**Министерство образования Российской Федерации**

**Пензенский государственный университет**

Кафедра «Математическое обеспечение и применение ЭВМ»

Пояснительная записка к курсовому проекту   
по дисциплине

«Объектно-ориентированное программирование».

Разработка программы с использованием объектно-ориентированного подхода.   
ИС «Канцелярия»

Автор работы:

студенты группы 23ВП2

Муравьев В.С.

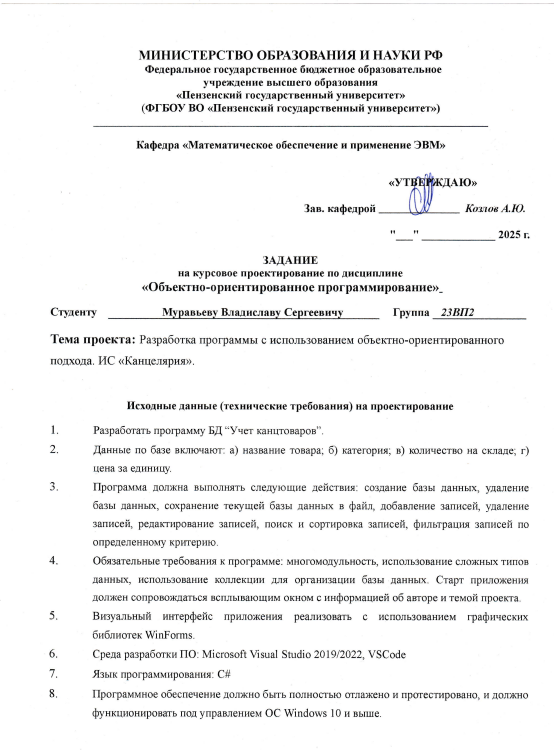
Принял:

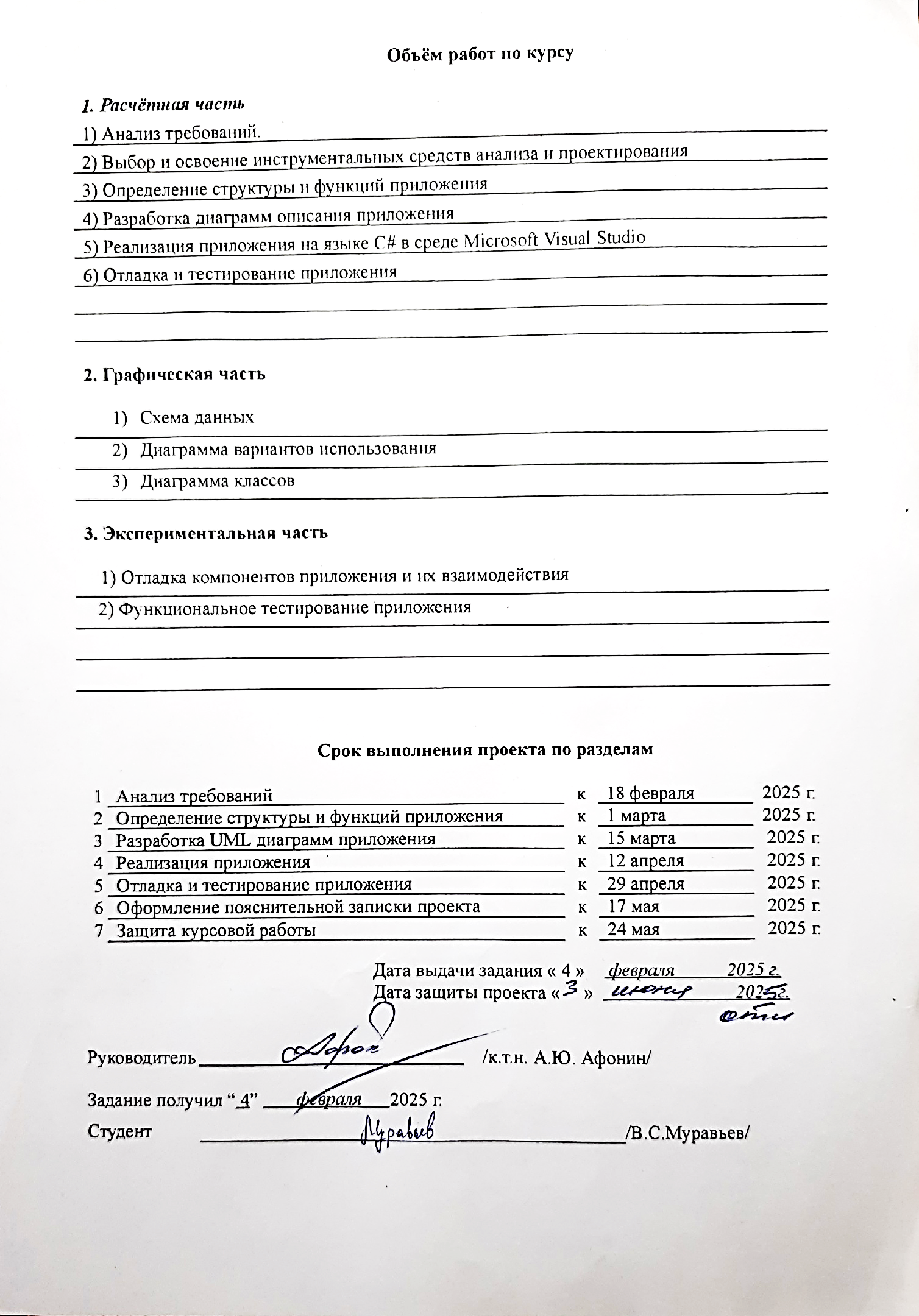
к.т.н. Афонин А.Ю.

Работа сдана \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пенза 2025





**Содержание**

[Введение 5](#_Toc199215312)

[1 Постановка задачи 6](#_Toc199215313)

[2 Выбор решения 7](#_Toc199215314)

[2.1 Определение необходимых модулей программы 8](#_Toc199215315)

[2.2 Определение структуры файла базы данных 12](#_Toc199215316)

[3 Описание разработки программы 13](#_Toc199215317)

[3.1 Кодирование 13](#_Toc199215318)

[3.2 Диаграмма компонентов 14](#_Toc199215319)

[4 Отладка и тестирование 15](#_Toc199215320)

[4.1 Функциональное тестирование 15](#_Toc199215321)

[4.2 Тестирование приложения 15](#_Toc199215322)

[5 Описание программы 21](#_Toc199215323)

[5.1 Разработка приложения KR\_23VP2\_Muraviev.exe 21](#_Toc199215324)

[Выводы 22](#_Toc199215325)

[Список использованных источников 23](#_Toc199215326)

[Приложение A – Код программы 24](#_Toc199215327)

[Приложение B – Руководство пользователя 46](#_Toc199215328)

# 

# Введение

Учет товарных позиций — важная составляющая работы с материально-техническими ресурсами. Это позволяет контролировать наличие товаров, следить за остатками, планировать пополнение и упрощать взаимодействие с ассортиментом. Особенно это актуально в тех сферах, где товары часто обновляются, имеют множество разновидностей и требуют системного подхода к хранению информации.

Канцелярские товары относятся к категории, где количество позиций может быть довольно большим, а потребность в быстрой и точной навигации по ним — критичной. Ведение такого учета вручную, например, в бумажных журналах или в простых таблицах, становится неудобным и подверженным ошибкам по мере роста ассортимента. Поэтому применение автоматизированных информационных систем позволяет повысить эффективность учета, снизить вероятность потерь и упростить повседневную работу.

В рамках данного курсового проектирования будет создано приложение «Канцелярский учет», предназначенное для ведения учета канцелярской продукции.

Разработка информационной системы будет вестись с использованием языка программирования C# в среде Visual Studio с применением технологии Windows Forms.

Исполнительная среда (или платформа) .NET Framework предложена и поддерживается компанией Microsoft как средство для выполнения приложений, компоненты (составные части) которых написаны на разных языках программирования. [1]

Назначение .NET Framework — служить средой для поддержки разработки и выполнения сильно распределенных компонентных приложений. Она обеспечивает совместное использование разных языков программирования, а также безопасность, переносимость программ и общую модель программирования для платформы Windows. [2]

Цель разработки — создать простое, наглядное и удобное решение, позволяющее вести учет канцелярских товаров и обеспечивать быстрый доступ к необходимой информации.

# 1 Постановка задачи

Необходимо разработать программу «Учет канцтоваров». Информация о товаре следующая:

* название товара;
* категория;
* количество на складе;
* цена за единицу.

Для реализации данного пункта необходимо определить структуру базы данных, максимальный размер, размер каждой записи базы данных. Это необходимо для выявления и минимизации ограничений программы.

Программа должна предоставить возможность добавления нового товара в список, удаления товара из списка, редактирования товара в списке, поиска товара в списке, сохранения существующего списка, загрузки списка, сортировки списка, фильтрации списка, удаления списка.

Для реализации этих требований нужно разработать интуитивно понятный интерфейс пользователя, определить названия пунктов меню. Определить тип интерфейса, выделить в консоли зоны меню и таблицы для вывода записей базы данных.

Изучить функции для работы с файлами. Это необходимо для четкого понимания функциональности программы.

# 2 Выбор решения

Для реализации программы «Канцелярский учет» было выбрано приложение с графическим интерфейсом на платформе Windows Forms с использованием языка программирования C# в среде разработки Visual Studio. В качестве базы данных используется SQLite, поскольку она проста в использовании, не требует установки сервера и хорошо подходит для локального хранения информации.

C# – это относительно новый язык программирования, который характеризуется следующими двумя преимуществами:

* он спроектирован и разработан специально для применения с Microsoft .NET Framework;
* это язык, основанный на современной объектно-ориентированной методологии проектирования, при разработке которого специалисты Microsoft опирались на опыт создания подобных языков, построенных в соответствии с объектно-ориентированными принципами, которые были впервые предложены около 40 лет назад. [3]

Язык C# сочетает строгость типизации с удобством объектно-ориентированного программирования, предоставляя разработчику мощные средства для создания надёжных и читаемых приложений [4].

API-интерфейс Windows Forms состоит из сотен типов (классов, интерфейсов, структур, перечислений и делегатов), которые организованы в различные пространства имен сборки System.Windows.Forms.dll [5]

Данное решение выбрано в связи с его удобством для быстрой разработки, доступностью, а также наличием всех необходимых инструментов для реализации функциональности системы.

Перед проектированием структуры базы данных и интерфейса была составлена диаграмма вариантов использования [6] (рисунок 1), отражающая ключевые сценарии взаимодействия пользователя с программой.

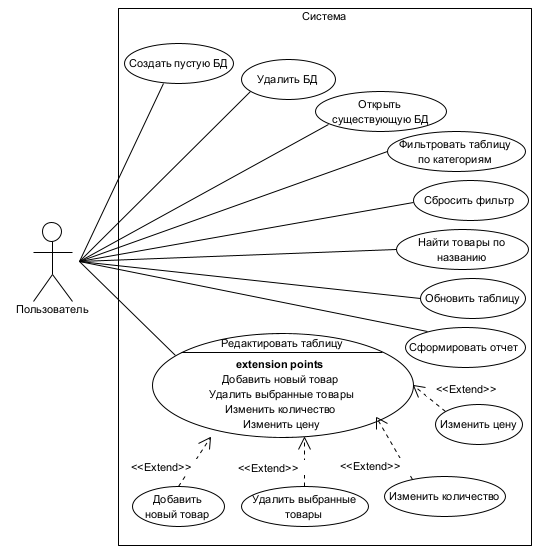


Рисунок 1 – Диаграмма вариантов использования

## **2.1 Определение необходимых модулей программы**

Программа построена по архитектурной модели MVC (Model–View–Controller), что обеспечивает разделение логики представления, управления и обработки данных. MVC состоит из объектов трех видов. Модель — это объект приложения, а представление — его внешний вид на экране. Контроллер описывает, как интерфейс реагирует на управляющие воздействия пользователя. [7]

Применение архитектурного шаблона MVC позволяет отделить бизнес-логику от пользовательского интерфейса, обеспечивая гибкость и упрощая поддержку кода. [8]

Для наглядного представления структуры программы и связей между компонентами ниже на рисунке 2 приведена диаграмма классов [9], отражающая реализацию архитектуры MVC и взаимодействие основных модулей приложения.

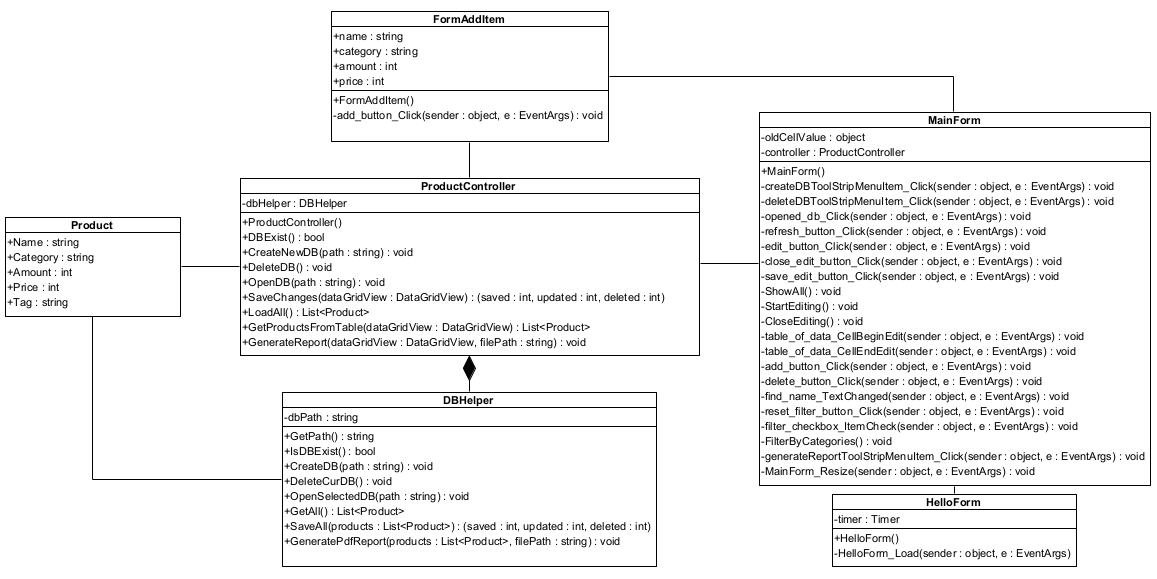


Рисунок 2 – Диаграмма классов

Определения классов с использованием ключевого слова C# class задают внутреннее состояние и поведение, ассоциируемое с объектами типа данного класса. Внутреннее состояние объекта представлено полями, объявляемыми в классе. Поведение объектов определяется реализуемыми методами, которые объявляются и определяются внутри определения класса. [10]

Ниже представлены описания классов.

1. **HelloForm –** всплывающее окно с информацией.

**Основное поле:**

* Timer timer – таймер для автоматического закрытия формы.

**Методы класса:**

* HelloForm() – конструктор формы;
* HelloForm\_Load(object sender, EventArgs e) – загрузка формы.

1. **MainForm** – основной интерфейс пользователя.

**Основные поля:**

* int oldCellValue – предыдущее значение ячейки в таблице. Используется для восстановления значения при некорректном редактировании;
* ProductController controller – контроллер для управления логикой работы с товарами.

**Методы класса:**

* MainForm() – конструктор формы;
* createDBToolStripMenuItem\_Click(object sender, EventArgs e) – создание пустой таблицы базы данных;
* deleteDBToolStripMenuItem\_Click(object sender, EventArgs e) – удаление таблицы базы данных;
* opened\_db\_Click(object sender, EventArgs e) – открытие выбранной таблицы базы данных;
* refresh\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Обновить таблицу»;
* edit\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Редактировать таблицу»;
* close\_edit\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Отменить изменения»;
* save\_edit\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Сохранить изменения»;
* ShowAll() – вывод всех элементов из БД;
* StartEditing() – открыть редактирование;
* CloseEditing() – закрыть редактирование;
* table\_of\_data\_CellBeginEdit(object sender, DataGridViewCellCancelEventArgs e) – сохранение изначального значения кол-ва/цены;
* table\_of\_data\_CellEndEdit(object sender, DataGridViewCellEventArgs e) – проверка на корректность введенного значения кол-ва/цены;
* add\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Добавить новый»;
* delete\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Удалить выбранные»;
* find\_name\_TextChanged(object sender, EventArgs e) – поиск по названию;
* reset\_filter\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Сбросить фильтр»;
* filter\_checkbox\_ItemCheck(object sender, ItemCheckEventArgs e) – смена элемента CheckedListBox;
* FilterByCategories() – фильтровать по выбранным категориям;
* generateReportToolStripMenuItem\_Click(object sender, EventArgs e) – формирование отчета в виде PDF-файла с таблицей базы данных;
* MainForm\_Resize(object sender, EventArgs e) – обработчик изменения размера главной формы.

1. **ProductController** – контроллер для работы с товарами и базой данных.

**Основное поле:**

* DBHelper dbHelper – объект-помощник для работы с базой данных.

**Методы класса:**

* ProductController() – конструктор класса;
* DBExist() – проверяет, существует ли текущая база данных;
* CreateNewDB(string path) – создает новую базу данных;
* DeleteDB() – удаляет текущую базу данных;
* OpenDB(string path) – открывает выбранную базу данных;
* SaveChanges(DataGridView dataGridView) - сохраняет изменения списка товаров в базе данных;
* LoadAll() – загружает все товары из базы данных;
* GetProductsFromTable(DataGridView dataGridView) – извлекает список товаров из таблицы;
* GenerateReport(DataGridView dataGridView, string filePath) – генерирует PDF-отчет по списку товаров и сохраняет его в указанный файл.

1. **DBHelper** – класс-помощник для работы с базой данных SQLite и генерацией PDF-отчетов.

**Основное поле:**

* string dbPath – путь к файлу базы данных SQLite. Используется для подключения к текущей базе данных проекта.

**Методы класса:**

* GetPath() – возвращает текущий путь к базе данных;
* IsDBExist() – проверяет, существует ли база данных по текущему пути;
* CreateDB(string path) – создает новую базу данных по указанному пути. Если файл уже существует – он будет удален. Создается таблица products;
* DeleteCurDB() – удаляет текущую базу данных и сбрасывает путь;
* OpenSelectedDB(string path) – открывает существующую базу данных по указанному пути;
* GetAll() – загружает все записи товаров из базы данных;
* SaveAll(List<Product> products) – сохраняет список товаров в базу данных. Добавляет, обновляет или удаляет записи в зависимости от значения поля Tag у товара;
* GeneratePdfReport(List<Product> products, string filePath) – генерирует PDF-отчет по товарам и сохраняет его в файл.

1. **Product** – класс, представляющий товар на складе.

**Основные поля:**

* string Name – название товара;
* string Category – категория товара;
* int Amount – количество товара на складе;
* int Price – цена за единицу товара;
* string Tag – тег состояния строки (new, edit, delete).

1. **FormAddItem** – класс формы для добавления товаров.

**Основные поля:**

* string name – название товара;
* string category – категория товара;
* int amount – количество товара на складе;
* int price – цена за единицу товара.

**Методы класса:**

* FormAddItem() – конструктор формы;
* add\_button\_Click(object sender, EventArgs e) – нажатие на кнопку «Добавить».

## **2.2 Определение структуры файла базы данных**

SQLite представляет библиотеку, которая написана на языке C (ANSI-C) и которая реализует движок реляционных баз данных. На сегодняшний день SQLite, возможно, самая используемая система баз данных. [11]

В SQLite каждая таблица хранится как B-дерево, где каждый узел — это страница фиксированного размера. Такая структура обеспечивает быструю выборку и эффективное хранение данных даже в условиях ограниченных ресурсов. [12]

В проекте используется база данных SQLite, хранящаяся в локальном файле с расширением «.db». Структура базы данных включает одну основную таблицу, содержащую информацию о товарах. Ее описание представлено в таблице 1.

Таблица 1 – Структура таблицы в базе данных

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип данных** | **Описание** |
| id | INTEGER | Уникальный идентификатор записи (PRIMARY KEY AUTOINCREMENT) |
| name | TEXT | Название товара |
| category | TEXT | Категория товара |
| amount | INTEGER | Количество товара на складе |
| price | INTEGER | Цена за единицу товара |

Основные характеристики:

* Поле id — является первичным ключом и создаётся автоматически;
* Типы TEXT и INTEGER выбраны для удобства хранения текстовых значений и целых чисел (количество и цена);
* Все поля (кроме id) заполняются пользователем вручную при добавлении записи через форму.

Файл базы данных создаётся в директории проекта и может быть открыт или удалён через меню программы. Пользователь имеет возможность создавать новую базу данных, удалять базу данных, выбирать существующую, а также генерировать по ней отчет в виде PDF-файла.

# 3 Описание разработки программы

## **3.1 Кодирование**

В ходе выполнения курсового проектирования было разработано приложение «Учет канцтоваров», код которого приведен в приложении А.

Для алгоритма создания новой базы данных была разработана диаграмма деятельности [13] (рисунок 3).

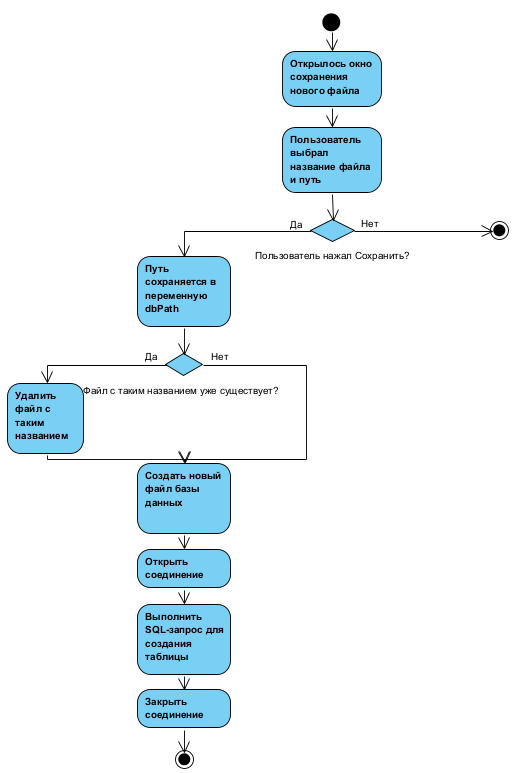


Рисунок 3 – Диаграмма деятельности

На диаграмме мы видим, что после нажатия пользователем на кнопку создания новой базы данных открывается диалоговое окно сохранения файла. После нажатия пользователем кнопки «Сохранить» программа создает новый файл с расширением «.db» (или перезаписывает существующий), открывает соединение для выполнения SQL-запросов, создает новую пустую таблицу БД и закрывает соединение. После чего приложение работает с созданной таблицей базы данных.

## **3.2 Диаграмма компонентов**

В процессе выполнения курсового проекта была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты и связи между ними (рисунок 4). Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами. [14]

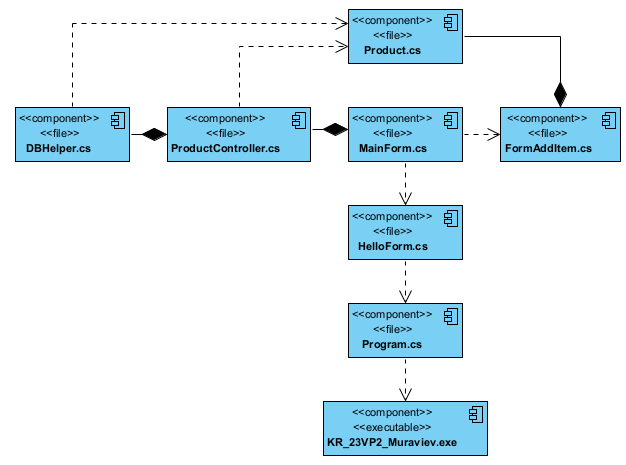


Рисунок 4 – Диаграмма компонентов

Описание компонентов диаграммы приведено в таблице 2.

Таблица 2 – Описание компонентов

|  |  |
| --- | --- |
| **Компонент** | **Назначение** |
| DBHelper.cs | Работа с базой данных |
| ProductController.cs | Логика работы с товарами |
| Product.cs | Данные о товаре |
| FormAddItem.cs | Форма добавления товара |
| MainForm.cs | Главная форма |
| HelloForm.cs | Всплывающее окно с информацией |
| Program.cs | Точка входа в приложение |
| KR\_23VP2\_Muraviev.exe | Исполняемый файл |

# 4 Отладка и тестирование

## **4.1 Функциональное тестирование**

Функциональное тестирование проводится для проверки функциональности приложения. Оно позволяет убедиться в том, что приложение работает корректно и выполняет функции, соответствующие требованиям пользователей и заказчика.

Функциональное тестирование проверяет отдельные функции и возможности приложения, а интеграционное тестирование проверяет взаимодействие компонентов системы в целом.

**Пример.** Веб-приложение для онлайн-бронирования номеров в отеле. Функциональное тестирование поможет убедиться в том, что приложение работает корректно и выполняет свои функции. [15]

## **4.2 Тестирование приложения**

В курсовом проекте было выполнено функциональное тестирование разработанного программного обеспечения. Результаты тестирования приведены в таблице 3.

Таблица 3 – Функциональное тестирование

|  |  |  |
| --- | --- | --- |
| **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| Создать базу данных | Создание новой базы данных и открытие взаимодействия с ней | Создана новая база данных и открыто взаимодействие с ней (рисунок 5) |
| Добавить товар с пустым полем | Появление окна с ошибкой и информацией о ней | Появилось окно с ошибкой и информацией о ней (рисунок 6) |
| Добавить товар с корректными данными | Появление в БД и в таблице на форме созданного товара | Появился в БД и в таблице на форме созданный товар (рисунок 7) |
| Выбрать и удалить несколько товаров | Выбранные пользователем товары будут удалены из БД после подтверждения | Выбранные пользователем товары были удалены из БД (рисунок 8) |
| Изменить цену товара на некорректное значение | Появление окна с ошибкой и информацией о ней, восстановление предыдущего значения | Появилось окно с ошибкой и информацией о ней, восстановилось предыдущее значение (рисунок 9) |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| Изменить количество товара на корректное значение | Успешное изменение значения в таблице и БД | Значение было изменено в таблице и БД (рисунок 10) |
| Удалить базу данных | Текущая база данных удалится, таблица на форме очистится | База данных была удалено, таблица на форме очищена (рисунок 11) |
| Открыть базу данных | Пользователь выбирает файл, база данных открывается и отображается в таблице на форме | База данных была открыта, элементы отображены в таблице на форме (рисунок 12) |
| Фильтровать таблицу по категории | Пользователь выбирает критерий фильтрации, в таблице отображаются только элементы, удовлетворяющие критерию | В таблице отображены только элементы, удовлетворяющие выбранному критерию (рисунок 13) |
| Найти товар по названию | Пользователь вводит в строку поиска название искомого товара, в таблице отображаются все товары, имеющие в названии введенную строку | В таблице отображены все товары, имеющие в названии введенную пользователем строку (рисунок 14) |

Ниже, на рисунках 5-14, изображены результаты проведенных тестирований.

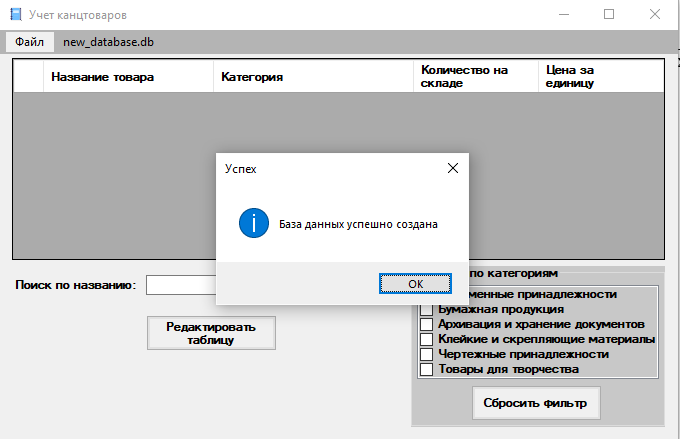


Рисунок 5 – Создать БД

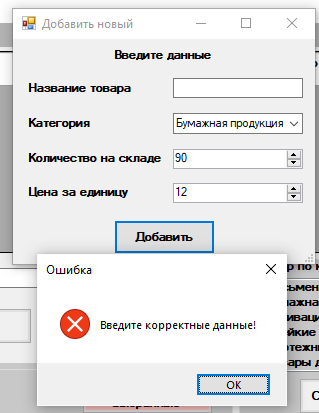


Рисунок 6 – Добавить товар с пустым полем

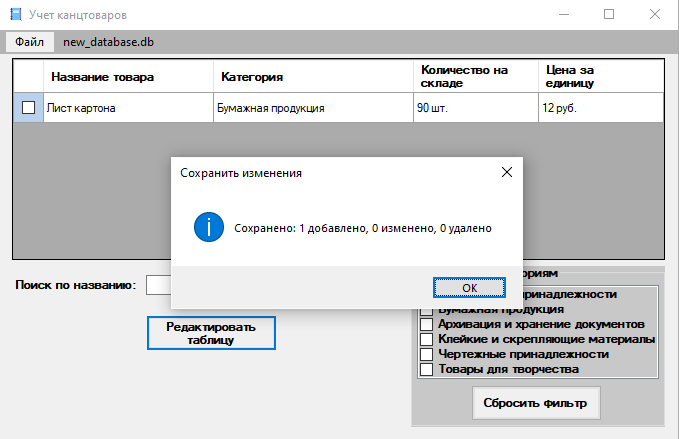


Рисунок 7 – Добавить товар с корректными данными

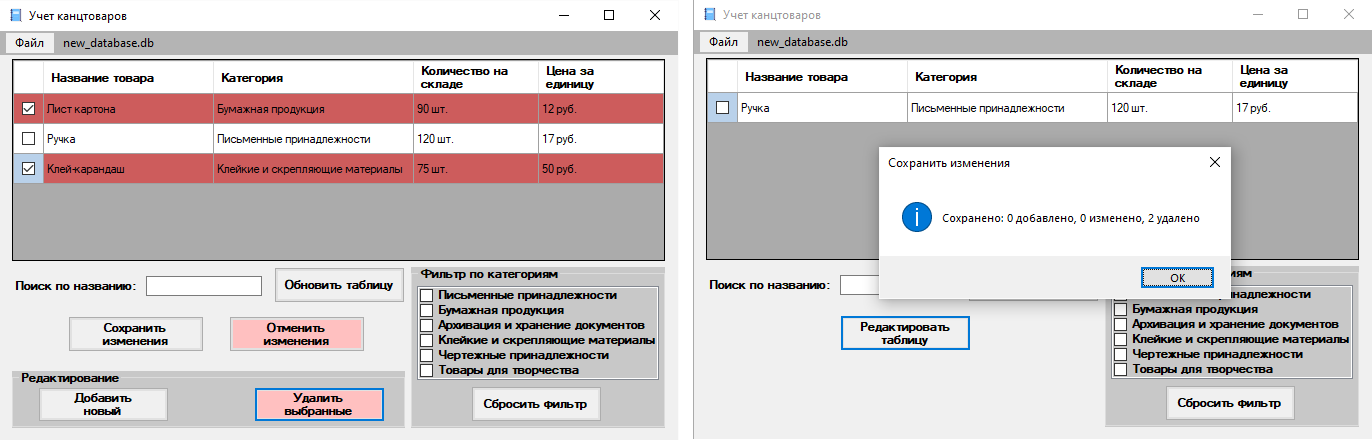


Рисунок 8 – Выбрать и удалить несколько товаров

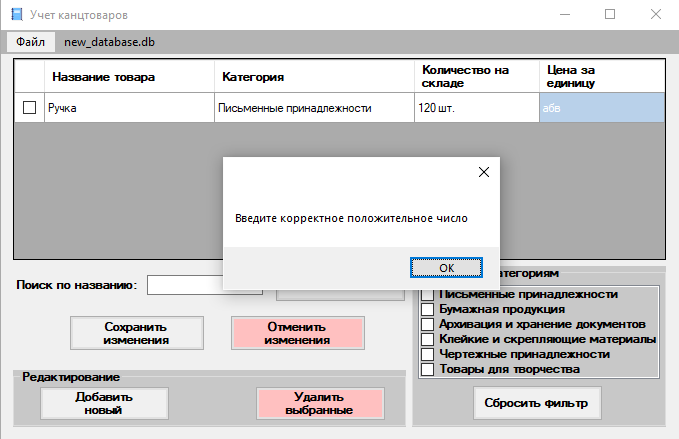


Рисунок 9 – Изменить цену товара на некорректное значение

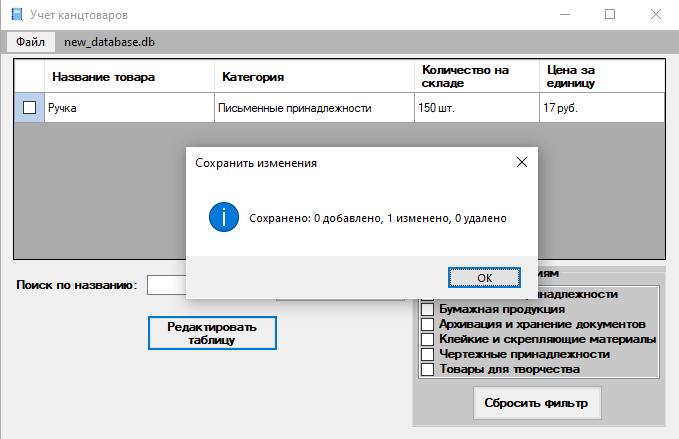


Рисунок 10 – Изменить количество товара на корректное значение

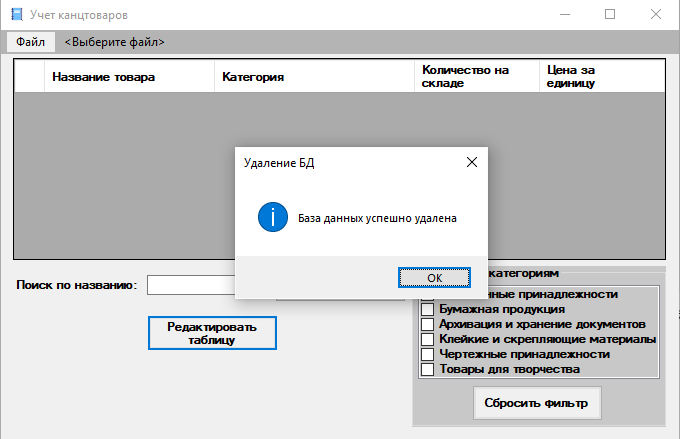


Рисунок 11 – Удалить БД

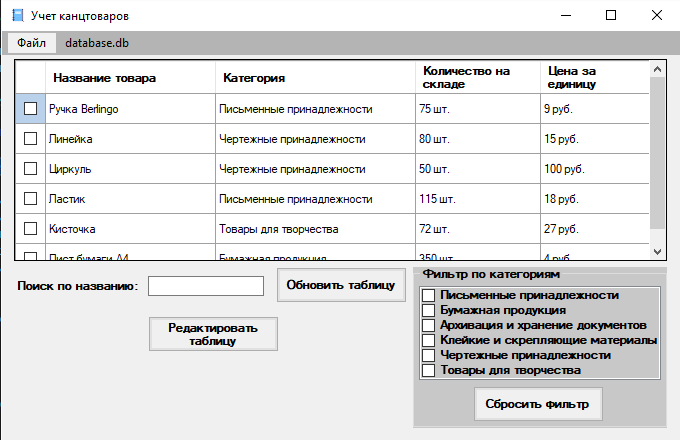


Рисунок 12 – Открыть БД

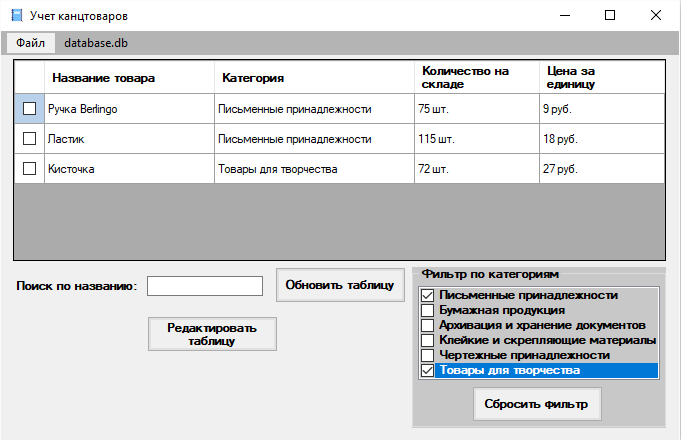


Рисунок 13 – Фильтровать таблицу по категории

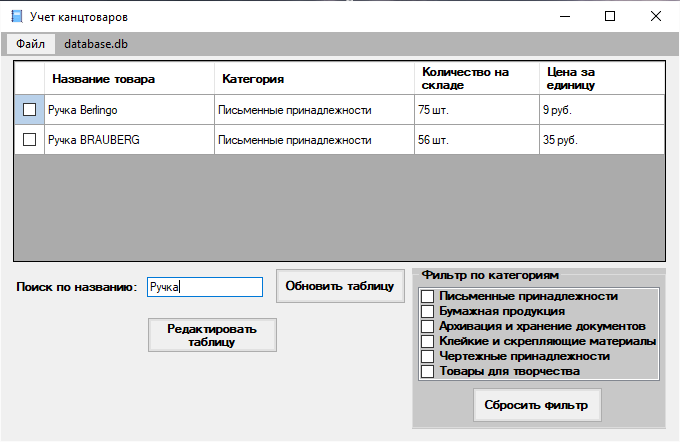


Рисунок 14 – Найти товар по названию

В ходе выполнения тестирования несовпадения ожидаемого и наблюдаемого результата не выявлены. Следовательно, можно сделать вывод, что программа работает корректно.

# 5 Описание программы

## **5.1 Разработка приложения KR\_23VP2\_Muraviev.exe**

Приложение KR\_23VP2\_Muraviev.exe является основным модулем программы. При запуске программы происходит вывод заставки - приветствия, после чего программа готова к выполнению своих функций. Происходит ожидание нажатия соответствующего пункта меню. После нажатия нужного пункта управление переходит в функции, реализованные в файле MainForm.cs. Описание всех состояний выполнено в руководстве пользователя в приложении В.

# Выводы

В ходе выполнения курсового проекта была разработана программа, выполняющая учет канцтоваров. Программа позволяет создавать и удалять базу данных, добавлять, редактировать, удалять и фильтровать записи, а также формировать отчеты в формате PDF.

Результаты работы программы подтверждают корректность реализованной логики: интерфейс прост и удобен для пользователя, данные корректно сохраняются и обрабатываются, функции поиска и фильтрации работают стабильно, отчеты формируются без ошибок.

В процессе выполнения курсового проектирования был получен практический опыт создания приложений с архитектурой MVC в среде Microsoft Visual Studio. Были закреплены навыки программирования на языке C#, работы с WinForms, взаимодействия с базой данных SQLite, применения паттернов проектирования и подготовки документации.

# Список использованных источников

1. Васильев А. C#. Объектно-ориентированное программирование: Учебный курс. – СПб.: Питер, 2012. – 320 с.: ил.
2. Герберт Шилдт. С# 3.0. Полное руководство. – Изд. Вильямс, 2010.
3. Нейгел К., Ивьен Б. и др. С# 2008 и платформа NET 3.5 для профессионалов. – Изд. Диалектика, 2008.
4. Стиллмен Э., Грин Дж. Изучаем C# [пер. с англ. И. Рузмайкина]. – 2-е изд. - Москва: Питер, 2012. – 694 с. : ил.
5. Троелсен Э. Язык программирования С# 2008 и платформа .NET 3.5 – Изд. Вильямс, 2010.
6. Использование диаграммы вариантов использования UML при проектировании программного обеспечения – URL: https://habr.com/ru/articles/566218/
7. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Приемы объектно-ориентированного программирования. Паттерны проектирования. – СПб: Питер, 2009. – 366 с.: ил.
8. Фаулер М. Архитектура корпоративных приложений. — СПб.: Питер, 2004.
9. Использование диаграммы классов UML при проектировании и документировании программного обеспечения – URL: https://habr.com/ru/articles/572234/
10. Трей Нэш. C# 2010. Ускоренный курс для профессионалов. – Изд. Вильямс, 2010.
11. Руководство по SQLite – URL: <https://metanit.com/sql/sqlite/>
12. Внутренние таблицы SQLite – URL: https://habr.com/ru/articles/223451/
13. Арлоу Дж., Нейштадт А. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование. – СПб. : Символ-Плюс, 2007. – 624 с.
14. UML: обзор основных типов диаграмм, диаграмма компонентов – URL: <https://habr.com/ru/articles/756552/>
15. Обзорная статья о видах тестирования IT-продуктов – URL: <https://habr.com/ru/companies/otus/articles/720664/>

# Приложение A – Код программы

Код файла Program.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace KR\_23VP2\_Muraviev

{

static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

using (var helloForm = new HelloForm())

{

helloForm.ShowDialog();

}

Application.Run(new MainForm());

}

}

}

Код файла HelperForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace KR\_23VP2\_Muraviev

{

/// <summary>

/// Всплывающее окно с информацией.

/// </summary>

public partial class HelloForm: Form

{

/// <summary>

/// Таймер для автоматического закрытия формы.

/// </summary>

private Timer timer;

/// <summary>

/// Конструктор формы.

/// </summary>

public HelloForm()

{

InitializeComponent();

timer = new Timer();

timer.Interval = 3000;

timer.Tick += (sender, e) =>

{

timer.Stop();

this.Close();

};

}

/// <summary>

/// Загрузка формы.

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void HelloForm\_Load(object sender, EventArgs e)

{

timer.Start();

}

}

}

Код файла MainForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Data.Entity.Migrations.Model;

using System.Data.SQLite;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.VisualStyles;

using iTextSharp.text;

using iTextSharp.text.pdf;

namespace KR\_23VP2\_Muraviev

{

/// <summary>

/// Класс главной формы.

/// </summary>

public partial class MainForm : Form

{

/// <summary>

/// Предыдущее значение ячейки в таблице.

/// Используется для восстановления значения при некорректном редактировании.

/// </summary>

private int oldCellValue;

/// <summary>

/// Контроллер для управления логикой работы с товарами.

/// </summary>

private ProductController controller;

/// <summary>

/// Конструктор формы.

/// </summary>

public MainForm()

{

InitializeComponent();

controller = new ProductController();

}

/// <summary>

/// Создание пустой таблицы базы данных.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void createDBToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "SQLite Database (\*.db)|\*.db";

saveFileDialog.Title = "Создать новую базу данных";

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

{

controller.CreateNewDB(saveFileDialog.FileName);

opened\_db.Text = Path.GetFileName(saveFileDialog.FileName);

ShowAll();

MessageBox.Show("База данных успешно создана", "Успех",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при создании базы данных: " + ex.Message, "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Удаление таблицы базы данных.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void deleteDBToolStripMenuItem\_Click(object sender, EventArgs e)

{

try

{

var confirm = MessageBox.Show("Вы действительно хотите удалить базу данных?", "Удаление БД",

MessageBoxButtons.YesNo, MessageBoxIcon.Question);

if (confirm == DialogResult.Yes)

{

controller.DeleteDB();

table\_of\_data.Rows.Clear();

opened\_db.Text = "<Выберите файл>";

MessageBox.Show("База данных успешно удалена", "Удаление БД",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при удалении базы данных: " + ex.Message, "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Открытие выбранной таблицы базы данных.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void opened\_db\_Click(object sender, EventArgs e)

{

try

{

OpenFileDialog openFileDialog = new OpenFileDialog();

openFileDialog.Filter = "SQLite Database (\*.db)|\*.db";

openFileDialog.Title = "Открыть БД";

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

{

controller.OpenDB(openFileDialog.FileName);

opened\_db.Text = Path.GetFileName(openFileDialog.FileName);

ShowAll();

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при открытии базы данных: " + ex.Message, "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Нажатие на кнопку "Обновить таблицу".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void refresh\_button\_Click(object sender, EventArgs e)

{

find\_name.Text = "";

ShowAll();

}

/// <summary>

/// Нажатие на кнопку "Редактировать таблицу".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void edit\_button\_Click(object sender, EventArgs e)

{

if (!controller.DBExist())

{

MessageBox.Show("База данных не найдена", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

table\_of\_data.Rows.Clear();

return;

}

StartEditing();

}

/// <summary>

/// Нажатие на кнопку "Отменить изменения".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void close\_edit\_button\_Click(object sender, EventArgs e)

{

if (!controller.DBExist())

{

MessageBox.Show("База данных не найдена", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

CloseEditing();

table\_of\_data.Rows.Clear();

return;

}

CloseEditing();

ShowAll();

}

/// <summary>

/// Нажатие на кнопку "Сохранить изменения".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void save\_edit\_button\_Click(object sender, EventArgs e)

{

try

{

var (saved, updated, deleted) = controller.SaveChanges(table\_of\_data);

CloseEditing();

ShowAll();

MessageBox.Show($"Сохранено: {saved} добавлено, {updated} изменено, {deleted} удалено",

"Сохранить изменения", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при сохранении: " + ex.Message, "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Вывод всех элементов из БД.

/// </summary>

private void ShowAll()

{

if (!controller.DBExist())

{

MessageBox.Show("База данных не найдена", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

table\_of\_data.Rows.Clear();

return;

}

table\_of\_data.Rows.Clear();

foreach (var product in controller.LoadAll())

{

table\_of\_data.Rows.Add(

false,

product.Name,

product.Category,

$"{product.Amount} шт.",

$"{product.Price} руб."

);

}

}

/// <summary>

/// Открыть редактирование.

/// </summary>

private void StartEditing()

{

edit\_button.Visible = false;

close\_edit\_button.Visible = true;

save\_edit\_button.Visible = true;

edit\_group.Visible = true;

table\_of\_data.Columns[0].ReadOnly = false;

table\_of\_data.Columns[3].ReadOnly = false;

table\_of\_data.Columns[4].ReadOnly = false;

}

/// <summary>

/// Закрыть редактирование.

/// </summary>

private void CloseEditing()

{

edit\_button.Visible = true;

close\_edit\_button.Visible = false;

save\_edit\_button.Visible = false;

edit\_group.Visible = false;

table\_of\_data.Columns[0].ReadOnly = true;

table\_of\_data.Columns[3].ReadOnly = true;

table\_of\_data.Columns[4].ReadOnly = true;

}

/// <summary>

/// Сохранение изначального значения кол-ва/цены.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void table\_of\_data\_CellBeginEdit(object sender, DataGridViewCellCancelEventArgs e)

{

var cell = table\_of\_data.Rows[e.RowIndex].Cells[e.ColumnIndex];

if (cell.Value != null)

{

string value = cell.Value.ToString();

if (e.ColumnIndex == 3) // Количество

value = value.Replace(" шт.", "");

else if (e.ColumnIndex == 4) // Цена

value = value.Replace(" руб.", "");

if (int.TryParse(value, out int parsedValue))

{

oldCellValue = parsedValue;

cell.Value = value;

}

}

}

/// <summary>

/// Проверка на корректность введенного значения кол-ва/цены.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void table\_of\_data\_CellEndEdit(object sender, DataGridViewCellEventArgs e)

{

if (e.ColumnIndex == 0) return;

var cell = table\_of\_data.Rows[e.RowIndex].Cells[e.ColumnIndex];

string input = cell.Value?.ToString().Trim();

if (!int.TryParse(input, out int result) || result < 0 || string.IsNullOrEmpty(input))

{

MessageBox.Show("Введите корректное положительное число");

if (e.ColumnIndex == 3) // Количество

cell.Value = $"{oldCellValue} шт.";

else if (e.ColumnIndex == 4) // Цена

cell.Value = $"{oldCellValue} руб.";

return;

}

if (e.ColumnIndex == 3) // Количество

cell.Value = $"{result} шт.";

else if (e.ColumnIndex == 4) // Цена

cell.Value = $"{result} руб.";

if (result != oldCellValue)

{

if (table\_of\_data.Rows[e.RowIndex].Tag == null)

{

table\_of\_data.Rows[e.RowIndex].Tag = "edit";

table\_of\_data.Rows[e.RowIndex].DefaultCellStyle.BackColor = Color.PaleGoldenrod;

}

}

}

/// <summary>

/// Нажатие на кнопку "Добавить новый".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void add\_button\_Click(object sender, EventArgs e)

{

if (!controller.DBExist())

{

MessageBox.Show("База данных не найдена", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

CloseEditing();

table\_of\_data.Rows.Clear();

return;

}

using (FormAddItem form = new FormAddItem())

{

if (form.ShowDialog() == DialogResult.OK)

{

DataGridViewRow newRow = new DataGridViewRow();

newRow.Height = 30;

newRow.Cells.Add(new DataGridViewCheckBoxCell() { Value = false });

newRow.Cells.Add(new DataGridViewTextBoxCell() { Value = form.name });

newRow.Cells.Add(new DataGridViewTextBoxCell() { Value = form.category });

newRow.Cells.Add(new DataGridViewTextBoxCell() { Value = form.amount + " шт." });

newRow.Cells.Add(new DataGridViewTextBoxCell() { Value = form.price + " руб." });

newRow.DefaultCellStyle.BackColor = Color.LightGreen;

newRow.Tag = "new";

table\_of\_data.Rows.Add(newRow);

}

}

}

/// <summary>

/// Нажатие на кнопку "Удалить выбранные".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void delete\_button\_Click(object sender, EventArgs e)

{

if (!controller.DBExist())

{

MessageBox.Show("База данных не найдена", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

CloseEditing();

table\_of\_data.Rows.Clear();

return;

}

bool anySelected = false;

for (int i = 0; i < table\_of\_data.Rows.Count; i++)

{

var cell = table\_of\_data.Rows[i].Cells[0];

if (cell.Value is bool selected && selected)

{

anySelected = true;

table\_of\_data.Rows[i].DefaultCellStyle.BackColor = Color.IndianRed;

cell.ReadOnly = true;

table\_of\_data.Rows[i].Tag = "delete";

}

}

if (!anySelected)

{

MessageBox.Show("Вы не выбрали ни один элемент", "Удалить выбранные",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

/// <summary>

/// Поиск по названию.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void find\_name\_TextChanged(object sender, EventArgs e)

{

string searchText = find\_name.Text.Trim().ToLower();

foreach (DataGridViewRow row in table\_of\_data.Rows)

{

string productName = row.Cells[1].Value?.ToString().ToLower();

bool isMatch = !string.IsNullOrEmpty(productName) && productName.Contains(searchText);

row.Visible = string.IsNullOrEmpty(searchText) || isMatch;

}

}

/// <summary>

/// Нажатие на кнопку "Сбросить фильтр".

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void reset\_filter\_button\_Click(object sender, EventArgs e)

{

for (int i = 0; i < filter\_checkbox.Items.Count; i++)

{

filter\_checkbox.SetItemChecked(i, false);

}

foreach (DataGridViewRow row in table\_of\_data.Rows)

{

row.Visible = true;

}

}

/// <summary>

/// Смена элемента CheckedListBox.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void filter\_checkbox\_ItemCheck(object sender, ItemCheckEventArgs e)

{

this.BeginInvoke((MethodInvoker)FilterByCategories);

}

/// <summary>

/// Фильтровать по выбранным категориям.

/// </summary>

private void FilterByCategories()

{

var selectedCategories = filter\_checkbox.CheckedItems.Cast<string>().ToList();

foreach (DataGridViewRow row in table\_of\_data.Rows)

{

if (selectedCategories.Count == 0)

{

row.Visible = true;

continue;

}

string category = row.Cells[2].Value?.ToString();

row.Visible = selectedCategories.Contains(category);

}

}

/// <summary>

/// Формирование отчета в виде PDF-файла с таблицей базы данных.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void generateReportToolStripMenuItem\_Click(object sender, EventArgs e)

{

if (table\_of\_data.Rows.Count == 0)

{

MessageBox.Show("Нет данных для формирования отчета", "Сформировать отчет",

MessageBoxButtons.OK, MessageBoxIcon.Information);

return;

}

using (SaveFileDialog saveFileDialog = new SaveFileDialog())

{

saveFileDialog.Filter = "PDF файл (\*.pdf)|\*.pdf";

saveFileDialog.Title = "Сохранить отчет как...";

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

{

try

{

controller.GenerateReport(table\_of\_data, saveFileDialog.FileName);

MessageBox.Show("Отчет успешно сформирован и сохранен", "Сформировать отчет",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при создании отчета: " + ex.Message, "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

/// <summary>

/// Обработчик изменения размера главной формы.

/// </summary>

/// <param name="sender">Объект, вызвавший событие.</param>

/// <param name="e">Аргументы события.</param>

private void MainForm\_Resize(object sender, EventArgs e)

{

table\_of\_data.Height = (int)(this.ClientSize.Height \* 0.45);

table\_of\_data.Columns[0].Width = (int)(table\_of\_data.Width \* 0.05);

table\_of\_data.Columns[1].Width = (int)(table\_of\_data.Width \* 0.26);

table\_of\_data.Columns[2].Width = (int)(table\_of\_data.Width \* 0.31);

table\_of\_data.Columns[3].Width = (int)(table\_of\_data.Width \* 0.19);

table\_of\_data.Columns[4].Width = (int)(table\_of\_data.Width \* 0.19);

refresh\_button.Left = (int)(this.ClientSize.Width \* 0.4);

refresh\_button.Top = table\_of\_data.Height + 34;

label1.Left = refresh\_button.Left - 262;

label1.Top = refresh\_button.Top + 12;

find\_name.Left = refresh\_button.Left - 128;

find\_name.Top = refresh\_button.Top + 9;

edit\_button.Left = refresh\_button.Left - 136;

edit\_button.Top = refresh\_button.Top + 49;

save\_edit\_button.Left = edit\_button.Left - 78;

save\_edit\_button.Top = edit\_button.Top;

close\_edit\_button.Left = edit\_button.Left + 83;

close\_edit\_button.Top = edit\_button.Top;

edit\_group.Left = refresh\_button.Left - 262;

edit\_group.Top = refresh\_button.Top + 104;

filter\_group.Left = refresh\_button.Left + 137;

filter\_group.Top = table\_of\_data.Height + 34;

}

}

}

Код файла FormAddItem.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Xml.Linq;

namespace KR\_23VP2\_Muraviev

{

/// <summary>

/// Класс формы для добавления товаров.

/// </summary>

public partial class FormAddItem : Form

{

/// <summary>

/// Название товара.

/// </summary>

public string name { get; set; }

/// <summary>

/// Категория товара.

/// </summary>

public string category { get; set; }

/// <summary>

/// Количество товара на складе.

/// </summary>

public int amount { get; set; }

/// <summary>

/// Цена за единицу товара.

/// </summary>

public int price { get; set; }

/// <summary>

/// Конструктор формы.

/// </summary>

public FormAddItem()

{

InitializeComponent();

this.ShowInTaskbar = false;

}

/// <summary>

/// Нажатие на кнопку "Добавить".

/// </summary>

/// <param name="sender"></param>

/// <param name="e"></param>

private void add\_button\_Click(object sender, EventArgs e)

{

if (!string.IsNullOrWhiteSpace(name\_input.Text) && category\_input.SelectedIndex > -1)

{

name = name\_input.Text;

category = category\_input.Text;

amount = (int)amount\_input.Value;

price = (int)price\_input.Value;

DialogResult = DialogResult.OK;

Close();

}

else

{

MessageBox.Show("Введите корректные данные!", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

Код файла Product.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace KR\_23VP2\_Muraviev

{

/// <summary>

/// Представляет товар на складе.

/// </summary>

public class Product

{

/// <summary>

/// Название товара.

/// </summary>

public string Name { get; set; }

/// <summary>

/// Категория товара.

/// </summary>

public string Category { get; set; }

/// <summary>

/// Количество товара на складе.

/// </summary>

public int Amount { get; set; }

/// <summary>

/// Цена за единицу товара.

/// </summary>

public int Price { get; set; }

/// <summary>

/// Тег состояния строки (new, edit, delete).

/// </summary>

public string Tag { get; set; }

}

}

Код файла ProductController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace KR\_23VP2\_Muraviev

{

/// <summary>

/// Контроллер для работы с товарами и базой данных.

/// </summary>

public class ProductController

{

/// <summary>

/// Объект-помощник для работы с базой данных.

/// </summary>

private DBHelper dbHelper;

/// <summary>

/// Конструктор класса.

/// </summary>

public ProductController()

{

dbHelper = new DBHelper();

}

/// <summary>

/// Проверяет, существует ли текущая база данных.

/// </summary>

/// <returns>true, если база данных существует, иначе false.</returns>

public bool DBExist()

{

return dbHelper.IsDBExist();

}

/// <summary>

/// Создаёт новую базу данных.

/// </summary>

/// <param name="path">Полный путь для создания файла базы данных.</param>

public void CreateNewDB(string path)

{

dbHelper.CreateDB(path);

}

/// <summary>

/// Удаляет текущую базу данных.

/// </summary>

public void DeleteDB()

{

dbHelper.DeleteCurDB();

}

/// <summary>

/// Открывает выбранную базу данных.

/// </summary>

/// <param name="path">Полный путь к файлу базы данных для открытия.</param>

public void OpenDB(string path)

{

dbHelper.OpenSelectedDB(path);

}

/// <summary>

/// Сохраняет изменения списка товаров в базе данных.

/// </summary>

/// <param name="dataGridView">Таблица с исходными данными.</param>

/// <returns>Кортеж с количеством сохранённых, обновлённых и удалённых записей.</returns>

public (int saved, int updated, int deleted) SaveChanges(DataGridView dataGridView)

{

var productList = new List<Product>();

foreach (DataGridViewRow row in dataGridView.Rows)

{

string name = row.Cells[1].Value?.ToString();

string category = row.Cells[2].Value?.ToString();

string amountStr = row.Cells[3].Value?.ToString().Replace(" шт.", "");

string priceStr = row.Cells[4].Value?.ToString().Replace(" руб.", "");

if (!int.TryParse(amountStr, out int amount) || !int.TryParse(priceStr, out int price))

continue;

string tag = row.Tag?.ToString();

productList.Add(new Product{

Name = name,

Category = category,

Amount = amount,

Price = price,

Tag = tag

});

}

return dbHelper.SaveAll(productList);

}

/// <summary>

/// Загружает все товары из базы данных.

/// </summary>

/// <returns>Список всех товаров.</returns>

public List<Product> LoadAll()

{

return dbHelper.GetAll();

}

/// <summary>

/// Извлекает список товаров из таблицы DataGridView.

/// </summary>

/// <returns>Список товаров, соответствующих строкам таблицы.</returns>

public List<Product> GetProductsFromTable(DataGridView dataGridView)

{

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

foreach (DataGridViewRow row in dataGridView.Rows)

{

products.Add(new Product

{

Name = row.Cells[1].Value?.ToString(),

Category = row.Cells[2].Value?.ToString(),

Amount = Convert.ToInt32(row.Cells[3].Value?.ToString().Replace(" шт.", "")),

Price = Convert.ToInt32(row.Cells[4].Value?.ToString().Replace(" руб.", ""))

});

}

return products;

}

/// <summary>

/// Генерирует PDF-отчёт по списку товаров и сохраняет его в указанный файл.

/// </summary>

/// <param name="dataGridView">Таблица с исходными данными.</param>

/// <param name="filePath">Путь для сохранения PDF-файла.</param>

public void GenerateReport(DataGridView dataGridView, string filePath)

{

List<Product> products = GetProductsFromTable(dataGridView);

dbHelper.GeneratePdfReport(products, filePath);

}

}

}

Код файла DBHelper.cs

using iTextSharp.text.pdf;

using iTextSharp.text;

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using iTextSharp.text.pdf.parser;

namespace KR\_23VP2\_Muraviev

{

/// <summary>

/// Класс-помощник для работы с базой данных SQLite и генерацией PDF-отчётов.

/// </summary>

class DBHelper

{

/// <summary>

/// Путь к файлу базы данных SQLite.

/// Используется для подключения к текущей базе данных проекта.

/// </summary>

private string dbPath;

/// <summary>

/// Возвращает текущий путь к базе данных.

/// </summary>

/// <returns>Путь к файлу базы данных.</returns>

public string GetPath()

{

return dbPath;

}

/// <summary>

/// Проверяет, существует ли база данных по текущему пути.

/// </summary>

/// <returns>true, если файл существует, иначе false.</returns>

public bool IsDBExist()

{

if (string.IsNullOrEmpty(dbPath) || !File.Exists(dbPath))

return false;

return true;

}

/// <summary>

/// Создаёт новую базу данных по указанному пути.

/// Если файл уже существует — он будет удалён.

/// Создаётся таблица products.

/// </summary>

/// <param name="path">Полный путь для создания базы данных.</param>

public void CreateDB(string path)

{

dbPath = path;

if (File.Exists(dbPath))

File.Delete(dbPath);

SQLiteConnection.CreateFile(dbPath);

using (SQLiteConnection conn = new SQLiteConnection($"Data Source={dbPath}"))

{

conn.Open();

string createTableQuery = @"

CREATE TABLE IF NOT EXISTS products (

id INTEGER PRIMARY KEY AUTOINCREMENT,

name TEXT NOT NULL,

category TEXT NOT NULL,

amount INTEGER NOT NULL,

price INTEGER NOT NULL

);";

using (SQLiteCommand cmd = new SQLiteCommand(createTableQuery, conn))

{

cmd.ExecuteNonQuery();

}

conn.Close();

}

}

/// <summary>

/// Удаляет текущую базу данных и сбрасывает путь.

/// </summary>

public void DeleteCurDB()

{

File.Delete(dbPath);

dbPath = null;

}

/// <summary>

/// Открывает существующую базу данных по указанному пути.

/// </summary>

/// <param name="path">Путь к существующему файлу базы данных.</param>

/// <exception cref="Exception">Если файл не найден.</exception>

public void OpenSelectedDB(string path)

{

if (!File.Exists(path))

throw new Exception("Файл базы данных не найден.");

dbPath = path;

}

/// <summary>

/// Загружает все записи товаров из базы данных.

/// </summary>

/// <returns>Список товаров.</returns>

public List<Product> GetAll()

{

var products = new List<Product>();

using (var conn = new SQLiteConnection($"Data Source={dbPath}"))

{

conn.Open();

string query = "SELECT Name, Category, Amount, Price FROM products";

using (var cmd = new SQLiteCommand(query, conn))

using (var reader = cmd.ExecuteReader())

{

while (reader.Read())

{

products.Add(new Product

{

Name = reader.GetString(0),

Category = reader.GetString(1),

Amount = reader.GetInt32(2),

Price = reader.GetInt32(3)

});

}

}

}

return products;

}

/// <summary>

/// Сохраняет список товаров в базу данных.

/// Добавляет, обновляет или удаляет записи в зависимости от значения поля Tag у товара.

/// </summary>

/// <param name="products">Список товаров для обработки.</param>

/// <returns>Кортеж с количеством добавленных, обновлённых и удалённых записей.</returns>

/// <exception cref="Exception">Если база данных не существует.</exception>

public (int saved, int updated, int deleted) SaveAll(List<Product> products)

{

if (string.IsNullOrEmpty(dbPath) || !File.Exists(dbPath))

{

throw new Exception("База данных не найдена");

}

int saved = 0, updated = 0, deleted = 0;

using (var conn = new SQLiteConnection($"Data Source={dbPath}"))

{

conn.Open();

foreach (var product in products)

{

switch (product.Tag)

{

case "new":

string insertQuery = @"

INSERT INTO products (Name, Category, Amount, Price)

VALUES (@Name, @Category, @Amount, @Price)";

using (var cmd = new SQLiteCommand(insertQuery, conn))

{

cmd.Parameters.AddWithValue("@Name", product.Name);

cmd.Parameters.AddWithValue("@Category", product.Category);

cmd.Parameters.AddWithValue("@Amount", product.Amount);

cmd.Parameters.AddWithValue("@Price", product.Price);

cmd.ExecuteNonQuery();

saved++;

}

break;

case "delete":

string deleteQuery = "DELETE FROM products WHERE Name = @Name";

using (var cmd = new SQLiteCommand(deleteQuery, conn))

{

cmd.Parameters.AddWithValue("@Name", product.Name);

cmd.ExecuteNonQuery();

deleted++;

}

break;

case "edit":

string updateQuery = @"

UPDATE products

SET Category = @Category, Amount = @Amount, Price = @Price

WHERE Name = @Name";

using (var cmd = new SQLiteCommand(updateQuery, conn))

{

cmd.Parameters.AddWithValue("@Name", product.Name);

cmd.Parameters.AddWithValue("@Category", product.Category);

cmd.Parameters.AddWithValue("@Amount", product.Amount);

cmd.Parameters.AddWithValue("@Price", product.Price);

cmd.ExecuteNonQuery();

updated++;

}

break;

}

}

}

return (saved, updated, deleted);

}

/// <summary>

/// Генерирует PDF-отчёт по товарам и сохраняет его в файл.

/// </summary>

/// <param name="products">Список товаров для отчёта.</param>

/// <param name="filePath">Путь для сохранения PDF-файла.</param>

public void GeneratePdfReport(List<Product> products, string filePath)

{

string fontPath = System.IO.Path.Combine(Application.StartupPath, "Fonts", "arial.ttf");

BaseFont baseFont = BaseFont.CreateFont(fontPath, BaseFont.IDENTITY\_H, BaseFont.EMBEDDED);

iTextSharp.text.Font titleFont = new iTextSharp.text.Font(baseFont, 16, iTextSharp.text.Font.BOLD);

iTextSharp.text.Font cellFont = new iTextSharp.text.Font(baseFont, 12);

Document doc = new Document(PageSize.A4, 20, 20, 20, 20);

PdfWriter.GetInstance(doc, new FileStream(filePath, FileMode.Create));

doc.Open();

Paragraph title = new Paragraph("Учет канцтоваров", titleFont);

title.Alignment = Element.ALIGN\_CENTER;

title.SpacingAfter = 20;

doc.Add(title);

PdfPTable pdfTable = new PdfPTable(4);

pdfTable.WidthPercentage = 100;

string[] headers = { "Название товара", "Категория", "Количество на складе", "Цена за единицу" };

foreach (string header in headers)

{

PdfPCell cell = new PdfPCell(new Phrase(header, cellFont))

{

BackgroundColor = new BaseColor(100, 255, 100),

HorizontalAlignment = Element.ALIGN\_CENTER

};

pdfTable.AddCell(cell);

}

foreach (Product p in products)

{

pdfTable.AddCell(new Phrase(p.Name, cellFont));

pdfTable.AddCell(new Phrase(p.Category, cellFont));

pdfTable.AddCell(new Phrase(p.Amount.ToString(), cellFont));

pdfTable.AddCell(new Phrase(p.Price.ToString(), cellFont));

}

doc.Add(pdfTable);

doc.Close();

}

}

}

# Приложение B – Руководство пользователя

Программа KR\_23VP2\_Muraviev.exe предназначена для хранения информации о канцтоварах. Программа имеет интуитивно понятный интерфейс и поддерживает такие операции как добавления нового товара, удаления товара, редактирования информации о товаре, поиск товара по названию, сохранение существующей БД, а также несколько второстепенных операций, как сортировка по алфавиту или фильтрация по категориям.

Для открытия и вывода на экран существующей БД необходимо нажать на пункт меню «<Выберите файл>» в левом верхнем углу (рисунок 15).

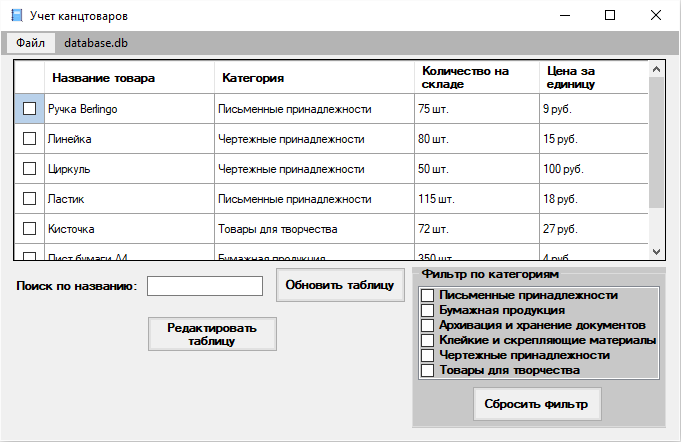


Рисунок 15 – Окно вывода полной информации о канцтоварах

Для создания пустой таблицы БД необходимо в меню «Файл» в левом верхнем углу выбрать пункт «Создать новую БД» (рисунок 16).

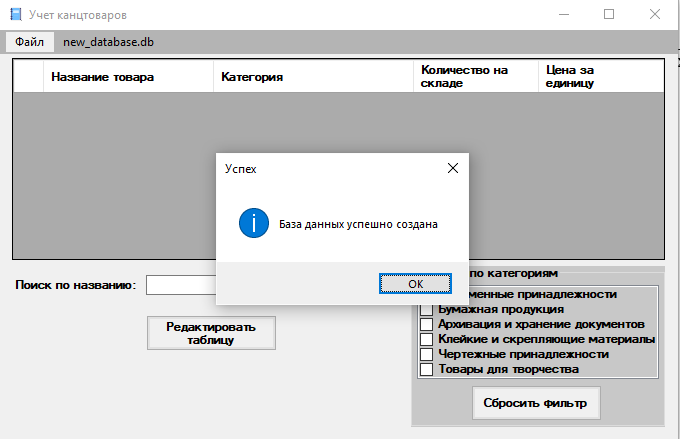


Рисунок 16 – Создание новой БД

Для поиска товара в каталоге необходимо ввести название в строку «Поиск по названию» (рисунок 17).

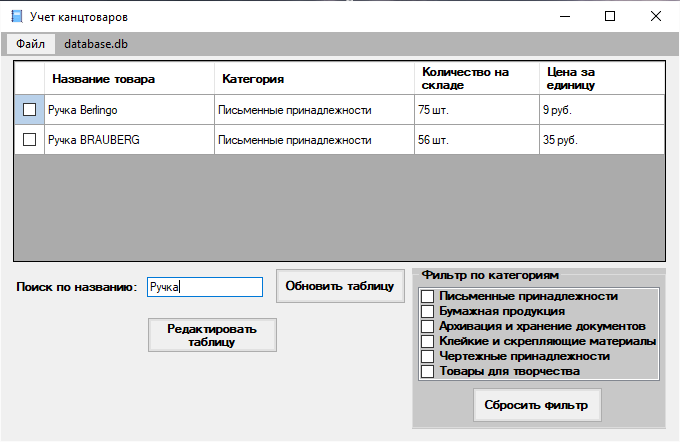


Рисунок 17 – Поиск по названию

Для редактирования данных нужно нажать на кнопку «Редактировать таблицу» (при этом БД должна быть создана/открыта), после чего в таблице на форме изменить нужный параметр (рисунок 18).

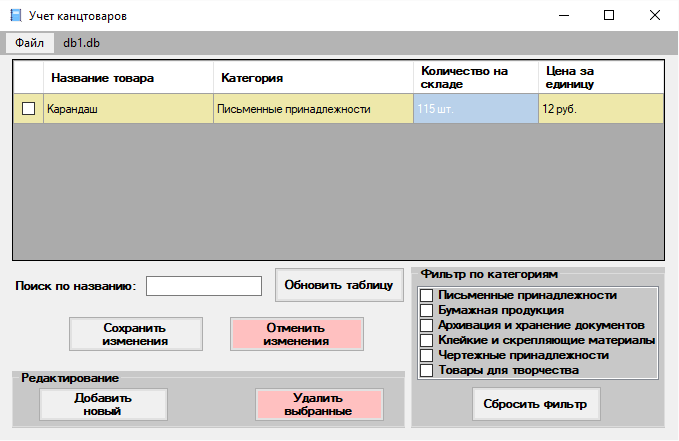


Рисунок 18 – Редактирование данных о товаре

Для добавления нового товара необходимо нажать кнопку «Редактировать таблицу», после чего нажать «Добавить новый». После чего в открывшейся форме заполнить данные о товаре и нажать «Добавить» (рисунок 19). Затем нужно нажать кнопку «Сохранить изменения» для добавления товара в БД.

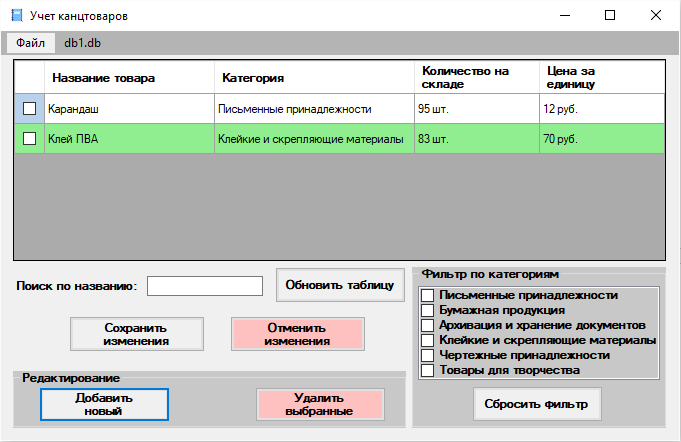


Рисунок 19 – Добавление нового товара

Для удаления записи необходимо нажать кнопку «Редактировать таблицу», выбрать в таблице товары, которые хотите удалить, после чего нажать кнопку «Удалить выбранные» (рисунок 20). Для удаления товаров из БД необходимо нажать кнопку «Сохранить изменения»

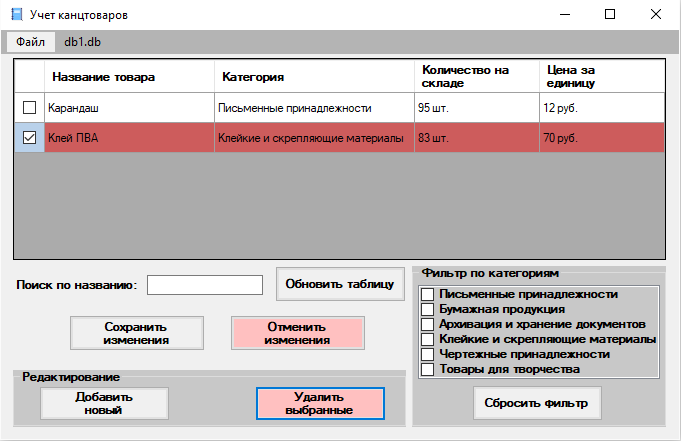


Рисунок 20 – Удаление товара

Для фильтрации по категориям нужно выбрать нужные категории в списке «Фильтр по категориям», после чего в таблице останутся только те элементы, которые соответствуют выбранным критериям (рисунок 21).

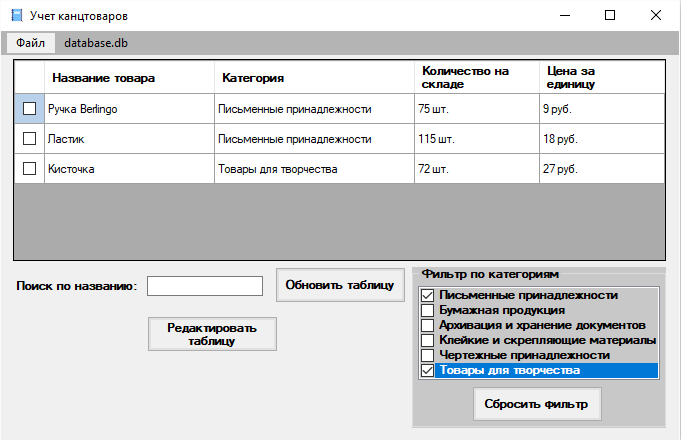


Рисунок 21 – Фильтрация таблицы по категориям

Для генерации PDF-отчета необходимо в меню «Файл» выбрать пункт «Сформировать отчет», после чего выбрать, куда сохранять файл. В случае успеха на экране будет уведомление «Отчет успешно сформирован и сохранен» (рисунок 22).

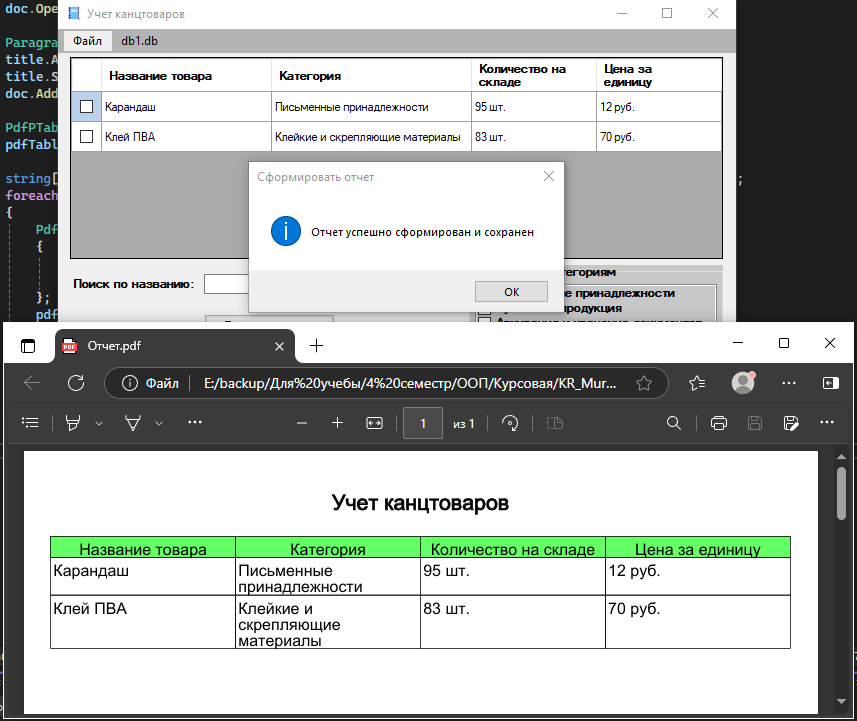


Рисунок 22 – Формирование PDF-отчета