ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ИВАНОВСКОЙ ОБЛАСТИ
Областное государственное бюджетное профессиональное образовательное учреждение
«Ивановский промышленно-экономический колледж»

КУРСОВОЙ ПРОЕКТ

Разработка приложения для агентства недвижимости

ИВПЭК. 09.02.07. 20

Специальность: **09.02.07 Информационные системы и программирование, базовая подготовка**

Руководитель курсового проекта _____ Вяткин Р.В
Выполнил обучающийся группы 407 _____ Соколов Д.С

Курсовой проект выполнен и защищен с оценкой_____

«___» _____ 2023 г.

Иваново 2023

Содержание

Введение

Целью данной курсовой работы является разработка простой системы для агентства недвижимости, которая позволит управлять базой данных недвижимости, а также вести учет сделок и клиентов. Для реализации системы будут использованы технологии C#, WinForms и Dotnet Framework.

В ходе работы будут решены следующие задачи:

- Проектирование архитектуры и интерфейса системы
- Программирование логики и функционала системы
- Тестирование и отладка системы
- Документирование системы

Документация - это набор текстовых и графических материалов, которые объясняют, как работает и как использовать программное обеспечение (ПО). Документация может быть разной в зависимости от того, для кого она предназначена. Существуют два основных типа документации:

- Техническая документация — это документация для специалистов по разработке и поддержке ПО. Она включает такие документы, как спецификации требований, архитектуры, дизайна, кода, тестирования и сопровождения ПО. Техническая документация помогает разработчикам создавать и улучшать ПО, а также находить и исправлять ошибки.
- Пользовательская документация — это документация для конечных пользователей ПО. Она включает такие документы, как руководства по установке, настройке, использованию и обслуживанию ПО, справочные системы, FAQ и обучающие материалы. Пользовательская документация помогает пользователям изучить и освоить ПО, а также решать проблемы и вопросы при его использовании.

Документация играет важную роль в разработке ПО по нескольким причинам:

- Документация улучшает коммуникацию между участниками проекта по разработке ПО. Документация позволяет согласовывать цели, задачи, роли и ответственности между заказчиками, менеджерами, разработчиками и тестировщиками. Документация также предотвращает недопонимания и конфликты в ходе работы над проектом.
- Документация упрощает процесс создания и поддержки ПО. Документация служит основой для анализа, проектирования, реализации, тестирования и сопровождения ПО. Документация также обеспечивает единообразие и согласованность кода и интерфейса ПО.
- Документация повышает удовлетворенность и лояльность пользователей ПО. Документация помогает пользователям легко и быстро научиться пользоваться ПО и получать от него максимальную пользу. Документация также снижает количество обращений в службу поддержки и повышает доверие к разработчику.

Для того, чтобы документация была качественной и полезной, необходимо учитывать следующие аспекты:

- Цель и аудитория документации. Нужно определить, зачем нужна документация и кому она адресована. Нужно адаптировать документацию к конкретной задаче и группе читателей.
- Структура и формат документации. Нужно логично и последовательно организовать документацию, чтобы облегчить поиск и понимание информации. Нужно использовать четкое разделение на разделы, подразделы, параграфы, списки, таблицы и т.д. Нужно выбрать единый и совместимый формат для текстовых и графических материалов.
- Содержание и стиль документации. Нужно сделать документацию полной, точной, актуальной и понятной. Нужно избегать лишней информации, повторений, противоречий и неясностей. Нужно использовать простой, ясный и корректный язык, а также соблюдать правила орфографии и грамматики.

- Редактирование и проверка документации. Нужно тщательно проверить документацию на ошибки, неточности и недостатки перед публикацией или распространением. Нужно исправлять, дополнять или обновлять документацию по мере необходимости. Нужно тестировать документацию на реальных пользователях или экспертах для получения обратной связи и улучшения

1 Концептуальное проектирование

1.1 Теоретический вопрос

Агенства недвижимости - это организации, которые предоставляют услуги по поиску, покупке, продаже, аренде и оценке недвижимости. Агенты по недвижимости - это специалисты, которые занимаются этими видами деятельности и представляют интересы клиентов.

Агенства недвижимости могут быть разных типов и размеров. Некоторые из них работают только на определенном рынке или с определенными видами недвижимости, другие охватывают широкий спектр услуг и регионов. Некоторые агенства имеют большую сеть филиалов и партнеров, другие – небольшие и локальные.

Выбрать агенство недвижимости не так просто, как может показаться. Нужно учитывать множество факторов, таких как:

- Репутация и опыт агентства на рынке. Это важный показатель того, насколько агентство доверенно и уважаемо среди клиентов и коллег. Чем дольше и успешнее агентство работает на рынке, тем больше шансов, что оно предоставит качественные и безопасные услуги. Для проверки репутации и опыта агентства можно обратиться к различным источникам, таким как сайты рейтингов и отзывов, справочники и каталоги, средства массовой информации, а также личные знакомые или рекомендации;
- Квалификация и профессионализм агентов. Это влияет на то, насколько агенты способны эффективно и грамотно вести переговоры, оформлять документы, решать проблемы и конфликты, а также удовлетворять потребности и пожелания клиентов. Для проверки квалификации и профессионализма агентов можно посмотреть их образование, сертификаты, награды, портфолио, отзывы и рекомендации;

- Стоимость и качество услуг. Это определяет то, насколько выгодно и удобно сотрудничать с агентством недвижимости. Чем ниже стоимость услуг, тем больше экономия для клиента. Однако, низкая стоимость не должна быть в ущерб качеству услуг. Качество услуг зависит от того, насколько агентство выполняет свои обязательства, соблюдает сроки и условия договора, защищает интересы клиента и гарантирует безопасность сделки. Для сравнения стоимости и качества услуг разных агентств можно использовать сайты сравнения цен и условий, а также личный опыт или рекомендации;
- Отзывы и рекомендации клиентов. Это помогает узнать о реальном опыте сотрудничества с агентством недвижимости, его достоинствах и недостатках, а также о том, насколько клиенты довольны результатами и процессом работы. Отзывы и рекомендации клиентов можно найти на разных сайтах, форумах, социальных сетях, а также у личных знакомых или рекомендаторов;
- Наличие лицензии и страховки. Это гарантирует то, что агентство недвижимости работает законно и несет ответственность за свои действия. Лицензия — это документ, который подтверждает право агентства на осуществление деятельности по недвижимости. Страховка - это документ, который защищает агентство и клиента от возможных рисков, связанных с сделкой. Для проверки наличия лицензии и страховки агентства можно обратиться к официальным органам или сайтам, которые выдают и контролируют эти документы.
- Способность агентства удовлетворить индивидуальные потребности и пожелания клиента. Это показывает то, насколько агентство гибко и внимательно относится к каждому клиенту и его ситуации. Способность агентства удовлетворить индивидуальные потребности и пожелания клиента зависит от того, насколько агентство:
    - Исследует и анализирует рынок недвижимости и предлагает оптимальные варианты для клиента;

- Слушает и понимает цели, бюджет и предпочтения клиента;
- Предоставляет полную и достоверную информацию о недвижимости и сделке;
- Консультирует и сопровождает клиента на всех этапах работы;
- Решает возникающие вопросы и проблемы в интересах клиента.

Для выяснения способности агентства удовлетворить индивидуальные потребности и пожелания клиента можно обратиться к агенту по недвижимости лично или по телефону и задать ему вопросы.

1.2 Спецификация требований программного обеспечения

Поскольку разрабатывается настольное приложение, то к нему предъявляются следующие требования:

- простота использования, включая понятность пользователю;
- управление базой данных недвижимости, включая добавление, редактирование и удаление объектов;
- управление клиентами и сделками, включая ручную регистрацию администратором и хранение контактных данных.
- Независимость системы от конкретной БД: база данных представлена в виде файла собственного формата.

1.3 Требование к программному средству

Вид устройств: настольные компьютеры

Операционные системы: Windows 7 и выше с установленным .Net Framework 4.5.1

Архитектуры: amd64

2 Техно-рабочий проект

2.1 Алгоритм решения поставленной задачи программного продукта

В качестве жизненного цикла разработки ПО был избран водопадный жизненный цикл. Диаграмма представлена на рисунке 1.

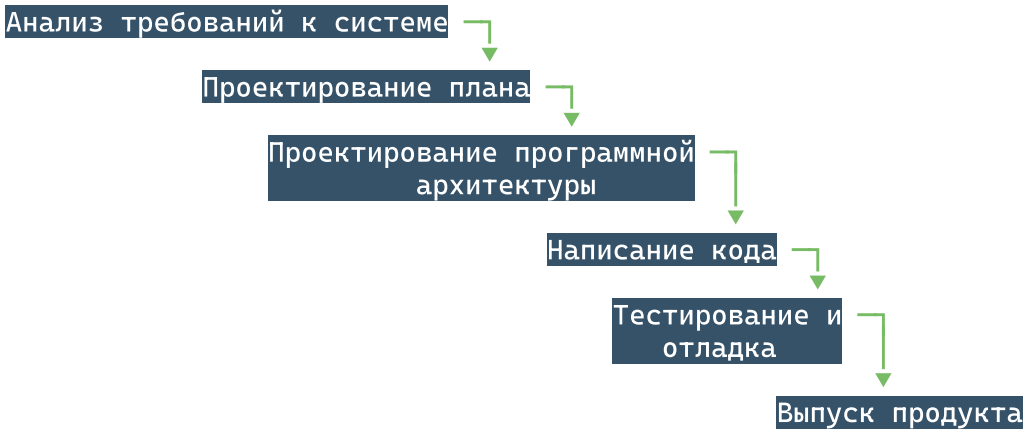

Рисунок 1 — Диаграмма жизненного цикла разработки приложения

2.2 Обоснование выбора средств разработки

В качестве средств разработки был использован язык программирования C#, технологии NetFramework 4.5.1 и Winforms.

C# - это высокоуровневый язык программирования, который был разработан Microsoft в 2000 году. Он имеет ряд преимуществ и недостатков по сравнению с другими языками, такими как Java, Python, C++ и т.д.

Некоторые преимущества C#:

- Простота и читаемость. Синтаксис C# похож на человеческий язык и имеет высокий уровень абстракции от машинного кода. Это делает его легким для изучения и понимания.
- Мультипарадигмальность. C# поддерживает различные парадигмы программирования, такие как объектно-ориентированное, функциональное, обобщенное, императивное и декларативное. Это дает программистам большую гибкость и возможность выбирать подходящий стиль для каждой задачи.

- Переносимость. C# работает на платформе .NET, которая позволяет запускать приложения на разных операционных системах и устройствах с помощью виртуальной машины CLR (Common Language Runtime). Также существуют альтернативные реализации .NET, такие как Mono и .NET Core, которые расширяют возможности переносимости C#.
- Богатая библиотека. C# имеет доступ к огромному количеству классов и методов, которые предоставляются фреймворком .NET. Эти библиотеки облегчают разработку различных типов приложений, таких как веб, мобильные, настольные, игровые и т.д.

Некоторые недостатки C#:

- Зависимость от .NET. Хотя .NET предоставляет много возможностей для C#, он также создает некоторые ограничения и проблемы совместимости. Например, некоторые функции C# могут не работать на старых версиях .NET или на других платформах. Также .NET может занимать много места на диске и потреблять много ресурсов системы.
- Низкая производительность. Поскольку C# является высокоуровневым языком, он требует компиляции в промежуточный код MSIL (Microsoft Intermediate Language), который затем интерпретируется виртуальной машиной CLR. Этот процесс может замедлять скорость выполнения приложений, особенно при работе с большими объемами данных или сложными алгоритмами. Также C# не имеет полного контроля над управлением памяти и сборкой мусора, что может приводить к утечкам памяти или задержкам.

WinForms - это технология для создания настольных приложений на платформе .NET с использованием C# или других языков. Она была выпущена в 2002 году и с тех пор получила несколько обновлений и улучшений. Однако она также столкнулась с конкуренцией со стороны других технологий, таких как WPF, UWP, Xamarin, Delphi и т.д.

Некоторые преимущества WinForms:

- Простота и скорость разработки. WinForms имеет простой и интуитивный интерфейс для создания форм и элементов управления с помощью перетаскивания и настройки свойств. Также WinForms имеет много готовых компонентов и библиотек, которые можно легко интегрировать в свои приложения. WinForms позволяет быстро создавать прототипы и простые приложения без необходимости изучать сложные концепции или фреймворки.
- Совместимость и поддержка. WinForms работает на всех версиях Windows, начиная с Windows XP, и поддерживается всеми версиями .NET Framework. Также WinForms имеет большое сообщество разработчиков и пользователей, которые могут помочь с решением проблем, обучением или советами. WinForms является проверенной и надежной технологией, которая используется во многих крупных и успешных проектах.
- Гибкость и настраиваемость. WinForms дает программистам полный контроль над внешним видом и поведением своих приложений. Они могут создавать собственные элементы управления, изменять стили, цвета, шрифты, анимации и т.д. Также они могут использовать сторонние библиотеки или инструменты для расширения функциональности или улучшения дизайна своих приложений.

Некоторые недостатки WinForms:

- Устаревший и ограниченный дизайн. WinForms основан на стандартных элементах управления Windows Forms, которые имеют устаревший и скучный вид. Они не поддерживают современные тенденции в дизайне, такие как плоский стиль, адаптивность, анимация и т.д. Также они не могут отображать сложную графику, видео или 3D-элементы. Для создания современных и красивых приложений на WinForms требуется много усилий и дополнительных ресурсов.
- Низкая производительность и эффективность. WinForms использует GDI+ для рендеринга графики, что является устаревшим и медленным

способом. Он не использует аппаратное ускорение или оптимизацию для отображения графических элементов. Это может приводить к низкой скорости отклика, задержкам или зависаниям приложений, особенно при работе с большим количеством элементов управления или данных. Также WinForms потребляет много памяти и ресурсов системы, поскольку он создает много объектов и хэндлов для каждого элемента управления. Это может приводить к утечкам памяти или снижению производительности приложений.

2.3 Разработка макета приложения

Макет изображен на рисунке 2:

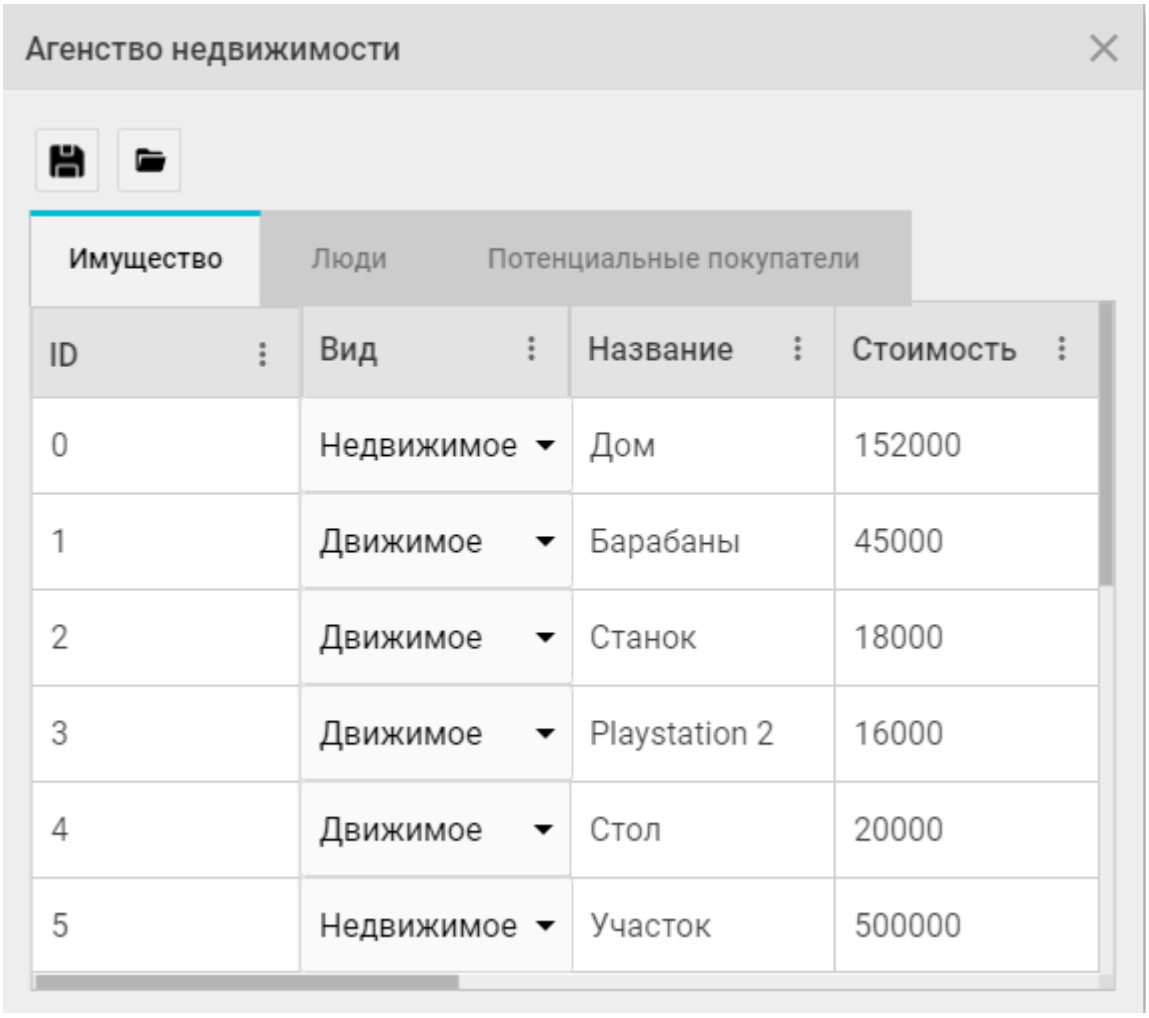

Рисунок 2 — Макет программы

2.4 Описание разработки приложения

Разработка приложения началось с составления плана разработки. Далее, мысленно была составлена архитектура приложения.

Далее, был создан проект Visual Studio, как показано на рисунке 3:

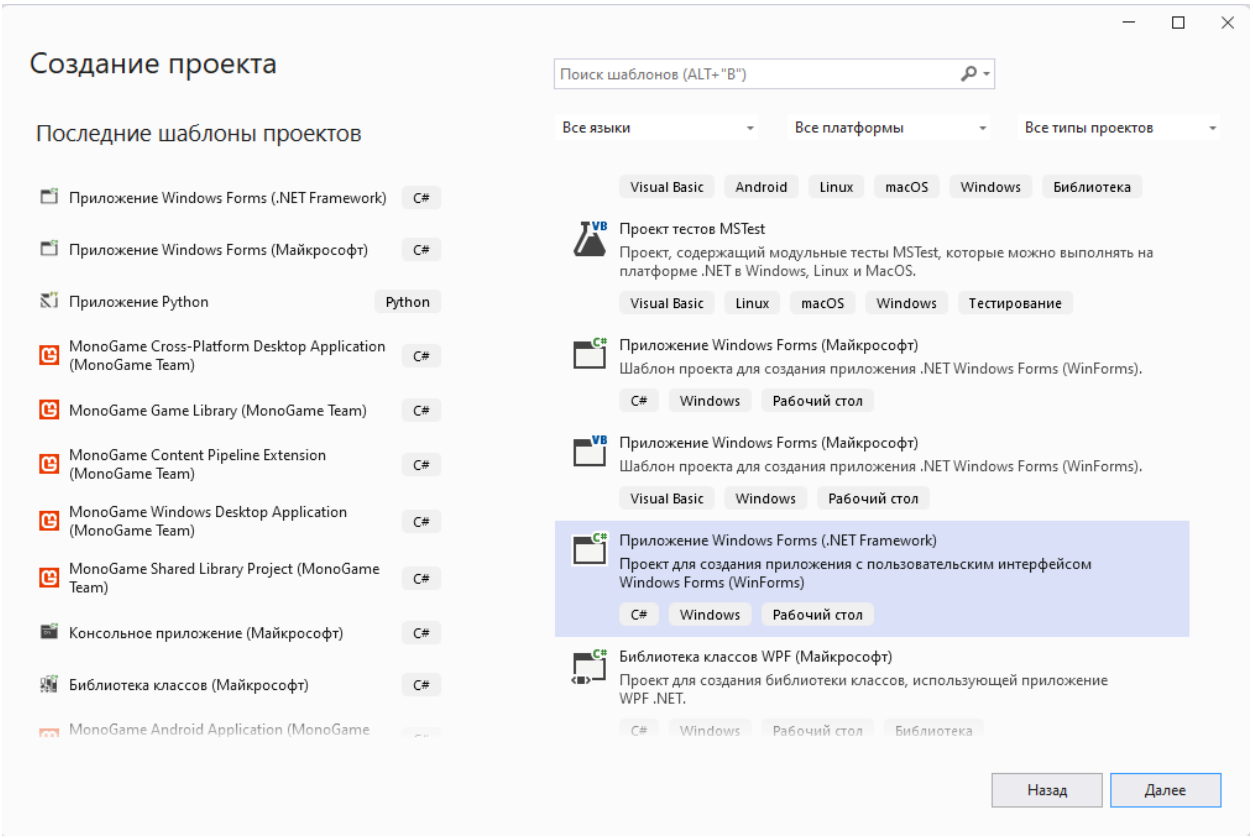

Рисунок 3 — создание проекта

Далее, был создан макет формы с элементами управления, как на рисунке 4. Затем был написан и отлажен код для функционирования формы и остальных компонентов, таких, как сохранение базы в файл и связь между таблицами через выпадающие списки. Наконец, было проведено финальное тестирование и составление таблицы с тест-кейсами и ожидаемыми результатами.

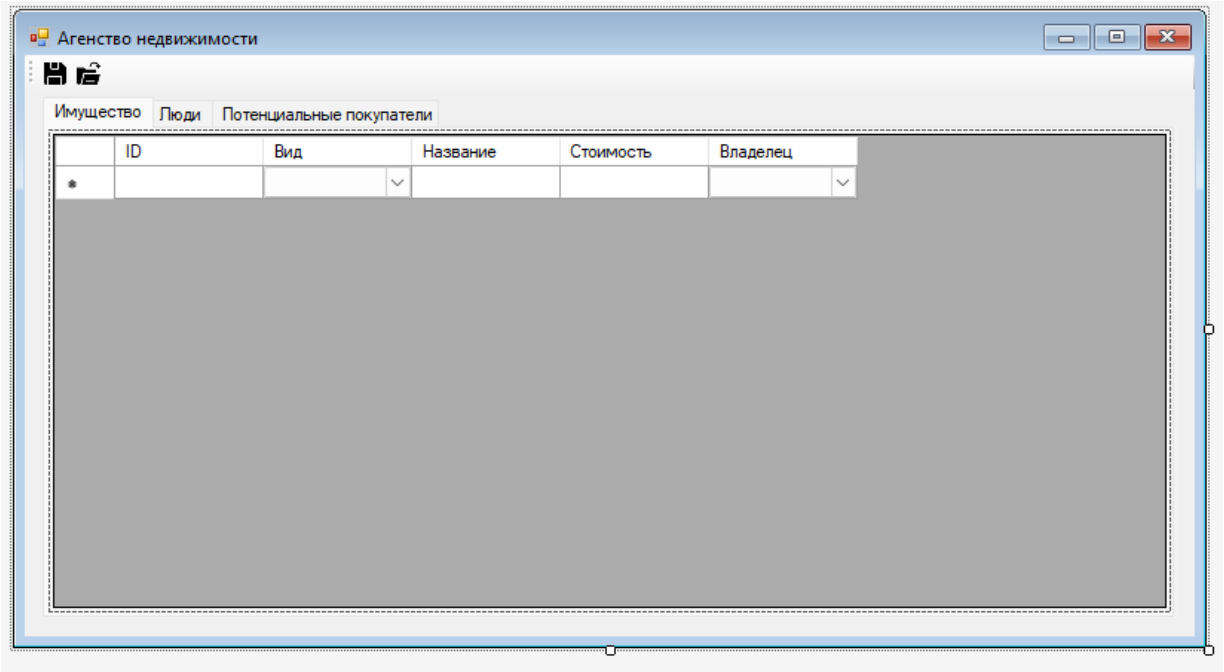

Рисунок 4 — макет WinForms

2.5 Тестирование приложения

Результаты ручного тестирования представлены в таблице 1:

Таблица 1 — Сводная ведомость по ручному тестированию

Ситуация	Ожидаемый результат	Фактический результат
Добавление записи	Запись добавляется, если есть проблемы форматирования чисел программа не даст создать запись.	Запись добавляется, если есть проблемы форматирования чисел программа не даст создать запись.
Сохранение	Программа не даст сохранить, если есть ошибки форматирования.	Программа не даст сохранить, если есть ошибки форматирования.
Закрытие окна	Сохранение перед закрытием, но если есть ошибки форматирования, программа не закроется и уведомит пользователя.	Сохранение перед закрытием, но если есть ошибки форматирования, программа не закроется и уведомит пользователя.
Открыт выпадающий список	Записи, ассоциированные со списком открываются и выбираются	Записи, ассоциированные со списком открываются и выбираются

3 Рабочая документация

3.1 Руководство пользователя

Программа не нуждается в установке. Эта программа портативна и работает везде, где есть ОС Windows и .Net Framework 4.5.1.

Запустите программу, вас встретит окно системы (рисунок 5). Сверху есть две кнопки (рисунок 6):

- Кнопка сохранения — сохраняет текущее состояние системы в файл «database.real_estate_agency» (рисунок 7)
- Кнопка загрузки — загружает последнее сохранённое состояние системы

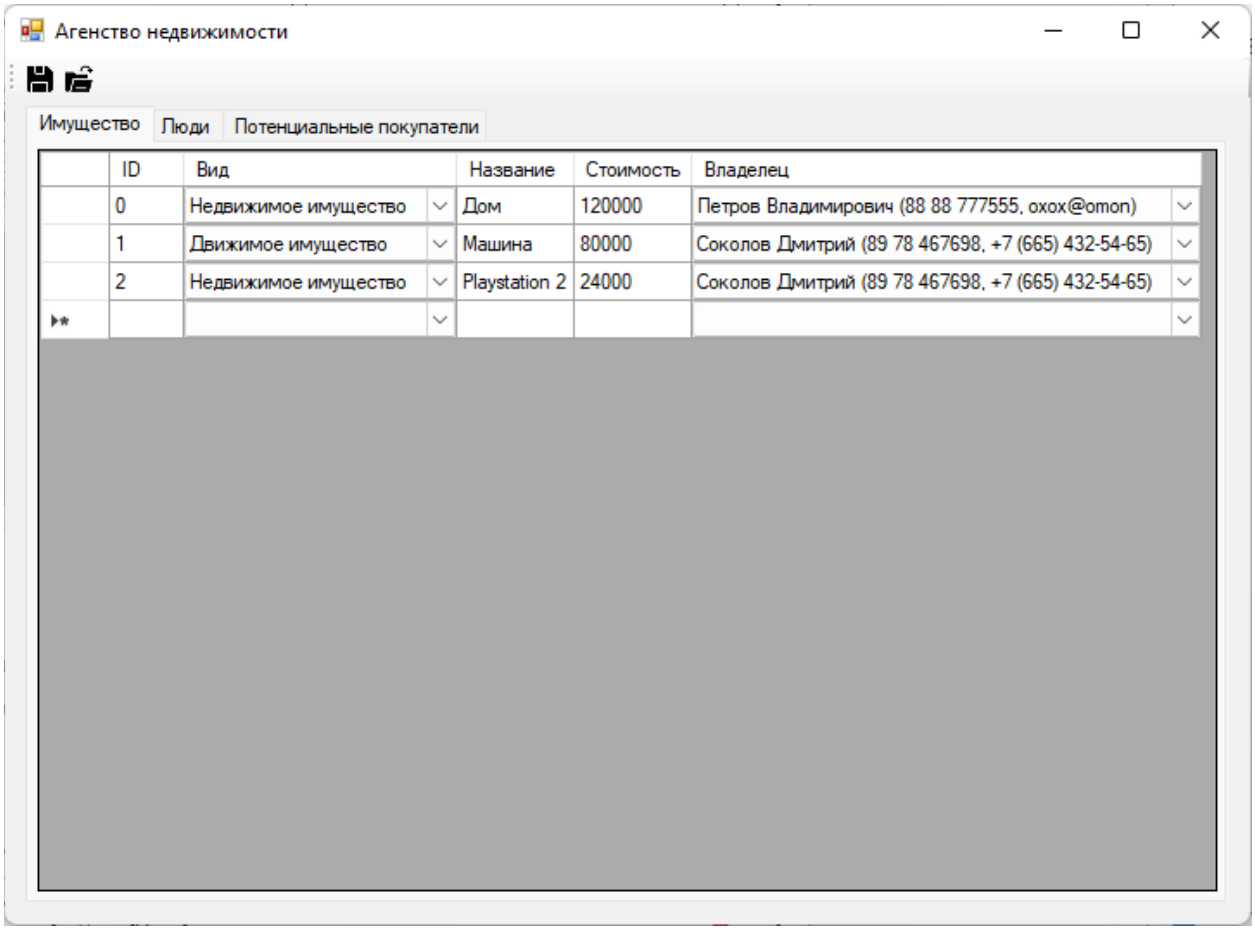

Рисунок 5 — Окно системы

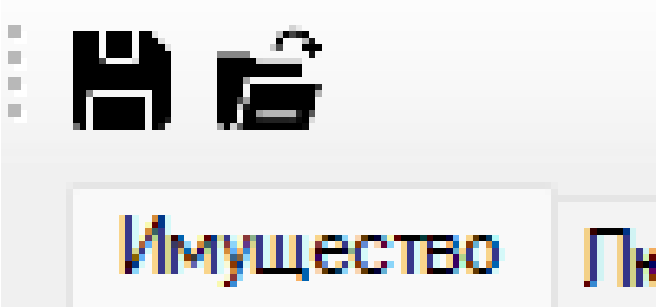

Рисунок 6 — сохранение (слева) и загрузка (справа)

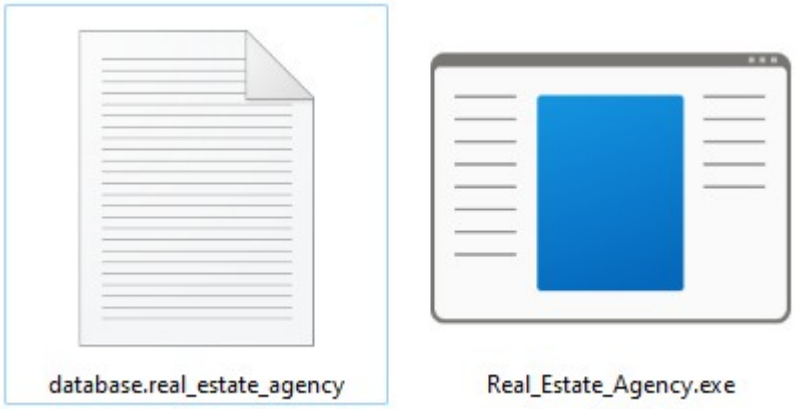

Рисунок 7 — файл базы данных (слева) и исполняемая программа (справа) в одной папке

3.2 Руководство программиста

В этом руководстве можно узнать не только о внутреннем устройстве приложения, но и о том, как разработчик может расширять его функциональность. Для начала рассмотрим диаграмму вариантов использования приложения, которая показывает, как приложение взаимодействует с пользователями. Диаграмма прецедентов изображена на рисунке 8:

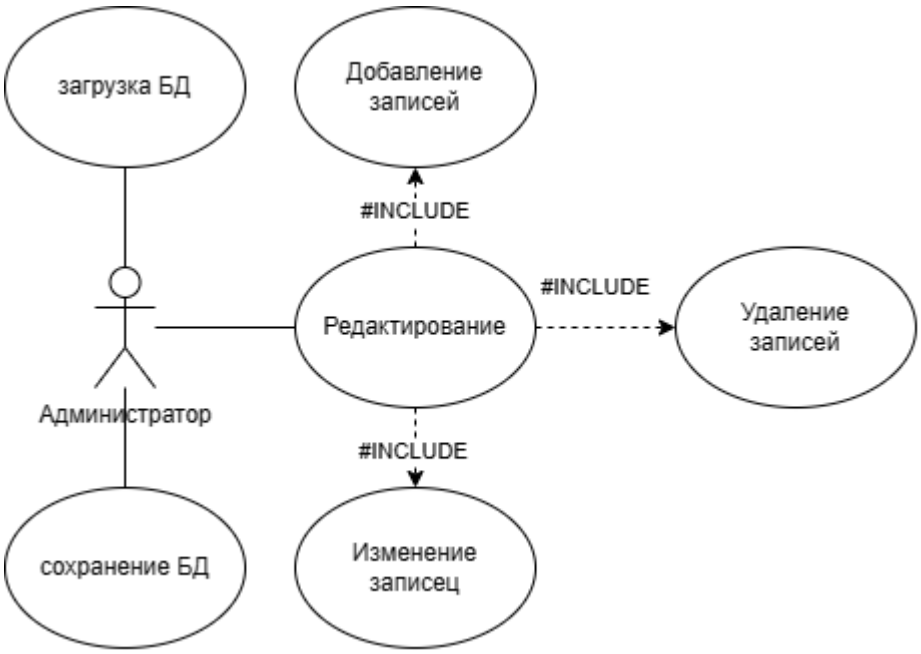

Рисунок 8 — Диаграмма прецедентов

UML диаграмма классов - это способ изображать структуру и связи между классами в системе с помощью специальных символов и линий. Классы - это основные блоки объектно-ориентированного программирования, которые определяют свойства (атрибуты) и поведение (методы) объектов. UML диаграмма классов показывает, какие классы есть в системе, какие атрибуты (поля) и методы они имеют, какие интерфейсы они реализуют и какие отношения (наследование, ассоциация, композиция и т.д.) они имеют с другими классами. UML диаграмма классов помогает проектировать и анализировать систему с точки зрения ее объектной модели, то есть того, как объекты взаимодействуют друг с другом и как они решают задачи. На UML диаграмме классов каждый класс представляется прямоугольником с тремя областями: название класса, атрибуты класса и методы класса. Отношения между классами показываются разными типами линий и стрелок. Например, наследование показывается пустой стрелкой от подкласса к суперклассу, а ассоциация показывается сплошной линией между двумя классами. Диаграмма классов для этого предложения представлена на рисунке 9:

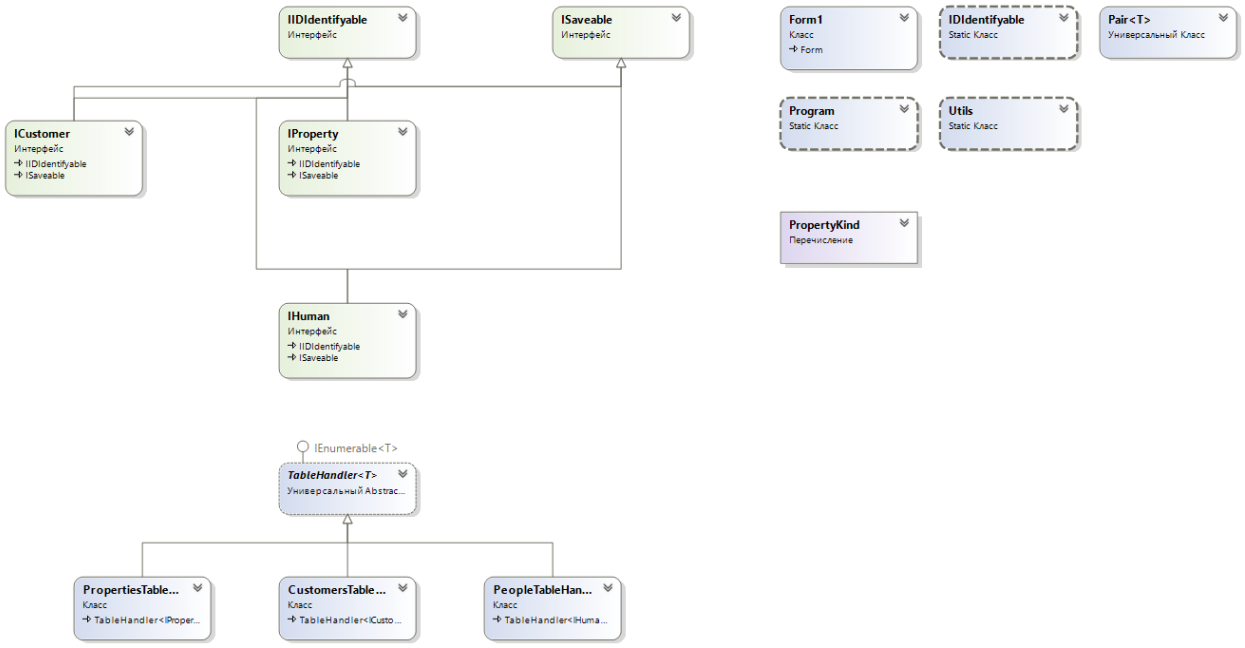

Рисунок 9 — диаграмма классов

Давайте рассмотрим, как сохраняются данные в базе данных. У нас есть разные таблицы, и для каждой из них есть свой класс. Эти классы наследуются от общего класса TableHandler<T>, который работает с любым типом данных T. Тип T означает одну строку в таблице. Класс TableHandler<T> знает, как сохранить всю таблицу, но он не знает, как сохранить каждую строку. Он доверяет эту задачу типу T, который должен иметь метод Save. Этот метод принимает поток (System.IO.Stream) и записывает в него данные строки. Таким образом, мы используем полиморфизм: один и тот же метод Save вызывается для разных типов данных T, и каждый из них делает свою работу. Чтобы гарантировать, что тип T имеет метод Save, мы используем интерфейс ISaveable, который определяет этот метод. Все классы, которые реализуют этот интерфейс, должны иметь метод Save. На рисунке 10 показана схема этого процесса:

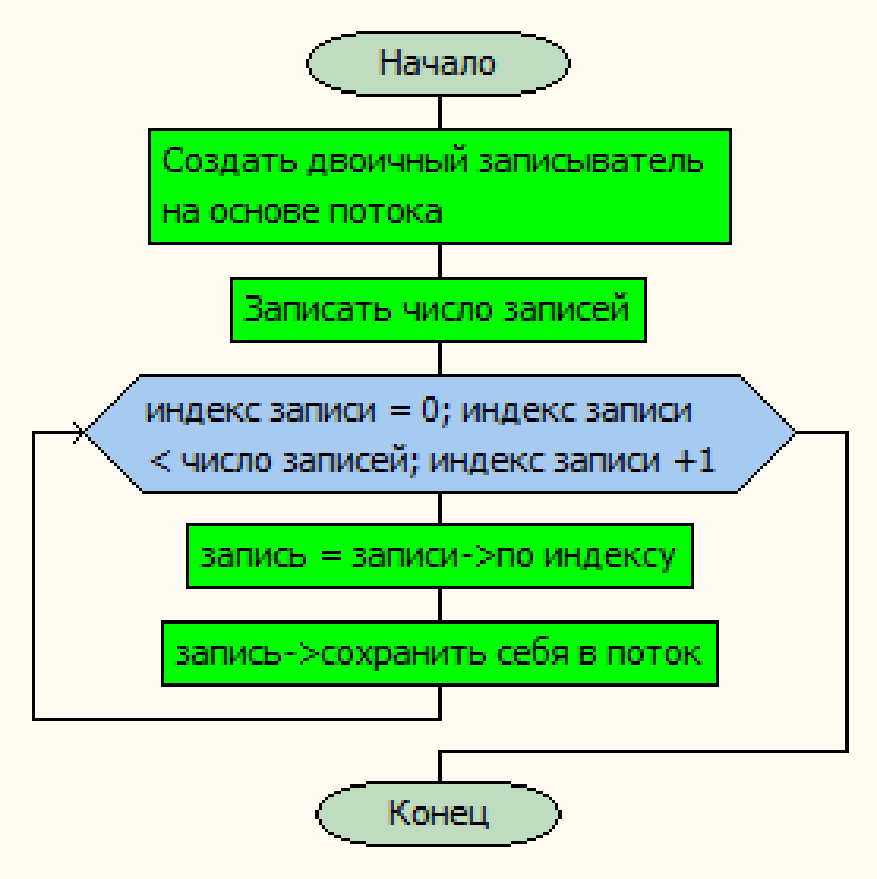

Рисунок 10 – Алгоритм сохранения БД (упрощённый)

Заключение

В ходе данной курсовой работы был разработана простая система для агентства недвижимости, которая позволяет управлять базой данных недвижимости, а также вести учет сделок и клиентов. Для реализации системы были использованы технологии C#, WinForms и Dotnet Framework.

В ходе работы были решены следующие задачи:

- Проектирование архитектуры и интерфейса системы
- Программирование логики и функционала системы
- Тестирование и отладка системы
- Документирование системы

Приложение А

Код формы (Form1) и сопутствующих классов:

```
using Real_Estate_Agency;
using System;
using System.Collections;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Data.SqlTypes;
using System.Diagnostics.Eventing;
using System.Drawing;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Runtime;
using System.Runtime.InteropServices;
using System.Runtime.Remoting.Messaging;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Real_Estate_Agency {
    public partial class Form1 : Form {

        const string DBName = "database.real_estate_agency";

        PropertiesTableHandler propH;
        PeopleTableHander peopleH;
        CustomersTableHandler customersH;
        public Form1() {
            InitializeComponent();
            peopleH = new PeopleTableHander(peopleGridView);
             propH = new PropertiesTableHandler(propertyGridView,
peopleH);
                                                customersH    =    new
CustomersTableHandler(customersDataGridView, peopleH, propH);

            propH.UpdateComboBoxes();
            LoadFromDisk();


        }

        private void LoadFromDisk() {



            peopleH.Validate();
```

```
                propH.Validate();
                customersH.Validate();

                 if(peopleH.CheckForErrors() || propH.CheckForErrors()
|| customersH.CheckForErrors()) {
                    MessageBox.Show(
                            "Ячейки содержат ошибки!",
                            "Ошибка",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Error
                        );
                    return;
                }


                if (File.Exists(DBName))
                    using (var stream = File.OpenRead(DBName)) {
                        peopleH.Load(stream);
                        propH.Load(stream);
                        customersH.Load(stream);
                    }
            }

            void AddMissplacedIDs() {
                peopleH.AddMissplacedIDs();
                propH.AddMissplacedIDs();
                customersH.AddMissplacedIDs();
            }

               private void saveToolStripButton_Click(object sender,
EventArgs e) {
                SaveToDisk();
            }

            private void SaveToDisk() {

                peopleH.Validate();
                propH.Validate();
                customersH.Validate();

                                    if (peopleH.CheckForErrors() ||
propH.CheckForErrors() || customersH.CheckForErrors()) {
                    MessageBox.Show(
                            "Ячейки содержат ошибки!",
                            "Ошибка",
                            MessageBoxButtons.OK,
                            MessageBoxIcon.Error
                        );
                    return;
                }
```

```
                var a = peopleH.ApplyChanges();
                var b = propH.ApplyChanges();
                var c = customersH.ApplyChanges();

                AddMissplacedIDs();

                using (var stream = File.Create(DBName)) {
                    peopleH.Save(stream);
                    propH.Save(stream);
                    customersH.Save(stream);
                }
            }

                private void tabControl1_SelectedIndexChanged(object
sender, EventArgs e) {
                AddMissplacedIDs();
                switch (tabControl1.SelectedIndex) {
                    case 0: propH.UpdateComboBoxes(); break;
                    case 1: peopleH.UpdateComboBoxes(); break;
                    case 2: customersH.UpdateComboBoxes(); break;
                }
            }

               private void loadToolStripButton_Click(object sender,
EventArgs e) {
                LoadFromDisk();
            }

                    private void Form1_FormClosing(object sender,
FormClosingEventArgs e) {
                SaveToDisk();
            }
        }

        public interface ISaveable {
            void Save(Stream output);
            void Load(Stream input);
        }


        // Строимся на предположении, что ID - это первая колонка.
          abstract class TableHandler<T> : IEnumerable<T> where T :
IIDIdentifyable, ISaveable {

            protected readonly DataGridView Grid;

            public virtual void Validate() { }

            public bool ApplyChanges() => Grid.EndEdit();
            public void RevertChanges() => Grid.CancelEdit();
```

```
            public bool CheckForErrors() {
                    bool hasErrorText = false;
                    foreach (DataGridViewRow row in this.Grid.Rows) {
                         foreach (DataGridViewCell cell in row.Cells)
{
                            if (cell.ErrorText.Length > 0) {
                                hasErrorText = true;
                                break;
                            }
                        }
                        if (hasErrorText)
                            break;
                    }
                    return hasErrorText;
                }

                public TableHandler(DataGridView grid) {
                Grid = grid;

            }

            public void AddMissplacedIDs() {
                int maxId = -1;

                // Выполним поиск в два прохода
                for (int row = 0; row < Grid.NewRowIndex; row++) {
                    var cell = Grid.Rows[row].Cells[0];
                    var value = cell.Value;

                    if (value is int rowId)
                        maxId = rowId > maxId ? rowId : maxId;
                }

                for (int row = 0; row < Grid.NewRowIndex; row++) {
                    var cell = Grid.Rows[row].Cells[0];
                    var value = cell.Value;

                    if (!(value is int))
                        cell.Value = ++maxId;
                }
            }

                  protected void UpdateList<E>(string updatedColumn,
TableHandler<E> source) where E : IIDIdentifyable, ISaveable {
                                var values = source.Where(x =>
x.ID.HasValue).Select(x => new Pair<E> {
                    Value = x,
                    Name = x.DisplayName
                }).ToArray();
```

```
                        var column = Grid.Columns[updatedColumn] as
DataGridViewComboBoxColumn;
                column.DisplayMember = "Name";
                column.ValueMember = "Value";
                column.DataSource = values;

                 for (int rowIndex = 0; rowIndex < Grid.NewRowIndex;
rowIndex++) {
                    var row = Grid.Rows[rowIndex];
                    var cell = row.Cells[column.Index];
                    cell.Value =
                        values.
                        Select(x => x.Value).
                                Where(x => IDIdentifyable.IDEquals(x,
cell.Value as IIDIdentifyable)).
                        FirstOrDefault();
                }
            }

            public int EntryCount => Grid.NewRowIndex;

            public virtual void UpdateComboBoxes() {

            }

            public void Save(Stream output) {
                using (BinaryWriter writer = new BinaryWriter(output,
Encoding.UTF8, true)) {
                    writer.Write(EntryCount);

                    foreach (var entry in this) {
                        entry.Save(output);
                    }
                }
            }

            public void Load(Stream input) {
                 using (BinaryReader reader = new BinaryReader(input,
Encoding.UTF8, true)) {
                    int count = reader.ReadInt32();
                    Clear();
                    for (int x = 0; x < count; x++) {
                        Grid.Rows.AddCopy(Grid.NewRowIndex);
                    }

                    foreach (var entry in this) {
                        entry.Load(input);
                    }

                    UpdateComboBoxes();
                }
```

```
            }

            private IEnumerable<T> UpdateEnumaerable() => Enumerable.
                    Range(0, Grid.NewRowIndex).
                    Select(x => CreateBasedOnRow(Grid.Rows[x]));

            public T FindByID(int id) =>
                                UpdateEnumaerable().Where(i  =>  i.ID  ==
id).FirstOrDefault();

                protected abstract T CreateBasedOnRow(DataGridViewRow
row);

            public void Clear() {
                Grid.Rows.Clear();
            }

                             public  IEnumerator<T>  GetEnumerator()  =>
UpdateEnumaerable().GetEnumerator();
                            IEnumerator  IEnumerable.GetEnumerator()  =>
GetEnumerator();
        }

        public interface IHuman : IIDIdentifyable, ISaveable {
            int? ID { get; }
            string LastName { get; set; }
            string FirstName { get; set; }
            string Patronymic { get; set; }
            string Pass { get; set; }
            string Phone { get; set; }
            string Email { get; set; }
        }


        static class Utils {

            public static string ExtractRawNumber(string formated) {
                formated = formated.Replace(" ", "").
                    Replace("-", "").
                    Replace("(", "").
                    Replace(")", "");

                if (formated.Length > 1 && formated[0] == '+') {
                     if (int.TryParse(formated[1].ToString(), out var
result))
                                          formated  =  (result  +  1)  +
formated.Substring(2);
                }

                return formated;
            }
```

```
            public static string FormatNumber(string numberRaw) {
                numberRaw = ExtractRawNumber(numberRaw);

                if (numberRaw.Length > 0) {

                    if (numberRaw.Length > 3) {
                        if (numberRaw.Length > 9) {
                                numberRaw = numberRaw.Substring(0, 7) +
"-" + numberRaw.Substring(7, 2) + "-" + numberRaw.Substring(9);
                        }

                                    numberRaw = numberRaw[0] + " (" +
numberRaw.Substring(1, 3) + ") " + numberRaw.Substring(4);
                    }

                    if (numberRaw[0] == '8') {
                        numberRaw = "+7" + numberRaw.Substring(1);
                    }
                }


                return numberRaw;
            }

            public static string ExtractRawPass(string formated) {
                return formated.Replace(" ", "");
            }

            public static string FormatPass(string raw) {
                raw = ExtractRawPass(raw);
                if (raw.Length > 1) {
                    if (raw.Length > 3) {
                                raw = raw.Substring(0, 4) + " " +
raw.Substring(4);
                    }

                                raw = raw.Substring(0, 2) + " " +
raw.Substring(2);
                }

                return raw;
            }
        }

        public enum PropertyKind {
            MovableProperty = 0,
            RealEstate = 1
        }

        static class IDIdentifyable {
```

```
                    public static bool IDEquals(IIDIdentifyable a,
IIDIdentifyable b) {
                if (a is null || b is null)
                    return false;

                return a.ID == b.ID;
            }
        }

        public interface IIDIdentifyable {
            int? ID { get; }
            string DisplayName { get; }
        }


        public interface IProperty : IIDIdentifyable, ISaveable {
            int? ID { get; }

            string Name { get; set; }

            decimal Cost { get; set; }

            IHuman Owner { get; set; }

            PropertyKind Kind { get; set; }
        }

        public interface ICustomer : IIDIdentifyable, ISaveable {
            int? ID { get; }
            IHuman People { get; set; }
            IProperty Property { get; set; }
        }



        class PeopleTableHander : TableHandler<IHuman> {
            private class HumanImpl : IHuman {
                private readonly DataGridViewRow m_row;

                public HumanImpl(DataGridViewRow row) {
                    m_row = row;
                }

                public int? ID => m_row.Cells[0].Value as int?;

                public string LastName {
                        get => m_row.Cells["PeopleLastName"].Value as
string;
                         set => m_row.Cells["PeopleLastName"].Value =
value;
                }
```

```
                public string FirstName {
                       get => m_row.Cells["PeopleFirstName"].Value as
string;
                        set => m_row.Cells["PeopleFirstName"].Value =
value;
                }
                public string Patronymic {
                      get => m_row.Cells["PeoplePatronymic"].Value as
string;
                       set => m_row.Cells["PeoplePatronymic"].Value =
value;
                }
                public string Pass {
                    get => m_row.Cells["PeoplePass"].Value as string;
                    set => m_row.Cells["PeoplePass"].Value = value;
                }
                public string Phone {
                          get => m_row.Cells["PeoplePhone"].Value as
string;
                    set => m_row.Cells["PeoplePhone"].Value = value;
                }
                public string Email {
                          get => m_row.Cells["PeopleEmail"].Value as
string;
                    set => m_row.Cells["PeopleEmail"].Value = value;
                }

                public string DisplayName {
                    get {
                        var name = "";

                        if (!string.IsNullOrWhiteSpace(LastName))
                            name = LastName.Trim();

                        if (!string.IsNullOrWhiteSpace(FirstName))
                            name += " " + FirstName.Trim();

                        if (!string.IsNullOrWhiteSpace(Patronymic))
                            name += " " + Patronymic.Trim();

                            if (!string.IsNullOrWhiteSpace($"{Pass}
{Phone}{Email}")) {
                            name += " (";

                            string innerName = "";

                            if (!string.IsNullOrWhiteSpace(Pass))
                                                innerName = ", " +
Utils.FormatPass(Pass);

                            if (!string.IsNullOrWhiteSpace(Phone))
```

```
                                                    innerName += ", " +
Utils.FormatNumber(Phone);

                                if (!string.IsNullOrWhiteSpace(Email))
                                    innerName += ", " + Email.Trim();

                                if (innerName.Length >= 2)
                                    name += innerName.Substring(2);

                                name += ")";
                            }

                            return name.Trim();
                        }
                    }

                    public void Load(Stream input) {
                                    using (BinaryReader reader = new
BinaryReader(input, Encoding.UTF8, true)) {
                            m_row.Cells[0].Value = reader.ReadInt32();

                            LastName = reader.ReadString();
                            FirstName = reader.ReadString();
                            Patronymic = reader.ReadString();

                            Pass = Utils.FormatPass(reader.ReadString());
                                                                Phone =
Utils.FormatNumber(reader.ReadString());
                            Email = reader.ReadString();
                        }
                    }

                    public void Save(Stream output) {
                                    using (BinaryWriter writer = new
BinaryWriter(output, Encoding.UTF8, true)) {

                            writer.Write(ID.Value);
                            writer.Write(LastName?.Trim() ?? "");
                            writer.Write(FirstName?.Trim() ?? "");
                            writer.Write(Patronymic?.Trim() ?? "");

                                writer.Write(Utils.ExtractRawPass(Pass ??
""));
                             writer.Write(Utils.ExtractRawNumber(Phone ??
""));
                            writer.Write(Email?.Trim() ?? "");
                        }
                    }
                }
```

```
            public PeopleTableHander(DataGridView grid) : base(grid)
{
            }

                                        protected    override    IHuman
CreateBasedOnRow(DataGridViewRow row)
                => new HumanImpl(row);
        }

        public class Pair<T> {
            public T Value { get; set; }
            public string Name { get; set; }
        }

        class PropertiesTableHandler : TableHandler<IProperty> {
            private readonly PeopleTableHander m_peopleHandler;

                    public  PropertiesTableHandler(DataGridView  grid,
PeopleTableHander peopleTableHandler) : base(grid) {
                m_peopleHandler = peopleTableHandler;

                    var kindColumn = Grid.Columns["PropertyKind"] as
DataGridViewComboBoxColumn;
                kindColumn.ValueMember = "Value";
                kindColumn.DisplayMember = "Name";
                kindColumn.DataSource = new Pair<PropertyKind>[] {
                    new Pair < PropertyKind >(){
                        Value = PropertyKind.MovableProperty,
                        Name = "Движимое имущество"
                    },
                    new Pair < PropertyKind >(){
                        Value = PropertyKind.RealEstate,
                        Name = "Недвижимое имущество"
                    }
                };

                grid.CellValidating += (sender, args) => {
                    var col = Grid.Columns["PropertyCost"];


                    if (col.Index == args.ColumnIndex) {
                                                        var   cell   =
Grid.Rows[args.RowIndex].Cells[args.ColumnIndex];
                        var str = args.FormattedValue?.ToString();

                        args.Cancel = !decimal.TryParse(
                            str,
                                                NumberStyles.Integer  |
NumberStyles.AllowDecimalPoint,
                            CultureInfo.InvariantCulture,
                            out _
```

```
                            ) && !string.IsNullOrWhiteSpace(str);


                            cell.ErrorText = args.Cancel ? "Должно быть
число. Может содержать точку в качестве разделителя" : null;
                        }

                    };

                    grid.RowValidating += (sender, args) => {
                        var col = Grid.Columns["PropertyCost"];


                        if (col.Index == args.ColumnIndex) {
                                                                var   cell   =
Grid.Rows[args.RowIndex].Cells[args.ColumnIndex];
                            var str = cell.FormattedValue?.ToString();

                            args.Cancel = !decimal.TryParse(
                                str,
                                                    NumberStyles.Integer  |
NumberStyles.AllowDecimalPoint,
                                CultureInfo.InvariantCulture,
                                out _
                                ) && !string.IsNullOrWhiteSpace(str);


                            cell.ErrorText = args.Cancel ? "Должно быть
число. Может содержать точку в качестве разделителя" : null;
                        }

                    };

                    grid.Validating += (sender, args) => {
                        var col = Grid.Columns["PropertyCost"];
                        for (int x = 0; x < EntryCount; x++) {
                            var cell = Grid.Rows[x].Cells[col.Index];

                            var str = cell.FormattedValue?.ToString();

                            args.Cancel = !decimal.TryParse(
                                str,
                                                    NumberStyles.Integer  |
NumberStyles.AllowDecimalPoint,
                                CultureInfo.InvariantCulture,
                                out _
                                ) && !string.IsNullOrWhiteSpace(str);


                            cell.ErrorText = args.Cancel ? "Должно быть
число. Может содержать точку в качестве разделителя" : null;
```

```
                        if (args.Cancel)
                            return;
                    }
                };
            }

            // TODO: DRY!!!!!
            public override void Validate() {
                var col = Grid.Columns["PropertyCost"];
                for (int x = 0; x < EntryCount; x++) {
                    var cell = Grid.Rows[x].Cells[col.Index];

                    var str = cell.FormattedValue?.ToString();

                    bool cancel = !decimal.TryParse(
                        str,
                                                NumberStyles.Integer  |
NumberStyles.AllowDecimalPoint,
                        CultureInfo.InvariantCulture,
                        out _
                        ) && !string.IsNullOrWhiteSpace(str);


                       cell.ErrorText = cancel ? "Должно быть число.
Может содержать точку в качестве разделителя" : null;
                    if (cancel)
                        return;
                }
            }

            class PropertyImpl : IProperty {
                private readonly DataGridViewRow m_row;
                private readonly PropertiesTableHandler m_handler;

                                public  PropertyImpl(DataGridViewRow  row,
PropertiesTableHandler owner) {
                    m_row = row;
                    m_handler = owner;
                }

                public int? ID => m_row.Cells[0].Value as int?;

                public string Name {
                          get  =>  m_row.Cells["PropertyName"].Value  as
string;
                    set => m_row.Cells["PropertyName"].Value = value;
                }
                public decimal Cost {
                    get => decimal.TryParse(
                         m_row.Cells["PropertyCost"].Value?.ToString()
?? "",
```

```
                                                    NumberStyles.Integer |
NumberStyles.AllowDecimalPoint,
                        CultureInfo.InvariantCulture, out var d)
                        ? d : 0;

                          set => m_row.Cells["PropertyCost"].Value =
value.ToString(CultureInfo.InvariantCulture);
                }
                public IHuman Owner {
                       get => m_row.Cells["PropertyOwnerID"].Value as
IHuman;
                        set => m_row.Cells["PropertyOwnerID"].Value =
value;
                }
                public PropertyKind Kind {
                          get => m_row.Cells["PropertyKind"].Value as
PropertyKind? ?? PropertyKind.RealEstate;
                    set => m_row.Cells["PropertyKind"].Value = value;
                }

                public string DisplayName {
                    get {
                        string name = Name;

                        if (Owner is object)
                                                name += $" (Владеет:
{Owner.DisplayName})";

                        return name;
                    }
                }

                public void Load(Stream input) {
                                using (BinaryReader reader = new
BinaryReader(input, Encoding.UTF8, true)) {
                        m_row.Cells[0].Value = reader.ReadInt32();
                        Name = reader.ReadString();
                        Cost = reader.ReadDecimal();
                        Kind = (PropertyKind)reader.ReadByte();
                                                                Owner =
m_handler.m_peopleHandler.FindByID(reader.ReadInt32());
                    }
                }

                public void Save(Stream output) {
                                using (BinaryWriter writer = new
BinaryWriter(output, Encoding.UTF8, true)) {
                        writer.Write(ID.Value);
                        writer.Write(Name?.Trim() ?? "");
                        writer.Write(Cost);
                        writer.Write((byte)Kind);
```

```
                        writer.Write(Owner?.ID ?? -1);
                    }
                }
            }



            public override void UpdateComboBoxes() {

                UpdateList("PropertyOwnerID", m_peopleHandler);
            }

                                    protected    override    IProperty
CreateBasedOnRow(DataGridViewRow row)
                => new PropertyImpl(row, this);
        }

        class CustomersTableHandler : TableHandler<ICustomer> {
            private readonly PeopleTableHander m_peopleHandler;
                            private   readonly   PropertiesTableHandler
m_propertiesHandler;

                     public  CustomersTableHandler(DataGridView  grid,
PeopleTableHander p, PropertiesTableHandler c) : base(grid) {
                m_peopleHandler = p;
                m_propertiesHandler = c;
            }

            class CustomerImpl : ICustomer {
                private readonly DataGridViewRow m_row;
                private readonly CustomersTableHandler m_handler;

                             public  CustomerImpl(DataGridViewRow  row,
CustomersTableHandler owner) {
                    m_row = row;
                    m_handler = owner;
                }

                public int? ID => m_row.Cells["CustomersID"].Value as
int?;

                public IHuman People {
                      get => m_row.Cells["CustomersHumanID"].Value as
IHuman;
                       set => m_row.Cells["CustomersHumanID"].Value =
value;
                }
                public IProperty Property {
                      get => m_row.Cells["CustomersPropertyID"].Value
as IProperty;
```

```
                    set => m_row.Cells["CustomersPropertyID"].Value =
value;

                }

                  public string DisplayName => $"{People.DisplayName}
хочет {Property.DisplayName}";

                public void Load(Stream input) {
                                using (BinaryReader reader = new
BinaryReader(input, Encoding.UTF8, true)) {

                        m_row.Cells[0].Value = reader.ReadInt32();
                                                            People =
m_handler.m_peopleHandler.FindByID(reader.ReadInt32());
                                                          Property =
m_handler.m_propertiesHandler.FindByID(reader.ReadInt32());
                    }
                }

                public void Save(Stream output) {
                                using (BinaryWriter writer = new
BinaryWriter(output, Encoding.UTF8, true)) {
                        writer.Write(ID.Value);
                        writer.Write(People?.ID ?? -1);
                        writer.Write(Property?.ID ?? -1);
                    }
                }
            }

            public override void UpdateComboBoxes() {

                UpdateList("CustomersHumanID", m_peopleHandler);
                                  UpdateList("CustomersPropertyID",
m_propertiesHandler);
            }

                                protected override ICustomer
CreateBasedOnRow(DataGridViewRow row)
                => new CustomerImpl(row, this);
        }
    }
```