинство элементов интерфейса (кнопки, поля ввода, списки) остаются неактивными до тех пор, пока пользователь не откроет файл базы данных. Это сделано для предотвращения ошибок, связанных с отсутствием данных.

Логика работы основной формы:

1. Открытие файла базы данных – после выбора файла (через меню или кнопку) интерфейс активируется, и пользователь получает доступ ко всем функциям.

2. Обработка действий – вся логика нажатий на кнопки, обработка ввода и взаимодействие с данными реализованы в коде файла MainForm.cs.

3. Ошибки и исключения – при некорректных действиях (например, попытке выполнить операцию без загруженных данных) программа выводит соответствующие сообщения, предотвращая аварийное завершение.

Подробное описание всех функций приведено в руководстве пользователя, которое следует изучить перед началом работы с приложением.

Список использованных источников

1. Силберштац А., Корт В., Судерланд К. Системы баз данных. — 6-е изд. — М. : Вильямс, 2011. — 1376 с.

2. Овчинников В. В. Базы данных: проектирование, реализация и сопровождение. — СПб. : Питер, 2019. — 304 с.

3. Панченко В. Ю. Базы данных: основы проектирования и использования. — М. : Форум, 2020. — 368 с.

4. Беспалов Д. А. Проектирование баз данных. — СПб. : Питер, 2018. — 240 с.

5. Абрамов С. М. Информационные системы и базы данных. — М. : КНОРУС, 2021. — 272 с.

6. Экспорт и импорт данных // MySQL Documentation. — URL: https://dev.mysql.com/doc/refman/8.0/en/mysqldump.html (дата обращения: 15.04.2025).

7. Создание приложения Windows Forms на C# // Microsoft Docs. — URL: https://learn.microsoft.com/en-us/visualstudio/get-started/csharp/tutorial-winforms?view=vs-2022 (дата обращения: 15.04.2025).

8. Пространства имён (руководство по программированию C#) // Microsoft Docs. — URL: https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/namespaces/ (дата обращения: 15.04.2025).

9. О SQLite // SQLite Documentation. — URL: https://www.sqlite.org/about.html (дата обращения: 17.04.2025).

10. Типы данных в SQLite // SQLite Documentation. — URL: https://www.sqlite.org/datatype3.html (дата обращения: 17.04.2025).

11. Макконнелл С. Совершенный код: Практическое руководство по разработке программного обеспечения. — М. : Вильямс, 2022. — 896 с.

12. Майерс Г. Дж., Сэндлер К., Баджетт Т. Искусство тестирования программного обеспечения = The Art of Software Testing. — 3-е изд. — Нью-Йорк : Wiley, 2011. — 336 с.

13. Канер К., Фалк Дж., Нгуен Х. Тестирование компьютерного программного обеспечения = Testing Computer Software. — 2-е изд. — Нью-Йорк : Wiley, 1999. — 480 с.

Приложение А

КОД ПРОГРАММЫ

(обязательное)

1. Код файла Program.cs

```
namespace OOP_Curs_Kanevskii_23VP2
{
    internal static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            ApplicationConfiguration.Initialize();

            using (HelloForm helloForm = new HelloForm())
            {
                // Создаем таймер для автоматического перехода через 10 секунд
                System.Windows.Forms.Timer autoCloseTimer = new
System.Windows.Forms.Timer();
                autoCloseTimer.Interval = 10000; // 10 секунд
                autoCloseTimer.Tick += (sender, e) =>
                {
                    autoCloseTimer.Stop();
                    helloForm.DialogResult = DialogResult.OK; // Имитируем нажатие
кнопки
                };
                autoCloseTimer.Start();

                // Показываем форму и ждем либо таймера, либо нажатия кнопки
                if (helloForm.ShowDialog() == DialogResult.OK)
                {
                    autoCloseTimer.Stop(); // Останавливаем таймер, если переход по
кнопке
                    Application.Run(new MainForm());
                }
            }
        }
    }
}
```

2. Код файла Product.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace OOP_Curs_Kanevskii_23VP2
{
    /// <summary>
    /// Класс "Товар" – базовую сущность товара в системе.
    /// </summary>
    public class Product
    {
        /// <summary>
        /// Уникальный идентификатор товара
        /// </summary>
        public int Id { get; set; }

        /// <summary>
        /// Название товара
        /// </summary>
        public string Name { get; set; }
```

```
        /// <summary>
        /// Масса товара (в кг)
        /// </summary>
        public double Mass { get; set; }

        /// <summary>
        /// Цена товара за единицу (в руб)
        /// </summary>
        public double Price { get; set; }

        /// <summary>
        /// Количество единиц товара в наличии (в штуках)
        /// </summary>
        public int Amount { get; set; }
    }
}
```

3. Код файла DatabaseContext.cs

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;

namespace OOP_Curs_Kanevskii_23VP2
{
    /// <summary>
    /// Класс для работы с базой данных SQLite, реализующий интерфейс IDisposable
    /// </summary>
    public class DatabaseContext : IDisposable
    {
        // Строка подключения к базе данных
        private readonly string _connectionString;

        // Кэшированный список продуктов для уменьшения обращений к БД
        public List<Product> CachedProducts { get; private set; } = new
List<Product>();

        // Свойство для доступа к строке подключения (только для чтения)
        public string ConnectionString => _connectionString;

        /// <summary>
        /// Конструктор класса DatabaseContext
        /// </summary>
        /// <param name="connectionString">Строка подключения к SQLite базе
данных</param>
        public DatabaseContext(string connectionString)
        {
            _connectionString = connectionString;
            InitializeDatabase(); // Инициализация структуры БД
            LoadProductsToCache(); // Загрузка данных в кэш
        }

        /// <summary>
        /// Инициализация структуры базы данных (создание таблицы, если не
существует)
        /// </summary>
        public void InitializeDatabase()
        {
            try
            {
                using (var connection = new SQLiteConnection(_connectionString))
                {
                    connection.Open();

                    // SQL-запрос для создания таблицы Product с необходимыми полями
```

```
                    var createTableCommand = new SQLiteCommand(@"
                        CREATE TABLE IF NOT EXISTS Product (
                            Id INTEGER PRIMARY KEY AUTOINCREMENT,
                            Name TEXT NOT NULL,
                            Mass REAL NOT NULL,
                            Price REAL NOT NULL,
                            Amount INTEGER NOT NULL
                        )", connection);
                    createTableCommand.ExecuteNonQuery();
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при инициализации базы данных.", ex);
            }
        }

        /// <summary>
        /// Загрузка всех продуктов из базы данных в кэш (CachedProducts)
        /// </summary>
        public void LoadProductsToCache()
        {
            CachedProducts.Clear(); // Очистка текущего кэша
            try
            {
                using (var connection = new SQLiteConnection(_connectionString))
                {
                    connection.Open();
                    var command = new SQLiteCommand("SELECT * FROM Product",
connection);
                    using (var reader = command.ExecuteReader())
                    {
                        while (reader.Read())
                        {
                            // Заполнение кэша данными из БД
                            CachedProducts.Add(new Product
                            {
                                Id = reader.GetInt32(0),
                                Name = reader.GetString(1),
                                Mass = reader.GetDouble(2),
                                Price = reader.GetDouble(3),
                                Amount = reader.GetInt32(4)
                            });
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при загрузке товаров в кэш.", ex);
            }
        }

        /// <summary>
        /// Добавление нового продукта в базу данных
        /// </summary>
        /// <param name="product">Объект продукта для добавления</param>
        public void AddProduct(Product product)
        {
            try
            {
                using (var connection = new SQLiteConnection(_connectionString))
                {
                    connection.Open();
                    // SQL-запрос для вставки нового продукта
```

```
                    var command = new SQLiteCommand(@"
                        INSERT INTO Product (Name, Mass, Price, Amount)
                        VALUES (@Name, @Mass, @Price, @Amount)", connection);
                    // Добавление параметров с защитой от SQL-инъекций
                    command.Parameters.AddWithValue("@Name", product.Name);
                    command.Parameters.AddWithValue("@Mass", product.Mass);
                    command.Parameters.AddWithValue("@Price", product.Price);
                    command.Parameters.AddWithValue("@Amount", product.Amount);
                    command.ExecuteNonQuery();
                }
                LoadProductsToCache(); // Обновление кэша после изменения
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при добавлении товара в базу данных.",
ex);
            }
        }

        /// <summary>
        /// Обновление существующего продукта в базе данных
        /// </summary>
        /// <param name="product">Объект продукта с обновленными данными</param>
        public void UpdateProduct(Product product)
        {
            try
            {
                using (var connection = new SQLiteConnection(_connectionString))
                {
                    connection.Open();
                    // SQL-запрос для обновления продукта по ID
                    var command = new SQLiteCommand(@"
                        UPDATE Product
                        SET Name = @Name, Mass = @Mass, Price = @Price, Amount =
@Amount
                        WHERE Id = @Id", connection);
                    command.Parameters.AddWithValue("@Name", product.Name);
                    command.Parameters.AddWithValue("@Mass", product.Mass);
                    command.Parameters.AddWithValue("@Price", product.Price);
                    command.Parameters.AddWithValue("@Amount", product.Amount);
                    command.Parameters.AddWithValue("@Id", product.Id);
                    command.ExecuteNonQuery();
                }
                LoadProductsToCache(); // Обновление кэша после изменения
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при обновлении товара в базе данных.",
ex);
            }
        }

        /// <summary>
        /// Удаление продукта из базы данных по ID
        /// </summary>
        /// <param name="id">Идентификатор продукта для удаления</param>
        public void DeleteProduct(int id)
        {
            try
            {
                using (var connection = new SQLiteConnection(_connectionString))
                {
                    connection.Open();
                    // SQL-запрос для удаления продукта по ID
```

```
                    var command = new SQLiteCommand("DELETE FROM Product WHERE Id =
@Id", connection);
                    command.Parameters.AddWithValue("@Id", id);
                    command.ExecuteNonQuery();
                }
                LoadProductsToCache(); // Обновление кэша после изменения
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при удалении товара из базы данных.",
ex);
            }
        }

        /// <summary>
        /// Реализация интерфейса IDisposable для освобождения ресурсов
        /// </summary>
        public void Dispose()
        {
            try
            {
                using (var connection = new SQLiteConnection(_connectionString))
                {
                    if (connection.State == System.Data.ConnectionState.Open)
                    {
                        connection.Close(); // Закрытие соединения, если оно открыто
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Ошибка при освобождении ресурсов базы данных.",
ex);
            }
        }
    }
}
```

4. Код файла HelloForm.cs

```
namespace OOP_Curs_Kanevskii_23VP2
{
    partial class HelloForm
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed;
otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code
```

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            labelNAME = new Label();
            label1 = new Label();
            label2 = new Label();
            SuspendLayout();
            //
            // labelNAME
            //
            labelNAME.AutoSize = true;
            labelNAME.Font = new Font("Segoe UI Semibold", 24F, FontStyle.Bold);
            labelNAME.Location = new Point(429, 130);
            labelNAME.Name = "labelNAME";
            labelNAME.Size = new Size(543, 65);
            labelNAME.TabIndex = 0;
            labelNAME.Text = "Каневский Глеб 23ВП2";
            //
            // label1
            //
            label1.AutoSize = true;
            label1.Font = new Font("Segoe UI Semibold", 24F, FontStyle.Bold);
            label1.Location = new Point(441, 308);
            label1.Name = "label1";
            label1.Size = new Size(514, 65);
            label1.TabIndex = 1;
            label1.Text = "База данных: \"Склад\"";
            //
            // label2
            //
            label2.AutoSize = true;
            label2.Font = new Font("Segoe UI Semibold", 24F, FontStyle.Bold);
            label2.Location = new Point(481, 231);
            label2.Name = "label2";
            label2.Size = new Size(406, 65);
            label2.TabIndex = 2;
            label2.Text = "Курсовая работа";
            //
            // HelloForm
            //
            AutoScaleDimensions = new SizeF(10F, 25F);
            AutoScaleMode = AutoScaleMode.Font;
            BackColor = Color.LightSteelBlue;
            ClientSize = new Size(1378, 544);
            Controls.Add(label2);
            Controls.Add(label1);
            Controls.Add(labelNAME);
            MaximumSize = new Size(1400, 600);
            MinimumSize = new Size(1400, 600);
            Name = "HelloForm";
            Text = "Каневский Глеб 23ВП2 БД \"СКЛАД\"";
            ResumeLayout(false);
            PerformLayout();
        }

        #endregion

        private Label labelNAME;
        private Label label1;
        private Label label2;
    }
}
```

5. Код файла MainForm.cs

```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Windows.Forms;
using System.Runtime.InteropServices;
using System.Diagnostics;
using System.IO;
using QuestPDF.Fluent;
using QuestPDF.Helpers;
using QuestPDF.Infrastructure;
using System.Globalization;


namespace OOP_Curs_Kanevskii_23VP2
{
    /// <summary>
    /// Главная форма приложения для работы с базой данных товаров
    /// </summary>
    public partial class MainForm : Form
    {
        private DatabaseContext _dbContext; // Контекст базы данных

        /// <summary>
        /// Конструктор главной формы
        /// </summary>
        public MainForm()
        {
            InitializeComponent();
            InitializeMenu();
            // Инициализация подключения к SQLite базе данных
            _dbContext = new DatabaseContext("Data Source=products.db");
            _dbContext.InitializeDatabase(); // Создание таблиц, если их нет
            LoadData(); // Первоначальная загрузка данных
        }

        /// <summary>
        /// Обработчик ввода для текстовых полей, разрешающий только цифры и точку
        /// </summary>
        private void txtBoxWithoutCharacters_KeyPress(object sender,
KeyPressEventArgs e)
        {
            // Разрешаем цифры, точку и управляющие клавиши (например, Backspace)
            if (!char.IsDigit(e.KeyChar) && e.KeyChar != '.' &&
!char.IsControl(e.KeyChar))
            {
                e.Handled = true; // Игнорируем ввод
            }

            // Запрещаем ввод более одной точки
            if (e.KeyChar == '.' && (sender as TextBox).Text.Contains("."))
            {
                e.Handled = true;
            }
        }

        /// <summary>
        /// Обработчик изменения выбранной строки в DataGridView
        /// </summary>
        private void DataGridView_SelectionChanged(object sender, EventArgs e)
        {
            if (dataGridView.SelectedRows.Count > 0)
            {
                var selectedRow = dataGridView.SelectedRows[0];
```

```
                // Заполняем поля формы данными из выбранной строки
                textBoxName.Text =
selectedRow.Cells["dataGridViewTextBoxColumn2"].Value.ToString();
                textBoxMass.Text =
selectedRow.Cells["dataGridViewTextBoxColumn3"].Value.ToString();
                textBoxPrice.Text =
selectedRow.Cells["dataGridViewTextBoxColumn4"].Value.ToString();
                numericUpDownAmount.Value =
Convert.ToInt32(selectedRow.Cells["dataGridViewTextBoxColumn5"].Value);
            }
        }

        /// <summary>
        /// Инициализация состояния пунктов меню
        /// </summary>
        private void InitializeMenu()
        {
            deleteDatabaseToolStripMenuItem.Enabled = false; // Удаление недоступно
            openDatabaseToolStripMenuItem.Enabled = true; // Открытие доступно
            createDatabaseToolStripMenuItem.Enabled = true; // Создание доступно
        }

        /// <summary>
        /// Обработчик загрузки формы
        /// </summary>
        private void ClientForm_Load(object sender, EventArgs e)
        {
            comboBoxFilter.SelectedIndex = 0; // Устанавливаем фильтр по умолчанию

            // Загружаем данные только если база данных открыта
            if (_dbContext != null)
            {
                LoadData();
            }
        }

        /// <summary>
        /// Загрузка данных из базы в DataGridView
        /// </summary>
        private void LoadData()
        {
            if (_dbContext == null || _dbContext.CachedProducts == null)
            {
                dataGridView.Rows.Clear();
                return;
            }

            try
            {
                dataGridView.Rows.Clear();
                foreach (var product in _dbContext.CachedProducts)
                {
                    // Форматируем числа для корректного отображения
                    string mass = product.Mass.ToString("F3",
CultureInfo.InvariantCulture);
                    string price = product.Price.ToString("F2",
CultureInfo.InvariantCulture);
                    string amount =
product.Amount.ToString(CultureInfo.InvariantCulture);

                    dataGridView.Rows.Add(product.Id, product.Name, mass, price,
amount);
                }
            }
            catch (Exception ex)
```

```
            {
                MessageBox.Show($"Ошибка при загрузке данных: {ex.Message}",
"Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        /// <summary>
        /// Обработчик кнопки добавления нового товара
        /// </summary>
        private void btnAdd_Click(object sender, EventArgs e)
        {
            // Проверка заполнения обязательных полей
            if (string.IsNullOrEmpty(textBoxMass.Text) ||
string.IsNullOrEmpty(textBoxPrice.Text))
            {
                MessageBox.Show("Заполните все поля.", "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            try
            {
                // Создаем новый товар на основе введенных данных
                var product = new Product
                {
                    Name = textBoxName.Text,
                    Mass = double.Parse(textBoxMass.Text,
CultureInfo.InvariantCulture),
                    Price = double.Parse(textBoxPrice.Text,
CultureInfo.InvariantCulture),
                    Amount = (int)numericUpDownAmount.Value
                };

                _dbContext.AddProduct(product); // Добавляем в базу данных
                LoadData(); // Обновляем отображение
            }
            catch (FormatException)
            {
                MessageBox.Show("Некорректный формат числа. Используйте точку как
разделитель.",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка при добавлении записи: {ex.Message}",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        /// <summary>
        /// Обработчик нажатия кнопки "Редактировать" - обновляет выбранный товар в
базе данных
        /// </summary>

private void btnEdit_Click(object sender, EventArgs e)
{
    // Проверяем, что выбрана строка для редактирования
    if (dataGridView.SelectedRows.Count > 0)
    {
        var selectedRow = dataGridView.SelectedRows[0];
        try
        {
            // Создаем объект товара с обновленными данными из формы
            var product = new Product
```

```
            {
                Id = (int)selectedRow.Cells[0].Value, // Берем ID из выбранной
строки
                Name = textBoxName.Text,
                Mass = double.Parse(textBoxMass.Text, CultureInfo.InvariantCulture),
                Price = double.Parse(textBoxPrice.Text,
CultureInfo.InvariantCulture),
                Amount = (int)numericUpDownAmount.Value
            };

            _dbContext.UpdateProduct(product); // Обновляем товар в базе
            LoadData(); // Перезагружаем данные для отображения
        }
        catch (FormatException)
        {
            MessageBox.Show("Некорректный формат числа. Используйте точку как
разделитель.",
                "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        catch (Exception ex)
        {
            MessageBox.Show($"Ошибка при редактировании записи: {ex.Message}",
                "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
}

/// <summary>
/// Обработчик нажатия кнопки "Удалить" - удаляет выбранный товар из базы данных
/// </summary>
private void btnDelete_Click(object sender, EventArgs e)
{
    // Проверяем, что выбрана строка для удаления
    if (dataGridView.SelectedRows.Count > 0)
    {
        var selectedRow = dataGridView.SelectedRows[0];

        // Проверяем, что это не новая пустая строка
        if (selectedRow.IsNewRow)
        {
            MessageBox.Show("Выберите строку для удаления.",
                "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
        }

        // Получаем ID товара для удаления
        int id = (int)selectedRow.Cells[0].Value;
        _dbContext.DeleteProduct(id); // Удаляем из базы
        LoadData(); // Обновляем отображение
    }
    else
    {
        MessageBox.Show("Выберите строку для удаления.",
            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

/// <summary>
/// Обработчик нажатия кнопки "Фильтровать" - фильтрует товары по выбранному
критерию
/// </summary>
private void btnFilter_Click(object sender, EventArgs e)
{
    string filterValue = txtFilterValue.Text; // Значение для фильтрации
```

```
    string filterColumn = comboBoxFilter.SelectedItem.ToString(); // Столбец для
фильтрации

    List<Product> products = new List<Product>();

    // Применяем фильтр в зависимости от выбранного столбца
    switch (filterColumn)
    {
        case "ID":
            products = _dbContext.CachedProducts.Where(f =>
f.Id.ToString().Contains(filterValue)).ToList();
            break;
        case "Название":
            products = _dbContext.CachedProducts.Where(f =>
f.Name.Contains(filterValue, StringComparison.OrdinalIgnoreCase)).ToList();
            break;
        case "Масса шт (кг)":
            products = _dbContext.CachedProducts.Where(f =>
f.Mass.ToString().Contains(filterValue)).ToList();
            break;
        case "Цена шт (руб)":
            products = _dbContext.CachedProducts.Where(f =>
f.Price.ToString().Contains(filterValue)).ToList();
            break;
        case "Всего шт":
            products = _dbContext.CachedProducts.Where(f =>
f.Amount.ToString().Contains(filterValue)).ToList();
            break;
    }

    // Обрабатываем случаи, когда ничего не найдено
    if (products.Count == 0)
    {
        MessageBox.Show("Совпадений не найдено.",
            "Результат поиска", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        // Отображаем отфильтрованные данные
        dataGridView.Rows.Clear();
        foreach (var product in products)
        {
            dataGridView.Rows.Add(product.Id, product.Name,
                product.Mass.ToString("F3", CultureInfo.InvariantCulture),
                product.Price.ToString("F2", CultureInfo.InvariantCulture),
                product.Amount);
        }

        // Показываем количество найденных записей
        textBoxFilter.Visible = true;
        labelFilter.Visible = true;
        textBoxFilter.Text = products.Count.ToString();
    }
}

/// <summary>
/// Обработчик нажатия кнопки "Очистить фильтр" - сбрасывает фильтрацию
/// </summary>
private void btnClearFilter_Click(object sender, EventArgs e)
{
    txtFilterValue.Clear(); // Очищаем поле фильтра
    textBoxFilter.Text = "0"; // Сбрасываем счетчик
    textBoxFilter.Visible = false; // Скрываем элементы фильтра
    labelFilter.Visible = false;
    LoadData(); // Загружаем полный список товаров
```

```
}
/// <summary>
/// Обработчик нажатия кнопки "Сортировать" - сортирует данные по выбранному столбцу
/// </summary>
private void btnSort_Click(object sender, EventArgs e)
{
    // Проверяем, что база данных подключена
    if (_dbContext == null)
    {
        MessageBox.Show("База данных не открыта.",
            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    string sortColumn = comboBoxFilter.SelectedItem.ToString(); // Столбец для
сортировки
    string sortOrder = comboBoxSortOrder.SelectedItem?.ToString(); // Направление
сортировки

    var ProductList = _dbContext.CachedProducts;

    // Применяем сортировку в зависимости от выбранного столбца и направления
    switch (sortColumn)
    {
        case "ID":
            ProductList = sortOrder == "По возрастанию"
                ? ProductList.OrderBy(f => f.Id).ToList()
                : ProductList.OrderByDescending(f => f.Id).ToList();
            break;

        case "Название":
            ProductList = sortOrder == "От А до Я"
                ? ProductList.OrderBy(f => f.Name).ToList()
                : ProductList.OrderByDescending(f => f.Name).ToList();
            break;

        case "Масса шт (кг)":
            ProductList = sortOrder == "По возрастанию"
                ? ProductList.OrderBy(f => f.Mass).ToList()
                : ProductList.OrderByDescending(f => f.Mass).ToList();
            break;

        case "Цена шт (руб)":
            ProductList = sortOrder == "По возрастанию"
                ? ProductList.OrderBy(f => f.Price).ToList()
                : ProductList.OrderByDescending(f => f.Price).ToList();
            break;
        case "Всего шт":
            ProductList = sortOrder == "По возрастанию"
                ? ProductList.OrderBy(f => f.Amount).ToList()
                : ProductList.OrderByDescending(f => f.Amount).ToList();
            break;

        default:
            MessageBox.Show("Неверный столбец для сортировки.",
                "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            return;
    }

    // Обновляем DataGridView отсортированными данными
    dataGridView.Rows.Clear();
    foreach (var product in ProductList)
    {
        dataGridView.Rows.Add(product.Id, product.Name,
            product.Mass.ToString("F3", CultureInfo.InvariantCulture),
```

```
            product.Price.ToString("F2", CultureInfo.InvariantCulture),
            product.Amount);
    }
}

/// <summary>
/// Обработчик изменения выбранного элемента в комбобоксе фильтра
/// Обновляет варианты сортировки в зависимости от выбранного столбца
/// </summary>
private void comboBoxFilter_SelectedIndexChanged(object sender, EventArgs e)
{
    // Для столбца "Название" используем текстовые варианты сортировки
    if (comboBoxFilter.SelectedItem.ToString() == "Название")
    {
        comboBoxSortOrder.Items.Clear();
        comboBoxSortOrder.Items.AddRange(new object[] { "От А до Я", "От Я до А" });
    }
    else // Для числовых столбцов используем стандартные варианты сортировки
    {
        comboBoxSortOrder.Items.Clear();
        comboBoxSortOrder.Items.AddRange(new object[] { "По возрастанию", "По
убыванию" });
    }

    comboBoxSortOrder.SelectedIndex = 0; // Устанавливаем первый элемент по
умолчанию
}

/// <summary>
/// Обработчик кнопки поиска - выполняет поиск по точному совпадению в выбранном
столбце
/// </summary>
private void buttonSearch_Click(object sender, EventArgs e)
{
    // Проверяем подключение к базе данных
    if (_dbContext == null)
    {
        MessageBox.Show("База данных не открыта.", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
        return;
    }

    // Проверяем наличие текста для поиска
    if (string.IsNullOrWhiteSpace(textBoxSearch.Text))
    {
        MessageBox.Show("Введите значение для поиска.", "Предупреждение",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        return;
    }

    string searchText = textBoxSearch.Text.Trim().ToLower(); // Нормализуем
поисковый запрос
    string filterColumn = comboBoxFilter.SelectedItem?.ToString();

    // Проверяем, что выбран столбец для поиска
    if (string.IsNullOrEmpty(filterColumn))
    {
        MessageBox.Show("Выберите столбец для поиска.", "Ошибка",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    // Определяем индекс столбца по его названию
    int columnIndex = filterColumn switch
    {
```

```
        "ID" => 0,
        "Название" => 1,
        "Масса шт (кг)" => 2,
        "Цена шт (руб)" => 3,
        "Всего шт" => 4,
        _ => -1
    };

    if (columnIndex == -1)
    {
        MessageBox.Show("Неверное имя столбца.", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        return;
    }

    // Сброс цвета всех строк перед новым поиском
    foreach (DataGridViewRow row in dataGridView.Rows)
    {
        row.DefaultCellStyle.BackColor = System.Drawing.Color.White;
    }

    // Поиск и подсветка строк с точным совпадением
    int count = 0;
    foreach (DataGridViewRow row in dataGridView.Rows)
    {
        if (row.IsNewRow) continue; // Пропускаем пустую строку для добавления

        var cellValue = row.Cells[columnIndex].Value?.ToString()?.ToLower();
        if (cellValue == searchText)
        {
            row.DefaultCellStyle.BackColor = System.Drawing.Color.LightGreen; //
Подсвечиваем найденные строки
            count++;
        }
    }

    // Выводим результаты поиска
    if (count == 0)
    {
        MessageBox.Show("Совпадений не найдено.", "Результат поиска",
MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        labelFilter.Visible = true;
        textBoxFilter.Visible = true;
        textBoxFilter.Text = count.ToString(); // Показываем количество найденных
записей
    }
}

        /// <summary>
        /// Обработчик пункта меню "Открыть базу данных"
        /// Позволяет выбрать существующий файл базы данных SQLite
        /// </summary>
        private void OpenDatabaseToolStripMenuItem_Click(object sender, EventArgs e)
        {
            using (var openFileDialog = new OpenFileDialog())
            {
                openFileDialog.Filter = "SQLite Database (*.sqlite)|*.sqlite|All
Files (*.*)|*.*";
                openFileDialog.Title = "Выберите файл базы данных";

                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
```

```
                    string selectedFile = openFileDialog.FileName;

                    try
                    {
                        // Создаем новое подключение к выбранной базе данных
                        _dbContext = new DatabaseContext($"Data
Source={selectedFile};Version=3;");
                        _dbContext.InitializeDatabase(); // Проверяем/создаем
необходимые таблицы
                        LoadData(); // Загружаем данные в таблицу

                        // Активируем элементы управления для работы с данными
                        EnableDataControls(true);

                        deleteDatabaseToolStripMenuItem.Enabled = true; // Разрешаем
удаление базы
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка при открытии базы данных:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        _dbContext = null; // Сбрасываем подключение при ошибке
                    }
                }
            }
        }

        /// <summary>
        /// Обработчик пункта меню "Создать базу данных"
        /// Создает новый файл базы данных SQLite
        /// </summary>
        private void CreateDatabaseToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            using (var saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "SQLite Database (*.sqlite)|*.sqlite|All
Files (*.*)|*.*";
                saveFileDialog.Title = "Создайте новый файл базы данных";

                if (saveFileDialog.ShowDialog() == DialogResult.OK)
                {
                    string newDatabasePath = saveFileDialog.FileName;

                    try
                    {
                        File.Create(newDatabasePath).Close(); // Создаем пустой файл
базы данных
                        _dbContext = new DatabaseContext($"Data
Source={newDatabasePath};Version=3;");
                        _dbContext.InitializeDatabase(); // Инициализируем структуру
таблиц
                        LoadData(); // Загружаем пустую таблицу

                        // Активируем элементы управления для работы с данными
                        EnableDataControls(true);

                        deleteDatabaseToolStripMenuItem.Enabled = true; // Разрешаем
удаление базы
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка при создании базы данных:
{ex.Message}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        _dbContext = null; // Сбрасываем подключение при ошибке
```

```
                    }
                }
            }
        }

        /// <summary>
        /// Включает или отключает элементы управления для работы с данными
        /// </summary>
        /// <param name="enable">True - включить, False - отключить</param>
        private void EnableDataControls(bool enable)
        {
            textBoxName.Enabled = enable;
            textBoxMass.Enabled = enable;
            textBoxPrice.Enabled = enable;
            btnAdd.Enabled = enable;
            btnEdit.Enabled = enable;
            btnDelete.Enabled = enable;
            comboBoxFilter.Enabled = enable;
            txtFilterValue.Enabled = enable;
            btnFilter.Enabled = enable;
            btnClearFilter.Enabled = enable;
            buttonSort.Enabled = enable;
            comboBoxSortOrder.Enabled = enable;
            numericUpDownAmount.Enabled = enable;
            textBoxSearch.Enabled = enable;
            buttonSearch.Enabled = enable;
            textBoxFilter.Visible = false;
            labelFilter.Visible = false;
        }

        /// <summary>
        /// Обработчик пункта меню "Сохранить базу данных как..."
        /// Позволяет сохранить текущую базу данных под новым именем
        /// </summary>
private void SaveAsDatabaseToolStripMenuItem_Click(object sender, EventArgs e)
 {
     // Проверяем, что база данных открыта
     if (_dbContext == null || string.IsNullOrEmpty(_dbContext.ConnectionString))
     {
         MessageBox.Show("База данных не открыта.", "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
         return;
     }

     try
     {
         using (var saveFileDialog = new SaveFileDialog())
         {
             saveFileDialog.Filter = "SQLite Database (*.sqlite)|*.sqlite|All Files
(*.*)|*.*";
             saveFileDialog.Title = "Сохранить базу данных как";

             if (saveFileDialog.ShowDialog() == DialogResult.OK)
             {
                 string newDatabasePath = saveFileDialog.FileName;

                 // Получаем путь к текущей базе данных из строки подключения
                 var connectionStringBuilder = new
SQLiteConnectionStringBuilder(_dbContext.ConnectionString);
                 string currentDatabasePath = connectionStringBuilder.DataSource;

                 // Освобождаем ресурсы перед копированием
                 _dbContext.Dispose();
                 _dbContext = null;
                 SQLiteConnection.ClearAllPools();
```

```
                GC.Collect();
                GC.WaitForPendingFinalizers();
                System.Threading.Thread.Sleep(500); // Даем время системе
освободить файл

                // Проверяем, не заблокирован ли файл другим процессом
                if (IsFileLocked(currentDatabasePath, out string
lockingProcessName))
                {
                    MessageBox.Show($"Файл базы данных используется процессом:
{lockingProcessName}.",
                        "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return;
                }

                // Копируем базу данных в новое место
                File.Copy(currentDatabasePath, newDatabasePath, overwrite: true);

                // Удаляем старый файл, если он существует
                if (File.Exists(currentDatabasePath))
                {
                    File.Delete(currentDatabasePath);
                }

                // Создаем новое подключение к сохраненной базе данных
                _dbContext = new DatabaseContext($"Data
Source={newDatabasePath};Version=3;");
                LoadData(); // Обновляем отображение данных

                MessageBox.Show("База данных успешно сохранена.", "Успех",
                    MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show($"Ошибка при сохранении базы данных: {ex.Message}",
            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

/// <summary>
/// Обработчик пункта меню "Сохранить таблицу как PDF"
/// Экспортирует содержимое DataGridView в PDF файл
/// </summary>

        private void SaveTableAsPDFToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            // Проверяем, есть ли данные для экспорта
            if (dataGridView.Rows.Count == 0)
            {
                MessageBox.Show("Таблица пуста. Нечего сохранять.", "Ошибка",
                    MessageBoxButtons.OK, MessageBoxIcon.Warning);
                return;
            }

            using (var saveFileDialog = new SaveFileDialog())
            {
                saveFileDialog.Filter = "PDF Files (*.pdf)|*.pdf|All Files
(*.*)|*.*";
                saveFileDialog.Title = "Сохранить таблицу как PDF";

                if (saveFileDialog.ShowDialog() == DialogResult.OK)
                {
```

```
                    string pdfFilePath = saveFileDialog.FileName;

                    // Валидация пути к файлу
                    if (string.IsNullOrEmpty(pdfFilePath) ||
Path.GetInvalidPathChars().Any(pdfFilePath.Contains))
                    {
                        MessageBox.Show("Недопустимый путь к файлу.", "Ошибка",
                            MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return;
                    }

                    try
                    {
                        // Удаляем существующий файл, если он есть
                        if (File.Exists(pdfFilePath))
                        {
                            File.Delete(pdfFilePath);
                        }

                        // Настраиваем лицензию для QuestPDF (бесплатная версия)
                        QuestPDF.Settings.License = LicenseType.Community;

                        // Генерируем PDF документ
                        Document.Create(container =>
                        {
                            container.Page(page =>
                            {
                                page.Content().Column(column =>
                                {
                                    // Добавляем заголовок документа
                                    column.Item().Text("Таблица
данных").FontSize(20).Bold();

                                    // Создаем таблицу в PDF
                                    column.Item().Table(table =>
                                    {
                                        // Настраиваем столбцы таблицы
                                        table.ColumnsDefinition(columns =>
                                        {
                                            foreach (DataGridViewColumn column in
dataGridView.Columns)
                                            {
                                                columns.RelativeColumn(); //
Автоматическая ширина столбцов
                                            }
                                        });

                                        // Добавляем заголовки столбцов
                                        table.Header(header =>
                                        {
                                            foreach (DataGridViewColumn column in
dataGridView.Columns)
                                            {

header.Cell().Text(column.HeaderText).Bold();
                                            }
                                        });

                                        // Добавляем данные из DataGridView
                                        foreach (DataGridViewRow row in
dataGridView.Rows)
                                        {
                                            if (row.IsNewRow) continue; //
Пропускаем пустые строки
```

```
                                            foreach (DataGridViewCell cell in
row.Cells)
                                            {
                                                string cellValue =
cell.Value?.ToString() ?? "";
                                                table.Cell().Text(cellValue);
                                            }
                                        }
                                    });
                                });
                            });
                        }).GeneratePdf(pdfFilePath); // Генерируем PDF файл

                        MessageBox.Show("Таблица успешно сохранена в PDF.", "Успех",
                            MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (Exception ex)
                    {
                        MessageBox.Show($"Ошибка при создании PDF:
{ex.Message}\n\nStack Trace:\n{ex.StackTrace}",
                            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                    }
                }
            }
        }

        /// <summary>
        /// Обработчик пункта меню "Удалить базу данных"
        /// Удаляет текущую открытую базу данных
        /// </summary>
        private void DeleteDatabaseToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            try
            {
                // Проверяем, что база данных открыта
                if (_dbContext == null ||
string.IsNullOrEmpty(_dbContext.ConnectionString))
                {
                    MessageBox.Show("База данных не открыта.", "Ошибка",
                        MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return;
                }

                // Получаем путь к файлу базы данных
                var connectionStringBuilder = new
SQLiteConnectionStringBuilder(_dbContext.ConnectionString);
                string databasePath = connectionStringBuilder.DataSource;

                if (!File.Exists(databasePath))
                {
                    MessageBox.Show("Файл базы данных не найден.", "Ошибка",
                        MessageBoxButtons.OK, MessageBoxIcon.Warning);
                    return;
                }

                // Запрашиваем подтверждение удаления
                if (MessageBox.Show("Вы уверены, что хотите удалить текущую базу
данных?",
                    "Подтверждение", MessageBoxButtons.YesNo) == DialogResult.Yes)
                {
                    try
                    {
                        // Освобождаем ресурсы перед удалением
                        _dbContext.Dispose();
```

```
                        _dbContext = null;
                        SQLiteConnection.ClearAllPools();
                        GC.Collect();
                        GC.WaitForPendingFinalizers();
                        System.Threading.Thread.Sleep(500);

                        // Проверяем блокировку файла
                        if (IsFileLocked(databasePath, out string
lockingProcessName))
                        {
                            MessageBox.Show($"Файл базы данных используется
процессом: {lockingProcessName}.",
                                "Ошибка", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
                            return;
                        }

                        // Перемещаем файл во временную папку перед удалением
                        string tempPath = Path.Combine(Path.GetTempPath(),
Path.GetFileName(databasePath));
                        File.Move(databasePath, tempPath);
                        File.Delete(tempPath);

                        // Очищаем интерфейс
                        dataGridView.Rows.Clear();

                        // Обновляем состояние меню
                        deleteDatabaseToolStripMenuItem.Enabled = false;
                        openDatabaseToolStripMenuItem.Enabled = true;
                        createDatabaseToolStripMenuItem.Enabled = true;

                        MessageBox.Show("База данных успешно удалена.", "Успех",
                            MessageBoxButtons.OK, MessageBoxIcon.Information);
                    }
                    catch (IOException ex)
                    {
                        // Восстанавливаем подключение в случае ошибки
                        MessageBox.Show($"Не удалось удалить файл базы данных:
{ex.Message}",
                            "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        _dbContext = new DatabaseContext($"Data
Source={databasePath};Version=3;");
                        LoadData();
                    }
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show($"Ошибка при удалении базы данных: {ex.Message}",
                    "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        /// <summary>
        /// Проверяет, заблокирован ли файл другим процессом
        /// </summary>
        /// <param name="filePath">Путь к проверяемому файлу</param>
        /// <param name="lockingProcessName">Имя процесса, блокирующего файл (если
есть)</param>
        /// <returns>True, если файл заблокирован; иначе False</returns>
        private bool IsFileLocked(string filePath, out string lockingProcessName)
        {
            lockingProcessName = null;

            try
```

```
            {
                // Пытаемся открыть файл с эксклюзивным доступом
                using (FileStream stream = File.Open(filePath, FileMode.Open,
FileAccess.ReadWrite, FileShare.None))
                {
                    stream.Close();
                }
            }
            catch (IOException)
            {
                // Если файл заблокирован, пытаемся определить процесс
                Process[] processes = Process.GetProcesses();
                foreach (Process process in processes)
                {
                    try
                    {
                        if (process.MainWindowHandle != IntPtr.Zero &&
!string.IsNullOrEmpty(process.MainWindowTitle))
                        {
                            foreach (ProcessModule module in process.Modules)
                            {
                                if (module.FileName.Equals(filePath,
StringComparison.OrdinalIgnoreCase))
                                {
                                    lockingProcessName = process.ProcessName;
                                    return true;
                                }
                            }
                        }
                    }
                    catch
                    {
                        // Пропускаем процессы, к которым нет доступа
                        continue;
                    }
                }

                return true; // Файл заблокирован, но процесс не определен
            }

            return false; // Файл доступен для записи
        }
    }
}
```

Приложение Б

Руководство пользователя

(обязательное)

Программа ООП OOP_Curs_Kanevskii_23VP2.exe предназначена для хранения информации о складированных товарах. Программа имеет интуитивно понятный интерфейс и поддерживает такие операции:

1. Открыть БД.

2. Создать БД.

3. Сохранить БД как.

4. Сохранить таблицу в PDF.

5. Удалить текущую БД

6. Добавить новую запись.

7. Удалить запись.

8. Редактировать запись.

9. Фильтровать текущие записи по выбранному полю и значению.

10. Сортировать текущие записи по выбранному полю и значению.

11. Подсветить (поиск) текущие записи по выбранному полю и значению.

Далее описанные необходимый порядок действий для выполнения соответствующих операций:

1. Открыть БД:

Чтобы «Открыть БД» необходимо нажать на вкладку «Файл», далее выбрать «Открыть БД» (рисунок 20).

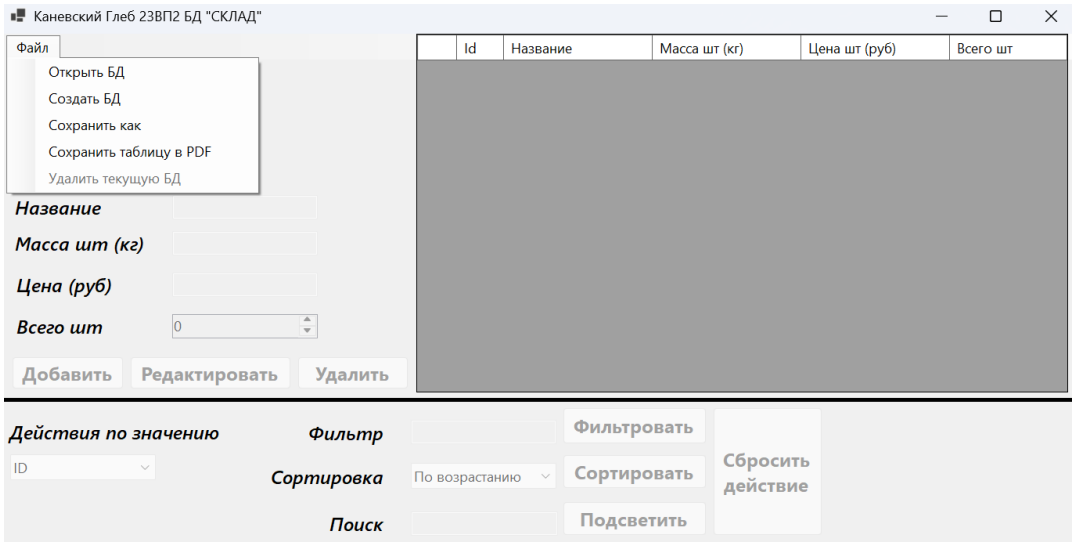

Рисунок 20 – «Открыть БД»

Далее выбираем путь к необходимому файлу и нажимаем «Открыть» (рисунок 21).

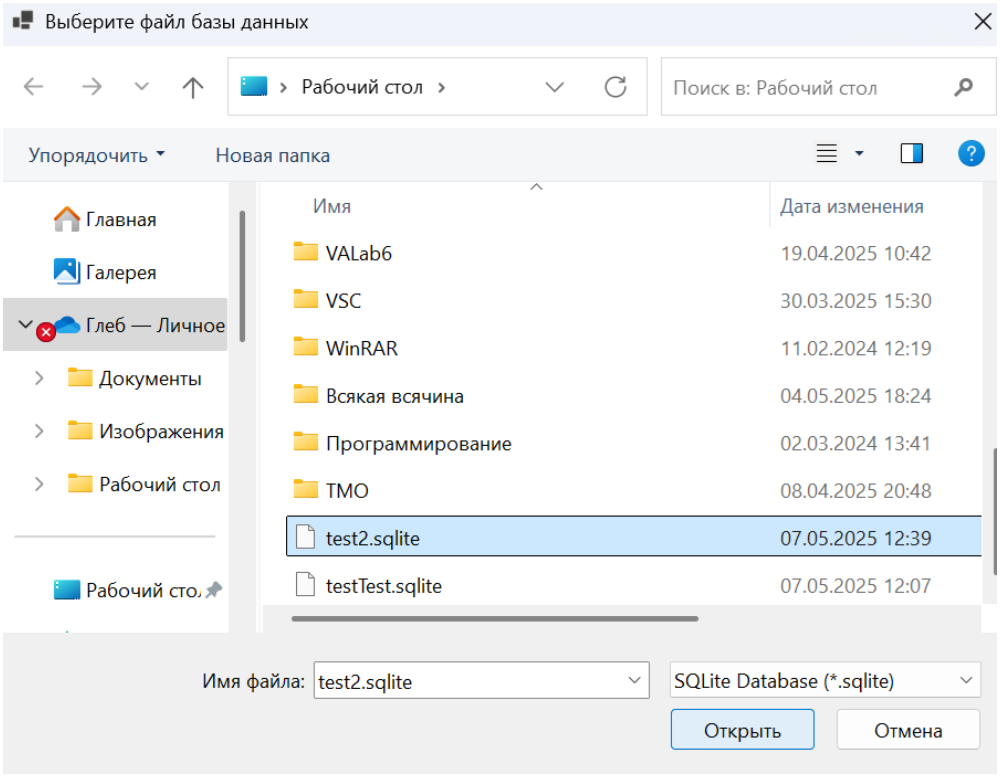

Рисунок 21 – Открыть нужную БД

2. Создать БД:

Чтобы «Создать БД» необходимо нажать на вкладку «Файл», далее выбрать «Создать БД» (рисунок 22). После чего необходимо выбрать расположение и имя новой БД. Нажать «Сохранить». В результате откроется пустая БД.

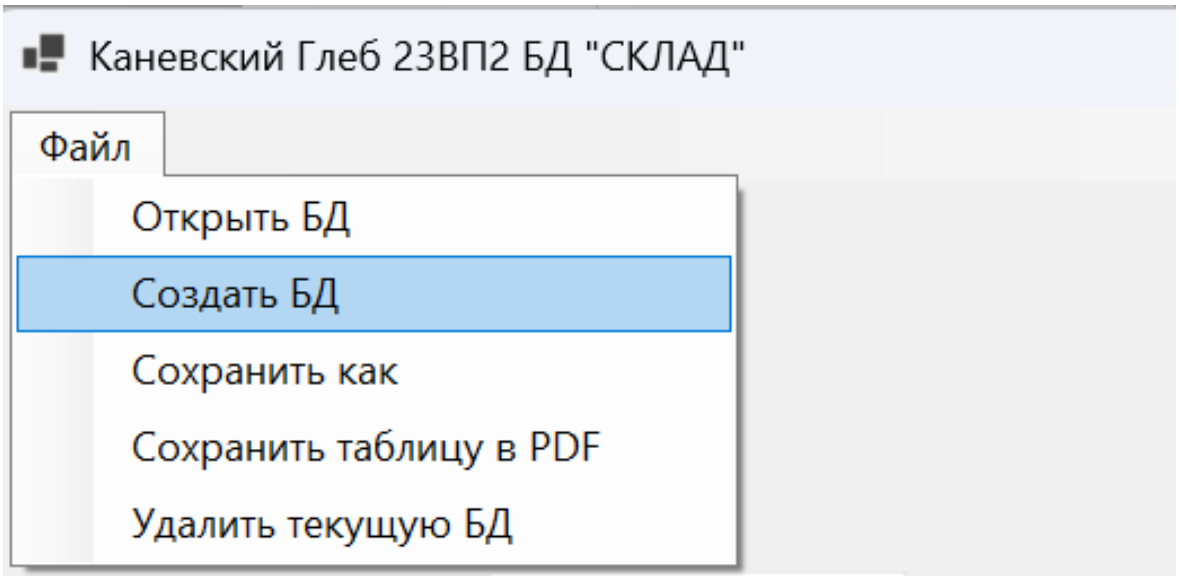

Рисунок 22 – Создать пустую БД

3. Сохранить БД как:

Чтобы «Сохранить БД как» необходимо нажать на вкладку «Файл», далее выбрать «Сохранить БД как» (рисунок 23). После чего необходимо выбрать новое имя БД. Нажать «Сохранить».

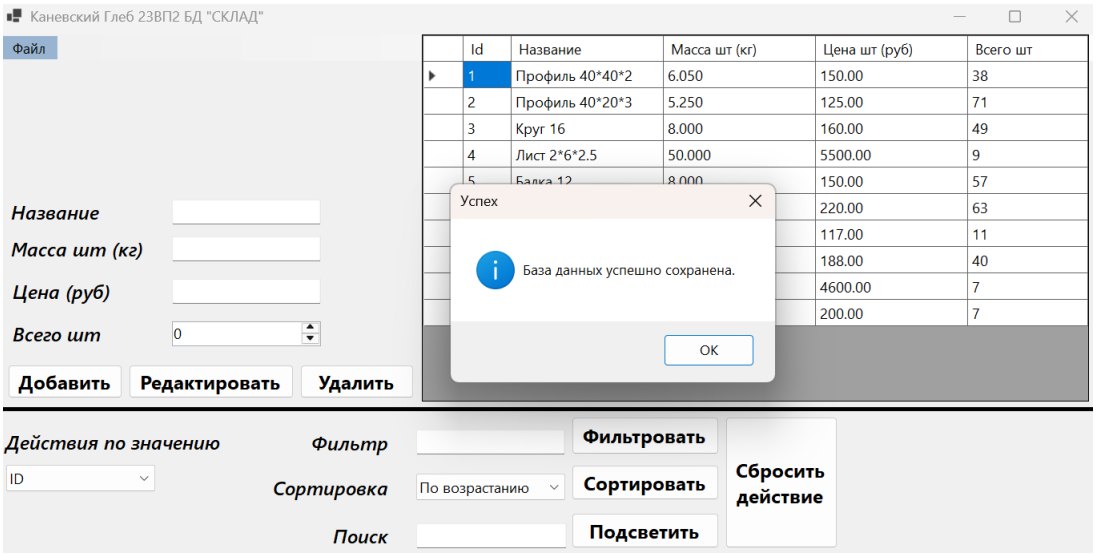

Рисунок 23 – Сохранить БД как

4. Сохранить таблицу в PDF:

Чтобы «Сохранить таблицу в PDF» необходимо нажать на вкладку «Файл», далее выбрать «Сохранить таблицу в PDF» (рисунок 24). После чего необходимо выбрать расположение и имя PDF файла. Нажать «Сохранить».

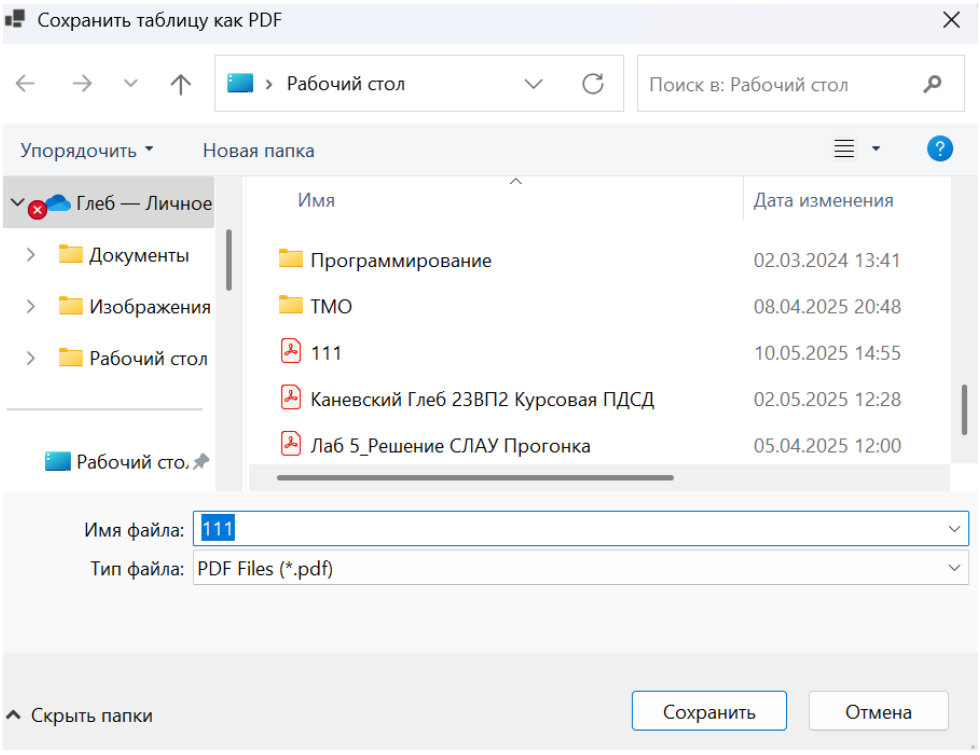

Рисунок 24 – Сохранить таблицу в PDF

5. Удалить текущую БД

Чтобы «Удалить текущую БД» необходимо нажать на вкладку «Файл», далее выбрать «Удалить текущую БД». После чего необходимо подтвердить удаление. Далее получаем результат (рисунок 25).

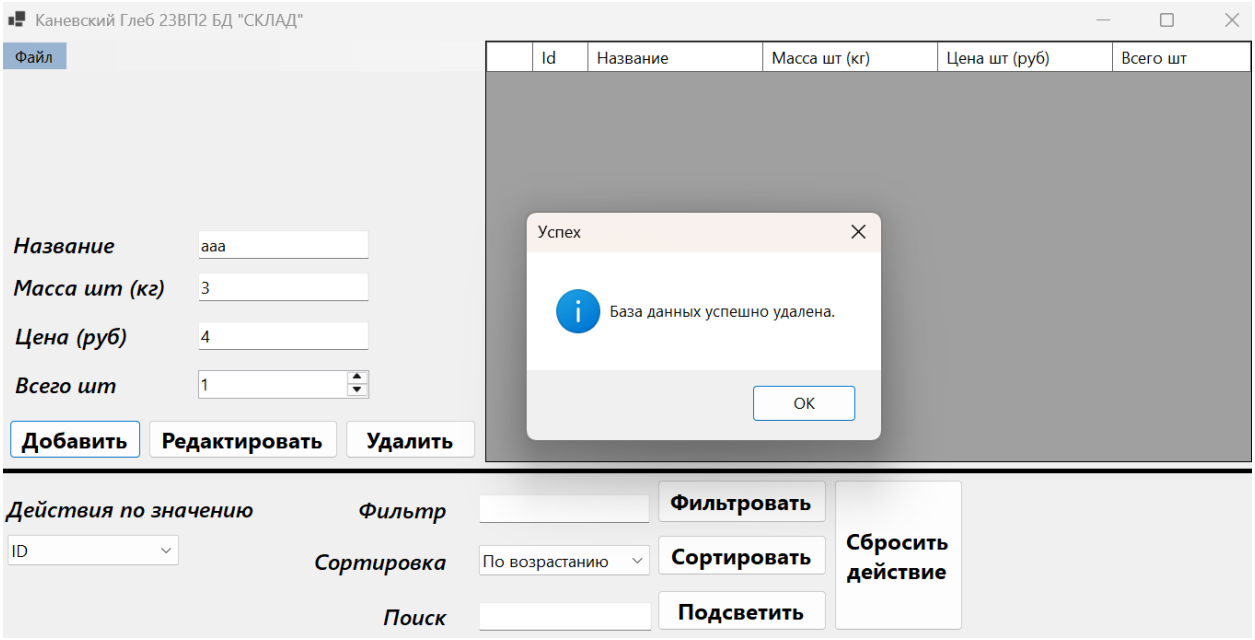

Рисунок 25 – Удалить текущую БД

6. Добавить новую запись.

Для добавления нового товара необходимо ввести данные в поля: название, масса, стоимость, всего шт и нажать «Добавить» (рисунок 26).

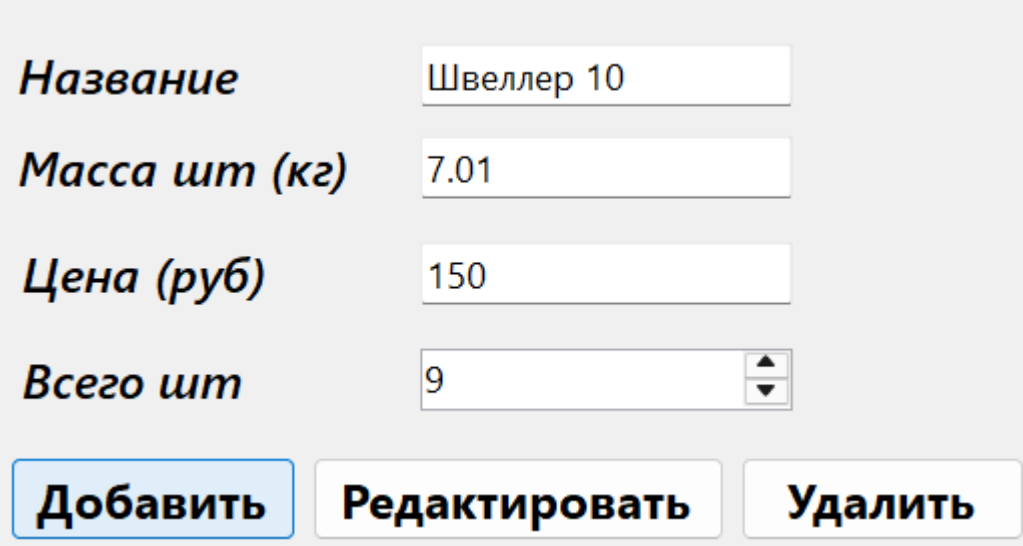

Рисунок 26 – Добавить новую запись

7. Удалить запись.

Для удаления товара необходимо щёлкнуть самому левому столбцу нужного товара. После чего нажать «Удалить» (рисунок 27).

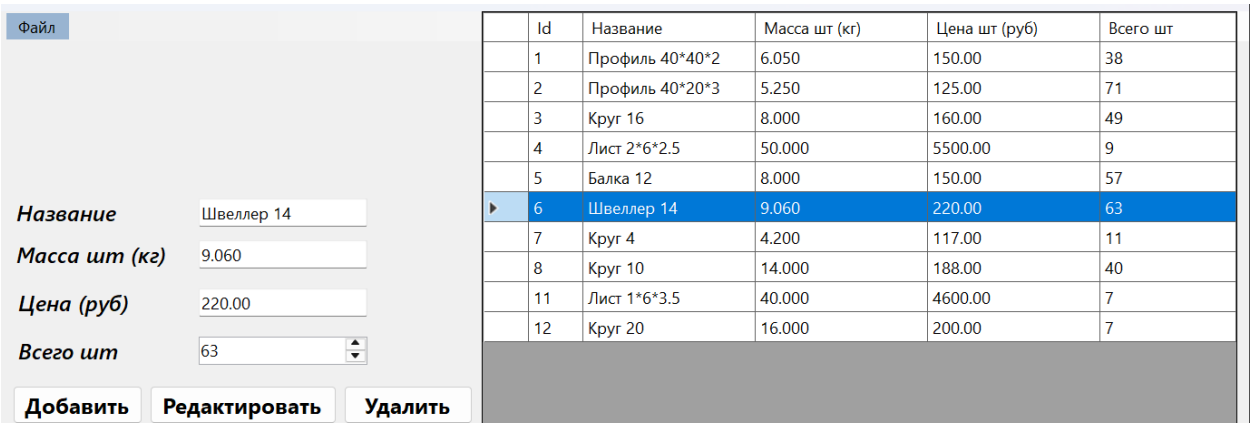

Рисунок 27 – Добавить новую запись

8. Редактировать запись.

Для редактирования товара необходимо щелкнуть по самому левому столбцу нужного товара. После чего автоматический заполнятся поля: название, масса, цена, всего шт. Далее необходимо провести необходимые преобразования и нажать кнопку «Редактировать» (рисунок 28).

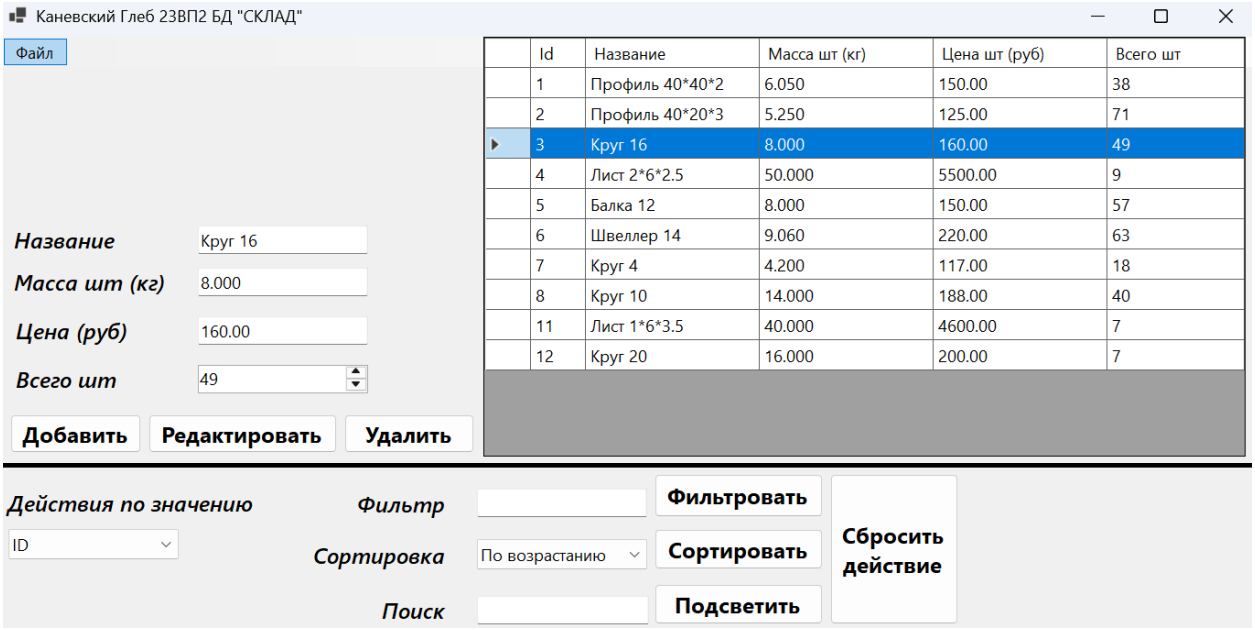

Рисунок 28 – Редактировать запись

9. Фильтровать текущие записи по выбранному полю и значению.

Для фильтрации необходимо:

1. Выбрать поле, по которому произойдет фильтрация (рисунок 29 цифра 1).

2. В поле «Фильтр» ввести значение, по которому произойдет фильтрация (рисунок 29 цифра 2).

3. Результат будет представлен в таблице (рисунок 29 цифра 3).

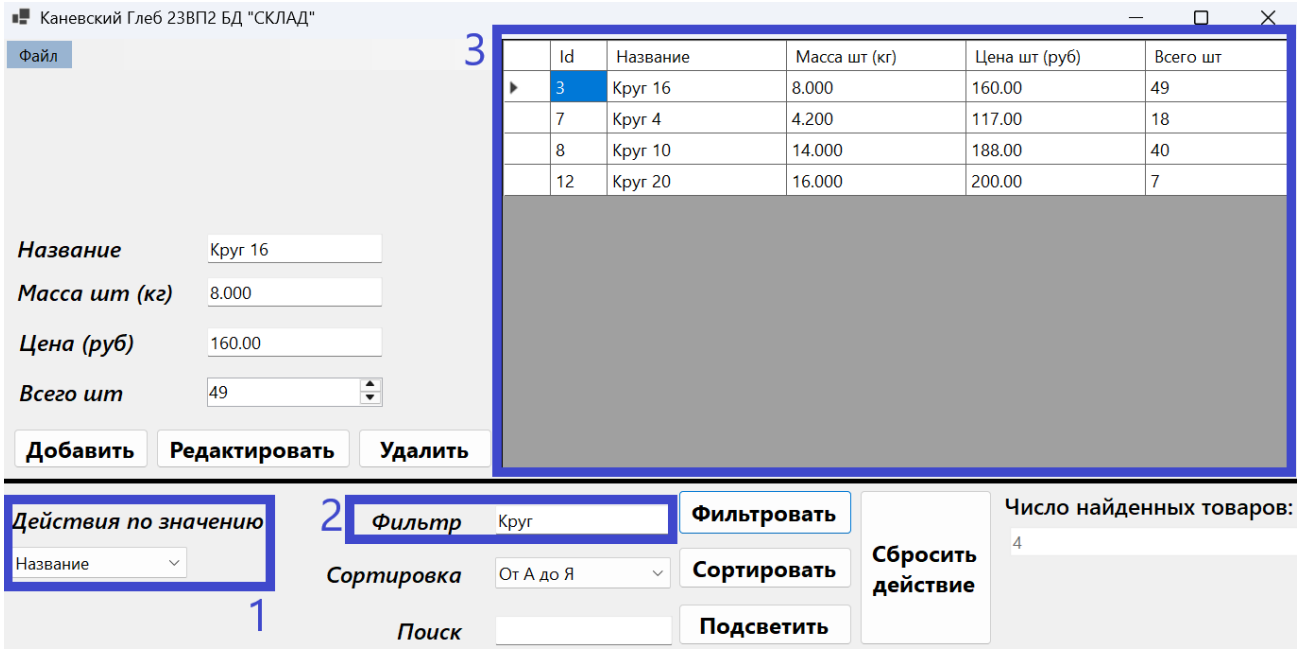

Рисунок 29 – Фильтровать текущие записи по выбранному полю и значению

10. Сортировать текущие записи по выбранному полю и значению.

Для сортировки необходимо:

1. Выбрать поле, по которому произойдет сортировка (рисунок 30 цифра 1).

2. В поле «Сортировка» выбрать порядок, по которому произойдет сортировка (рисунок 30 цифра 2).

3. Результат будет представлен в таблице (рисунок 30 цифра 3).

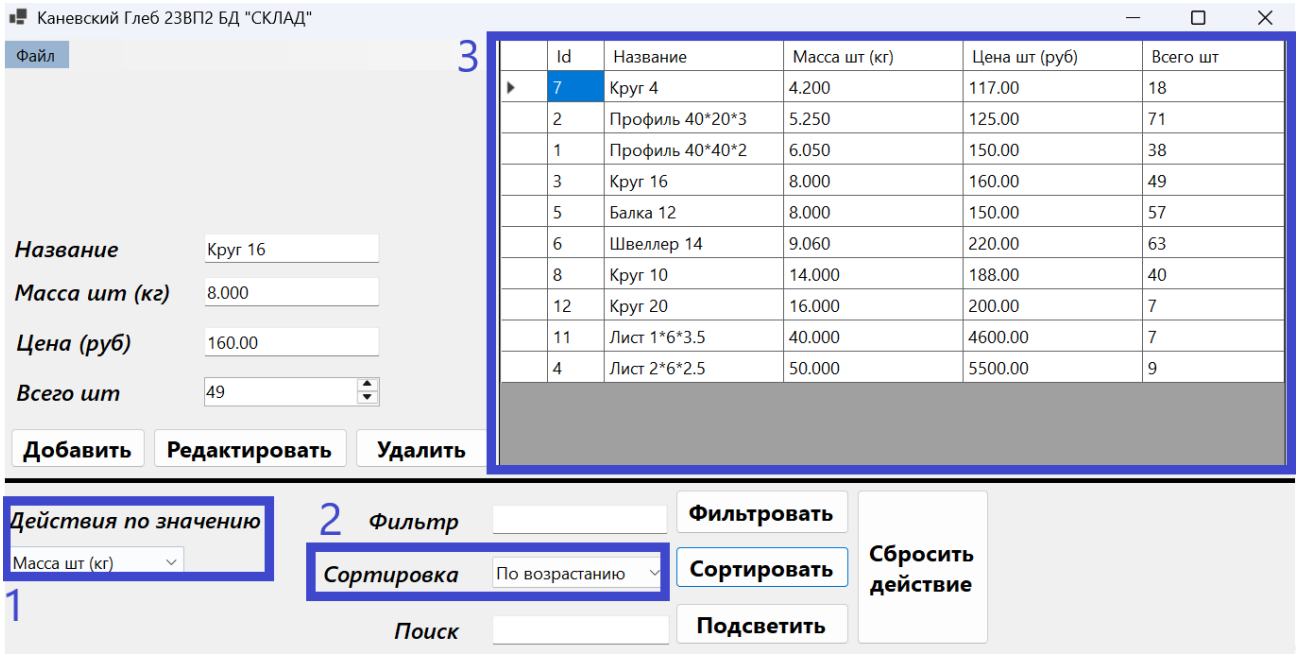

Рисунок 30 – Сортировать текущие записи по выбранному полю и значению

11. Подсветить (поиск) текущие записи по выбранному полю и значению.

Для поиска необходимо:

1. Выбрать поле, по которому произойдет поиск (рисунок 31 цифра 1).

2. В поле «Поиск» выбрать значение, по которому произойдет поиск (рисунок 31 цифра 2).

3. Результат будет представлен в таблице (рисунок 31 цифра 3).

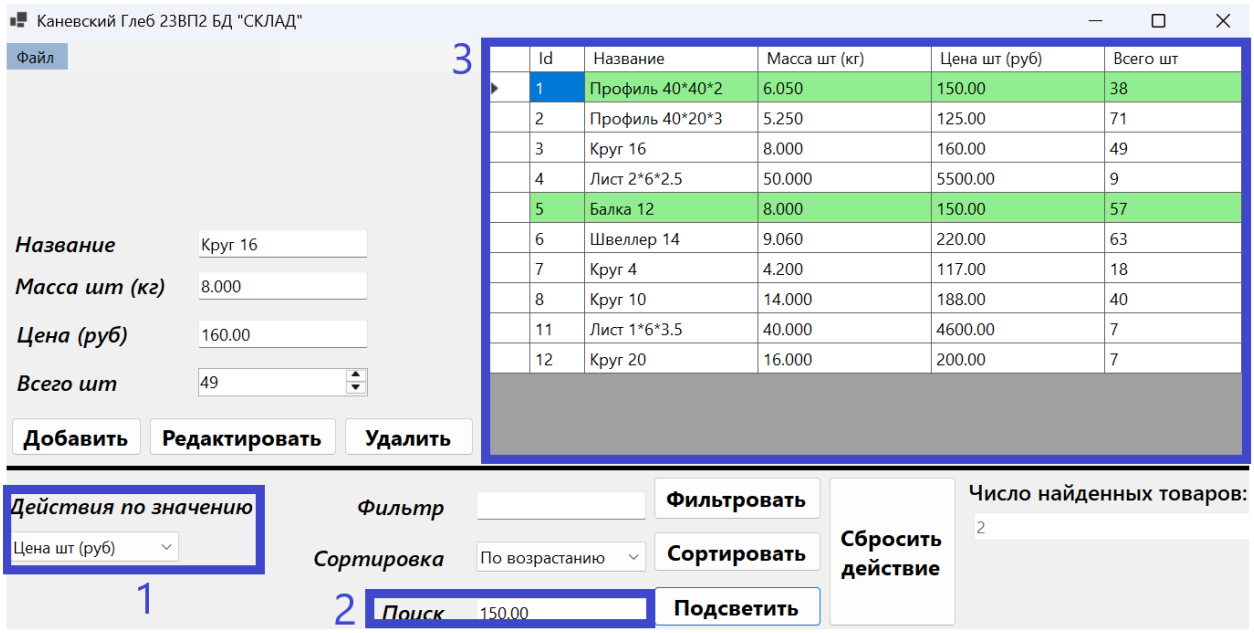

Рисунок 31 – Подсветить (поиск) текущие записи по выбранному полю и значению