МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

«Новосибирский национальный исследовательский государственный университет»

(Новосибирский государственный университет)

Структурное подразделение Новосибирского государственного университета –

Высший колледж информатики Университета

КАФЕДРА ИНФОРМАТИКИ

ОТЧЁТ КУРСОВОГО ПРОЕКТА

ПМ.01. Разработка программных модулей

**РАЗРАБОТКА приложения для управления базой данных «MANAGESTAFF» на wpf**

|  |  |
| --- | --- |
| Преподаватель доцент ВКИ НГУ | Голкова Н.В.  «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_2021 г. |
| Студент 2 курса гр. 007сб2 | Колесова Г.И.  «\_\_\_\_»\_\_\_\_\_\_\_\_\_\_2021 г. |

Новосибирск

2021

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc90764021)

[1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ 4](#_Toc90764022)

[2 ПОСТАНОВКА ЗАДАЧИ 5](#_Toc90764023)

[3 АНАЛОГИ 6](#_Toc90764024)

[4 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ 10](#_Toc90764025)

[4.1 Диаграмма претендентов 10](#_Toc90764026)

[4.2 ER-диаграмма базы данных 10](#_Toc90764027)

[5 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ 11](#_Toc90764028)

[5.1 Требования к программному обеспечению 11](#_Toc90764029)

[5.2 Требования к аппаратному обеспечению 11](#_Toc90764030)

[6 ХАРАКТЕРИСТИКИ ВЫБРАННЫХ ПРОГРАММНЫХ СРЕД И СРЕДСТВ 12](#_Toc90764031)

[6.1 Среда разработки Visual Studio 2019 12](#_Toc90764032)

[6.2 Применённые технологии 13](#_Toc90764033)

[7 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ 14](#_Toc90764034)

[7.1 Алгоритмы реализации программного средства 14](#_Toc90764035)

[7.2 Функционирование программного средства 14](#_Toc90764036)

[8 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ 16](#_Toc90764037)

[8.1 Описание базы данных 16](#_Toc90764038)

[9 ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА 18](#_Toc90764039)

[9.1 Работа приложения 19](#_Toc90764040)

[9.2 Элементы управления 19](#_Toc90764041)

[9.3 Функциональные возможности 19](#_Toc90764042)

[10 ТЕСТИРОВАНИЕ И ОТЛАДКА 21](#_Toc90764043)

[10.1 Сведения о ходе испытаний 21](#_Toc90764044)

[10.2 Вывод по результатам испытаний 21](#_Toc90764045)

[11 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ 23](#_Toc90764046)

[ЗАКЛЮЧЕНИЕ 26](#_Toc90764047)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc90764048)

[ПРИЛОЖЕНИЕ 28](#_Toc90764049)

# **ВВЕДЕНИЕ**

По мере развития предприятия, оно расширяется, включая в себя всё больше новых отделов. С появлением отделов, вырастает и количество вакансий, пропорционально с которыми увеличивается число работников. Компания должна хранить информацию о каждом сотруднике каждого отдела. Процесс эволюции предприятия ставит людей перед новой задачей – создание базы данных для структурирования информации.

Базы данных позволяют хранить, изменять и удалять информацию о конкретном сотруднике и об отделах в целом. Для этого необходимы специализированных программы, к которым предъявляется ряд требований, ключевым из которых является требование к простоте и функциональности интерфейса программы.

В настоящее время трудно представить себе организацию, не имеющую собственной базы данных. Свою информацию имеют торговые фирмы и промышленные предприятия, больницы и школы, органы власти и спортивные команды, институты и военкоматы, и большое количество разноплановых фирм. Более того – наличие систематизированной базы данных в наши дни является необходимостью.

# **1 ОПИСАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ**

В качестве предметной области выбрана программа – приложение, которое предназначено для управления базой данных сотрудников предприятия. Информационная система (ИС) данной программы занимается добавлением, удалением и редактированием информации о сотрудниках предприятия и занимаемых ими должностях.

Приложение ManageStaff – состоит из восьми окон. Содержит сведения об отделах предприятия, должностях, представленных в отделах, и о сотрудниках, занимаемых конкретные должности. То есть, программа представляет собой базу данных, содержащую в себе информацию о сотрудниках.

Разрабатываемая программа структурирует объёмы информации, связанные с сотрудниками, а именно: информацию об их имени и фамилии, номера телефонов и занимаемых ими должностях в отделах компании.

# **2 ПОСТАНОВКА ЗАДАЧИ**

Цель данного курсового проекта – получить практические навыки по разработке приложений на языке программирования C# и вёрстке на XAML.

Задачей проекта является разработка специализированной системы управления базой данных для управления информацией о сотрудниках предприятия. Объём требований к информационному наполнению данной программы позволяет создать быстрое и удобное для пользователя приложение на основе конкретных функций. Каждое окно формируется из специализированных блоков. Каждый блок имеет свой функционал и может содержать как заранее введённую информацию, так и поля для ввода новой информации.

Для достижения поставленной цели необходимо решить следующие задачи:

1. Изучить функциональные возможности языка C#;
2. Изучить функциональные возможности языка XAML;
3. Определить, какое приложение будет реализовано на данном языке;
4. Спроектировать концепцию и алгоритмы, необходимые для работы программы;
5. Спроектировать дизайн программы при помощи языка XAML;
6. Реализовать алгоритмы в среде программирования Visual Studio 2019;
7. Организовать простой и удобный интерфейс для работы программы;

# **3 АНАЛОГИ**

В процессе исследования предметной области, был найден и изучен ряд приложений со схожей тематикой.

1. Workbench – инструмент для визуального проектирования баз данных, интегрирующий проектирование, моделирование, создание и эксплуатацию БД в единое бесшовное окружение для системы баз данных MySQL.

Среди достоинств программы можно выделить следующие параметры:

* Возможность представить модель БД в графическом виде, а также редактирование данных в таблице;
* Наличие простого и функционального механизма по созданию связей между полями таблиц, среди которых реализована связь «многие-ко-многим» с возможностью создания таблицы связей;
* Функция Reverse Engineering позволяет восстанавливать структуру таблиц и связей из той, которая была реализована ранее и хранится на сервере БД;
* Наличие редактора SQL-запросов, который дает возможность при отправке на сервер получать ответ в табличном виде и другие возможности;

К недостаткам программы можно отнести следующие параметры:

* Необходимо знать язык SQL-запросов;
* Большие функциональные возможности, большинство из которых не пригодится для базовых, повседневных задач в небольшой компании;

1. Navicat – инструмент для разработки и администрирования баз данных, который работает на любом сервере MySQL, начиная с версии 3.21.

Среди достоинств программы можно выделить следующие параметры:

* Наличие визуального конструктора запросов;
* Возможность импорта, экспорта и резервного копирования данных;
* Возможность создавать отчеты;
* SSH и HTTP туннелинг;
* Миграция и синхронизация данных и структуры;
* Инструмент для планирования задач;

К недостаткам программы можно отнести следующие параметры:

* Необходимо знать язык SQL-запросов;
* Большие функциональные возможности, большинство из которых не пригодится для базовых, повседневных задач в небольшой компании;

1. Microsoft Office Access – реляционная система управления базами данных корпорации Microsoft.

Среди достоинств программы неоспоримо выделяются:

* Широкий функционал;
* Единая технология создания и использования баз данных, форм, запросов и отчетов;
* Возможность совместной работы базы данных с другими внешними источниками данных (базы данных с СУБД Access и других типов СУБД, электронная таблица Excel, текстовые файлы);
* Наличие различных мастеров, конструкторов и построителей;
* Полная русификация СУБД Access;
* Простота в работе;
* Отсутствие собственного языка программирования;
* Запрос может использоваться наравне с таблицами в качестве источника записей для отчетов, форм и запросов;
* База данных и все ее объекты (таблицы, запросы, формы, отчеты, макросы и модули) храняться в одном файле;

Среди недостатков программы выделяются следующие параметры:

* Входит в состав пакета Microsoft Office Professional Plus;
* Слабые средства защиты и восстановления информации;
* Ограничения на объем информации;
* Отсутствие собственного языка программирования;
* Низкая скорость при работе с большими объемами информации;
* Размер файла базы данных (с расширением mdb) — 2 Гб за вычетом места, необходимого системным объектам. Реально размер ограничивается доступным местом на диске, так как БД может включать присоединенные таблицы;
* Число объектов в базе данных — 32768;
* Количество одновременно работающих пользователей — 255;
* Максимальный размер таблицы — 2 Гбайт;
* Максимальное количество полей в таблице — 255;
* Максимальное количество индексов в таблице — 32;
* Максимальное число символов в записи (не считая поля Memo и поля объектов OLE) — 2000;
* Максимальное число символов в поле Memo — 65 535 при вводе данных через интерфейс пользователя и 1 Гбайт при программном вводе данных;
* Максимальный размер объекта OLE — 1 Гбайт;
* Максимальное количество таблиц в запросе — 32;

Для подведения итогов по исследованию аналогов, выделим основные критерии для сравнения аналогов и разрабатываемого нами ПО (данные сайта будут оцениваться о трёх бальной шкале, где 3 – это отлично, 2 – нормально, 1 – плохо):

1. Дизайн очень важная составляющая приложения, способная как привлекать новых пользователей, так и отталкивать имеющихся);
2. Удобство (свойство приложения, позволяющее пользователям достигать своих целей с нужной им результативностью и удовлетворенностью);
3. Функциональность (набор полезных опций, которые есть в приложении. Совокупность «инструментов», которые помогают пользователю);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название | Дизайн | Удобство | Функциональность | Итого |
| «Workbench» | 2 | 1 | 2 | 5 |
| «Navicat» | 2 | 1 | 2 | 5 |

Продолжение Таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| «Microsoft Office Access» | 3 | 3 | 2 | 8 |

Таблица 1 – Сравнение аналогов.

Подводя итоги по данным таблицы 1, можно сделать вывод, что большинство из представленных программ в основном соответствуют указанным критериям. Но при этом, большая половина приложений проигрывает в таких критериях, как «дизайн» и «удобство». Можно предположить, что это происходит из-за перегрузки функционалом, который создатели хотели преподнести пользователям.

# **4 ФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ**

## **4.1 Диаграмма претендентов**

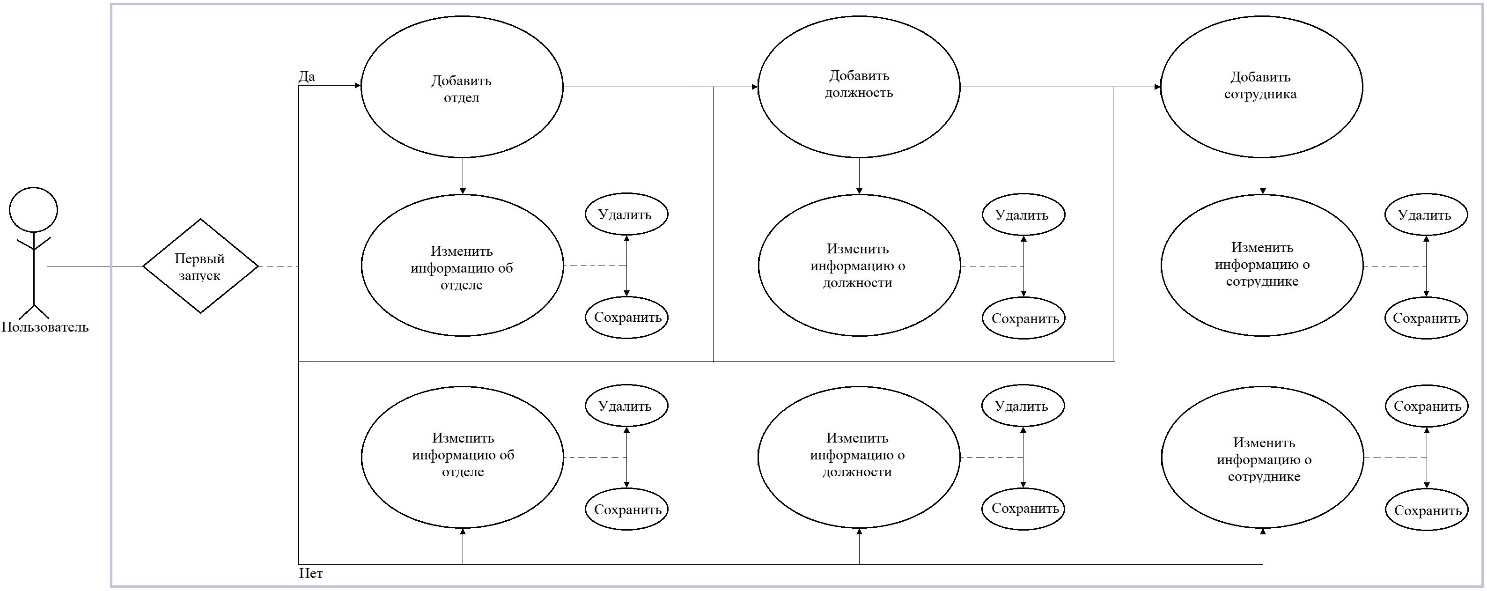
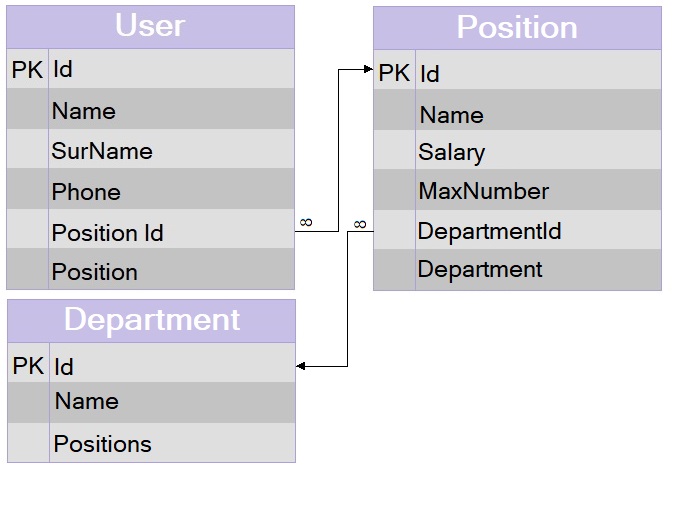


Рисунок 1 – Диаграмма претендентов

## **4.2 ER-диаграмма базы данных**

****Рисунок 2 – ER-диаграмма базы данных.

# **5 НЕФУНКЦИОНАЛЬНЫЕ ТРЕБОВАНИЯ К ПРОГРАММНОМУ ПРОДУКТУ**

5.1 Требования к программному обеспечению

Для функционирования приложения ManageStaff необходимо наличие следующего программного обеспечения:

1. Microsoft .NET Framework версии 3.5 или выше;
2. Операционная система Windows 7, 8.1, 10 или 11.

5.2 Требования к аппаратному обеспечению

Приложение ManageStaff относительно не требовательна к аппаратным ресурсам компьютера. Ниже приведены минимальные системные требования, которые необходимы для поддержания корректной работы приложения:

* 32-разрядный (x86) или 64-разрядный (x64) процессор с тактовой частотой 1 ГГц или выше;
* 2 ГБ (для 32-разрядного процессора) или 4 ГБ (для 64-разрядного процессора) ОЗУ;
* 250 МБ (для 32-разрядной системы) или 300 МБ (для 64-разрядной системы) свободного места на жестком диске;
* Наличие устройств периферии: мыши и клавиатуры.

# **6 ХАРАКТЕРИСТИКИ ВЫБРАННЫХ ПРОГРАММНЫХ СРЕД И СРЕДСТВ**

Требованиями к приложению являются стабильная работа без сбоев и корректное отображение контента при уменьшении и расширении окна.

Основными критериями отбора программных средств и сред для разработки проекта послужили доступность и многофункциональность, простота использования.

Исходя из этого, в качестве среды разработки была выбрана Visual Studio 2019, а также язык программирования C#.

## **6.1 Среда разработки Visual Studio 2019**

Visual Studio 2019 – среда разработки, включающая интегрированную среду разработки программного обеспечения и ряд других инструментов. Данные продукты позволяют разрабатывать как консольные приложения, так и игры, приложения с графическим интерфейсом, в том числе с поддержкой технологии Windows Forms, Windows Presentation Forms, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

К неоспоримым преимуществам Visual Studio 2019 можно отнести:

* Множество настроек (как всей программы, так и интерфейса);
* Расширяемая библиотека дополнений и готовых решений;
* Мультифункциональность (редактор поддерживает почти все языки, используемые для создания приложений);
* Простота и гибкость;
* Встроенные инструменты интеграции с Git и GitHub.
* Интерфейс можно разделить на две панели для сравнения и удобного редактирования кода;

## **6.2 Применённые технологии**

1. Управляемые классы, которые хранят в себе методы, обеспечивающие работу программного продукта;
2. Методы работы с базой данных;
3. Связи и переходы между окнами в WPF приложении;
4. Элементы WPF форм, позволяющие реализовать графический интерфейс для простой работы с программой. Среди них присутствуют: TabControl, ContextMenu, GridView, StackPanel, Button, TextBlock, TextBox, ComboBox и другие;
5. Фреймворк ADO.NET Entity Framework – объектно-ориентированная технология доступа к данным, является object-relational mapping решением для .NET Framework от Microsoft. Предоставляет возможность взаимодействия с объектами как посредством LINQ в виде LINQ to Entities, так и с использованием Entity SQL;
6. Паттерн MVVM (Model-View-ViewModel) – паттерн проектирования, основной сутью которого является отделение логики приложения от его визуальной части. Методы и вёрстка реализуются отдельно и не зависят друг от друга;

# **7 АЛГОРИТМ РЕШЕНИЯ ЗАДАЧИ**

7.1 Алгоритмы реализации программного средства

Алгоритм реализации программы ManageStaff состоит из пяти этапов. Наименование, последовательность и описание каждого из этапов перечислены ниже.

1. Анализ задачи – подробный разбор условий поставленной задачи, а также е требований. На данном этапе было сформировано общее видение программы и основной функционал, который программа должна нести;
2. Построение модели задачи – оформление результатов анализа, построение схемы программы (диаграмма претендентов, представленная на рисунке 1, расположенном выше);
3. Поиск способа решения задачи – разбиение крупной задачи на подзадачи. Поиск оптимальных вариантов решения каждой из них;
4. Осуществление решения задачи – начало реализации визуальной составляющей программы, создание и заполнение основных моделей (User.cs, Departments.cs, Position.cs). Подключение Entity Framework, создание миграций. Реализация классов DataWorker.cs и DataManageVM.cs. Создание базы данных и подключение к проекту через класс ApplicationContext. Связывание View и Model;
5. Проверка решения задачи – тестирование программы методом Чёрного ящика (результаты тестирования приведены ниже). Исправление возникших ошибок.

7.2 Функционирование программного средства

Работа с приложением начинается с запуска главного окна программы. На верхней панели меню перед пользователем представлены три кнопки:

1. «Сотрудники» – открывает таблицу, в которой содержится информация о работниках предприятия. В таблице присутствуют шесть колонок: «Имя», «Фамилия», «Телефон», «Зарплата», «Должность», «Отдел»;
2. «Должности» – открывает таблицу, в которой содержится информация о должностях, представленных в предприятии. В таблице присутствуют пять колонок: «Название», «Зарплата», «Количество сотрудников», «Максимальное количество», «Отдел»;
3. «Отдел» – открывает таблицу, в которой содержится информация о каждом из отделов в предприятии. Таблица состоит из двух колонок: «Название» и «Количество вакансий»;

При первичном запуске программы все таблицы пусты. Количество строк будет пополняться по мере заполнения базы данных пользователем. Информацию в каждой из таблиц можно редактировать и удалять.

На нижней панели расположены три кнопки:

1. «Новый сотрудник» – открывает окно для добавления сотрудника;
2. «Новая должность» – открывает окно для добавления должности;
3. «Новый отдел» – открывает окно для добавления отдела;

Каждое окно состоит из дополнительных блоков, осуществляющих определённые функции для взаимодействия с базой данных. Подробное описание каждого окна будет представлено далее, в блок-схеме программы.

В процессе работы с приложением пользователь имеет возможность добавлять, удалять и редактировать данные об отделах предприятия, добавлять, удалять и редактировать информацию о сотрудниках, работающих на предприятии, а также добавлять, удалять и редактировать информацию о должностях, занимаемых сотрудниками и обновлять количество вакантных мест.

# **8 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ**

8.1 Описание базы данных

Одноимённая база данных в программе ManageStaff создаётся автоматически, при помощи метода EnsureCreated, который находится в конструкторе класса ApplicationContext.

1. User.cs – класс, описывающий модель сотрудника. На его основе создаётся одноимённая таблица в базе данных. Подробное описание полей класса представлены в таблице 2, расположенной ниже.

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| Id | INTEGER, PRIMARY KEY | Первичный ключ |
| Name | STRING | Хранит имя сотрудника |
| Surname | STRING | Хранит фамилию сотрудника |
| Phone | STRING | Хранит номер телефона сотрудника |
| PositionId | INTEGER | Внешний ключ к таблице «Position» |
| Position | STRING | Хранит наименование должности |

Таблица 2 – Описание таблицы User.

1. Position.cs – класс, описывающий модель должности. На его основе создаётся одноимённая таблица в базе данных. Подробное описание полей класса представлено в таблице 3, расположенной ниже.

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| Id | INTEGER, PRIMARY KEY | Первчиный ключ |
| Name | STRING | Хранит наименование должности |

Продолжение таблицы 3

|  |  |  |
| --- | --- | --- |
| Salary | DECIEMAL | Хранит информацию о зарплате |
| MaxNumber | INTEGER | Хранит информацию о максимальном числе вакансий |
| DepartmentId | INTEGER | Внешний ключ к таблице «Department» |
| Department | STRING | Хранит наименование отдела |

Таблица 3 – Описание таблицы Position.

1. Department.cs – класс, описывающий модель должности. На его основе создаётся одноимённая таблица в базе данных. Подробное описание полей класса представлено в таблице 4, расположенной ниже.

|  |  |  |
| --- | --- | --- |
| Наименование | Тип | Описание |
| Id | INTEGER, PRIMARY KEY | Первичный ключ |
| Name | STRING | Хранит наименование должности |
| Positions | STRING | Внешний ключ к таблице «Position» |

Таблица 4 – описание таблицы Department.

# **9 ПРОЕКТИРОВАНИЕ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА**

Программа представлена интерфейсом в виде графического приложения для работы с базой данных сотрудников предприятия. Пользовательский интерфейс реализован с использованием технологии Windows Presentation Foundation и языка XAML. Программа разработана под операционную систему Windows.

XAML – расширяемый язык разметки для приложений, включающий в себя основные четыре категории элементов: панели, элементы управления, элементы, связанные с документом, и графические фигуры.

В разработанном приложении были использованы следующие элементы WPF форм:

1. GridView – для отображения данных в таблице, каждый столбец которой представляет поле, а каждая строка представляет запись. Элемент управления GridView позволяет выбрать, сортировать и изменять эти элементы. Используется во вкладках «Сотрудники», «Должности», «Отделы», показывающие данные из одноимённых таблиц в базе данных;
2. Button – элемент управления «кнопка Windows». Используется во всех окнах для открытия других окон и сохранения данных;
3. TextBlock – элемент предназначен для вывода текстовой информации. Ключевым свойством здесь является свойство Text, которое задает текстовое содержимое;
4. TextBox – текстовое поле, используемое ввода и редактирования текста.
5. TabControl – используется для создания вкладок TabItem;
6. TabItem – элемент управления содержимым, в который вложены другие элементы. Используется для открытия таблиц «Сотрудники», «Должности», «Отделы»;
7. StackPanel – используется во всех окнах для расположения всех элементов в ряд либо по горизонтали, либо по вертикали в зависимости от ориентации.
8. ContextMenu – представляет контекстное меню, которое используется для редактирования и удаления элементов из таблиц «Сотрудники», «Должности», «Отделы»;
9. ComboBox – содержит коллекцию элементов и образует выпадающий список. Используется при выборе должности, которая привязана к конкретному отделу в окнах создания и редактирования сотрудников, а также при выборе отдела, к которому будет привязана должность в окнах создания и редактирования отделов;

Также, при проектировании был применён паттерн проектирования MVVM.

Паттерн MVVM (Model-View-ViewModel) – паттерн проектирования, основной сутью которого является отделение логики приложения от его визуальной части. Методы и вёрстка реализуются отдельно и не зависят друг от друга.

## **9.1 Работа приложения**

Программа позволяет пользователю управлять базой данных сотрудников предприятия без дополнительных знаний о языке SQL или программе Access. Простой и интуитивно понятный интерфейс даёт возможность добавлять, удалять и редактировать базу данных простыми нажатиями на кнопки.

## **9.2 Элементы управления**

Левая кнопка мыши – основное управление, применяемое во всём приложении для добавления и изменения данных. Для управления контекстным меню используется правая кнопка мыши.

## **9.3 Функциональные возможности**

Пользователь может добавлять в базу данных отделы, должности и сотрудников. Каждый параметр может быть изменён и отредактирован в любое время. Данные в таблице обновляются автоматически. Также, для удобства просмотра таблиц добавлена функция расширения и сужения столбцов.

Предусмотрены функции разворота приложения на весь экран, растягивания окон и уменьшения до определённых минимальных размеров. Ограничение по размерам существует для корректного отображения интерфейса. Если уменьшить окно слишком сильно, элементы интерфейса будут наслаиваться друг на друга, что, в конечном итоге, приведёт к невозможности работы с программой.

# **10 ТЕСТИРОВАНИЕ И ОТЛАДКА**

В ходе тестирования программы ManageStaff был применён метод чёрного ящика.

Суть тестирования заключается в проверке всех обязательных функций программы, заявленных разработчиком в руководстве пользователя. Тестирование программного продукта производилось без затрагивания внутренней структуры кода, что является ключевой особенностью данного метода.

## **10.1 Сведения о ходе испытаний**

В приведённой ниже таблице представлен список проведённых проверок и заключительный результат по каждой из них.

|  |  |  |  |
| --- | --- | --- | --- |
| № | Проверка | Результат | Замечание |
| 1 | Приложение открывается и корректно отображает данные | Положительно | Отсутствует |
| 2 | Корректная работа кнопок | Положительно | Отсутствует |
| 3 | Добавление информации в  таблицы | Положительно | Отсутствует |
| 4 | Редактирование информации | Положительно | Отсутствует |
| 5 | Удаление информации | Положительно | Отсутствует |
| 6 | Адаптация интерфейса под размер окна | Положительно | Отсутствует |

Таблица 5 – Сведения о ходе испытаний.

## **10.2 Вывод по результатам испытаний**

В процессе испытаний дефекты и ошибки не были обнаружены. При изменении размеров окна элементы интерфейса адаптируются под размер. Благодаря установленным минимальным размерам окон, при уменьшении элементы не накладываются друг на друга. Программа функционирует без сбоев. Весь заявленный функционал выполняется. Пользователь не сталкивается с ошибками. Интерфейс программы выполнен лаконично, без лишних деталей. Весь функционал прост, благодаря чему не вызовет затруднений при работе с ним, что является неоспоримым плюсом для конечного потребителя – рядового пользователя компьютера, не обладающего знаниями в сфере разработки баз данных.

# **11 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ**

Открывая приложение, пользователь попадает на главную страницу. Скриншот главной страницы сайта представлен на рисунке 3, приведённом ниже.

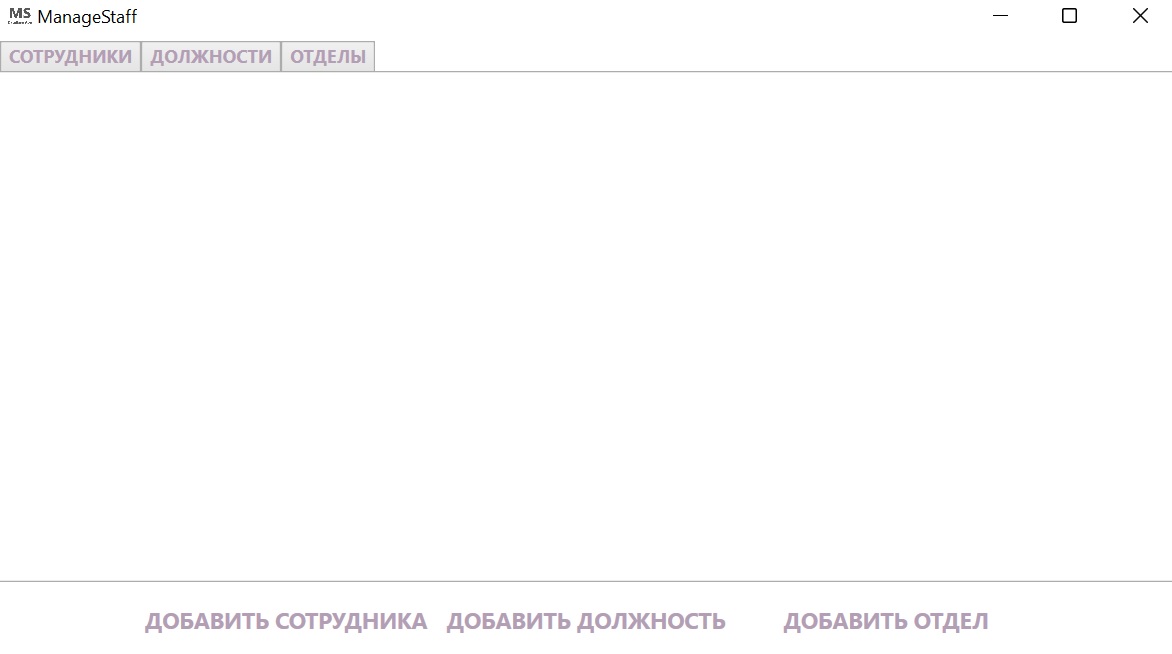


Рисунок 3 – главная страница приложения ManageStaff

В левом верхнем углу страницы представлены три вкладки, с помощью которых осуществляется просмотр таблиц из базы данных, а также логотип и название программы. В правом верхнем углу три кнопки: свернуть программу, развернуть окно на полный экран, закрыть программу.

При нажатии на одну из трёх вкладок открывается соответствующая таблица из базы данных. При первоначальном запуске программы все таблицы пусты. В ходе работы с программой пользователь постепенно заполняет каждую из таблиц. На рисунке 4, расположенном ниже, представлен вариант с уже заполненной таблицей «Сотрудники». Данные в таблице можно удалять и редактировать в любое время. Окна редактирования идентичны окнам создания, с разницей в том, что поля в окнах создания изначально пусты, а при редактировании они заполнены введённой ранее информацией.

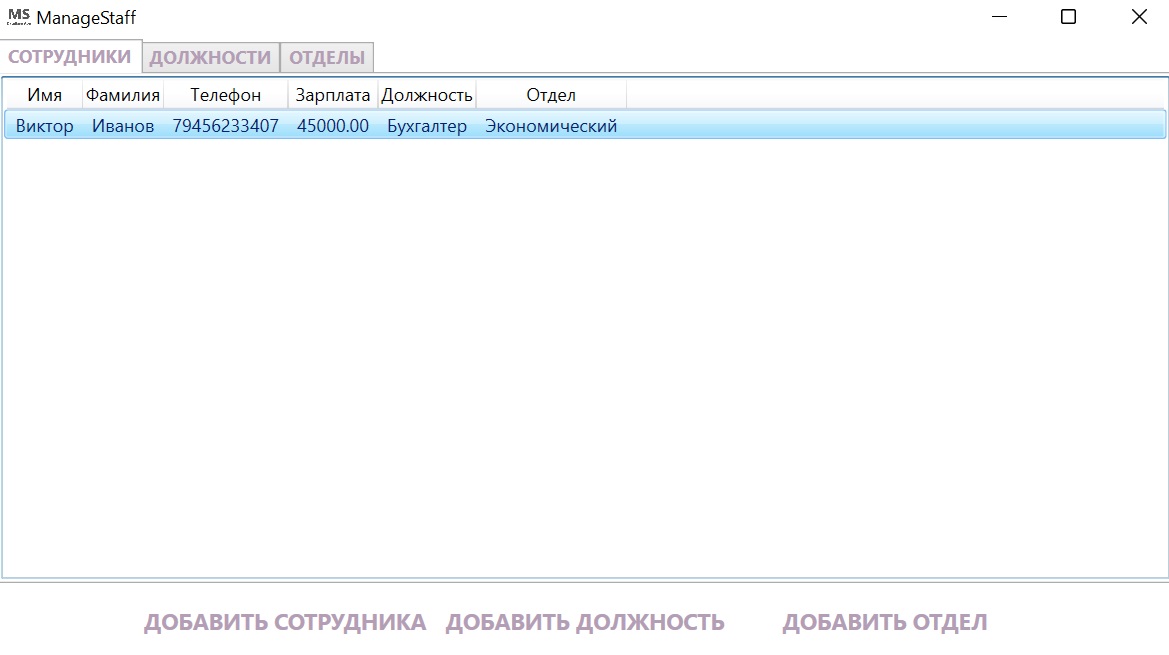


Рисунок 4 – Заполненная таблица «Сотрудники»

Внизу экрана расположены три кнопки, с помощью которых осуществляется взаимодействие с базой данных. Детальное рассмотрение кнопок представлено на рисунке 5, расположенном ниже.

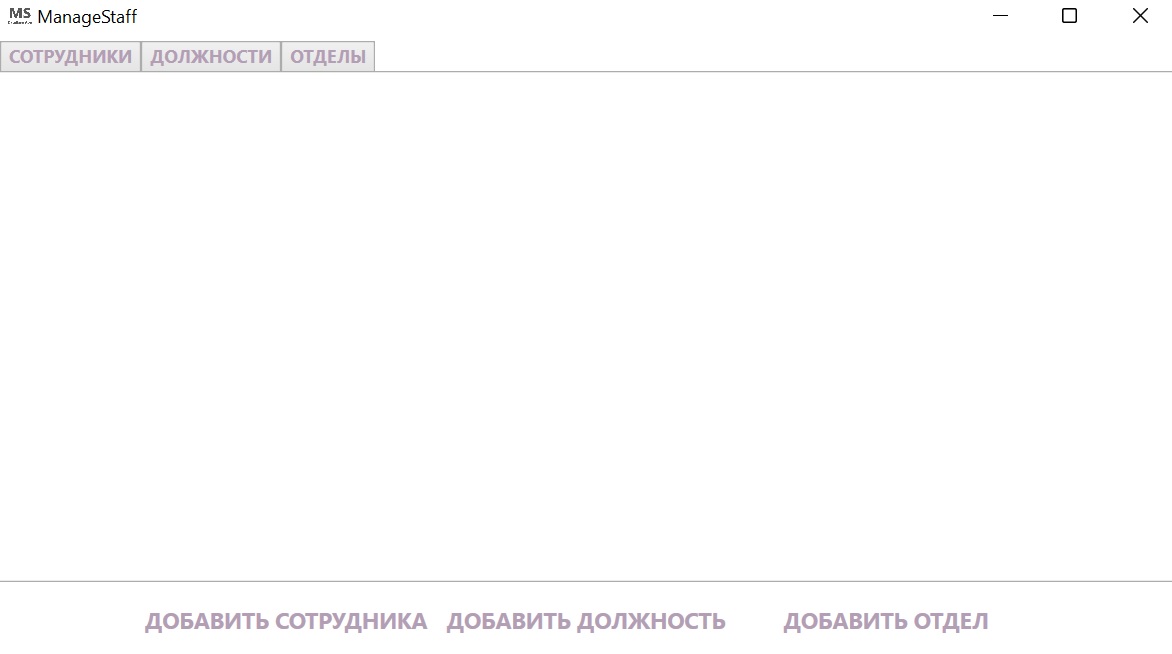


Рисунок 5 – Кнопки для работы с базой данных.

Для начала работы с программой, в первую очередь необходимо добавить отделы предприятия. Для этого на нижней панели представлена кнопка «Добавить отдел». Скриншот окна для добавления отдела представлен на рисунке 6, расположенном ниже.

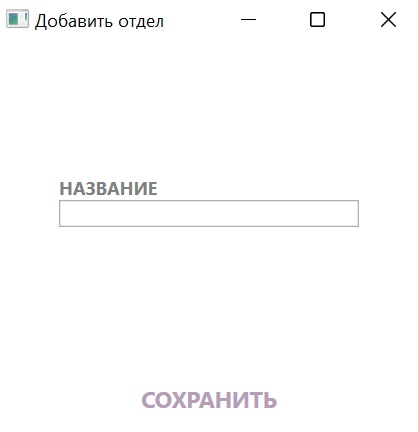


Рисунок 6 – Окно добавления отдела

Окна добавления и редактирования отделов, должностей и сотрудников идентичны. На 7 и 8 рисунках, расположенных ниже, представлены скриншоты окон добавления должности и добавления сотрудника.

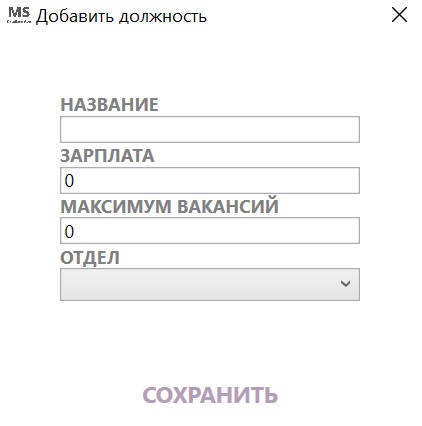


Рисунок 7 – Окно добавления должности

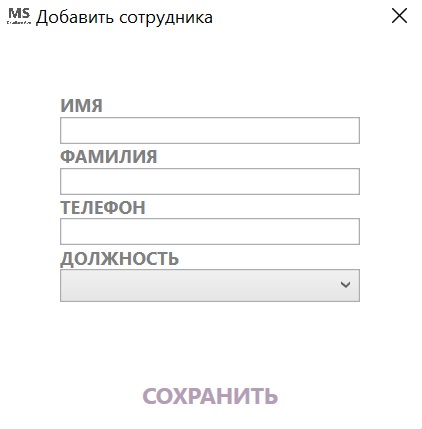


Рисунок 8 – Окно добавления сотрудника

При успешном выполнении операции программа показывает окно с оповещением.

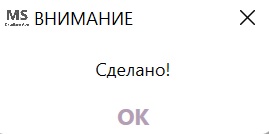


Рисунок 9 – Оповещение

# **ЗАКЛЮЧЕНИЕ**

В ходе выполнения работы, было спроектировано и реализовано приложение для простой и удобной работы с базой данных. Мною были приобретены следующие навыки:

* Работа в среде разработки Visual Studio 2019;
* Навык написания кода на языке C#;
* Навык вёрстки окон на языке XAML;
* Работа с фреймворком ADO.NET Entity Framework;
* Проектирование программы согласно паттерну проектирования MVVM;
* Выполнение поставленных задач в указанные сроки;

Поставленную задачу считаю в полной мере выполненной.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#/ Джеффри Рихтер / «CLR via C#», 4-е издание, 2020 –– C. 140-398.
2. Разработка программных модулей [Курс лекций]: лектор / Д.В. Лазарев: ВКИ НГУ, 2021.
3. Технология разработки и защиты баз данных [Курс лекций]: лектор/ Т.А. Иванчева: ВКИ НГУ, 2021.
4. Сайт о программировании на C# [<https://metanit.com>].
5. Документация Microsoft [<https://docs.microsoft.com/ru-ru/documentation/>].

# **ПРИЛОЖЕНИЕ**

Ссылка на полный код программы: https://github.com/KolesovaGI/ManageStaff

ApplicationContext.cs

public DbSet<User> Users { get; set; } // Таблица сотрудников

public DbSet<Position> Positions { get; set; } // Таблица должностей

public DbSet<Department> Departments { get; set; } // Таблица отделов

/// <summary>

/// Конструктор класса

/// </summary>

public ApplicationContext()

{

Database.EnsureCreated(); // Если база данных не существует, метод создаст её

}

/// <summary>

/// Подключение базы данных

/// </summary>

/// <param name="optionsBuilder"></param>

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

{

otionsBuiler.UseSqlServer ("Server=(localdb)\\mssqllocaldb;Database=ManageStaffDBAppDB;Trusted\_Connection=True;");

}

DataWorker.cs

/// <summary>

/// Получение всех отделов

/// </summary>

/// <returns>Список отделов</returns>

public static List<Department> GetAllDepartments()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Departments.ToList(); // Результат - возврат списка отделов

return result;

}

}

/// <summary>

/// Получение всех должностей

/// </summary>

/// <returns>Список должностей</returns>

public static List<Position> GetAllPositions()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Positions.ToList(); // Результат - возврат списка должностей

return result;

}

}

/// <summary>

/// Получение всех сотрудников

/// </summary>

/// <returns>Список всех сотрудников</returns>

public static List<User> GetAllUsers()

{

using (ApplicationContext db = new ApplicationContext())

{

var result = db.Users.ToList(); // Результат - вовзрат списка сотрудников

return result;

}

}

/// <summary>

/// Создание отдела

/// </summary>

/// <param name="name">Название отдела</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string CreateDepartment(string name)

{

string result = "Уже существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

bool checkIsExist = db.Departments.Any(el => el.Name == name); // Проверка на существование отдела

if (!checkIsExist) // Если отдел не существует

{

Department newDepartment = new Department { Name = name }; // Новый отдел

db.Departments.Add(newDepartment); // Добавление отдела в таблицу отделов

db.SaveChanges(); // Сохранение изменений

result = "Сделано!"; // Оповещение

}

return result;

}

}

/// <summary>

/// Создание должности

/// </summary>

/// <param name="name">Название</param>

/// <param name="salary">Зарплата</param>

/// <param name="maxNumber">Максимум вакансий</param>

/// <param name="department">Отдел</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string CreatePosition(string name, decimal salary, int maxNumber, Department department)

{

string result = "Уже существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

// Проверка на существование должности

bool checkIsExist = db.Positions.Any(el => el.Name == name && el.Salary == salary);

if (!checkIsExist) // Если должность не существует

{

Position newPosition = new Position // Новая должность

{

Name = name, // Название

Salary = salary, // Зарплата

MaxNumber = maxNumber, // Число вакансий

DepartmentId = department.Id // первичный ключ отдела

};

db.Positions.Add(newPosition); // Добавление должности в таблицу должностей

db.SaveChanges(); // Сохранение изменений

result = "Сделано!"; // Оповещение

}

return result;

}

}

/// <summary>

/// Создание сотрудника

/// </summary>

/// <param name="name">Имя</param>

/// <param name="surName">Фамилия</param>

/// <param name="phone">Телефон</param>

/// <param name="position">Должность</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string CreateUser(string name, string surName, string phone, Position position)

{

string result = "Уже существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

// Проверка на существование сотрудника

bool checkIsExist = db.Users.Any(el => el.Name == name && el.SurName == surName && el.Position == position);

if (!checkIsExist) // Если сотрудник не существует

{

User newUser = new User // Новый сотрудник

{

Name = name, // Имя

SurName = surName, // Фамилия

Phone = phone, // Номер телефона

PositionId = position.Id // Первичный ключ занимаемой должности

};

db.Users.Add(newUser); // Добавление сотрудника в таблицу сотрудников

db.SaveChanges(); // Сохранение изменений

result = "Сделано!"; // Оповещение

}

return result;

}

}

/// <summary>

/// Удаление отдела

/// </summary>

/// <param name="department">Отдел</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string DeleteDepartment(Department department)

{

string result = "Такого отела не существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

db.Departments.Remove(department); // Удаление отдела

db.SaveChanges(); // Сохранение изменений

result = "Сделано! Отдел " + department.Name + " удален"; // Оповещение

}

return result;

}

/// <summary>

/// Удаление должности

/// </summary>

/// <param name="position">Должность</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string DeletePosition(Position position)

{

string result = "Такой должности не существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

db.Positions.Remove(position); // Удаление должности

db.SaveChanges(); // Сохранение изменений

result = "Сделано! Должность " + position.Name + " удалена"; // Оповещение

}

return result;

}

/// <summary>

/// Удаление сотрудника

/// </summary>

/// <param name="user">Сотрудник</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string DeleteUser(User user)

{

string result = "Такого сотрудника не существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

db.Users.Remove(user); // Удаление сотрудника

db.SaveChanges(); // Сохранение изменений

result = "Сделано! Сотрудник " + user.Name + " уволен"; // Оповещение

}

return result;

}

/// <summary>

/// Редактирование отдела

/// </summary>

/// <param name="oldDepartment">Старая информация</param>

/// <param name="newName">Новое название</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string EditDepartment(Department oldDepartment, string newName)

{

string result = "Такого отела не существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

// Проверка на существование отдела

Department department = db.Departments.FirstOrDefault(d => d.Id == oldDepartment.Id);

department.Name = newName; // Присваивание нового названия

db.SaveChanges(); // Сохранение изменений

result = "Сделано! Отдел " + department.Name + " изменен"; // Оповещение

}

return result;

}

/// <summary>

/// Редактирование должности

/// </summary>

/// <param name="oldPosition">Старая информация</param>

/// <param name="newName">Новое название</param>

/// <param name="newMaxNumber">Новый максимум вакансий</param>

/// <param name="newSalary">Новая зарплата</param>

/// <param name="newDepartment">Изменённая должность</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string EditPosition(Position oldPosition, string newName, int newMaxNumber, decimal newSalary, Department newDepartment)

{

string result = "Такой должности не существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

// Проверка на существование должности

Position position = db.Positions.FirstOrDefault(p => p.Id == oldPosition.Id);

position.Name = newName; // Присваивание нового названия

position.Salary = newSalary; // Присваивание новой зарплаты

position.MaxNumber = newMaxNumber; // Присваивание нового числа вакансий

position.DepartmentId = newDepartment.Id; // Присваивание нового первичного ключа

db.SaveChanges(); // Сохранение изменений

result = "Сделано! Должность " + position.Name + " изменена"; // Оповещение

}

return result;

}

/// <summary>

/// Редактирование сотрудника

/// </summary>

/// <param name="oldUser">Старая информация о сотруднике</param>

/// <param name="newName">Новое имя</param>

/// <param name="newSurName">Новая фамилия</param>

/// <param name="newPhone">Новый телефон</param>

/// <param name="newPosition">Новая должность</param>

/// <returns>Сообщение об успешном выполнении операции</returns>

public static string EditUser(User oldUser, string newName, string newSurName, string newPhone, Position newPosition)

{

string result = "Такого сотрудника не существует"; // Оповещение

using (ApplicationContext db = new ApplicationContext())

{

// Проверка на существование сотрудника

User user = db.Users.FirstOrDefault(p => p.Id == oldUser.Id);

if (user != null) // Если сотрудник существует

{

user.Name = newName; // Присваивание нового имени

user.SurName = newSurName; // Присваивание новой фамилии

user.Phone = newPhone; // Присваивание нового номера телефона

user.PositionId = newPosition.Id; // Присваивание нового первичного ключа

db.SaveChanges(); // Сохранение изменений

result = "Сделано! Сотрудник " + user.Name + " изменен"; // Оповещение

}

}

return result;

}

/// <summary>

/// Получение должности по первичному ключу

/// </summary>

/// <param name="id">Первичный ключ</param>

/// <returns>Должность</returns>

public static Position GetPositionById(int id)

{

using(ApplicationContext db = new ApplicationContext())

{

Position pos = db.Positions.FirstOrDefault(p => p.Id == id); // Поиск должности по первичному ключу

return pos;

}

}

/// <summary>

/// Получение отдела по первичному ключу

/// </summary>

/// <param name="id">Первичный ключ</param>

/// <returns>Отдел</returns>

public static Department GetDepartmentById(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

Department pos = db.Departments.FirstOrDefault(p => p.Id == id); // Поиск отдела по первичному ключу

return pos;

}

}

/// <summary>

/// Получение всех сотрудников по первичному ключу должности

/// </summary>

/// <param name="id">Первичный ключ</param>

/// <returns>Сотрудники</returns>

public static List<User> GetAllUsersByPositionId(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

// Выбрать сотрудников из списка сотрудников, которые относятся к конкретной должности

List<User> users = (from user in GetAllUsers() where user.PositionId == id select user).ToList();

return users;

}

}

/// <summary>

/// Получение всех должностей по первичному ключу отдела

/// </summary>

/// <param name="id">Первичный ключ</param>

/// <returns>Должности</returns>

public static List<Position> GetAllPositionsByDepartmentId(int id)

{

using (ApplicationContext db = new ApplicationContext())

{

// Выбрать должности из списка должностей, которые относятся к конкретному отделу

List<Position> positions = (from position in GetAllPositions() where position.DepartmentId == id select position).ToList();

return positions;

}

}

Department.cs

public int Id { get; set; } // Первичный ключ

public string Name { get; set; } // Название

public List<Position> Positions { get; set; } // Список должностей

[NotMapped] // Поле не участвует в базе данных

public List<Position> DepartmentPositions // Список должностей в отделе

{

get

{

// Возвращает метод поиска должностей по первичному ключу отдела

return DataWorker.GetAllPositionsByDepartmentId(Id);

}

}

Position.cs

public int Id { get; set; } // Первичный ключ

public string Name { get; set; } // Название

public decimal Salary { get; set; } // Зарплата

public int MaxNumber { get; set; } // Максимум вакансий

public List<User> Users { get; set; } // Список сотрудников

public int DepartmentId { get; set; } // Первичный ключ отдела

public virtual Department Department { get; set; } // Отдел

[NotMapped] // Поле не участвует в базе данных

public Department PositionDepartment

{

get

{

// Возвращает метод получения отдела по первичному ключу

return DataWorker.GetDepartmentById(DepartmentId);

}

}

[NotMapped] // Поле не участвует в базе данных

public List<User> PositionUsers // Список с должностями сотрудников

{

get

{

// Возвращает метод получения всех сотрудников по первичному ключу должности

return DataWorker.GetAllUsersByPositionId(Id);

}

}

RelayCommand.cs

private Action<object> \_execute;

private Func<object, bool> \_canExecute;

/// <summary>

/// Событие, вызывается при изменении условий, указывающее, может ли команда выполняться

/// </summary>

public event EventHandler CanExecuteChanged

{

add { CommandManager.RequerySuggested += value; }

remove { CommandManager.RequerySuggested -= value; }

}

/// <summary>

/// Конструктор класса

/// </summary>

/// <param name="execute">Выполнение команды</param>

/// <param name="canExecute">Событие</param>

public RelayCommand(Action<object> execute, Func<object, bool> canExecute = null)

{

\_execute = execute;

\_canExecute = canExecute;

}

/// <summary>

/// Определение, может ли команда выполняться

/// </summary>

public bool CanExecute(object parameter)

{

return \_canExecute == null || \_canExecute(parameter);

}

/// <summary>

/// Выполнение логики команды

/// </summary>

/// <param name="parameter"></param>

public void Execute(object parameter)

{

\_execute(parameter);

}

User.cs

public int Id { get; set; } // Первичный ключ

public string Name { get; set; } // Имя

public string SurName { get; set; } // Фамилия

public string Phone { get; set; } // Телефон

public int PositionId { get; set; } // Первичный ключ должности

public virtual Position Position { get; set; } // Должность

[NotMapped] // Поле не участвует в базе данных

public Position UserPosition

{

get

{

// Возвращает метод получения должности по первичному ключу

return DataWorker.GetPositionById(PositionId);

}

}

DataManageVM.cs

// Все отделы

private List<Department> allDepartments = DataWorker.GetAllDepartments(); // При первом подключении данных к View, поля будут наполняться данными

public List<Department> AllDepartments

{

get { return allDepartments; } // Возвращение всех отделов

set

{

allDepartments = value; // Отделам присваивается новое значение

NotifyPropertyChanged("AllDepartments"); // Уведомление системы об изменении данных

}

}

// Все должности

private List<Position> allPositions = DataWorker.GetAllPositions(); // При первом подключении данных к View, поля будут наполняться данными

public List<Position> AllPositions

{

get

{

return allPositions; // Возвращение всех должностей

}

private set

{

allPositions = value; // Должностям присваивается новое значение

NotifyPropertyChanged("AllPositions"); // Уведомление системы об изменении данных

}

}

// Все сотрудники

private List<User> allUsers = DataWorker.GetAllUsers(); // При первом подключении данных к View, поля будут наполняться данными

public List<User> AllUsers

{

get

{

return allUsers; // Возвращение всех сотрудников

}

private set

{

allUsers = value; // Сотрудникам присваивается новое значение

NotifyPropertyChanged("AllUsers"); // Уведомление системы об изменении данных

}

}

// Свойства для отдела

public static string DepartmentName { get; set; } // Название

// Свойства для должностей

public static string PositionName { get; set; } // Название

public static decimal PositionSalary { get; set; } // Зарплата

public static int PositionMaxNumber { get; set; } // Максимум вакансий

public static Department PositionDepartment { get; set; } // К какому отделу относится должность

// Свойства для сотрудника

public static string UserName { get; set; } // Имя

public static string UserSurName { get; set; } // Фамилия

public static string UserPhone { get; set; } // Номер телефона

public static Position UserPosition { get; set; } // Какую должность занимает

// Свойства для выделенных элементов

public TabItem SelectedTabItem { get; set; } // Выбранный элемент

public static User SelectedUser { get; set; } // Выбранный сотрудник

public static Position SelectedPosition { get; set; } // Выбранная должность

public static Department SelectedDepartment { get; set; } // Выбранный отдел

#region COMMANDS TO ADD

private RelayCommand addNewDepartment; // Добавить новый отдел

/// <summary>

/// Реализация команды добавления нового отдела

/// </summary>

public RelayCommand AddNewDepartment

{

get

{

return addNewDepartment ?? new RelayCommand(obj => // Проверка на пустоту поля, если пусто - возвращение

{

Window wnd = obj as Window; // Окно

string resultStr = "";

if (DepartmentName == null || DepartmentName.Replace(" ", "").Length == 0)

{

SetRedBlockControll(wnd, "NameBlock");

}

else

{

resultStr = DataWorker.CreateDepartment(DepartmentName);

UpdateAllDataView();

ShowMessageToUser(resultStr);

SetNullValuesToProperties();

wnd.Close();

}

}

);

}

}

private RelayCommand addNewPosition;

public RelayCommand AddNewPosition

{

get

{

return addNewPosition ?? new RelayCommand(obj =>

{

Window wnd = obj as Window;

string resultStr = "";

if(PositionName == null || PositionName.Replace(" ", "").Length == 0)

{

SetRedBlockControll(wnd, "NameBlock");

}

if (PositionSalary == 0)

{

SetRedBlockControll(wnd, "SalaryBlock");

}

if (PositionMaxNumber == 0)

{

SetRedBlockControll(wnd, "MaxNumberBlock");

}

if (PositionDepartment == null)

{

MessageBox.Show("Укажите отдел");

}

else

{

resultStr = DataWorker.CreatePosition(PositionName, PositionSalary, PositionMaxNumber, PositionDepartment);

UpdateAllDataView();

ShowMessageToUser(resultStr);

SetNullValuesToProperties();

wnd.Close();

}

}

);

}

}

private RelayCommand addNewUser;

public RelayCommand AddNewUser

{

get

{

return addNewUser ?? new RelayCommand(obj =>

{

Window wnd = obj as Window;

string resultStr = "";

if (UserName == null || UserName.Replace(" ", "").Length == 0)

{

SetRedBlockControll(wnd, "NameBlock");

}

if (UserSurName == null || UserSurName.Replace(" ", "").Length == 0)

{

SetRedBlockControll(wnd, "SurNameBlock");

}

if (UserPosition == null)

{

MessageBox.Show("Укажите позицию");

}

else

{

resultStr = DataWorker.CreateUser(UserName, UserSurName, UserPhone, UserPosition);

UpdateAllDataView();

ShowMessageToUser(resultStr);

SetNullValuesToProperties();

wnd.Close();

}

}

);

}

}

#endregion

private RelayCommand deleteItem; // Удаление элемента

/// <summary>

/// Команда для удаления элемента

/// </summary>

public RelayCommand DeleteItem

{

get

{

return deleteItem ?? new RelayCommand(obj =>

{

string resultStr = "Ничего не выбрано"; // Строка оповещения

if(SelectedTabItem.Name == "UsersTab" && SelectedUser != null) // Если сотрудник

{

resultStr = DataWorker.DeleteUser(SelectedUser); // Строке оповещения присваивается информация об удалённом сотруднике

UpdateAllDataView(); // Обновление информации во всех окнах

}

if (SelectedTabItem.Name == "PositionsTab" && SelectedPosition != null) // Если должность

{

resultStr = DataWorker.DeletePosition(SelectedPosition); // Строке оповещения присваивается информация об удалённой должности

UpdateAllDataView(); // Обновление информации во всех окнах

}

if (SelectedTabItem.Name == "DepartmentsTab" && SelectedDepartment != null) // Если отдел

{

resultStr = DataWorker.DeleteDepartment(SelectedDepartment); // Строке оповещения присваивается информация об удалённом отделе

UpdateAllDataView(); // Обновление информации во всех окнах

}

// Обновление

SetNullValuesToProperties(); // Обнулить все свойства

ShowMessageToUser(resultStr); // Показать окно с уведомлением

}

);

}

}

#region EDIT COMMANDS

private RelayCommand editUser; // Редактирования сотрудника

/// <summary>

/// Реализация команды редактирования сотрудника

/// </summary>

public RelayCommand EditUser

{

get

{

return editUser ?? new RelayCommand(obj =>

{

Window window = obj as Window; // Окно

string resultStr = "Не выбран сотрудник"; // Текст оповещения

string noPositionStr = "Не выбрана новая должность"; // Текст оповещения

if (SelectedUser != null) // Если выбранный сотрудник существует

{

if(UserPosition != null) // Если занятая сотрудником должность существует

{

// Информация о сотруднике изменяется

resultStr = DataWorker.EditUser(SelectedUser, UserName, UserSurName, UserPhone, UserPosition);

UpdateAllDataView(); // Обновление всей информации во всех окнах

SetNullValuesToProperties(); // Обнуление всех свойств

ShowMessageToUser(resultStr); // Показать окно оповещения

window.Close(); // Закрыть окно

}

else ShowMessageToUser(noPositionStr); // Иначе показать окно оповещения об ошибке - не выбрана новая должность

}

else ShowMessageToUser(resultStr); // Иначе показать окно оповещения об ошибке - не выбран сотрдуник

}

);

}

}

// Редактирование должности

private RelayCommand editPosition; // Редактирование должности

/// <summary>

/// Реализация команды редактирования должности

/// </summary>

public RelayCommand EditPosition

{

get

{

return editPosition ?? new RelayCommand(obj =>

{

Window window = obj as Window; // Окно

string resultStr = "Не выбрана позиция"; // Текст оповещения

string noDepartmentStr = "Не выбран новый отдел"; // Текст оповещения

if (SelectedPosition != null) // Если выбранная должность существует

{

if (PositionDepartment != null) // Если должность существует

{

// Информация о должности изменяется

resultStr = DataWorker.EditPosition(SelectedPosition, PositionName, PositionMaxNumber, PositionSalary, PositionDepartment);

UpdateAllDataView(); // Обновление всей информации во всех окнах

SetNullValuesToProperties(); // Обнуление всех свойств

ShowMessageToUser(resultStr); // Показать окно оповещения

window.Close(); // Закрыть окно

}

else ShowMessageToUser(noDepartmentStr); // Иначе показать окно оповещения об ошибке - не выбран новый отдел

}

else ShowMessageToUser(resultStr); // Иначе показать окно оповещения об ошибке - не выбрана должность

}

);

}

}

private RelayCommand editDepartment; // Редактирование отдела

/// <summary>

/// Реализация команды редактирования отдела

/// </summary>

public RelayCommand EditDepartment

{

get

{

return editDepartment ?? new RelayCommand(obj =>

{

Window window = obj as Window;

string resultStr = "Не выбран отдел"; // Текст оповещения

if (SelectedDepartment != null) // Если выбранный отдел существует

{

// Информация об отделе изменяется

resultStr = DataWorker.EditDepartment(SelectedDepartment, DepartmentName);

UpdateAllDataView(); // Обновление всей информации во всех окнах

SetNullValuesToProperties(); // Обнулить все свойства

ShowMessageToUser(resultStr); // Показать окно оповещения

window.Close(); // Закрыть окно

}

else ShowMessageToUser(resultStr); // Иначе показать окно оповещения

}

);

}

}

#endregion

#region COMMANDS TO OPEN WINDOWS

private RelayCommand openAddNewDepartmentWnd; // Открыть окно добавления отдела

/// <summary>

/// Реализация команды открытия окна добавления отдела

/// </summary>

public RelayCommand OpenAddNewDepartmentWnd

{

get

{

return openAddNewDepartmentWnd ?? new RelayCommand(obj => // Проверка на пустоту поля, если пусто - возвращение

{

OpenAddDepartmentWindowMethod(); // Открытие окна добавления отдела

}

);

}

}

private RelayCommand openAddNewPositionWnd; // Открыть окно добавления должности

/// <summary>

/// Реализация команды открытия окна добавления должности

/// </summary>

public RelayCommand OpenAddNewPositionWnd

{

get

{

return openAddNewPositionWnd ?? new RelayCommand(obj => // Проверка на пустоту поля, если пусто - возвращение

{

OpenAddPositionWindowMethod(); // Открытие окна добавления должности

}

);

}

}

private RelayCommand openAddNewUserWnd; // Открыть окно добавления сотрудника

/// <summary>

/// Реализация команды открытия окна добавления сотрудника

/// </summary>

public RelayCommand OpenAddNewUserWnd

{

get

{

return openAddNewUserWnd ?? new RelayCommand(obj => // Проверка на пустоту поля, если пусто - возвращение

{

OpenAddUserWindowMethod(); // Открытие окна добавления сотрудника

}

);

}

}

private RelayCommand openEditItemWnd; // Открыть контекстное меню для редактирования поля

/// <summary>

/// Реализация команды открытия контекстного меня для редактирования поля

/// </summary>

public RelayCommand OpenEditItemWnd

{

get

{

return openEditItemWnd ?? new RelayCommand(obj => // Проверка на пустоту поля, если пусто - возвращение

{

string resultStr = "Ничего не выбрано"; // Оповещение

if (SelectedTabItem.Name == "UsersTab" && SelectedUser != null) // Если выбран сотрудник

{

OpenEditUserWindowMethod(SelectedUser); // Открыть окно редакирования сотрудника для выбранного сотрудника

}

if (SelectedTabItem.Name == "PositionsTab" && SelectedPosition != null) // Если выбрана должность

{

OpenEditPositionWindowMethod(SelectedPosition); // Открыть окно редактирования должности для выбранной должности

}

if (SelectedTabItem.Name == "DepartmentsTab" && SelectedDepartment != null) // Если выбран отдел

{

OpenEditDepartmentWindowMethod(SelectedDepartment); // Открыть окно редактирования отдела для выбранного отдела

}

}

);

}

}

#endregion

#region METHODS TO OPEN WINDOW

/// <summary>

/// Открытие окна добавления отдела

/// </summary>

private void OpenAddDepartmentWindowMethod()

{

AddNewDepartmentWindow newDepartmentWindow = new AddNewDepartmentWindow(); // Окно добавления отдела

SetCenterPositionAndOpen(newDepartmentWindow); // Окно отроется посередине экрана

}

/// <summary>

/// Открытие окна добавления должности

/// </summary>

private void OpenAddPositionWindowMethod()

{

AddNewPositionWindow newPositionWindow = new AddNewPositionWindow(); // Окно добавления должности

SetCenterPositionAndOpen(newPositionWindow); // Окно отроется посередине экрана

}

/// <summary>

/// Открытие окна добавления сотрудника

/// </summary>

private void OpenAddUserWindowMethod()

{

AddNewUserWindow newUserWindow = new AddNewUserWindow(); // Окно добавления сотрудника

SetCenterPositionAndOpen(newUserWindow); // Окно отроется посередине экрана

}

/// <summary>

/// Открытие окна редактирвания отдела

/// </summary>

/// <param name="department">Отдел</param>

private void OpenEditDepartmentWindowMethod(Department department)

{

EditDepartmentWindow editDepartmentWindow = new EditDepartmentWindow(department); // Окно редактирования отдела

SetCenterPositionAndOpen(editDepartmentWindow); // Окно отроется посередине экрана

}

/// <summary>

/// Открытие окна редактирвания должности

/// </summary>

/// <param name="position">Должность</param>

private void OpenEditPositionWindowMethod(Position position)

{

EditPositionWindow editPositionWindow = new EditPositionWindow(position); // Окно редактирования должности

SetCenterPositionAndOpen(editPositionWindow); // Окно отроется посередине экрана

}

/// <summary>

/// Открытие окна редактирования сотрудника

/// </summary>

/// <param name="user">Сотрудник</param>

private void OpenEditUserWindowMethod(User user)

{

EditUserWindow editUserWindow = new EditUserWindow(user); // Окно редактирования сотрудника

SetCenterPositionAndOpen(editUserWindow); // Окно отроется посередине экрана

}

/// <summary>

/// Задание места открытия и открытия окон

/// </summary>

/// <param name="window">Окно</param>

private void SetCenterPositionAndOpen(Window window)

{

window.Owner = Application.Current.MainWindow; // Главное окно - главное

window.WindowStartupLocation = WindowStartupLocation.CenterOwner; // Позиция - центр главоного окна

window.ShowDialog(); // Окно не закрывается, пока не выполнится действие

}

#endregion

private void SetRedBlockControll(Window wnd, string blockName)

{

Control block = wnd.FindName(blockName) as Control;

block.BorderBrush = Brushes.Red;

}

#region UPDATE VIEWS

/// <summary>

/// Обнулить все свойства

/// </summary>

private void SetNullValuesToProperties()

{

// Для сотрудника

UserName = null; // Имя

UserSurName = null; // Фамилия

UserPhone = null; // Номер телефона

UserPosition = null; // Занимаемая должность

// Для должности

PositionName = null; // Название

PositionSalary = 0; // Зарплата

PositionMaxNumber = 0; // Число вакансий

PositionDepartment = null; // Отдел, к которому относится должность

// Для отдела

DepartmentName = null; // Название

}

/// <summary>

/// Обновление всей информации во всех окнах

/// </summary>

private void UpdateAllDataView()

{

UpdateAllDepartmentsView(); // Обновление информации об отделах

UpdateAllPositionsView(); // Обновление информации о должностях

UpdateAllUsersView(); // Обновление информации о сотрудниках

}

/// <summary>

/// Обновление информации об отделах

/// </summary>

private void UpdateAllDepartmentsView()

{

AllDepartments = DataWorker.GetAllDepartments(); // Получение списка отделов

MainWindow.AllDepartmentsView.ItemsSource = null; // Обнулить информацию в элементах

MainWindow.AllDepartmentsView.Items.Clear(); // Очистить данные элементов

MainWindow.AllDepartmentsView.ItemsSource = AllDepartments; // Присвоить элементам информацию из списка всех отделов

MainWindow.AllDepartmentsView.Items.Refresh(); // Обновить значения элементов

}

/// <summary>

/// Обновление информации о должностях

/// </summary>

private void UpdateAllPositionsView()

{

AllPositions = DataWorker.GetAllPositions(); // Получение списка должностей

MainWindow.AllPositionsView.ItemsSource = null; // Обнулить информацию в элементах

MainWindow.AllPositionsView.Items.Clear(); // Очистить данные элементов

MainWindow.AllPositionsView.ItemsSource = AllPositions; // Присвоить элементам информацию из списка всех должностей

MainWindow.AllPositionsView.Items.Refresh(); // Обновить значения элементов

}

/// <summary>

/// Обновление информации о сотрудниках

/// </summary>

private void UpdateAllUsersView()

{

AllUsers = DataWorker.GetAllUsers(); // Получение списка сотрудников

MainWindow.AllUsersView.ItemsSource = null; // Обнулить информацию в элементах

MainWindow.AllUsersView.Items.Clear(); // Очистить данные элементов

MainWindow.AllUsersView.ItemsSource = AllUsers; // Присвоить элементам информацию из списка всех сотрудников

MainWindow.AllUsersView.Items.Refresh(); // Обновить значения элементов

}

#endregion

/// <summary>

/// Показать окно оповещения

/// </summary>

/// <param name="message">Текст оповещения</param>

private void ShowMessageToUser(string message)

{

MessageView messageView = new MessageView(message); // Окно оповещений

SetCenterPositionAndOpen(messageView); // Окно откроется по центру экрана

}

public event PropertyChangedEventHandler PropertyChanged; // Событие изменения данных

/// <summary>

/// Уведомление об изменении данных

/// </summary>

/// <param name="propertyName">Имя свойства</param>

private void NotifyPropertyChanged(String propertyName)

{

if (PropertyChanged != null) // Если данные изменены

{

PropertyChanged(this, new PropertyChangedEventArgs(propertyName)); // Событие принимает новые данные

}

}