СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Программы для создания и управления списками покупок стали неотъемлемой частью жизни современных пользователей. Они помогают организовать процесс покупок, контролировать расходы и экономить время. В данном разделе будут рассмотрены такие функции программного продукта: категоризация товаров, история покупок, автоматическое заполнение списков, анализ расходов и расчёт стоимости. Также будут предложены дополнительные функции.

Цель работы: Разработка программного продукта — программы для создания и управления списками покупок на языке C++.

Задачи работы:

1. Описание концепции программы для управления списками покупок;
2. Изучение существующих решений (например, *Bring!*, *Out of Milk*, *AnyList*);
3. Составление требований к программе (категоризация товаров, история покупок, автоматическое заполнение, анализ расходов и др.);
4. Реализация программы в соответствии с требованиями с применением принципов ООП;
5. Проведение тестирования программы на корректность работы и удобство интерфейса;
6. Составление отчёта по работе, включая описание архитектуры и функционала;
7. Презентация отчёта и его защита.

Объектом данного исследования является Программный продукт для создания и управления списками покупок.

Предметом исследования в данной работе является Разработка программы для управления покупками на языке C++ для ПК с использованием объектно-ориентированного подхода.

В качестве основных методов исследования применены анализ существующих приложений (сравнение функций, выявление недостатков), синтез идей для улучшения (автоматическая категоризация, визуализация истории покупок), моделирование архитектуры программы (классы *Product*, *Category*, *ShoppingList* и др.), тестирование и оптимизация кода. **Практическая реализация**: программа реализована на C++ с применением принципов ООП (инкапсуляция, наследование, полиморфизм). Основные функции: создание, редактирование и удаление товаров и категорий; сохранение истории покупок с фильтрацией по дате и категории; автоматическое заполнение списков на основе предыдущих покупок; анализ расходов и сравнение периодов; расчёт общей стоимости списка.

Информационной базой исследования являются открытые источники, а также материалы курса «Технологии индустриального программирования», доступные через систему дистанционного обучения РТУ МИРЭА.

В данном отчёте будет представлен процесс разработки программного продукта, технологическое проектирование и описание системы. а также непосредственно результаты разработки.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

1.1 Описание программируемой системы

Программы для создания и управления списками покупок представляют собой специализированные программные инструменты, предназначенные для систематизации и контроля приобретаемых товаров. Основная цель таких программ заключается в упрощении процесса планирования покупок, минимизации временных затрат и снижении риска забыть необходимые товары. В условиях современного ритма жизни, когда время становится критически важным ресурсом, использование структурированных списков помогает оптимизировать бытовые задачи.

Эффективный список покупок должен обладать несколькими ключевыми характеристиками. Во-первых, он должен быть удобен для внесения и редактирования позиций, чтобы пользователь мог быстро добавлять новые товары или удалять уже приобретённые. Во-вторых, желательна возможность группировки товаров по категориям, таким как «Молочные продукты», «Овощи и фрукты» или «Бакалея», что ускоряет процесс поиска в магазине. Кроме того, полезной функцией является сортировка товаров, например, по приоритету или алфавиту, что позволяет упорядочить список в соответствии с предпочтениями пользователя.

Современные программы для управления списками покупок основываются на различных технологических решениях. Некоторые из них представляют собой простые текстовые редакторы с возможностью сохранения списка в файл, другие — сложные мобильные или веб-приложения с синхронизацией между устройствами. Для хранения данных могут использоваться локальные базы данных, облачные хранилища или комбинированные подходы. В ряде случаев применяются алгоритмы автоматического предложения товаров на основе предыдущих покупок, что ускоряет формирование списка.

Среди методов создания списков выделяют ручное добавление товаров, при котором пользователь самостоятельно вносит все необходимые позиции. Альтернативой является использование шаблонов или предустановленных категорий, позволяющих быстрее заполнять список. Некоторые программы поддерживают совместное редактирование, что особенно полезно для семей или групп, планирующих покупки вместе. Также существуют решения с интеграцией в голосовые помощники, позволяющие добавлять товары с помощью голосовых команд.

Программы для управления списками покупок могут быть интегрированы в более крупные системы, такие как приложения для ведения домашнего бюджета или планировщики задач. Они также встречаются в составе мобильных операционных систем или в виде отдельных сервисов, доступных через веб-браузер. Некоторые приложения предлагают дополнительные функции, например, сравнение цен в разных магазинах, создание меню на неделю или формирование чеков.

Однако существуют определённые риски, связанные с использованием подобных программ. Ненадёжные приложения могут терять данные из-за ошибок синхронизации или отсутствия резервного копирования. Излишне сложный интерфейс может затруднять использование программы, особенно для людей с ограниченными техническими навыками. Кроме того, зависимость от электронных устройств создаёт риск недоступности списка в случае разряда батареи или отсутствия интернет-соединения.

Современные тенденции в развитии таких программ включают внедрение искусственного интеллекта для анализа покупательских привычек и автоматического формирования рекомендаций. Также растёт популярность интеграции с умными домами, где список покупок может автоматически обновляться на основе данных о расходе продуктов. Ещё одним направлением является использование технологий дополненной реальности, которые могут помогать пользователю находить товары в магазине на основе заранее составленного списка.

1.2 Обзор существующих решений

1. Категоризация товаров

Категоризация товаров — одна из ключевых функций программ для управления списками покупок. Она позволяет пользователям группировать товары по категориям (например, «Молочные продукты», «Овощи и фрукты», «Бакалея» и т.д.), что упрощает процесс добавления и поиска товаров.

Существующие решения:

1) многие приложения, такие как *Bring!* и *Out of Milk*, предлагают предопределённые категории, которые можно редактировать;

2) в некоторых программах категории создаются автоматически на основе частоты покупок.

Предложение для улучшения: добавить возможность автоматической категоризации на основе анализа текста (например, если пользователь вводит «молоко», программа автоматически добавляет товар в категорию «Молочные продукты»).

2. История покупок

История покупок позволяет пользователям отслеживать, какие товары они покупали ранее. Это полезно для анализа привычек и планирования будущих покупок.

Существующие решения:

1) приложения, такие как *AnyList* и *OurGroceries*, сохраняют историю покупок и позволяют быстро добавлять товары из прошлых списков;

2) некоторые программы предлагают фильтрацию по дате или магазину.

Предложение для улучшения: добавить функцию анализа истории покупок с визуализацией (например, диаграммы частоты покупок определённых товаров).

3. Автоматическое заполнение

Автоматическое заполнение списка покупок на основе предыдущих покупок — это удобная функция, которая экономит время пользователя.

Существующие решения:

1) приложения, такие как *Google Keep* и *Todoist*, предлагают подсказки на основе часто добавляемых товаров;

2) в некоторых программах используются алгоритмы машинного обучения для предсказания покупок.

Предложение для улучшения: реализовать предложение товаров из категории, наиболее потребляемой пользователем (например, человек часто покупает «молоко», при новой покупке ему будет предложено купить «кефир»).

4. Анализ расходов

Анализ расходов помогает пользователям контролировать свой бюджет. Программа может предоставлять статистику по тратам за определённый период.

Существующие решения:

1) приложения, такие как *YNAB (You Need A Budget)*, интегрируют списки покупок с финансовым планированием;

2) некоторые программы предоставляют сводки по категориям расходов.

Предложение для улучшения: добавить функцию сравнения расходов за разные периоды (например, «За эту неделю Вы потратили на 10% больше, чем за прошлую»).

5. Расчёт стоимости

Расчёт стоимости списка покупок позволяет пользователям заранее оценить свои траты.

Существующие решения:

1) в приложениях, таких как *Listonic*, пользователи могут вручную вводить цены товаров;

2) некоторые программы интегрируются с онлайн-магазинами для автоматического расчёта стоимости.

Предложение для улучшения: добавить возможность автоматического подсчёта общей стоимости покупки при вводе цены каждого товаров

пользователем. Также программа будет запоминать стоимости товаров, но при изменении цены в магазине, пользователь сможет вручную изменить цену в приложении.

6. Дополнительные функции

Напоминания: уведомления о необходимости покупки определённых товаров.

В Таблице 1.1 представлено сравнение функций существующих приложений.

Таблица 1.1 – Сравнение функций существующих приложений

Функция	**Bring!**	**Out of Milk**	**AnyList**	**Google Keep**	**Предложение для улучшения**
Категоризация товаров	Да	Да	Да	Нет	Автоматическая категоризация
История покупок	Да	Да	Да	Нет	Визуализация истории
Автоматическое заполнение	Нет	Да	Да	Да	Сезонные предложения
Анализ расходов	Нет	Нет	Нет	Нет	Сравнение периодов
Расчёт стоимости	Нет	Да	Нет	Нет	Сканирование чеков

Также ниже представлен Рисунок 1.1, на котором приведена столбиковая диаграмма популярности функций в рассмотренных решениях

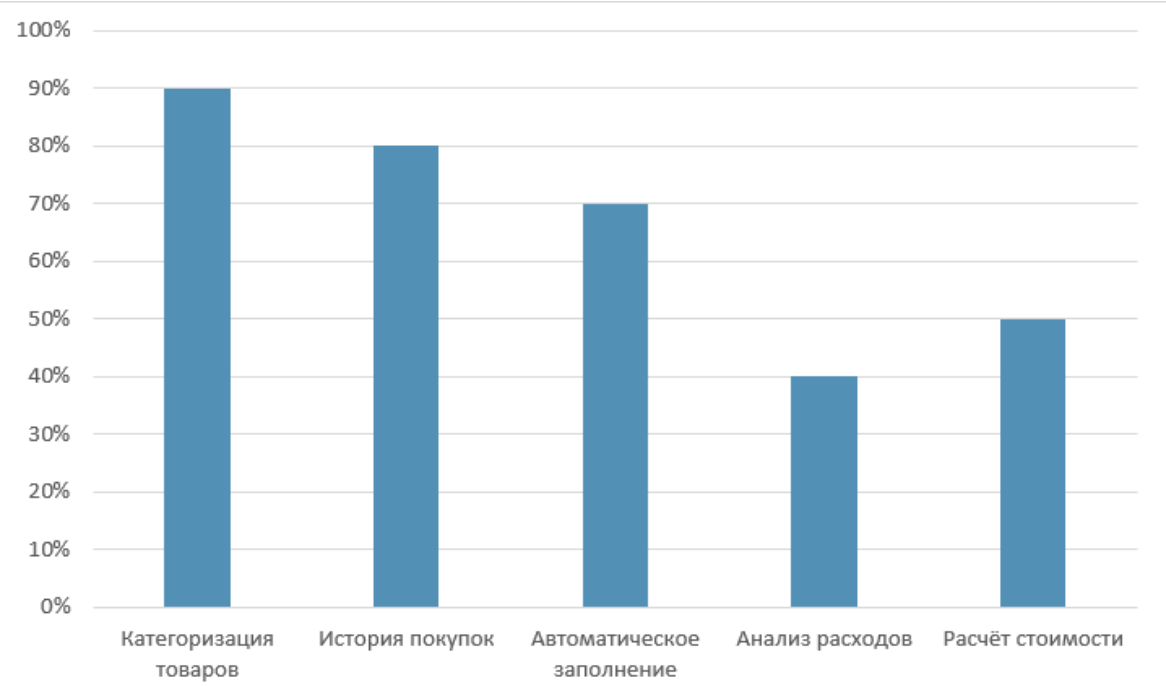

Рисунок 1.1 – Диаграмма популярности функций в существующих приложениях

1.3 Анализ алгоритма создания программного продукта

1. Проектирование

1) Модульная структура: отдельные модули для работы с данными, интерфейсом и логикой.

2) Использование ООП (объектно-ориентированного программирования) для создания классов (например, Product, Category, ShoppingList).

2. Разработка

Основные функции**:**

1) Добавление, удаление и редактирование товаров.

2) Категоризация товаров.

3) Сохранение истории покупок.

4) Автоматическое заполнение списка.

5) Расчёт стоимости и анализ расходов.

3. Издание

4. Поддержка

1.4 Требования к реализации программного продукта

В Таблице 1.2 представлены требования к программируемой системе.

Таблица 1.2 – Требования к продукту

№	Требование	Значение
1	Язык программирования	C++
2	Корректность работы	Программа стабильно работает на всех этапах: от запуска до завершения
3	Применение принципов ООП	Использование классов, инкапсуляции, наследования и полиморфизма
4	Интерфейс пользователя	Интуитивно понятный интерфейс с поддержкой всех необходимых функций
5	Инструкция по эксплуатации	Написана инструкция по эксплуатации, содержащая, в том числе, основные рекомендации по использованию и пояснения к возможным ошибкам в программе
6	Управление вводом	Клавиатура ПК
7	Категоризация товаров	Возможность создания, редактирования и удаления категорий
8	Расчёт стоимости	Автоматический расчёт общей стоимости списка покупок
9	История покупок	Сохранение и просмотр истории покупок с фильтрацией по дате и категории
10	Анализ расходов	Статистика по расходам с возможностью сравнения за разные периоды
11	Система управления версиями	Использование системы управления версиями Git для отслеживания изменений в коде и ресурсах, обеспечивающее безопасность

2 ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

2.1 Описание состояний системы

1. Состояния системы

Текстовое описание:

Состояния системы:

1. Главное меню
    - Начальное состояние, где пользователь выбирает действие.
2. Добавление товара
    - Пользователь добавляет новый товар в список.
3. Редактирование товара
    - Пользователь изменяет информацию о товаре.
4. Просмотр истории покупок
    - Пользователь просматривает список ранее купленных товаров.
5. Анализ расходов
    - Пользователь анализирует свои расходы за определённый период.
6. Завершение работы
    - Пользователь завершает работу с программой.

Переходы между состояниями:

1. Из Главное меню в Добавление товара при выборе *«Добавить товар»*.
2. Из Добавление товара обратно в Главное меню после завершения добавления.
3. Из Главное меню в Редактирование товара при выборе *«Редактировать товар»*.
4. Из Редактирование товара обратно в Главное меню после завершения редактирования.
5. Из Главное меню в Просмотр истории покупок при выборе *«История покупок»*.

6. Из Просмотр истории покупок обратно в Главное меню после завершения просмотра.
7. Из Главное меню в Анализ расходов при выборе *«Анализ расходов»*.
8. Из Анализ расходов обратно в Главное меню после завершения анализа.
9. Из Главное меню в Завершение работы при выборе *«Выход»*.

2.2 Описание классов системы

Текстовое описание:

Класс Product:

Атрибуты:

1) *name* (название товара).
2) *category* (категория товара).
3) *price* (цена товара).

Методы:

4) *addProduct*() (добавить товар).
5) *editProduct*() (редактировать товар).
6) *deleteProduct*() (удалить товар).

Класс Category:

Атрибуты:

1) *name* (название категории).
2) *productList* (список товаров в категории).

Методы:

3) *addCategory*() (добавить категорию).
4) *editCategory*() (редактировать категорию).
5) *deleteCategory*() (удалить категорию).

Класс ShoppingList:

Атрибуты:

1) *productList* (список товаров).

2) *totalCost* (общая стоимость списка).

Методы:

3) *addToShoppingList*() (добавить товар в список).
4) *removeFromShoppingList*() (удалить товар из списка).
5) *calculateTotalCost*() (рассчитать общую стоимость).

Класс PurchaseHistory:

Атрибуты:

1) *purchaseList* (список покупок).
2) *date* (дата покупки).

Методы:

3) *addPurchase*() (добавить покупку в историю).
4) *viewHistory*() (просмотреть историю покупок).

Класс ExpenseAnalyzer:

Атрибуты:

1) *expenseData* (данные о расходах).

Методы:

2) *analyzeExpenses*() (проанализировать расходы).
3) *comparePeriods*() (сравнить расходы за разные периоды).

Класс UserInterface:

Методы:

1) *displayMenu*() (отобразить главное меню).
2) *handleUserInput*() (обработать ввод пользователя).

Связи между классами:

- Класс *ShoppingList* содержит список объектов класса Product.
- Класс *Category* содержит список объектов класса Product.
- Класс *PurchaseHistory* связан с классом *ShoppingList* (история покупок хранит информацию о списках покупок).
- Класс *ExpenseAnalyzer* использует данные из класса *PurchaseHistory* для анализа расходов.

2.3 Описание последовательности системы

Текстовое описание:

Сценарий: Добавление товара в список покупок

1. Пользователь выбирает опцию *«Добавить товар»* в *UserInterface*.
2. *UserInterface* отправляет запрос в *ShoppingList* на добавление товара.
3. *ShoppingList* создаёт новый объект класса Product.
4. *ShoppingList* добавляет товар в список.
5. *ShoppingList* возвращает подтверждение в *UserInterface*.
6. *UserInterface* отображает обновлённый список пользователю.

Сценарий: Просмотр истории покупок

1. Пользователь выбирает опцию *«Просмотреть историю покупок»* в *UserInterface*.
2. *UserInterface* отправляет запрос в *PurchaseHistory*.
3. *PurchaseHistory* возвращает список покупок в *UserInterface*.
4. *UserInterface* отображает историю покупок пользователю.

Сценарий: Анализ расходов

1. Пользователь выбирает опцию *«Анализ расходов»* в *UserInterface*.
2. *UserInterface* отправляет запрос в *ExpenseAnalyzer*.
3. *ExpenseAnalyzer* запрашивает данные о расходах из *PurchaseHistory*.
4. *PurchaseHistory* возвращает данные в *ExpenseAnalyzer*.
5. *ExpenseAnalyzer* анализирует данные и возвращает результат в *UserInterface*.
6. *UserInterface* отображает результаты анализа пользователю.

3 ПРОГРАММНАЯ ЧАСТЬ

3.1 Реализация требований к системе

1. Язык программирования

Программа для создания и управления списками покупок разработана на языке программирования C++. Это обеспечивает высокую производительность и гибкость в управлении данными, а также позволяет использовать объектно-ориентированный подход для структурирования кода. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

2. Корректность работы

Программа стабильно работает на всех этапах: от запуска до завершения. Обеспечена корректная обработка данных, включая добавление, удаление и редактирование товаров, а также сохранение и загрузку списков покупок. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Часть 3.2, Рисунки 3.1-3.5).

3. Применение принципов ООП

В ходе разработки были применены ключевые принципы ООП, такие как инкапсуляция, наследование и полиморфизм. Это позволило создать модульную структуру программы с классами, такими как Product, Category, ShoppingList, что способствует расширяемости и поддерживаемости кода. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

4. Интерфейс пользователя

Программа оснащена интуитивно понятным интерфейсом, который позволяет пользователю легко добавлять, удалять и редактировать товары, а

также управлять категориями и просматривать историю покупок. Интерфейс разработан с учетом удобства использования и доступности для пользователей с различным уровнем опыта. Увидеть реализацию данного требования можно обратившись к скриншотам работы программы (Часть 3.2, Рисунки 3.1-3.5).

5. Инструкция по эксплуатации

Для обеспечения удобства использования программы разработана подробная инструкция по эксплуатации. Она включает описание основных функций программы, руководство по добавлению и редактированию товаров, а также рекомендации по работе с категориями и историей покупок. Увидеть реализацию данного требования можно обратившись к специально отведенной части (3.3 Инструкция по эксплуатации).

6. Управление вводом

Управление вводом в приложении реализовано с использованием клавиатуры ПК, что обеспечивает точный и отзывчивый прием команд пользователя и способствует эффективному взаимодействию с интерфейсом. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

7. Категоризация товаров

Программа поддерживает категоризацию товаров, что позволяет пользователю группировать товары по категориям (например, «Молочные продукты», «Овощи и фрукты», «Бакалея»). Категории можно создавать, редактировать и удалять. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

8. Расчёт стоимости

Программа позволяет пользователю вручную вводить цены товаров и автоматически рассчитывать общую стоимость списка покупок. Также программа запоминает цены товаров, что упрощает процесс расчёта в

будущем. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

9. История покупок

Программа сохраняет историю покупок, что позволяет пользователю отслеживать, какие товары он покупал ранее. История покупок может быть отфильтрована по дате или категории. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

10. Анализ расходов

Программа предоставляет возможность анализа расходов, включая статистику по тратам за определённый период. Пользователь может просматривать сводки по категориям и сравнивать расходы за разные периоды. Увидеть реализацию данного требования можно обратившись к исходному коду программы (Приложение А).

11. Система управления версиями

В процессе разработки программы используется система управления версиями Git. Это позволяет эффективно отслеживать изменения в коде, обеспечивать безопасность и возможность отката к предыдущим версиям. Увидеть реализацию данного требования можно обратившись к проекту, расположенному на github.

3.2 Функциональное тестирование программного продукта

Работа с программой начинается с вывода основного окна (Рисунок 3.1), где пользователь может добавлять товары и переходить во вторичные окна.

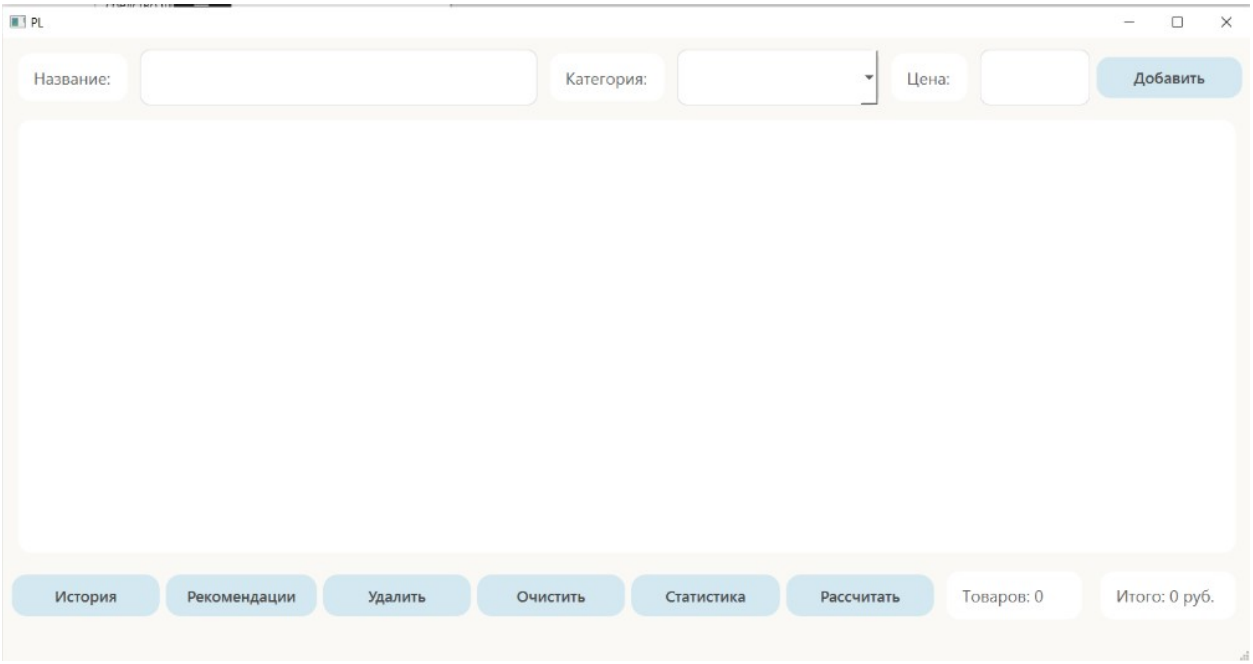

Рисунок 3.1 – Основное окно

Далее пользователь может ввести в соответствующие окошки название товара, его категорию и цену (Рисунок 3.2).

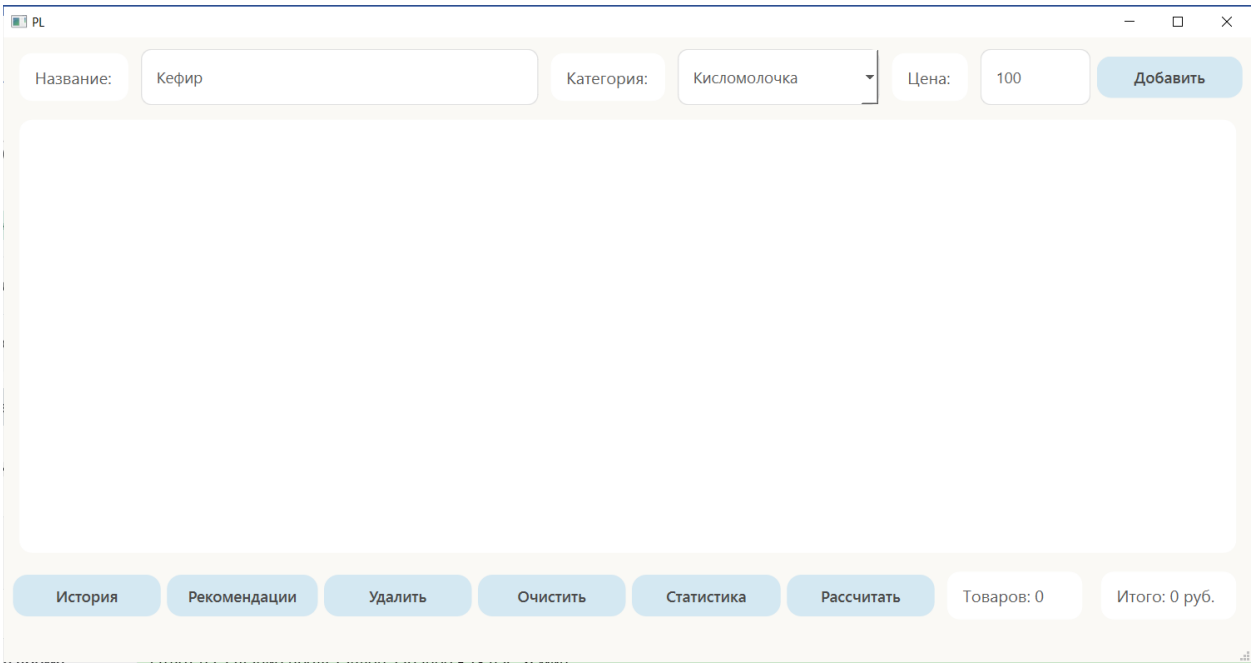

Рисунок 3.2 – Ввод соответствующего текста в окошки

При нажатии кнопки «Добавить» товар сохраняется на экране. При нажатии кнопки «Рассчитать» выводится общая стоимость всей покупки (Рисунок 3.3).

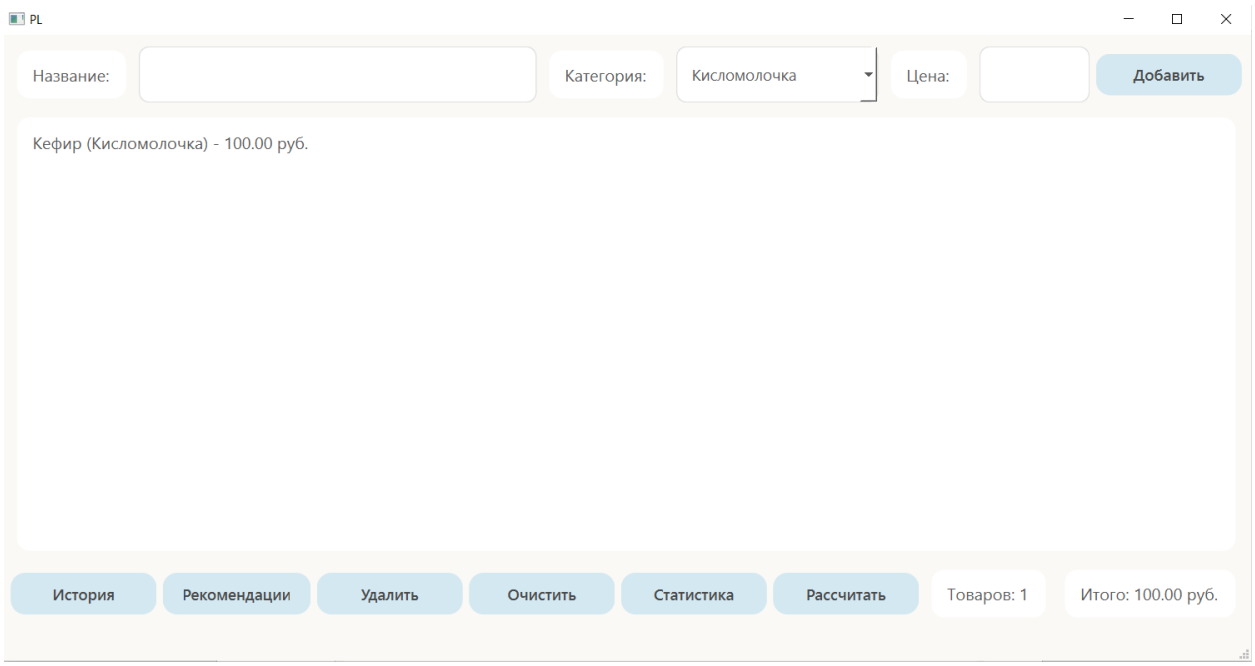

Рисунок 3.3 – Сохранение товара и подсчёт суммы

Пользователь может добавить любое количество товаров (Рисунок 3.4).

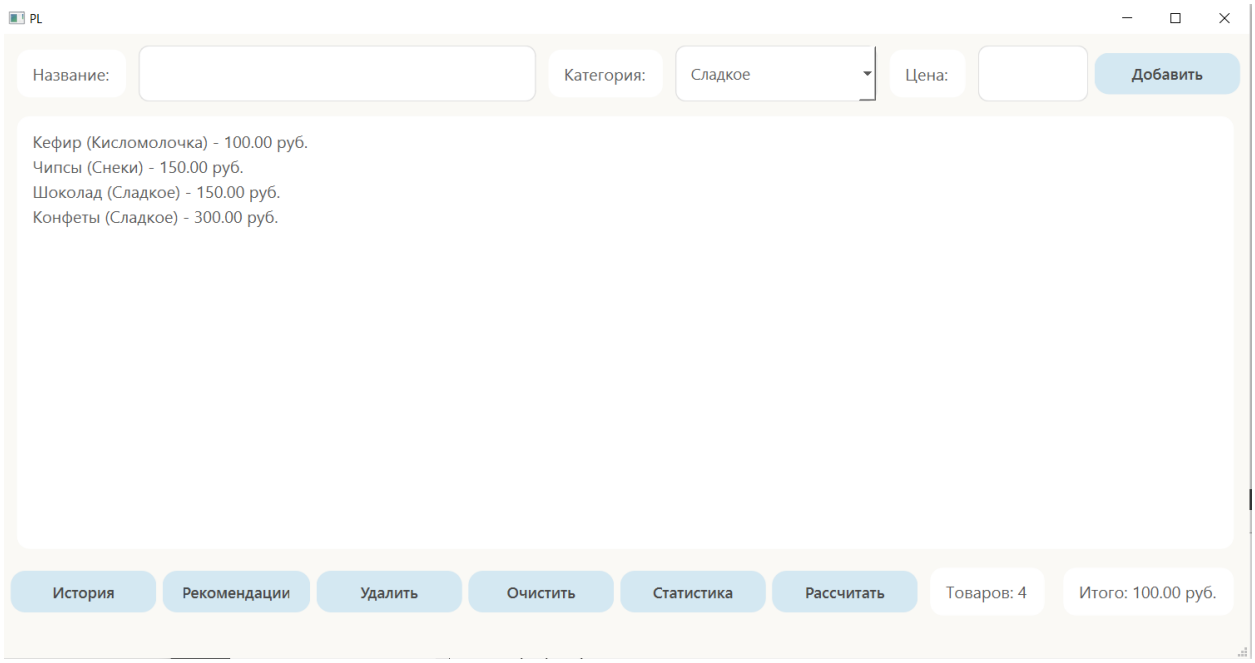

Рисунок 3.4 – Добавление нескольких товаров

В процессе добавления товаров, сохраняются и их категории, в которые в дальнейшем пользователь сможет добавить новые товары (Рисунок 3.5).

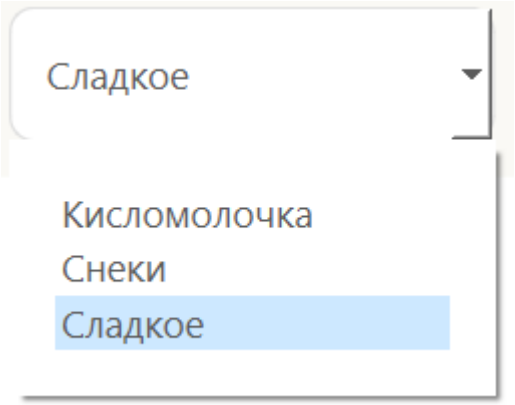

Рисунок 3.5 – Категории

Все товары, их стоимость и время покупки сохраняются в истории покупок (Рисунок 3.6).

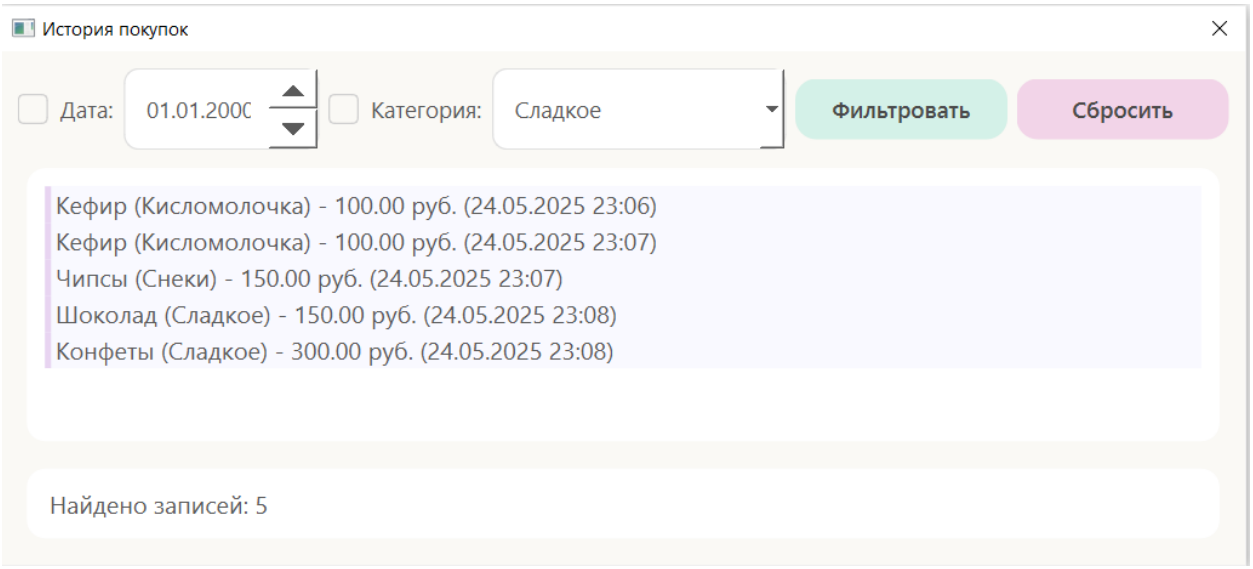

Рисунок 3.6 – История покупок

Товары можно фильтровать по дате и категории (Рисунок 3.7).

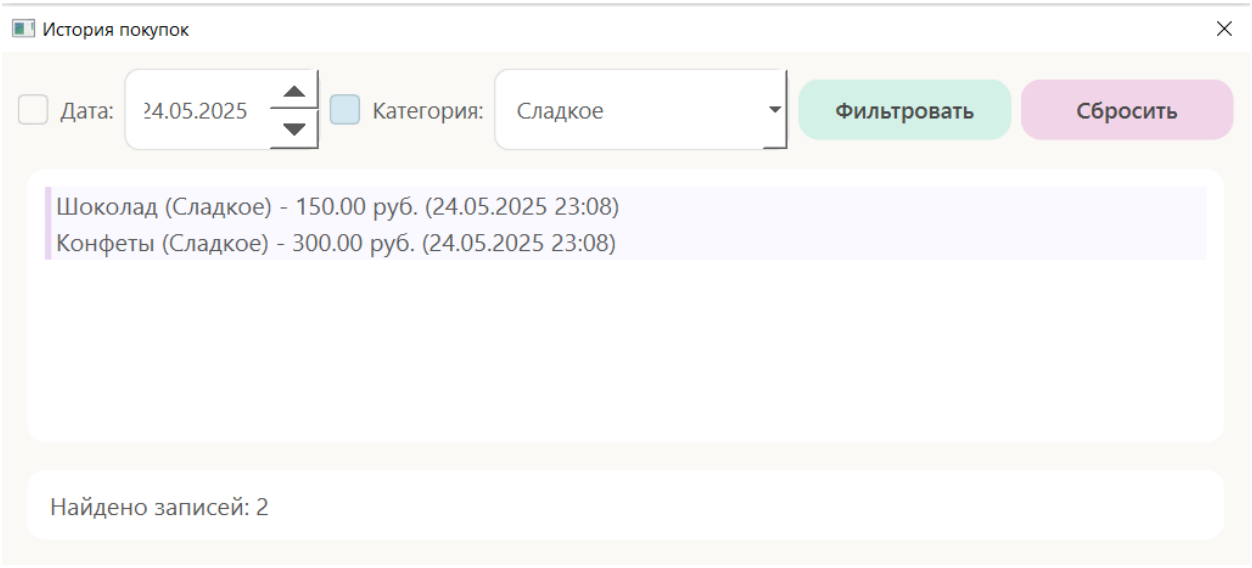

Рисунок 3.7 – Фильтрование товаров

В процессе добавления товаров формируются рекомендации, основанные на прошлых покупках (Рисунок 3.8).

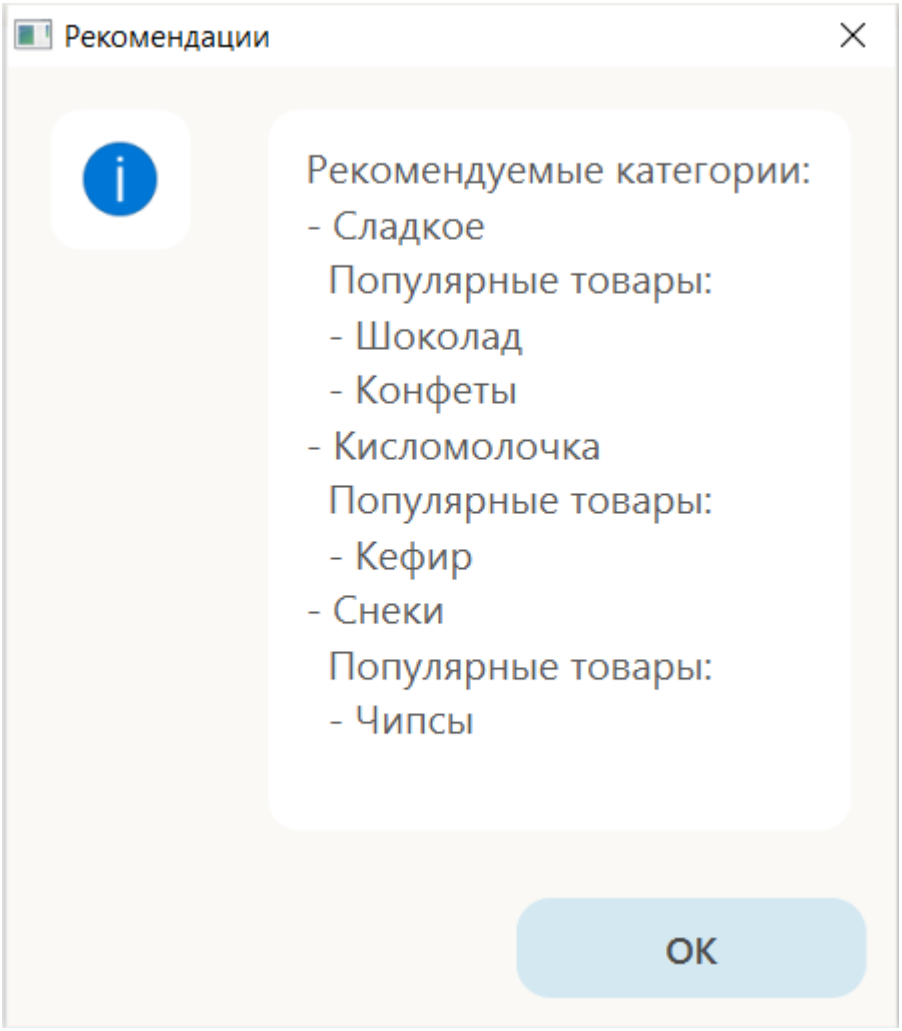

Рисунок 3.8 – Рекомендации

Во вкладке «Статистика расходов» пользователь сможет рассчитать стоимость купленных товаров и сравнить расходы в конкретные периоды (Рисунок 3.9), (Рисунок 3.10).

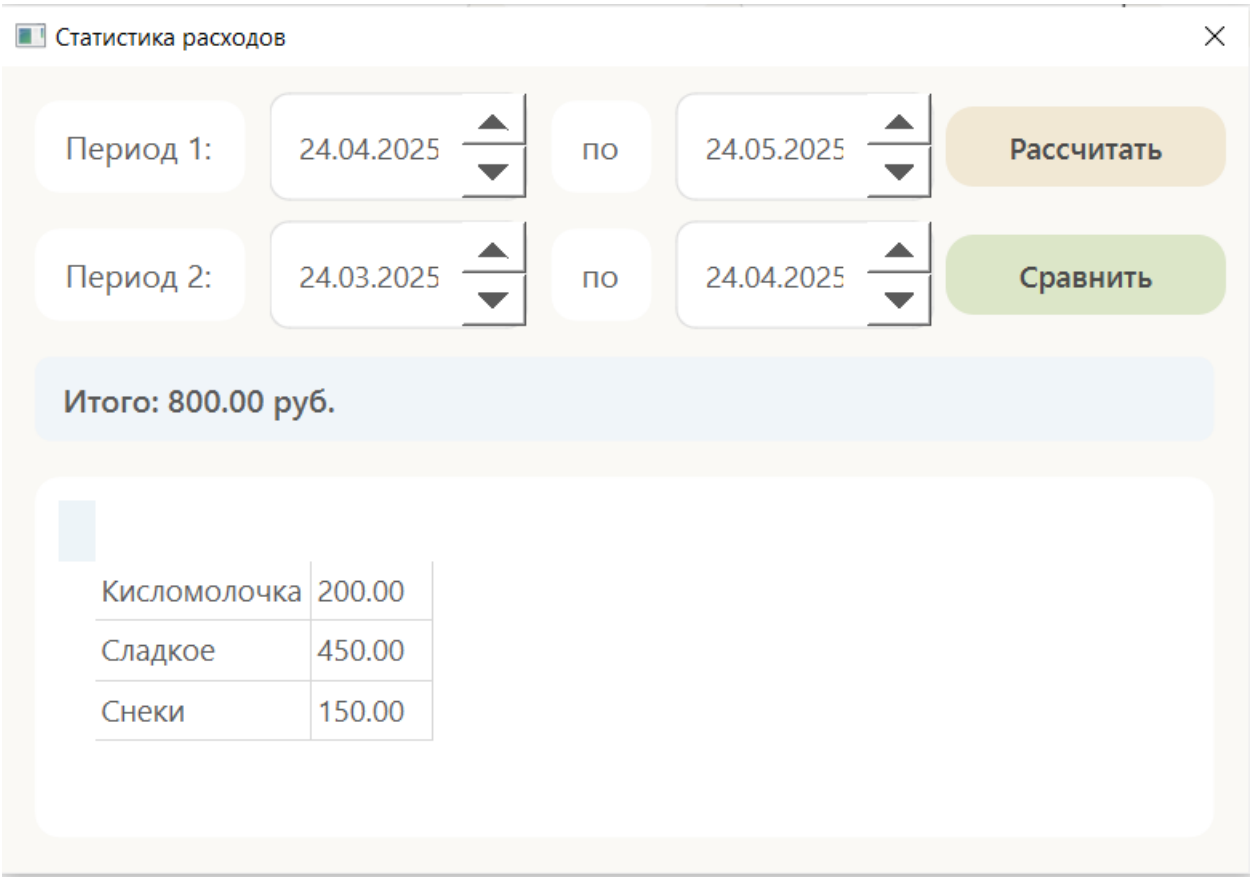

Рисунок 3.9 – Расчёт за период

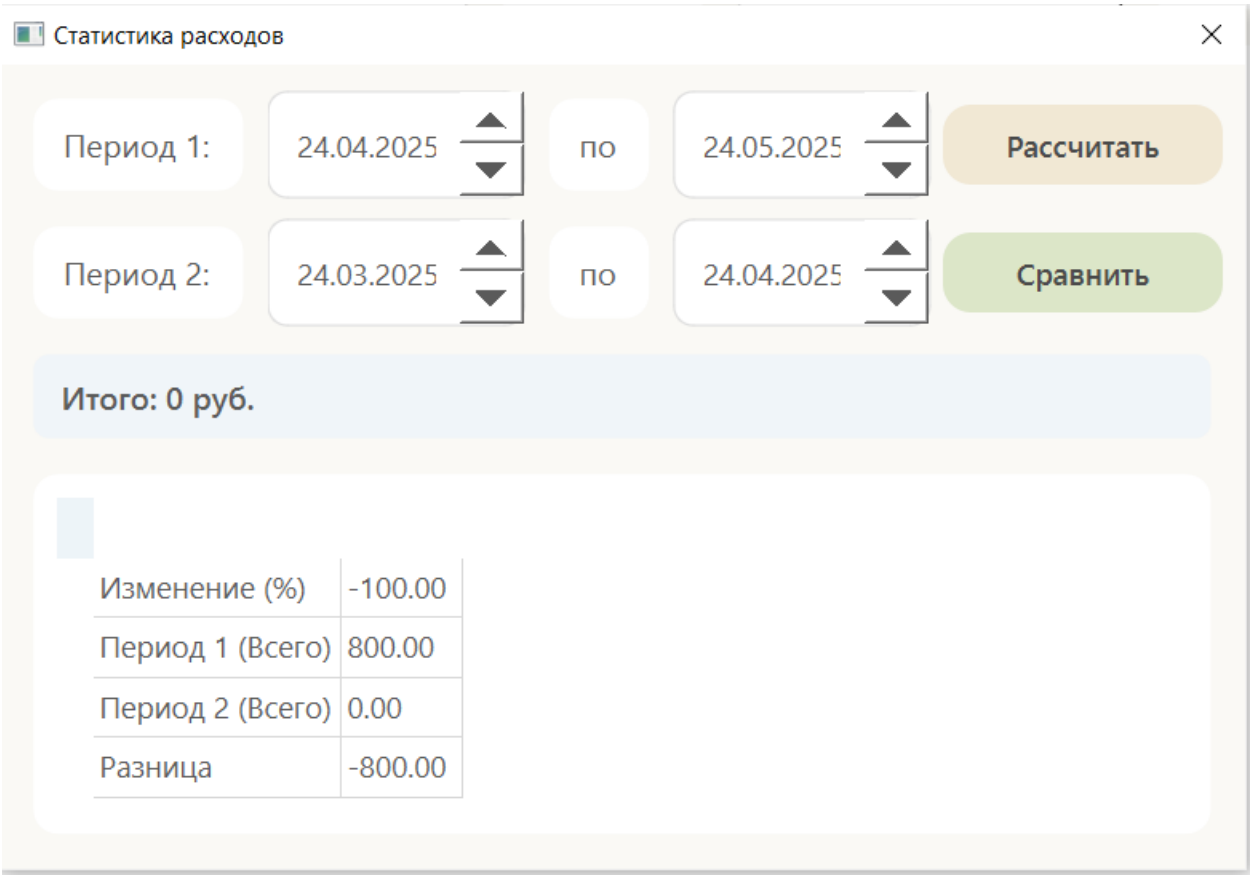

Рисунок 3.10 – Сравнение периодов

3.3 Инструкция по эксплуатации

После запуска программы пользователь попадает на главное окно приложения «Список покупок». В этом окне пользователь может выполнять следующие действия:

1. Добавление товаров в список покупок
   1) В поле «Название» введите наименование товара.
   2) В поле «Категория» выберите или введите категорию товара (например, «Продукты», «Одежда» и т.д.).
   3) В поле «Цена» укажите стоимость товара.
   4) Нажмите кнопку «Добавить», чтобы включить товар в список покупок.
2. Управление списком покупок

1) Удаление товара: Выберите товар из списка и нажмите кнопку «Удалить».
2) Очистка списка: Нажмите кнопку «Очистить», чтобы удалить все товары из списка.
3) Расчет общей суммы: Нажмите кнопку «Рассчитать», чтобы отобразить общую стоимость всех товаров в списке. Сумма отобразится в поле «Итого».

3. Просмотр истории покупок
   1) Нажмите кнопку «История», чтобы открыть окно с историей покупок.
   2) В окне истории можно: фильтровать записи по дате или категории, используя соответствующие чекбоксы и выпадающие списки нажать кнопку «Сбросить», чтобы отменить фильтрацию и отобразить все записи; просмотреть количество найденных записей в поле «Найдено записей».
4. Получение рекомендаций
   1) Нажмите кнопку «Рекомендации», чтобы увидеть список рекомендуемых категорий и популярных товаров на основе истории покупок.
5. Просмотр статистики расходов
   1) Нажмите кнопку «Статистика», чтобы открыть окно статистики.
   2) В окне статистики можно: указать период для анализа расходов и нажать кнопку «Рассчитать», чтобы увидеть сумму расходов по категориям; сравнить расходы за два разных периода, нажав кнопку «Сравнить», результаты сравнения отобразятся в таблице; просмотреть общую сумму расходов за выбранный период в поле «Итого».
6. Закрытие программы

1) Для завершения работы программы закройте главное окно, нажав на крестик в правом верхнем углу. Все данные автоматически сохранятся в файл.

Программа предназначена для удобного ведения списка покупок, анализа расходов и получения рекомендаций на основе истории покупок. Все изменения сохраняются автоматически, что обеспечивает безопасность данных.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы была разработана программа для создания и управления списками покупок, соответствующая всем поставленным требованиям. На этапе проектирования проведён анализ существующих решений, сформулированы функциональные и технические требования к системе, а также разработаны диаграммы состояний, классов и последовательностей, отражающие логику работы приложения.

Программный продукт реализован на языке C++ с применением принципов объектно-ориентированного программирования, что обеспечило модульность, расширяемость и поддерживаемость кода. Основные функции, такие как категоризация товаров, история покупок, автоматическое заполнение списков, анализ расходов и расчёт стоимости, были успешно интегрированы в систему. Пользовательский интерфейс разработан с учётом удобства и интуитивной понятности.

В процессе разработки проведено тестирование, подтвердившее стабильность работы программы на всех этапах — от запуска до завершения. Для управления версиями кода использовалась система **Git**, что позволило эффективно отслеживать изменения и обеспечивать контроль над разработкой.

Курсовая работа продемонстрировала не только владение технологиями программирования, но и умение анализировать предметную область, проектировать архитектуру приложения и реализовывать сложные функциональные требования. Полученный опыт в области ООП, работы с данными и проектирования интерфейсов стал важным этапом профессионального роста.

Разработанный продукт обладает потенциалом для дальнейшего развития. Перспективными направлениями усовершенствования являются: добавление мобильной версии приложения; интеграция с онлайн-магазинами для автоматического обновления цен; внедрение функции сканирования чеков; улучшение визуализации статистики расходов.

Проделанная работа заложила прочную основу для будущих исследований и разработок в области программного обеспечения для управления повседневными задачами. Приобретённые знания и навыки будут применяться в дальнейшем обучении и профессиональной деятельности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Bring! — Shopping List / [Электронный ресурс] // Приложение для создания и редактирования списков покупок: [сайт]. — URL: https://www.getbring.com (дата обращения: 16.03.2025).

2. Out of Milk / [Электронный ресурс] // Приложение для создания и редактирования списков покупок: [сайт]. — URL: https://www.outofmilk.com (дата обращения: 16.03.2025).

3. AnyList / [Электронный ресурс] // Приложение для создания и редактирования списков покупок: [сайт]. — URL: https://www.anylist.com (дата обращения: 16.03.2025).

4. Google Keep / [Электронный ресурс] // Приложение для создания и редактирования списков покупок: [сайт]. — URL: https://keep.google.com (дата обращения: 16.03.2025).

5. YNAB (You Need A Budget) / [Электронный ресурс] // Приложение для создания и редактирования списков покупок: [сайт] — URL: https://www.ynab.com (дата обращения: 16.03.2025).

6. Listonic / [Электронный ресурс] // Приложение для создания и редактирования списков покупок: [сайт] — URL: https://www.listonic.com (дата обращения: 16.03.2025).

ПРИЛОЖЕНИЯ

Приложение А – Исходный код центрального класса

Приложение А – Исходный код центрального класса

Листинг А.1 – Исходный код программы

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include "historywindow.h"
#include "productrecommender.h"
#include "expensestatistics.h"
#include "statisticswindow.h"
#include <QMessageBox>
#include <QCloseEvent>
#include <QDebug>
#include <QApplication>

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    ////СТАТИСТИКА/////
    QPushButton *statisticsButton = new QPushButton("Статистика", this);
    statisticsButton->setStyleSheet("background-color: #D4E8F2;");
    ui->horizontalLayout_2->insertWidget(2, statisticsButton);
    connect(statisticsButton, &QPushButton::clicked, this,
&MainWindow::on_statisticsButton_clicked);

    //ЗАГРУЗКА ИЗ ФАЙЛА/////
    if (!purchaseHistory.loadFromFile(HISTORY_FILE)) {
        QMessageBox::warning(this, "Ошибка загрузки",
                             "Не удалось загрузить историю покупок. Будет
создана новая история.\n"
                             "Проверьте целостность файла " + HISTORY_FILE);
    }

    loadCategories();

    updateProductList();


    // Добавляем кнопки в интерфейс
    QPushButton *historyButton = new QPushButton("История", this);
    historyButton->setStyleSheet("background-color: #D4E8F2;");
    QPushButton *recommendationsButton = new QPushButton("Рекомендации",
this);
    recommendationsButton->setStyleSheet("background-color: #D4E8F2;");

    ui->horizontalLayout_2->insertWidget(0, historyButton);
    ui->horizontalLayout_2->insertWidget(1, recommendationsButton);

    connect(historyButton, &QPushButton::clicked, this,
&MainWindow::on_historyButton_clicked);
    connect(recommendationsButton, &QPushButton::clicked, this,
&MainWindow::on_recommendationsButton_clicked);

}

MainWindow::~MainWindow()
{
    delete ui;
}
```

Продолжение Листинга А.1

```
void MainWindow::on_addButton_clicked()
{
    QString name = ui->nameEdit->text();
    QString category = ui->categoryCombo->currentText();
    double price = ui->priceEdit->text().toDouble();

    if (name.isEmpty() || category.isEmpty()) {
        QMessageBox::warning(this, "Ошибка", "Заполните все поля");
        return;
    }

    // Добавляем категорию, если её нет в списке
    if (ui->categoryCombo->findText(category) == -1) {
        ui->categoryCombo->addItem(category);
        saveCategoriesToFile(); // Сохраняем обновленный список категорий
    }

    Product product(name, category, price);
    shoppingList.addProduct(product);
    purchaseHistory.addRecord(product);
    updateProductList();

    ui->nameEdit->clear();
    ui->priceEdit->clear();
}


void MainWindow::on_removeButton_clicked()
{
    int row = ui->productList->currentRow();
    if (row == -1) {
        QMessageBox::warning(this, "Ошибка", "Выберите товар для удаления");
        return;
    }

    shoppingList.removeProduct(row);
    updateProductList();
}

void MainWindow::on_clearButton_clicked()
{
    shoppingList.clear();
    updateProductList();
}

void MainWindow::on_calculateButton_clicked()
{
    double total = shoppingList.calculateTotalCost();
    ui->totalLabel->setText(QString("Итого: %1 руб.").arg(total, 0, 'f', 2));
}

void MainWindow::updateProductList()
{
    ui->productList->clear();
    const QList<Product>& products = shoppingList.getProducts();

    for (const Product& product : products) {
        QString itemText = QString("%1 (%2) - %3 руб.")
                               .arg(product.getName())
```

Продолжение Листинга А.1

```
                                .arg(product.getCategory())
                                .arg(product.getPrice(), 0, 'f', 2);
        ui->productList->addItem(itemText);
    }

    ui->countLabel->setText(QString("Товаров:
%1").arg(shoppingList.getProductCount()));
}

//////////////////////HISTORY_AND_RECOMENDATION_BUTTON/////////////////

void MainWindow::on_historyButton_clicked()
{
    HistoryWindow *historyWindow = new HistoryWindow(purchaseHistory, this);
    historyWindow->exec();
    delete historyWindow;
}

void MainWindow::on_recommendationsButton_clicked()
{
    ProductRecommender recommender(purchaseHistory);
    QStringList categories = recommender.getRecommendedCategories();

    QString message = "Рекомендуемые категории:\n";
    for (const QString& category : categories) {
        message += "- " + category + "\n";

        QStringList products = recommender.getRecommendedProducts(category);
        if (!products.isEmpty()) {
            message += "  Популярные товары:\n";
            for (const QString& product : products) {
                message += "  - " + product + "\n";
            }
        }
    }

    QMessageBox::information(this, "Рекомендации", message);
}
////////////////JSON////////////////
void MainWindow::closeEvent(QCloseEvent *event) {
    if (!purchaseHistory.saveToFile(HISTORY_FILE)) {
        QMessageBox::critical(this, "Ошибка сохранения",
                              "Не удалось сохранить историю покупок!\n"
                              "Ваши данные могут быть потеряны.");
    }

    // Сохраняем категории при закрытии
    saveCategoriesToFile();

    event->accept();
}
void MainWindow::loadHistory()
{
    purchaseHistory.loadFromFile(HISTORY_FILE);
}

void MainWindow::saveHistory()
{
    purchaseHistory.saveToFile(HISTORY_FILE);
}
```

Продолжение Листинга А.1

```
bool MainWindow::checkHistoryFileIntegrity() {
    QFile file(HISTORY_FILE);
    if (!file.exists()) return true;

    if (!file.open(QIODevice::ReadOnly)) return false;

    QByteArray data = file.readAll();
    file.close();

    QJsonParseError error;
    QJsonDocument::fromJson(data, &error);

    return (error.error == QJsonParseError::NoError);
}
////////////////STATS//////
void MainWindow::on_statisticsButton_clicked() {
    ExpenseStatistics stats(purchaseHistory);
    StatisticsWindow *statsWindow = new StatisticsWindow(stats, this);
    statsWindow->exec();
    delete statsWindow;
}


//////////CATEGORIES////////
void MainWindow::saveCategoriesToFile() {
    QFile file("categories.json");
    if (!file.open(QIODevice::WriteOnly)) {
        qWarning() << "Could not open categories file for writing";
        return;
    }

    QJsonArray categoriesArray;
    for (int i = 0; i < ui->categoryCombo->count(); ++i) {
        categoriesArray.append(ui->categoryCombo->itemText(i));
    }

    QJsonDocument doc(categoriesArray);
    file.write(doc.toJson());
    file.close();
}

void MainWindow::loadCategoriesFromFile() {
    QFile file("categories.json");
    if (!file.exists()) return;

    if (!file.open(QIODevice::ReadOnly)) {
        qWarning() << "Could not open categories file for reading";
        return;
    }

    QByteArray data = file.readAll();
    file.close();

    QJsonParseError error;
    QJsonDocument doc = QJsonDocument::fromJson(data, &error);
    if (error.error != QJsonParseError::NoError) {
        qWarning() << "Error parsing categories file:" <<
error.errorString();
        return;
    }
```

Продолжение Листинга А.1

```
    if (!doc.isArray()) {
        qWarning() << "Categories file is not an array";
        return;
    }

    QJsonArray categoriesArray = doc.array();
    for (const QJsonValue& value : categoriesArray) {
        QString category = value.toString();
        if (!category.isEmpty() && ui->categoryCombo->findText(category) ==
-1) {
            ui->categoryCombo->addItem(category);
        }
    }
}

void MainWindow::loadCategories()
{
    ui->categoryCombo->clear();

    // Сначала загружаем сохраненные категории
    loadCategoriesFromFile();

    // Затем добавляем категории из истории покупок
    QSet<QString> categories;
    for (const auto& record : purchaseHistory.getRecords()) {
        categories.insert(record.getProduct().getCategory());
    }

    for (const QString& category : categories) {
        if (ui->categoryCombo->findText(category) == -1) {
            ui->categoryCombo->addItem(category);
        }
    }
}
```