Министерство образования Республики Беларусь

Учреждение образования «Белорусский государственный университет

информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

Дисциплина: Конструирование программ и языки программирования

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

БАНКОВСКИЕ УСЛУГИ

БГУИР КП 1-58 01 01 25 ПЗ

Студент: гр. 110901 Русак А. Д.

Руководитель: ассистент кафедры ЭВМ   
Яночкин А. Л.

Минск 2013

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 5](#_Toc346060832)

1 Обзор МЕТОДОВ И АЛГОРИТМОВ ПОСТАВЛЕННОЙ ЗАДАЧИ 6

[1.1 Задание: 6](#_Toc346060833)

[1.2 Обзор методов и алгоритмов решения поставленной задачи: 6](#_Toc346060834)

[2 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ 7](#_Toc346060835)

[2.1 Обоснование выбора языка программирования: 7](#_Toc346060836)

[2.2 Описание инструментов разработки программы: 7](#_Toc346060837)

[3 РАЗРАБОТКА ПРОГРАММЫ: 10](#_Toc346060838)

[3.1 Общее описание: 10](#_Toc346060839)

[3.2 Описание использованных структур данных: 10](#_Toc346060840)

[4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ](#_Toc346060841) 14

[5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ](#_Toc346060842) 15

[5.1 Требования к программному и аппаратному обеспечению:](#_Toc346060843) 15

[ЗАКЛЮЧЕНИЕ](#_Toc346060845) 20

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc346060846) 21

[ПРИЛОЖЕНИЕ А- листинг программы](#_Toc346060847) 22

# 

# ВВЕДЕНИЕ

Сегодня на дворе 21 век - век информационных технологий, которые стремительно развиваются каждую секунду нашей жизни. Технологический, а также информационный прогресс, особенно за последние годы, сделал огромный шаг вперед. Современное человечество быстро превращается в информационное общество, а в особенности это происходит в быстроразвивающихся странах, которые делают большой упор именно на развитие техники и информационных технологий.

Почти к каждой семье имеется персональный компьютер потенциал которого огромен, использовать его возможно в различных сферах жизнедеятельности: учеба, развлечение, общение.

Также информационные технологии активно используются в сферах банковской деятельности. Пластиковые карточки, счета, виртуальные деньги. За всем этим стоит информационная система, разработанная программистами.

**1 ОБЗОР МЕТОДОВ И АЛГОРИТМОВ ПОСТАВЛЕННОЙ ЗАДАЧИ**

## 1.1 Задание

Написать программу реализации банковских услуг. Программа должна иметь удобный пользовательский интерфейс с необходимыми пунктами меню. Работа с информацией должна производиться в окнах. В программе должна быть предусмотрена возможность создания информации в виде файлов. Программа позволяет организовать открытие нового счета, аннулирование счета, перевод денег с одного счета на другой, снятие денег со счета, депозиты и займ под проценты. Все данные содержатся в файлах. Банковский счет связывается с именем вкладчика. Необходимо реализовать оконный интерфейс при помощи ассемблерных вставок. Или реализовать ввод данных через собственные функции, реализованные на ассемблере.

## 1.2 Обзор методов и алгоритмов решения поставленной задачи

В программе используется база данных. Средство управления базами данных (СУБД) – Microsoft SQL Server 2012. Взаимодействие происходит c помощью ORM Entity Framework. Был выбран подход Code First.

Проектирование производилось на основе архитектуры DDD(Domain-Driven Design).

# 2 ОБОСНОВАНИЕ ВЫБРАННЫХ МЕТОДОВ

## 2.1 Обоснование выбора языка программирования

Для решения поставленной задачи была выбрана интегрированная среда разработки Microsoft Visual Studio 2013.

- в ней доступны широкие возможности языка C#

- используется широким кругом программистов-профессионалов, разрабатывающих в ОС Windows и Windows Phone

- она имеет все необходимые средства и инструменты для разработки приложений любой сложности

- удобна в использовании, гибка в настройке

- создает очень эффективный код

- в ней имеется возможность использования готовых библиотек классов

- имеет встроенный механизм обмена данными по сети при помощи socket-ов

Так как ИСР выбрана Microsoft Visual Studio 2013, соответственно языком программирования выбран C#, поскольку:

- подлинная объектная ориентированность (всякая языковая сущность претендует на то, чтобы быть объектом)

- событийно-ориентированное программирование

- безопасный код (managed code)

## 2.2 Описание инструментов разработки программы

## В середине 2000 года компания Microsoft объявила о работе над новой платформой для создания программ, которая получила имя *платформа .NET* (.NET Framework). Платформа .NET образует каркас, включающий среду исполнения, библиотеку классов и набор технологий для построения приложений и служб. Основной инструмент разработки для платформы .NET – интегрированная среда Microsoft Visual Studio.

Microsoft Visual Studio 2013 – это передовое решение для разработки, позволяющее командам любого размера проектировать и создавать привлекательные приложения, которые удовлетворят самым взыскательным требованиям заказчиков. Вы можете использовать инструменты гибкого планирования — такие как планирование объема работ, панели задач и управление невыполненной работой — для внедрения методов последовательной разработки и применения гибких методологий в удобном для вас темпе. Используйте расширенные средства моделирования, обнаружения и проектирования архитектуры, чтобы описать свою систему и обеспечить полную реализацию вашей концепции архитектуры. Добейтесь взаимодействия между командами разработчиков и операторов с помощью таких программных средств, как IntelliTrace, Operations Manager Connector и Preemptive Analytics. Повышайте качество и сокращайте время устранения неполадок путем создания ошибок из развернутого программного обеспечения, включающих конкретный перечень действий, и обеспечивайте взаимодействие с операторами для получения данных, которые позволят разработчикам более детально анализировать производственные проблемы. Одним словом, вы можете создавать высококачественные инновационные решения, снижая при этом расходы на разработку.

Основой платформы .NET является *общеязыковая среда исполнения* (Common Language Runtime, CLR). CLR работает как «прослойка» между опера-ционной системой и программой для платформы .NET. Каждая программа для .NET состоит из одной или нескольких сборок. *Сборка* (assembly) является ре-зультатом компиляции исходных текстов на некотором языке программирования для платформы .NET и содержит метаданные и код на Common Intermediate Lan-guage. *Метаданные* – это информационные таблицы с полным описанием всех типов, размещённых в сборке. *Common Intermediate Language* (CIL или IL) – внутренний язык платформы .NET, он не зависит от типа процессора. В процессе работы программы CIL компилируется в машинный код специальным *JIT-ком-пилятором* (Just-in-Time compiler).

В состав платформы .NET входит обширная библиотека классов *Framework Class Library* (FCL). Частью этой библиотеки является базовый набор классов *Base Class Library* (BCL), в который входят классы для работы со строками и коллекциями данных, для поддержки многопоточности и множество других классов. В FCL также входят компоненты, поддерживающие различные технологии обработки данных и организации взаимодействия с пользователем. Это классы для работы с XML и базами данных, для создания пользовательских интерфейсов.

В стандартную поставку платформы .NET включено несколько компиляторов. Это компиляторы языков C#, F#, Visual Basic .NET, C++/CLI. Благодаря открытым спецификациям компиляторы для .NET предлагаются различными сторонними производителями. Необходимо подчеркнуть, что любой язык для платформы .NET является верхним элементом архитектуры. Имена элементов библиотеки FCL не зависят от языка программирования. Специфичной частью языка остаётся только синтаксис. Этот факт упрощает межъязыковое взаимодействие, перевод текста программы с одного языка на другой. Конечно, в синтаксисе любого языка программирования для .NET неизбежно находит своё отражение тесная связь с CLR.

**• Единая модель программирования для всех приложений** - При создании приложений ранее разработчикам приходилось использовать различные приемы программирования, которые существенным образом зависели от типа приложения — технологии разработки клиентского программного обеспечения, общедоступных веб-приложений, программного обеспечения для мобильных устройств и бизнес-логики промежуточного уровня значительно различались между собой. Интегрированная среда разработки Microsoft Visual Studio 2013 решает данную проблему, предоставляя в распоряжение разработчиков единую модель создания приложений всех категорий. Эта интегрированная модель обладает привычным и одновременно интуитивно понятным интерфейсом, позволяя разработчикам использовать свои навыки и знания для эффективного создания широкого спектра приложений.

**• Всесторонняя поддержка жизненного цикла разработки** – интегрированная среда Microsoft Visual Studio 2013 обеспечивает поддержку всего жизненного цикла разработки: начиная с планирования и проектирования через разработку и тестирование и вплоть до развертывания и последующего управления. Microsoft Visual Studio 2013 предоставляет всестороннюю адаптируемую среду для создания всех приложений, жизненно необходимых для успешной работы современных компаний.

**• Описание использованных технологий. Язык программирования C#.**

Специально для платформы .NET был разработан новый язык программирования C#. Этот язык сочетает простой синтаксис, похожий на синтаксис языков C++ и Java, и полную поддержку всех современных объектно-ориентирован-ных концепций и подходов. В качестве ориентира при разработке языка было выбрано безопасное программирование, нацеленное на создание надёжного и простого в сопровождении кода. Здесь и далее рассматривается синтаксис пятой версии языка C#, доступной в составе .NET Framework 4.5.

**C#** (произносится *си шарп*) — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет статическую типизацию, поддерживает полиморфизм, перегрузку операторов (в том числе операторов явного и неявного приведения типа), делегаты, атрибуты, события, свойства, обобщённые типы и методы, итераторы, анонимные функции с поддержкой замыканий, LINQ, исключения, комментарии в формате XML.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и в особенности Java — С#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

# 3 ОПИСАНИЕ ПРОГРАММЫ ДЛЯ ПРОГРАММИСТА

# 

## 3.1 Общее описание

## 

Проектирование программы производилось на основе архитектуры DDD (Domain-Driven Design).

Используется подход Entity Framework Code First. Используя этот подход, сначала описываются POCO (Playing old c-sharp classes) классы, которые являются сущностями. Сущности связаны между собой. По этим сущностям Entity Framework генерирует базу данных с соответствующими связями.

В программе 5 проектов.

1. CourseProject.Core – содержит описание доменных сущностей, интерфейсов репозиториев, интерфейса UnitOfWork.
2. CourseProject.EFData – содержит реализацию репозиториев, настройки для базы данных, реализацию UnitOfWork.
3. CourseProject.Services – содержит в себе сервисы, которые работают с репозиториями, на этом уровне производится валидация данных.
4. CourseProject.UIBank – графический интерфейс программы, на этом уровне используются сервисы.
5. CourseProject.Infrastructure – содержит инфраструктуру проекта.

Информация выводится на компонент DataGridView.

## 3.2 Описание использованных структур данных

Таблица 3.1 – Поля класса Account

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | Int | Уникальный идентификатор |
| CreateDate | DateTime | Дата создания |
| Summary | Double | Сумма счёта |
| Customer | Customer | Владелец |
| CustomerId | Int | Уникальный идентификатор владельца |
| Currency | Currency | Валюта |
| CurrencyId | Int | Уникальный идентификатор валюты |

Таблица 3.2 – Поля класса Currency

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | Int | Уникальный идентификатор |
| Value | String | Значение |

Таблица 3.3 – Поля класса Customer

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | int | Уникальный идентификатор |
| Name | string | Имя |
| Surname | string | Фамилия |
| Patronymic | string | Отчество |
| BirthDate | DateTime | Дата рождения |
| Address | string | Адрес |
| Age | int | Возраст |
| PassportData | string | Паспортные данные |
| Accounts | Account | Счета владельца |
| Deposits | Deposit | Вклады владельца |
| Loans | Loan | Займы владельца |

Таблица 3.4 – Поля класса Deposit

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | int | Уникальный идентификатор |
| Summary | double | Сумма вклада |
| OpeningSummary | double | Начальная сумма вклада |
| CreateDate | DateTime | Дата открытия |
| Customer | Customer | Владелец |
| CustomerId | int | Уникальный идентификатор владельца |
| Currency | Currency | Валюта |
| CurrencyId | int | Уникальный идентификатор валюты |
| OptionDeposit | OptionDeposit | Условия депозита |
| OptionDepositId | int | Уникальный идентификатор условий |

Таблица 3.5 – Поля класса Loan

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | int | Уникальный идентификатор |
| Summary | double | Сумма вклада |
| OpeningSummary | double | Начальная сумма займа |
| CreateDate | DateTime | Дата открытия |
| Customer | Customer | Владелец |
| CustomerId | int | Уникальный идентификатор владельца |
| Currency | Currency | Валюта |
| CurrencyId | int | Уникальный идентификатор валюты |
| OptionLoan | OptionLoan | Условия займа |
| OptionLoanId | int | Уникальный идентификатор условий |

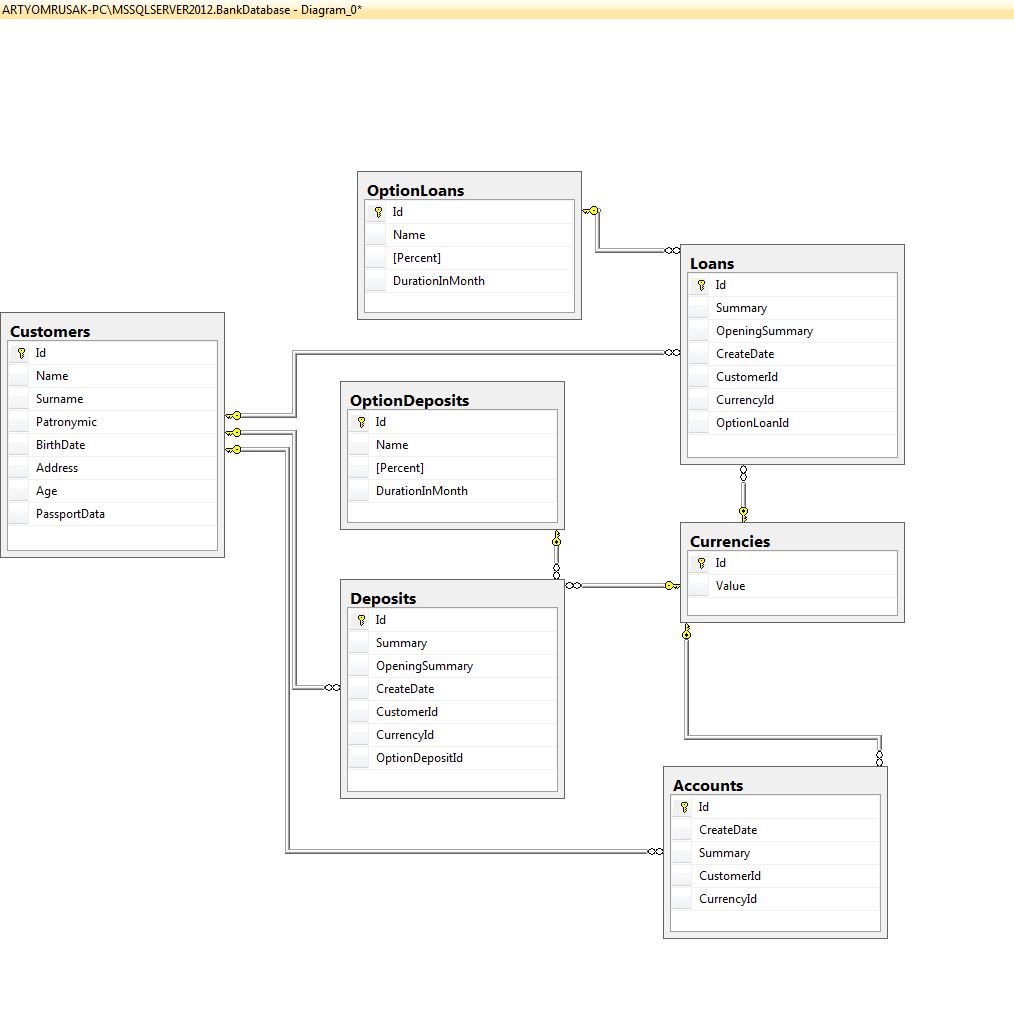
Таблица 3.6 – Поля класса OptionDeposit

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | int | Уникальный идентификатор |
| Name | string | Имя |
| Percent | int | Проценты |
| DurationInMonth | int | Длительность в месяцах |

Таблица 3.7 – Поля класса OptionLoan

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| Id | int | Уникальный идентификатор |
| Name | string | Имя |
| Percent | int | Проценты |
| DurationInMonth | int | Длительность в месяцах |

**3.3 Схема базы данных**

****

# 4 ОПИСАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

**4.1 Рассмотрим алгоритм добавления владельца**

Шаг 1: Создаём форму. При создании формы создаётся объект класса BankContext.

Шаг 2: Создаём UnitOfWork (единицу работы).

Шаг 3: Проверяем введённые данные, если владелец с такими паспортными данными существует, уведомляем об этом.

Шаг 4: Сохраняем изменения или не сохраняем.

Шаг 5: Удаляем объект класса BankContext, при этом закрывается соединение с базой данных.

**4.2 Рассмотрим алгоритм добавления счёта, вклада, займа**

Шаг 1: Создаём форму. При создании формы создаётся объект класса BankContext.

Шаг 2: Создаём UnitOfWork (единицу работы).

Шаг 3: Проверяем введённые данные, если номер паспорта существует в базе данных, то создаём счёт. Если нет, то переводим на форму создания владельца, затем создаём счёт.

Шаг 4: Сохраняем изменения или не сохраняем.

Шаг 5: Удаляем объект класса BankContext, при этом закрывается соединение с базой данных.

# 5 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Внимание! Для корректной работы программы необходимо, чтобы в вашей системе были установлено средства .NET Framework 4.5 или более поздняя версия. Если данный компонент не установлен у Вас в системе, то его можно найти на официальном сайте Microsoft.

Программа состоит из одного \*.exe файла самой программы и базы данных. Главное окно. На нём есть кнопка добавления пользователя, счёта, вклада, займа. Здесь же производится поиск владельцев. Они выводятся в таблице. По двойному клику можно зайти в управление владельцем.

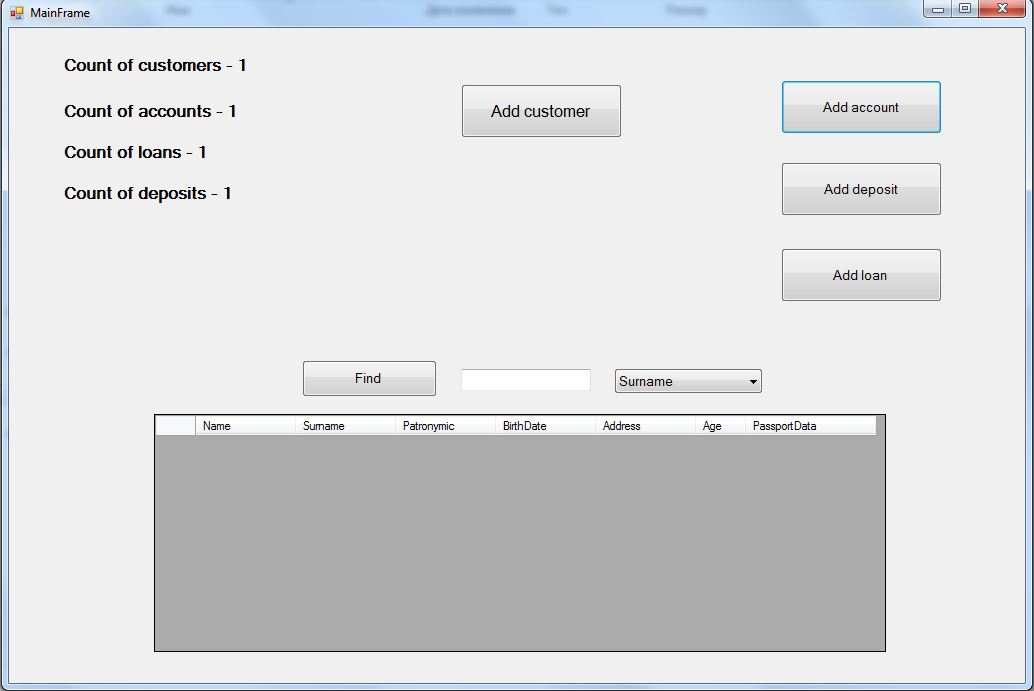


Рисунок 5.1 – Главное окно

Окно добавления пользователя. Введённая информация проверяется и создаётся запись в базе данных, после нажатия кнопки “Add customer”.

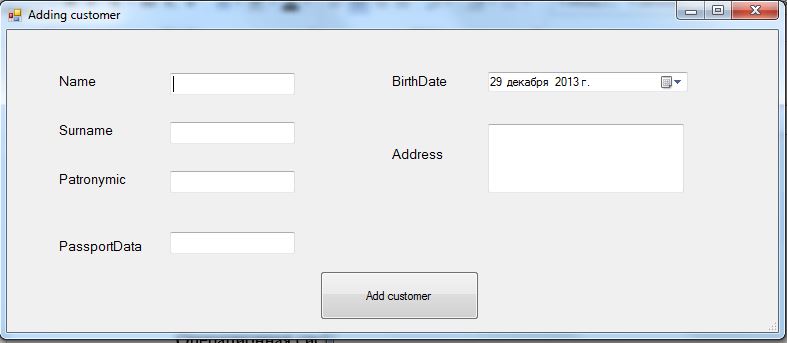


Рисунок 5.2 – Окно добавления пользователя

Окно добавления счёта. Введённая информация проверяется и создаётся запись в базе данных, после нажатия кнопки “Create”.

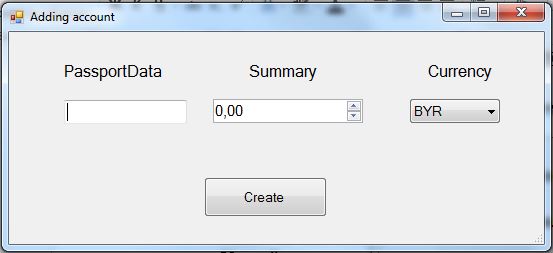


Рисунок 5.3 – Окно добавления счёта

Окно добавления вклада. Введённая информация проверяется и создаётся запись в базе данных, после нажатия кнопки “Create”.

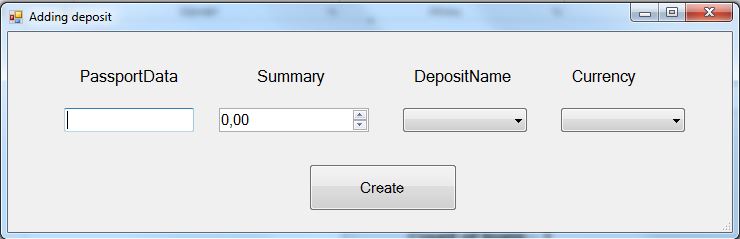


Рисунок 5.4 – Окно добавления вклада

Окно добавления займа. Введённая информация проверяется и создаётся запись в базе данных, после нажатия кнопки “Create”.

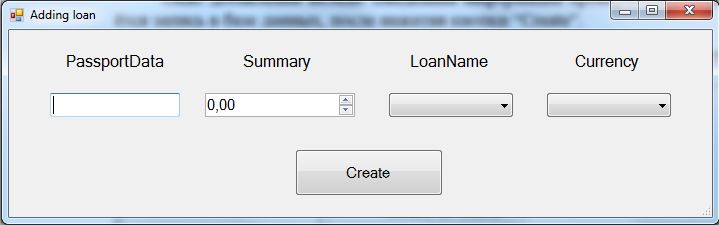


Рисунок 5.4 – Окно добавления займа

Окно управления владельцем. Здесь можно просмотреть счета, вклады, займы владельца, его паспортные данные, возраст, адрес и др.

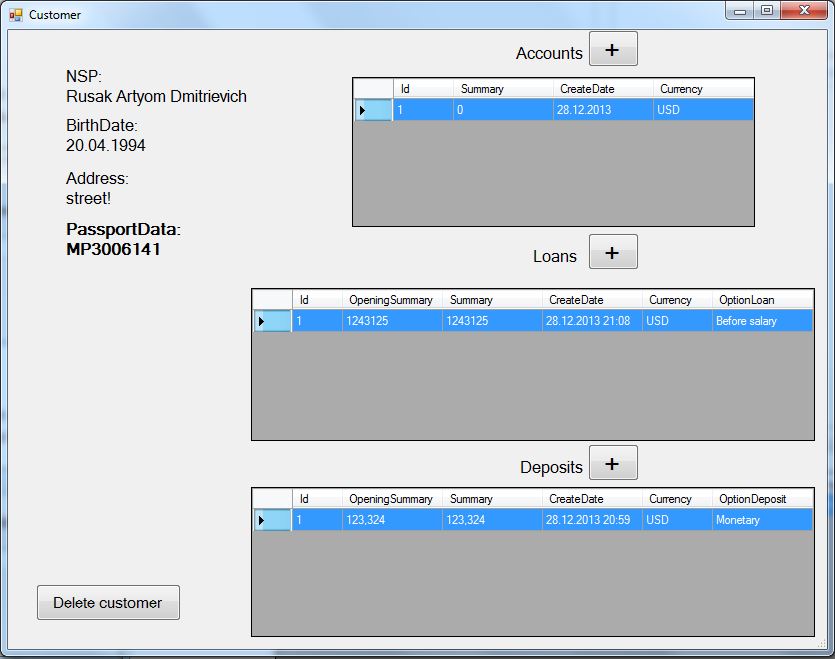


Рисунок 5.5 – Окно управления владельцем

Окно управления счётом. Здесь можно произвести транзакции между счетами, пополнить счёт, снять деньги со счёта, аннулировать счёт.

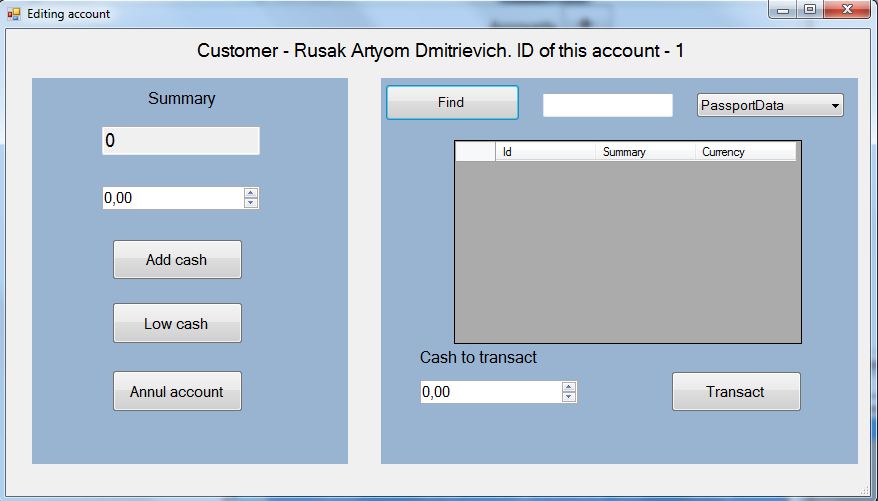


Рисунок 5.6 – Окно управления счётом

ЗАКЛЮЧЕНИЕ

В результате курсовой работы создана программа «Банковские услуги». Программа позволяет добавлять счета, вклады, займы, владельцев, осушествлять поиск данных по различным параметрам. Данные загружаются вручную пользователем, сохранение производится с помощью соответствующего пункта меню.

Программный продукт выполнен в интегрированной среде разработки Microsooft Visual Studio 2013 с использованием СУБД Microsoft SQL Server 2012 и ORM – Entity Framework.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Рихтер Дж., CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 4-е изд. — СПб.: Питер, 2013. — 896 с.: ил. — (Серия «Мастер-класс»).

[2]  Эндрю Троелсен, Язык программирования C# 5.0 и платформа .NET / Э. Троелсен; . Пер. с англ. – СПб. : BHV, 2012. – 1098 с, 6-е издание.

[3] [Электроный ресурс]. - Электронный ресурс. – Режим доступа: http://msdn.com

# ПРИЛОЖЕНИЕ А - листинг программы

**IRepository.cs**

namespace CourseProject.Core.InterfaceRepository

{

public interface IRepository

{

}

}

**IRepository`1.cs**

using System;

using System.Linq;

using System.Linq.Expressions;

namespace CourseProject.Core.InterfaceRepository

{

public interface IRepository<TEntity> where TEntity : Entity

{

void Create(TEntity value);

void Update(TEntity value);

void Remove(TEntity value);

TEntity GetEntityById(int id);

TEntity Find(Expression<Func<TEntity, bool>> predicate);

IQueryable<TEntity> All();

IQueryable<TEntity> Filter(Expression<Func<TEntity, bool>> predicate);

}

}

**IUnitOfWork.cs**

using System;

namespace CourseProject.Core

{

public interface IUnitOfWork : IDisposable

{

void Commit();

void Rollback();

void PreSave();

}

}

**UnitOfWork.cs**

using System;

using System.Data.Entity;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Core.InterfaceRepository;

using CourseProject.EFData.DBContext;

using CourseProject.EFData.Repositories;

namespace CourseProject.EFData

{

public class UnitOfWork : IUnitOfWork, IRepositoryFactory

{

private readonly BankContext \_context;

private readonly DbContextTransaction \_transaction;

private IRepository<Account> \_accountRepository;

private IRepository<Currency> \_currencyRepository;

private IRepository<Customer> \_customerRepository;

private IRepository<Deposit> \_depositRepository;

private IRepository<History> \_historyRepository;

private IRepository<Loan> \_loanRepository;

private IRepository<OptionDeposit> \_optionDepositRepository;

private IRepository<OptionLoan> \_optionLoanRepository;

private bool \_disposed;

private bool \_isTransactionActive;

public UnitOfWork(BankContext context)

{

\_context = context;

\_transaction = \_context.Database.BeginTransaction();

\_isTransactionActive = true;

}

public UnitOfWork()

{

\_context = new BankContext();

\_transaction = \_context.Database.BeginTransaction();

}

public IRepository<Account> GetAccountRepository()

{

return \_accountRepository ?? (\_accountRepository = new Repository<Account>(\_context));

}

public IRepository<Currency> GetCurrencyRepository()

{

return \_currencyRepository ?? (\_currencyRepository = new Repository<Currency>(\_context));

}

public IRepository<Customer> GetCustomerRepository()

{

return \_customerRepository ?? (\_customerRepository = new Repository<Customer>(\_context));

}

public IRepository<Deposit> GetDepositRepository()

{

return \_depositRepository ?? (\_depositRepository = new Repository<Deposit>(\_context));

}

public IRepository<Loan> GetLoanRepository()

{

return \_loanRepository ?? (\_loanRepository = new Repository<Loan>(\_context));

}

public IRepository<OptionDeposit> GetOptionDepositRepository()

{

return \_optionDepositRepository ?? (\_optionDepositRepository = new Repository<OptionDeposit>(\_context));

}

public IRepository<OptionLoan> GetOptionLoanRepository()

{

return \_optionLoanRepository ?? (\_optionLoanRepository = new Repository<OptionLoan>(\_context));

}

#region Implementation of IDisposable

public void Dispose()

{

if (\_isTransactionActive)

{

try

{

\_context.SaveChanges();

\_transaction.Commit();

\_isTransactionActive = false;

}

catch (Exception e)

{

\_transaction.Rollback();

\_context.Dispose();

\_disposed = true;

\_isTransactionActive = false;

throw new RepositoryException(e);

}

}

if (!\_disposed)

{

\_context.Dispose();

\_disposed = true;

}

}

#endregion

#region Implementation of IUnitOfWork

public void Commit()

{

try

{

if (\_isTransactionActive && !\_disposed)

{

\_context.SaveChanges();

\_transaction.Commit();

\_isTransactionActive = false;

}

}

catch (Exception e)

{

\_transaction.Rollback();

\_isTransactionActive = false;

throw new RepositoryException(e.Message);

}

}

public void Rollback()

{

if (\_isTransactionActive && !\_disposed)

{

\_transaction.Rollback();

\_isTransactionActive = false;

}

}

public void PreSave()

{

\_context.SaveChanges();

}

#endregion

}

}

**Repository.cs**

using CourseProject.Core.InterfaceRepository;

using CourseProject.EFData.DBContext;

namespace CourseProject.EFData.Repositories

{

public class Repository : IRepository

{

protected BankContext Context;

public Repository(BankContext context)

{

Context = context;

}

}

}

**Repository`1.cs**

using System;

using System.Data.Entity;

using System.Linq;

using System.Linq.Expressions;

using CourseProject.Core;

using CourseProject.Core.Exceptions;

using CourseProject.Core.InterfaceRepository;

using CourseProject.EFData.DBContext;

using CourseProject.Infrastructure.Guard.Validation;

namespace CourseProject.EFData.Repositories

{

public class Repository<TEntity> : Repository, IRepository<TEntity> where TEntity : Entity

{

#region [Private members]

private readonly DbSet<TEntity> \_entities;

#endregion

#region [Ctor's]

public Repository(BankContext context)

: base(context)

{

\_entities = Context.Set<TEntity>();

}

#endregion

#region Implementation of IRepository<TEntity>

public void Create(TEntity value)

{

Guard.AgainstNullReference(value, "value");

\_entities.Add(value);

}

public void Update(TEntity value)

{

Guard.AgainstNullReference(value, "value");

\_entities.Attach(value);

Context.Entry(value).State = EntityState.Modified;

}

public void Remove(TEntity value)

{

Guard.AgainstNullReference(value, "value");

\_entities.Remove(value);

}

public TEntity GetEntityById(int id)

{

Guard.AgainstNullReference(id, "id");

try

{

return \_entities.Find(id);

}

catch (Exception ex)

{

throw new RepositoryException(ex.Message);

}

}

public TEntity Find(Expression<Func<TEntity, bool>> predicate)

{

Guard.AgainstNullReference(predicate, "predicate");

try

{

return \_entities.Where(predicate).SingleOrDefault();

}

catch (Exception e)

{

throw new RepositoryException(e.Message);

}

}

public IQueryable<TEntity> All()

{

return \_entities;

}

public IQueryable<TEntity> Filter(Expression<Func<TEntity, bool>> predicate)

{

Guard.AgainstNullReference(predicate, "predicate");

return \_entities.Where(predicate);

}

#endregion

}

}

**BankContext.cs**

using System.Data.Entity;

using CourseProject.Core.Entities;

using CourseProject.EFData.DBContext.Configurations;

namespace CourseProject.EFData.DBContext

{

public class BankContext : DbContext

{

public DbSet<Account> Accounts { get; set; }

public DbSet<Deposit> Deposits { get; set; }

public DbSet<Loan> Loans { get; set; }

public DbSet<Customer> Customers { get; set; }

public DbSet<History> Histories { get; set; }

public DbSet<Currency> Currencies { get; set; }

public DbSet<OptionDeposit> OptionDeposits { get; set; }

public DbSet<OptionLoan> OptionLoans { get; set; }

public BankContext()

{

}

public BankContext(string connectionString) : base(connectionString)

{

}

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

modelBuilder.Configurations.Add(new AccountConfiguration());

modelBuilder.Configurations.Add(new CurrencyConfiguration());

modelBuilder.Configurations.Add(new CustomerConfiguration());

modelBuilder.Configurations.Add(new DepositConfiguration());

modelBuilder.Configurations.Add(new HistoryConfiguration());

modelBuilder.Configurations.Add(new OptionDepositConfiguration());

modelBuilder.Configurations.Add(new OptionLoanConfiguration());

modelBuilder.Configurations.Add(new LoanConfiguration());

base.OnModelCreating(modelBuilder);

}

}

}

**BankInitialization.cs**

using System;

using System.Data.Entity;

using System.Transactions;

using CourseProject.Core.Entities;

namespace CourseProject.EFData.DBContext.Initializers

{

public class BankInitialization : IDatabaseInitializer<BankContext>

{

public void InitializeDatabase(BankContext context)

{

bool databaseExists;

using (new TransactionScope(TransactionScopeOption.Suppress))

{

databaseExists = context.Database.Exists();

}

if (databaseExists)

{

if (context.Database.CompatibleWithModel(true))

{

return;

}

context.Database.Delete();

}

context.Database.Create();

context.Database.ExecuteSqlCommand("ALTER TABLE Customers ADD CONSTRAINT PassportDataUnique UNIQUE (PassportData)");

context.Database.ExecuteSqlCommand("ALTER TABLE OptionDeposits ADD CONSTRAINT NameDataUnique UNIQUE (Name)");

context.Database.ExecuteSqlCommand("ALTER TABLE Currencies ADD CONSTRAINT ValueDataUnique UNIQUE (Value)");

try

{

Seed(context);

context.SaveChanges();

}

catch (Exception ex)

{

context.Dispose();

throw;

}

}

public void Seed(BankContext context)

{

var byr = new Currency { Value = "BYR" };

var usd = new Currency { Value = "USD" };

context.Currencies.Add(byr);

context.Currencies.Add(usd);

var sber = new OptionDeposit { Name = "Saving", Percent = 3, DurationInMonth = 12 };

var orig = new OptionDeposit { Name = "Original", Percent = 5, DurationInMonth = 6 };

var money = new OptionDeposit { Name = "Monetary", Percent = 9, DurationInMonth = 3 };

context.OptionDeposits.Add(sber);

context.OptionDeposits.Add(orig);

context.OptionDeposits.Add(money);

var beforeSalary = new OptionLoan { Name = "Before salary", Percent = 1, DurationInMonth = 12 };

var loan = new OptionLoan { Name = "Loan at interest", Percent = 4, DurationInMonth = 6 };

context.OptionLoans.Add(beforeSalary);

context.OptionLoans.Add(loan);

context.SaveChanges();

}

}

}

**AccountService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class AccountService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public AccountService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public Account CreateAccount(double summary, int customerId, int currencyId)

{

Guard.AgainstNegatives((int) summary);

var account = new Account

{

CreateDate = DateTime.Now,

Summary = summary,

CustomerId = customerId,

CurrencyId = currencyId

};

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

accountRepository.Create(account);

try

{

\_unitOfWork.PreSave();

}

catch (Exception exception)

{

throw new ServiceException(exception);

}

return account;

}

public void UpdateAccount(Account account)

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

try

{

accountRepository.Update(account);

}

catch (Exception e)

{

throw new AccountServiceException(e);

}

}

public void RemoveAccount(Account account)

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

accountRepository.Remove(account);

}

public Account GetAccountById(int accountId)

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

try

{

return accountRepository.GetEntityById(accountId);

}

catch (RepositoryException e)

{

throw new AccountServiceException(e.Message);

}

}

public List<Account> GetAccountsByPassportData(string passportData)

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

return accountRepository.Filter(e => e.Customer.PassportData == passportData).ToList();

}

public List<Account> GetAccountByCustomerSurname(string surname)

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

return accountRepository.Filter(e => e.Customer.Surname.Contains(surname)).ToList();

}

public List<Account> GetAccountsByCustomerId(int customerId)

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

return accountRepository.Filter(e => e.CustomerId == customerId).ToList();

}

public int GetCountOfAllAccounts()

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

return accountRepository.All().Count();

}

public IQueryable<Account> GetAllAccounts()

{

var accountRepository = \_factoryOfRepositories.GetAccountRepository();

return accountRepository.All();

}

}

}

**CurrencyService.cs**

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class CurrencyService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public CurrencyService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public List<Currency> GetCurrencies()

{

var currencyRepository = \_factoryOfRepositories.GetCurrencyRepository();

return currencyRepository.All().ToList();

}

public Currency GetCurrencyByValue(string value)

{

var currencyRepository = \_factoryOfRepositories.GetCurrencyRepository();

try

{

return currencyRepository.Find(e => e.Value == value);

}

catch (RepositoryException ex)

{

throw new CurrencyServiceException(ex.Message);

}

}

}

}

**DepositService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class DepositService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public DepositService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public Deposit CreateDeposit(double startSummary, int customerId, int currencyId, int optionDepositId)

{

Guard.AgainstNegatives((int) startSummary);

var deposit = new Deposit

{

CreateDate = DateTime.Now,

OpeningSummary = startSummary,

Summary = startSummary,

CustomerId = customerId,

CurrencyId = currencyId,

OptionDepositId = optionDepositId

};

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

depositRepository.Create(deposit);

try

{

\_unitOfWork.PreSave();

}

catch (Exception exception)

{

throw new ServiceException(exception);

}

return deposit;

}

public void UpdateDeposit(Deposit deposit)

{

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

try

{

depositRepository.Update(deposit);

}

catch (Exception ex)

{

throw new DepositServiceException(ex);

}

}

public Deposit GetDepositById(int depositId)

{

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

try

{

return depositRepository.GetEntityById(depositId);

}

catch (RepositoryException ex)

{

throw new DepositServiceException(ex.Message);

}

}

public void RemoveDeposit(Deposit deposit)

{

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

depositRepository.Remove(deposit);

}

public List<Deposit> GetDepositsByCustomerId(int customerId)

{

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

return depositRepository.Filter(e => e.CustomerId == customerId).ToList();

}

public int GetCountOfAllDeposits()

{

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

return depositRepository.All().Count();

}

public IQueryable<Deposit> GetAllDeposits()

{

var depositRepository = \_factoryOfRepositories.GetDepositRepository();

return depositRepository.All();

}

}

}

**LoanService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class LoanService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public LoanService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public Loan CreateLoan(double startSummary, int customerId, int currencyId, int optionLoanId)

{

Guard.AgainstNegatives((int) startSummary);

var loan = new Loan

{

OpeningSummary = startSummary,

Summary = startSummary,

CreateDate = DateTime.Now,

CustomerId = customerId,

CurrencyId = currencyId,

OptionLoanId = optionLoanId

};

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

loanRepository.Create(loan);

try

{

\_unitOfWork.PreSave();

}

catch (Exception exception)

{

throw new ServiceException(exception);

}

return loan;

}

public void UpdateLoan(Loan loan)

{

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

try

{

loanRepository.Update(loan);

}

catch (Exception ex)

{

throw new LoanServiceException(ex);

}

}

public void RemoveLoan(Loan loan)

{

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

try

{

loanRepository.Remove(loan);

}

catch (Exception ex)

{

throw new LoanServiceException(ex);

}

}

public Loan GetLoanById(int loanId)

{

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

try

{

return loanRepository.GetEntityById(loanId);

}

catch (Exception ex)

{

throw new LoanServiceException(ex);

}

}

public List<Loan> GetLoansByCustomerId(int customerId)

{

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

return loanRepository.Filter(e => e.CustomerId == customerId).ToList();

}

public int GetCountOfAllLoans()

{

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

return loanRepository.All().Count();

}

public IQueryable<Loan> GetAllLoans()

{

var loanRepository = \_factoryOfRepositories.GetLoanRepository();

return loanRepository.All();

}

}

}

**OptionLoanService.cs**

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class OptionLoanService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public OptionLoanService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public List<OptionLoan> GetOptionLoans()

{

var optionLoanRepository = \_factoryOfRepositories.GetOptionLoanRepository();

return optionLoanRepository.All().ToList();

}

public OptionLoan GetOptionLoanByName(string name)

{

var optionLoanRepository = \_factoryOfRepositories.GetOptionLoanRepository();

try

{

return optionLoanRepository.Find(e => e.Name == name);

}

catch (RepositoryException e)

{

throw new OptionLoanServiceException(e.Message);

}

}

}

}

**OptionDepositService.cs**

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class OptionDepositService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public OptionDepositService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public List<OptionDeposit> GetOptionDeposits()

{

var optionDepositRepository = \_factoryOfRepositories.GetOptionDepositRepository();

return optionDepositRepository.All().ToList();

}

public OptionDeposit GetOptionDepositByName(string name)

{

var optionDepositRepository = \_factoryOfRepositories.GetOptionDepositRepository();

try

{

return optionDepositRepository.Find(e => e.Name == name);

}

catch (RepositoryException e)

{

throw new OptionDepositServiceException(e.Message);

}

}

}

}

**MembershipService.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using CourseProject.Core;

using CourseProject.Core.Entities;

using CourseProject.Core.Exceptions;

using CourseProject.Infrastructure.Guard.Validation;

using CourseProject.Services.Exceptions;

namespace CourseProject.Services.Services

{

public class MembershipService : IService

{

private readonly IUnitOfWork \_unitOfWork;

private readonly IRepositoryFactory \_factoryOfRepositories;

public MembershipService(IUnitOfWork unitOfWork, IRepositoryFactory factoryOfRepositories)

{

Guard.AgainstNullReference(unitOfWork, "unitOfWork");

Guard.AgainstNullReference(factoryOfRepositories, "factoryOfRepositories");

\_unitOfWork = unitOfWork;

\_factoryOfRepositories = factoryOfRepositories;

}

public Customer CreateCustomer(string name, string surname, string patronymic, string passportData,

string address, DateTime birthDate)

{

var customer = new Customer()

{

Name = name,

Surname = surname,

Patronymic = patronymic,

PassportData = passportData.ToUpper(),

Address = address,

BirthDate = birthDate

};

customer.SetAge();

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

customerRepository.Create(customer);

try

{

\_unitOfWork.PreSave();

}

catch (Exception e)

{

throw new ServiceException(e);

}

return customer;

}

public Customer GetCustomerByPassportData(string passportData)

{

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

try

{

return customerRepository.Find(e => e.PassportData == passportData);

}

catch (RepositoryException e)

{

throw new MembershipServiceException(e.Message);

}

}

public List<Customer> GetCustomersBySurname(string surname)

{

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

try

{

return customerRepository.Filter(e => e.Surname.Contains(surname)).ToList();

}

catch (Exception e)

{

throw new MembershipServiceException(e.Message);

}

}

public Customer GetCustomerById(int userId)

{

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

var customer = customerRepository.GetEntityById(userId);

if (customer == null)

{

throw new MembershipServiceException("Customer doesn't exist.");

}

return customer;

}

public void UpdateCustomer(Customer customer)

{

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

try

{

customerRepository.Update(customer);

}

catch (Exception e)

{

throw new MembershipServiceException(e);

}

}

public void RemoveCustomer(Customer customer)

{

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

customerRepository.Remove(customer);

}

public int GetCountOfAllCustomers()

{

var customerRepository = \_factoryOfRepositories.GetCustomerRepository();

return customerRepository.All().Count();

}

}

}