Учреждение образования

«Гродненский государственный политехнический колледж»

**ОТЧЁТ  
ПО УЧЕБНОЙ ПРАКТИКЕ ПО ПРОГРАММИРОВАНИЮ**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Учащегося |  | 3 | | | |  | курса, группы |  | ПЗТ-40 |
| специальности |  | | 2 - 40 01 01 «Программное обеспечение информационных | | | | | | |
|  |  | | технологий» | | | | | | |
| Место прохождения практики | | | |  | УО «Гродненский государственный | | | | |
|  | | | |  | политехнический колледж» | | | | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тема проекта: |  | «Разработка десктопного приложения «FinanceFlow» для анализа, | | |
|  |  | учёта и планирования личных финансов» | | |
| Ссылка на проект: | | |  | https://github.com/ВПИСАТЬ |
|  | | |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Выполнил |  |  |  | А.С. Кирилюк |
|  |  |  |  | (инициалы, фамилия) |
| Руководитель практики от колледжа |  |  |  | А.Г. Бабуль |
|  |  |  | (инициалы, фамилия) |

Содержание

[Введение. Описание структуры предприятия 3](#_Toc194324354)

[1 Анализ предметной области и формулировка требований к программе](#_Toc194324355) 5

[1.1 Исследование предметной области 5](#_Toc194324356)

[1.2 Инструменты разработки 5](#_Toc194324357)

[2 Проектирование](#_Toc194324358) 7

[2.1 Диаграмма вариантов использования](#_Toc194324359) 7

[2.2 Диаграмма деятельности](#_Toc194324360) 8

[2.3 Описание тестов](#_Toc194324361) 10

[3 Построение программы 11](#_Toc194324362)

[4 Тестирование 12](#_Toc194324363)

[5 Применение 13](#_Toc194324364)

[5.1 Назначение и условия применения программы 13](#_Toc194324365)

[5.2 Инсталляция 13](#_Toc194324366)

[5.3 Выполнение программы](#_Toc194324367) 15

[Заключение](#_Toc194324368) 19

[Список использованных источников](#_Toc194324369) 20

[Приложение A Листинг программы](#_Toc194324370) 21

Введение. Описание структуры предприятия

На данной учебной практике была поставлена задача разработать десктопное приложение для управления личными финансами. Данный программный продукт позволит пользователю эффективно анализировать доходы и расходы, ставить финансовые цели и визуализировать прогресс, создавая интуитивный и удобный опыт управления бюджетом.

Пользователи получат удовольствие от процесса финансового планирования, открытия новых возможностей для экономии и достижения финансовой независимости. FinanceFlow объединяет детальный учет, аналитику, прогнозирование и визуальные элементы, чтобы сделать финансовый менеджмент простым и увлекательным.

Создаваемое приложение ориентировано на пользователей старше 15 лет, которые стремятся упростить управление личными финансами и достичь финансовой стабильности.

Далее приведено краткое описание разделов пояснительной записки.

Первый раздел носит название «Анализ предметной области и формулировка требований к программе». В нем можно ознакомиться с постановкой задачи, которая включает в себя: исследование предметной области поставленной задачи, функциональные и нефункциональные требования к программному продукту. В подразделе «Инструменты разработки» рассмотрена среда, в которой создается данный программный продукт. Здесь также установлены минимальные требования к аппаратным характеристикам, обеспечивающим правильное функционирование поставленной задачей.

В разделе «Проектирование задачи» рассмотрены основные аспекты разработки программного продукта. Здесь можно узнать об организации данных в контексте среды разработки. В данном разделе будут составлены алгоритмы процесса обработки информации.

«Построение программы» – третий раздел отчета по практике, в котором описываются все элементы и объекты, которые использованы при реализации данного приложения. В этом разделе четко описаны функции пользователя и их структура.

Четвертый раздел – «Тестирование». В нем описано функциональное тестирование данной программы, смоделированы все основные действия пользователя при работе с программой.

В разделе «Применение» будет описано назначение программы, область применения и среда функционирования программного обеспечения.

«Заключение» содержит краткую формулировку задачи, результаты проделанной работы, описание использованных методов и средств, описание степени автоматизации процессов на различных этапах разработки.

В разделе «Список использованных источников» приведен список используемых при разработке источников.

В приложении А будет приведен листинг программы.

1. Анализ предметной области и формулировка требований к программе
   1. Исследование предметной области

На данной учебной практике поставлена задача разработать десктопное приложение, которое поможет пользователю упростить учет доходов и расходов, анализировать финансовые потоки и планировать бюджет.

FinanceFlow создаст интуитивный и удобный опыт управления личными финансами, позволяя пользователю в несколько шагов фиксировать операции, отслеживать статистику и формировать отчетность в Excel.

Функциональные возможности

* Выбор типа операции: доход или расход
* Заполнение данных: сумма, категория, описание, дата
* Добавление записи в общий список
* Просмотр статистики по доходам и расходам
* Графическая аналитика с диаграммами и отчетами
* Экспорт данных в Excel, где все операции представлены в виде таблиц и графиков.

Назначение продукта

Приложение FinanceFlow ориентировано на пользователей, которым важно структурированно вести учет личных финансов, анализировать расходы и доходы, а также планировать бюджет. Это инструмент для тех, кто хочет упростить процесс учета и получить визуально наглядные отчеты.

FinanceFlow позволяет гибко управлять финансами, анализировать тенденции расходов и доходов, а также формировать удобные отчеты для дальнейшего использования.

* 1. Инструменты разработки

Для разработки данного проекта выбрана среда Visual Studio — мощная интегрированная среда разработки (IDE), обеспечивающая удобные инструменты для написания, тестирования и отладки программного кода.

Основным языком программирования проекта является C# — современный, универсальный язык, который активно развивается и применяется как в образовательных целях, так и при разработке крупных коммерческих решений.

Иные инструменты, используемые при разработке и написании сопутствующей документации:

* WEB-ресурс DRAW.IO – будет использоваться для создания графической части и разработки UML-диаграмм;
* Microsoft Office Word 2013 – для написания документации к программному продукту;
* Smart Install Maker – утилита для создания инсталляторов;
* Dr.Explain – инструмент разработки пользовательской документации;
* GitHub – веб-сервис для хостинга IT-проектов;
* Google Drive – сервис хранения, редактирования и синхронизации файлов, разработанный компанией Google.

Разработка ведется на ноутбуке ASUS Vivobook. У данного ноутбука следующие параметры:

* процессор 12th Gen Intel(R) Core(TM) i3-1220P 1.50 GHz;
* объем ОЗУ 4 гб;
* оперативная система 8 ГБ
* объем места на HDD – 500 гб ;
* видеоподсистема 1920х1200 точек с глубиной цвета 16 Bit;
* ОС – Windows 10 Pro.

1. Проектирование
   1. Диаграмма вариантов использования

Диаграмма вариантов использования – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

Суть такой диаграммы состоит в следующем: проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью так называемых вариантов использования.

В данной диаграмме вариантов использования актером является пользователь.

Пользователю доступны следующие варианты использования: кнопка "+", статистика, история, отчет.

У варианта использования кнопка "+" наблюдаются следующие «extend»-связи: Выбрать тип операции (доход/расход), Ввести сумму, Выбрать категорию (продукты, транспорт, зарплата и т. д.), Написать описание, Выбрать дату, Добавить в список операций.

У варианта использования «Статистика» наблюдаются следующие «extend»-связи: Отобразить диаграмму расходов, Отобразить диаграмму доходов.

У варианта использования «Отчет» наблюдаются следующие «extend»-связи: Отображение списка операций, Сохранение отчета в Excel.

У варианта использования «Сохранение отчета в Excel» наблюдаются следующие «include»-связи: Сгенерировать таблицу с данными, Добавить диаграммы расходов и доходов, Экспортировать в файл.

Диаграмма вариантов использования представлена на рисунке 1.

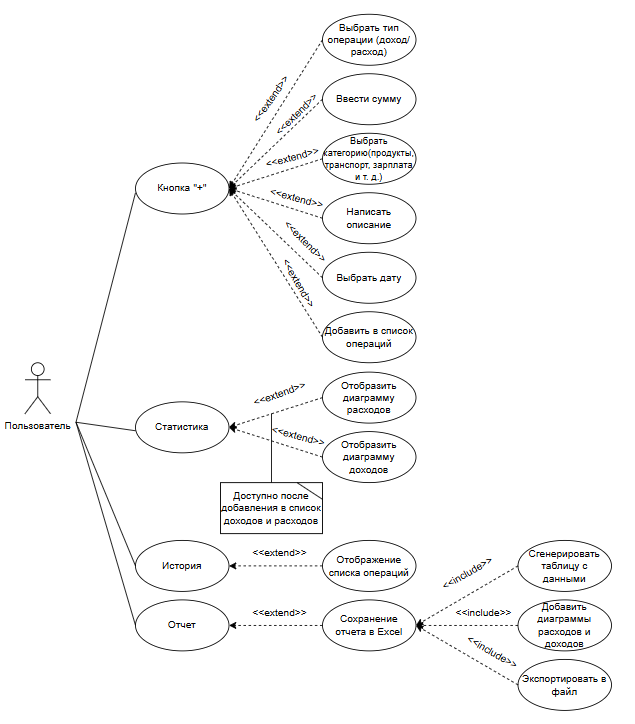


Рисунок 1 – Диаграмма вариантов использования

* 1. Диаграмма деятельности

Диаграмма деятельности – это поведенческая диаграмма, которая показывает поток работы или действий в рамках системы или процесса. Она отображает последовательность шагов и возможные варианты выполнения работы, включая параллельные процессы и ветвления. Диаграмма деятельности включает в себя такие элементы, как начальные и конечные узлы, узлы действий, узлы решений, вилки и слияния, а также потоки управления, которые связывают эти узлы.

На диаграмме отображен процесс добавления финансовой операции в список.

Диаграмма деятельности представлена на рисунке 2.

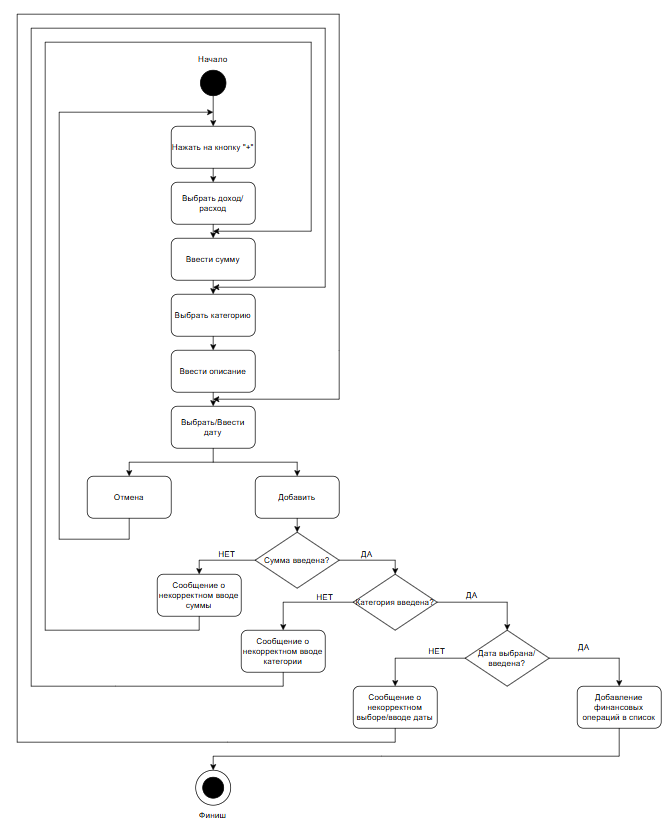


Рисунок 2 – Диаграмма деятельности

* 1. Описание тестов

При разработке приложения необходимо будет провести тест по игровому процессу

Таблица 1 – Тест-кейсы с крайне высоким приоритетом

|  |  |
| --- | --- |
| **Заглавие и шаги выполнения** | **Ожидаемый результат** |
| **1** | **2** |
| 1. Запуск приложения «FinanceFlow» | 1. Приложение запускается. Приложение запускается. Экран загрузки отображается. Появляется главное окно с кнопками: «+» (Добавить операцию), «Статистика», «История», «Отчет». |
| 1. Добавление финансовой операции «+» | 1. Нажать кнопку «+». Выбрать тип операции: доход/расход. Ввести сумму. Выбрать категорию. Написать описание. Указать дату. Нажать кнопку «Добавить». Операция успешно добавлена в список транзакций. Статистика обновлена. |
| 1. Валидация данных при добавлении операции | 1. Нажать «+». Оставить поле суммы пустым. Оставить поле категории пустым. Оставить поле даты пустым. Нажать «Добавить». Окно с предупреждением: «Некорректный формат суммы/Укажите категорию/Укажите дату». Операция не добавляется до корректного ввода данных. |
| 1. Отмена добавления операции | 1. Нажать «+». Заполнить все поля. Нажать «Отмена». Окно закрывается, данные не сохраняются. |
| 1. Просмотр статистики | 1. Нажать «Статистика». Отображены диаграммы доходов и расходов. Корректные расчеты по данным пользователя. |
| 1. Экспорт отчета в Excel | 1. **Нажать «Отчет».** Выбрать расположение отчета и дать название. Нажать сохранить **«Сохранить». Файл Excel создан, содержит таблицу и диаграммы.** |

1. Построение программы

Диаграмма классов — это один из видов диаграмм в языке моделирования UML (Unified Modeling Language), который используется для визуализации структуры системы. Она отображает классы, их свойства и методы, а также показывает, как они взаимосвязаны друг с другом. Диаграмма классов используется для проектирования архитектуры программного обеспечения, анализа логики работы системы и документирования её структуры.

Диаграмма классов представлена на рисунке 3.

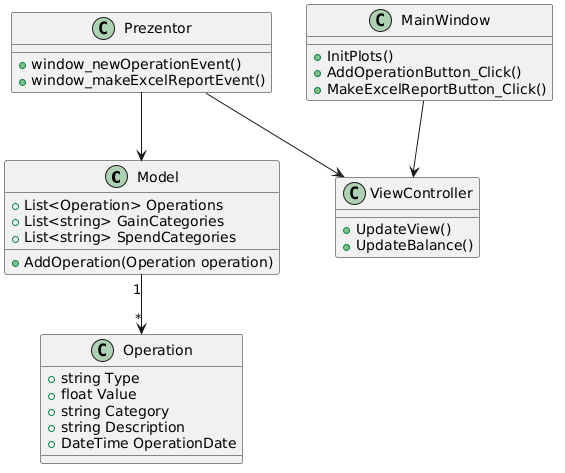


Рисунок 3 – Диаграмма классов

1. Тестирование

При разработке данной программы многие возникающие ошибки и недоработки были исправлены на этапе реализации проекта. После завершения испытания реализации программы было проведено тщательное функциональное тестирование. Функциональное тестирование должно гарантировать работу всех элементов программы в автономном режиме.

Элементы программы были проверены, и было установлено, что все они работают правильно и выполняют задачи, указанные в классах.

Расписание проведения и время, затраченное на тестирование, описано в таблице 2.

Таблица 2 – Расписание работ над проектом

|  |  |  |  |
| --- | --- | --- | --- |
| **Имя** | **Дата** | **Деятельность** | **Продолжительность, ч** |
| Кирилюк Ангелина | 29.04.2025 | Разработка тест-кейсов | 2 |
| Кирилюк Ангелина | 28.04.2025 | Тестирование функционала приложения | 2 |
| Кирилюк Ангелина | 04.05.2025 | Исправление найденных ошибок | 3 |
| Кирилюк Ангелина | 06.05.2025 | Проведение регрессионного тестирования | 2 |
| Кирилюк Ангелина | 08.05.2025 | Составление отчета о результатах тестирования | 3 |

Элементы программы были проверены, и было установлено, что все они работают правильно и выполняют задачи, указанные в процедурах.

Статистика по всем дефектам представлена в таблице 3.

Таблица 3 – Статистика по всем дефектам

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Статус** | **Количество** | **Важность** | | | |
| **Низкая** | **Средняя** | **Высокая** | **Критическая** |
| Найдено | 0 | 0 | 0 | 0 | 0 |
| Исправлено | 0 | 0 | 0 | 0 | 0 |
| Проверено | 0 | 0 | 0 | 0 | 0 |
| Открыто заново | 0 | 0 | 0 | 0 | 0 |
| Отклонено | 0 | 0 | 0 | 0 | 0 |

1. Применение
   1. Назначение и условия применения программы

Цель данного программного продукта заключается в разработке десктопного приложения, которое позволит пользователю упростить управление личными финансами, получая эстетическое удовольствие от удобного интерфейса и наглядных аналитических инструментов.

FinanceFlow создаст неповторимый пользовательский опыт, сочетая планирование, анализ, учет и визуальную отчетность, помогая пользователю рационально управлять бюджетом, отслеживать динамику доходов и расходов, а также формировать полезные финансовые привычки.

Пользователь сможет легко добавлять операции, анализировать статистику и экспортировать данные в удобный отчет, что делает FinanceFlow интуитивным, функциональным и незаменимым инструментом для управления финансами.

Создаваемое приложение будет рассчитано на любого пользователя.

Качество и скорость работы приложения всегда зависит от самих характеристик персонального компьютера. Поэтому приложение должно было быть протестировано на разных машинах. Тестирование проводилось на разных персональных компьютерах и результаты были удовлетворительные.

Сама программа была разработана на программном устройстве со следующими характеристиками:

* процессор 12th Gen Intel(R) Core(TM) i3-1220P 1.50 GHz;
* объем ОЗУ 4 гб;
* оперативная система 8 ГБ
* объем места на HDD – 500 гб ;
* видеоподсистема 1920х1200 точек с глубиной цвета 16 Bit;
* ОС – Windows 10 Pro.
  1. Инсталляция

Для того, чтобы установить программу необходимо запустить файл Inno Setup.exe. Появляется возможность выбора места для установки программного продукта, представленное на рисунке 4.

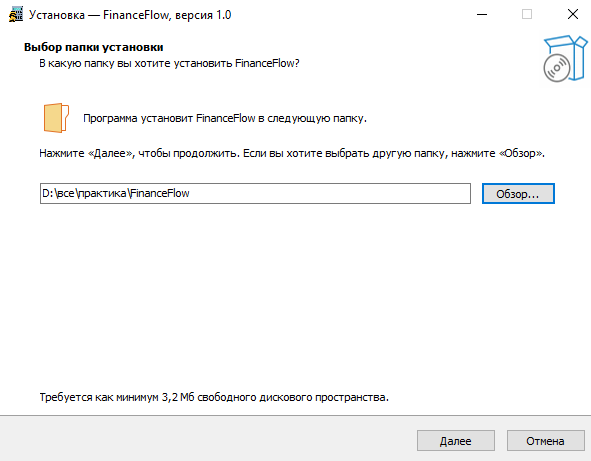


Рисунок 4 – Мастер установки игрового приложения «Путь авантюриста»

После нажатия на кнопку «Далее» появляется возможность выбрать создавать ли ярлыка игрового приложения, представленная на рисунке 5.

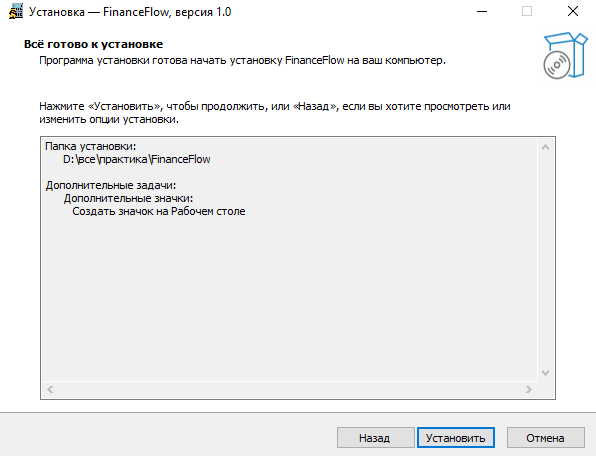


Рисунок 5 – Создание ярлыка

После нажатия на кнопку «Далее» открывается форма завершения установки, представленная на рисунке 6.

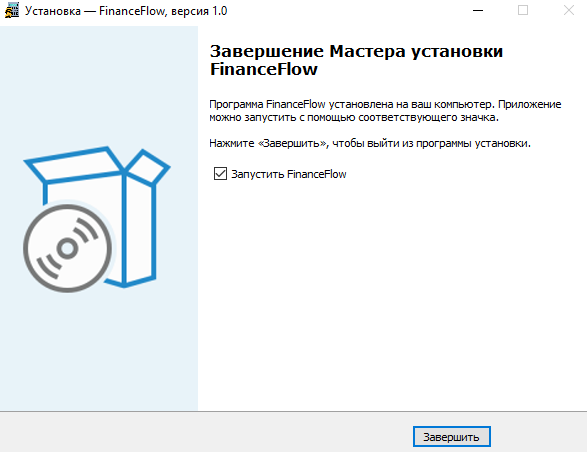


Рисунок 6 ­­– Завершение установки

После установки на «Рабочем столе» появится ярлык для запуска приложения, который представлен на рисунке 7.



Рисунок 7 – Ярлык приложения

* 1. Выполнение программы

При запуске приложения пользователь попадает на Главное меню, которое представлено на рисунке 9.

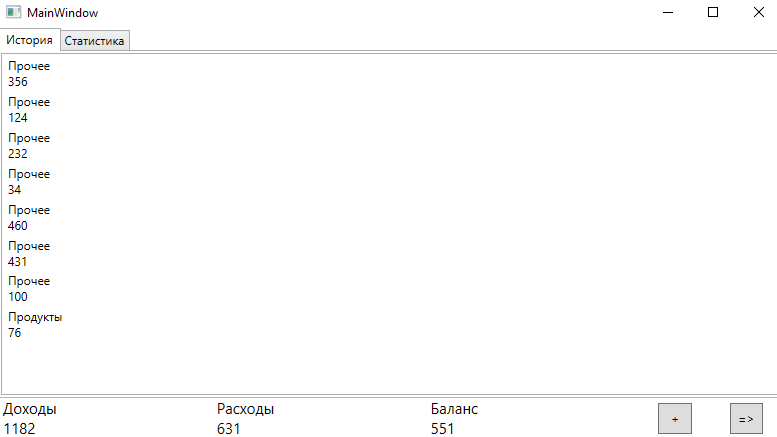


Рисунок 9 – Экран загрузки

У пользователя есть возможность кликнуть по кнопке «Статистика», «Добавить финансовую операцию», «История», «Отчет».

При нажатии на кнопку в виде плюсика будет открыта окно для добавления финансовой операции предоставлено на рисунке 10.

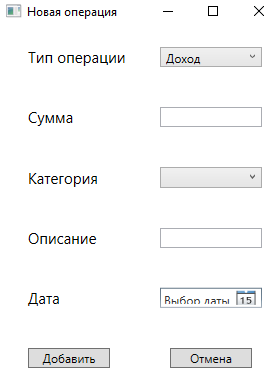


Рисунок 10 –Добавление финансовой операции

После внесения данных нажав на кнопку «Добавить», данные будут внесены в список на главную форму представлено на рисунке 11.

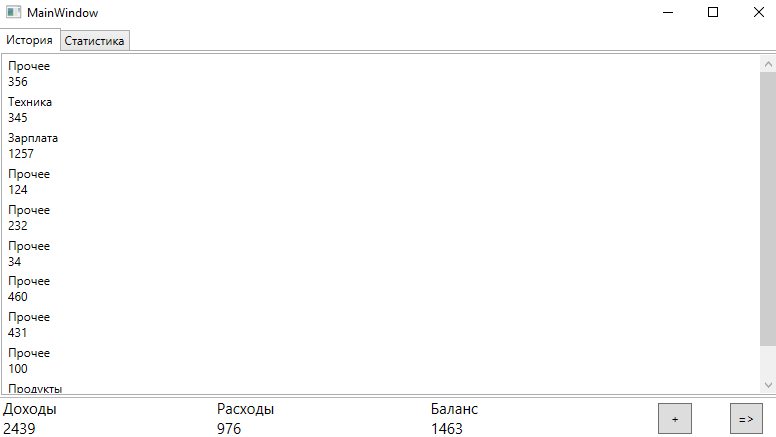


Рисунок 11 – Список данных

Нажав на кнопку «Статистика» будет открыто окно, на котором расположены диаграммы доходов и расходов, в процентах, исходя из данных, представленны на рисунке 12.

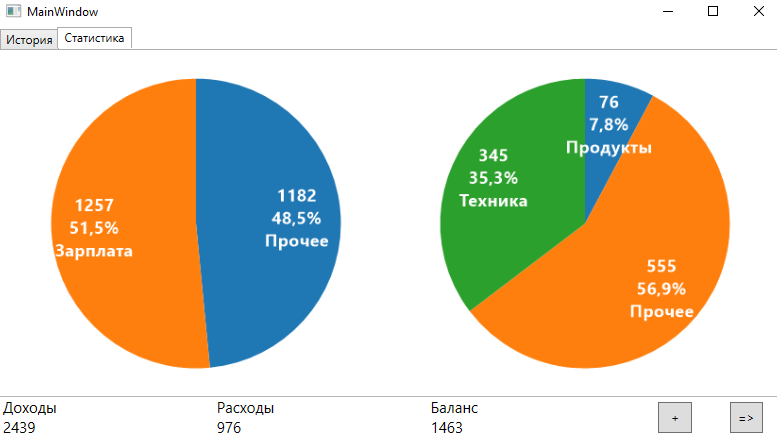


Рисунок 12 – Диаграммы доходов и расходов

Затем нажав на кнопку в виде знака больше либо равно, будет предоставлена возможность выбора расположения отчета в Excel отображена на рисунке 13.

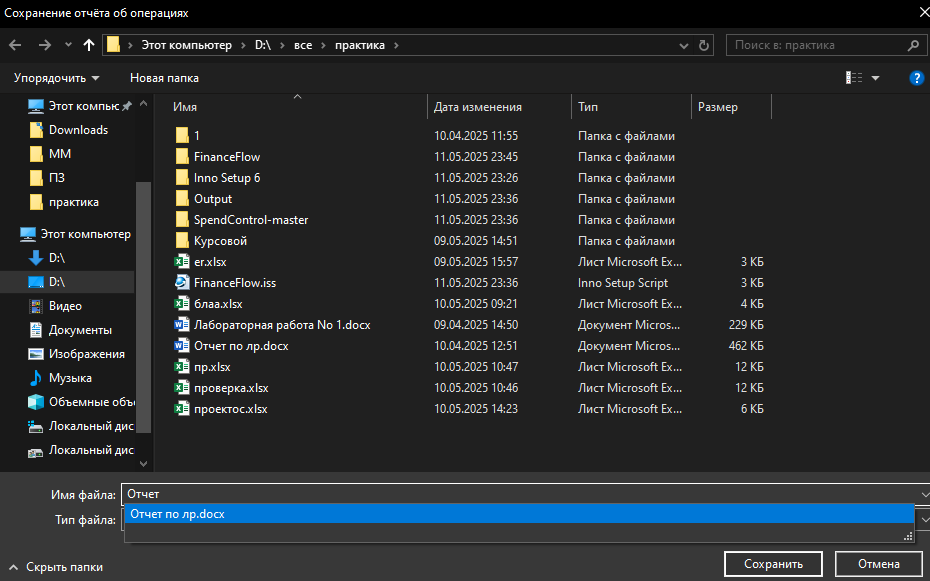


Рисунок 13 – Сохранение отчета

После сохранения появится окно с подтверждением, о сохранении отчета, показано на рисунке 14.

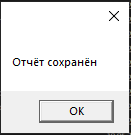


Рисунок 14 – Подтверждение о сохранении

Открыв отчет, мы можем увидеть все информацию о доходах/расходах, а также диаграммы, предоставлено на рисунке 15.

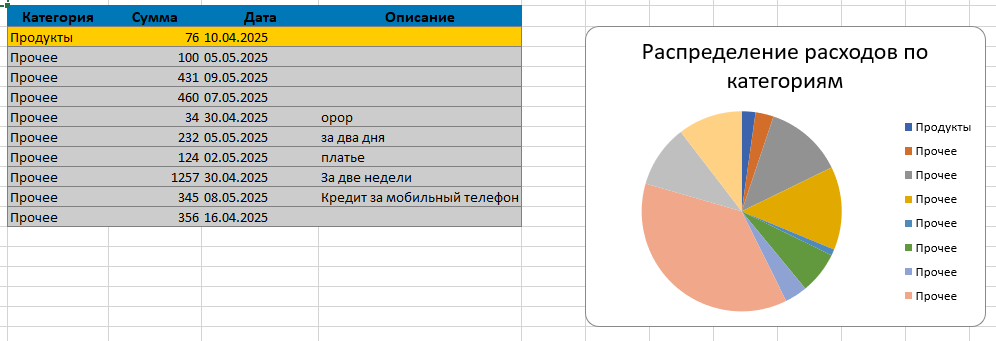


Рисунок 15 – Отчет

Заключение

Целью данной учебной практики была разработка десктопного приложения FinanceFlow, предназначенного для анализа, учета и планирования личных финансов. В ходе тестирования не было выявлено критических ошибок, исключительные ситуации обработаны корректно, а функциональность приложения полностью соответствует заявленным требованиям.

В процессе разработки были получены ценные знания по созданию пользовательского интерфейса, работе с динамическими финансовыми расчетами и оптимизации функциональности на платформе C#. Были закреплены навыки обработки данных, визуализации отчетов и реализации адаптивных интерфейсов.

Все основные компоненты приложения успешно созданы и протестированы. Реализован интуитивно понятный интерфейс, позволяющий пользователю легко добавлять финансовые операции, анализировать статистику и формировать отчеты в Excel.

Исходя из проведенных тестов, можно сделать вывод, что программа реализована успешно и готова к использованию.

Список использованных источников

1. Документация C# и .NET – официальные руководства по разработке десктопных приложений. - https://learn.microsoft.com/ru-ru/dotnet/
2. Материалы по финансовому анализу – статьи и учебные материалы по управлению личными финансами. - https://www.bankrate.com/
3. Документация по библиотекам и фреймворкам – использованные библиотеки для работы с графиками, таблицами и базами данных. - https://github.com/ScottPlot/ScottPlot
4. Руководства по созданию установщиков – информация по использованию Inno Setup, NSIS и других инструментов. - https://jrsoftware.org/isinfo.php
5. Материалы по тестированию ПО – принципы написания тест-кейсов и проведения регрессионного тестирования. - https://softwaretestingfundamentals.com/

Приложение A  
Листинг программы

Файл MainWindow.xaml.cs:

using System;

using System.Windows;

using System.Windows.Controls;

namespace SpendControl

{

/// <summary>

/// Логика взаимодействия с главным окном приложения

/// </summary>

public partial class MainWindow : Window

{

// События для обработки операций, создания отчёта и закрытия приложения

public event EventHandler newOperationEvent = null;

public event EventHandler makeExcelReportEvent = null;

public event EventHandler applicationCloseEvent = null;

public MainWindow()

{

InitializeComponent(); // Инициализация компонентов окна

// Инициализация графиков с пустыми значениями

double[] values = { 0 };

string[] labels = { "label" };

InitPlots(values, labels, values, labels);

// Создание презентора (логика взаимодействия)

new Prezentor(this);

}

// Метод для инициализации круговых диаграмм

public void InitPlots(double[] valuesGain, string[] labelsGain, double[] valuesSpend, string[] labelsSpend)

{

var plt = ChartGain.Plot;

plt.Clear();

var pie = plt.AddPie(valuesGain);

pie.SliceLabels = labelsGain;

pie.ShowPercentages = true;

pie.ShowValues = true;

pie.ShowLabels = true;

var plt2 = ChartSpend.Plot;

plt2.Clear();

var pie2 = plt2.AddPie(valuesSpend);

pie2.SliceLabels = labelsSpend;

pie2.ShowPercentages = true;

pie2.ShowValues = true;

pie2.ShowLabels = true;

try

{

// Обновление графиков после изменения данных

ChartGain.Refresh();

ChartSpend.Refresh();

}

catch (Exception)

{ }

}

// Запрет на выделение текста в текстовом поле

private void TextBox\_SelectionChanged(object sender, RoutedEventArgs e)

{

if (sender != null)

{

e.Handled = true;

if ((sender as TextBox).SelectionLength != 0)

(sender as TextBox).SelectionLength = 0;

}

}

// Обработчик кнопки добавления операции

private void AddOperationButton\_Click(object sender, RoutedEventArgs e)

{

newOperationEvent.Invoke(sender, e);

}

// Обработчик кнопки создания отчёта в Excel

private void MakeExcelReportButton\_Click(object sender, RoutedEventArgs e)

{

makeExcelReportEvent.Invoke(sender, e);

}

// Обработчик закрытия окна приложения

private void Window\_Closed(object sender, EventArgs e)

{

applicationCloseEvent.Invoke(sender, e);

}

// Двойной клик по элементу списка операций

private void OperationListBox\_MouseDoubleClick(object sender, System.Windows.Input.MouseButtonEventArgs e)

{

if (OperationListBox.SelectedItem != null)

{

// TODO: редактирование/удаление элемента

}

}

}

}

Файл OherationWindow.xaml.cs:

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

namespace SpendControl

{

/// <summary>

/// Логика взаимодействия для OperationWindow.xaml

/// </summary>

public partial class OperationWindow : Window

{

private Model \_model; // Модель данных

public OperationWindow(Model model)

{

InitializeComponent(); // Инициализация компонентов окна

\_model = model;

OperationTypeComboBox.SelectedIndex = 0; // Установка типа операции по умолчанию

CategoryComboBox.SelectedItem = 0; // Установка категории по умолчанию

}

// Изменение типа операции (доходы/расходы)

private void OperationTypeIndexChanged(object sender, RoutedEventArgs e)

{

List<string> source =

OperationTypeComboBox.SelectedIndex == 0 ? // Если индекс 0 - доходы

\_model.GainCategories : // Список категорий доходов

\_model.SpendCategories; // Список категорий расходов

var CategoryList = CategoryComboBox.Items;

CategoryList.Clear(); // Очистка списка категорий

foreach (var item in source)

{

CategoryList.Add(item); // Добавление категорий в список

}

CategoryList.Add("Новая категория"); // Добавление возможности создать новую категорию

}

// Обработчик выбора категории

private void CategoryComboBox\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (CategoryComboBox.SelectedIndex != CategoryComboBox.Items.Count - 1) return;

// Если выбрано "новая категория", можно добавить дополнительную логику

}

// Добавление новой операции

private void AddButtonClick(object sender, RoutedEventArgs e)

{

var description = DiscribtionTextBox.Text;

//--------------------Проверка типа операции---------------------------

if (OperationTypeComboBox.SelectedItem == null)

{

MessageBox.Show("Укажите тип операции!");

return;

}

int id = OperationTypeComboBox.SelectedIndex;

string type = id == 0 ? "Доход" : "Расход";

//---------------------------------------------------------------------

//--------------------Проверка суммы-----------------------------------

float price;

try

{

price = float.Parse(PriceTextBox.Text);

}

catch (FormatException)

{

MessageBox.Show("Некорректный формат суммы!");

return;

}

//---------------------------------------------------------------------

//--------------------Проверка категории-------------------------------

List<string> categories = new List<string>();

foreach (var item in CategoryComboBox.Items)

{

categories.Add(item.ToString());

}

if (CategoryComboBox.SelectedItem == null)

{

MessageBox.Show("Укажите категорию!");

return;

}

var category = CategoryComboBox.SelectedItem.ToString();

var LAST = CategoryComboBox.Items.Count - 1;

if (!categories.Contains(category) || category == CategoryComboBox.Items[LAST].ToString())

{

MessageBox.Show("Неизвестная категория!");

return;

}

//---------------------------------------------------------------------

//--------------------Проверка даты------------------------------------

var date = datePicker.SelectedDate;

if (date == null)

{

MessageBox.Show("Укажите дату!");

return;

}

else if (date.Value.Date > DateTime.Now.Date)

{

MessageBox.Show("Дата не может быть больше текущей!\n" +

"(нельзя создавать будущие операции)");

return;

}

//---------------------------------------------------------------------

\_model.Buff = new Operation(price, description, category, type, (DateTime)date);

this.Close(); // Закрытие окна

}

// Отмена операции и закрытие окна

private void CancelButton\_Click(object sender, RoutedEventArgs e)

{

this.Close();

}

// Метод проверки корректности операции

private bool CorrectOperation(string Ftype, float Fprice, string Fcategory, DateTime? Fdate)

{

return true; // Здесь может быть дополнительная логика

}

// Получение категории из строки

private string GetCategory(string str)

{

var ans = "";

int i;

for (i = str.Length - 1; i >= 0; i--)

{

if (str[i] == ' ') break;

}

ans = str.Substring(i + 1, str.Length - 1 - i);

return str;

}

}

}

Файл ExcelReportGenerator.cs:

using System;

using System.Collections.Generic;

using System.IO;

using OfficeOpenXml;

using OfficeOpenXml.Drawing.Chart;

using OfficeOpenXml.Style;

using System.Linq;

using System.Drawing;

using Ookii.Dialogs.Wpf;

using System.Windows.Forms;

namespace SpendControl

{

static class ExcelReportGenerator

{

#region RowAndColumnConsts

private const int START\_ROW = 2; // Начальная строка данных

private const int START\_COLUMN = 2; // Начальный столбец данных

private const int END\_COLUMN = 5; // Конечный столбец данных

private const int CATEGORY\_COLUMN = 2; // Колонка категории

private const int VALUE\_COLUMN = 3; // Колонка суммы

private const int DATE\_COLUMN = 4; // Колонка даты

private const int DESCRIPTION\_COLUMN = 5; // Колонка описания

#endregion

#region ColorConsts

private const string HEADER\_COLOR = "#0077b6"; // Цвет заголовка таблицы

private static readonly Dictionary<string, string> CategoryColors = new Dictionary<string, string>

{

{ "Продукты", "#ffcc00" }, // Желтый цвет для продуктов

{ "Транспорт", "#0099ff" }, // Голубой для транспорта

{ "Развлечения", "#ff6699" }, // Розовый для развлечений

{ "Коммунальные услуги", "#66cc66" }, // Зеленый для коммунальных услуг

{ "Здоровье", "#ff3300" }, // Красный для здоровья

{ "Прочее", "#cccccc" } // Серый по умолчанию

};

#endregion

public static void MakeReport(List<Operation> data)

{

var package = new ExcelPackage();

var sheet = package.Workbook.Worksheets.Add("История операций");

// Создаем заголовок таблицы

var header = sheet.Cells[START\_ROW, START\_COLUMN, START\_ROW, END\_COLUMN];

header.LoadFromArrays(new object[][] { new[] { "Категория", "Сумма", "Дата", "Описание" } });

header.Style.Fill.PatternType = ExcelFillStyle.Solid;

header.Style.Fill.BackgroundColor.SetColor(ColorTranslator.FromHtml(HEADER\_COLOR));

header.Style.Font.Bold = true;

header.Style.Font.Size = 12;

header.Style.HorizontalAlignment = ExcelHorizontalAlignment.Center;

int row = 3;

foreach (var op in data)

{

// Заполняем строку данными

sheet.Cells[row, CATEGORY\_COLUMN].Value = op.Category;

sheet.Cells[row, VALUE\_COLUMN].Value = op.Value;

sheet.Cells[row, DATE\_COLUMN].Value = op.OperationDate.ToString("dd.MM.yyyy");

sheet.Cells[row, DESCRIPTION\_COLUMN].Value = op.Description;

var dataRow = sheet.Cells[row, START\_COLUMN, row, END\_COLUMN];

// Задаем цвет строки в зависимости от категории

string rowColor = CategoryColors[op.Category];

dataRow.Style.Fill.PatternType = ExcelFillStyle.Solid;

dataRow.Style.Fill.BackgroundColor.SetColor(ColorTranslator.FromHtml(rowColor));

// Добавляем границы ячеек

dataRow.Style.Border.BorderAround(ExcelBorderStyle.Thin, Color.Gray);

row++;

}

row--;

// Форматируем колонки

sheet.Cells[START\_ROW, DATE\_COLUMN, row, DATE\_COLUMN].Style.Numberformat.Format = "dd.MM.yyyy"; // Даты

sheet.Cells[START\_ROW, VALUE\_COLUMN, row, VALUE\_COLUMN].Style.Numberformat.Format = "#,##0.00 BYN"; // Валюта

// Создаем диаграмму расходов, если есть данные

var chartData = data.Where(op => !string.IsNullOrEmpty(op.Category)).ToList();

if (chartData.Any())

{

var chart = sheet.Drawings.AddChart("ExpenseChart", eChartType.Pie);

chart.SetPosition(START\_ROW, 0, END\_COLUMN + 1, 0);

chart.SetSize(400, 300);

chart.Series.Add(sheet.Cells[START\_ROW + 1, VALUE\_COLUMN, row, VALUE\_COLUMN],

sheet.Cells[START\_ROW + 1, CATEGORY\_COLUMN, row, CATEGORY\_COLUMN]);

chart.Title.Text = "Распределение расходов по категориям";

}

MakeFile(package.GetAsByteArray());

}

private static void MakeFile(byte[] data)

{

// Окно сохранения файла

SaveFileDialog dialog = new SaveFileDialog();

dialog.Title = "Сохранение отчёта об операциях";

dialog.DefaultExt = ".xlsx";

dialog.CheckPathExists = true;

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

{

File.WriteAllBytes(dialog.FileName, data);

MessageBox.Show("Отчёт сохранён");

}

}

}

}

Файл Prezentor.cs:

using System;

using System.IO;

using System.Xml.Serialization;

using System.Collections.Generic;

namespace SpendControl

{

class Prezentor

{

private MainWindow \_window; // Главное окно приложения

private Model \_model; // Модель данных

private ViewController \_view; // Контроллер отображения

public Prezentor(MainWindow window)

{

LoadData(); // Загрузка данных

\_window = window;

// Подписка на события главного окна

\_window.newOperationEvent += new EventHandler(window\_newOperationEvent);

\_window.makeExcelReportEvent += new EventHandler(window\_makeExcelReportEvent);

\_window.applicationCloseEvent += new EventHandler(window\_applicationCloseEvent);

\_view = new ViewController(\_model, \_window); // Создание контроллера отображения

UpdateView(); // Обновление представления

}

// Обработчик события добавления новой операции

private void window\_newOperationEvent(object sender, EventArgs e)

{

OperationWindow operationWindow = new OperationWindow(\_model);

operationWindow.Owner = \_window;

operationWindow.ShowDialog();

if (\_model.Buff == null) return; // Проверка буфера

\_model.AddOperation(\_model.Buff); // Добавление операции

\_model.Buff = null; // Очистка буфера

UpdateView(); // Обновление представления

}

// Обработчик события создания отчёта в Excel

private void window\_makeExcelReportEvent(object sender, EventArgs e)

{

ExcelReportGenerator.MakeReport(\_model.Operations);

}

// Обработчик события закрытия приложения

private void window\_applicationCloseEvent(object sender, EventArgs e)

{

SaveData(); // Сохранение данных

}

private void SaveData()

{

SaveSystem.SaveData(\_model); // Сохранение модели

}

private void LoadData()

{

\_model = SaveSystem.LoadData(); // Загрузка модели из сохранённых данных

}

private void UpdateView()

{

\_view.UpdateView(); // Обновление представления

}

}

}

Файл SaveSystem.cs:

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Xml.Serialization;

namespace SpendControl

{

static class SaveSystem

{

// Тип для сериализации и десериализации объектов класса Model

private static XmlSerializer serializer = new XmlSerializer(typeof(Model));

// Метод для сохранения данных в XML-файл

public static void SaveData(Model model)

{

FileStream stream = new FileStream("Serialization.xml", FileMode.Create, FileAccess.Write, FileShare.Read);

// Сериализация объекта model и запись его в XML-файл

serializer.Serialize(stream, model);

stream.Close();

}

// Метод для загрузки данных из XML-файла

public static Model LoadData()

{

Model model = new Model();

try

{

FileStream stream = new FileStream("Serialization.xml", FileMode.Open, FileAccess.Read, FileShare.Read);

// Десериализация объекта model из XML-файла

model = serializer.Deserialize(stream) as Model;

}

catch (Exception)

{

// Если файл отсутствует или произошла ошибка, создаётся новый объект Model

model = new Model();

model.AddStartCategories(); // Добавление стандартных категорий

}

return model;

}

}

}

Файл ViewController.cs:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace SpendControl

{

class ViewController

{

private Model \_model; // Модель данных

private MainWindow \_window; // Главное окно приложения

public ViewController(Model model, MainWindow window)

{

\_model = model;

\_window = window;

}

// Обновление представления (история, баланс, диаграммы)

public void UpdateView()

{

UpdateHistory();

UpdateBalance();

UpdateCharts();

}

// Обновление списка операций

private void UpdateHistory()

{

var history = \_window.OperationListBox.Items;

history.Clear(); // Очистка списка

foreach (var item in \_model.Operations)

{

history.Insert(0, item.Category + "\n" + item.Value); // Добавление новой операции

}

}

// Обновление баланса доходов и расходов

private void UpdateBalance()

{

float sumGain = 0;

float sumSpend = 0;

foreach (var item in \_model.Operations)

{

if (item.Type == "Доход")

sumGain += item.Value;

else

sumSpend += item.Value;

}

// Отображение данных в текстовых полях

\_window.GainTextBox.Text = "Доходы\n" + sumGain.ToString();

\_window.SpendTextBox.Text = "Расходы\n" + sumSpend.ToString();

\_window.BalanceTextBox.Text = "Баланс\n" + (sumGain - sumSpend).ToString();

}

// Обновление диаграмм распределения доходов и расходов

private void UpdateCharts()

{

Dictionary<string, float> percentageGain = new Dictionary<string, float>(); // Распределение доходов по категориям

Dictionary<string, float> percentageSpend = new Dictionary<string, float>(); // Распределение расходов по категориям

foreach (var item in \_model.Operations)

{

if (item.Type == "Доход")

{

if (!percentageGain.ContainsKey(item.Category))

percentageGain[item.Category] = item.Value;

else

percentageGain[item.Category] += item.Value;

}

else

{

if (!percentageSpend.ContainsKey(item.Category))

percentageSpend[item.Category] = item.Value;

else

percentageSpend[item.Category] += item.Value;

}

}

// Создание массивов для диаграмм

string[] gainKeys = new string[percentageGain.Count];

string[] spendKeys = new string[percentageSpend.Count];

double[] gainValues = new double[percentageGain.Count];

double[] spendValues = new double[percentageSpend.Count];

int id = 0;

foreach (var item in percentageGain)

{

gainKeys[id] = item.Key;

gainValues[id] = item.Value;

id++;

}

id = 0;

foreach (var item in percentageSpend)

{

spendKeys[id] = item.Key;

spendValues[id] = item.Value;

id++;

}

\_window.InitPlots(gainValues, gainKeys, spendValues, spendKeys); // Обновление диаграмм

}

}

}

Файл Model.cs:

using System.Collections.Generic;

namespace SpendControl

{

public class Model

{

private const string PATH = "data.db"; // Путь к базе данных

private List<Operation> \_operations; // Список операций

private List<string> \_gainCategories; // Список категорий доходов

private List<string> \_spendCategories; // Список категорий расходов

public Operation Buff { get; set; } // Буфер для новых операций

public Model()

{

\_operations = new List<Operation>();

\_gainCategories = new List<string>();

\_spendCategories = new List<string>();

Buff = null; // Инициализация буфера

}

// Свойства для работы с операциями и категориями

public List<Operation> Operations

{ get => \_operations; set => \_operations = value; }

public List<string> GainCategories

{ get => \_gainCategories; set => \_gainCategories = value; }

public List<string> SpendCategories

{ get => \_spendCategories; set => \_spendCategories = value; }

// Добавление новой операции с сортировкой по дате

public void AddOperation(Operation operation)

{

\_operations.Add(operation);

int SIZE = \_operations.Count - 1;

int id;

for (id = SIZE - 1; id >= 0; id--)

{

if (\_operations[SIZE].OperationDate >= \_operations[id].OperationDate)

break;

}

if (id < 0) id = 0;

var buff = \_operations[id];

\_operations[id] = \_operations[SIZE];

\_operations[SIZE] = buff;

}

// Заполнение списка категорий доходов и расходов

public void AddStartCategories()

{

// Категории доходов

\_gainCategories.Add("Зарплата");

\_gainCategories.Add("Инвестиции");

// Категории расходов

\_spendCategories.Add("Продукты");

\_spendCategories.Add("Жильё");

\_spendCategories.Add("Связь");

\_spendCategories.Add("Одежда");

\_spendCategories.Add("Транспорт");

\_spendCategories.Add("Хозяйственные товары");

\_spendCategories.Add("Развлечения");

\_spendCategories.Add("Техника");

\_spendCategories.Add("Товары для дома");

\_spendCategories.Add("Медицина");

}

}

}

Файл Operation.cs:

using System;

namespace SpendControl

{

public class Operation

{

// Конструктор по умолчанию

public Operation()

{

Type = "spend"; // Тип операции (расход)

Value = 0; // Значение операции (по умолчанию 0)

Category = "shoping"; // Категория операции (по умолчанию "shoping")

Description = ""; // Описание операции

OperationDate = DateTime.Now; // Дата операции (текущий момент)

}

// Конструктор с параметрами

public Operation(float value, string discription, string category, string type, DateTime operationDate)

{

Type = type; // Заданный тип операции

Value = value; // Заданная сумма операции

Category = category; // Заданная категория операции

Description = discription; // Заданное описание операции

OperationDate = operationDate; // Заданная дата операции

}

// Свойства для хранения данных об операции

public string Type { get; set; } // Тип операции ("spend" или "gain")

public float Value { get; set; } // Сумма операции

public string Category { get; set; } // Категория операции

public string Description { get; set; } // Описание операции

public DateTime OperationDate { get; set; } // Дата операции

}

}