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

Направление специальности 1-40 01 01 10 Программное обеспечение информационных технологий (программирование интернет-изданий)

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**КУРСОВОГО ПРОЕКТА:**

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

Тема Программное средство «MoneyManager»

Исполнитель

студент 2 курса группы 4 Федоров Дмитрий Андреевич

(Ф.И.О.)

Руководитель работы ассистент Рауба А.А.

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

Курсовой проект защищен с оценкой

Председатель Рауба А.А.

(подпись)

Минск 2020

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc41304230)

[1 Анализ прототипов, литературных источников и формирование требований к проектируемому программному средству 4](#_Toc41304231)

[2 Анализ требований к программному средству и разработка функциональных требований 5](#_Toc41304232)

[3 Проектирование программного средства 6](#_Toc41304233)

[3.1 Структуры и взаимоотношения между классами 6](#_Toc41304234)

[3.2 Описание данных 8](#_Toc41304235)

[4 Реализация программного средства 10](#_Toc41304236)

[4.1 Реализация классов для работы с данными 10](#_Toc41304237)

[4.2 Реализация классов шифрования 14](#_Toc41304238)

[4.3 Реализация классов интерфейса 15](#_Toc41304239)

[4.4 Сборка проекта 17](#_Toc41304240)

[5 Тестирование, проверка работоспособности и анализ полученных результатов 18](#_Toc41304241)

[5.1 Тестирование страницы авторизации и регистрации 18](#_Toc41304242)

[5.2 Тестирование главного меню пользователей 20](#_Toc41304243)

[5.3 Тестирование форм добавления данных 20](#_Toc41304244)

[6 Руководство по использованию 22](#_Toc41304245)

[6.1 Системные требования 22](#_Toc41304246)

[6.2 Использование 22](#_Toc41304247)

[ЗАКЛЮЧЕНИЕ 23](#_Toc41304248)

[Список использованных источников 24](#_Toc41304249)

[ПРИЛОЖЕНИЕ А 25](#_Toc41304250)

[ПРИЛОЖЕНИЕ Б 27](#_Toc41304251)

[ПРИЛОЖЕНИЕ В 33](#_Toc41304252)

[ПРИЛОЖЕНИЕ Г 35](#_Toc41304253)

ВВЕДЕНИЕ

Приложение «MoneyManager» - это программное средство предназначенное для контроля расходов. Подобное ПО позволяет человеку проще контролировать свои расходы, путем записи своих расходов и доходов, а также составлением плана расходов на календарный день. Грамотное планирование расходов и контроль своих финансов является важной частью современной жизни, поэтому данное программное средство (далее ПС) окажется полезным для большого количества людей. Особенно полезным данное ПС может оказаться для студентов и школьников, поскольку в период обучения многие из них не обладают стабильным источником дохода, поэтому контроль немногочисленных материальных средств является немаловажным элементом их повседневной жизни.

Целью данного курсового проекта являются проектирование и разработка такого программного средства.

При выполнении курсового проекта используются принципы и приемы ООП. Разработка осуществляется на языке C#. Для разработки графического интерфейса пользователя используется технология Windows Forms, являющаяся частью платформы .NET Framework.

В первом разделе пояснительной записки представлен анализ прототипов и литературных источников.

Во втором разделе производится анализ и разработка требований к ПС.

В третьем разделе рассматривается проектирование ПС.

В четвертом разделе рассматривается разработка ПС.

В пятом разделе производится тестирование ПС.

В шестом разделе содержится руководство по использованию ПС.

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

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

На сегодняшний день можно встретить достаточное количество подобных программных средств. Примером такого ПС является программа «Семейный бюджет».

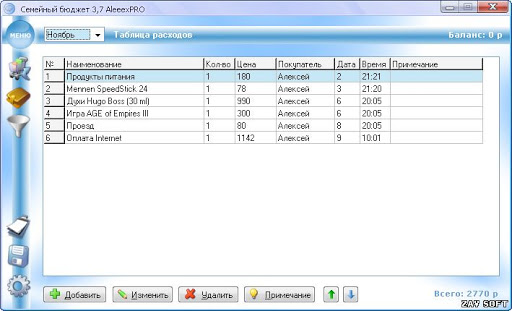


Рис.1.1 – Программа «Семейный бюджет»

Она позволяет записывать расходы и доходы, а также составлять план расходов.

Проанализировав часть программы можно выделить её основные плюсы и минусы.

Основные плюсы:

* Возможность составления плана расходов.
* Просмотр расходов и доходов пользователя.

Основные минусы:

* Направленность на семейное использование.
* Отсутствие возможности добавления типов частых доходов и расходов.

На основе проведенного анализа и с учетом требований, указанных в задании

на курсовое проектирование, сформированы следующие требования:

* Разработать бизнес логику приложения;
* Реализовать программный интерфейс;
* Реализовать работу с БД.

Для разработки данного ПС были выбраны следующие средства разработки:

* Интегрированная среда разработки Microsoft Visual Studio 2019;
* Программная платформа .NET Framework 4.7;
* Язык программирования C#;
* Технология Windows Forms;
* Технология ADO.NET.

2 Анализ требований к программному средству и разработка функциональных требований

Описание функциональности ПС:

* Возможность авторизации и регистрации;
* Наличие интерфейса для управления приложением;
* Возможность просматривать список ваших расходов и доходов;
* Возможность создавать план расходов по датам;
* Возможность добавления типов расходов и доходов;
* Наличие оповещения о перерасходе средств через email;

Вариант использования: пользователю необходимо осуществлять контроль за своими финансами. Ему необходимо зарегистрироваться в системе, затем он может добавлять типы расходов и доходов, данные типы являются общими для всех клиентов. Если пользователю требуется, то он составляет план расходов по датам. Далее пользователь имеет возможность фиксировать все свои доходы и расходы.

Спецификация функциональных требований:

1. Авторизация должна быть безопасна для пользователей, т.е. пароли будут храниться в БД в зашифрованном виде;
2. Основная часть функций будет реализовывать работу с БД;
3. Функциональность оповещений по Email будет нуждаться в информации (логин и пароль для smtp сервера);

Ниже, а также в графических материалах приведена диаграмма использования ПС (Use Case).

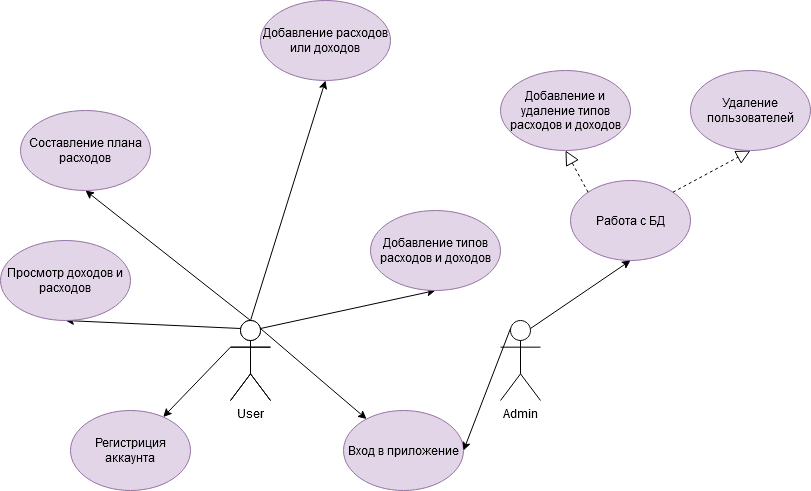


Рисунок 2.1 – Диаграмма использования

На данной диаграмме приведены возможные применения программы для таких ролей как «Администратор» и «Пользователь».

3 Проектирование программного средства

3.1 Структуры и взаимоотношения между классами

Приложение построено на основе взаимодействия форм Windows Forms с соответствующими им классами. Формы выполняют роль графического интерфейса пользователя, функционал же описан в соответствующих классах. Команды отдаются классам исполнителям по нажатию кнопок на форме.

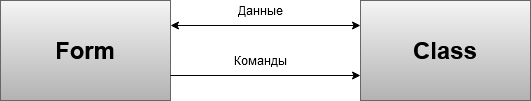


Рисунок 3.1 – Визуальное представление работы ПС

Ниже приведены диаграммы классов ПС. Более подробные диаграммы приведены в графических материалах.

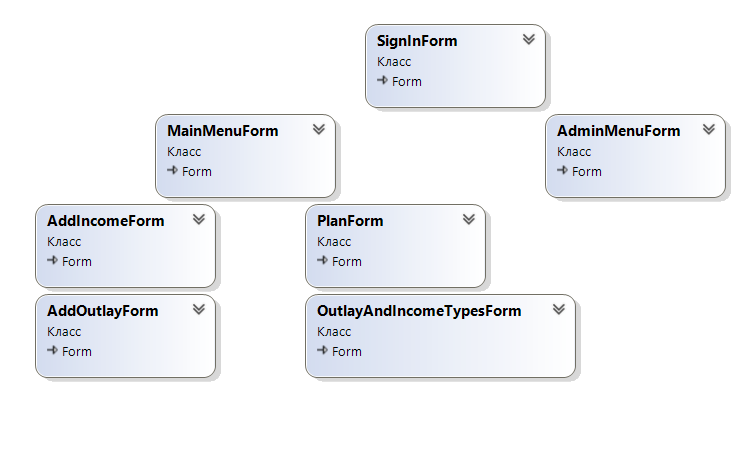


Рисунок 3.2 – Диаграмма классов форм

Класс SignInForm наследуется от класса Form и является окном входа в приложение. Классы MainMenuForm и AdminMenuForm выполняют функцию главного меню для обычного пользователя и администратора соответственно.

Остальные классы форм открываются по нажатию кнопок на форме MainMenuForm.

Классы AdminMenu и Menu содержат функционал форм AdminMenuForm и MainMenuForm соответственно.



Рисунок 3.3 – Классы содержащие функционал главного меню

Подобные классы существуют для каждой формы в приложении.

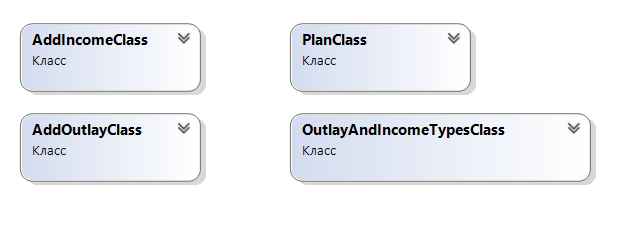


Рисунок 3.4 – Классы других форм

Класс Connect предназначен для соединения с БД. Класс SaltedHash предназначен для хэшировнаие данных. Классы Admin и OrdinaryUser представляют собой пользователя-администратора и обычного пользователя соответсвенно.

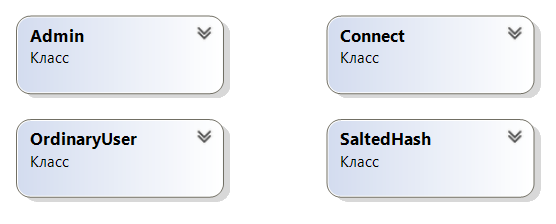


Рисунок 3.5 – Вспомогательные классы не относящиеся к классам формам

Подробно функционал этих классов будет описан в разделе 4.

3.2 Описание данных

Диаграмма базы данных сформированна в программе Microsoft SQL Server Management Studio 18 и представлена на рисунке 3.6. Для реализации необходимого функционала ПС и удобной разработки, в базе данных создано 7 таблиц: Admins для хранения системных данных администратора. Users для хранения системных данных обычного пользователя. Outlay\_Types для хранения типов расходов. Income\_Types для хранения типов доходов. Outlays для хранения расходов конкретного пользователя. Incomes для хранения доходов конкретного пользователя. OutlayPlan для хранения планов расходов конкретного пользователя.

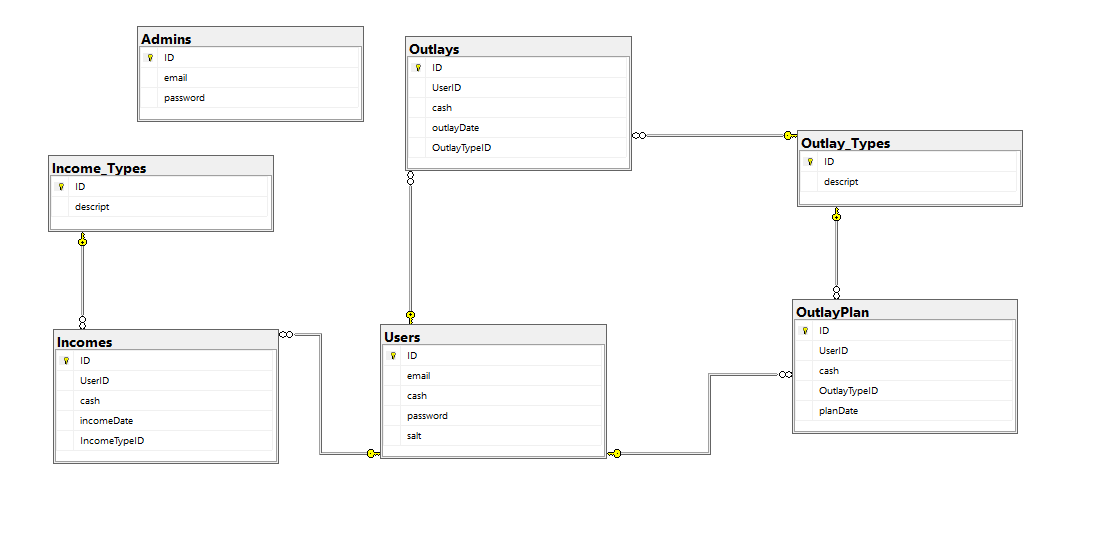


Рисунок 3.6 – Диаграмма базы данных

Таблица Users состоит из 5 столбцов:

* Идентификатор ID (первичный ключ);
* Электронная почта email;
* Количество средств cash;
* Захешированный пароль password;
* Соль хешированного пароля salt;

Таблица Admins состоит из 3 столбцов:

* Идентификатор Id (первичный ключ);
* Электронная почта email;
* Захешированный пароль password;

Таблица Outlay\_Types состоит из 2 столбцов:

* Идентификатор Id (первичный ключ);
* Описание расхода

Таблица Income\_Types состоит из 2 столбцов:

* Идентификатор Id (первичный ключ);
* Описание расхода;

Таблица Incomes состоит из 5 столбцов:

* Идентификатор Id (первичный ключ);
* Идентификатор пользователя userID(вторичный ключ);
* Количество полученных средств cash;
* Дата дохода incomeDate;
* Идентификатор типа дохода incomeTypeID(вторичный ключ);

Таблица Outlays состоит из 5 столбцов:

* Идентификатор Id (первичный ключ);
* Идентификатор пользователя userID(вторичный ключ);
* Количество потраченных средств cash;
* Дата расхода outlayDate;
* Идентификатор типа расхода outlayTypeID(вторичный ключ);

Таблица OurtlayPlan состоит из 5 столбцов:

* Идентификатор ID(первичный ключ);
* Идентификатор пользователя userID(вторичный ключ);
* Количество доступных средств cash;
* Идентификатор типа расхода outlayTypeId(вторичный ключ);
* Дата расхода planDate;

Идентификаторы в таблицах Users и Admins генерируются автоматически (identity) [1].

Ниже приведена таблица преобразования типов данных.

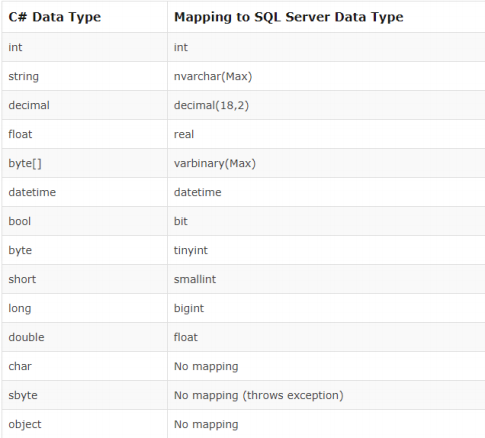


Рисунок 3.7 – Сопоставление типов

Все таблицы хранятся в базе данных на Microsoft SQL Server 2019, что позволяет удалённо и одновременно нескольким пользователям работать с данными в таблицах.

4 Реализация программного средства

4.1 Реализация классов для работы с данными

Связь ПС с базой данных реализована при помощи технологии ADO.NET. Классы реализующие архитектуру ADO.NET содержатся в пространстве имен *System.Data*[2]*.* Классы определяющие функциональность провайдера для MS SQL Server содержатся в пространстве имен *System.Data.SqlClient*[3]*.*

Класс отвечающий за соединение приложения БД Connect, реализован на основе паттерна Singleton, для того, чтобы предотвратить многократное подключение к базе данных.

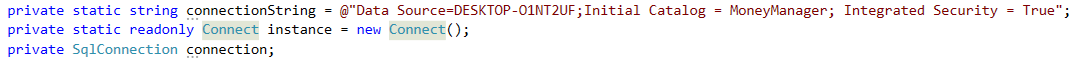


Рисунок 4.1 – Поля класса Connect

Конструктор класса Connect создает объект класса SqlConnection из пространства имен Data.SqlClient. Из-за особенностей реализации класса, при запуске программы создается единственный объект класса Connect с глобальной точкой доступа. Этот доступ реализует метод GetInstance().

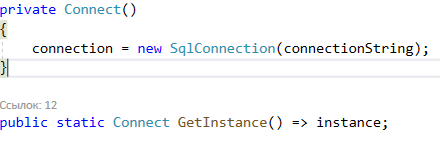


Рисунок 4.2 – Конструктор класса Connect и метод доступа GetInstance()

Для работы с подключением в классе содержиться 3 метода:

* GetConnection() для получения строки подключения;
* OpenConnection() для открытия соединения;
* CloseConnection() для закрытия соединения;

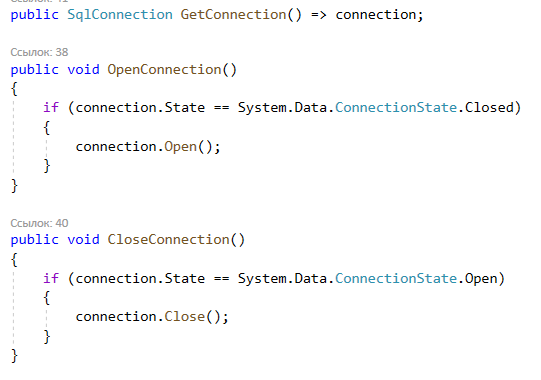


Рисунок 4.3 – Методы работы с подключением класса Connect

При входе для получения информации о пользователе используется класс OrdinaryUser. Он получает ID и количество доступных средств пользователя из БД.

Получение ID происходит с помощью метода GetID() осуществляющего подключение к базе данных и чтения данных с помощью класса SqlDataReader.



Рисунок 4.4 – Реализация метода для получения ID пользователя

Получение доступных средств происходит при помощи метода GetCash() осуществляющего подключение к БД и чтение данных с помощью класса SqlDataReader.

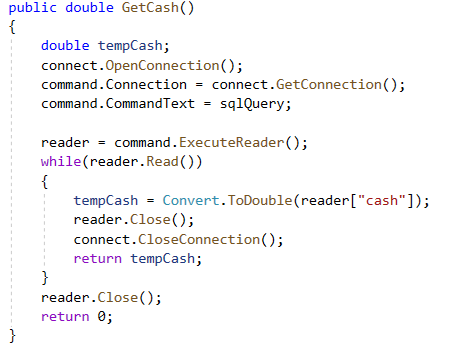


Рисунок 4.5 – Реализация метода на получение средств пользователя

Возможность удаления пользователей доступна администратору и реализована в классе AdminMenu с помощью метода DeleteUser().



Рисунок 4.6 – Реализация метода на удаление пользователя

Добавление доходов происходит при выполнении метода Add() класса AddIncome.

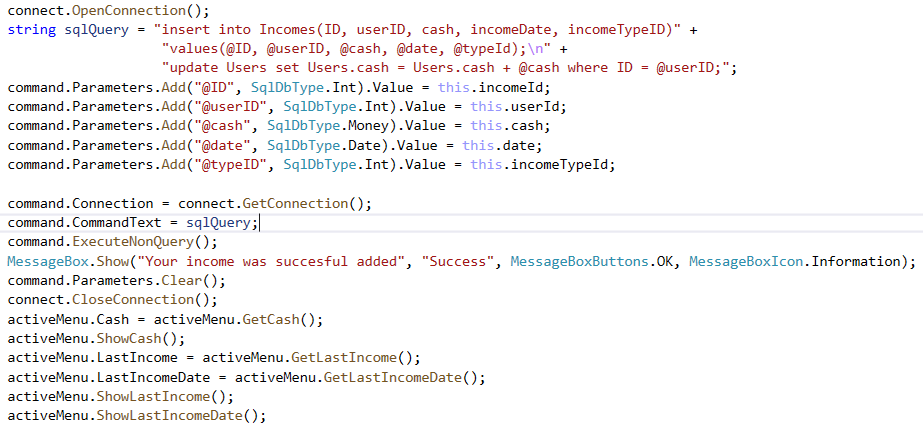


Рисунок 4.7 – Реализация метода на добавление дохода в БД.

В данном методе реализована проверка вводимых данных (конкретно даты) на основе RegEx.

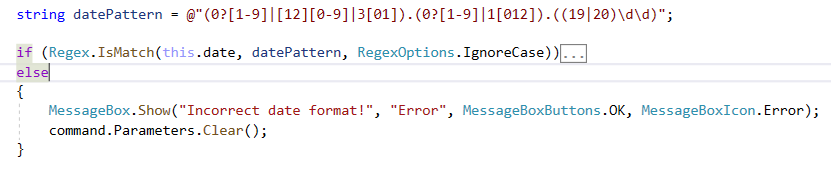
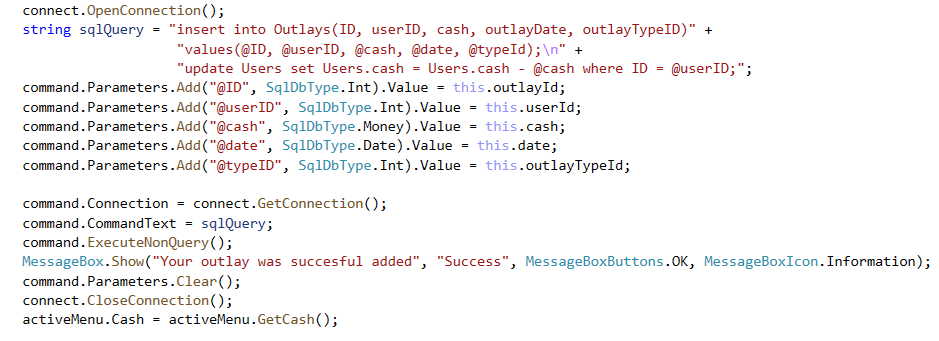


Рисунок 4.8 – Реализация проверки валидности введенной даты в методе Add() .

Добавление расходов происходит при выполнении метода Add() класса AddOutlay Рисунок 4.9 – Реализация метода на добавление расхода в БД.



В данном методе также реализована проверка валидности даты (см. Рисунок 4.8).

Подробную реализацию вышеприведенных классов можно найти в приложении A.

4.2 Реализация классов шифрования

Классом шифрования паролей является SaltedHash. Он предназначен для хеширования и проверки пароля пользователя.

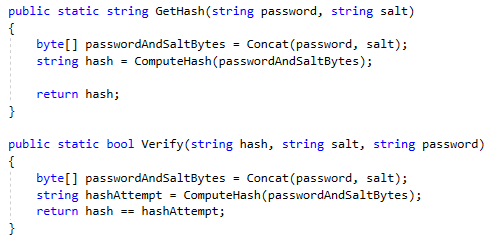


Рисунок 4.10 – Публичные методы SaltedHash

В методе GetHash с помощью приватных методов сначала получают массив байт, полученный из пароля и «соли». Соль нужна для предотвращения полной утечки паролей пользователей, без неё злоумышленник с помощью общедоступных утилит может распознать алгоритм хеширования и получить доступ сразу ко вмем паролям. При использовании динамической соли злоумышленнику придётся расшифровывать каждый пароль по отдельности.

Далее используется функция ComputeHash которая реализованна по алгоритму SHA256.

Метод Verify повторяет логику GetHash и сравнивает хеши паролей.

Подробную реализацию вышеприведённых классов можно найти в приложении Б.

4.3 Реализация классов интерфейса

Первый класс интерфейса, который встречает пользователь это форма SignInForm для входа в приложение или регистрации нового пользователя. При регестрации осуществляется проверка валидности email с помощью регулярного выражения, а также проверка пароля на длину не менее 5 символов.

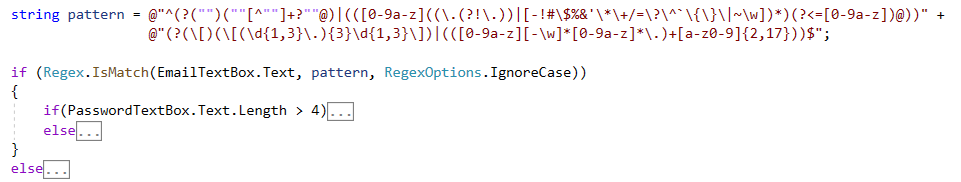


Рисунок 4.11 – Часть метода SignUp класса SignInForm

При авторизации пользователя, проверяется валидность введенных данных (email и пароля) через запрос к базе данных. В том случае, если какое-то из полей пустое, то программа оповещает об этом пользователя. Данный механизм описан в методе SignIn класса SignInForm.



Рисунок 4.12 – Часть метода SignIn класса SignInForm

При добавлении данных (доходы, расходы, внесение поправок в план расходов, добавление типов доходов/расходов) осуществляется проверка на корректность типа введенных данных, а также заполнение нужных полей информацией.

Для примера рассмотрим класс AddIncome, содержащий в себе функционал формы AddIncomeForm.



Рисунок 4.13 – Проверка валидности данных в классе AddIncome

Проверка осуществляется за счет обработки исключений, которые могут быть вызваны в ходе выполнения программы. Исключение FormatException вызывается в случае, если поле класса не заполнено вовсе, или данные, которые пользователь попытался передать в качестве значения поля класса не соответсвует типу поля класса. Как было описано выше в этом пункте данного раздела, также осуществляется проверка на валидность формата введенной пользователем даты, если это необходимо.

Подобная проверка осуществляется в каждом классе, отвечающем за класс интерфейса пользователя.

Подробное описание этих классов можно найти в приложении В.

4.4 Сборка проекта

Сборка для разработки происходит автоматически и не требует дополнительных настроек.

Сборка проекта в установочный файл происходит в следующем порядке:

* Открыть вкладку «Сборка» в Visual Studio.
* Выбрать пункт «Собрать решение».
* Скомпилировать проект.
* Для дальнейшего запуска программы используется .exe файл расположенный в файле Debug с проектом.

5 Тестирование, проверка работоспособности и анализ полученных результатов

5.1 Тестирование страницы авторизации и регистрации

Авторизация в обычной ситуации работает стабильно.



Рисунок 5.1 – Вид окна авторизации ПС

При регистрации пользователя, может возникнуть ситуаация, что email введен некорректно. Посмотрим как приложение оповестит об этом пользователя.

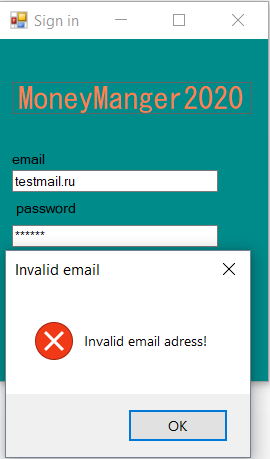


Рисунок 5.2 – Результат теста при неверном вводе email

На представленном рисунке видно, что email не соответсвует правилам валидации. Программа оповещает об этом пользователя.

Далее рассмотрим ситуацию, когда пользователь с данным email уже зарегистрирован.

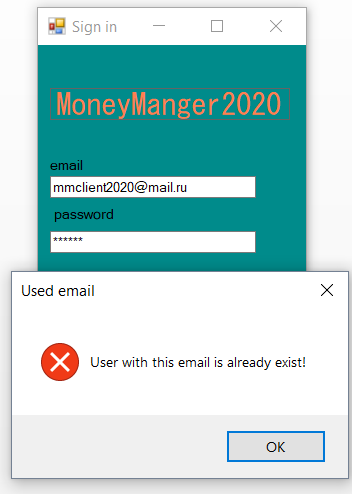


Рисунок 5.3 – Результат теста при повтороной попытке регистрации аккаунта с данным email

Далее рассмотрим попытку входа в приложение использую незарегистрированный почтовый адрес.

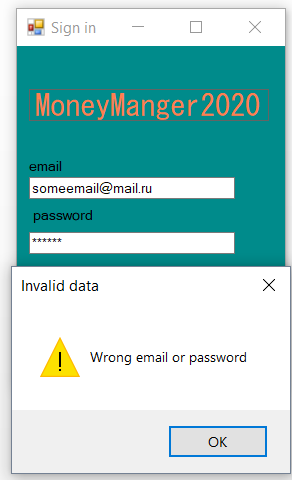


Рисунок 5.4 – Результат теста при несуществующем email

На представленном рисунке видно, что пользователь по заданному email не найден.

5.2 Тестирование главного меню пользователей

Зайдем в приложение с уже зарегистрированного аккаунта.

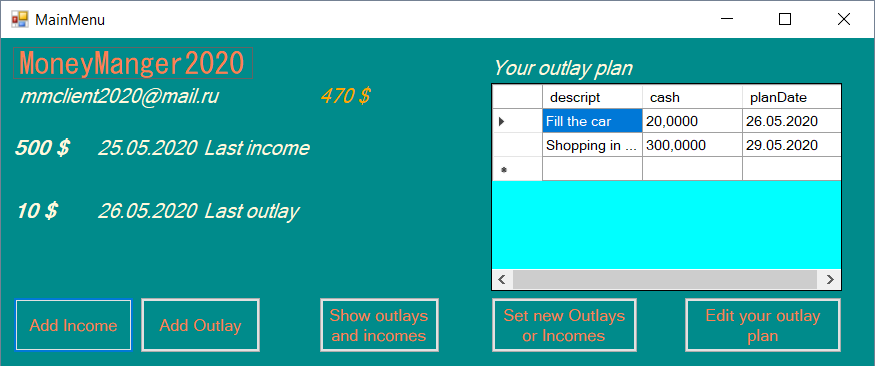


Рисунок 5.5 – Главное меню пользователя

Мы наблюдаем главное меню с данными доступными для пользователя

5.3 Тестирование форм добавления данных

Рассмотрим данный пункт на примере добавления расходов пользователем. Меню для добавления расхода откроется по нажатию кнопки Add Outlay в главном меню.

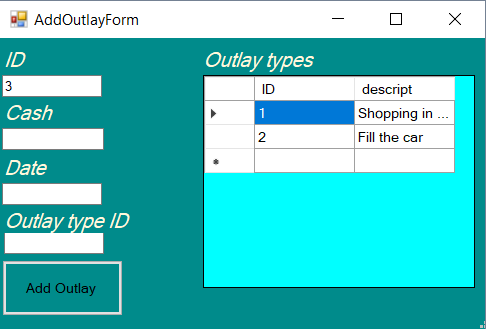


Рисунок 5.5 – Меню добавления расходов.

Как мы можем наблюдать, поле с идентификатором расхода автоматически заполняется первым доступным ID. Если же пользователь изменит эти данные таким образом, что этот ID уже будет занят другим расходом, то программа об этом оповестит.

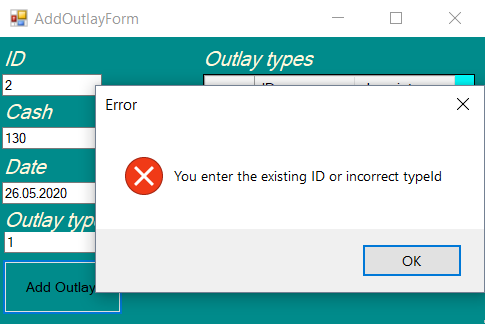


Рисунок 5.6 – Оповещение пользователя при попытке добавить расход с уже существующим идентификатором

Также протестируем поле для ввода даты расхода. Введем некорректную дату.

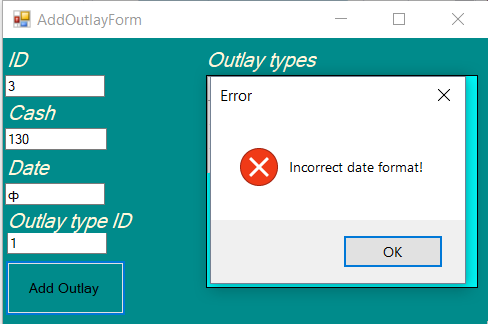


Рисунок 5.7 – Оповещение пользователя о некорректно введенной дате

6 Руководство по использованию

6.1 Системные требования

В первую очередь используемое устройство должно быть подключено к сети интернет (для полного функционирования) и находится в локальной сети с развёрнутым SQL сервером (для минимально необходимого функционирования). Так же должен быть установлен .NET Framework.

Поддерживаемые операционные системы: Windows 10, Windows 8, Windows 7, большинство версий Windows Server.

Поддерживаемая разрядность: x86 и x64.

Минимальные требования к оборудованию, обусловленные .NET Framework:

* Частота процессора 1 ГГц и более;
* Оперативная память 512 МБ;
* Место на диске 4.5 ГБ.

6.2 Использование

Для использования программы, необходимо запустить файл MoneyManager2020.exe

Программа осуществит соединение с базой данных. Далее требуется авторизоваться или зарегистрироваться, если вы еще не имеет аккаунт. После входа будет доступен весь функционал приложения.

ЗАКЛЮЧЕНИЕ

В процессе курсового проектирования было разработанно программное средство для управления расходами. Были выполнены все требования к ПС. Таким образом цель курсового проекта была достигнута.

В программном средстве был реализован следующий функционал:

* Возможность авторизации и регистрации;
* Наличие интерфейса для управления приложением;
* Возможность просматривать список ваших расходов и доходов;
* Возможность создавать план расходов по датам;
* Возможность добавления типов расходов и доходов;
* Оповещение пользователя о перерасходе средств через email;

В переспективе в систему нужно добавить клиент-серверную архитектуру, чтобы достичь большей сохранности данных и иметь возможность реализовать клиентские приложения под разные операционные системы, а так же браузер. Также ввиду темы курсового приложения, можно портировать данное приложение под системы IOS и Android.

Список использованных источников

1. Сайт документации Microsoft: свойство IDENTITY [Электронный ресурс] / Режим доступа: https://docs.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql-identity-property?view=sql-server-ver15 Дата доступа: 25.04.2020

2. Сайт документации Microsoft: System.Data пространство имен [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/api/system.data?view=netframework-4.8> Дата доступа: 18.02.2019

3. Сайт документации Microsoft: System.Data.SqlClient пространство имен [Электронный ресурс] / Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/api/system.data.sqlclient?view=netframework-4.8> Дата доступа: 18.02.2019

ПРИЛОЖЕНИЕ А

Листинг 1. Код класса Connect

public class Connect

{

private static string connectionString = @"Data Source=DESKTOP-O1NT2UF;Initial Catalog = MoneyManager; Integrated Security = True";

private static readonly Connect instance = new Connect();

private SqlConnection connection;

private Connect()

{

connection = new SqlConnection(connectionString);

}

public static Connect GetInstance() => instance;

public SqlConnection GetConnection() => connection;

public void OpenConnection()

{

if (connection.State == System.Data.ConnectionState.Closed)

{

connection.Open();

}

}

public void CloseConnection()

{

if (connection.State == System.Data.ConnectionState.Open)

{

connection.Close();

}

}

Листинг 2. Код класса Admin

public class Admin

{

private int id;

private string email;

private string password;

private Connect connect;

private SqlCommand command = new SqlCommand();

private string sqlQuery = "select \* from Admins where email = @mail and password = @pass;";

private SqlDataReader reader;

public Admin(SignInForm inForm)

{

this.connect = Connect.GetInstance();

this.email = inForm.GetEmail();

this.password = inForm.GetPassword();

this.id = GetID();

}

public int ID

{

get => this.id;

}

public string Email

{

get => this.email;

}

public string Password

{

get => this.password;

}

public int GetID()

{

int tempID;

connect.OpenConnection();

command.Connection = connect.GetConnection();

command.Parameters.Add("@mail", System.Data.SqlDbType.VarChar).Value = email;

command.Parameters.Add("@pass", System.Data.SqlDbType.VarChar).Value = password;

command.CommandText = sqlQuery;

reader = command.ExecuteReader();

while (reader.Read())

{

tempID = Convert.ToInt32(reader["ID"]);

reader.Close();

connect.CloseConnection();

return tempID;

}

return 0;

}

Листинг 3. Код класса OrdinaryUser

public class OrdinaryUser

{

private int id;

private string email;

private string password;

private double cash;

private Connect connect;

private SqlCommand command = new SqlCommand();

private string sqlQuery = "select \* from Users where email = @mail;";

private SqlDataReader reader;

public OrdinaryUser(SignInForm inForm)

{

this.connect = Connect.GetInstance();

this.email = inForm.GetEmail();

this.password = inForm.GetPassword();

this.id = GetID();

this.cash = GetCash();

}

public int ID

{

get => this.id;

}

public string Email

{

get => this.email;

}

public string Password

{

get => this.password;

}

public double Cash

{

get => this.cash;

}

public int GetID()

{

int tempID;

connect.OpenConnection();

command.Connection = connect.GetConnection();

command.Parameters.Add("@mail", System.Data.SqlDbType.VarChar).Value = email;

command.CommandText = sqlQuery;

reader = command.ExecuteReader();

while (reader.Read())

{

tempID = Convert.ToInt32(reader["ID"]);

reader.Close();

connect.CloseConnection();

return tempID;

}

reader.Close();

return 0;

}

public double GetCash()

{

double tempCash;

connect.OpenConnection();

command.Connection = connect.GetConnection();

command.CommandText = sqlQuery;

reader = command.ExecuteReader();

while(reader.Read())

{

tempCash = Convert.ToDouble(reader["cash"]);

reader.Close();

connect.CloseConnection();

return tempCash;

}

reader.Close();

return 0;

}

}

ПРИЛОЖЕНИЕ Б

Листинг 1. Класс SaltedHash

public class SaltedHash

{

public string Hash { get; private set; }

public string Salt { get; private set; }

public SaltedHash(string password)

{

var saltBytes = new byte[32];

new Random().NextBytes(saltBytes);

Salt = Convert.ToBase64String(saltBytes);

var passwordAndSaltBytes = Concat(password, saltBytes);

Hash = ComputeHash(passwordAndSaltBytes);

}

static string ComputeHash(byte[] bytes)

{

using(var sha256 = SHA256.Create())

{

return Convert.ToBase64String(bytes);

}

}

static byte[] Concat(string password, byte[] saltBytes)

{

var passwordBytes = Encoding.UTF8.GetBytes(password);

return passwordBytes.Concat(saltBytes).ToArray();

}

public static bool Verify(string salt, string hash, string password)

{

var saltBytes = Convert.FromBase64String(salt);

var passwordAndSaltBytes = Concat(password, saltBytes);

var hashAttempt = ComputeHash(passwordAndSaltBytes);

return hash == hashAttempt;

}

}

ПРИЛОЖЕНИЕ В

Листинг 1. Класс SignInForm

public partial class SignInForm : Form

{

public SignInForm()

{

InitializeComponent();

}

private void SignInForm\_Load(object sender, EventArgs e)

{

}

public string GetEmail() => EmailTextBox.Text;

public string GetPassword() => PasswordTextBox.Text;

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

{

Connect connect = Connect.GetInstance();

SqlCommand passCommand = new SqlCommand();

SqlDataReader reader;

string hash = "";

string salt = "";

string mail = "";

string passSqlQuery = "select email, password, salt from Users where email = @mail";

connect.OpenConnection();

passCommand.CommandText = passSqlQuery;

passCommand.Connection = connect.GetConnection();

passCommand.Parameters.Add("@mail", SqlDbType.VarChar).Value = EmailTextBox.Text;

reader = passCommand.ExecuteReader();

while (reader.Read())

{

mail = Convert.ToString(reader["email"]);

hash = Convert.ToString(reader["password"]);

salt = Convert.ToString(reader["salt"]);

}

reader.Close();

if(mail == EmailTextBox.Text && SaltedHash.Verify(salt, hash, PasswordTextBox.Text))

{

OrdinaryUser user = new OrdinaryUser(this);

MainMenuForm mainMenu = new MainMenuForm();

Menu menu = new Menu(mainMenu, user);

mainMenu.SetMenu(menu);

this.Hide();

MessageBox.Show("Welcome to the MoneyManager2020!", "Success");

mainMenu.Show();

connect.CloseConnection();

}

else

{

connect.CloseConnection();

if(EmailTextBox.Text.Trim().Equals(""))

{

MessageBox.Show("Enter your email", "Email field is empty", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

else if(PasswordTextBox.Text.Trim().Equals(""))

{

MessageBox.Show("Enter your password", "Password field is empty", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

else

{

MessageBox.Show("Wrong email or password", "Invalid data", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

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

{

Connect connect = Connect.GetInstance();

DataTable table = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter();

SqlCommand command = new SqlCommand();

SqlCommand checkCommand = new SqlCommand();

SqlDataReader reader;

bool isUserExist = false;

string sqlQuery = "select \* from Users where email = \'@log\' and password = \'@pass\';";

string checkQuery = "select \* from Users;";

connect.OpenConnection();

checkCommand.CommandText = checkQuery;

checkCommand.Connection = connect.GetConnection();

reader = checkCommand.ExecuteReader();

while(reader.Read())

{

if(EmailTextBox.Text == reader["email"].ToString())

{

isUserExist = true;

reader.Close();

break;

}

}

reader.Close();

command.CommandText = sqlQuery;

command.Connection = connect.GetConnection();

string pattern = @"^(?("")(""[^""]+?""@)|(([0-9a-z]((\.(?!\.))|[-!#\$%&'\\*\+/=\?\^`\{\}\|~\w])\*)(?<=[0-9a-z])@))" +

@"(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-z][-\w]\*[0-9a-z]\*\.)+[a-z0-9]{2,17}))$";

if (Regex.IsMatch(EmailTextBox.Text, pattern, RegexOptions.IgnoreCase))

{

if(PasswordTextBox.Text.Length > 4)

{

SaltedHash hashPass = new SaltedHash(PasswordTextBox.Text);

command.Parameters.Add("@mail", SqlDbType.VarChar).Value = EmailTextBox.Text;

command.Parameters.Add("@pass", SqlDbType.VarChar).Value = hashPass.Hash;

command.Parameters.Add("@salt", SqlDbType.VarChar).Value = hashPass.Salt;

adapter.SelectCommand = command;

adapter.Fill(table);

if (!isUserExist)

{

string sqlQuery1 = "insert into Users(email, password, salt) values(@mail, @pass, @salt);";

SqlCommand command1 = new SqlCommand();

command.Connection = connect.GetConnection();

command.CommandText = sqlQuery1;

command.ExecuteNonQuery();

connect.CloseConnection();

OrdinaryUser user = new OrdinaryUser(this);

MainMenuForm mainMenu = new MainMenuForm();

Menu menu = new Menu(mainMenu, user);

mainMenu.SetMenu(menu);

MessageBox.Show("Your account was signed up", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

this.Hide();

MessageBox.Show("Welcome to the MoneyManager2020!", "Success");

mainMenu.Show();

}

else

{

MessageBox.Show("User with this email is already exist!", "Used email", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("Password must be longer than 5 symbols!","Invalid password", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

else

{

MessageBox.Show("Invalid email adress!", "Invalid email", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

Connect connect = Connect.GetInstance();

DataTable table = new DataTable();

SqlDataAdapter adapter = new SqlDataAdapter();

SqlCommand command = new SqlCommand();

string sqlQuery = "select \* from Admins where email = @mail and password = @pass;";

command.CommandText = sqlQuery;

command.Connection = connect.GetConnection();

command.Parameters.Add("@mail", SqlDbType.VarChar).Value = EmailTextBox.Text;

command.Parameters.Add("@pass", SqlDbType.VarChar).Value = PasswordTextBox.Text;

adapter.SelectCommand = command;

adapter.Fill(table);

if (table.Rows.Count > 0)

{

Admin admin = new Admin(this);

AdminMenuForm menuForm = new AdminMenuForm();

AdminMenu menu = new AdminMenu(menuForm, admin);

menuForm.SetMenu(menu);

this.Hide();

MessageBox.Show("Welcome to the MoneyManager2020 admin mode!", "Success");

menuForm.Show();

connect.CloseConnection();

}

else

{

if (EmailTextBox.Text.Trim().Equals(""))

{

MessageBox.Show("Enter your email", "Email field is empty", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

else if (PasswordTextBox.Text.Trim().Equals(""))

{

MessageBox.Show("Enter your password", "Password field is empty", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

else

{

MessageBox.Show("Wrong email or password", "Invalid data", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

}

Листинг 2. Класс AddIncome

public class AddIncomeClass

{

private int userId;

private int incomeId;

private double cash;

private string date;

private int incomeTypeId;

private Menu activeMenu;

private AddIncomeForm activeForm;

private Connect connect = Connect.GetInstance();

private SqlCommand command = new SqlCommand();

public AddIncomeClass(int userId, AddIncomeForm form, Menu menu)

{

this.userId = userId;

this.activeForm = form;

this.activeMenu = menu;

this.incomeId = SetIdTextBox() + 1;

this.activeForm.IDTextBox.Text = incomeId.ToString();

}

public void ShowIncomeTypes()

{

SqlDataAdapter adapter;

DataSet dataSet;

string sqlQuery = "select \* from Income\_Types;";

adapter = new SqlDataAdapter(sqlQuery, connect.GetConnection());

dataSet = new DataSet();

connect.OpenConnection();

adapter.Fill(dataSet, "Income\_Types");

activeForm.DataGridViewIncomeTypes.DataSource = dataSet.Tables["Income\_Types"];

connect.CloseConnection();

}

public int SetIdTextBox()

{

int tempId;

SqlDataReader reader;

string sqlQuery = "select top 1 ID from Incomes order by ID desc;";

connect.OpenConnection();

command.CommandText = sqlQuery;

command.Connection = connect.GetConnection();

reader = command.ExecuteReader();

while (reader.Read())

{

tempId = Convert.ToInt32(reader["ID"]);

reader.Close();

connect.CloseConnection();

return tempId;

}

reader.Close();

return 0;

}

public void Add()

{

try

{

this.incomeId = Convert.ToInt32(activeForm.IDTextBox.Text);

this.cash = Convert.ToDouble(activeForm.CashTextBox.Text);

this.date = activeForm.DateTextBox.Text;

this.incomeTypeId = Convert.ToInt32(activeForm.IncomeTypeIDTextBox.Text);

string datePattern = @"(0?[1-9]|[12][0-9]|3[01]).(0?[1-9]|1[012]).((19|20)\d\d)";

if (Regex.IsMatch(this.date, datePattern, RegexOptions.IgnoreCase))

{

try

{

connect.OpenConnection();

string sqlQuery = "insert into Incomes(ID, userID, cash, incomeDate, incomeTypeID)" +

"values(@ID, @userID, @cash, @date, @typeId);\n" +

"update Users set Users.cash = Users.cash + @cash where ID = @userID;";

command.Parameters.Add("@ID", SqlDbType.Int).Value = this.incomeId;

command.Parameters.Add("@userID", SqlDbType.Int).Value = this.userId;

command.Parameters.Add("@cash", SqlDbType.Money).Value = this.cash;

command.Parameters.Add("@date", SqlDbType.Date).Value = this.date;

command.Parameters.Add("@typeID", SqlDbType.Int).Value = this.incomeTypeId;

command.Connection = connect.GetConnection();

command.CommandText = sqlQuery;

command.ExecuteNonQuery();

MessageBox.Show("Your income was succesful added", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

command.Parameters.Clear();

connect.CloseConnection();

activeMenu.Cash = activeMenu.GetCash();

activeMenu.ShowCash();

activeMenu.LastIncome = activeMenu.GetLastIncome();

activeMenu.LastIncomeDate = activeMenu.GetLastIncomeDate();

activeMenu.ShowLastIncome();

activeMenu.ShowLastIncomeDate();

}

catch(SqlException e)

{

MessageBox.Show("You enter the existing ID or incorrect typeId", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

else

{

MessageBox.Show("Incorrect date format!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

catch (FormatException e)

{

MessageBox.Show("Some field are empty or has incorrect format", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

}

Листинг 3. Класс PlanClass

public class PlanClass

{

private int id;

private int userId;

private double cash;

private int typeId;

private string date;

private Menu activeMenu;

private PlanForm activeForm;

private Connect connect = Connect.GetInstance();

private SqlCommand command = new SqlCommand();

public Menu ActiveMenu

{

get => this.activeMenu;

}

public PlanClass(int userId, Menu menu, PlanForm form)

{

this.userId = userId;

this.activeMenu = menu;

this.activeForm = form;

}

public void ShowTypes()

{

SqlDataAdapter adapter;

DataSet dataSet;

string sqlQuery = "select \* from Outlay\_Types";

adapter = new SqlDataAdapter(sqlQuery, connect.GetConnection());

dataSet = new DataSet();

connect.OpenConnection();

adapter.Fill(dataSet, "Outlay\_Types");

activeForm.DataGridViewTypes.DataSource = dataSet.Tables["Outlay\_Types"];

connect.CloseConnection();

}

public void ShowPlan()

{

SqlDataAdapter adapter;

DataSet dataSet;

string sqlQuery = "select OutlayPlan.ID, Outlay\_Types.descript, OutlayPlan.cash, OutlayPlan.planDate " +

"from OutlayPlan inner join Outlay\_Types on OutlayPlan.OutlayTypeID = Outlay\_Types.ID " +

"and OutlayPlan.userID = " + this.userId + ";";

adapter = new SqlDataAdapter(sqlQuery, connect.GetConnection());

dataSet = new DataSet();

connect.OpenConnection();

adapter.Fill(dataSet, "OutlayPlan");

activeForm.DataGridViewPlan.DataSource = dataSet.Tables["OutlayPlan"];

connect.CloseConnection();

}

public void Add()

{

try

{

string datePattern = @"(0?[1-9]|[12][0-9]|3[01]).(0?[1-9]|1[012]).((19|20)\d\d)";

this.id = Convert.ToInt32(activeForm.IDTextBox.Text);

this.cash = Convert.ToDouble(activeForm.CashTextBox.Text);

this.date = activeForm.DateTextBox.Text;

this.typeId = Convert.ToInt32(activeForm.OutlayTypeIDTextBox.Text);

if (Regex.IsMatch(this.date, datePattern, RegexOptions.IgnoreCase))

{

try

{

connect.OpenConnection();

string sqlQuery = "insert into OutlayPlan(ID, userID, cash, planDate, outlayTypeID)" +

"values(@ID, @userID, @cash, @date, @typeID);";

command.Parameters.Add("@ID", SqlDbType.Int).Value = this.id;

command.Parameters.Add("@userID", SqlDbType.Int).Value = this.userId;

command.Parameters.Add("@cash", SqlDbType.Money).Value = this.cash;

command.Parameters.Add("@date", SqlDbType.Date).Value = this.date;

command.Parameters.Add("@typeID", SqlDbType.Int).Value = this.typeId;

command.Connection = connect.GetConnection();

command.CommandText = sqlQuery;

command.ExecuteNonQuery();

MessageBox.Show("New outlay was successfuly added into plan", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

command.Parameters.Clear();

connect.CloseConnection();

ShowPlan();

activeMenu.ShowPlanTable();

}

catch (SqlException e)

{

MessageBox.Show("You enter the existing ID or incorrect type id", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

else

{

MessageBox.Show("Incorrect date format!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

catch (FormatException e)

{

MessageBox.Show("Some field are empty or has incorrect format", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

public void Delete()

{

try

{

this.id = Convert.ToInt32(activeForm.IDTextBox.Text);

try

{

connect.OpenConnection();

string sqlQuery = "delete from OutlayPlan where ID = @ID;";

command.Parameters.Add("@ID", SqlDbType.Int).Value = this.id;

command.Connection = connect.GetConnection();

command.CommandText = sqlQuery;

command.ExecuteNonQuery();

MessageBox.Show("Outlay was successfuly deleted into plan", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

command.Parameters.Clear();

connect.CloseConnection();

ShowPlan();

activeMenu.ShowPlanTable();

}

catch (SqlException e)

{

MessageBox.Show("You enter the existing ID or incorrect type id", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

catch (FormatException e)

{

MessageBox.Show("Some field are empty or has incorrect format", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

public void Update()

{

try

{

string datePattern = @"(0?[1-9]|[12][0-9]|3[01]).(0?[1-9]|1[012]).((19|20)\d\d)";

this.id = Convert.ToInt32(activeForm.IDTextBox.Text);

this.cash = Convert.ToDouble(activeForm.CashTextBox.Text);

this.date = activeForm.DateTextBox.Text;

this.typeId = Convert.ToInt32(activeForm.OutlayTypeIDTextBox.Text);

if (Regex.IsMatch(this.date, datePattern, RegexOptions.IgnoreCase))

{

try

{

connect.OpenConnection();

string sqlQuery = "update OutlayPlan set cash = @cash, planDate = @date, OutlayTypeID = @typeID where ID = @ID";

command.Parameters.Add("@ID", SqlDbType.Int).Value = this.id;

command.Parameters.Add("@cash", SqlDbType.Money).Value = this.cash;

command.Parameters.Add("@date", SqlDbType.Date).Value = this.date;

command.Parameters.Add("@typeID", SqlDbType.Int).Value = this.typeId;

command.Connection = connect.GetConnection();

command.CommandText = sqlQuery;

command.ExecuteNonQuery();

MessageBox.Show("Outay was successfuly updated in plan", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);

command.Parameters.Clear();

connect.CloseConnection();

ShowPlan();

activeMenu.ShowPlanTable();

}

catch (SqlException e)

{

MessageBox.Show("You enter the existing ID or incorrect type id", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

}

catch (FormatException e)

{

MessageBox.Show("Some field are empty or has incorrect format", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);

command.Parameters.Clear();

}

}

}