МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Учреждение образования «БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ

ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет Информационных Технологий

Кафедра Программной инженерии

Специальность 1-40 01 01 Программное обеспечение информационных технологий

Специализация Программирование интернет-приложений

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**К КУРСОВОМУ ПРОЕКТУ НА ТЕМУ:**

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

Тема Программное средство «Учет финансов»

Исполнитель

студент 2 курса 4 группы Селицкий Данил Евгеньевич

(Ф.И.О.)

Руководитель работы ассистент Рауба А. А.

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

Курсовой проект защищен с оценкой

Председатель ассистент Рауба А. А.

(подпись)

Минск 2020

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc41169577)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc41169578)

[2 Анализ требований к программному средству и разработка функциональных требований 9](#_Toc41169579)

[2.1 Описание функциональности ПС 9](#_Toc41169580)

[2.2 Диаграмма вариантов использования 9](#_Toc41169581)

[2.3 Спецификация функциональных требований 11](#_Toc41169582)

[2.4 Вариант использования 11](#_Toc41169583)

[3 Проектирование архитектуры проекта 12](#_Toc41169584)

[3.1 Общая архитектура приложения 12](#_Toc41169585)

[3.2 Диаграмма классов 13](#_Toc41169586)

[3.3 Проектирование логической структуры базы данных 14](#_Toc41169587)

[3.4 Проектирование окон и страниц 17](#_Toc41169588)

[3.5 Диаграмма размещения 19](#_Toc41169589)

[3.6 Диаграмма последовательностей 20](#_Toc41169590)

[4 Проектирование программного средства 21](#_Toc41169591)

[4.1 Работа с базой данных 21](#_Toc41169592)

[4.2 Шифрование паролей 22](#_Toc41169593)

[4.3 Используемые сервисы 23](#_Toc41169594)

[4.3.1 Сервис отправки сообщений на почту 23](#_Toc41169595)

[4.3.2 Сервес генирации случайной строки 23](#_Toc41169596)

[4.4 Бизнес логика приложения 25](#_Toc41169597)

[4.4.1 Пересчет данных 25](#_Toc41169598)

[Приложение А 25](#_Toc41169599)

[Приложение Б 32](#_Toc41169600)

[Приложение В 33](#_Toc41169601)

ВВЕДЕНИЕ

В наше время у всех людей есть много планов и идей, которые требует денежных средств. Однако сложно контролировать свои деньги. Каждый день нас провоцируют тратить на вещи которые нам не нужны. Решение этой проблемы может быть найдено с помощью программного средства, позволяющего вам более тонко контролировать финансы. Возможность увидеть статистику позволит удержаться от бесполезных и бездумных трат.

Приложение EconoMiC – служебный инструмент пользовательского интерфейса для учета финансов и помощи в достижении целей. С помощью данного инструмента можно ставить цели, следить за прогрессом, получить статистику.

Темой курсового проекта является разработка программного средства «Учет финансов(EconoMiC)». Данное ПО должно позволить пользователям лучше контролировать свои финансы.

Целью курсового проекта является проектирование и реализация программного средства для контроля и отслеживания денежных средств.

При использовании данного приложения, каждый пользователь компьютера сможет увидеть статистику использования своих активов. Язык разработки проекта – C#. При выполнении курсового проекта будут использованы принципы и приемы ООП. Также будут использоваться технологии Windows Presentation Foundation(WPF

1. Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству

Важным этапом в разработке программного продукта является аналитический обзор прототипов и литературных источников.

На сегодняшний день можно встретить достаточное количество доступных программных решений, разработанных для управления расходами. Рассмотрим наиболее популярные программы для учета финансов:

«CoinKeeper» ¾ это ПО для учета расходов и доходов. Есть бесплатная версия с ограниченным набором возможностей, и платная, с полным функционалом и без каких-либо ограничений по количеству категорий.

Проанализировав программу, можно выделить её основные плюсы и минусы.

Основные плюсы:

* данные хранятся в облаке. Следовательно переход на другое устройство не будет болезненным;
* есть браузерная и мобильная версия;
* все денежные передвижения делятся на 4 группы.

Основные минусы:

* слишком много функционала который в котором трудно ориентироваться;
* нет авторизации. Данные хранятся на устройстве;
* статистика и достижение целей доступно только в платной версии;
* из-за большого количества категорий статистика получается чересчур разнонаправленной.

Интерфейс программы представлен на рисунке 1.1.



Рис. 1.1 – «CoinKeeper»

«Monefy» ¾ так же ПО для учета финансов. Так же имеется бесплатная и платная версии.

Основные плюсы:

* есть возможность ставить иконки для категорий;
* более простое и понятное приложение в сравнении с «CoinKeeper»;
* можно изменять записи.

Основные минусы:

* основной функционал доступно только в платной версии;
* из-за большого количества категорий статистика получается слишком разнонаправленной.

Интерфейс программы представлен на рисунке 1.2.

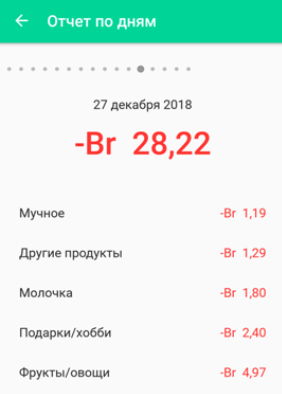


Рис. 1.2 – «Monefy»

«Дзен-мани: учет расходов»

Основные плюсы:

* привязка к картам и смс оповещениям;
* предложения от партнеров.

Основные минусы:

* в бесплатной версии очень мало функционала;
* скучный дизайн.

Интерфейс программы представлен на рисунке 1.3.

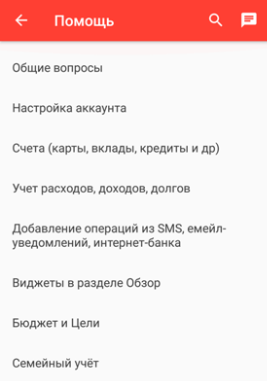


Рис. 1.3 – «Дзен-мани: учет расходов»

«PingFin»

Основные плюсы:

* привязка к Гугл аккаунту;
* может читать СМС из банков и обрабатывать эту информацию.

Основные минусы:

* есть только платная версия;
* приложение сложновато в использовании, имеет не всегда логически понятные цепочки действий и немного перегружено информацией.

Интерфейс программы представлен на рисунке 1.4.

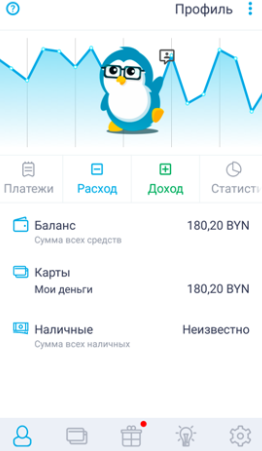


Рис. 1.4 – «PingFin»

«Домашняя бухгалтерия»

Основные плюсы:

* есть возможность сохранять резервную копию в Гугл диск;
* много возможностей в бесплатной версии;
* понятный интерфейс.

Основные минусы:

* слабая визуальная составляющая;
* сложная для понимания отображение статистики.

Интерфейс программы представлен на рисунке 1.5.

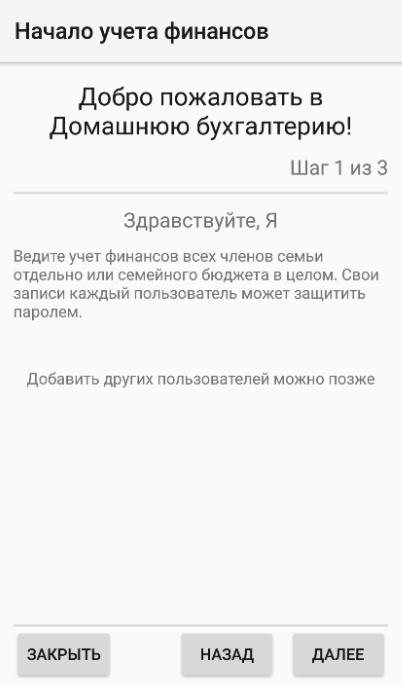
* 

Рис. 1.5 – «Домашняя бухгалтерия»»

На основе проведенного анализа и с учетом требований, указанных в задании на курсовое проектирование, сформированы следующие требования:

* разработать бизнес логику приложения;
* реализовать хранение информации на удаленном сервере;
* реализовать вычисление и отображение статистики;
* реализовать авторизацию пользователей.

Для разработки данного ПС были выбраны следующие средства разработки:

* интегрированная среда разработки Microsoft Visual Studio 2017;
* программная платформа .NET Framework 4.7.1;
* язык программирования C#;
* технология Windows Presentation Foundation (WPF);
* библиотека стилей графического дизайна Material Design;
* расширяемый язык разметки XAML;
* технология ADO.NET Entity Framework 6.

1. Анализ требований к программному средству и разработка функциональных требований

## 2.1 Описание функциональности ПС

Основные возможности:

* возможность авторизации и регистрации;
* возможность создания целей на основе даты;
* возможность создания целей на основе месячных расходов;
* возможность редактирования, удаления целей;
* возможность создания, редактирования, удаления категорий;
* возможность создания, редактирования, удаления доходов;
* возможность создания, редактирования, удаления транзакций;
* возможность просмотра статистики по расходам/доходам;
* возможность просмотра статистики по целям;
* возможность смены пароля из личного аккаунта;
* возможность сохранения записей;
* возможность смены аккаунта, без закрытия программы;
* возможность корректировки баланса;
* возможность отмены удаления;
* восстановление пароля через email.

## Диаграмма вариантов использования

Диаграмма вариантов использования (сценариев поведения, прецедентов) является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношений между ними. При построении диаграммы могут использоваться также общие элементы нотации: примечания и механизмы расширения.

Суть данной диаграммы состоит в следующем: проектируемая система представляется в виде множества актеров, взаимодействующих с системой с помощью так называемых вариантов использования. При этом актером называется любой объект, субъект или система, взаимодействующая с моделируемой системой извне. В свою очередь вариант использования – это спецификация сервисов, которые система предоставляет актеру. Другими словами, каждый вариант использования определяет некоторый набор действий, совершаемых системой при взаимодействии с актером. При этом в модели никак не отражается то, каким образом будет реализован этот набор действий.

Диаграмма вариантов использования представлена в приложении А.

В приложении предусмотрена одна роль – пользователь.

После входа в аккаунт пользователь попадает на главное окно.

На главном окне расположены навигационные кнопки для перехода во все части программы. При входе У пользователя открывается домашняя страница. Так же на главном окне есть кнопка обновления информации и поле, где он может увидеть текущий баланс.

На странице записей пользователь может разрешить или запретить редактирование записи, а также изменить ее.

На странице настроек у пользователя есть возможность сменить пароль, отредактировать текущий баланс, указать почту для восстановления пароля, а также сменить пользователя.

На домашней странице он может ознакомиться с основными советами по управлению финансами, а также увидеть цели, на которые уже собрана полная сумма. Внизу страницы графически изображен прогресс по текущей цели с самой большой итоговой суммой.

На окне целей пользователь может просмотреть все типы целей (активные, приостановленные, завершенные). А также создать новую цель. Цель можно будет изменить или удалить. Есть возможность отменить удаление цели.

На окне категорий пользователь может просмотреть текущие категории, создать новую категорию. А также можно будет изменить или удалить ее. Есть возможность отменить удаление категории.

На окне доходов пользователь может просмотреть текущие доходы, а также создать, изменить или удалить его. Есть возможность отменить удаление дохода.

На окне транзакции пользователь может просмотреть текущие транзакции, а также создать, изменить или удалить их. Есть возможность отменить удаление транзакции.

На окне статистики пользователь может узнать аналитику своих расходов и доходов. Увидеть их отношение. Так же узнать количество уже завершенных целей. И соотношение самых больших целей.

## 2.3 Спецификация функциональных требований

Спецификация функциональных требований:

1. Авторизация должна быть безопасна для пользователей, т.е. пароли будут храниться в БД в зашифрованном виде (Алгоритм шифрования SHA1);
2. Функции, работающие с БД, не должны блокировать пользовательский интерфейс при отработке. Для этого требуется сделать их асинхронными;
3. Восстановление паролей по Email будет нуждаться в секретной информации (логин и пароль для smtp сервера), поэтому данная информация будет храниться в конфигурационных файлах.
4. При входе пользователя приложение должно пересчитать большое количество данных. Эти вычисления не должны блокировать пользовательский интерфейс при отработке. Следовательно эти вычисления следует выполнять в отдельном потоке.
5. Пользователь может вводить неверные данные. Из-за этого вычисления будут неверными. Существует необходимость в валидации введенных данных.

## 2.4 Вариант использования

У человека появилось желание купить какую-нибудь дорогую вещь или просто отложить деньги на отпуск. Он создает новую цель, где указывает сколько он хочет отложить финансов. Программа просчитывает и отображает, когда человек накопит нужную сумму. Так же программа будет графически отображать прогресс накопления, что может послужить дополнительной мотивацией.

1. Проектирование архитектуры проекта

## 3.1 Общая архитектура приложения

Паттерн MVVM (Model-View-ViewModel) позволяет отделить логику приложения от визуальной части (представления). Данный паттерн является архитектурным, то есть он задает общую архитектуру приложения.

Классическая модель MVVM (рисунке 3.1) состоит из трех компонентов: модели (Model), модели представления (ViewModel) и представления (View).



Рис. 3.1 – «классическая модель MVVM»

Из-за большого количества выполняемых функций, которые в классической модели должны располагаться во ViewModel было принято решение о создании вспомогательных классов, назначение которых хранить большие функции и производить сложные вычисления. Эти классы я решил назвать контроллерами, потому что их поведение схоже с поведением компонента контроллер в архитектурном паттерне MVC. А именно принимают параметры с компонента View и возвращает состояние. Но в моем случае входящие параметры предварительно обрабатываются еще во ViewModel.

В конечном итоге мой вариант архитектуры представлен на рисунке 3.2.

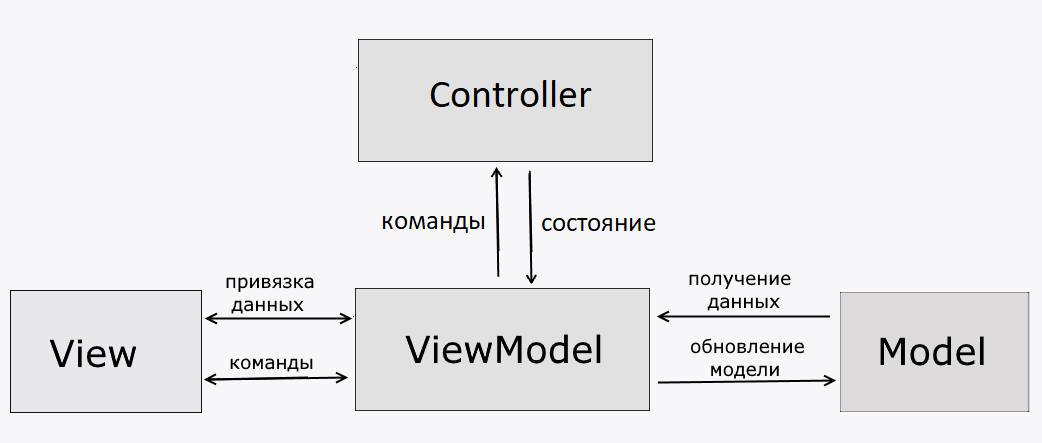


Рис. 3.2 – «моя расширенная модель MVVM»

ViewModel получает команды и переменные от View и подготавливают их для передачи в Controller. Контроллер производит вычисления и отправляет измененные параметры в ViewModel. Далее ViewModel обновляет модели и изменяет View в соответствии с состоянием.

## Диаграмма классов

Диаграмма классов определяет типы классов системы и различного рода статические связи, которые существуют между ними.

Диаграмма классов представлена в приложении Б.

Основной класс – это класс User(Пользователь).

Свойства класса User:

* login - имя пользователя;
* password - пароль пользователя в зашифрованном виде;
* total money - общее количество средств на балансе;
* free money - средства, которые не определены;
* last activity - последняя дата посещения пользователя;
* active targets - список активных целей (на которые пользователь откладывает средства;
* suspended targets - список приостановленных целей (на которые пользователь временно не откладывает средства;
* old targets - список завершенных целей (на которые пользователь уже собрал полную сумму;
* categories - список категорий (постоянные расходы, которые вычитаются каждый месяц);
* incomes - список доходов (постоянные ежемесячные доходы);
* transactions - список транзакций (один перевод средств);
* node - записка (личный блокнот).

Классы ActiveTarget, SuspendedTarget, OldTarget наследуются от класса Target.

Свойства класса Target:

* targetname - название цели;
* totalsum - полная стоимость цели;
* currentsum - текущие средства, отложенные для этой цели;
* spend - ежемесячный расход на данную цель;
* targetTime - время, к которому цель будет выполнена.

Класс Сategory:

* categoryname - название категории;
* spend - ежемесячный расход на данную категорию.

Класс Income:

* incomename - название дохода;
* money - ежемесячный доход;
* incomeTime – дата дохода.

Класс Transaction:

* transactionname - название транзакции;
* money – сумма транзакции;
* transactionDate – дата транзакции.

Класс Note:

* note - текст блокнота;

## 3.3 Проектирование логической структуры базы данных

Для реализации поставленной задачи была создана база данных. Для создания диаграммы использовались система управления реляционными базами данных Microsoft SQL Server 2017.

В самом приложении используется удаленный сервер базы данных Azure Server. Все данные хранятся удаленно.

База данных – совокупность данных, организованных по определенным правилам, предусматривающим общие принципы описания, хранения и манипулирования данными, независимо от прикладных программ.

Система управления базами данных – совокупность программ и языковых средств, предназначенных для управления данными в базе данных, ведения базы данных и обеспечения взаимодействия ее с прикладными программами. База данных нашей программной подсистемы состоит из 8 таблиц, ее схема изображена в приложении В

База данных спроектирована с помощью ADO.NET Entity Framework 6.

Данная диаграмма построена в соответствии с диаграммой классов из приложения Б.

Таблица Users используется для хранения данных о пользователе. Включает 9 столбцов: ID, Login, Password, Total\_money, Reserve\_Money, Last\_activity, Note\_id, Statistic\_id, Email. Столбец ID является ключевым и уникальным, в нём хранится идентификатор пользователя. Столбец Login хранит в себе уникальное имя пользователя. Столбец Total\_money хранит данные о полном балансе пользователя. Столбец Password хранит пароль в зашифрованном виде. Столбец Reserve\_Money содержит количество занят средств (зарезервированы под цели). Столбец Last\_activity содержит дату, когда пользователь последний раз взаимодействовал с программой. Столбец Note\_id является вторичным ключом к таблице Notes. Столбец Statistic\_id является вторичным ключом к таблице Statistics. Столбец Email хранит почту пользователя.

Таблица Targets используется для хранения данных о целях пользователя. Включает 10 столбцов: ID, TargetName, Spend, TotalSum, CurrentSum, TargetTime, Discriminator, User\_id, User\_id1, User\_id2. Столбец ID является ключевым и уникальным, в нём хранится идентификатор цели. Столбец TargetName хранит в себе название цели. Столбец Spend хранит данные о ежемесячных платах, относящихся к этой цели. Столбец TotalSum хранит полную стоимость цели. Столбец CurrentSum содержит количество собранных средств на данный момент. Столбец TargetTime содержит предполагаемую дату, когда средства на цель будут собраны. Столбец Discriminator хранит данные, которые определяют к какому типу цели относиться данная цель (активная, приостановленная, завершенная). Столбец User\_id является вторичным ключом к таблице Users и обозначает принадлежность текущей задачи к активным. Столбец User\_id1 является вторичным ключом к таблице Users и обозначает принадлежность текущей задачи к приостановленным. Столбец User\_id2 является вторичным ключом к таблице Users и обозначает принадлежность текущей задачи к завершенным.

Таблица Incomes используется для хранения данных о целях пользователя. Включает 5 столбцов: ID, IncomeName, Money, Date, User\_id. Столбец ID является ключевым и уникальным, в нём хранится идентификатор дохода. Столбец IncomeName хранит в себе название дохода. Столбец Money хранит данные о размере дохода. Столбец Date хранит дату дохода. Столбец User\_id является вторичным ключем к таблице Users.

Таблица Categories используется для хранения данных о транзакциях пользователя. Включает 4 столбцов: ID, CategoryName, Spend, User\_id. Столбец ID является ключевым и уникальным, в нём хранится идентификатор категории. Столбец CategoryName хранит в себе название категории. Столбец Spend хранит данные о размере ежемесячных тратах, связанных с этой категорией. Столбец User\_id является вторичным ключом к таблице Users.

Таблица Statistics используется для хранения статистических данных пользователя. Включает 4 столбцов: ID, CounterTargets, MoodChange, LargestTarget\_Id. Столбец ID является ключевым и уникальным, в нём хранится идентификатор статистики. Столбец CounterTargets хранит в себе количество завершенных целей. Столбец MoodChange хранит данные о количестве нажатий на кнопку со смайликом. Столбец LargestTarget\_Id является вторичным ключом к таблице Targets, и обозначает самую большую завершенную цель этого пользователя.

Таблица Notes используется для хранения записей пользователя. Включает 2 столбцов: ID, text. Столбец ID является ключевым и уникальным, в нём хранится идентификатор записки. Столбец text хранит в себе строку, которую пользователь вводит в блокноте.

Таблица Tips используется для хранения советов. Включает 3 столбцов: ID, Header, Body. Столбец ID является ключевым и уникальным, в нём хранится идентификатор совета. Столбец Header хранит в себе строку, которая будет заголовком совета. Столбец Body хранит в себе строку, которая будет телом совета.

## 3.4 Проектирование окон и страниц

В приложении всего 2 окна.

Первое окно предоставляет возможности по:

* регистрации нового пользователя;
* входа существующего пользователя;
* восстановлению пароля существующего пользователя.

Граф состояний первого окна представлен на рисунке 3.3.



Рис. 3.3 – «Граф состояний первого окна»

Начальная страница – страница входа. Со страницы входа можно перейти на страницу регистрации или на страницу восстановления пароля. Со страницы регистрации можно вернуться на страницу входа или перейти на страницу восстановления пароля. Со страницы восстановления пароля можно перейти только на страницу входа.

Второе окно предоставляет весь основной функционал:

* просмотр, изменение удаление целей;
* просмотр, изменение удаление категорий;
* просмотр, изменение удаление транзакций;
* просмотр, изменение удаление доходов;
* просмотр статистики;
* просмотр, изменение записи;
* изменение настроек

Граф состояний первого окна представлен на рисунке 3.4.



Рис. 3.4 – «Граф состояний второго окна»

Все переходы во втором окне осуществляются с главного окна. Одна из частей главного окна меняется в зависимости от выбранного контента.

Изначально открыта домашняя страница. Сразу есть возможность сменить ее на одну по выбору:

* страницу доходов;
* страницу транзакций;
* страницу статистики;
* страницу блокнота;
* страницу настроек;
* страницу категорий;
* страницу целей;

Так же со страницы доходов можно перейти на страницу добавления (изменения) доходов. Со страницы транзакций можно перейти на страницу добавления (изменения) транзакций. Со страницы категорий можно перейти на страницу добавления (изменения) категорий. Со целей транзакций можно перейти на страницу добавления (изменения) целей.

Граф переходов между окнами представлен на рисунке 3.5

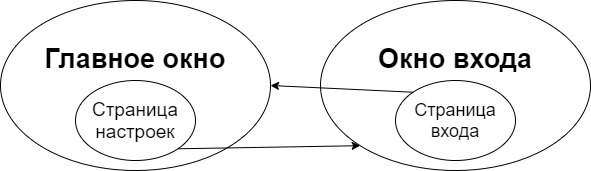


Рис. 3.5 – «Граф переходов между окнами»

Из окна входа переходит вызывается главное окно если на на странице входы были введены правильные учетные данные пользователя. Из главного окна вызывается окно входа если на странице настроек была нажата кнопка смена аккаунта.

При нажатии на кнопку закрытия окна в любом из двух окон происходит закрытие обоих окон.

## 3.5 Диаграмма размещения

Диаграмма размещения отражает физические взаимосвязи между программными и аппаратными компонентами системы. Она является хорошим средством для того, чтобы показать размещение объектов и компонентов в распределенной системе.

Так как использование приложения не подразумевает одновременную работу с несколькими клиентами был выбран бес серверный вариант архитектуры.

Диаграмма размещения представлена в приложении Г.

По диаграмме размещения видно, что приложение напрямую взаимодействует с базой данных. Осуществляется это взаимодействие через специальный класс, который контролирует доступ к базе данных. Для получения данных приложение обращается к этому классу, а класс в свою очередь формирует и производит запрос к удаленному серверу. После получения информации класс информирует остальные классы что поступили новые данные.

## 3.6 Диаграмма последовательностей

Диаграмма последовательности — диаграмма, на которой для некоторого набора объектов на единой временной оси показан жизненный цикл какого-либо определённого объекта и взаимодействие актеров в рамках какого-либо определённого прецедента.

На рисунке 3.6 изображена диаграмма последовательностей.

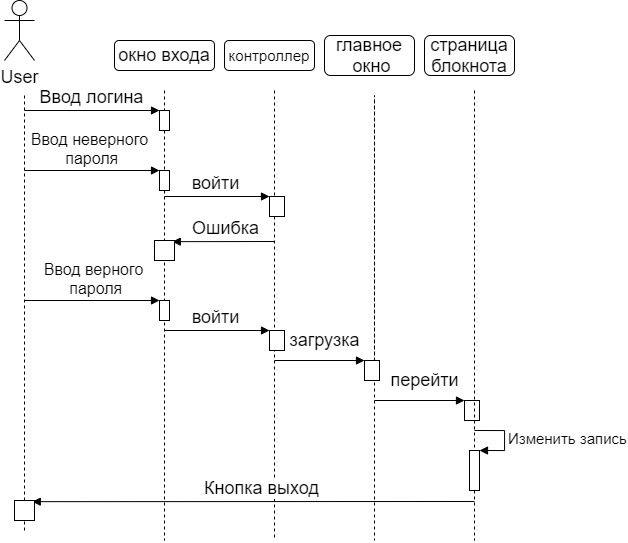


Рис. 3.6 – «Диаграмма последовательностей»

На данной диаграмме изображен случай, когда пользователь изначально ввел неправильный пароль. При проверке пароли не совпали и приложения выдало ошибку на экран входа. Пользователь ввел правильный пароль. Так как пароли совпали происходит загрузка главной страницы. Далее пользователь заходит на страницу с блокнотом и изменяет данные. Пользователь нажимает на кнопку закрыть окно. Происходит закрытие обоих окно и программа заканчивает свое выполнение.

1. Проектирование программного средства Проектирование архитектуры проекта

## 4.1 Работа с базой данных

Приложение EconiMiC работает с удаленным сервером Azure. Принцип работы такой же, как и с MS SQL SERVER. В приложении используется Entity Framework 6. Как способ взаимодействия с базой данных был выбран Code first способ.

Code first: разработчик создает класс модели данных, которые будут храниться в бд, а затем Entity Framework по этой модели генерирует базу данных и ее таблицы.

Так как была выбрана бес серверная архитектура, появилась необходимость в создании классов, которые бы производили работу с базой данных. В приложении был выделено специальное пространство имен: DataBaseLayer.

Это пространство имен содержит:

* econoMiCDBContext – контекст обращения к базе данных;
* repository - паттерн репозиторий позволяет абстрагироваться от конкретных подключений к источникам данных, с которыми работает программа, и является промежуточным звеном между классами, непосредственно взаимодействующими с данными, и остальной программой;
* unitOfWork - паттерн, который позволяет упростить работу с различными репозиториями и дает уверенность, что все репозитории будут использовать один и тот же контекст данных;
* unitOfWorkSingleton - порождающий паттерн, который гарантирует, что для определенного класса будет создан только один объект, а также предоставит к этому объекту точку доступа. Используется для класса unitOfWork.

Листинги классов пространства имен DataBaseLayer приведены в приложении Д.

В файле Repository реализован паттерн для вся всех таблиц описанных в econoMiCDBContext. Все классы в этом файле реализуют интерфейс IRepository который представлен на рисунке 4.1

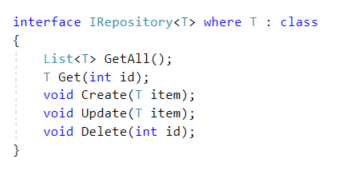


Рис. 4.1 – «интерфейс IRepository»

Реализация этого интерфейса обеспечивает что с таблицами будут существовать базовые операции CRUD. Все репозиторий реализованы по одному шаблону, который указан в приложении Д. Изменения есть только в классе UserRepository. Дополнительный метод представлен на рисунке 4.2.

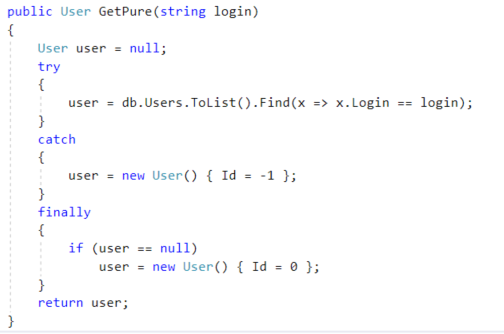


Рис. 4.2 – «Дополнительный метод»

Этот метод получает пользователя без привязанных к нему остальных классов. Если сети нет метод вернет пользователя с Id = -1, если пользователь не был найден, то метод вернет пустого пользователя с Id = 0.

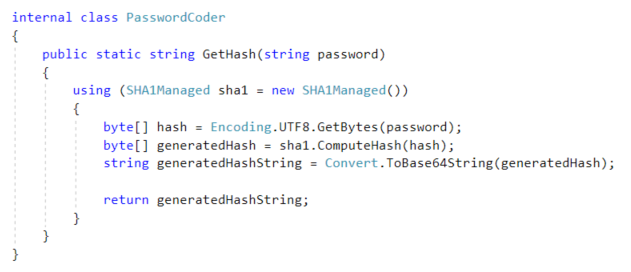
Класс UnitOfWork реализует паттерн UnitOfWork. А именно гарантирует что все репозитории будут работать с одним и тем же контекстом данных, а также упрощает работу с базой данных за счет еще одного уровня абстракции. Так же класс UnitOfWork наследует интерфейс IDisposable что позволяет при завершении работы программы эффективно очищать ресурсы.

Класс UnitOfWorkSingleton реализует паттерн Singleton. Этот класс используется для работы с базой данных во всех остальных классах.

## Шифрование паролей

Для безопасности данных пользователя в базе данных пароль имеет зашифрованный вид. При сверке паролей введенный пользователем пароль хэшируется и в зашифрованном виде сравнивается с паролем, который храниться в базе данных.

Класс который производит шифрование изображен на рисунке 4.3.

 Рис. 4.3 – «Класс PasswordCoder»

Класс PasswordCoder использует встроенные классы из пространства имен

System.Security.Cryptography. Используется метод шифрования SHA1. Secure Hash Algorithm 1 — алгоритм криптографического хеширования. Сначала строка пароля записывается массив байтов. Далее с помощью экземпляра класса SHA1Managed массив байтов шифруется. После чего массив байтов преобразуется в строку для более удобной работы с зашифрованным набором байт.

## Используемые сервисы

### Сервис отправки сообщений на почту

Листинг сервиса отправки сообщений на почту находится в приложении Е.

Функция на вход принимает 4 параметра:

* emailTo – почта, на которую будет отправлено письмо;
* title – заголовок, который будет указан в письме;
* displayName – имя, которое будет отображаться как имя отправителя;
* htmlBody – тело сообщения. Поддерживает html теги.

Логин и пароль от почты лежат в конфигурационном файле. Так же в этом сервисе используются классы из пространства имен System.Net. Класс MailAddress представляет собой структуру хранящую в себе почту и псевдоним. Класс MailMessage представляет формирует сообщение и принимает в конструктор почту отправителя и почту получателя. Класс SmtpClient реализует SMTP протокол (протокол для передачи писем на электронную почту).

### 4.3.2 Сервис генерации случайной строки

Код функции представлен на рисунке 4.4.

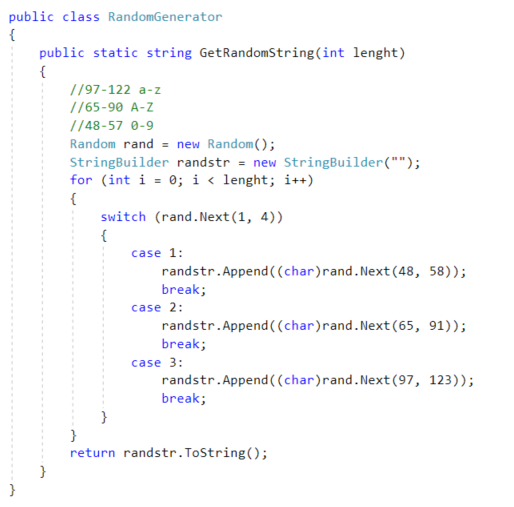


Рис. 4.4 – «Генерация случайной строки»

Функция принимает на вход длину требуемой строки.

Алгоритм получает на вход длину строки, которую необходимо сгенерировать. Далее выполняется цикл, у которого столько итераций, сколько букв необходимо сгенерировать. В теле цикла сначала генерируется число от 1 до 3. Если сгенерированное число равно 1, то к выходной строке конкатенируется число. Если сгенерированное число равно 2, то к выходной строке конкатенируется заглавная буква латинского алфавита. Если сгенерированное число равно 3, то к выходной строке конкатенируется строчная буква латинского алфавита. Определение конкатенируемого символа происходит так же случайным образом. Нужный символ берётся путем приведения целочисленного значения к символьному типу. Таким образом число — это номер символа в таблице ASCII. Так как строка может изменяться много раз, используется класс StringBuilder вместо string для уменьшения времени выполнения функции, так как строка будет не перезаписываться каждый раз при изменении, а дописываться предыдущая.

Алгоритм выполнения функции представлен в приложении Ж.

Этот класс используется при восстановлении пароля. Сгенерированная строка отправляется пользователю в письме. А также записывается в зашифрованном виде в базу данных.

## Бизнес логика приложения

### Пересчет данных

Пересчетом данных в приложении занимается класс Calculator находящийся в пространстве имен Economic\_v2.Logic. Листинг класса Calculator представлен в приложении И.

При запуске приложения существует необходимость пересчитать текущий баланс и цели, а также обновить статистику. Так же пересчет баланса и обновление статистики должно происходить при добавлении(обновлении) целей, категорий, доходов и при добавлении транзакции. Так как вычисления могут занять много времени, они будут выполняться в разных потоках, а позже синхронизироваться. Пересчет запускается методом Calculate. В основном потоке сначала высчитывается разница во времени между текущей датой и датой последней активности. Далее в отдельном потоке запускается расчёт текущего баланса по ежемесячным доходам и по расходом связанными с категориями. Для вычисления расходов по категориям высчитывается количество месяцев между текущим месяцем и месяцем последней активности пользователя. Потом перебираются все категории, и количество месяцев умножается на месячные расходы категории. Расчёт доходов производиться с помощью маркера. Так как число поступления дохода может быть 31, а не во всех месяцах есть такое число, появилась необходимость в маркере, который будет корректировать число поступления дохода. Алгоритм установки маркера изображен в приложении К.

В отдельном потоке вычисляются затраты на цели. Выполняет вычисления метод CalculateTargets. В качестве параметра метод принимает ссылку на пользователя. Средства на цели высчитываться так же, как и на категории с единственным отличием, а именно происходит проверка на достижение цели. Если цель достигнута, то она автоматически переводиться в достриженные. Так же общая сумма затрат на все цели не отнимается от общего количества финансов, а записывается в поле у класса User как зарезервированные средства. Так же, если ежемесячная плата будет больше, чем разница общей цены и уже собранных средств, то будет отниматься только необходимое количество средств.

### Работа с целями

Для работы с целями был разработан дополнительный класс TargetCalcController. Листинги этого класса находятся в приложении Л. Первый метод CalcTarget помогает методу CalculateTargets рассчитывать необходимую сумму для резервирования. Второй метод CalculateTargetWithSpend предназначен для создания цели с указанным месячным расходом. Метод просчитывает дату, когда цель будет достигнута и возвращает построенную цель. Третий метод CalculateTargetWithTime предназначен так же для постройки цели, но уже на основе желаемой даты получения цели. Метод рассчитывает сколько должен тратить пользователь, чтобы достичь цели к желаемой дате.

### Контроллеры

Так как у некоторых страниц есть 2-3 подстраницы, но они работают с одними и теми же данными, было принято решения использовать контроллеры которые бы управляли всеми ViewModel с этих страниц. Такими страницами являются страница целей, страница категорий, страница доходов и страница транзакций. Все страницы построены по одному принципу. Листинг контролера страницы целей представлен в приложении Л.

Контролеры содержат общие обьекты и общие методы которые могут вызываться из любого ViewModel относящегося к этой страницы. Controller меняет обьекты и оповещает об этом ViewModel. ViewModel в свою очередь изменяет View.

## Другие используемые паттерны

### Паттерн Builder

Cтроитель (Builder) - шаблон проектирования, который инкапсулирует создание объекта и позволяет разделить его на различные этапы.

В приложении паттерн строитель применяеться для класса Target. Код строителя приведен в приложени Н. Использование паттерна обусловлено тем что именно класс Target создается не сразу, а динамически вычисляются его свойства в зависимости от данных введенных пользователем. Таким образом строитель обеспечивает гипкость в создании экземпляра класса Target.

### Паттерн Command

Паттерн Command позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. Этот объект запроса на действие и называется командой. При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие.

Команды могут использовать параметры, которые передают ассоциированную с командой информацию. Кроме того, команды могут ставиться в очередь и также могут быть отменены.

Листинги кода паттерна Command представленны в приложении П.

1. Тестирование, проверка работоспособности и анализ полученных результатов

## 5.1 Тестирование авторизации

Так как приложение обращается к удаленному серверу необходимо обработать исключение когда нет подключения к интернету. Приложение 15 екунд пытается подключится к серверу, если соеденение не было установлено то выдается ошибка показанная на рисунке 5.1.Если подключение произошло успешно но пользователь не был найден то появиться ошибка изображенная на рисунке 5.2.

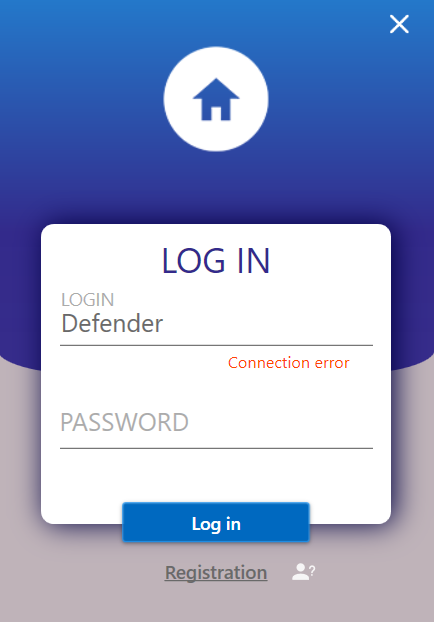
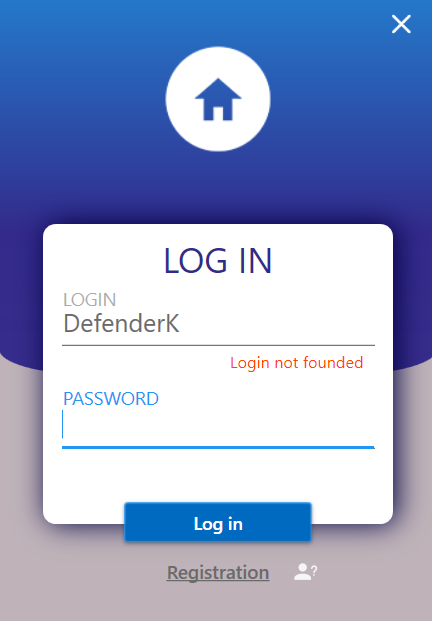
 

Рис. 5.1 – «Нет соединения» Рис. 5.2 – «Пользователь не найден»

Так же ошибки могут возникнуть когда пользователь правильно ввел логин но неправильно ввел пароль. Пример такой ошибки изображен на рисунке 5.3. Так же пользователь может забыть ввести пароль и тогда появиться ошибка изображенная на рисунке 5.4. Если пользователь первым символом логина введет пробел, то появится ошибка изображенная на рисунке 5.5. Если логин или пароль будет длинной менее 2-х символов то пользователь увидит ошибку изображенную на рисунунке 5.6

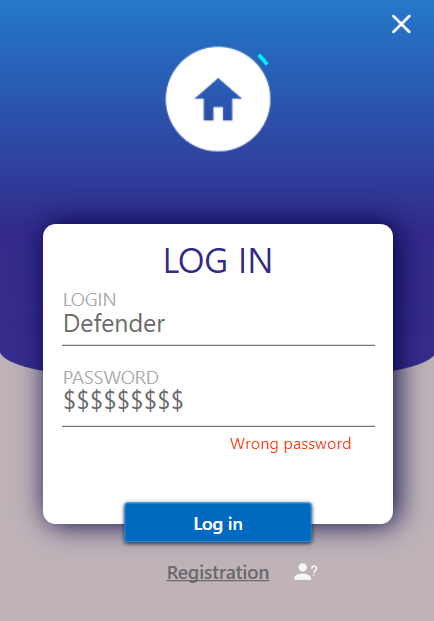
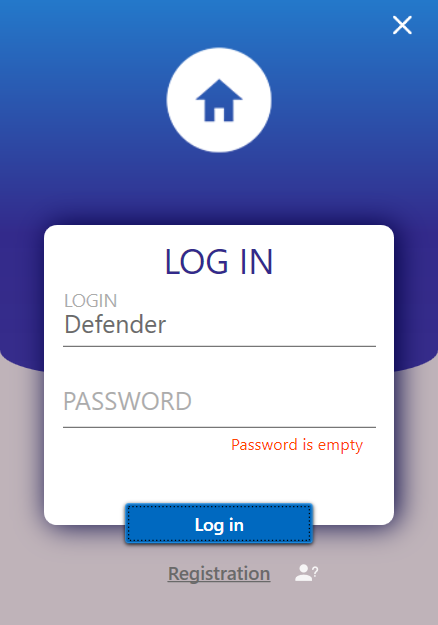
 

Рис. 5.3 – «Неверный пароль» Рис. 5.4 – «Пустой пароль»

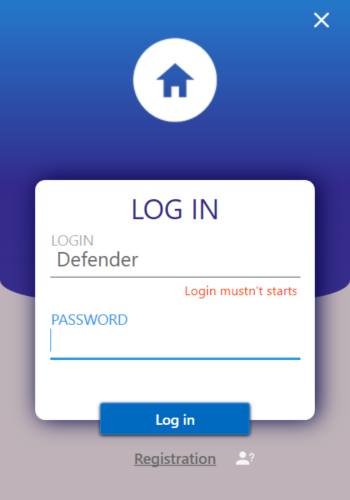
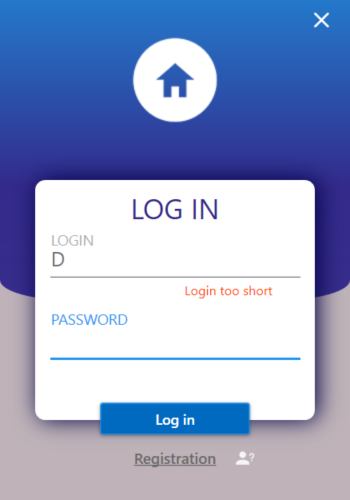
 

Рис. 5.5 – «Логин начинается с ‘ ’» Рис. 5.6 – «Пустой пароль»

На странице востановления пароля, если пользователь попытается восстановить пароль к аккаунту который не привязан ни к какой почте то он увидит ошибку изображенную на рисунке 5.7.

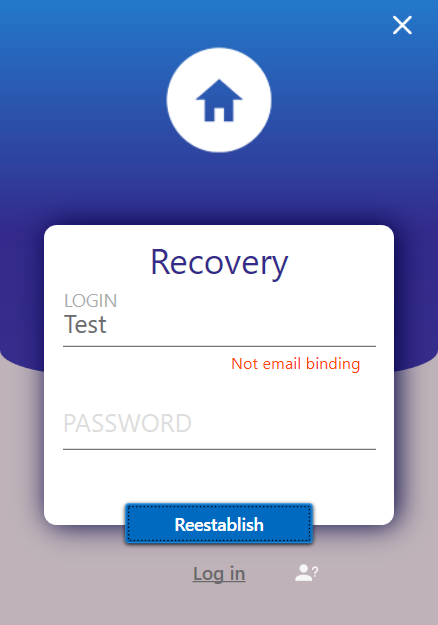


Рис. 5.7 – «Непривязанный аккаунт»

Так же ошибка может возникать если пользователь попробует создать аккаунт с уже существующим логином. Пример такой ошибки изображен на рисунке 5.8.

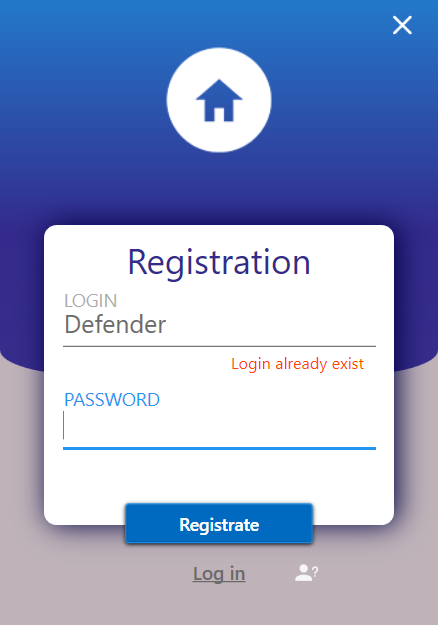


Рис. 5.8 – «Логин занят»

## 5.2 Тестирование основного окна

Пользователь может получить ошибку там где он пытается ввсести данные. Начнем со страницы настроек. Пользователь неверно указывает текущий баланс. После нажатия на кнопку подтвердить он получит ошибку. Ситуация смоделирована на рисунке 5.9. В результате пользователь получит ошибку изображенную на рисунке 5.10.

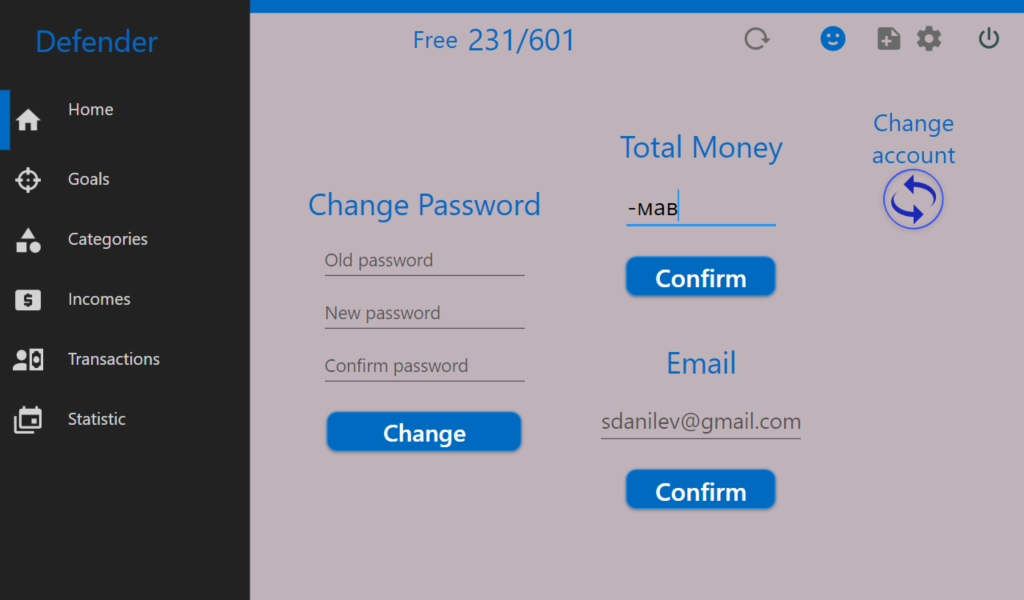


Рис. 5.9 – «Ввод неверного значения»

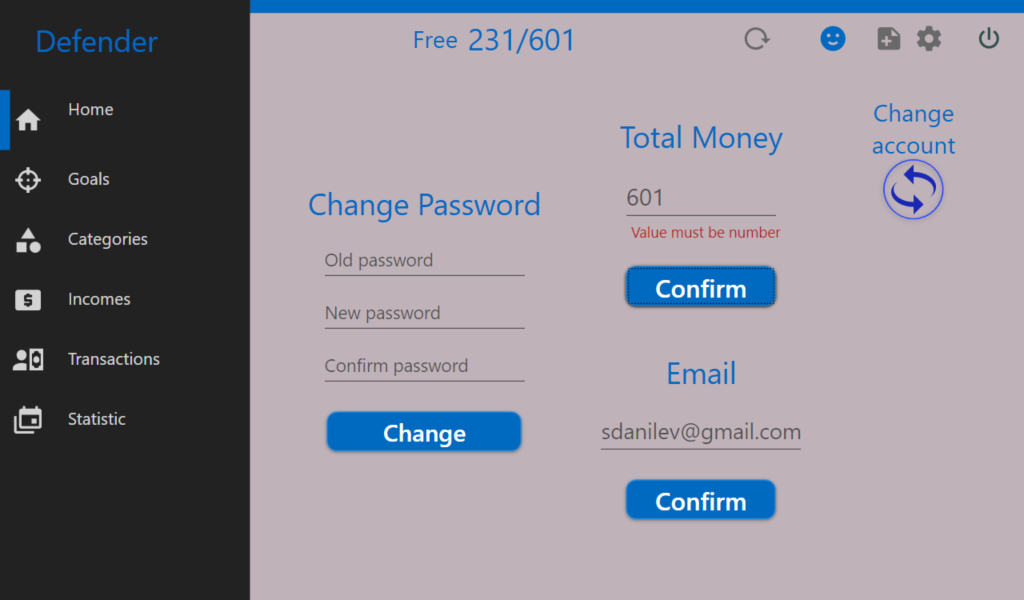


Рис. 5.10 – «Вывод ошибки»

Так же пользователь получит изключение если он будет вводить неверный пароль в поле восстановления пароля. А так же если новый пароль и пароль подтверждения не будет совпадать. Еще одну ошибку на этой странице пользователь получит если попробует ввести неправильную почту. Такая ситуация смоделирована на русинке 5.11. Сообщение об ошибке показано на рисунке 5.12.

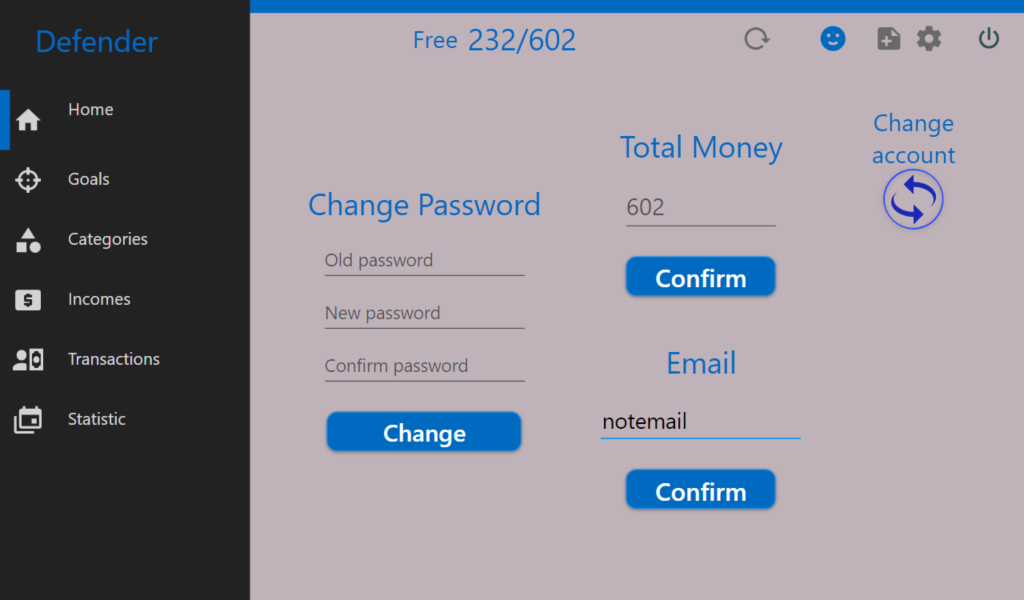


Рис. 5.11 – «Неверная почта»

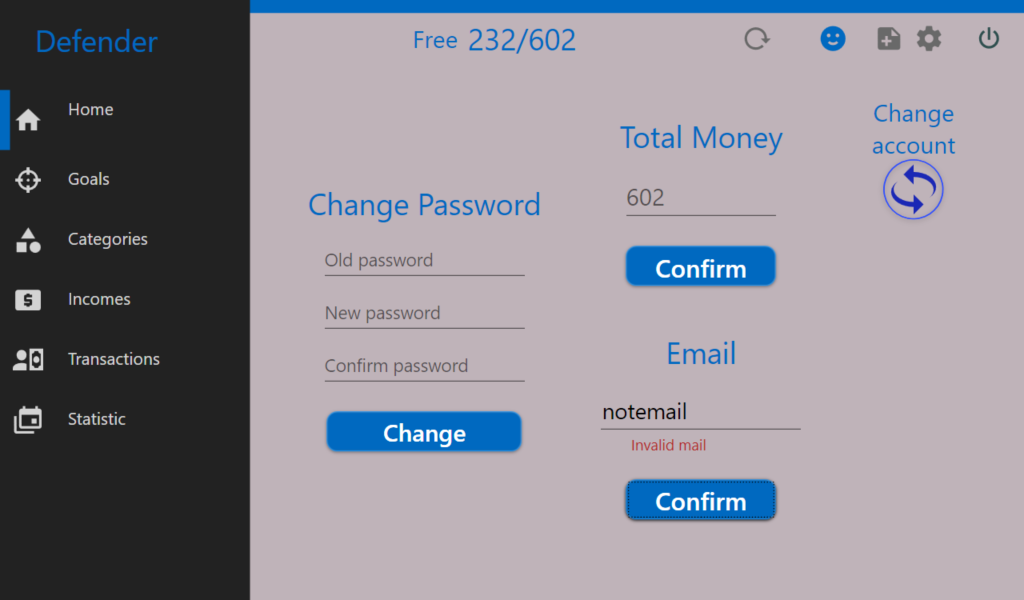


Рис. 5.11 – «Сообщение об ошибке»

Так же пользователь может получить ошибки при добавлении или изменении целей, категорий, доходов или транзакций. Рассмотрим основные ошибки на примере страницы добавления цели. При загрузке страницы пользователю будут выданы сообщения о вводе необходимых значений. После открытия страницы добавления новой цели пользователь видит окно содержимое которого изображено на рисунке 5.12. Имя должно быть заполненно. Общая сумма должна быть не отрицательна. Текущая сумма не должна быть отрицательна и должна быть меньше общей суммы. Месячные зараты должны быть не отприцательны и меньше общей суммы. Желаемая дата должны быть хотя бы на следующий месяц. Все эти ошибки показаны на рисунке 5.13.

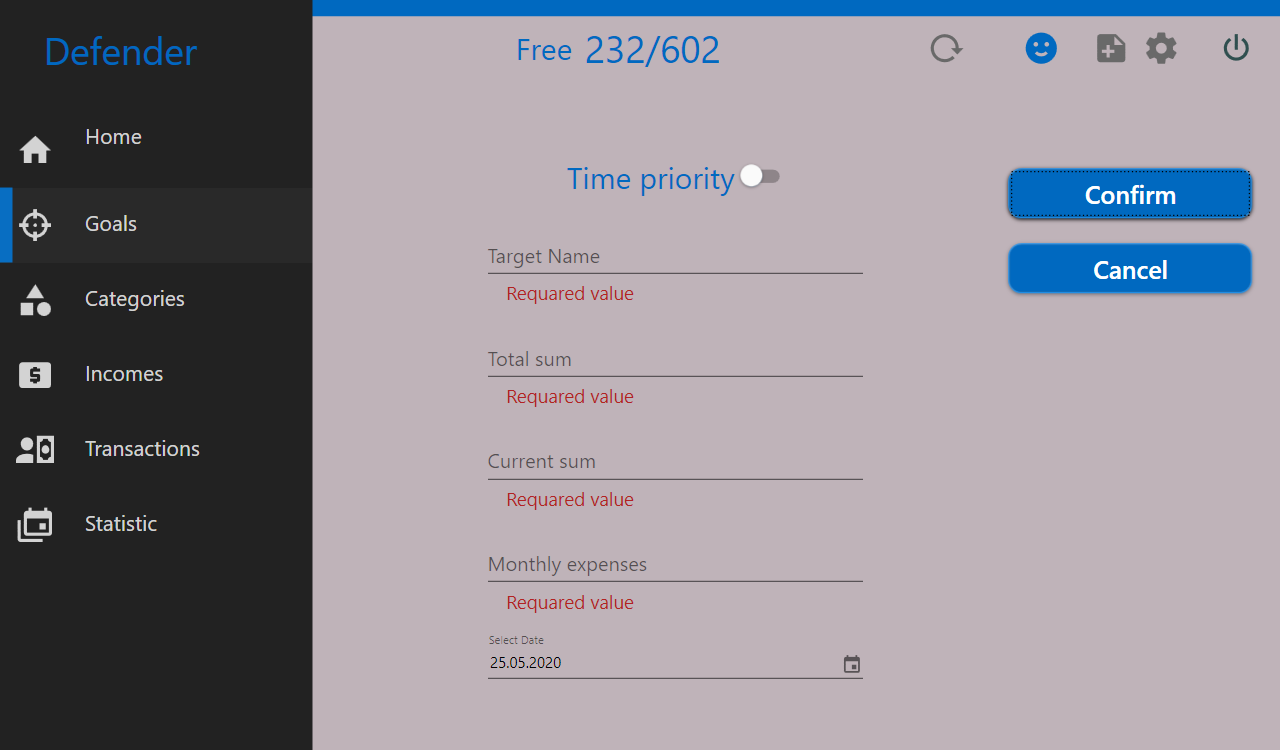


Рис. 5.12 – «Требуемые значения»

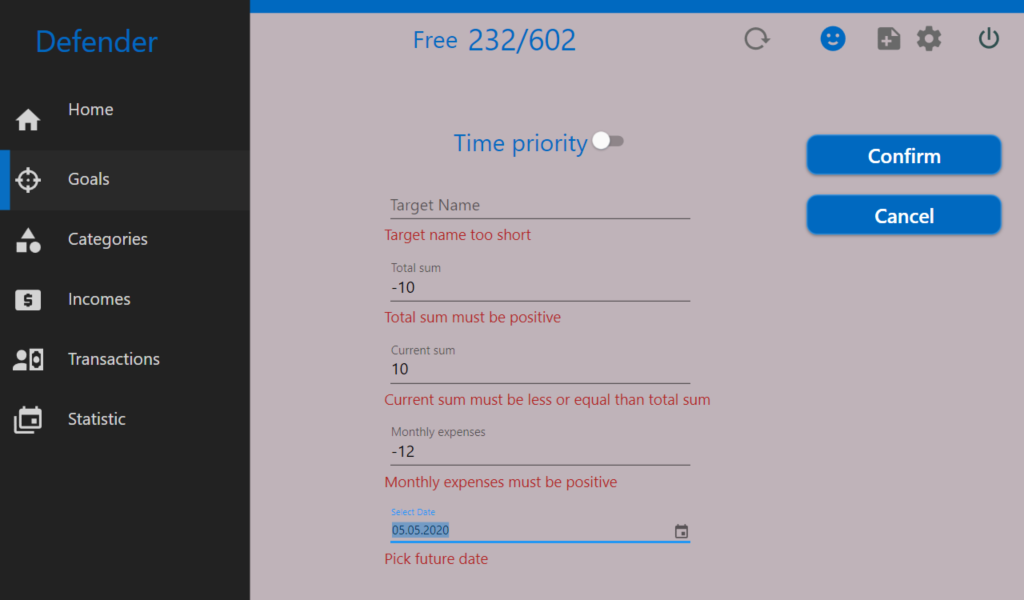


Рис. 5.13 – «Основные ошибки при вводе данных»

1. Руководство по установке и использованию

## 6.1 Установка приложения

Для установки програмного обеспечения скачайте архив с GitHub <https://github.com/SelDanilEv/EconoMiC_installer>.

Распакуйте его. После распоковки запустите файл setup.exe (рисунок 6.1)

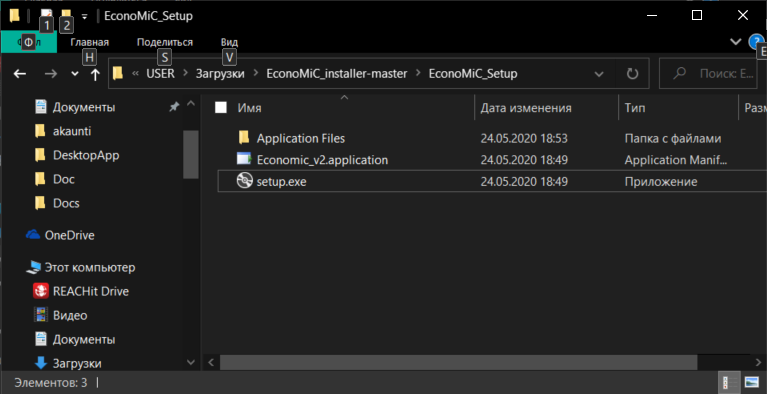


Рис. 6.1 – «Распакованный архив»

В диалоговом окне нажмите кнопку установить (рисунок6.2).

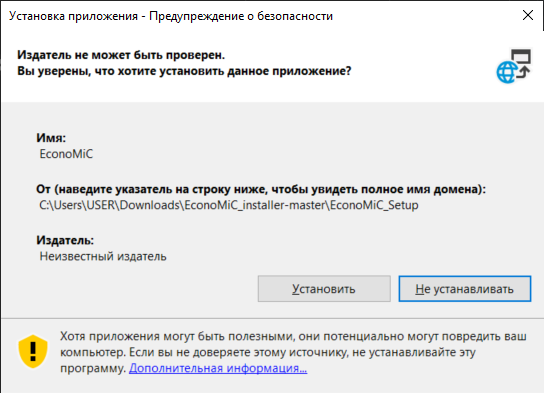


Рис. 6.2 – «Диалоговое окно»

Далее установка пройдет в автоматическом режиме. Ссылка на приложение будет добавлено в меню Windows. А так же сам файл программы можно быдет найти по пути:

C:\Users\USER\AppData\Roaming\Microsoft\Windows\Start\Menu\Programs\ DefenderSD\EconoMic

## 6.2 Использование приложения.

Приложение требует стабильное подключение к интернету. В противном случае вы либо не сможете войти в аккаунт лиюо выши данные не сохраняться.

При запуске приложения вы попадете на окно входа. Если у вас еще нет аккаунта то перейдите во вкладку регистрации. Зарегестрируйтесь. После регистрации автоматически включиться страница входа. Если введенные данные были правильными, то после нажатия кнопки входа откроется главное окно.

Основные элементы главного окна показаны на рисунке 6.3.

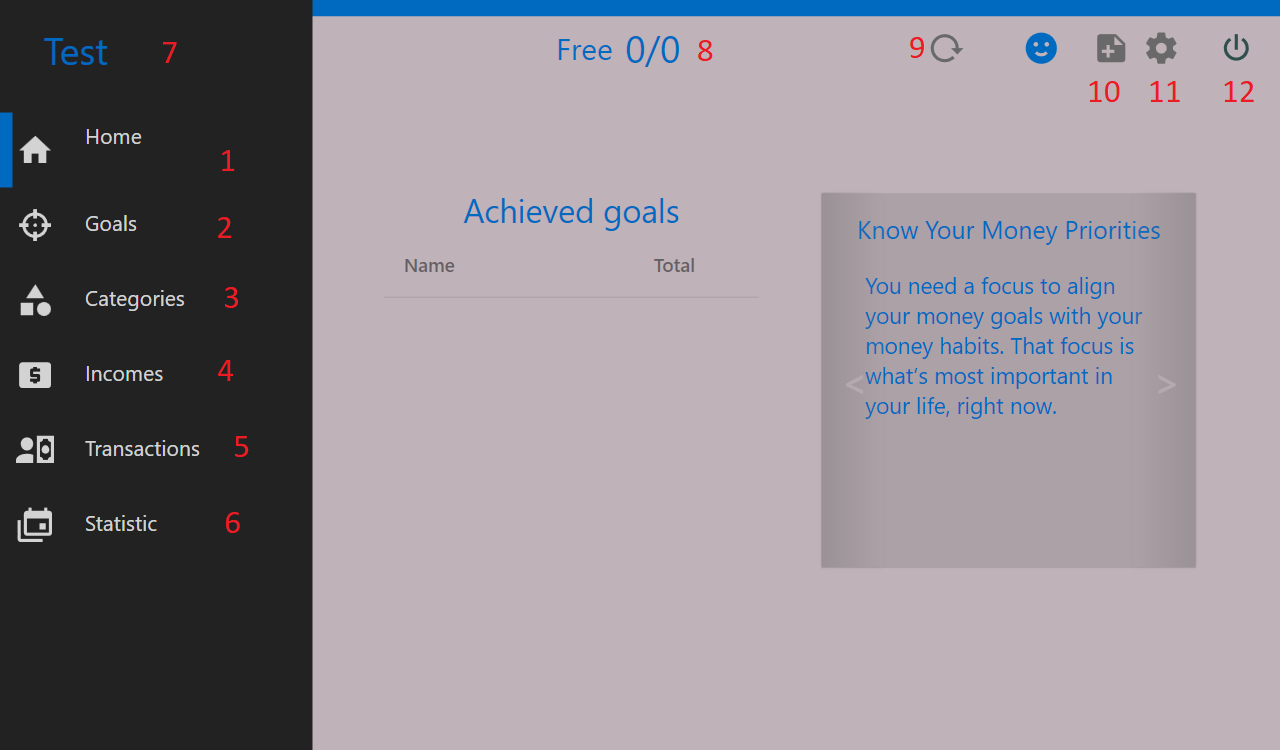


Рис. 6.3 – «Основные элементы»

Обозначения на рисунке 6.3:

1. Домашняя страница;
2. Страница целей;
3. Страница категорий;
4. Страница доходов;
5. Страница транзакций;
6. Страница статистики;
7. Логин текущего пользователя;
8. Баланс (свобоные средства/всего средств);
9. Обновить данные;
10. Личный блокнот;
11. Страница настроек;
12. Закрыть приложение.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсового проекта были рассмотрены основные аналоги программного средства, их преимущества и недостатки, были просмотрены и изучены их возможности.

В ходе написания курсового проекта была разработана бизнес логика приложения, которая позволяет полностью функционировать приложению. Были изучены и применены такие паттерны как:

* command;
* builder;
* adapter;
* repository;
* unit of work;
* singleton.

Так же был изучен технология Entity Framework 6. Использованы и доработаны стили и темы Material Design. Иследован и применен алгоритм шифрования SHA1.

В рамках курсового проекта был исследован и испытан Azure Server. А так же принцип работы и администрирования удаленных баз данных. Показан принцип работы протокола SMTP для автоматизированной отправки сообщений на электронную почту.

При разработке архитектуры была изучена, применена и расширенна структура архитектурного паттерна MVVM.

Реализованна авторизация пользователей.

Реализованно вычисление и отоборажение статистических данных по рассходам и доходам.

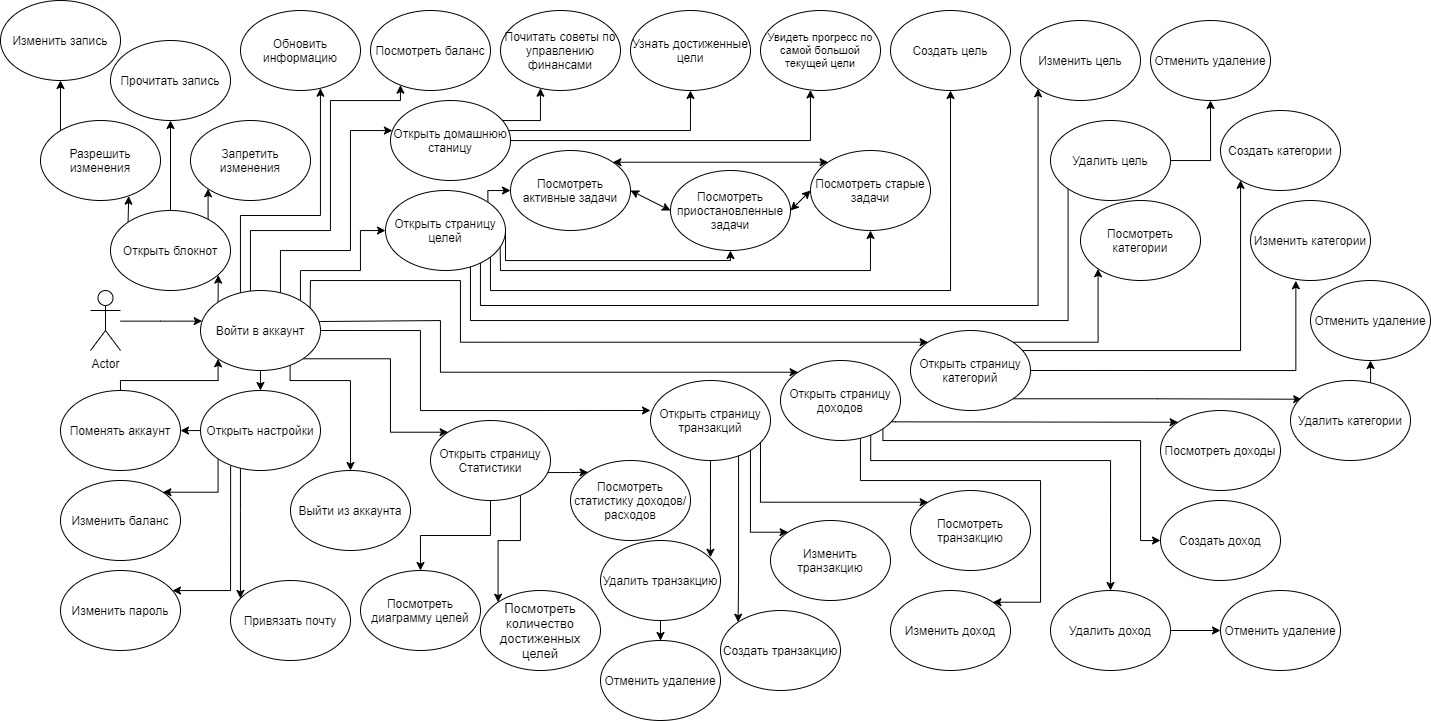
В рамках курсового проекта было разработано программное средство «Учет финансов(EconoMiC)», которое соответствует предъявленным требованиям и поставленным задачам.

Список используемых источников

1. Сайт о программировании Metanit: Руководство по WPF;
2. Сайт о программировании Metanit: Руководство по Entity Framework Core;
3. Сайт о программировании Habr: Использование Entity Framework Core code-first;
4. Сайт о программировании professorweb: Работа с Entity Framework 6;
5. Сайт о программировании professorweb: Пример публикации ClickOnce;
6. Сайт о программировании Metanit: Работа с электронной почтой.Отправка почты. SmtpClient;
7. Сайт о программировании Metanit: Паттерн MVVM;
8. Сайт о программировании Habr: MVVM полное понимание;
9. Сайт о программировании professorweb: Паттерн MVVM;
10. Сайт о программировании Metanit: Паттерны проектирования в C# и .NET;
11. Сайт-справочник по стилям material design material.io;
12. Банковский сайт mtbank.by: Популярные приложения по учету финансов.

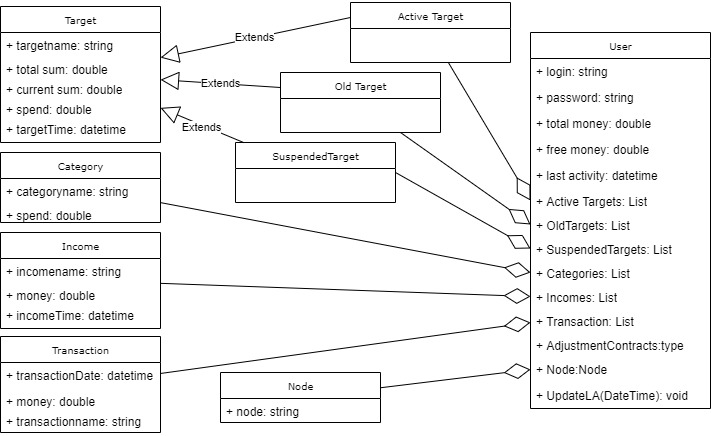
Приложение А

Диаграмма вариантов использования



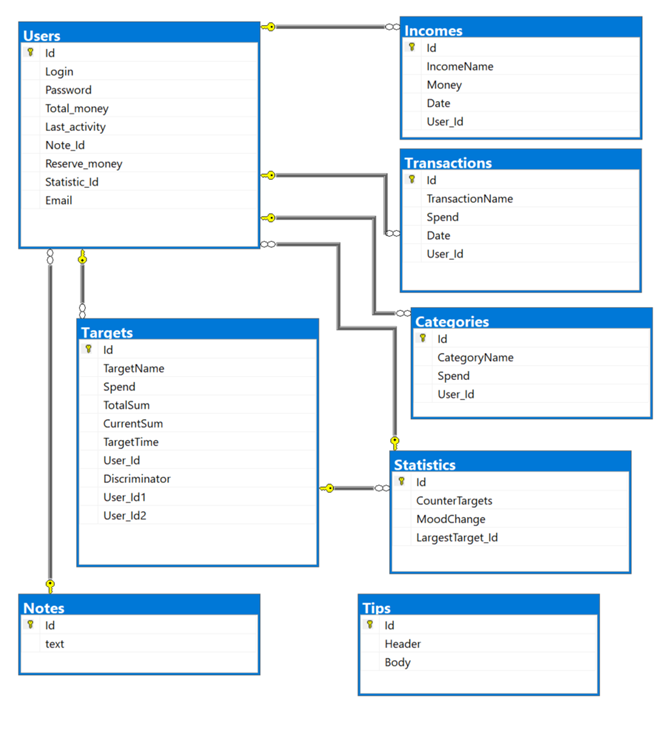
Приложение Б

Диаграмма классов



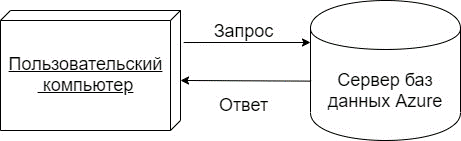
Приложение В

Логическая схема базы данных



Приложение Г

Диаграмма размещений



Приложение Д

Листинг 1. Код классов пространства имен DataBaseLayer.

EconoMiCDBContext

using Economic\_v2.Models;

using System.Data.Entity;

namespace Economic\_v2.DataBaseLayer

{

public class EconoMiCDBContext:DbContext

{

public EconoMiCDBContext() : base() { }

public DbSet<User> Users { get; set; }

public DbSet<Note> Nodes { get; set; }

public DbSet<Income> Incomes { get; set; }

public DbSet<ActiveTarget> ActiveTargets{ get; set; }

public DbSet<OldTarget> OldTargets{ get; set; }

public DbSet<Target> Targets{ get; set; }

public DbSet<SuspendedTarget> SuspendedTargets{get;set; }

public DbSet<Category> Categories { get; set; }

public DbSet<Transaction> Transactions { get; set; }

public DbSet<Statistic> Statistics { get; set; }

public DbSet<Tip> Tips { get; set; }

}

}

Repository

Все репазитории реализованы по одному шаблону

public class TransactionRepository : IRepository<Transaction>

{

private EconoMiCDBContext db;

public TransactionRepository(EconoMiCDBContext context)

{

this.db = context;

}

public List<Transaction> GetAll()

{

return db.Transactions.ToList();

}

public Transaction Get(int id)

{

return db.Transactions.Find(id);

}

public void Create(Transaction Transaction)

{

db.Transactions.Add(Transaction);

}

public void Update(Transaction Transaction)

{

db.Entry(Transaction).State = EntityState.Modified;

}

public void Delete(int id)

{

Transaction = db.Transactions.Find(id);

if (Transaction != null)

db.Transactions.Remove(Transaction);

}

}

UnitOfWork

namespace Economic\_v2.DataBaseLayer

{

public class UnitOfWork : IDisposable //use pattern Unit of work

{

private EconoMiCDBContext db = new EconoMiCDBContext();

private UserRepository;

private NodeRepository;

private CategoryRepository;

private ActiveTargetRepository;

private OldTargetRepository;

private SuspendedTargetRepository;

private AdjustmentContractRepository;

private AdjustmentRepository;

private IncomeRepository;

private TransactionRepository;

private StatisticRepository;

private TipRepository;

public UserRepository Users

{

get

{

if (userRepository == null)

userRepository = new UserRepository(db);

return userRepository;

}

}

public StatisticRepository Statistics

{

get

{

if (statisticRepository == null)

statisticRepository = new StatisticRepository(db);

return statisticRepository;

}

}

public OldTargetRepository OldTargets

{

get

{

if (oldTargetRepository == null)

oldTargetRepository = new OldTargetRepository(db);

return oldTargetRepository;

}

}

public SuspendedTargetRepository SuspendedTargets

{

get

{

if (suspendedTargetRepository == null)

suspendedTargetRepository = new SuspendedTargetRepository(db);

return suspendedTargetRepository;

}

}

public AdjustmentRepository Adjustments

{

get

{

if (adjustmentRepository == null)

adjustmentRepository = new AdjustmentRepository(db);

return adjustmentRepository;

}

}

public AdjustmentContractRepository AdjustmentContracts

{

get

{

if (adjustmentContractRepository == null)

adjustmentContractRepository = new AdjustmentContractRepository(db);

return adjustmentContractRepository;

}

}

public NodeRepository Notes

{

get

{

if (nodeRepository == null)

nodeRepository = new NodeRepository(db);

return nodeRepository;

}

}

public TipRepository Tips

{

get

{

if (tipRepository == null)

tipRepository = new TipRepository(db);

return tipRepository;

}

}

public CategoryRepository Categories

{

get

{

if (categoryRepository == null)

categoryRepository = new CategoryRepository(db);

return categoryRepository;

}

}

public ActiveTargetRepository ActiveTargets

{

get

{

if (activeTargetRepository == null)

activeTargetRepository = new ActiveTargetRepository(db);

return activeTargetRepository;

}

}

public IncomeRepository Incomes

{

get

{

if (incomeRepository == null)

incomeRepository = new IncomeRepository(db);

return incomeRepository;

}

}

public TransactionRepository Transactions

{

get

{

if (transactionRepository == null)

transactionRepository = new TransactionRepository(db);

return transactionRepository;

}

}

public void Save()

{

try

{

db.SaveChanges();

}

catch { }

}

private bool disposed = false;

public virtual void Dispose(bool disposing)

{

if (!this.disposed)

{

if (disposing)

{

db.Dispose();

}

this.disposed = true;

}

}

public void Dispose()

{

Dispose(true);

GC.SuppressFinalize(this);

}

}

}

UnitOfWorkSingleton

namespace Economic\_v2.DataBaseLayer

{

public class UnitOfWorkSingleton

{

private UnitOfWorkSingleton() { }

private static object locker = new object();

private static UnitOfWork \_unitOfWork;

public static UnitOfWork GetUnitOfWork

{

get

{

lock (locker)

{

if (\_unitOfWork == null)

\_unitOfWork = new UnitOfWork();

return \_unitOfWork;

}

}

}

}

}

Приложение Е

Листинг 1. Код сервиса по отправке сообщений на почту.

using System.Configuration;

using System.Net;

using System.Net.Mail;

using System.Threading.Tasks;

namespace Economic\_v2.Services

{

public class MailsService

{

public static void SendEmail(string emailTo, string title, string displayName, string htmlBody = "")

{

(new Task(() =>

{

string login = ConfigurationManager.AppSettings["emailLogin"];

string password = ConfigurationManager.AppSettings["emailPass"];

MailAddress from = new MailAddress(login, displayName);

MailAddress to = new MailAddress(emailTo);

MailMessage message = new MailMessage(from, to)

{

Subject = title,

Body = htmlBody,

IsBodyHtml = true

};

SmtpClient smtp = new SmtpClient("smtp.gmail.com", 587)

{

Credentials = new NetworkCredential(login, password),

EnableSsl = true,

};

smtp.SendMailAsync(message);

})).Start();

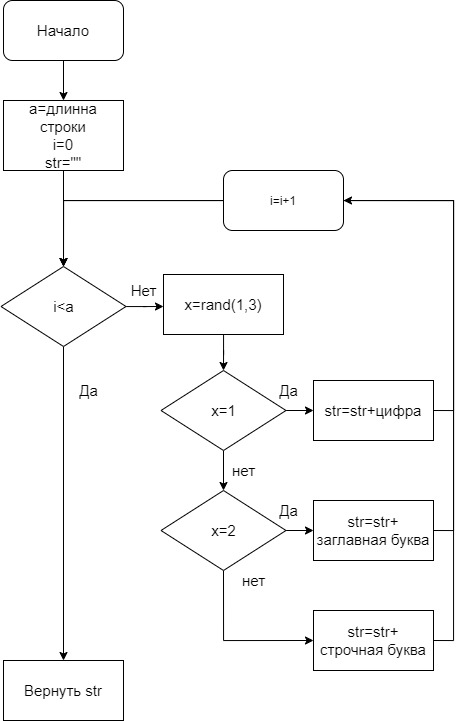
}

}

}

Приложение Ж

Алгоритм генерации случайной строки



Приложение И

using Economic\_v2.DataBaseLayer;

using Economic\_v2.Models;

using Economic\_v2.ViewModels;

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace Economic\_v2.Logic

{

public class Calculator

{

private DateTime CurDateTime = DateTime.Now;

private int month;

public Calculator()

{

}

public void MakeTransaction(User user, double spend)

{

user.Total\_money -= spend;

Save(user); Set(user); Update();

}

public void Calculate(User user)

{

if (user.Last\_activity.Date < CurDateTime.Date)

{

month = CurDateTime.Month - user.Last\_activity.Month +

12 \* (CurDateTime.Year - user.Last\_activity.Year);

Task taskTotalMoney = new Task(() =>

{

user.Total\_money += CalculateIncomes(user) - CalculateCategories(user);

});

Task taskReservedMoney = new Task(() =>

{

CalculateTargets(ref user);

});

taskTotalMoney.Start(); taskReservedMoney.Start();

Task.WaitAll(taskTotalMoney, taskReservedMoney);

Set(user);

Save(user);

}

Update();

}

private void Save(User user)

{

new Task(() =>

{

UnitOfWorkSingleton.GetUnitOfWork.Users.Update(user);

UnitOfWorkSingleton.GetUnitOfWork.Save();

}).Start();

}

private void Set(User user)

{

user.Last\_activity = CurDateTime;

MainViewModel.GetContext.CurrentUser = user;

}

private void Update()

{

MainViewModel.GetContext.UpdateInfo();

}

private int GetIncomeDay(DateTime, int day)

{

if (dateTime.Day != DateTime.DaysInMonth(dateTime.Year, dateTime.Month))

{

while (dateTime.Day != day) { dateTime = dateTime.AddDays(1); }

}

return dateTime.Day;

}

private DateTime GetDateWithRightDay(DateTime, int day)

{

if (day > 27)

if (dateTime.Day != DateTime.DaysInMonth(dateTime.Year, dateTime.Month))

{

while (dateTime.Day != day) { dateTime = dateTime.AddDays(1); }

}

return dateTime;

}

private double CalculateIncomes(User user)

{

double totalIncome = 0;

int SavedIncomeDay, nowDay = CurDateTime.Day;

DateTime marker;

if (user.Incomes != null)

{

if (user.Last\_activity.Date != CurDateTime.Date)

{

foreach (Income in user.Incomes)

{

SavedIncomeDay = income.Date.Day;

marker = DateTime.Parse(user.Last\_activity.ToShortDateString());

while (marker.Day != SavedIncomeDay)

{

marker = marker.AddDays(1);

SavedIncomeDay = GetIncomeDay(marker, income.Date.Day);

}

while (marker <= CurDateTime)

{

totalIncome += income.Money;

marker = GetDateWithRightDay(marker.AddMonths(1), nowDay);

}

}

}

}

return totalIncome;

}

private double CalculateCategories(User user)

{

double monthlyExpenses = 0;

if (user.Categories != null)

{

foreach (Category cat in user.Categories)

{

monthlyExpenses += cat.Spend;

}

}

double totalSpend = month \* monthlyExpenses;

return totalSpend;

}

private void CalculateTargets(ref User user)

{

if (user.ActiveTargets != null)

{

TargetCalcController taskCacl = new TargetCalcController();

double reservedSum = 0;

ActiveTarget temp;

List<ActiveTarget> targets = new List<ActiveTarget>();

List<OldTarget> oldTargets = new List<OldTarget>();

foreach (ActiveTarget target in user.ActiveTargets)

{

temp = taskCacl.CalcTarget(target, month, ref reservedSum);

if (temp.TotalSum == temp.CurrentSum)

{

oldTargets.Add(temp.CopyTo<OldTarget>(null));

}

else

{

targets.Add(temp);

}

}

foreach(OldTarget target in oldTargets)

{

user.OldTargets.Add(target);

}

user.ActiveTargets = targets;

if (user.SuspendedTargets != null)

{

foreach (SuspendedTarget target in user.SuspendedTargets)

{

reservedSum += target.CurrentSum;

}

}

user.Reserve\_money = reservedSum;

}

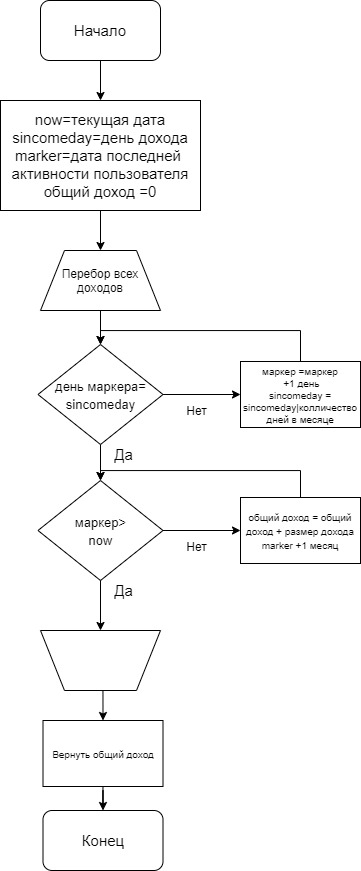
}

}

}

Приложение К

Алгоритм вычисления общего дохода



Приложение Л

using Economic\_v2.Models;

using System;

namespace Economic\_v2.Logic

{

class TargetCalcController

{

public TargetCalcController()

{

}

public ActiveTarget CalcTarget(ActiveTarget target, int month, ref double reservedSum)

{

double reserve = target.CurrentSum + month \* target.Spend;

if (reserve > target.TotalSum)

reserve = target.TotalSum;

target.CurrentSum = reserve;

reservedSum += reserve;

return target;

}

public Target CalculateTargetWithSpend(Target target)

{

double neededMoney = target.TotalSum - target.CurrentSum;

DateTime targetTime = DateTime.Parse("1/" + DateTime.Now.Month.ToString() + '/' + DateTime.Now.Year.ToString());

int month = (int)(neededMoney / target.Spend);

if (month > 120 || month < 0)

{

month = 120;

}

if (neededMoney % target.Spend != 0)

{

targetTime = targetTime.AddMonths(1);

}

target.TargetTime = targetTime.AddMonths(month);

return target;

}

private double RoundUp(double number, int digits)

{

var factor = Convert.ToDouble(Math.Pow(10, digits));

return Math.Ceiling(number \* factor) / factor;

}

public Target CalculateTargetWithTime(Target target)

{

double neededMoney = target.TotalSum - target.CurrentSum;

DateTime targetTime = DateTime.Parse(

"1/" + DateTime.Now.Month.ToString() + '/' + DateTime.Now.Year.ToString());

int month = target.TargetTime.Month - DateTime.Today.AddMonths(1).Month +

12 \* (target.TargetTime.Year - DateTime.Today.Year);

if (neededMoney % month == 0)

target.Spend = RoundUp(neededMoney / month, 2);

else

target.Spend = RoundUp(neededMoney / month + 1, 2);

return target;

}

}

}

Приложение М

using Economic\_v2.Builders;

using Economic\_v2.DataBaseLayer;

using Economic\_v2.Pages;

using Economic\_v2.ViewModels;

using System.Collections.Generic;

using System.Threading.Tasks;

namespace Economic\_v2.Models

{

public class TargetsModel

{

object[] DataContexts = new object[3]; //here data context all controls where was Target

object[] Views = new object[3]; //here Views

UnitOfWork Data = UnitOfWorkSingleton.GetUnitOfWork;

public TargetsModel(object context)

{

Views[0] = new AddOrEditTarget();

Views[1] = new TargetsListView();

DataContexts[0] = ((AddOrEditTarget)Views[0]).DataContext;

DataContexts[1] = ((TargetsListView)Views[1]).DataContext;

DataContexts[2] = context;

}

#region Contexts

public AddOrEditTargetsViewModel AddOrEditContext

{

get => (AddOrEditTargetsViewModel)DataContexts[0];

}

public TargetsListViewViewModel ListContext

{

get => (TargetsListViewViewModel)DataContexts[1];

}

public TargetsPageViewModel PageContext

{

get => (TargetsPageViewModel)DataContexts[2];

}

#endregion

#region Views

public AddOrEditTarget AddOrEditView

{

get => (AddOrEditTarget)Views[0];

}

public TargetsListView ListView

{

get => (TargetsListView)Views[1];

}

public TargetsPage PageView

{

get => (TargetsPage)Views[2];

}

#endregion

public void NotifyEndableButtonsEditAndDelete()

{

PageContext.NotifyEndableButtonsEditAndDelete();

}

public bool DismissDelete;

public Target DeletedTarget;

public int deleteMode;

public void DeleteTarget()

{

new Task(() => //start method in context

{

ListContext.OnDeleteTarget(

new TargetBuilder(DeletedTarget).Build(),deleteMode);

DeletedTarget = null;

deleteMode = -1;

}).Start();

}

public void ConfirmButton(bool isEdit, int mode)

{

new Task(() =>

{

if (MainViewModel.GetContext.CurrentUser.ActiveTargets == null) //if current not any note

{

MainViewModel.GetContext.CurrentUser.ActiveTargets = new List<ActiveTarget>();

}

if (MainViewModel.GetContext.CurrentUser.SuspendedTargets == null) //if current not any note

{

MainViewModel.GetContext.CurrentUser.SuspendedTargets = new List<SuspendedTarget>();

}

if (MainViewModel.GetContext.CurrentUser.OldTargets == null) //if current not any note

{

MainViewModel.GetContext.CurrentUser.OldTargets = new List<OldTarget>();

}

Target target = AddOrEditContext.GetTarget.CopyTo<Target>(null); //convert to needed type

double newSum = target.CurrentSum;

double oldSum = 0;

bool flagCreateNew = false;

switch (mode)

{

case 0:

if (isEdit && target.Id != 0)

{

oldSum = MainViewModel.GetContext.CurrentUser.ActiveTargets.Find(x => x.Id == target.Id).CurrentSum;

if (target.CurrentSum == target.TotalSum)

{

MainViewModel.GetContext.CurrentUser.ActiveTargets.RemoveAll(x => x.Id == target.Id);

MainViewModel.GetContext.CurrentUser.OldTargets.Add(target.CopyTo<OldTarget>(null));

}

else

{

if (target.Spend == 0)

{

MainViewModel.GetContext.CurrentUser.ActiveTargets.RemoveAll(x => x.Id == target.Id);

MainViewModel.GetContext.CurrentUser.SuspendedTargets.Add(target.CopyTo<SuspendedTarget>(null));

}

else

{

MainViewModel.GetContext.CurrentUser.ActiveTargets[MainViewModel.GetContext.

CurrentUser.ActiveTargets.FindIndex(x => x.Id == target.Id)] = target.CopyTo<ActiveTarget>(null);

}

}

}

else

{

flagCreateNew = true;

}

break;

case 1:

if (isEdit && target.Id != 0)

{

oldSum = MainViewModel.GetContext.CurrentUser.SuspendedTargets.Find(x => x.Id == target.Id).CurrentSum;

if (target.CurrentSum == target.TotalSum)

{

MainViewModel.GetContext.CurrentUser.SuspendedTargets.RemoveAll(x => x.Id == target.Id);

MainViewModel.GetContext.CurrentUser.OldTargets.Add(target.CopyTo<OldTarget>(null));

}

else

{

if (target.Spend == 0)

{

MainViewModel.GetContext.CurrentUser.SuspendedTargets[MainViewModel.GetContext.

CurrentUser.SuspendedTargets.FindIndex(x => x.Id == target.Id)] = target.CopyTo<SuspendedTarget>(null);

}

else

{

MainViewModel.GetContext.CurrentUser.SuspendedTargets.RemoveAll(x => x.Id == target.Id);

MainViewModel.GetContext.CurrentUser.ActiveTargets.Add(target.CopyTo<ActiveTarget>(null));

}

}

}

else

{

flagCreateNew = true;

}

break;

case 2:

if (isEdit && target.Id != 0)

{

oldSum = MainViewModel.GetContext.CurrentUser.OldTargets.Find(x => x.Id == target.Id).CurrentSum;

if (target.CurrentSum == target.TotalSum)

{

MainViewModel.GetContext.CurrentUser.OldTargets[MainViewModel.GetContext.

CurrentUser.OldTargets.FindIndex(x => x.Id == target.Id)] = target.CopyTo<OldTarget>(null);

}

else

{

if (target.Spend == 0)

{

MainViewModel.GetContext.CurrentUser.OldTargets.RemoveAll(x => x.Id == target.Id);

MainViewModel.GetContext.CurrentUser.SuspendedTargets.Add(target.CopyTo<SuspendedTarget>(null));

}

else

{

MainViewModel.GetContext.CurrentUser.OldTargets.RemoveAll(x => x.Id == target.Id);

MainViewModel.GetContext.CurrentUser.ActiveTargets.Add(target.CopyTo<ActiveTarget>(null));

}

}

}

else

{

flagCreateNew = true;

}

break;

}

if (flagCreateNew)

{

if (target.CurrentSum == target.TotalSum)

{

MainViewModel.GetContext.CurrentUser.OldTargets.Add(target.CopyTo<OldTarget>(null));

}

else

{

if (target.Spend == 0)

{

MainViewModel.GetContext.CurrentUser.SuspendedTargets.Add(target.CopyTo<SuspendedTarget>(null));

}

else

{

MainViewModel.GetContext.CurrentUser.ActiveTargets.Add(target.CopyTo<ActiveTarget>(null));

}

}

}

ListContext.SelectedTarget = null;

MainViewModel.GetContext.CurrentUser.Reserve\_money += newSum - oldSum;

ListContext.NotifyTargetList();

MainViewModel.GetContext.UpdateInfo();

if (StatisticViewModel.GetContext != null)

StatisticViewModel.GetContext.MakeCalculate();

Data.Users.Update(MainViewModel.GetContext.CurrentUser);

Data.Save();

}).Start();

}

}

}

Приложение Н

using Economic\_v2.Models;

using System;

namespace Economic\_v2.Builders

{

public class TargetBuilder : Target

{

public TargetBuilder()

{

}

public TargetBuilder(Target target)

{

if (target != null)

{

this.Id = target.Id;

this.TargetName = target.TargetName;

this.TotalSum = target.TotalSum;

this.CurrentSum = target.CurrentSum;

this.Spend = target.Spend;

this.TargetTime = target.TargetTime;

}

}

public TargetBuilder SetName(string name)

{

this.TargetName = name;

return this;

}

public TargetBuilder SetSpend(double spend)

{

this.Spend = spend;

return this;

}

public TargetBuilder SetTotalSum(double totalsum)

{

this.TotalSum = totalsum;

return this;

}

public TargetBuilder SetCurrentSum(double currentSum)

{

this.CurrentSum = currentSum;

return this;

}

public TargetBuilder setTargetTime(DateTime dateTime)

{

this.TargetTime = dateTime;

return this;

}

public Target Build()

{

return this;

}

}

}

Приложение П

using System;

using System.Windows.Input;

namespace Economic\_v2.Commands

{

public class RelayCommand<T> : ICommand //command with parameter

{

private Action<T> \_execute;

public event EventHandler CanExecuteChanged;

public bool CanExecute(object parameter)

{

return true;

}

public void Execute(object parameter)

{

\_execute?.Invoke((T)parameter);

}

public RelayCommand(Action<T> execute)

{

\_execute = execute;

}

}

public class Command : ICommand // command without parameter

{

public Command() { }

private Action \_execute;

public event EventHandler CanExecuteChanged;

public Command(Action execute)

{

\_execute = execute;

}

public bool CanExecute(object parameter)

{

return true;

}

public void Execute(object parameter)

{

\_execute?.Invoke();

}

}

}