МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждения образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных технологий

Кафедра Программной инженерии

Специальность 1-40 05 01 Программное обеспечение информационных технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

по дисциплине «Объектно-ориентированные технологии программирования и стандарты проектирования»

Тема Программное средство «Менеджер доходов и расходов»

Исполнитель

Студент(ка) 2 курса группы 5 Таболич А.С.

(Ф.И.О.)

Руководитель работы ассистент Северинчик Н. А.

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

Курсовой проект защищен с оценкой

Председатель Северинчик Н. А.

(подпись)

Минск 2021

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc73093437)

[1 Аналитический обзор литературы 4](#_Toc73093438)

[1.1 Обзор прототипов 4](#_Toc73093439)

[1.2 Постановка требований 6](#_Toc73093440)

[2 Анализ требований к программному средству и разработка функциональных требований 7](#_Toc73093441)

[2.1 Постановка требований 7](#_Toc73093442)

[2.2 Диаграмма вариантов использования 8](#_Toc73093443)

[3 Разработка архитектуры проекта 10](#_Toc73093444)

[3.1 Обобщение структуры 10](#_Toc73093445)

[3.2 Схема работы программы 12](#_Toc73093446)

[3.3 Диаграммы UML 12](#_Toc73093447)

[3.4 Диаграмма классов 13](#_Toc73093448)

[3.5 Логическая модель базы данных 16](#_Toc73093449)

[4 Создание программного средства 19](#_Toc73093450)

[4.1 Реализация авторизации и регистрации пользователей 19](#_Toc73093451)

[4.2 Реализация функций программного средства 21](#_Toc73093452)

[4.3 Реализация импортирования транзакций из сервиса «Интернет-банкинг Беларусбанк» 23](#_Toc73093453)

[4.4 Функции для работы с базой данных 25](#_Toc73093454)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 26](#_Toc73093455)

[6 Руководство по установке и использованию 30](#_Toc73093456)

[6.1 Регистрация и авторизация 30](#_Toc73093457)

[6.2 Использование приложения пользователем 31](#_Toc73093458)

[ЗАКЛЮЧЕНИЕ 36](#_Toc73093459)

[ПРИЛОЖЕНИЕ А 37](#_Toc73093460)

[ПРИЛОЖЕНИЕ Б 38](#_Toc73093461)

# ВВЕДЕНИЕ

В современном мире одной из наиболее актуальных проблем общества является проблема трат и сбережений денежных средств. Большинство людей не следят сколько и на что они тратят деньги.

Грамотное планирование – залог успеха вашего бюджета.

Именно планирование и анализ бюджета поможет выявить бесполезные траты и отказаться от них. Среди будничной жизни такие покупки не откладываются в памяти. Однако в реальности составляют 10-20-30% от заработка и за год превышают размер зарплаты.

Деньги кончаются раньше, чем получаете очередную зарплату, и кажется, что на ваш доход невозможно прожить. При этом хочется приобрести что-то новое, большое и грандиозное. Возможно, Вам просто не хватает надёжного программного средства для учёта доходов и расходов. С ним Вы будете знать точно, на что уходят деньги, сможете спланировать крупные покупки, поймете, как сэкономить и приумножить деньги, а также защитите себя от финансовых рисков.

Приложение под названием «Money Manager» имеет удобный и понятный графический интерфейс, который не позволит потеряться даже самому не опытному пользователю. В данном приложение собраны основные функции для учёта доходов и расходов пользователя. Программное средство поможет вести учёт доходов и расходов, отслеживать операции и изменения за любые промежутки времени, просматривать статистику изменений и удобный календарь с отображением доходов и расходов.

Приложение «Money Manager» предоставит пользователю возможность просто и удобно следить за бюджетом каждый день.

1. **Аналитический обзор литературы**
   1. **Обзор прототипов**

Money Manager Expense & Budget – это полезная программа, помогающая управлять личными денежными ресурсами. Она предоставляет возможность контролировать доходы, расходы, общий денежный поток помесячно и формирует статистические данные.

Внешний вид интерфейса приложения приведен на рисунке 1.1.

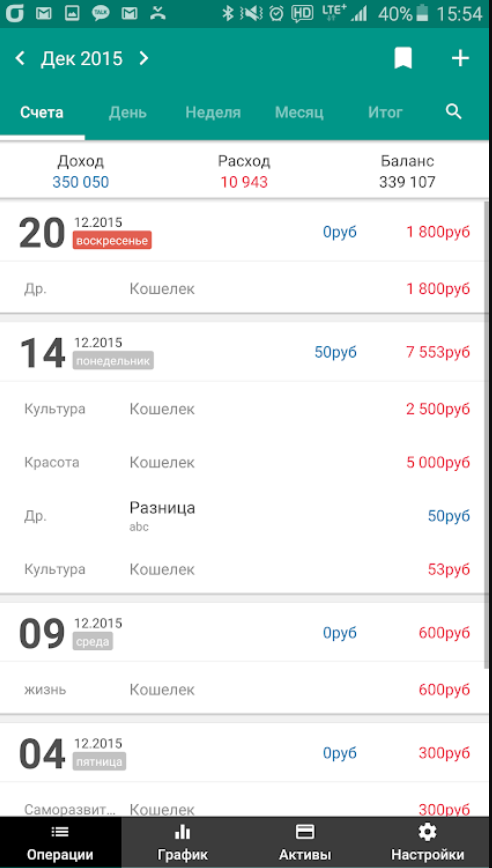


Рисунок 1.1 – Приложение Money Manager Expense & Budget

* Функция доступа с ПК Используя подключение Wi-fi, Вы можете редактировать и сортировать операции, категории доходов и расходов, группы счетов и другие данные прямо на экране Вашего компьютера. При помощи этой функции Вы можете отслеживать изменения ваших доходов и расходов, просматривая графики на Вашем ПК;
* Переводы и повторяющиеся операции Вы можете переводить средства между активами, что делает управление ими более удобным. Так же вы можете оптимизировать управление зарплатами, рассроченными платежами и выплатами по кредитам, создавая повторяющиеся операции;
* Удобная статистика В любой момент времени Вы можете отследить распределение Ваших доходов и расходов по категориям, изучить графики изменения состояния активов, отследить доходы, расходы по каждому активу;
* Управление бюджетами Вы можете планировать бюджеты по отдельным категориям и легко отслеживать расходы по ним с помощью удобной и понятной инфографики;
* Также имеются другие функции, доступные по платной подписке.

Monefy — планирование бюджета и учет расходов. Monefy поможет вам вести учет расходов. Это ваш личный финансовый менеджер и планировщик. Простое управление, легкий подсчет финансов, разумная экономия и домашняя бухгалтерия.

Внешний вид интерфейса приложения приведен на рисунке 1.2.

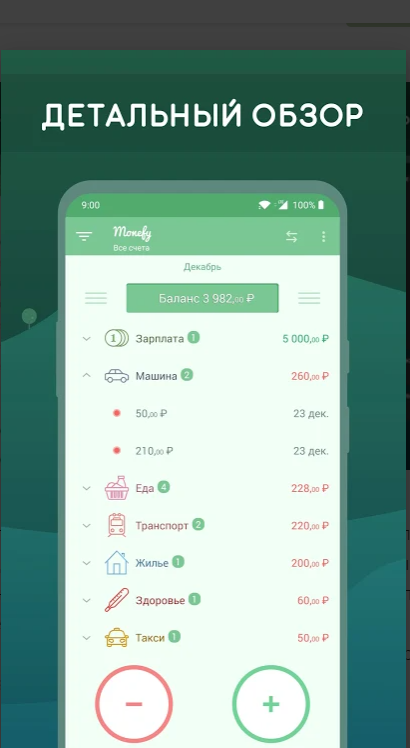


Рисунок 1.2 – Приложение Monefy

Основные функции Monefy:

* Интуитивно понятный и удобный интерфейс. Ничего лишнего;
* Мгновенное добавление новых записей;
* Просмотр расходов на красивом и информативном графике и детальная информация в списке под ним;
* Управление категориями;
* Безопасная синхронизация через Ваш личный Dropbox аккаунт;
* Выбор отчетного периода;
* Выбор валюты;
* Режим бюджета для более эффективного контроля затрат за определенный период;
* Также имеются другие функции, доступные по платной подписке.

Budget Planning - Finance Control — приложение, которое совмещает в себе функции планировщика бюджета и финансового калькулятора для успешного контроля денежных средств.

Внешний вид интерфейса приложения приведен на рисунке 1.3.

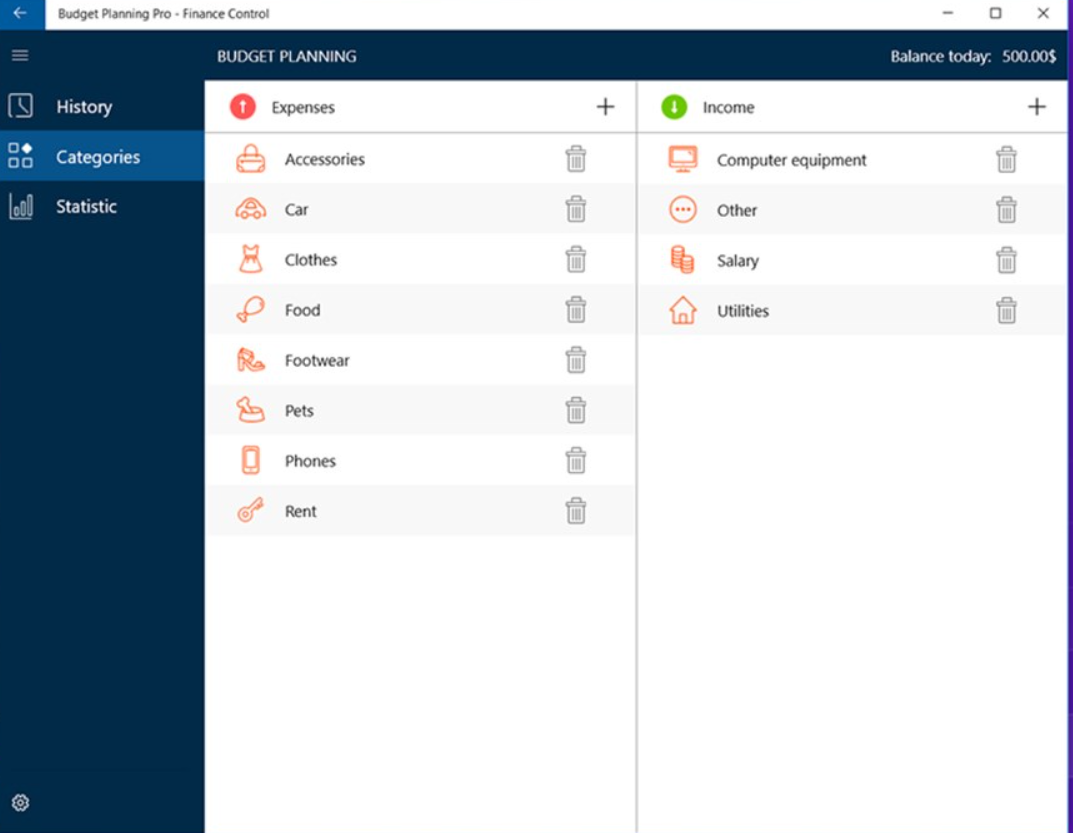


Рисунок 1.3 – Приложение Budget Planning - Finance Control

Основные функции Budget Planning - Finance Control:

* Добавление, удаление, редактирование транзакций;
* Добавление, удаление, редактирование категорий;
* Просмотр статистики в виде круговой диаграммы;
* Просмотр сводки в виде текстовой информации за день, месяц и год;
* Фильтры и сортировка транзакций;
* Уведомления о планируемых транзакциях.
  1. **Постановка требований**

Задачей данной курсовой работы является разработка приложения, которое будет представлять собой учёт доходов и расходов.

Функционально ПС должно выполнять следующие задачи:

* Управление доходами и расходами (добавление, редактирование, удаление);
* Типизация доходов и расходов на виды их источников (зарплата, подарок, покупки и т.д.);
* Предоставление информации о доходах и расходах за определённый промежуток времени;
* Добавление повторяющихся доходов и расходов.

1. **Анализ требований к программному средству и разработка функциональных требований**

Цель анализа требований в проектах – получить максимум информации о заказчике и специфике его задач, уточнить рамки проекта, оценить возможные риски. На этом этапе происходит идентификация принципиальных требований методологического и технологического характера, формулируются цели и задачи проекта, а также определяются критические факторы успеха, которые впоследствии будут использоваться для оценки результатов внедрения. Определение и описание требований – шаги, которые во многом определяют успех всего проекта, поскольку именно они влияют на все остальные этапы.

* 1. **Постановка требований**

Различают три уровня требований к проекту:

* бизнес-требования;
* пользовательские требования;
* функциональные требования.

Бизнес-требования содержат высокоуровневые цели организации или заказчиков системы. Как правило, их высказывают те, кто финансируют проект, покупатели системы, менеджер реальных пользователей, отдел маркетинга. Курсовой проект не подразумевает наличие заказчика, который мог бы выдвинуть бизнес-требования, поэтому в качестве таких высокоуровневых требований можно рассматривать общие требования к разрабатываемому средству. К их числу относятся:

* простота и лёгкость интерфейса;
* использование принципов объектно-ориентированного программирования;
* использование архитектурных шаблонов проектирования;
* использование системы управления базами данных (СУБД);

Весь дальнейший процесс проектирования и разработки программного средства должен находиться в очерченных бизнес-требованиями границах.

Следующими требованиями являются требования пользователей. Данные требования описывают цели и задачи, которые пользователям позволит решить система. Таким образом, в пользовательских требованиях указано, что клиенты смогут делать с помощью системы. Пользователь данного программного средства имеет возможность:

* регистрировать себя в системе;
* входить в приложение, после ввода данных, необходимых для аутентификации;
* просматривать доходы и расходы;
* добавлять доходы и расходы для требуемой категории;
* изменять свои доходы и расходы;
* добавлять повторяющиеся доходы и расходы;
* просматривать статистику и календарь своих изменений;
* фильтровать доходы и расходы;
* импортировать данные о доходах и расходах из системы «Интернет-банкинг Беларусбанк».

После проведения анализа были выявлены следующие функциональные требования:

* архитектура приложения должна соответствовать шаблонy проектирования MVVM;
* вся информация должна храниться в базе данных;
* приложение должно производить валидацию вводимых пользователем данных;
* приложение должно корректным образом обрабатывать возникающие исключительные ситуации: отображать понятное для пользователя сообщение о возникшей ошибке;
* приложение должно предоставлять пользователям возможность создания нового аккаунта в виде регистрационной формы;
* приложение должно предоставлять возможность пользователям проходить аутентификацию и входить в систему под соответствующим введенным данным пользовательским именем;
* приложение должно предоставлять возможность поиска по балансу по следующим критериям: категория, дата, только доходы, только расходы.

Таким образом, был проведен тщательный анализ требований к программному средству, который позволил разработать список функциональных требований. Разработка данной программной системы должна проводиться в соответствии с сформированными списком.

* 1. **Диаграмма вариантов использования**

В диаграмме использования проектируемая система представляется в виде множества сущностей или актеров, взаимодействующих с системой с помощью, так называемых прецедентов. При этом актером или действующим лицом называется любая сущность, взаимодействующая с системой извне. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемый системой при диалоге с пользователем. При этом ничего не говорится о том, каким образом будет реализовано взаимодействие пользователя с системой. Диаграмма вариантов использования для пользователя приведена на рисунке 2.1.

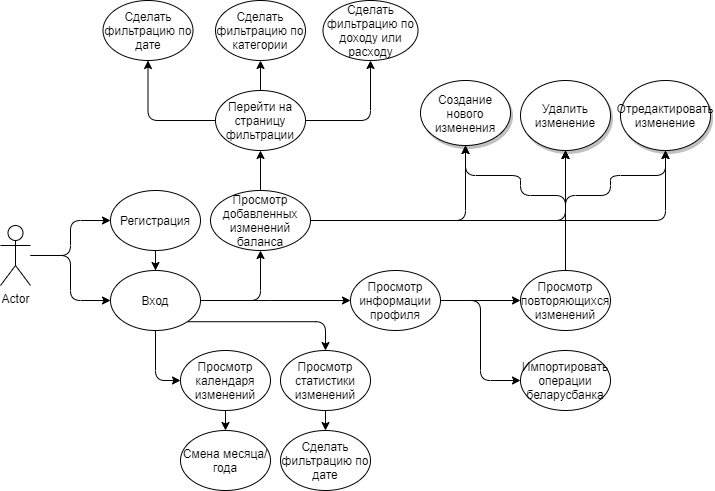
****

Рисунок 2.1. – Диаграмма вариантов использования для обычного пользователя

На данной диаграмме можно увидеть основные действия, которые доступ­ны пользователю: регистрация, авторизация в приложении, работа с историей добавленных изменений и сопутствующие действия: создание нового изменения, редактирование ранее добавленного изменения, удаление ранее добавленного изменения, фильтрация добавленных доходов и расходов по таким критериям как: дата, категория, также можно выбрать только доходы или только расходы. Пользователь также может добавить повторяющиеся изменения, которые будут повторяться каждый месяц. Пользователь может просматривать расходы на круговой диаграмме, просматривать доходы и расходы на календаре. У пользователя есть возможность импортировать транзакции из системы «Интернет-банкинг Беларусбанк» и добавить их в список своих доходов и расходов.

1. **Разработка архитектуры проекта**
   1. **Обобщение структуры**

Решение состоит из пяти проектов: MoneyManager.Core, MoneyManager.Infrastructure, MoneyManager.Main, PieChart, WPF.EventCalendar. Обобщённая структура решения представлена на рисунке 3.1.

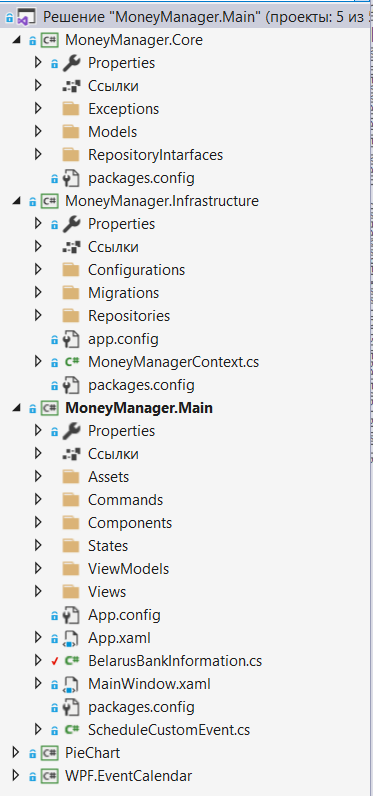


Рисунок 3.1 – Обобщённая структура решения

Данный курсовой проект основывается на реализации нескольких паттернов. В первую очередь, говоря об архитектуре проекта, необходимо упомянуть паттерн MVVM (Model-View-ViewModel), который позволяет отделить логику приложения от визуальной части. Разберём каждый из пяти проектов решения по отдельности.

Рассмотрим составляющие проекта MoneyManager.Core, в котором описаны Models паттерна MVVM:

* Exceptions – папка, в которой хранятся пользовательские классы исключений для всего проекта, чтобы обрабатывать конкретные типы исключений;
* Models – папка с моделями, описывающая используемые в приложении данные;
* RepositoryInterfaces ­– папка, в которой хранятся интерфейсы для паттерна Repository.

Далее рассмотрим проект MoneyManager.Infrastructure, который отвечает за работу с базой данных нашего решения. В данном проекте осуществляется основная реализация паттерна Repository. Данный паттерн позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает приложение, это своего рода промежуточное звено между моделями и остальной программой.

Перейдём к составляющим данного проекта:

* Configurations – папка, в которой хранятся конфигурации, благодаря которым осуществлялось непосредственное создание ключей в моделях и связей между ними для базы данных;
* Migrations – папка миграций базы данных, которые позволяют вносить изменения в базу данных при изменениях моделей и контекста данных;
* Repositories ­– папка, хранящая репозитории моделей, благодаря которым осуществлялась связь между моделями и базой данных. В данной папке также реализован паттерн UnitOfWork, который на практике упрощает использование репозиториев в проекте.
* MoneyManagerContext – класс, в котором создаётся соединение с MS SQL Server и непосредственно создаётся база данных со всеми таблицами.

MoneyManager.Main – проект в котором обрабатывается основная логика приложения. В данном проекте хранится две из трёх частей паттерна MVVM, которые отвечают за отображение информации на экран, интерфейс. В этом проекте реализован паттерн Command. За счёт этого паттерна взаимосвязь между элементами пользовательского интерфейса и программной логикой осуществляется посредством привязки (Binding). Разберём основные элементы проекта:

* Assets – папка, содержащая картинки приложения.
* Commands – папка, содержащая в себе все команды, благодаря которым мы можем связать логику и интерфейс;
* Components – папка, в которой хранится элемент управления BindablePasswordBox;
* States – папка, в которой содержатся классы и интерфейсы для определения аккаунтов;
* ViewModels – папка с классами, предназначенными для определения методов для работы с интерфейсов;
* Views – папка, содержащая пользовательские элементы управления, которые отвечают за интерфейс приложения.

PieChart – проект, отвечающий за создание круговой диаграммы на основе доходов и расходов.

WPF.EventCalendar – проект, отвечающий за создание календаря изменений на основе доходов и расходов.

* 1. **Схема работы программы**

Благодаря блок-схемам, описывающим алгоритм работы программы, иногда можно сделать много выводов по работе программы. В приложении А приведена блок-схема для отдельного варианта использования. Допустим, пользователю необходимо добавить доход или расход в свою историю. На блок-схеме показана логика, которой необходимо следовать, чтобы достичь желаемого результата.

* 1. **Диаграммы UML**

UML – унифицированный язык моделирования (Unified Modeling Language) – это система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования.

Его можно использовать для визуализации, спецификации, конструирова­ния и документирования программных систем.

Диаграмма – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей).

Диаграммы последовательностей используются для уточнения диаграмм прецедентов, более детального описания логики сценариев использования. Это отличное средство документирования проекта с точки зрения сценариев использования.

Диаграмма последовательности приведена на рисунке 3.2.

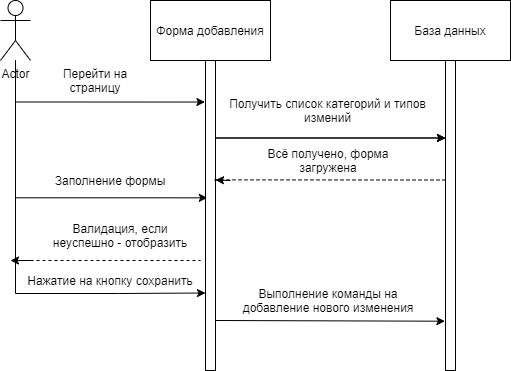


Рисунок 3.2 – Диаграмма последовательности для добавления дохода или расхода

Диаграмма компонентов – элемент языка моделирования UML, статическая структурная диаграмма, которая показывает разбиение программной системы на структурные компоненты и связи (зависимости) между компонентами. В качестве физических компонентов могут выступать файлы, библиотеки, модули, исполняемые файлы, пакеты и т. п.

Диаграммы компонентов приведена на рисунке 3.3.

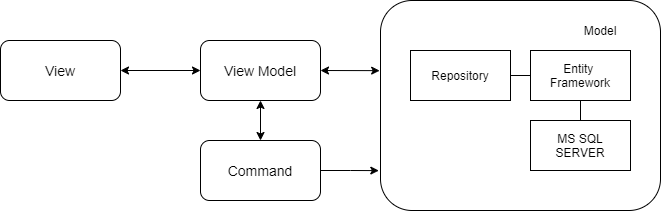
****

Рисунок 3.3 – Диаграмма компонентов

Так как приложение построено на архитектуре паттерна MVVM, то присутствует 3 основных блока, View, ViewModel и Model. В Model входит паттерн репозиторий, классы, созданные Entity Framework способом подключения Code First, и несколько вспомогательных классов, среди которых есть паттерн Singleton. View представляет собой внешний вид приложения, а два эти компонента связаны между собой благодаря ViewModel. Данный паттерн позволяет легко извлекать информацию из базы данных и передать ее сначала во ViewModel, а затем уже отобразить во View. Благодаря паттерну Command связываются Model и ViewModel.

* + 1. **Диаграмма классов**

Диаграмма классов служит для представления статической структуры мо­дели системы в терминологии классов объектно-ориентированного программирования. Диаграмма классов может отражать, в частности, различные взаимосвязи между отдельными сущностями предметной области, такими как объекты и подсистемы, а также описывает их внутреннюю структуру (поля, методы) и типы отношений (наследование, реализация интерфейсов). На данной диаграмме не указывается информация о временных аспектах функционирования системы. С этой точки зрения диаграмма классов является дальнейшим развитием концептуальной модели проектируемой системы.

Диаграмма классов моделей паттерна MVVM представлена в приложении Б.

Все классы, представленные в данной диаграмме, кроме Entity (этот класс предназначен для создания идентификаторов), создаются для создания таблиц в базе данных.

Диаграмма всех классов проекта MoneyManager.Core (без папки Exceptions) представлена на рисунке 3.4.

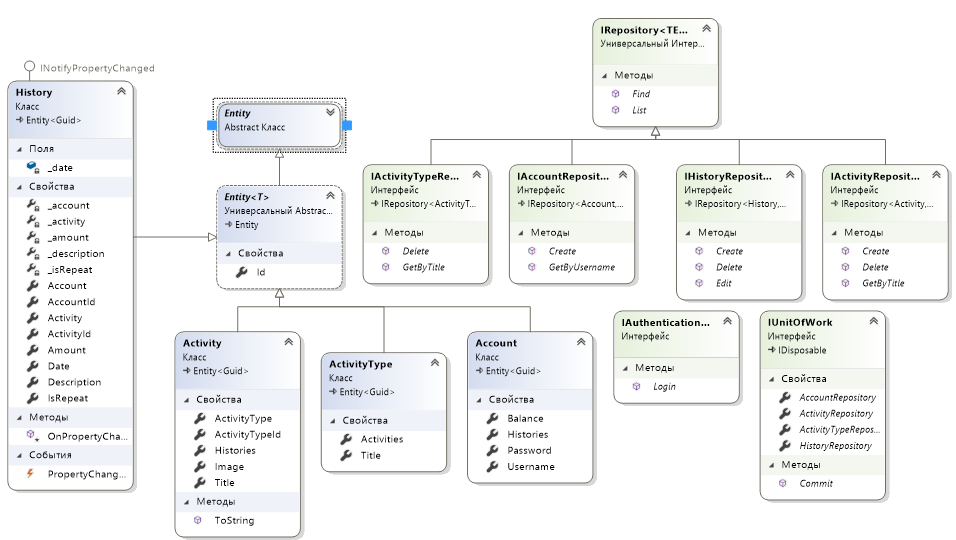
****

Рисунок 3.4 – Диаграмма классов проекта MoneyManager.Core

Классы моделей были описаны ранее. В данном проекте также описываются интерфейсы паттерна Repository. В этих интерфейсах объявляются методы, которые обязательно должны быть реализованы в соответствующих файлах-репозиториях. Интерфейсы создаются для реализации паттерна UnitOfWork.

Диаграмма всех классов проекта MoneyManager.Infrastructure представлена на рисунке 3.5.

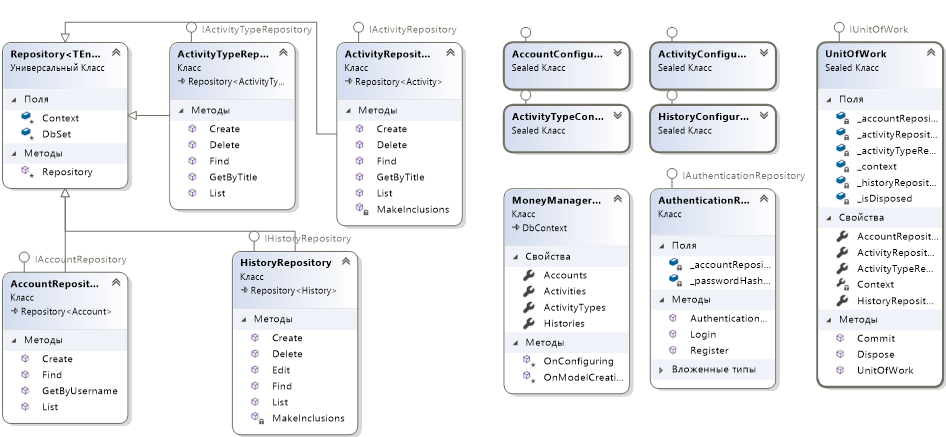


Рисунок 3.5 – Диаграмма классов проекта MoneyManager.Infrastructure

Классы папки Configurations необходимы для создания связей и ключей соответствующих по названиям таблиц.

Классы папки Repositories необходимы для реализации паттерна Repository. В этих классах описана логика взаимодействия моделей с базой данных. Определяется возможность создания, поиска и обновления объектов в базе данных, также их удаление из БД.

Класс UnitOfWork предназначен для реализации одноимённого паттерна.

В классе MoneyManagerContext описывается непосредственное соединение с сервером MS SQL и создания таблиц БД.

Диаграмма всех классов проекта MoneyManager.Main представлена на рисунке 3.6.

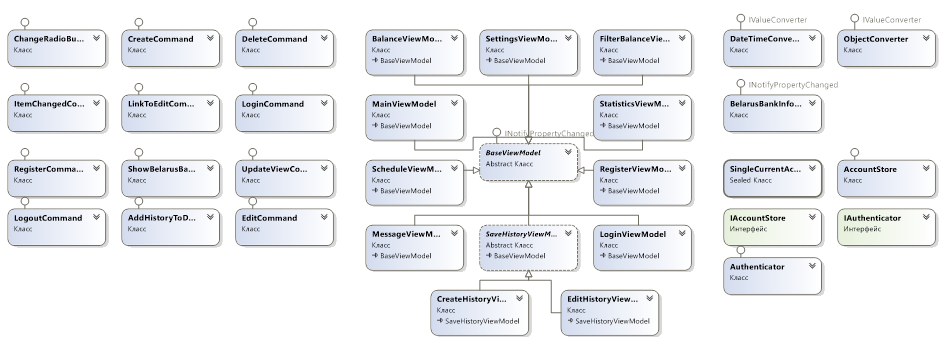


Рисунок 3.6 – Диаграмма классов проекта MoneyManager.Main

В папке Commands содержатся все команды, которые позволяют связывать логику и интерфейс.

Класс AddHistoryToDatabaseCommand реализует команду добавления пользователем доходов или расходов, полученных импортированием из сервиса «Интернет-банкинг Беларусбанк», в свою историю; ChangeRadioButtonCommand – команда обновляет список доходов или расходов в зависимости от выбранного параметра; CreateCommand ­– команда создания дохода или расхода; DeleteCommand – команда удаления дохода или расхода из истории пользователя; EditCommand – команда редактирования дохода или расхода в истории; ItemChangedCommand – команда обновляет список категорий в зависимости от выбранного параметра; LinkToEditCommand – команда, которая запоминает объект редактирования; LoginCommand – команда для регистрации пользователя; RegisterCommand – команда для регистрации пользователя; ShowBelarusBankHistoriesCommand – команда отображения импортированных доходов и расходов из сервиса «Интернет-банкинг Беларусбанк»; UpdateViewCommand – команда смены пользовательского элемента управления; LogoutCommand – команда для выхода из аккаунта.

В папке States хранятся файлы для работы с аккаунтом пользователя. Класс AccountStore определяет данные авторизованного пользователя в текущей сессии, Authenticator содержит методы для аутентификации пользователя.

В папке ViewModels хранятся классы, связывающие Models и Views паттерна MVVM, а также вызывающие команды, которые реализуют логику для отображения в интерфейсе.

Класс BaseViewModel является базовым для остальных ViewModel в этой папке, в нём реализован метод OnPropertyChanged, предназначенный для обновления данных в интерфейсе.

Остальные классы в папке ViewModels связывают одноимённые пользовательские элементы в папке Views и соответствующую Model.

BelarusBankInformation класс, предназначенный для обработки импортирования и отображения доходов и расходов из сервиса «Интернет-банкинг Беларусбанк».

ScheduleCustomEvent класс, используемый для инициализации событий на календаре изменений.

DatetimeConverter предназначен для конвертирования входной строки в строку dd/MM/yyyy. ObjectConverter используется для конвертирования ActivityType в логический тип.

* 1. **Логическая модель базы данных**

Логическое (даталогическое) проектирование – создание схемы базы данных на основе конкретной модели данных, например, реляционной модели данных. Для реляционной модели дaнных логическая модель – набор схем отношений, обычно с указанием первичных ключей, а также «связей» между отношениями, представляющих собой внешние ключи. На этапе логического проектирования учитывается специфика конкретной модели данных, но может не учитываться специфика конкретной СУБД.

В реляционной модели, в отличие от иерархической или сетевой, не суще­ствует физических отношений. Вся информация хранится в виде таблиц (отношений), состоящих из рядов и столбцов. А данные двух таблиц связаны общими столбцами, а не физическими ссылками или указателями. Для манипуляций с рядами данных существуют специальные операторы.

В отличие от двух других типов СУБД, в реляционных моделях данных нет необходимости просматривать все указатели, что облегчает выполнение запросов на выборку информации по сравнению с сетевыми и иерархическими СУБД. Это одна из основных причин, почему реляционная модель оказалась более удобна. В реляционной модели, как объекты, так и их отношения представлены только таблицами, и ничем более.

Для реализации приложения была разработана база данных. Для создания базы данных был выбран Microsoft SQL Server 2019.

Модель базы данных — тип модели данных, которая определяет логическую структуру базы данных и принципиально определяет, каким образом данные могут быть сохранены, организованы и обработаны.

Логическая модель базы данных представлена на рисунке 3.7.

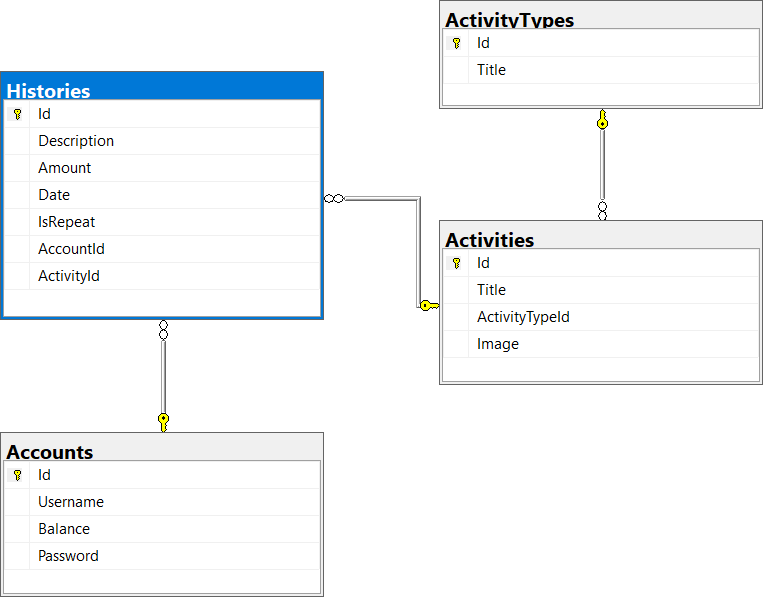


Рисунок 3.7 – Логическая модель базы данных

В базе данных содержится 4 таблицы.

В таблице Accounts хранятся учётные записи всех пользователей.

Таблица Accounts состоит из 4 столбцов:

* Id
* Username
* Balance
* Password

В столбце Username хранится информация об имени пользователя, в Password хранится хэшированный пароль пользователя, Balance – начальный баланс пользователя.

В таблице Histories хранится история доходов и расходов пользователя.

Таблица History состоит из 7 столбцов:

* Id
* Description
* Amount
* Date
* IsRepeat
* AccountId
* ActivityId

В столбце Description хранится описание дохода или расхода, в столбце Amount хранится информация о сумме дохода или расхода, Date – дата дохода или расхода, IsRepeat – является ли доход или расход повторяющимся каждый месяц, AccountId – идентификатор аккаунта, которому принадлежит доход или расход, ActivityId – идентификатор категории, к которой относится доход или расход.

В таблице Activities хранятся категории доходов расходов.

В таблице Activities 4 столбца:

* Id
* Title
* ActivityTypeId
* Image

В столбце Title хранится информация о названии категории, в ActivityTypeId – идентификатор типа изменения, Image – путь к картинке категории.

В таблице Activities хранятся категории доходов расходов.

В таблице Activities 2 столбца:

* Id
* Title

В столбце Title хранится информация о названии изменения.

1. **Создание программного средства**
   1. **Реализация авторизации и регистрации пользователей**

Чтобы у пользователя была возможность воспользоваться приложением, ему необходимо авторизоваться, если у него уже имеется аккаунт. В противном случае ему необходимо зарегистрироваться.

Далее на рисунках 4.1 и 4.2 находится код, реализующий регистрацию.



Рисунок 4.1 – Команда, вызывающая метод регистрации

Команда, представленная на рисунке 4.1 выполняет первый этап валидации входных данных из текстовых блоков для проверки их корректности, далее вызывается метод, которые непосредственно выполняет регистрацию, и после выполнения этого метода, сравнивает результат выполнения метода и либо выводит результат второго этапа валидации, либо переходит на страницу логина.



Рисунок 4.2 – Метод, осуществляющий регистрацию

В данном фрагменте кода определяется, возможна ли регистрация, если да, то осуществляется хэширование пароля и регистрация пользователя.

Код, обеспечивающий авторизацию, представлен на рисунках 4.3 и 4.4.

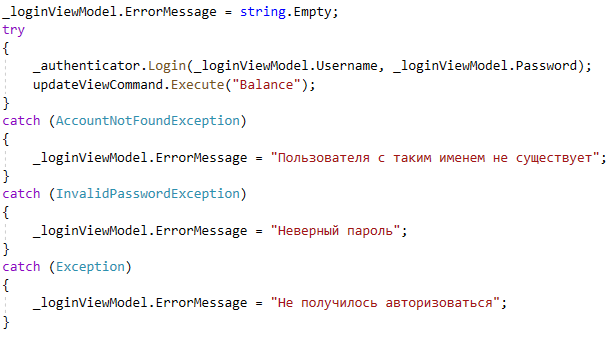


Рисунок 4.3 – Команда, вызывающая метод авторизации

Команда, представленная на рисунке 4.3 вызывает метод, который непосредственно выполняет авторизацию, и после выполнения этого метода, сравнивает результат выполнения метода и либо выводит результат валидации, либо переходит на страницу логина.

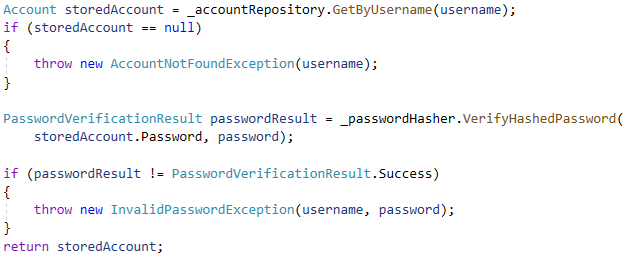


Рисунок 4.4 – Метод, осуществляющий авторизацию

В данном методе осуществляется поиск аккаунта с заданным именем в базе данных. При авторизации проверяется, введены ли данные, существует ли пользователь с таким именем, совпадают ли хэшированные пароли. Если аккаунт с таким именем существует и пароль верен, авторизация завершается успешно, иначе появляется сообщение об ошибке. Когда пользователь авторизуется, его данные записываются в класс SingleCurrentAccount, который реализован согласно паттерну Singleton.

* + - 1. **Реализация функций программного средства**

Из самой идеи приложения понятно, что одной из важнейших функций является возможность добавления доходов и расходов в историю. Код такой команды представлен на рисунке 4.5.

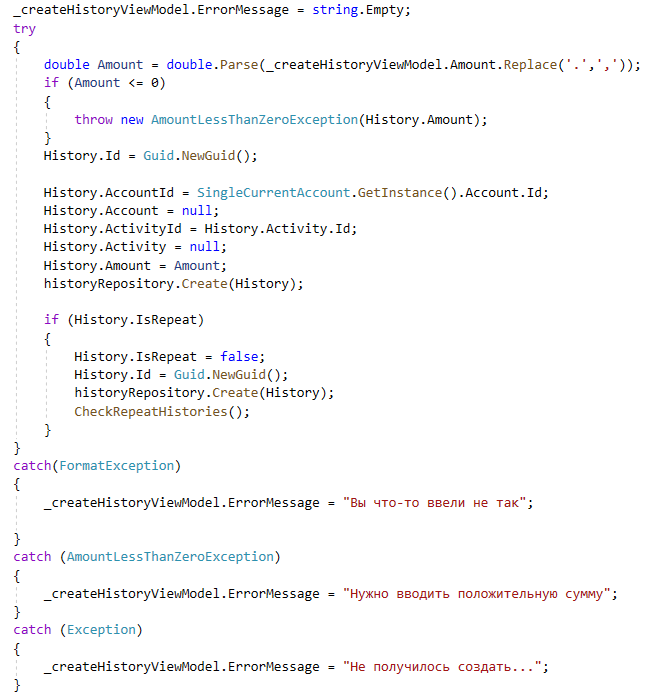
****

Рисунок 4.5 – Команда добавления дохода или расхода в историю

В данном коде осуществляется проверка на корректность введённых данных. Если данные корректны, определяем, какой аккаунт был авторизован, определяем категорию дохода или расхода, заполняем остальными данными и добавляем объект в базу данных, если пользователь выбрал повторяющийся доход или расход, будет проводится проверка по дате.

Проверка осуществляется в методе, представленном на рис. 4.6.

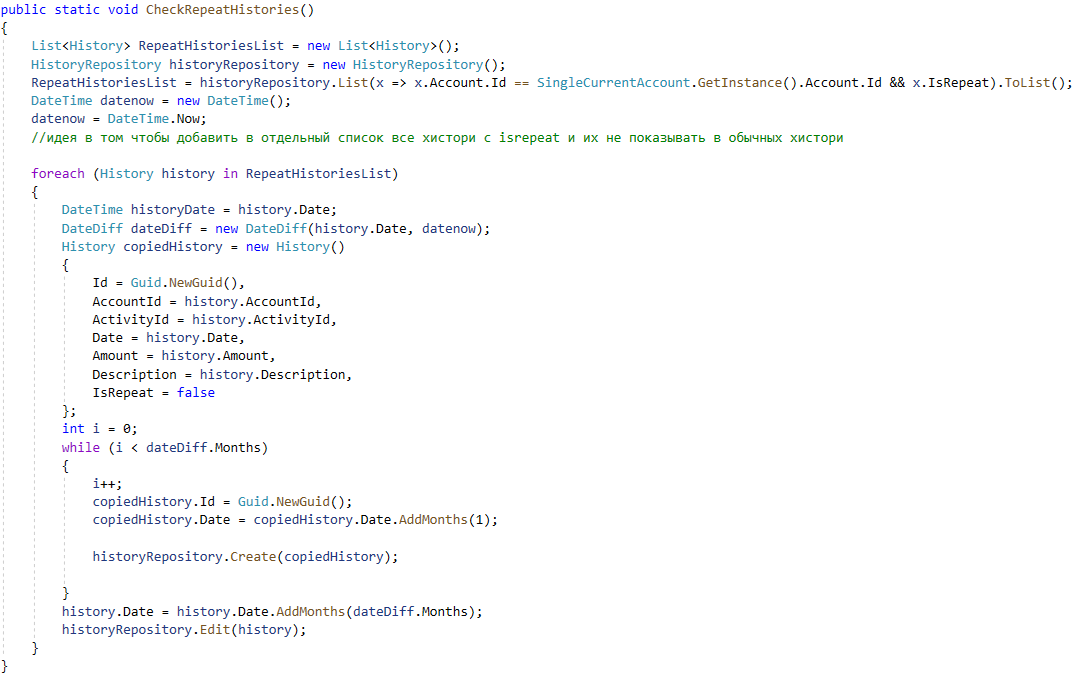


Рисунок 4.6 – Реализация проверки на повторяющиеся истории

В данном методе проверяется текущая дата с датой повторяющийся истории, если она меньше на месяц, то добавляется новая история.

1. **Реализация импортирования транзакций из сервиса «Интернет-банкинг Беларусбанк»**

Для импортирования транзакций из сервиса «Интернет-банкинг Беларусбанк» пользователь должен иметь доступ в Интернет и быть зарегистрированным на сервисе.

Реализация данной функции происходит с использованием Selenium WebDriver. По назначению Selenium WebDriver представляет собой драйвер браузера, то есть программную библиотеку, которая позволяет разрабатывать программы, управляющие поведением браузера. Изначально пользователь должен авторизоваться в системе «Интернет-банкинг» и далее произойдёт переход на страницу с таблицей транзакций.

На рисунке 4.7 приведён код перехода на страницу с таблицей

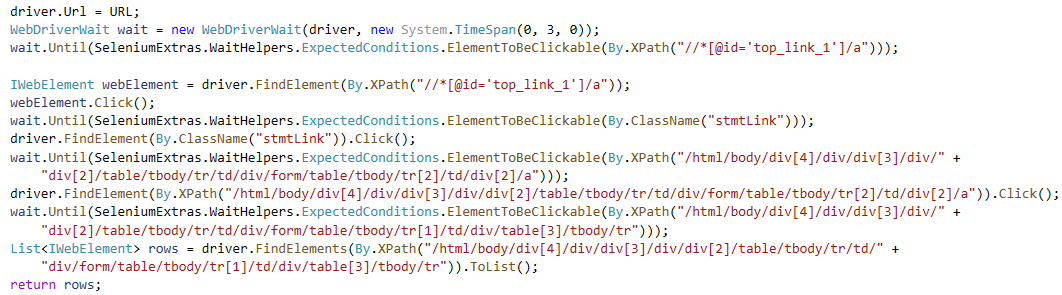


Рисунок 4.7 – Реализация перехода на страницу с транзакциями Беларусбанка

После перехода на страницу начнётся парсинг каждой строки таблицы транзакций и создание дохода или расхода для этой строки, также возможен парсинг по категориям. На рисунке 4.8 приведён код парсинга и создание списка History.

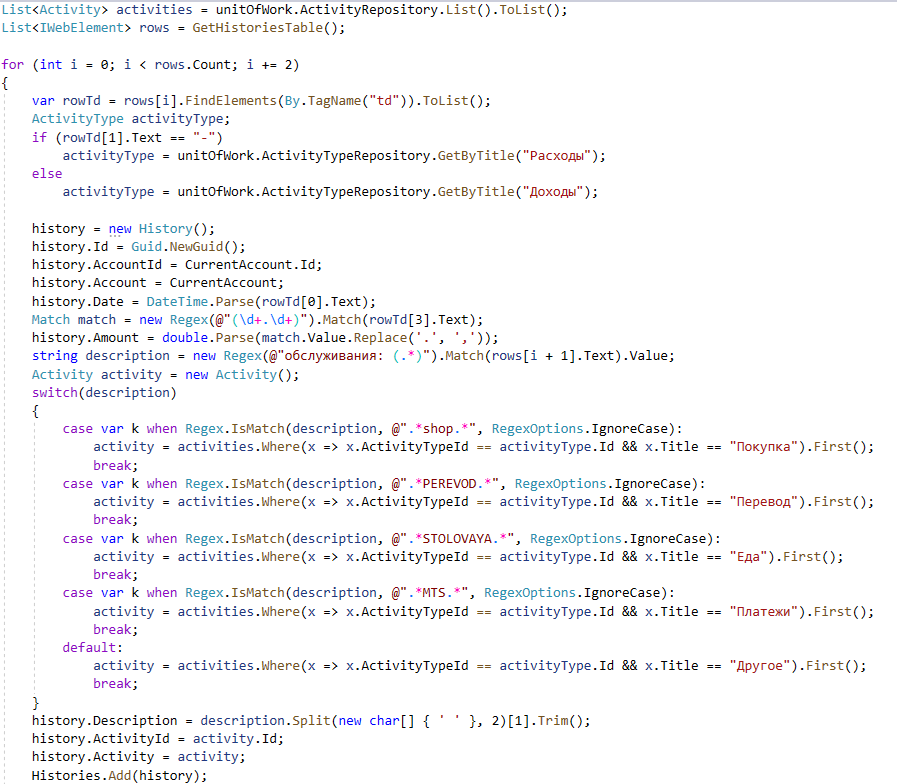


Рисунок 4.8 – Реализация парсинга страницы

* 1. **Функции для работы с базой данных**

Работа с базой данных осуществляется согласно паттерну Repository, который создает абстракцию при работе с базой данных, и дает возможность без каких-либо проблем изменить используемую базу данных.

На рисунке 4.7 представлен интерфейс, который обязательно необходимо реализовать.

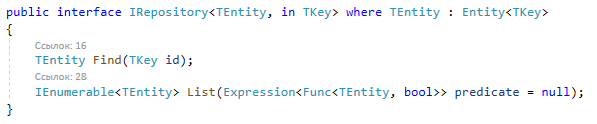


Рисунок 4.9 – Интерфейс для паттерна Repository

Интерфейс содержит функции поиска в базе данных по идентификатору и вывода информации в список с возможностью написания LINQ-запроса. Это те функции, которые должны быть реализованы у всех репозиториев в обязательном порядке, однако репозитории могут иметь свои функции.

Пример интерфейса для репозитория одного из классов приведён на рисунке 4.10.

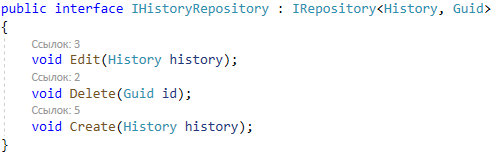


Рисунок 4.10 – Интерфейс для класса HistoryRepository

Данный интерфейс содержит следующие функции: удаление объекта по Id, редактирование объекта, создание объекта.

1. **Тестирование, проверка работоспособности и анализ полученных результатов**

В курсовом проекте задействует обработка исключений. Как только срабатывает исключение, пользователь получает соответствующее сообщение об ошибке.

Ниже на рисунках 5.1, 5.2 и 5.3 представлены результаты обработки исключений при неверно введённых данных при авторизации.

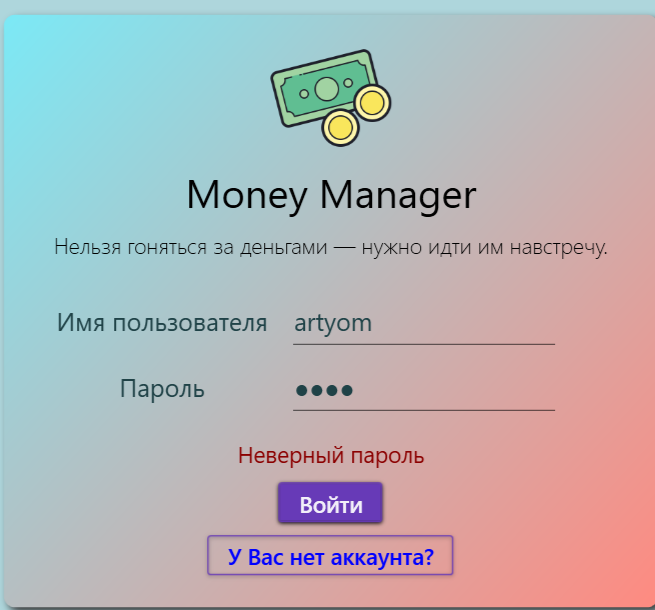


Рисунок 5.1 – Сообщение о неверно введённом пароле

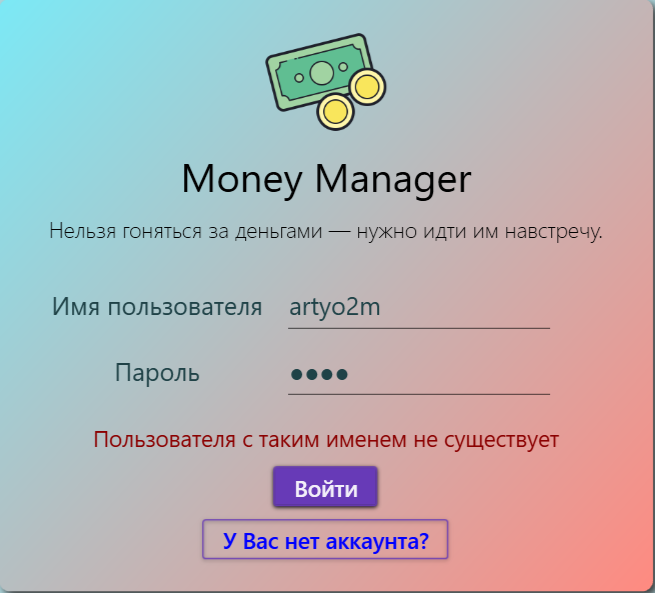


Рисунок 5.2 – Сообщение об отсутствии пользователя



Рисунок 5.3 – Блокировка кнопки при не введённых данных

На рисунках 5.4, 5.5, 5.6 и 5.7 представлены результаты обработки исключений при неверно введённых данных во время заполнения формы регистрации.

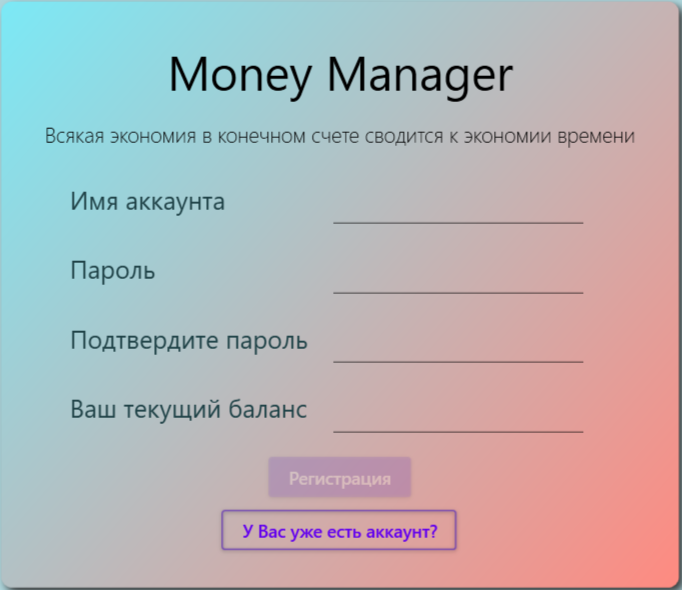


Рисунок 5.4 – Блокировка кнопки при не введённых данных

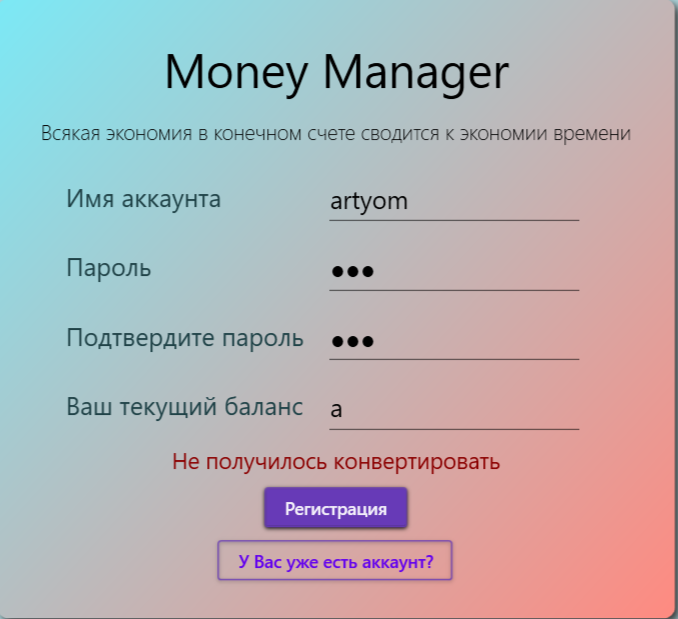


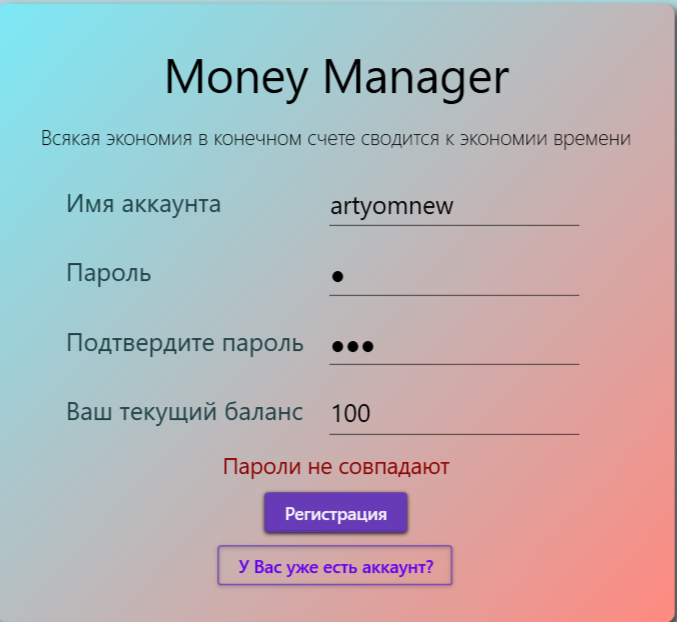
Рисунок 5.5 – Сообщение при неверных данных 

Рисунок 5.6 – Сообщение о разных паролях

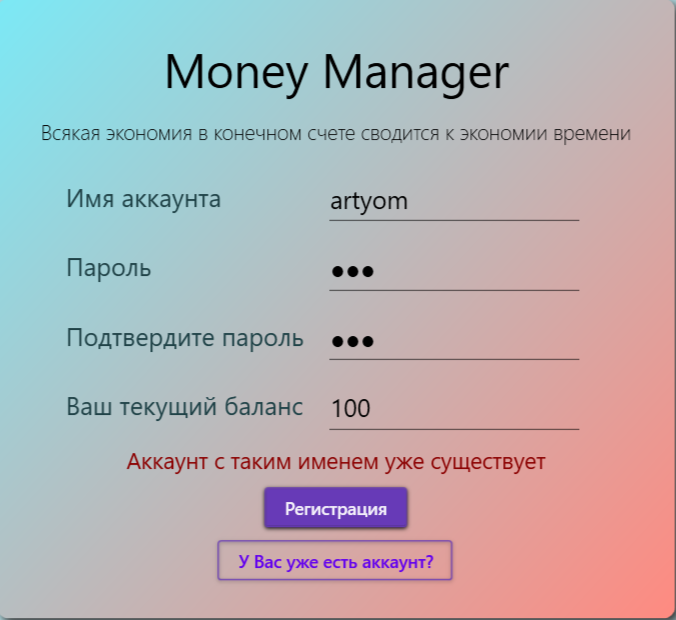


Рисунок 5.8 – Сообщение при регистрации пользователя с уже существующим именем пользователя

Во всех остальных окнах где присутствуют входные данные имеется такая же система обработки ошибок. Например, на рисунках 5.8 отображён вывод сообщения об ошибке в случае неверного ввода.

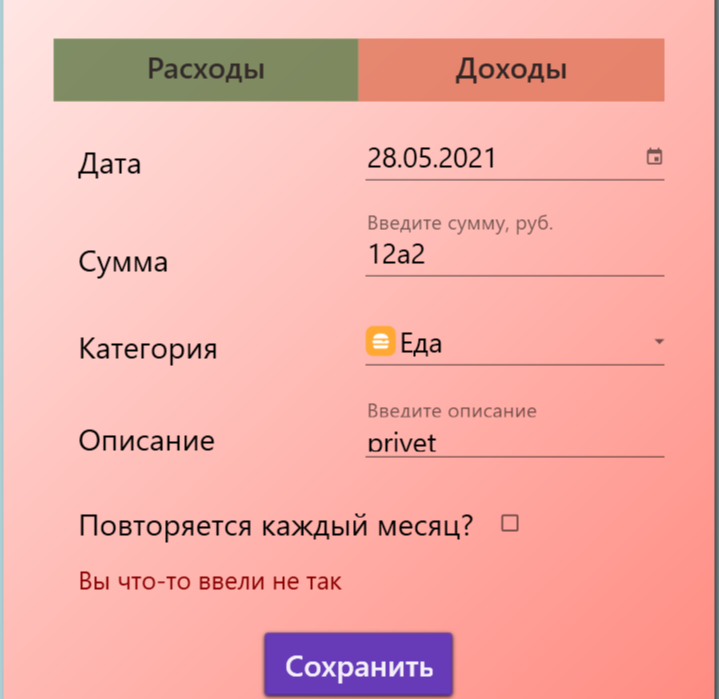


Рисунок 5.8 – Сообщение об ошибке

Таким образом, в приложении осуществляется максимально возможная обработка исключений и пользователь всегда будет в курсе, корректны ли вводимые им данные.

1. **Руководство по установке и использованию**
   1. **Регистрация и авторизация**

После запуска приложения пользователь видит главное окно приложения, на котором он может авторизоваться и получить возможность работать со своими проектами. Окно представлено на рисунке 6.1.

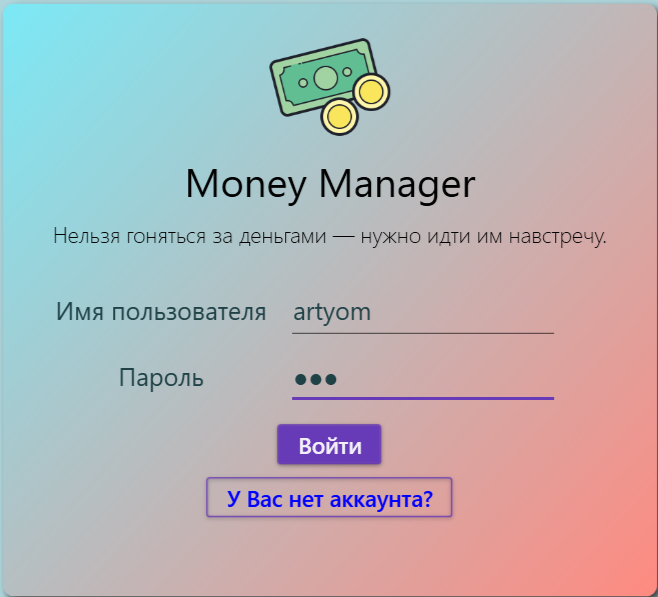


Рисунок 6.1 – Окно авторизации

Если пользователь ещё не имеет своего аккаунта, то по нажатию на соответствующую кнопку можно перейти к окну регистрации. Тут ему необходимо ввести свои личные данные: имя, текущий баланс. Так же пользователь должен придумать собственный пароль, который будет использовать для авторизации в приложении. Окно регистрации представлено на рисунке 6.2.

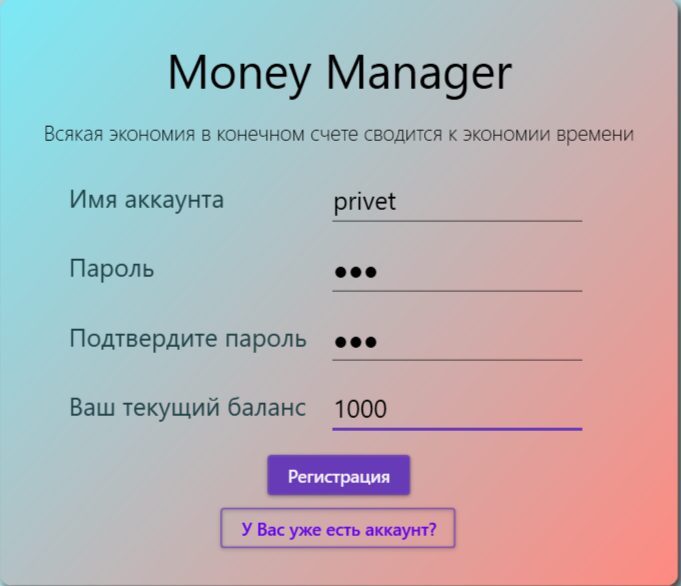


Рисунок 6.2 – Окно регистрации

1. **Использование приложения пользователем**

В случае успешного входа пользователь переходит на главную страницу приложения – история доходов и расходов, сгруппированных по дате. На этой странице будут отображаться все доходы и расходы, которые он добавил. Эта страница отображена на рисунке 6.3.

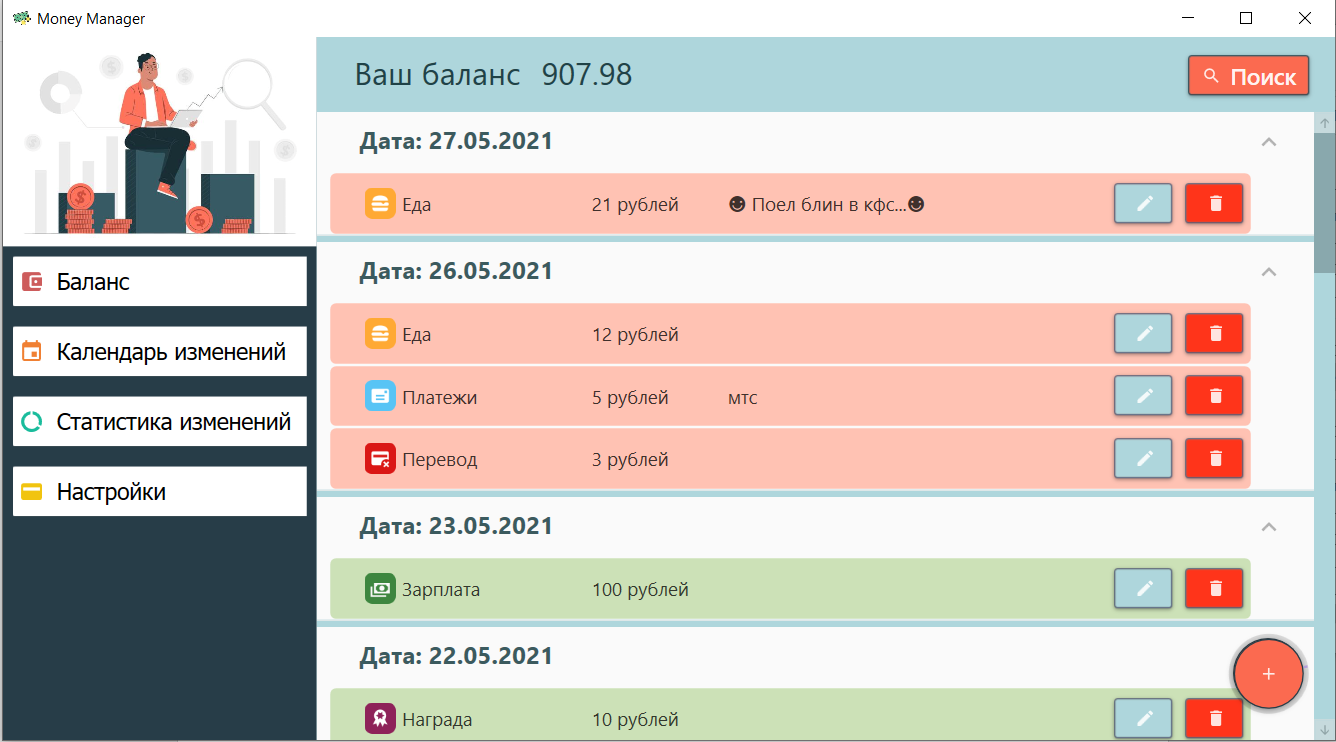
****

Рисунок 6.3 – Страница «Баланс»

При нажатии на кнопку добавления дохода или расхода пользователь переходит на новую страницу, изображённую на рисунке 6.4, где он может выбрать нужную ему категорию, ввести сумму, описание и, при необходимости, нажать на кнопку для повторения каждый месяц.

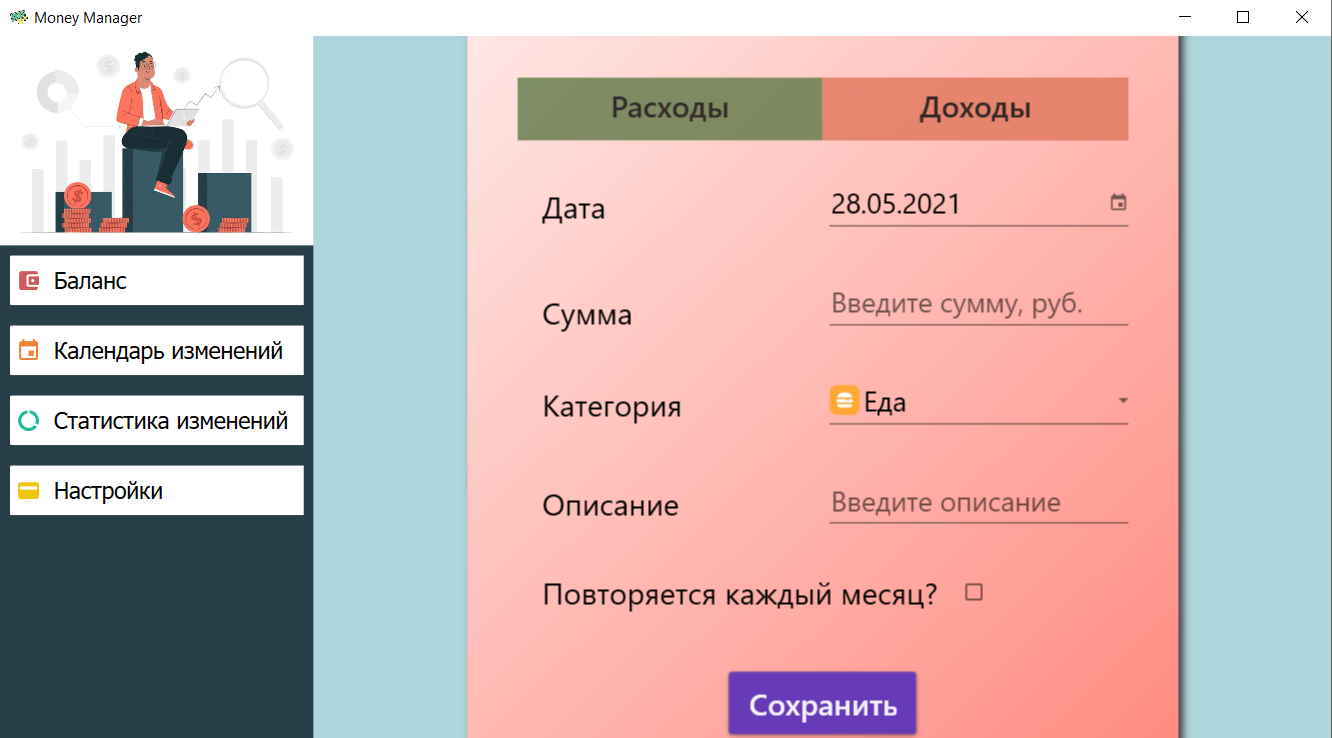


Рисунок 6.4 – Страница добавления дохода или расхода

Если же пользователь хочет увидеть конкретные категории доходов или расходов за определённый промежуток времени, он может перейти на страницу фильтрации и отфильтровать историю. Страница фильтрации представлена на рисунке 6.5.



Рисунок 6.5 – Страница фильтрации истории

При нажатии на кнопку редактирования пользователь попадает на страницу редактирования, которая представлена на рисунке 6.6. на этой странице пользователь может изменить ранее введённую категорию, сумму, описание, дату, а также поменять доход или расход на повторяющийся.

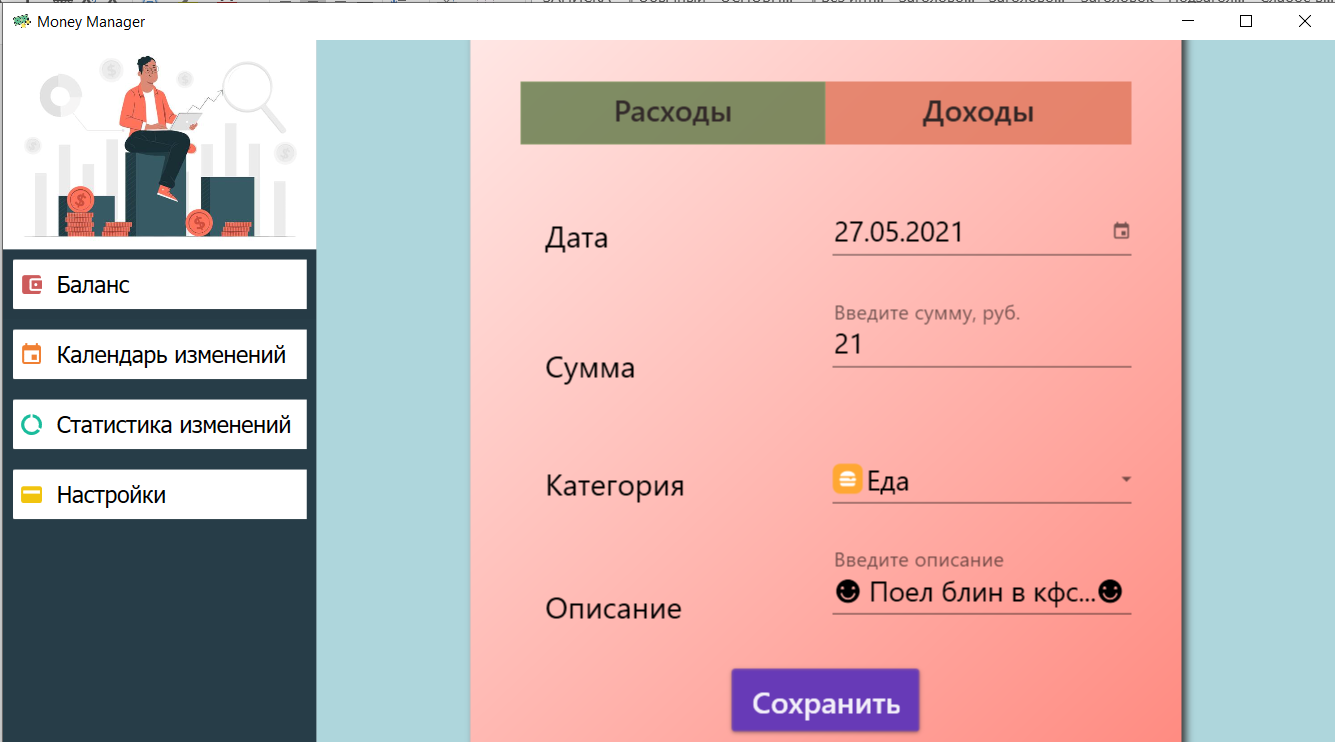


Рисунок 6.6 – Страница редактирования

При нажатии на кнопку меню «Календарь изменений» пользователь попадает на страницу, представленную на рисунке 6.7. На этой странице пользователь может просматривать свои доходы и расходы в календарном формате.

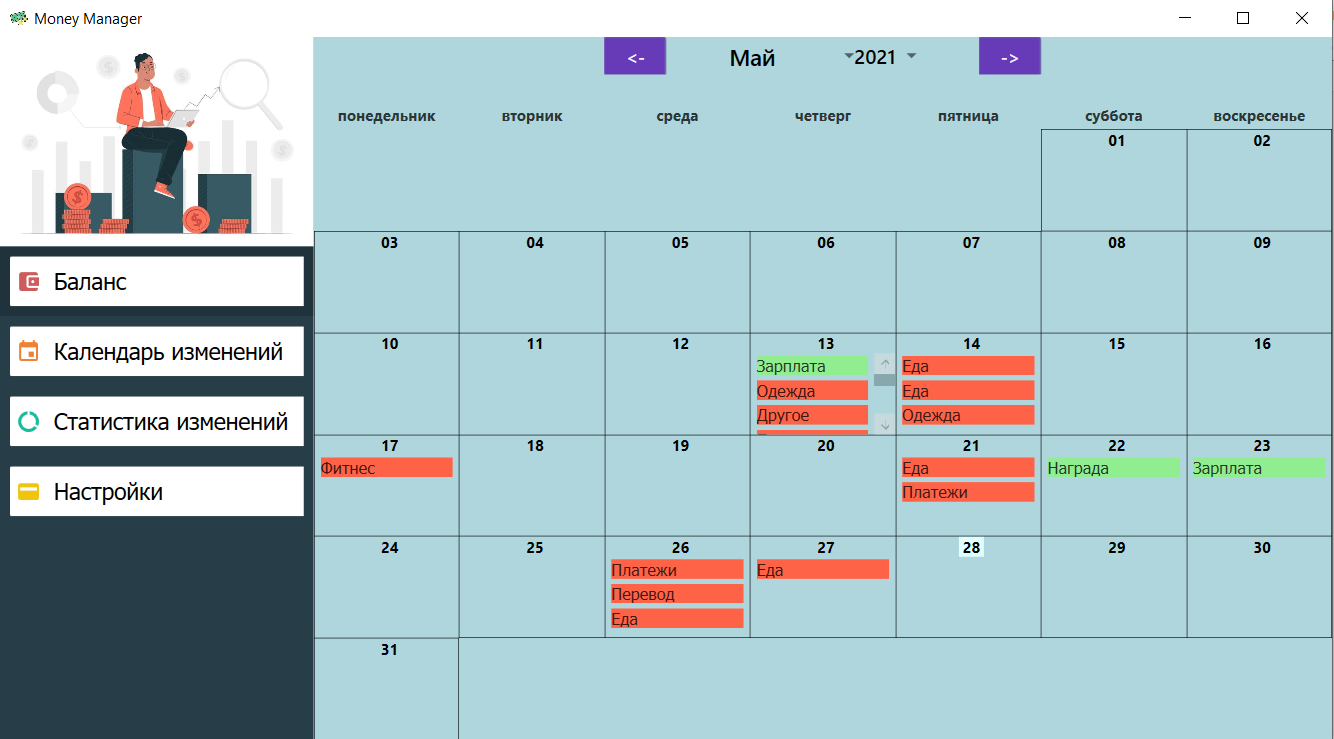


Рисунок 6.7 – Страница «Календарь изменений»

При нажатии на кнопку меню «Статистика изменений» пользователь попадёт на страницу, представленную на рисунке 6.8, где он может просмотреть статистику расходов и изменения за определённый промежуток времени.

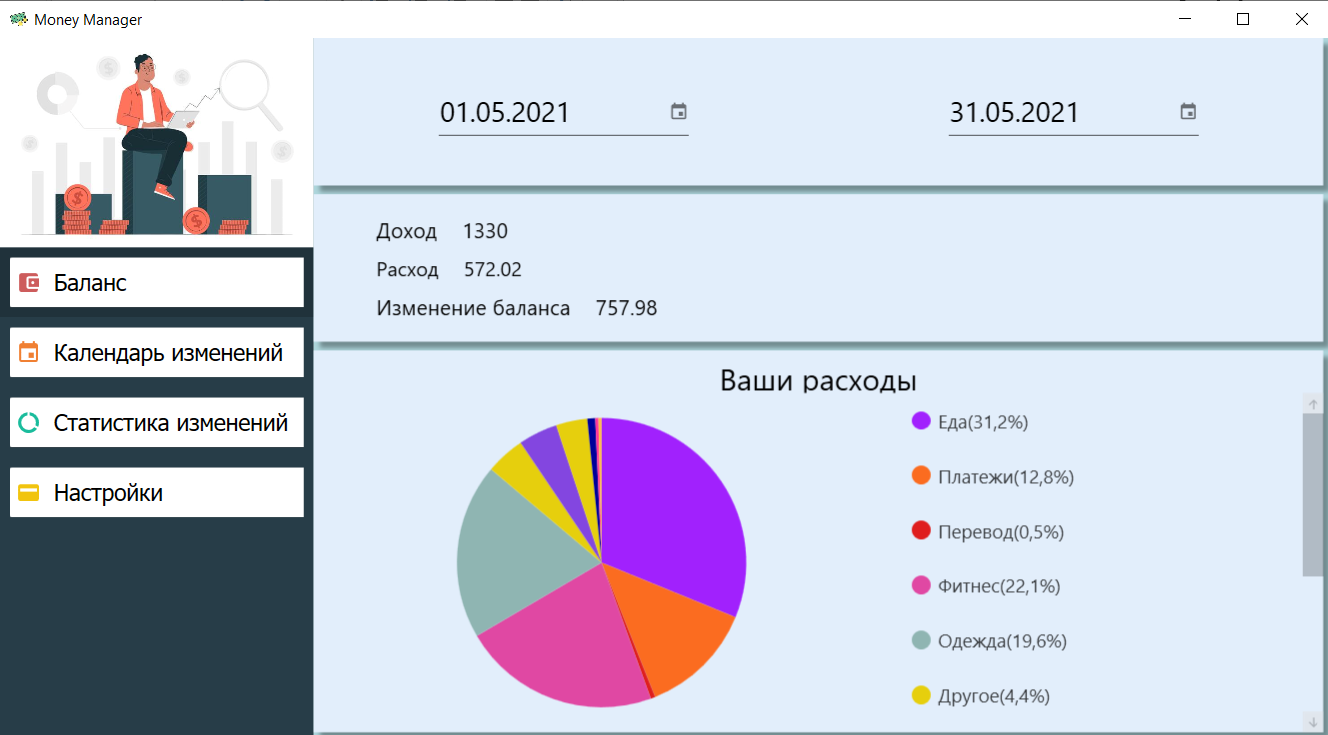


Рисунок 6.8 – Страница «Статистика изменений»

При нажатии на кнопку меню «Настройки» пользователь попадает на страницу настроек. На этой странице пользователь может просматривать свои данные, выйти из аккаунта, просмотреть, удалить отредактировать повторяющиеся доходы и расходы, а также импортировать транзакции из системы «Интернет-банкинг Беларусбанк». Импортирование происходит по нажатию на кнопку «Показать операции из карты», далее откроется браузер, где надо будет авторизоваться и далее все транзакции спарсятся в приложение. Страница настроек представлена на рисунке 6.9

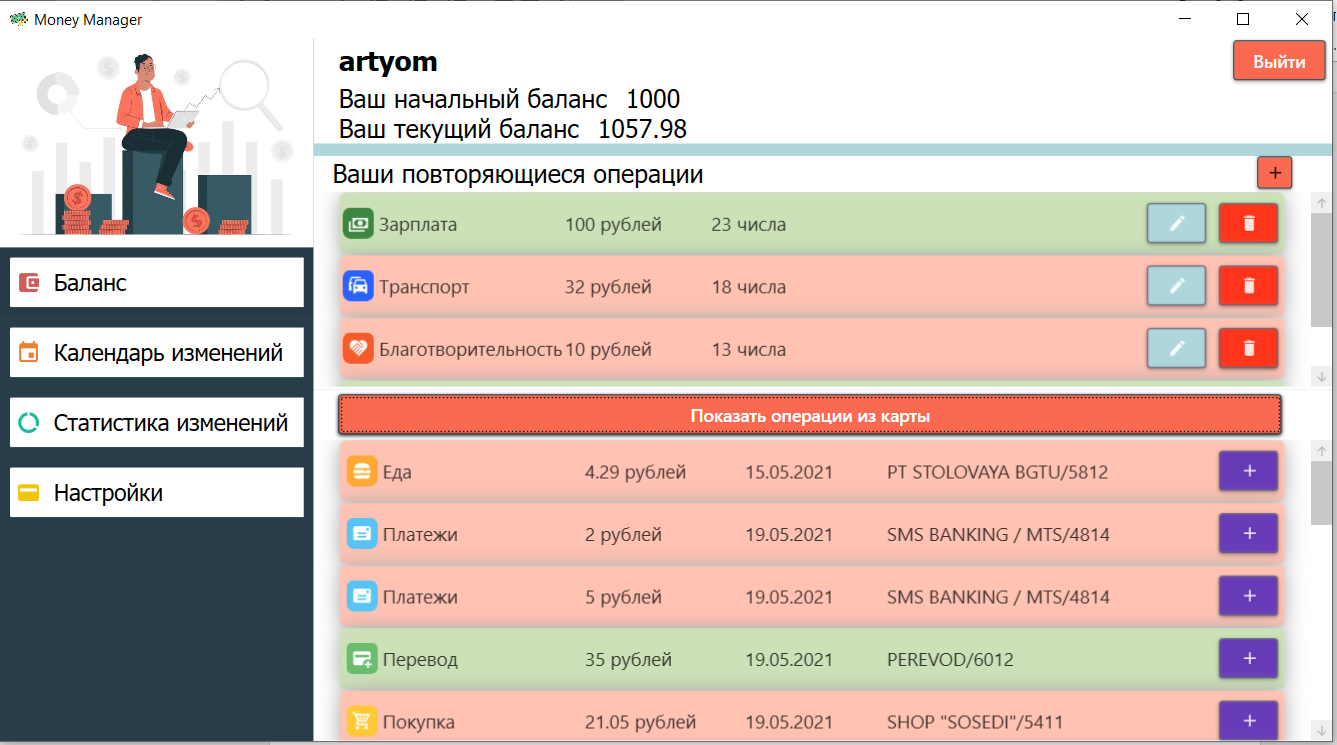


Рисунок 6.9 – Страница «Настройки»

Таким образом, было разработано программное средство, которое является интуитивно понятным и удобным.

**ЗАКЛЮЧЕНИЕ**

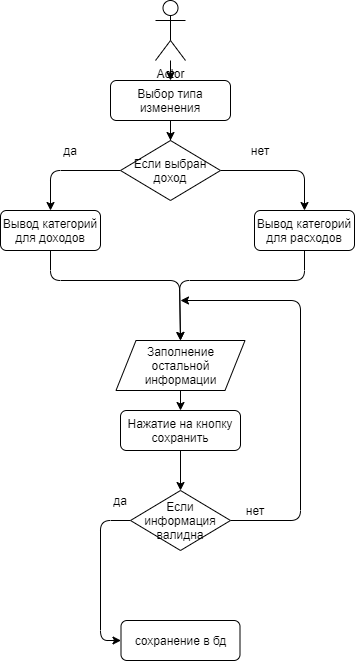
В итоге выполнения данного курсового проекта было разработано про­граммное средство «Менеджер доходов и расходов», представляющее возможность вести учёт доходов и расходов, просматривать изменения, просматривать статистику, делать импорт транзакций из «Интернет-банкинг Беларусбанк» и многое другое.

В процессе реализации программного средства были достигнуты выше указанные требования и задачи.

Также в процессе выполнения данного курсового проекта были закреплены навыки в программировании на языке C#, создании приложений на WPF, использование Entity Framework 3.1, работа с современными паттернами, например, MVVM, Repository, UnitOfWork, проектирование базы данных и реализация их в СУБД MS SQL Server 2019.

Был получен практический опыт в создании базы данных методом Code First, работы с удалённой базой данных.

**ПРИЛОЖЕНИЕ А**

****

**ПРИЛОЖЕНИЕ Б**

Листинг кода страницы истории доходов и расходов

public class BalanceViewModel : BaseViewModel

{

public ICommand DeleteCommand { get; set; }

public Account \_currentAccount { get; set; }

public Account CurrentAccount

{

get { return \_currentAccount; }

set

{

\_currentAccount = value;

OnPropertyChanged(nameof(CurrentAccount));

}

}

public static ICommand LinkToEditCommand { get; set; }

public ICommand UpdateViewCommand { get; set; }

private ICollectionView \_historyCollectionView { get; set; }

public ICollectionView HistoryCollectionView

{

get { return \_historyCollectionView; }

set

{

\_historyCollectionView = value;

OnPropertyChanged(nameof(HistoryCollectionView));

}

}

private List<History> \_histories { get; set; }

public List<History> Histories

{

get { return \_histories; }

set

{

\_histories = value;

OnPropertyChanged(nameof(Histories));

if(HistoryCollectionView != null)

RefreshHistoryCollectionView();

}

}

public double \_balance { get; set; }

public double Balance

{

get { return \_balance; }

set

{

\_balance = value;

OnPropertyChanged(nameof(Balance));

}

}

private HistoryRepository historyRepository;

public BalanceViewModel()

{

SingleCurrentAccount currentAccount = SingleCurrentAccount.GetInstance();

CurrentAccount = currentAccount.Account;

historyRepository = new HistoryRepository();

Histories = new List<History>();

GetHistories();

CreateCommand.CheckRepeatHistories();

Balance = GetBalance(Histories);

UpdateViewCommand = new UpdateViewCommand(MainWindow.MainView);

RefreshHistoryCollectionView();

DeleteCommand = new DeleteCommand(this);

LinkToEditCommand = new LinkToEditCommand();

}

public override void GetHistories()

{

Histories = historyRepository.List(x => x.Account.Id == CurrentAccount.Id && !x.IsRepeat).ToList();

Balance = GetBalance(Histories);

}

public static double GetBalance(List<History> histories)

{

double SumOfHistories = SingleCurrentAccount.GetInstance().Account.Balance;

foreach (History history in histories)

{

if (history.Activity.ActivityType.Title == "Доходы")

SumOfHistories += history.Amount;

else if (history.Activity.ActivityType.Title == "Расходы")

SumOfHistories -= history.Amount;

}

return SumOfHistories;

}

public void RefreshHistoryCollectionView()

{

HistoryCollectionView = CollectionViewSource.GetDefaultView(Histories);

HistoryCollectionView.GroupDescriptions.Add(new PropertyGroupDescription(nameof(History.Date), new DateTimeConverter()));

HistoryCollectionView.Refresh();

}

}