МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Информационные системы и технологии

Специальность 1-98 01 03 «Программное обеспечение информационной

безопасности мобильных систем»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

Разработка программного средства «Домашняя бухгалтерия»

Выполнил студент Сакович Александр Сергеевич

(Ф.И.О.)

Руководитель проекта асс. Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Заведующий кафедрой д.к.т.н., Смелов В.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Консультанты асс. Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

ст. преп. Блинова Е.А.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Нормоконтроллер асс. Гладкий М.В.

(учен. степень, звание, должность, подпись, Ф.И.О.)

Курсовой проект защищен с оценкой

Минск 2017

**Реферат**

Пояснительная записка курсового проекта содержит 33 страницы, 12 рисунков, 3 источника литературы, 1 приложение.

БУХГАЛТЕРИЯ, БАЗА ДАННЫХ, ООП, C#, WINDOWS FORMS.

Основной целью курсового проекта является: разработка программного средства «Домашняя бухгалтерия».

Пояснительная записка состоит из введения, шести разделов, заключения.

Во введении представлена общая информация, дающая представление о предстоящей работе, определены цели.

В первом разделе рассматривается информация о используемых технологиях и утилит.

Во втором разделе рассматривается постановка задачи и обзор прототипов.

В третьем разделе рассматривается архитектура приложения.

В четвертом разделе рассматривается архитектура базы данных.

В пятом разделе представлено руководство пользователя.

В шестом разделе представлено ручное тестирование приложения.

В заключении представлены итоги курсового проектирования и задачи, которые были решены в ходе разработки программного средства.

Оглавление

[**Введение** 6](#_Toc483168196)

[**1. Теоретическое введение** 7](#_Toc483168197)

[**2. Постановка задачи** 12](#_Toc483168198)

[**3. Архитектура приложения** 14](#_Toc483168199)

[**4. Проектирование и создание базы данных** 16](#_Toc483168200)

[**5. Руководство пользователя** 18](#_Toc483168201)

[**6. Тестирование программы** 21](#_Toc483168202)

[**Заключение** 22](#_Toc483168203)

[**Список использованных источников** 23](#_Toc483168204)

[**Приложение** 24](#_Toc483168205)

# **Введение**

В нашем стремительном веке движущей силой является внедрение информационных технологий, которое не обошло стороной и бухгалтерию, что неудивительно, ведь бухгалтерский учет ­– один из наиболее трудоемких процессов, который охватить, что называется «вручную», - сложно, а порой и невозможно. Наличие же трудоемкого процесса заявило о необходимости превращения этого процесса в автоматизированный. Но в то же время возникла проблема выбора. Если на заре автоматизации бухгалтерского учета выбор программного обеспечения был не так велик, то с развитием научно-технического процесса программных продуктов становится все больше. Это разнообразие, с одной стороны, упрощает, а с другой усложняет проблему выбора.

Одними из самых популярных программ по ведению бухгалтерии являются:

* 1С: Бухгалтерия;
* Галактика;
* БЭСТ;
* Инфо-Бухгалтер.

Но не смотря на свою популярность данные программы не востребованы среди людей, которые только начинают планировать свой бюджет, так как являются слишком сложными для понимания обычного пользователя.

Целью моего курсового проекта является создание программного средства «Домашняя бухгалтерия». Особенностями данного приложения являются:

* интуитивно-понятный интерфейс;
* возможность хранения изображения;
* просмотр данных за конкретный промежуток времени.

Человек с легкостью сможет разобраться в работе приложения. Именно поэтому программное средство «Домашняя бухгалтерия» будет полезно для пользователей и значительно сэкономит их время.

## **Теоретическое введение**

**1.1 Объектно-ориентированное программирование**

*Объектно-ориентированное программирование (ООП)* - это метод программирования, при использовании которого главными элементами программ являются объекты. В языках программирования понятие объекта реализовано как совокупность свойств (структур данных, характерных для данного объекта), методов их обработки (подпрограмм изменения их свойств) и событий, на которые данный объект может реагировать и, которые приводят, как правило, к изменению свойств объекта. Объединение данных и свойственных им процедур обработки в одном объекте, называется инкапсуляцией и является одним из важнейших принципов ООП.

Другим фундаментальным понятием является *класс*.

*Класс* - это шаблон, на основе которого может быть создан конкретный программный объект, он описывает свойства и методы, определяющие поведение объектов этого класса. Каждый конкретный объект, имеющий структуру этого класса, называется экземпляром класса.

Следующими важнейшими принципами ООП являются *наследование* и *полиморфизм*.

*Наследование* - такая организация классов, которая предусматривает создание новых классов на базе существующих и позволяет классу потомку иметь (наследовать) все свойства класса – родителя.

*Полиморфизм* - означает, что рожденные объекты обладают информацией о том, какие методы они должны использовать в зависимости от того, в каком месте цепочки (дерева классов) они находятся, иными словами это концепция, реализующая "множество методов в одном интерфейсе".

Другим важнейшим принципом ООП является модульность.

*Модульность* - это такая организация объектов, когда они заключают в себе полное определение их характеристик, никакие определения методов и свойств не должны располагаться вне его, это делает возможным свободное копирование и внедрение одного объекта в другие.

Развитием объектно-ориентированной парадигмы (методологии), стала объектно-событийная парадигма, опирающаяся на понятия объекта и события. Эта парадигма позволяет конструировать, программировать распределенные вычислительные среды, в том числе среды реального времени, SCADA и пр.

Событие в объектно-ориентированном программировании - это сообщение, которое возникает в различных точках исполняемого кода при выполнении определённых условий. События предназначены для того, чтобы иметь возможность предусмотреть реакцию программного обеспечения. Для решения поставленной задачи создаются обработчики событий: как только программа попадает в заданное состояние, происходит событие, посылается сообщение, а обработчик перехватывает это сообщение. В общем случае в обработчик не передаётся ничего, либо передаётся ссылка на объект, инициировавший (породивший) обрабатываемое событие. В особых случаях в обработчик передаются значения некоторых переменных или ссылки на какие-то другие объекты, чтобы обработка данного события могла учесть контекст возникновения события. Самое простое событие - это событие, сообщающее о начале или о завершении некоторой процедуры. Событие, по сути, сообщает об изменении состояния некоторого объекта. Наиболее наглядно события представлены в пользовательском интерфейсе, когда каждое действие пользователя порождает цепочку событий, которые, затем обрабатываются в приложении. В объектно-ориентированном анализе для описания динамического поведения объектов принято использовать модель состояний. Событие - это переход объекта из одного состояния в другое. Взаимодействие объектов также осуществляется при помощи событий: изменение состояния одного объекта приводит к изменению состояния другого объекта, а событие оказывается средством связи между объектами. Событие - это абстракция инцидента или сигнала в реальном мире, который сообщает нам о перемещении чего- либо в новое состояние. Далее, выделяются четыре аспекта события:

* метка – уникальный идентификатор события;
* значение – текстовое сообщение о сути произошедшего;
* предназначение – модель событий, которая принимает событие;
* данные – данные, которые переносятся от одного объекта к другому.

Первый ряд примеров событий доставляет собственно сам жизненный цикл объекта:

* создание объекта;
* уничтожение объекта.

Более сложные примеры событий возникают тогда, когда у объекта появляются внутренние состояния, которые описываются соответствующей диаграммой переходов (из одного состояния в другое).

Современными языками объектно-ориентированного программирования являются С++ и Java. С середины 90-х годов многие объектно–ориентированные языки реализуются как системы визуального программирования, в которых интерфейсная часть программного продукта создается в диалоговом режиме, практически без написания программных операторов. К объектно – ориентированным системам визуального проектирования относятся Visual Basic, Delphi, C++ Builder, Visual C++. Язык VBA (Visual Basic for Applications) – язык приложений Microsoft Office (Excel, Word, Access, Power Point и др). VBA соблюдает основной синтаксис языка и правила программирования языков Basic – диалектов, позволяет создавать макросы для автоматизации выполнения некоторых операций и графический интерфейс пользователя, интеграцию между различными программными продуктами.

**1.2 Язык программирования *C#***

На сегодняшний момент язык программирования C# один из самых мощных, быстро развивающихся и востребованных языков в ИТ-отрасли. В настоящий момент на нем пишутся самые различные приложения: от небольших десктопных программок до крупных веб-порталов и веб-сервисов, обслуживающих ежедневно миллионы пользователей.

По сравнению с другими языками C# достаточно молодой, но в то же время он уже прошел большой путь. Первая версия языка вышла вместе с релизом Microsoft Visual Studio .NET в феврале 2002 года. Текущей версией языка является версия C# 7.0, которая вышла в 7 марта 2017 года вместе с Visual Studio 2017.

C# является языком с Си-подобным синтаксисом и близок в этом отношении к C++ и Java. Поэтому, если вы знакомы с одним из этих языков, то овладеть C# будет легче.

C# является объектно-ориентированным и в этом плане много перенял у Java и С++. Например, C# поддерживает полиморфизм, наследование, перегрузку операторов, статическую типизацию. Объектно-ориентированный подход позволяет решить задачи по построению крупных, но в тоже время гибких, масштабируемых и расширяемых приложений. И C# продолжает активно развиваться, и с каждой новой версией появляется все больше интересных функциональностей, как, например, лямбды, динамическое связывание, асинхронные методы и т.д.

**1.3 Windows Form**

*Windows Forms* — интерфейс программирования приложений (API), отвечающий за графический интерфейс пользователя и являющийся частью Microsoft .NET Framework. Данный интерфейс упрощает доступ к элементам интерфейса Microsoft Windows за счет создания обёртки для существующего Win32 API в управляемом коде. Причём управляемый код — классы, реализующие API для Windows Forms, не зависят от языка разработки. То есть программист одинаково может использовать Windows Forms как при написании ПО на C#, С++, так и на VB.Net, J# и др.

Windows Forms уже оказал влияние на разработки для Windows. Теперь, когда приложение находится на начальной стадии проектирования, принять решение о том, нужно ли строить Web-ориентированное приложение либо же полностью клиентское, стало немного труднее. Клиентские приложения Windows могут быть разработаны быстро и эффективно, при этом они предлагают пользователям гораздо более широкие возможности.

Windows Forms покажется вам знакомым, если у вас есть опыт разработки на Visual Basic. Вы создаете новые формы (также известные как окна или диалоги) в той же манере - перетаскивая и размещая элементы управления из панели инструментов на поверхность визуального дизайнера форм (Form Designer).

**1.4 Microsoft SQL Server**

*SQL Server Management Studio* — утилита из Microsoft SQL Server 2012 и более поздних версий для конфигурирования, управления и администрирования всех компонентов Microsoft SQL Server. Утилита включает скриптовый редактор и графическую программу, которая работает с объектами и настройками сервера.

Главным инструментом SQL Server Management Studio является Object Explorer, который позволяет пользователю просматривать, извлекать объекты сервера, а также полностью ими управлять.

SQL является прежде всего информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. SQL можно назвать языком программирования, при этом он не является тьюринг-полным, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.

Изначально SQL был основным способом работы пользователя с базой данных и позволял выполнять следующий набор операций:

• создание в базе данных новой таблицы;

• добавление в таблицу новых записей;

• изменение записей;

• удаление записей;

• выборка записей из одной или нескольких таблиц (в соответствии с заданным условием);

• изменение структур таблиц.

При всех своих изменениях SQL остаётся единственным механизмом связи между прикладным программным обеспечением и базой данных. В то же время современные СУБД, а также информационные системы, использующие СУБД, предоставляют пользователю развитые средства визуального построения запросов.

Таким образом использование SQL сводится, по сути, к формированию всевозможных выборок строк и совершению операций над всеми записями, входящими в набор.

**1.5 Архитектура платформы .NET Framework**

Программа на языке C# выполняется в среде .*NET Framework* – интегрированном компоненте Windows, содержащем виртуальную систему выполнения (среда CLR) и унифицированный набор библиотек классов. Среда CLR представляет собой коммерческую реализацию Майкрософт инфраструктуры CLI (common language infrastructure), международного стандарта, основы сред выполнения и разработки с тесным взаимодействием языков и библиотек.

Исходный код, написанный на языке C#, компилируется в промежуточный язык (IL) в соответствии со спецификацией CLI. Код IL и ресурсы, такие как растровые изображения и строки, хранятся на диске в исполняемом файле, называемом сборкой, с расширением EXE или DLL в большинстве случаев. Сборка содержит манифест со сведениями о типах сборки, версии, языке и региональных параметрах, и требованиях безопасности.

При выполнении программы на C# сборка загружается в среду CLR в зависимости от сведений в манифесте. Далее, если требования безопасности соблюдены, среда CLR выполняет JIT-компиляцию для преобразования кода IL в инструкции машинного кода. Среда CLR также предоставляет другие службы, относящиеся к автоматическому сбору мусора, обработке исключений и управлению ресурсами. Код, выполняемый средой CLR, иногда называют "управляемым кодом" в противопоставление "неуправляемому коду", который компилируется в машинный код, предназначенный для определенной системы. Далее показаны отношения во время компиляции и время выполнения между файлами с исходным кодом C#, библиотеками классов .NET Framework, сборками и средой CLR.

Взаимодействие между языками является ключевой особенностью .NET Framework. Поскольку код на промежуточном языке (IL), создаваемый компилятором C#, соответствует спецификации CTS, код IL на основе C# может взаимодействовать с кодом, создаваемым версиями языков Visual Basic, Visual C++, Visual J# платформы .NET Framework и еще более чем 20 CTS-совместимых языков. В одной сборке может быть несколько модулей, написанных на разных языках платформы .NET Framework, и типы могут ссылаться друг на друга, как если бы они были написаны на одном языке.

Помимо служб времени выполнения, в .NET Framework также имеется обширная библиотека, состоящая из более чем 4000 классов, организованных по пространствам имен, которые обеспечивают разнообразные полезные функции для любых действий, начиная от ввода и вывода файлов для управления строками для разбивки XML, и заканчивая элементами управления Windows Forms. В обычном приложении на языке C# библиотека классов .NET Framework интенсивно используется для "устройства" кода.

## **Постановка задачи**

Главной целью данного курсового проекта является разработка приложения «Домашняя Бухгалтерия».

После запуска программы пользователь может добавить данные о своих доходах и расходах, а также просмотреть их. При добавлении данных о расходах можно добавить фотографию чека, который будет хранится в БД и в любой момент пользователь может посмотреть его. Также имеется возможность просмотреть данные о доходах и расходах за определенный период.

Для достижения поставленной цели необходимо разработать:

* Визуальное представление приложения.
* Алгоритм работы приложения.
* Базу данных приложения.

При обзоре прототипов было найдено 2 прототипа. Они представлены ниже.

**2.1 Обзор прототипов**

**«Домашняя бухгалтерия: ведем Семейный Бюджет» –** Используйте Домашнюю бухгалтерию для учета личных финансов и ведения семейного бюджета на компьютере и мобильных устройствах Андроид и iPhone/iPad. Используйте синхронизацию и получайте доступ к записям на компьютере, смартфоне и планшете.

Кроме учета личных финансов и контроля семейного бюджета, Домашняя бухгалтерия поможет вести финансовый учет индивидуальным предпринимателям и небольшим компаниям.

Приложение «Домашняя бухгалтерия: ведем Семейный Бюджет» представлено на рисунке 2.1.

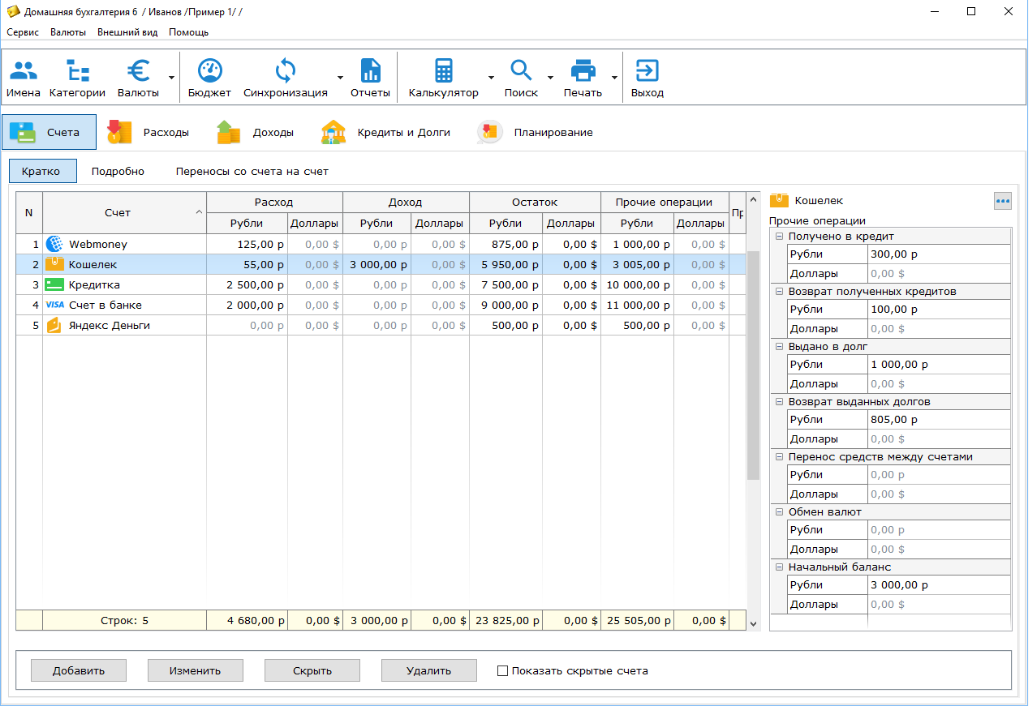


Рисунок 2.1. – Домашняя бухгалтерия: ведем Семейный Бюджет

*«AlzexFinance»* – Бесплатная домашняя бухгалтерия отлично подходит для начинающих и включает в себя все необходимые инструменты для учета финансов и планирования бюджета.

Приложение «AlzexFinance» представлено на рисунке 2.2.

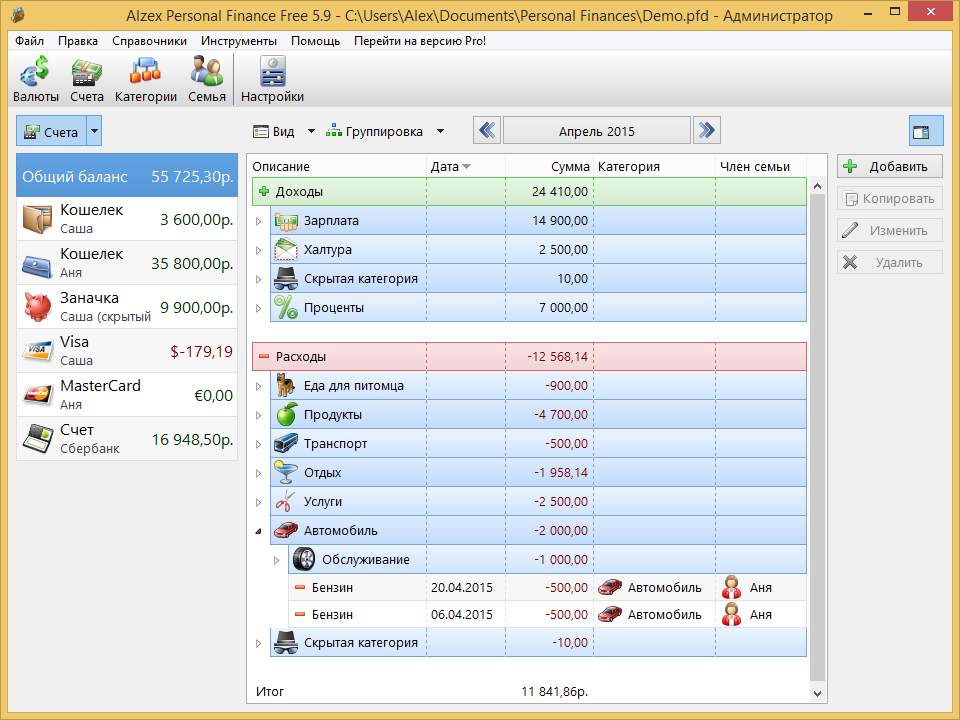


Рисунок 2.2. – AlzexFinance

## **Архитектура приложения**

Отношения между формами и классами представлены на Рисунке 3.2.

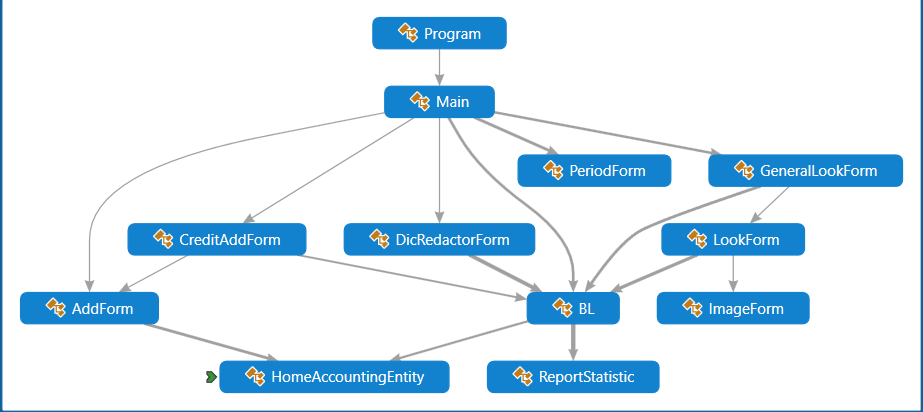


Рисунок 3.2 – Отношения между формами и классами

UML диаграмма классов представлена на Рисунке 3.3.

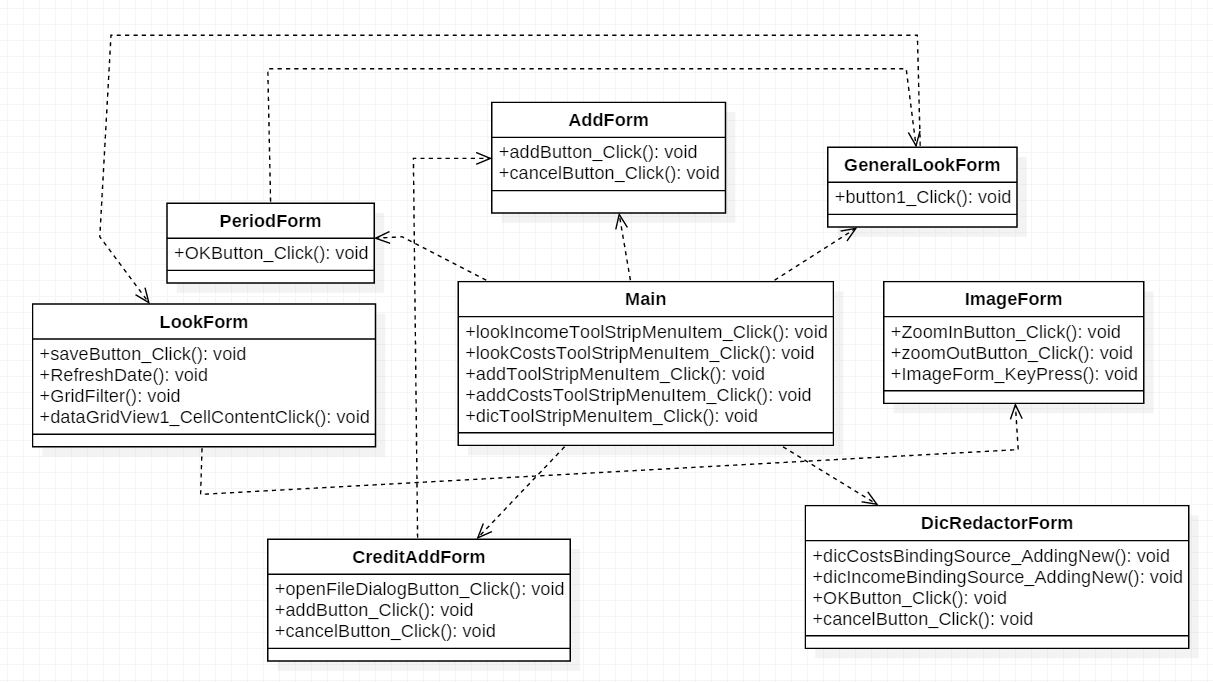


Рисунок 3.3 – UML диаграмма классов

Описание классов представлено в Таблице 3.

|  |  |
| --- | --- |
| Класс | Описание |
| Main | Стартовая форма, где пользователь может посмотреть финансовую информацию за какой-то период, просмотреть и добавить информацию по доходам и расходам. |
| AddForm | Форма, предназначенная для добавления информации о доходах. |
| CreditAddForm | Форма, предназначенная для добавления информации о расходах, с возможностью привязки фотографии чека. |
| DicRedactorForm | Форма, предназначенная для редактирования справочников. |
| GeneralLookForm | Форма, предназначенная для просмотра информации о доходах или расходах. |
| ImageForm | Форма, предназначенная для просмотра фотографии чека. |
| LookForm | Форма, предназначенная для детального просмотра информации о доходах или расходах. |
| PeriodForm | Форма, позволяющая выбрать период для просмотра доходов или расходов. |

Таблица 3 – Описание классов

## **Проектирование и создание базы данных**

Хранение всей информации осуществляется при помощи базы данных Microsoft SQL Server 2012. Выполним физическое проектирование базы данных. Зададим имена таблиц и полей, определим типы данных и размерность полей таблиц. Так же для поля определим его основные характеристики – является ли это поле внешним или первичным ключом, создан ли индекс по этому полю.

Всего было создано 5 таблиц, а именно «Income», «Costs», «dicIncome», «dicCosts» и «Check».

Пример создания одной из таблиц представлен на Рисунке 4.1.

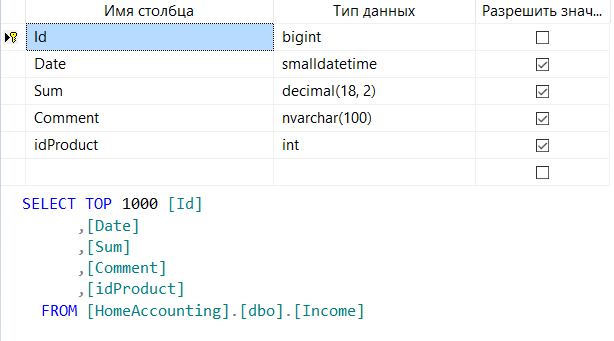


Рисунок 4.1 – Создание таблицы

Рисунок 4.2 иллюстрирует информацию одной из таблиц базы данных «HomeAccounting».

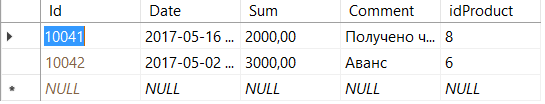


Рисунок 4.2 – Таблица в базе данных

Структура базы данных представлена на Рисунке 4.3.

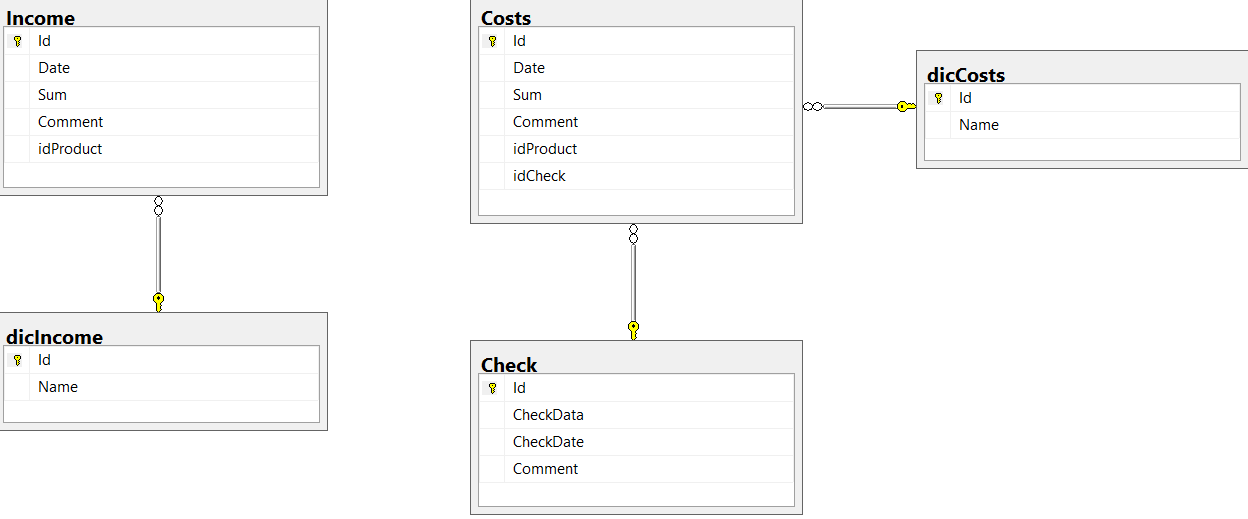


Рисунок 4.1 – Схема базы данных

Для примера рассмотрим одну из таблиц, а именно таблицу «Costs».

Описание полей таблицы «Costs» представлено в Таблице 4.

|  |  |
| --- | --- |
| Поле | Описание |
| Id | Первичный ключ. |
| Date | Дата |
| Sum | Сумма |
| Comment | Комментарий |
| idProduct | Id продукта |
| idCheck | Id чека, ссылается на таблицу «Check» |

Таблица 4 – Описание полей

## **Руководство пользователя**

При запуске программы «Домашняя бухгалтерия» появляется главная форма. На данной форме присутствуют:

* Кнопка «Доходы» для просмотра информации о доходах за определенный промежуток времени;
* Кнопка «Расходы» для просмотра информации о расходах за определённый промежуток времени;
* Кнопка «Справочники» для редактирования справочников о доходах и расходов;
* Блок «Отчетный период» для выбора месяца и года просматриваемой информации;
* Блок для просмотра информации за отчетный период;
* Кнопка «Просмотр» в блоке «Доходы по статьям» для добавления данных о доходах;
* Кнопка «Добавление» в блоке «Доходы по статьям» для просмотра данных о доходах;
* Кнопка «Просмотр» в блоке «Расходы по статьям» для просмотра данных о расходах;
* Кнопка «Добавление» в блоке «Расходы по статьям» для добавления данных о расходах.

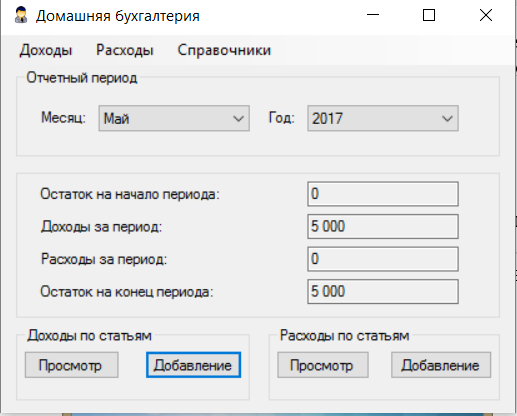


Рисунок 5.1 – Главная форма

Кнопка «Просмотр», в блоке «Доходы по статьям», открывает форму «Просмотр данных» (Рисунок 5.2), содержащую информацию о доходах. Также форма содержит кнопку «Детальный просмотр», в которой можно посмотреть подробную информацию о доходах.

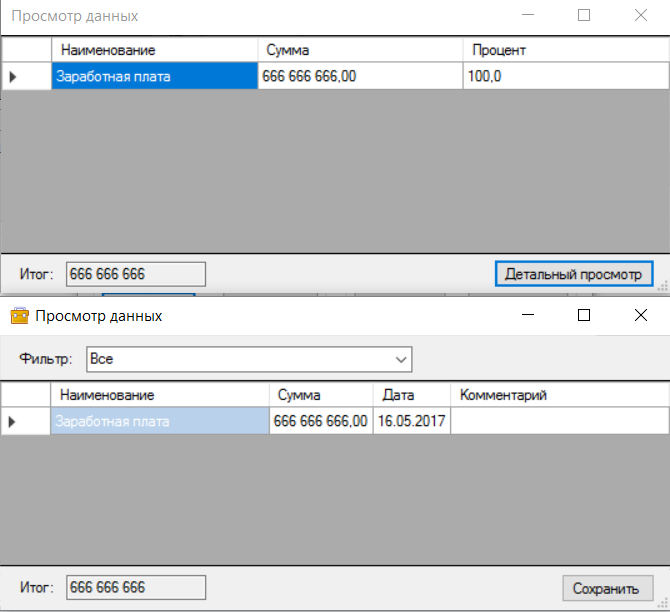


Рисунок 5.2 – Просмотр данных

Кнопка «Просмотр», в блоке «Расходы по статьям», открывает форму «Просмотр данных» (Рисунок 5.3), содержащую информацию о доходах. Также форма содержит кнопку «Детальный просмотр», в которой можно посмотреть подробную информацию о доходах, а также открыть фотографию чека.

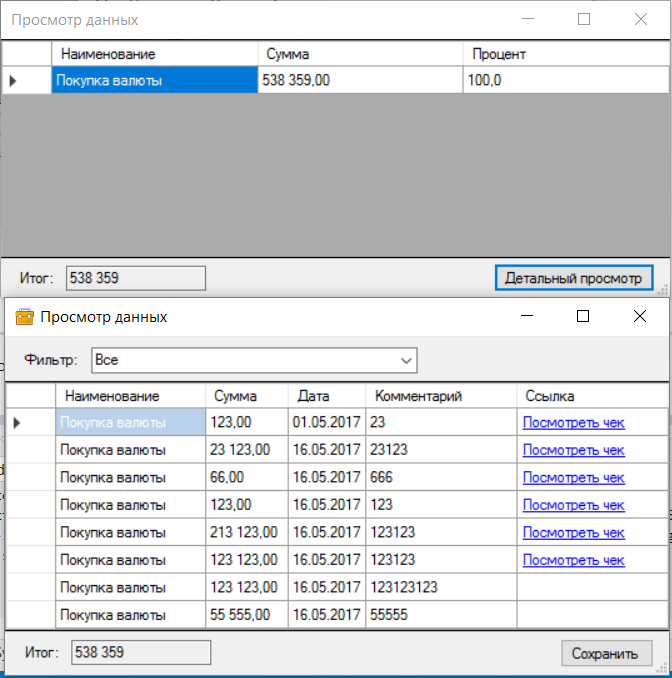


Рисунок 5.3 – Просмотр данных

Кнопка «Добавление», в блоке «Доходы по статьям», открывает форму «Добавление данных» (Рисунок 5.4), в которой можно заполнить поле «Сумма» и «Комментарий», а также выбрать дату и группу доходов.

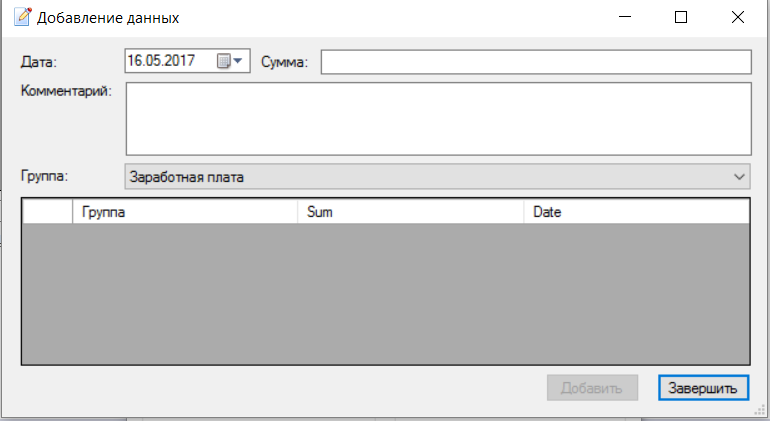


Рисунок 5.4 – Добавление данных

Кнопка «Добавление», в блоке «Расходы по статьям», открывает форму «Добавление данных» (Рисунок 5.5), в которой можно добавить информацию о расходах аналогично доходам, а также добавить фотографию чека.

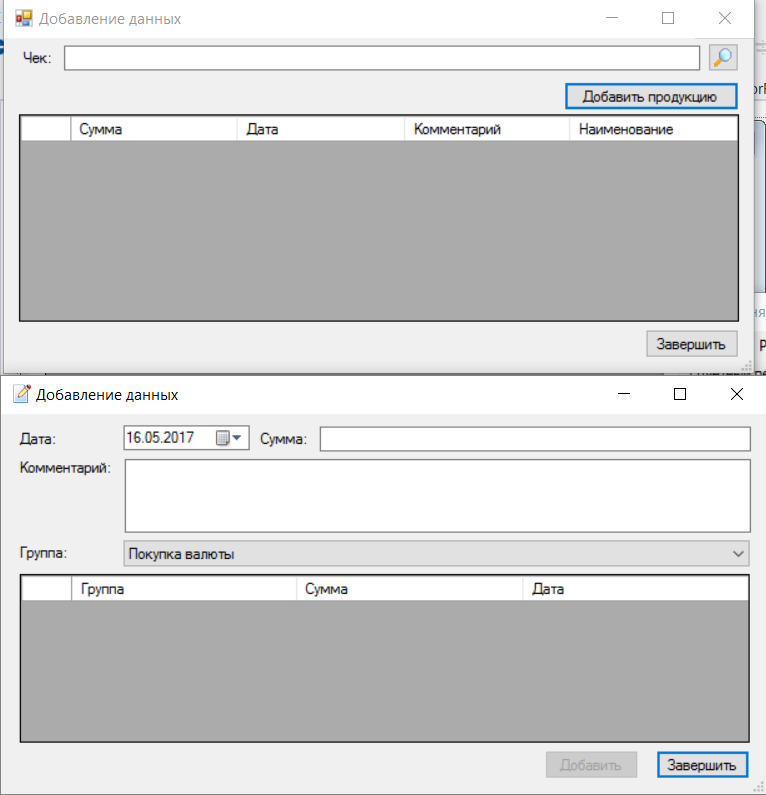


Рисунок 5.5 – Добавление данных

## **Тестирование программы**

Объектом испытания является программа «Домашняя бухгалтерия», находящаяся в папке «Курсовой проект» (листинг программы находится в Приложении).

Испытания проводятся с целью проверки правильности функционирования программы и выявления имеющих недостатков.

Приложение работает в среде Windows 10.

Программа разработана на языке C#.

Для проведения испытания необходимо запустить приложение и проверить работу всех вариантов использования системы. Разработанное приложение может использоваться на персональном компьютере с операционной системой Windows.

Испытания проводятся по алгоритму:

1. Запустить программу «Домашняя Бухгалтерия»;
2. В проявившейся на экране форме выбрать, что вы хотите сделать:
   1. Добавить данные о доходах/расходах;
   2. Просмотреть данные о доходах/расходах (с возможностью детального просмотра).
3. Просмотреть данные за определенный промежуток времени;
4. Добавить категории в справочники;
5. При ошибочном добавлении данных есть возможность удалить информацию.

Проведенные испытания показали, что программа работает верно и без каких-либо ошибок.

## **Заключение**

В данном курсовом проекте было разработано программное средство для управления салоном видеопроката при помощи C# Windows Form. При разработке данного программного средства использовалась база данных на языке T-SQL.

Решая поставленную задачу, пришли к удовлетворительному результату среди поиска существующих систем.

Программная система имеет простой, интуитивно понятный пользовательский интерфейс.

В соответствии с полученным результатом работы программы можно сделать вывод, что разработанная программа работает верно, а требования технического задания выполнены в полном объеме.

Выполненные цели:

* Сохранять рабочую информацию в централизованной базе данных.
* Позволять просматривать доход и расход за определенный промежуток времени.
* Рассчитывать остаток денежных средств на конец периода.
* Предоставлять удобный интерфейс пользователя.

## **Список использованных источников**

* 1. MSDN сеть разработчиков в Microsoft [Электронный ресурс] / Э– Режим доступа: <http://msdn.microsoft.com/library/rus/> – Дата доступа 14.04.2017.
  2. Уотсон К., Нейгел К. Visual C# 2010. Полный курс/ Уотсон К., Нейгел К. – СПб.: питер, 201 – 653 с.
  3. Троелсен Э. C# и платформа .NET. Библиотека программиста СПб., 2004. 796 c.

## **Приложение**

1. **Main.cs**

namespace HomeAccounting

{

public partial class Main : Form

{

#region initialize

public Main()

{

InitializeComponent();

AddItemYearComboBox();

yearComboBox.SelectedIndex = 0;

var index = DateTime.Now.Month - 1;

monthComboBox.SelectedIndex = index;

Select();

}

private void AddItemYearComboBox()

{

yearComboBox.Items.Add(DateTime.Now.Year);

yearComboBox.Items.Add(DateTime.Now.Year - 1);

yearComboBox.Items.Add(DateTime.Now.Year - 2);

}

#endregion

#region Button's click

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

{

var periodForm = new PeriodForm();

periodForm.ShowDialog();

if(periodForm.DialogResult == DialogResult.OK)

{

var lookForm = new GeneralLookForm("Income", periodForm.startPeriod, periodForm.endPeriod);

lookForm.ShowDialog();

UpdateFields();

}

}

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

{

var periodForm = new PeriodForm();

periodForm.ShowDialog();

if (periodForm.DialogResult == DialogResult.OK)

{

var lookForm = new GeneralLookForm("Costs", periodForm.startPeriod, periodForm.endPeriod);

lookForm.ShowDialog();

UpdateFields();

}

}

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

{

var addForm = new AddForm(dicEnum.dicIncome, CloseFormOptionEnum.WithSave);

addForm.ShowDialog();

UpdateFields();

}

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

{

using (var creditAddForm = new CreditAddForm())

{

creditAddForm.ShowDialog();

}

UpdateFields();

}

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

{

var dicRedactorForm = new DicRedactorForm(sender);

dicRedactorForm.ShowDialog();

}

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

{

var data = new DateTime(Int32.Parse(yearComboBox.Text), monthComboBox.SelectedIndex + 1, 1);

// LookForm lookForm = new LookForm("Income", data, data.AddMonths(1));

// lookForm.ShowDialog();

var generalLookForm = new GeneralLookForm("Income", data, data.AddMonths(1));

generalLookForm.ShowDialog();

UpdateFields();

}

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

{

var data = new DateTime(Int32.Parse(yearComboBox.Text), monthComboBox.SelectedIndex + 1, 1);

// LookForm lookForm = new LookForm("Costs", data, data.AddMonths(1));

// lookForm.ShowDialog();

var generalLookForm = new GeneralLookForm("Costs", data, data.AddMonths(1));

generalLookForm.ShowDialog();

UpdateFields();

}

#endregion

#region Events

private void ComboBox\_SelectedValueChanged(object sender, EventArgs e)

{

if (!String.IsNullOrEmpty(yearComboBox.Text) && !String.IsNullOrEmpty(monthComboBox.Text))

{

UpdateFields();

}

}

#endregion

#region Вспомогательные функции

private void UpdateFields()

{

try

{

var data = new DateTime(Int32.Parse(yearComboBox.Text), monthComboBox.SelectedIndex + 1, 1);

decimal ostatokStart = BL.ItogFromPeriod(new DateTime(1900, 1, 1), data, "Все");

decimal costs = BL.Itog(BL.FromDateCosts(data, data.AddMonths(1), "Все"));

decimal income = BL.Itog(BL.FromDateIncome(data, data.AddMonths(1), "Все"));

decimal ostatokEnd = ostatokStart + income - costs;

incomePeriodTextBox.Text = income.ToString("N0");

costsInPeriodTextBox.Text = costs.ToString("N0");

incomeStartPeriodTextBox.Text = ostatokStart.ToString("N0");

incomeEndPeriodTextBox.Text = ostatokEnd.ToString("N0");

}

catch (Exception exc)

{

MessageBox.Show(exc.Message);

}

}

#endregion

}

}

1. **AddForm.cs**

namespace HomeAccounting

{

/// <summary>

/// В форме реализовано 2 режима по нажатию на кнопку завершить(с сохранением в базу и без сохранения)

/// </summary>

public partial class AddForm : Form

{

#region initialize

private dicEnum \_dicName;

private bool \_withSave = false;

private List<Income> incomeList = new List<Income>();

private List<Costs> costList = new List<Costs>();

//Загружаем справочники

private List<dicCosts>\_dicCosts = new List<dicCosts>();

private List<dicIncome> \_dicIncome = new List<dicIncome>();

private BindingSource bs = new BindingSource();

private AddForm()

{

InitializeComponent();

CompleteDictioonary();

dateTimePicker.Value = DateTime.Now.Date;

dicCostsBindingSource.DataSource = \_dicCosts;

dicIncomeBindingSource.DataSource = \_dicIncome;

sumTextBox.Tag = false;

groupComboBox.Tag = false;

}

private void CompleteDictioonary()

{

using (var dataContext = HomeAccountingEntity.ServiceControllerBD.Instance.CreateModel())

{

\_dicCosts = dataContext.dicCosts.ToList();

\_dicIncome = dataContext.dicIncome.ToList();

}

}

public AddForm (dicEnum nameIn, CloseFormOptionEnum optionClose) : this()

{

\_dicName = nameIn;

\_withSave = optionClose == CloseFormOptionEnum.WithSave;

//анализируем как биндится в зависимости от нажатой клавиши

if (\_dicName == dicEnum.dicIncome)

{

groupComboBox.DataSource = dicIncomeBindingSource;

if(dicIncomeBindingSource.Count != 0)

{

groupComboBox.Tag = true;

}

bs.DataSource = incomeList;

}

else

{

groupComboBox.DataSource = dicCostsBindingSource;

if (dicCostsBindingSource.Count != 0)

{

groupComboBox.Tag = true;

}

bs.DataSource = costList;

}

dataGridView1.DataSource = bs;

}

public AddForm(dicEnum nameIn, CloseFormOptionEnum optionClose, List<Costs> listCosts) : this(nameIn, optionClose)

{

costList = listCosts;

bs.DataSource = costList;

dataGridView1.DataSource = bs;

}

#endregion

#region Button's click

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

{

if(\_dicName == dicEnum.dicIncome)

{

var item = \_dicIncome.FirstOrDefault(x => x.Id == (int) groupComboBox.SelectedValue);

//добавляем данные

var row = new Income

{

Comment = commentTextBox.Text,

Date = dateTimePicker.Value,

Sum = Convert.ToDecimal(sumTextBox.Text),

NameProd = item.Name,

idProduct = item.Id

};

incomeList.Add(row);

}

else

{

var item = \_dicCosts.FirstOrDefault(x => x.Id == (int)groupComboBox.SelectedValue);

var row = new Costs

{

Comment = commentTextBox.Text,

Date = dateTimePicker.Value,

Sum = Convert.ToDecimal(sumTextBox.Text),

NameProd = item.Name,

idProduct = item.Id

};

costList.Add(row);

}

bs.ResetBindings(false);

commentTextBox.Text = String.Empty;

sumTextBox.Text = String.Empty;

}

//при завершении формы вносим изменения в базу

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

{

if (\_withSave)

{

using (var dataContext = HomeAccountingEntity.ServiceControllerBD.Instance.CreateModel())

{

foreach (var income in incomeList)

{

dataContext.AddToIncome(income);

}

foreach (var costse in costList)

{

dataContext.Costs.AddObject(costse);

}

dataContext.SaveChanges();

}

incomeList.Clear();

costList.Clear();

}

}

#endregion

#region Защита от пользователей

private void sumTextBox\_KeyPress(object sender, KeyPressEventArgs e)

{

if(!Char.IsDigit(e.KeyChar) && !Char.IsControl(e.KeyChar))

e.Handled = true;

}

private void sumTextBox\_TextChanged(object sender, EventArgs e)

{

TextBox tb = (TextBox) sender;

tb.Tag = !String.IsNullOrEmpty(tb.Text);

AllValidated();

}

private void AllValidated()

{

addButton.Enabled = (bool) sumTextBox.Tag && (bool) groupComboBox.Tag;

}

private void groupComboBox\_Validated(object sender, EventArgs e)

{

ComboBox cb = (ComboBox)sender;

cb.Tag = !String.IsNullOrEmpty(cb.Text);

}

private void sumTextBox\_Validating(object sender, CancelEventArgs e)

{

const char a = (char)160;

sumTextBox.Text = sumTextBox.Text.Replace(a.ToString(), string.Empty);

if(!String.IsNullOrEmpty(sumTextBox.Text))

sumTextBox.Text = int.Parse(sumTextBox.Text).ToString("#,#");

}

#endregion

}

}

1. **CreditAddForm.cs**

namespace HomeAccounting

{

public partial class CreditAddForm : Form

{

private BindingSource bs = new BindingSource();

private List<Costs> listCost = new List<Costs>();

public CreditAddForm()

{

InitializeComponent();

bs.DataSource = listCost;

dataGridView1.DataSource = bs;

}

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

{

//первоначальные настройки диалога открытия файла

openFileDialog1.FileName = "";

openFileDialog1.InitialDirectory = "C:\\";

DialogResult dr = openFileDialog1.ShowDialog();

if (dr == DialogResult.OK)

fileTextBox.Text = openFileDialog1.FileName;

}

/// <summary>

/// Возвращает файл как поток байт(null и сообщение в случае ошибки)

/// </summary>

/// <returns></returns>

private byte[] AddCheck()

{

//добавляем данные

if (!String.IsNullOrEmpty(fileTextBox.Text) && !File.Exists(fileTextBox.Text))

{

MessageBox.Show("Файл не найден");

return null; //MessageBox.Show("Файл не найден");

}

else

{

return !String.IsNullOrEmpty(fileTextBox.Text) ? File.ReadAllBytes(fileTextBox.Text) : null;

}

return null;

}

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

{

var addForm = new AddForm(dicEnum.dicCosts, CloseFormOptionEnum.WithoutSave, listCost);

addForm.ShowDialog();

bs.ResetBindings(false);

}

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

{

var file = AddCheck();

if (file != null)

{

//Можно расширить форму и добавить дату и кмментарий для чека

var check = new Check()

{

CheckDate = DateTime.Now,

CheckData = file,

Comment = "Изображение чека"

};

//сохраняем файл если есть

BL.dataContext.Check.AddObject(check);

foreach (var costs in listCost)

{

costs.Check = check;

}

}

using (var dataContext = HomeAccountingEntity.ServiceControllerBD.Instance.CreateModel())

{

foreach (var costse in listCost)

{

dataContext.Costs.AddObject(costse);

}

dataContext.SaveChanges();

}

listCost.Clear();

BL.dataContext.SaveChanges();

}

}

}

1. **BL.cs**

namespace HomeAccounting

{

public static class BL

{

public static readonly HomeAccountingEntities dataContext = HomeAccountingEntity.ServiceControllerBD.Instance.CreateModel();

/// <summary>

/// Возвращает траты по датам.

/// </summary>

/// <param name="beginDate"></param>

/// <param name="endDate"></param>

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

/// <returns></returns>

public static IEnumerable<Costs> FromDateCosts(DateTime beginDate, DateTime endDate, string name)

{

IEnumerable<Costs> cost;

// using (var dataContext = HomeAccountingEntity.ServiceControllerBD.Instance.CreateModel())

// {

if (name == "Все")

{

cost = (from rows in dataContext.Costs

where rows.Date != null && (rows.Date.Value >= beginDate && rows.Date.Value < endDate)

select rows).ToList();

}

else

{

cost = (from rows in dataContext.Costs

where

rows.Date != null && (rows.Date.Value >= beginDate && rows.Date.Value < endDate) &&

rows.dicCosts.Name == name

select rows).ToList();

}

// }

return cost;

}

/// <summary>

/// Возвращает получения по датам.

/// </summary>

/// <param name="beginDate"></param>

/// <param name="endDate"></param>

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

/// <returns></returns>

public static IEnumerable<Income> FromDateIncome(DateTime beginDate, DateTime endDate, string name)

{

IEnumerable<Income> income;

if (name == "Все")

{

income = (from rows in dataContext.Income

where rows.Date != null && (rows.Date.Value >= beginDate && rows.Date.Value < endDate)

select rows).ToList();

}

else

{

income = (from rows in dataContext.Income

where rows.Date != null && (rows.Date.Value >= beginDate && rows.Date.Value < endDate) && rows.dicIncome.Name == name

select rows).ToList();

}

return income;

}

/// <summary>

/// Возвращает итоговую сумму

/// </summary>

/// <param name="list"></param>

/// <returns></returns>

public static decimal Itog(IEnumerable<Costs> list)

{

decimal itog = 0;

foreach (var costse in list)

{

if (costse.Sum != null) itog = itog + costse.Sum.Value;

}

return itog;

}

/// <summary>

/// Возвращает итоговую сумму

/// </summary>

/// <param name="list"></param>

/// <returns></returns>

public static decimal Itog(IEnumerable<Income> list)

{

decimal itog = 0m;

foreach (var costse in list)

{

if (costse.Sum != null) itog = itog + costse.Sum.Value;

}

return itog;

}

/// <summary>

/// Остаток средств за период

/// </summary>

/// <param name="beginDate"></param>

/// <param name="endDate"></param>

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

/// <returns></returns>

public static decimal ItogFromPeriod(DateTime beginDate, DateTime endDate, string name)

{

decimal itog = Itog(FromDateIncome(beginDate, endDate, name)) - Itog(FromDateCosts(beginDate, endDate, name));

return itog;

}

/// <summary>

/// Коллекция платежей по группам

/// </summary>

/// <param name="list"></param>

/// <param name="sum"></param>

/// <returns></returns>

public static IEnumerable<ReportStatistic> EnteringFromReportStatistic(IEnumerable<Costs> list, decimal sum)

{

var OutList = new List<ReportStatistic>();

foreach (var row in list)

{

var rowInOutList = OutList.Find(t => t.Name == row.NameProd);

if(rowInOutList!= null)

{

if (row.Sum != null) rowInOutList.Cost += row.Sum.Value;

}

else

{

if (row.Sum != null)

{

var per = new ReportStatistic()

{

Cost = row.Sum.Value,

Name = row.NameProd,

};

OutList.Add(per);

}

}

}

for (int i = 0; i < OutList.Count; i++)

{

OutList[i].Percent = (OutList[i].Cost/sum)\*100;

}

return OutList;

}

public static IEnumerable<ReportStatistic> EnteringFromReportStatistic(IEnumerable<Income> list, decimal sum)

{

var OutList = new List<ReportStatistic>();

foreach (var row in list)

{

var rowInOutList = OutList.Find(t => t.Name == row.NameProd);

if (rowInOutList != null)

{

if (row.Sum != null) rowInOutList.Cost += row.Sum.Value;

}

else

{

if (row.Sum != null)

{

var per = new ReportStatistic()

{

Cost = row.Sum.Value,

Name = row.NameProd,

};

OutList.Add(per);

}

}

}

for (int i = 0; i < OutList.Count; i++)

{

OutList[i].Percent = (OutList[i].Cost / sum) \* 100;

}

return OutList;

}

}

}