Федеральное государственное образовательное бюджетное учреждение высшего образования

«ФИНАНСОВЫЙ УНИВЕРСИТЕТ ПРИ ПРАВИТЕЛЬСТВЕ

РОССИЙСКОЙ ФЕДЕРАЦИИ»

(Финансовый университет)

**Колледж информатики и программирования**

ДОПУСКАЮ К ЗАЩИТЕ

заместитель директора колледжа

по учебно-производственной работе

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Л.В. Фокина

« » июня 2019г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему: Разработка автоматизированной информационной системы фирмы по ремонту квартир

Студент группы 4ПКС-115

Малкеров Геннадий Александрович «\_\_\_\_» июня 2019г.

Основная профессиональная образовательная программа по специальности

09.02.03 Программирование в компьютерных системах

Форма обучения очная

Руководитель ВКР \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Морозова М.В.

Председатель предметно-цикловой комиссии \_\_\_\_\_\_\_\_\_Пестов А.И.

Москва

2019

Содержание

[Введение 3](#_Toc11252934)

[Глава 1. Теоретическая часть 6](#_Toc11252935)

[1.1 Предпроектное исследование предметной области 6](#_Toc11252936)

[1.2 Характеристика инструментальных средств разработки 11](#_Toc11252937)

[1.3 Анализ некоторых технологий, используемых при реализации программного комплекса 16](#_Toc11252938)

[1.4 Анализ требований и определение спецификаций программного обеспечения 21](#_Toc11252939)

[1.5 Проектирование программного обеспечения 26](#_Toc11252940)

[Глава 2. Практическая часть 32](#_Toc11252941)

[2.1 Разработка пользовательских интерфейсов программного обеспечения 32](#_Toc11252944)

[2.2 Тестирование и отладка программного обеспечения 38](#_Toc11252945)

[2.3 Руководства по использованию программного обеспечения 41](#_Toc11252947)

[Заключение 76](#_Toc11252948)

[Список литературы 78](#_Toc11252949)

[Приложение А - Исходный код программы 80](#_Toc11252950)

[Приложение Б - Титульный лист презентации 219](#_Toc11252951)

# Введение

Работа с рутинными однообразными операциями не стимулирует работников выполнять свою работу качественно, а при условии возрастания количества данных операций стоит задуматься об автоматизации большинства процессов, происходящих в фирме. Для данных целей разрабатываются специализированные информационные системы.

Данные системы призваны облегчить работу сотрудников и их часто называют «автоматизированными информационными системами».

В основе каждой автоматизированной информационной системы лежит база данных, которая предназначена для хранения данных. С базой данных работает как определенный комплекс, так и одиночная программа. Суть данной программы в автоматизации получения, записи и редактирования данных.

Цель выпускной квалификационной работы заключается в разработке автоматизированной информационной системы фирмы по ремонту квартир. В ходе выполнения данной работы были приобретены и подтверждены навыки, приобретённые в период обучения по специальности 09.02.03 «Программирование в компьютерных системах» в колледже информатики и программирования.

Данная автоматизированная информационная система необходима для автоматизации некоторых процессов в работе небольшой фирмы, выполняющей ремонты квартир.

Система будет состоять из приложения под операционную систему Windows, Rest API сервиса, который будет работать на удаленном сервере, что позволит работать нескольким работникам одновременно, а также база данных, с которой будет держать связь Rest API сервис.

Основной задачей будет автоматизация и повышения эффективности работы персонала в рамках определенных задач.

Стоит отметить, данная автоматизация решит вопросы, связанные с количеством времени, которое тратится на создание заказа на ремонт, а также на некоторые другие технологические операции. После решения данных вопросов сократится время обслуживания клиентов и повысится качество данного обслуживания, а также повысится обратная отдача между клиентом и теми работниками, которые будут выполнять данный ремонт.

Процесс автоматизации всех процессов в настоящее время с развитием технологий приобрел большой интерес у некоторых директоров и в целом у всего руководящего персонала, который хочет повысить качество и скорость обслуживания клиентов.

Как уже говорилось ранее, цель данного проекта заключается в разработке автоматизированной информационной системы фирмы по ремонту квартир, в которой будет реализован набор всех требуемых для данной программы функций.

Для достижения данной цели необходимо решить следующие задачи:

* проектирование базы данных, как для сервера, так и для локального компьютера;
* создание универсальных методов для обработки ошибок и работы сервера;
* описание необходимых моделей для работы с клиентским приложением;
* создание контроллеров обработки данных поступаемых от клиентского приложения;
* разработка интерфейсов для работы клиентского приложения;
* разработка модуля для загрузки справочных данных на сервер;
* создание необходимых обработок для работы со сторонним сервером;
* создание необходимых шаблонов для вывода документов о смете и договоре по каждому заказу;
* написание документации о программе.

Для решения перечисленных выше задач в процессе работы над созданием программы необходимо пройти следующие стадии и этапы разработки:

* предпроектное исследование предметной области;
* выбор инструментальных средств разработки программы;
* проектирование программного обеспечения;
* разработка пользовательского интерфейса;
* тестирование и отладка программного обеспечения;
* разработка справочной системы;
* разработка руководств по использованию программы.

На каждом этапе данной работы будут применяться различные инструментальные средства разработки программного обеспечения:

* Visual Studio 2019 Enterprise;
* draw.io;
* система управления базами данных DB Browser for SQLite;
* сервер базы данных SQL Server Management Studio;
* система управления базами данных SQL Server Management Studio.

# Глава 1. Теоретическая часть

## Предпроектное исследование предметной области

Темой автоматизации фирм по ремонту квартир в настоящее время занимается очень маленькое количество фирм и компаний по разработке программных продуктов, при условии, что рынок ремонтов квартир в данное время очень быстро развивается.

Основной причиной развития рынка такого программного обеспечения можно считать рост рынка ремонтно-строительных работ, который стабилизировался в условиях современных экономических потрясений и состояния экономики России, связанных с ростом количества жилья сданных без отделки квартир относительно квартир, сданных с минимальным ремонтом, а также вследствие роста количества вторичного жилья.

Стоит отметить, что непрофессионалы отсеиваются с рынка быстрее и уступают место специалистам, которые делают качественный ремонт за адекватную цену, что подталкивает людей платить, а не делать ремонт самостоятельно [7].

В качестве существующих аналогов можно рассмотреть похожую по концепцию программу, распространяемую платно по средствам лицензии и разработанную компанией «ПростойСофт» [8], конфигурацию «Ремонт квартир» [9] которая устанавливается на базовую версию их программного обеспечения под название «Учет клиентов».

Данное программно обеспечение имеет следующие основные функции:

* учет клиентов;
* учет событий;
* формирование отчетов;
* хранение справочников;
* формирование документов по шаблонам;
* учет платежей.

Стоит отметить, что данная система является слишком громоздкой и не может работать на нескольких компьютерах одновременно, так как база располагается локально на компьютере, таким образом, использование данной программы невозможно при работе нескольких людей одновременно.

Также стоит отметить, что данное программно обеспечение поставляется за 8000 рублей за одно рабочее место, что несоизмеримо с функционалом при покупке более чем одной копии в компанию.

Также приведу фотографию пользовательского интерфейса, который представлен на рисунке 1.

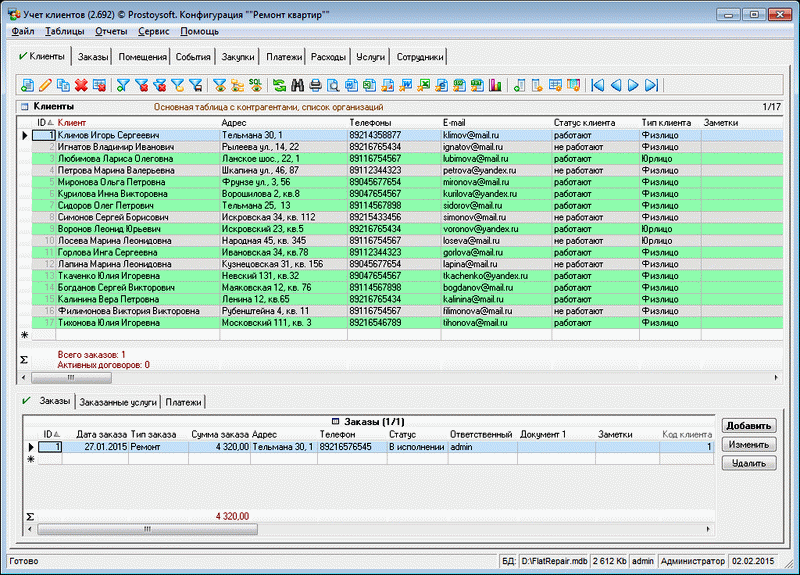


Рис. – Пример пользовательского интерфейса аналога

Также аналогами данного программного комплекса можно представить некоторые конфигурации 1С Предприятия, но они помимо того, что имеют изначально высокую стоимость, так еще и потребует некоторых доработок за отдельную сумму денег.

Данный проект разработан в целях увеличения скорости работы компании и представляет собой набор клиент-серверного программного обеспечения:

* базы данных для хранения всей информации;
* сервера, который выполняет прослойку между базой данных и прочими программами в данном комплексе;
* приложения для операционной системы Windows.

Для более глубокого понимания предметной области были изучены аналоги программы. Можно выделить следующие необходимые функции программ такого рода:

* подсчет стоимости заказа на ремонт;
* создание договора на основе шаблона, созданного заранее;
* импорт информации о материалах и услугах, предоставляемых данной организацией из файла программы Excel;
* возможность добавления, редактирования информации об услугах;
* возможность работы с данными о сотрудниках (прием на работу, увольнение и кадровые перемещения) с документированием в формате Word;
* возможность раздельного доступа к программе путем назначения прав доступа для сотрудников, а также возможность их редактирования;
* учет данных о клиентах и сотрудниках, а также редактирование данных об этом факте;
* учет оказанных услуг и учет платежей, а также система отчетов о проделанных работах и проданных услугах;
* фиксирование замеров помещений и вычисление площади некоторых поверхностей.

Среди основных функций разрабатываемого программного обеспечения можно отметить следующие:

* подсчет стоимости заказа на ремонт;
* создание договора на основе шаблона, созданного заранее;
* импорт информации о материалах и услугах, предоставляемых данной организацией из файла программы Excel;
* возможность добавления, редактирования информации об услугах;
* возможность работы с данными о сотрудниках (прием на работу, увольнение и кадровые перемещения) с документированием в формате Word;
* возможность раздельного доступа к программе путем назначения прав доступа для сотрудников, а также возможность их редактирования;
* учет данных о клиентах и сотрудниках, а также редактирование данных об этом факте;
* учет оказанных услуг и учет платежей, а также система отчетов о проделанных работах и проданных услугах;
* фиксирование замеров помещений и вычисление площади некоторых поверхностей.

В качестве входной информации можно отметить следующие основные данные, необходимые для работы приложения:

* данные о сотрудниках и клиентах;
* данные о заказе, в том числе размеры помещений;
* данные о материалах и услугах, предоставляемых данной компанией;
* данные о должностях в данной компании;
* данные о выполненных этапах и услугах;
* данные о платежах и выплатах.

Справочную информацию, которую необходимо загрузить на сервер, предполагается загружать в виде специально подготовленного файла. Данный Excel файл содержит информацию обо всех типах справочной информации, которую предполагается использовать.

Пример данного файла представлен на рисунке 2. Также стоит отметить, что данный файл содержит в себе скрипт, который создает уникальный идентификатор определенного элемента, что позволяет помимо добавления производить и обновление данных.

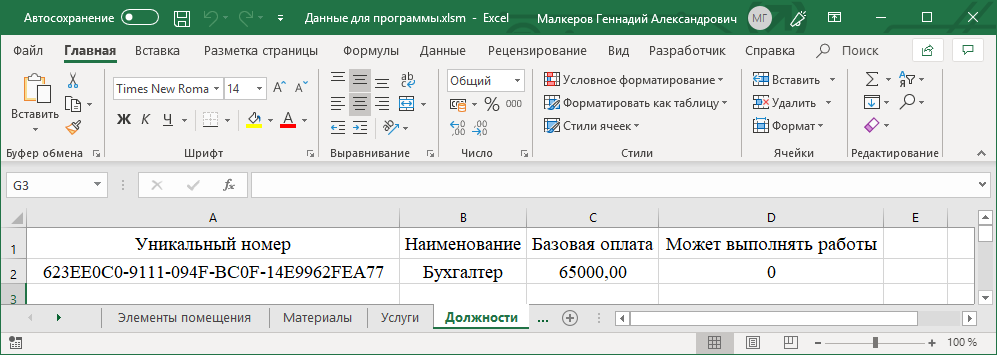


Рис. – Пример отображения файла Excel

В качестве выходной информации в данной программе служат отчеты об операциях, выполняемых в данной АИС, а также смета, договор и справочная информация по каждому заказу.

Необходимыми программными средствами для работы будут являться:

* операционная система Windows 10;
* приложение Microsoft Word и Microsoft Excel;
* наличие .Net Framework 4.7 версии и выше.

Необходимыми аппаратными средствами являются:

* процессор с тактовой частотой 2,7 ГГц или выше. Рекомендуется использовать двухъядерный процессор;
* 1 ГБ ОЗУ (3 ГБ при выполнении в виртуальной машине);
* 10 ГБ доступного пространства на жестком диске;
* видеоадаптер с поддержкой DirectX 9 (разрешение 1600 x 900 или больше);
* монитор с разрешением не меньше, чем (1600 x 900);
* наличие постоянного соединения с сетью интеренет;
* клавиатура и мышь.

Стоит отметить, что программу следует запускать от имени админстратора при установке данной программы в корень диска «C», в противном случае программа перестанет работать.

Необходимые функциональные требования:

* программа должна осуществлять добавление, редактирование, удаление, просмотр информации, а также ограничить доступ к данным посторонним лицам;
* работать с заданным алгоритмом;
* поддерживать диалоговый режим в рамках предоставляемых пользователю возможностей;
* производить бесперебойную работу по преобразованию информации.

Данная АИС должна быть реализована с использованием языка программирования C# и принципов объектно-ориентированного программирования.

В программном продукте должна быть предусмотрена защита от действий пользователя, которые могли бы привести к выдаче ошибки и экстренного закрытия программного продукта:

* защита от SQL- инъекций;
* защита от некорректного ввода значений в программе;

Требования к интерфейсу клиентского приложения.

Интерфейс разработываемой АИС должен быть понятным и удобным для пользователя, так как данный программный продукт пользователь будет использовать большую часть рабочего времени.

## Характеристика инструментальных средств разработки

### Характеристика среды программирования

В качестве средств рисования диаграмм было использовано бесплатное программно решение draw.io [10]. Данное программное решение представляет полный перечень возможностей по разработке диаграмм.

Выбранное средство возможно использовать как в браузере, так и локально на компьютере пользователя. Среди плюсов также можно отметить удобный интерфейс, пример которого представлен на рисунке 3.

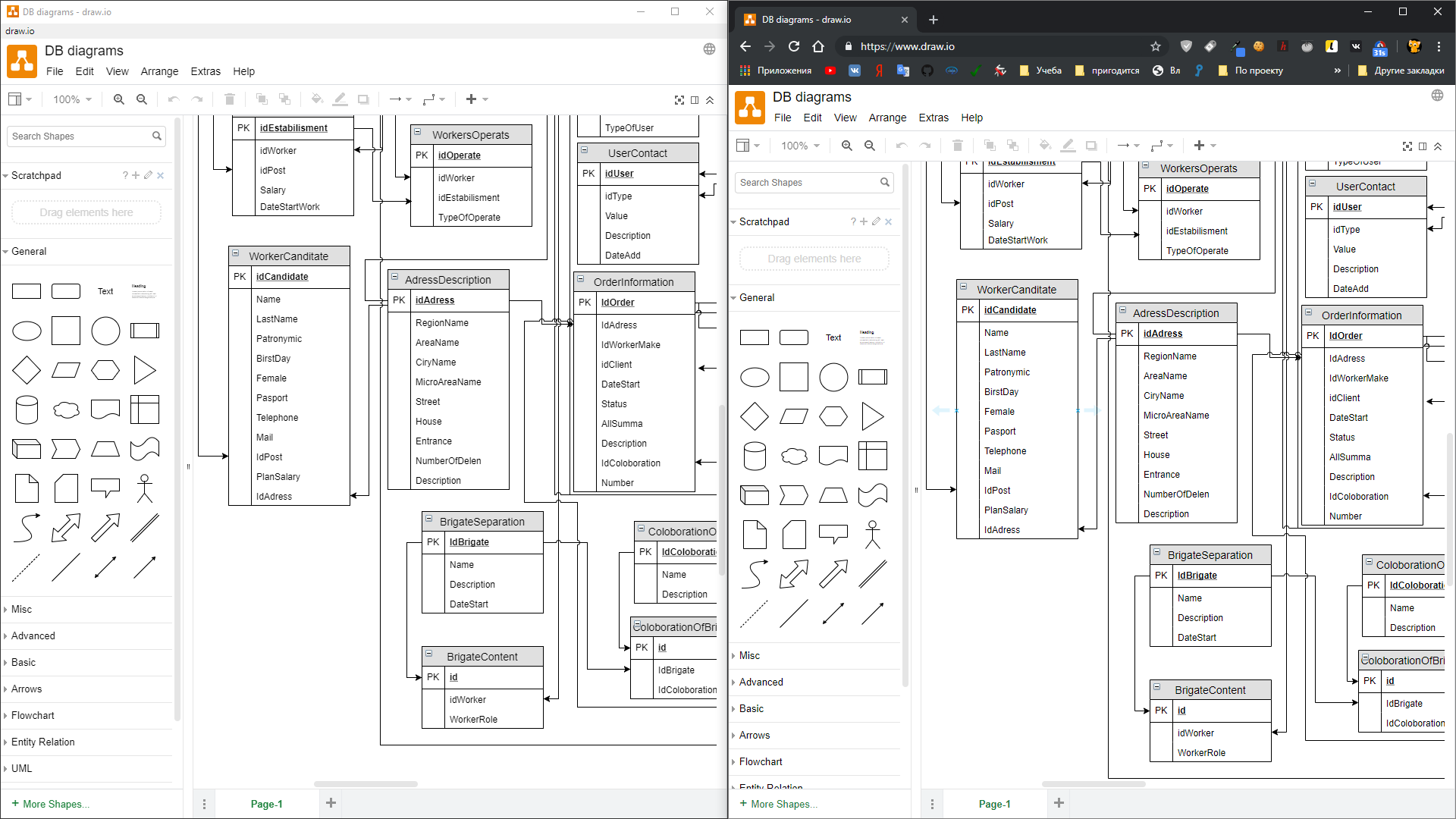


Рис. – Интерфейс Draw.io

В качестве системы управления базами данных для использования на локальном компьютере был выбран «DB Browser for SQLite» [11]. Данное приложение было выбрано за счет удобного пользовательского интерфейса, пример которого представлен на рисунке 4, а также по причине простоты самого приложения и высокой скорости работы.

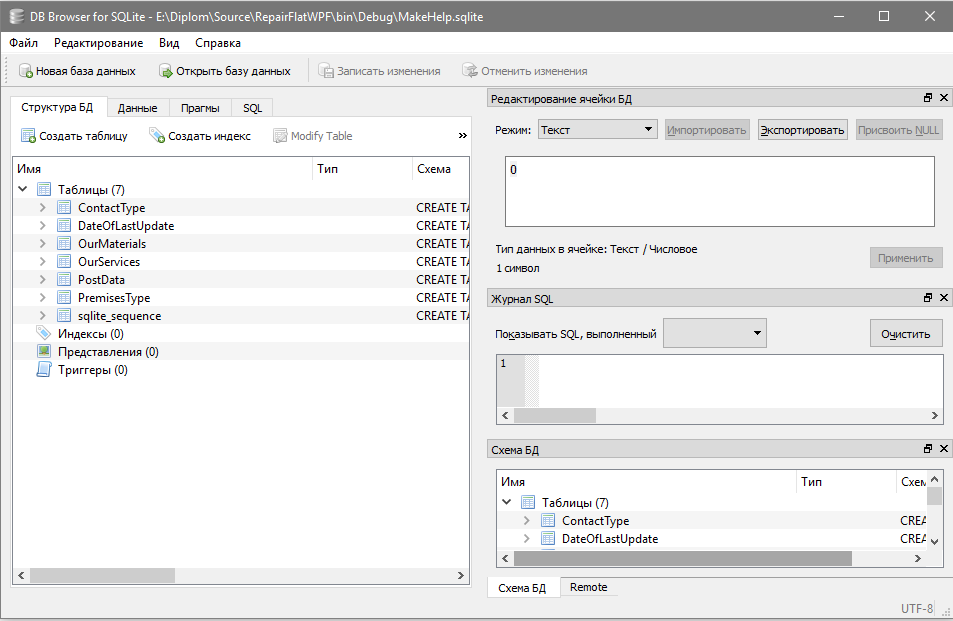


Рис. 4 – Интерфейс DB Browser for SQLite

Кроме локальной базы данных, также встала необходимость использования серверной системы управления базами данных. В качестве данной системы управления базами данных был выбран «Microsoft SQL Server» [12]. Данная система управления базами данных была выбрана по причине простоты развертки данных баз данных, как на Linux, так и на Windows серверах, что повышает совместимость и уменьшает стоимость развертки данной системы управления базами данных на сервере.

В качестве инструмента взаимодействия c серверной системой управления базами данных был выбран «SQL Server Management Studio»[13]. Данное приложение представляет полный доступ к возможностям Microsoft SQL Server» через удобный пользовательский интерфейс, пример которого представлен на рисунке 5.

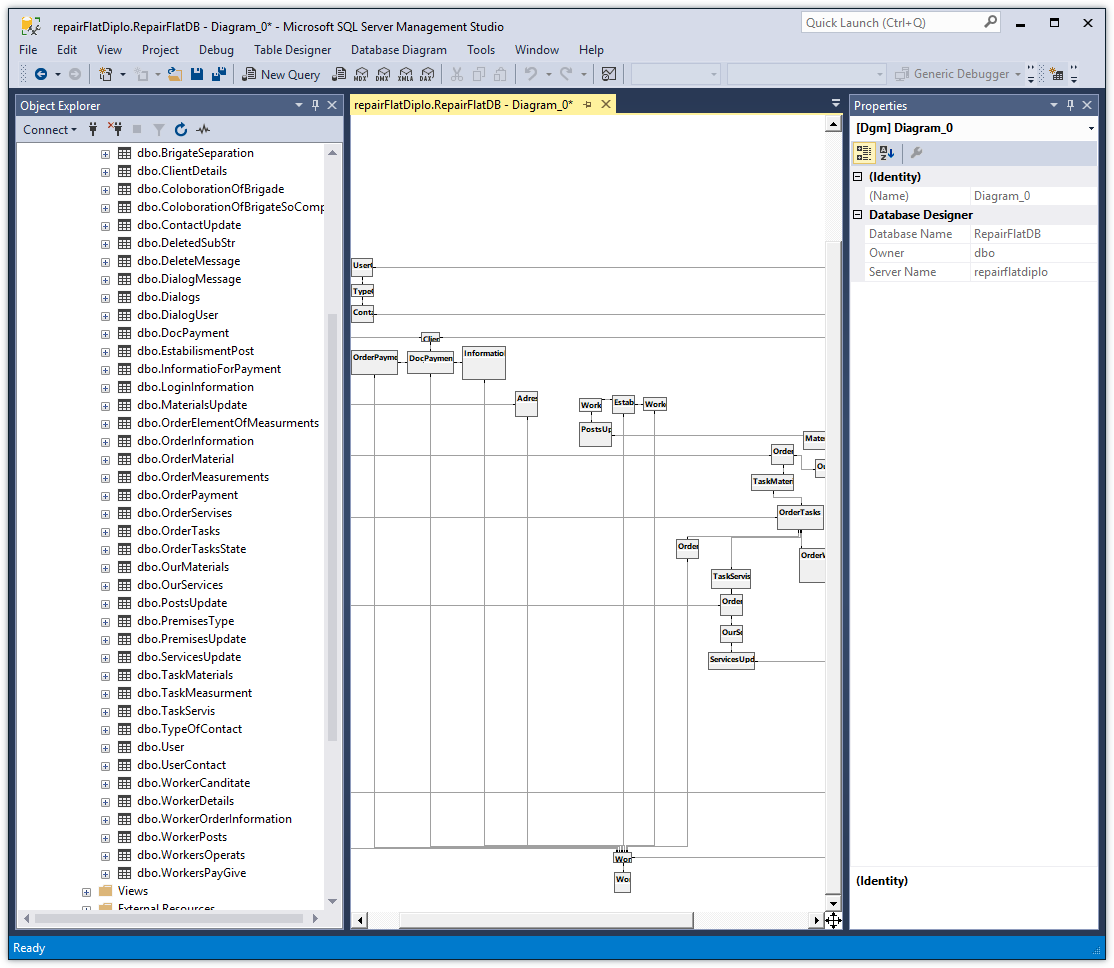


Рис. 5 – Интерфейс SQL Server Management Studio

В качестве среды разработки была выбрана Visual Studio 19 с, удобным для пользователя интерфейсом и наличием отладчика кода.

Программирование выполнялось на языке программирования C#, тип проекта:

* для Windows приложения — проект графического окна, основанного на технологии WPF;
* для серверной части данного комплекса — ASP.NET Web Application (.NET Fraemwork);
* для создания базы данных — проект базы данных.

Visual Studio 2019 (VS) представляет собой интегрированную среду разработки (Integrated Development Environment, IDE).

Пример пользовательского интерфейса VS представлен на рисунках 6-7. Стоит отметить, что в новой версии VS был переработан дизайн некоторых объектов, пример интерфейса, который и приведен ниже.

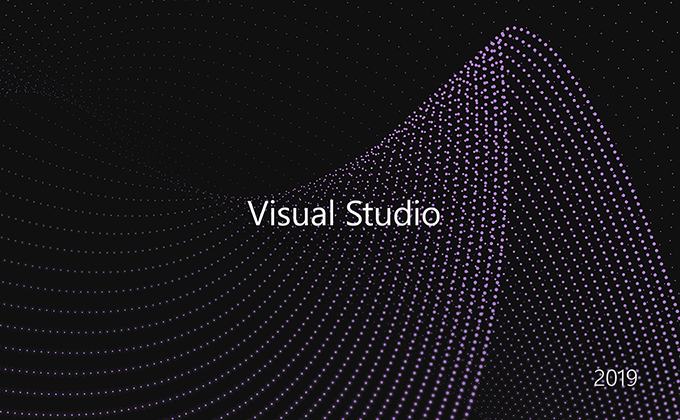


Рис. 6 – Обновленный интерфейс загрузки Visual Studio 2019

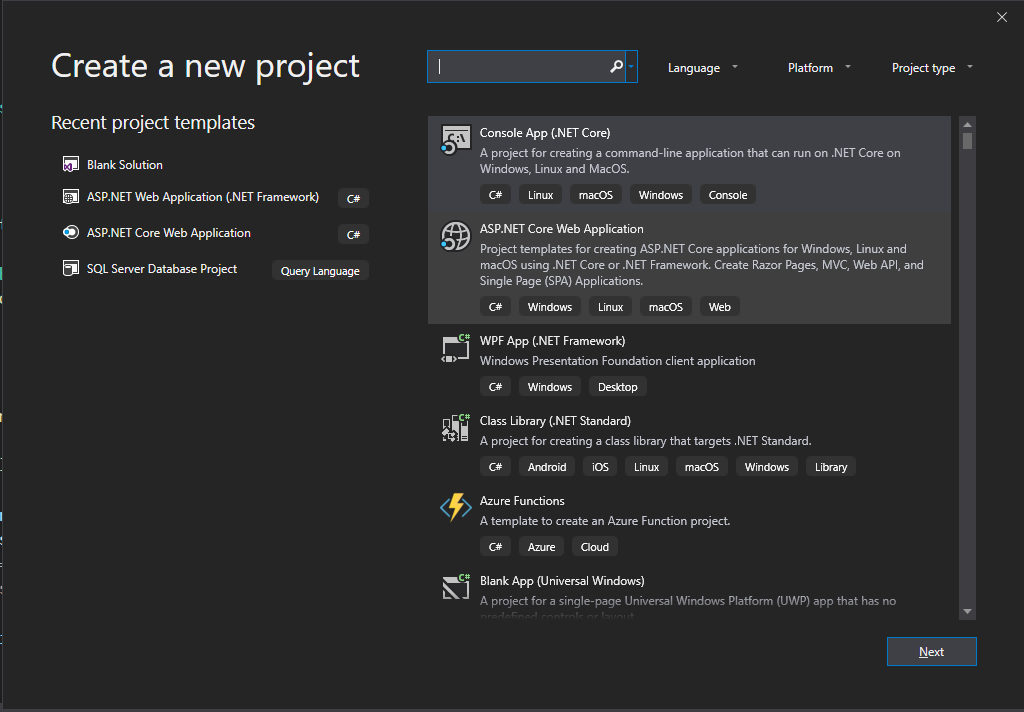


Рис. 7 – Обновленный интерфейс выбора типа приложения Visual Studio 2019

С помощью VS многие из этих рутинных и трудоемких задач, связанных с повседневной работой программиста, можно упростить и облегчить.

В состав VS входит целый набор типовых проектов, из которых каждый разработчик может подобрать именно то, что ему в данный момент требуется. Каждый раз, когда вы создаете новый проект, VS автоматически создаст "скелет" будущего приложения, причем этот код можно немедленно скомпилировать и запустить на исполнение. В составе каждого типового проекта имеются элементы, которые по желанию можно добавлять в проект.

Проект в Visual Studio содержит некоторый процент автоматически сгенерированного кода, который представляет собой основу будущей программы. VS предлагает множество готовых к использованию элементов управления, включая и код, необходимый для их создания. Это экономит время разработчиков, избавляя их от необходимости каждый раз заново создавать типовой программный код для решения часто встречающихся задач [14].

Для отладки серверной части программной части разрабатываемого программного обеспечения был выбран соответствующий инструмент «Postman» [15]. Интерфейс данной программы представлен на рисунке 8.

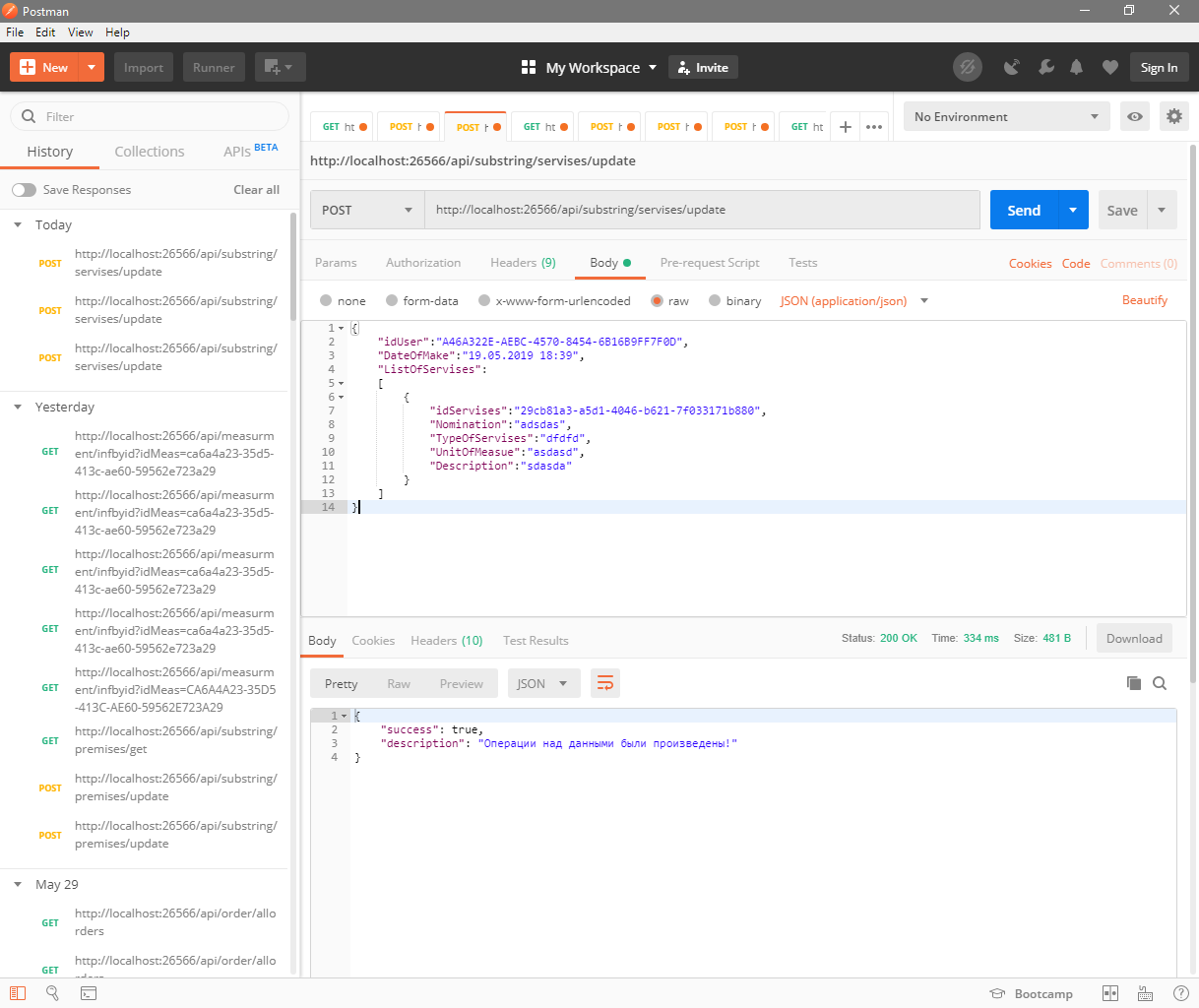


Рис. – Интерфейс программы Postman

### Характеристика языка программирования

Дипломная работа выполнена с использованием объектно-ориентированного языка программирования C# и некоторых сторонних библиотеках, предназначенных для данного языка программирования.

C# — язык программирования, сочетающий объектно—ориентированные и контекстно-ориентированные концепции. Разработан в 1998—2001 годах группой инженеров под руководством Андерсa Хейлсбергa в компании Microsoft как основной язык разработки приложений для платформы Microsoft .NET. Компилятор C# входит в стандартную установку самой .NET.

C# относится к семье языков с C-подобным синтаксисом, из них его синтаксис наиболее близок к C++ и Java. Язык имеет строгую статическую типизацию, поддерживает полиморфизм, перегрузку операторов, указатели на функции-члены классов, атрибуты, события, свойства, исключения, комментарии в формате XML [16].

## Анализ некоторых технологий, используемых при реализации программного комплекса

### Технология Entity Framework

Разработка автоматизированной информационной системы подразумевает использование базы данных как средство хранения обработанной информации.

Для работы с базами данных можно использовать язык запросов SQL. Данный язык программирования разработан для управления реляционными базами данных. Язык был создан в 1970x годах и назывался «SEQUEL» и был предназначен для управления системой баз данных System R. Через некоторое время данный язык был переименован «SQL» и в 1979 году был опубликован в качестве коммерческого продукта Oracle V2.

В 1986 году был создан первый официальный стандарт языка, после чего были созданы еще несколько стандартов языка. Несмотря на существование стандартов, большинство средств управления СУБД используют встроенный язык запросов, который иногда очень кардинально отличается от стандартов, что объясняется большим объемом и сложностью описанного стандарта, а также нехваткой в нем некоторых спецификаций в важных областях [17].

При реализации программного комплекса появится необходимость работы с запросами. Данную работу можно проводить многими способами, но помимо использования языка T-SQL, который необходимо было бы использовать по причине использования в качестве системы управления базами данных «Microsoft SQL Server». Использование данного языка запросов может привести к ошибкам после переименования некоторых полей.

В целях обеспечения безопасности была выбрана соответствующая технология, а именно «ADO.NET Entity Framework».

Данная технология предоставляет возможность работы с базами данных как с некоторыми сущностями. Сущностями в данном случае называют некий набор данных, которые ассоциативны с определенным объектом, но считать таблицы базы данных набором сущностей не совсем корректно, так как работа выполняется с объектами и их наборами.

При этом сущности могут быть связаны ассоциативной связью один-ко-многим, один-ко-одному и многие-ко-многим, подобно тому, как в реальной базе данных происходит связь через внешние ключи.

Отличительной чертой Entity Framework является использование запросов LINQ для выборки данных из БД. С помощью LINQ мы можем не только извлекать определенные строки, хранящие объекты, из базы данных, но и получать объекты, связанные различными ассоциативными связями.

Еще одним ключевым понятием Entity Data Model. Эта модель сопоставляет классы сущностей с реальными таблицами в БД.

Entity Data Model состоит из трех уровней:

* концептуальном на котором происходит определение классов сущностей, используемых в приложении;
* уровень хранилища, который определяет таблицы, столбцы, отношения между таблицами и типами данных, которыми сопоставляется используемая база данных;
* уровень сопоставления (маппинга) который служит посредником между свойствами класса, сущности и столбцами таблицы.

Таким образом, мы можем через классы, определенные в приложении, взаимодействовать с таблицами из базы данных [18].

Для работы с Entity Framework необходимо изначально скачать NuGet пакет Entity Framework. Для скачивания NuGet пакетов необходимо правой кнопкой нажать на название своего проекта на платформе .Net, после чего выбрать пункт, выделенный на рисунке 9. После чего перейти на первую вкладку «Browse» и в строку поиска ввести Entity Framework, в результатах выбрать пункт «EntityFramework», нажать кнопку «Install». Пример выбора данного пакета представлен на рисунке 10.

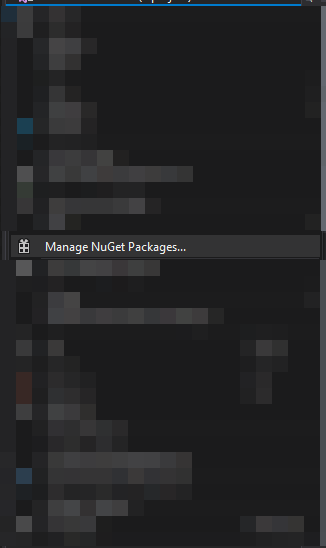


Рис. – Открытие менеджера пакетов NuGet

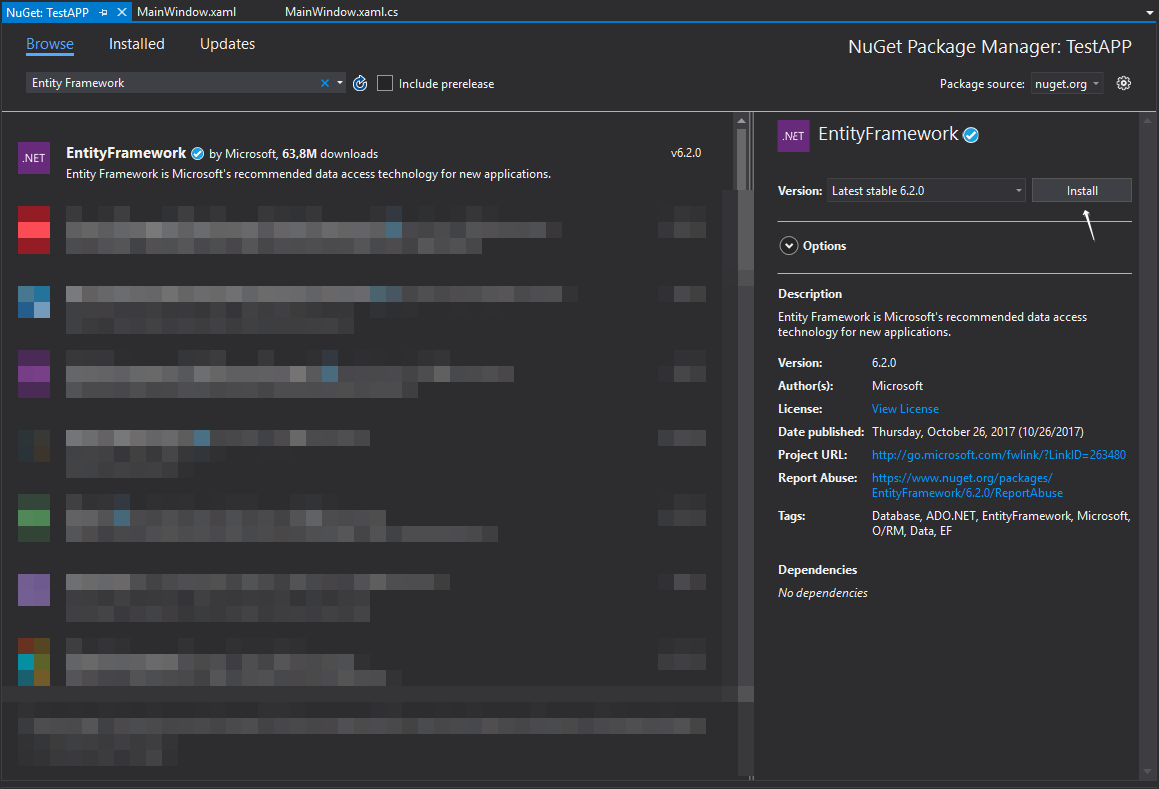


Рис. – Выбор необходимого пакета

После установки пакета необходимо в проект добавить объект «ADO.NET Entity Data Model», который добавляется путем нажатия правой кнопки на название проекта на платформе .Net, после чего выбрать пункт «Добавить» затем «Новый элемент», затем перейти в раздел «Данные» и выбрать пункт «ADO.NET Entity Data Model», ввести название и нажать кнопку «Добавить». Пример добавления представлен на рисунке 11.

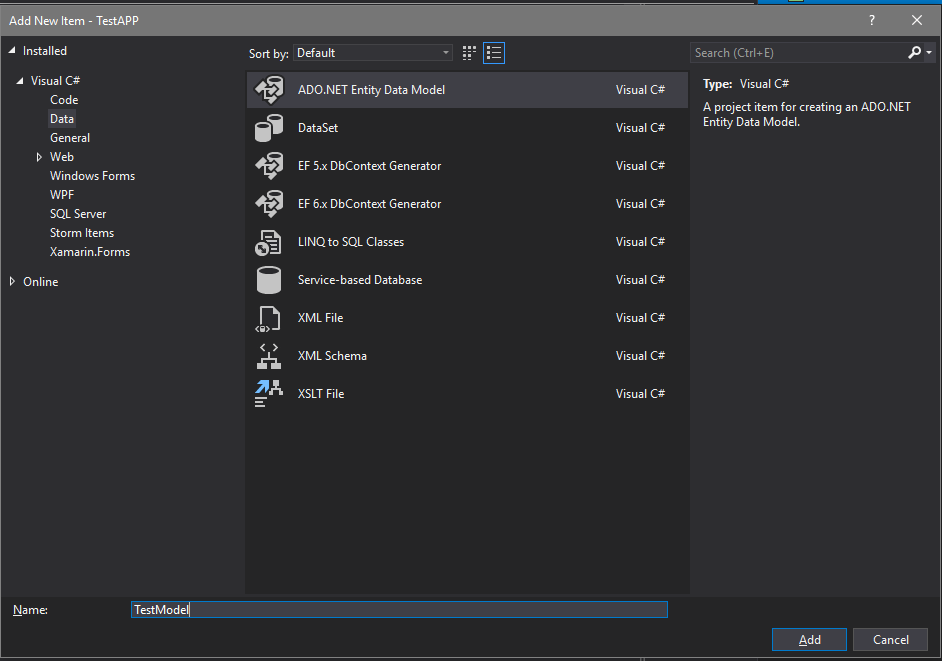


Рис. 11 – Добавление «ADO.NET Entity Data Model» в проект

После нажатия кнопки «Добавить» необходимо нажать «Далее», после чего указать строку подключения к базе данных путем нажатия кнопки «New Connection» на форме, представленной на рисунке 12. После указания строки подключения необходимо выбрать пункт «Yes, include sensitive data in the connection string», после чего нажать «Далее» и в открывшемся окне выбрать необходимые объекты для копирования в базу данных (пример выбора данных представлен на рисунке 13) после чего нажать «Выполнить».

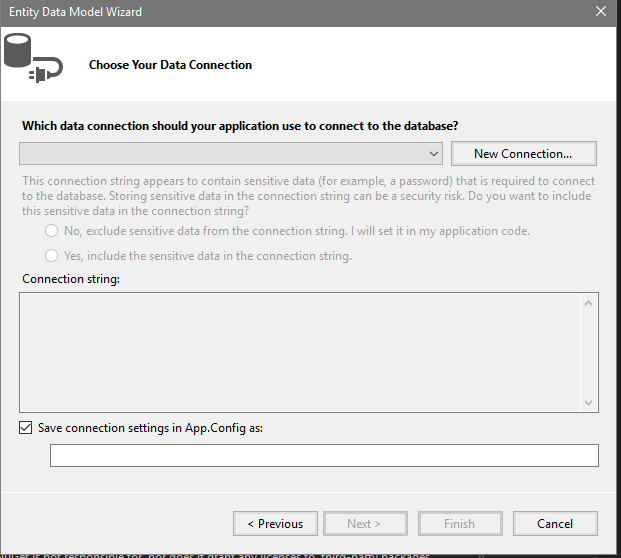


Рис. 12 – Окно для указания строки подключения

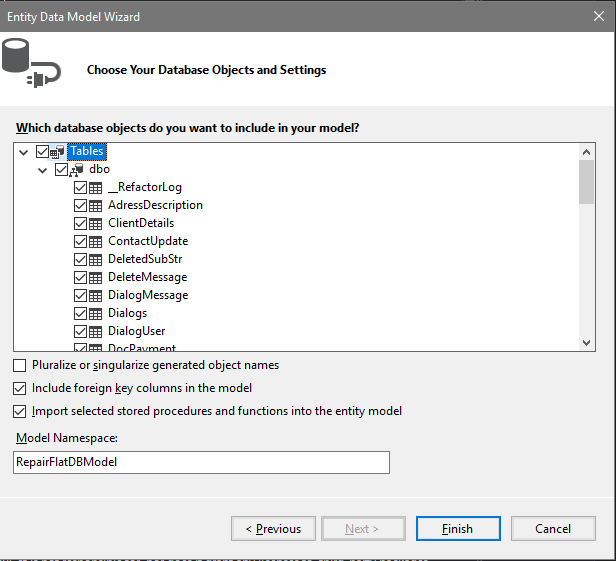


Рис. 13 – Окно выбора необходимых компонентов базы данных

Через некоторое время данные будут скопированы и представлены в решении в соответствующем пункте в качестве примера, представленного на рисунке 14, и указана модель, которая была скопирована из базы данных.

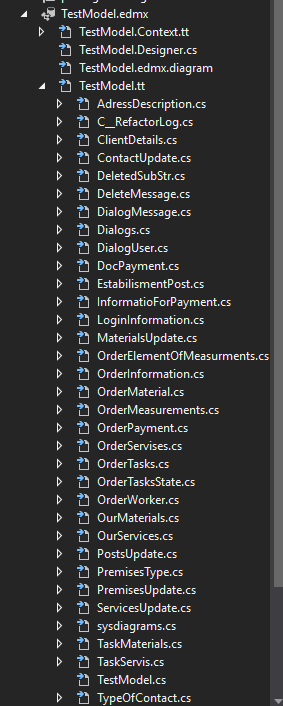


Рис. 14 – Пример отображения классов, отображающих «сущности»

## Анализ требований и определение спецификаций программного обеспечения

Диаграмма вариантов использования клиентского приложения представлена на рисунке 15. Данная диаграмма описывает работу приложения по соответствующим сценариям, которые прописаны в зависимости от роли, которая выданная определенному логину и паролю.

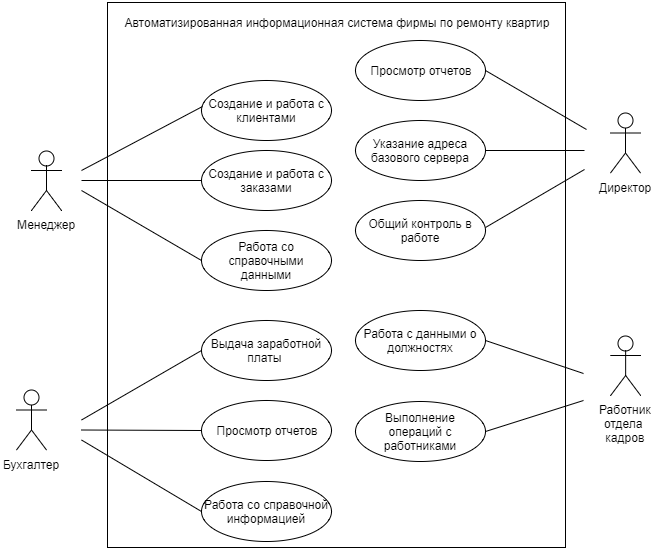


Рис. - Диаграмма вариантов использования системы

Контекстная диаграмма потоков данных, которая представлена на рисунке 16. Описывает взаимодействие между пользователями и сервером обработки информации посредством клиентского приложения.

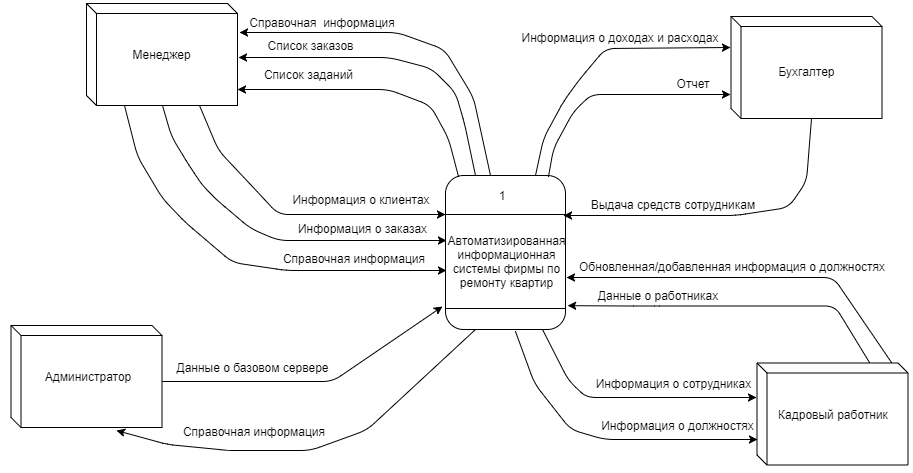


Рис. – Контекстная диаграмма потоков данных

На данной диаграмме отображены данные, которые предоставляются пользователям:

* о работниках;
* о клиентах;
* о деталях заказа;
* о справочной информации.

Программа в качестве ответа демонстрирует информацию, о том, что данные были добавлены, а также демонстрирует общий список данных, которые необходимы в данном контексте.

Также ниже представлена детализированная диаграмма потоков данных в программном комплексе, представленная на рисунке 17. Данная диаграмма детализирует контекстную диаграмму, приведенную выше.

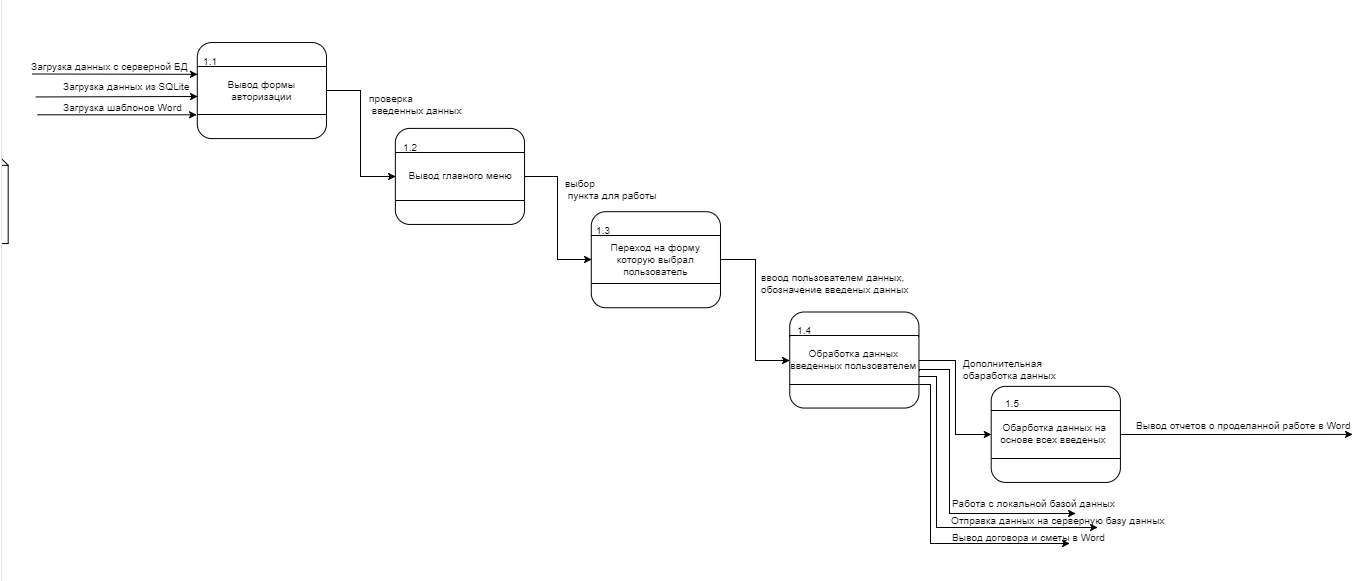


Рис. – Детализированная диаграмма потоков данных

Для разрабатываемого программного комплекса была составлена контекстная функциональная диаграмма, описывающая схему взаимодействия серверного модуля с прочими модулями, она представлена на рисунке 18. На данной диаграмме показано, что в качестве основного модуля можно считать модуль сервера, который обрабатывает все данные. Сервер в свою очередь выполняет полную связь с базой данных.

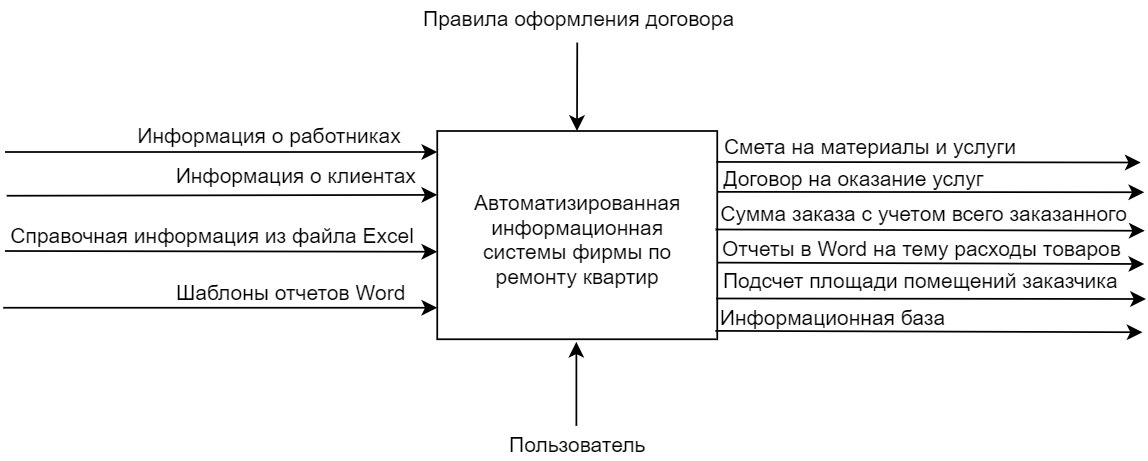


Рис. – Контекстная функциональная диаграмма

Далее была составлена детализированная функциональная диаграмма, представленная на рисунке 19. Данная диаграмма описывает основные функции со стороны серверной части при стандартной работе пользователя с системой, аналогичной детализированной диаграмме потоков данных.

На вход при стандартной работе подается строка подключения, которая берется из настроек и может быть изменена, а также данные от пользователя, либо вызов соответствующей операции по ссылке.

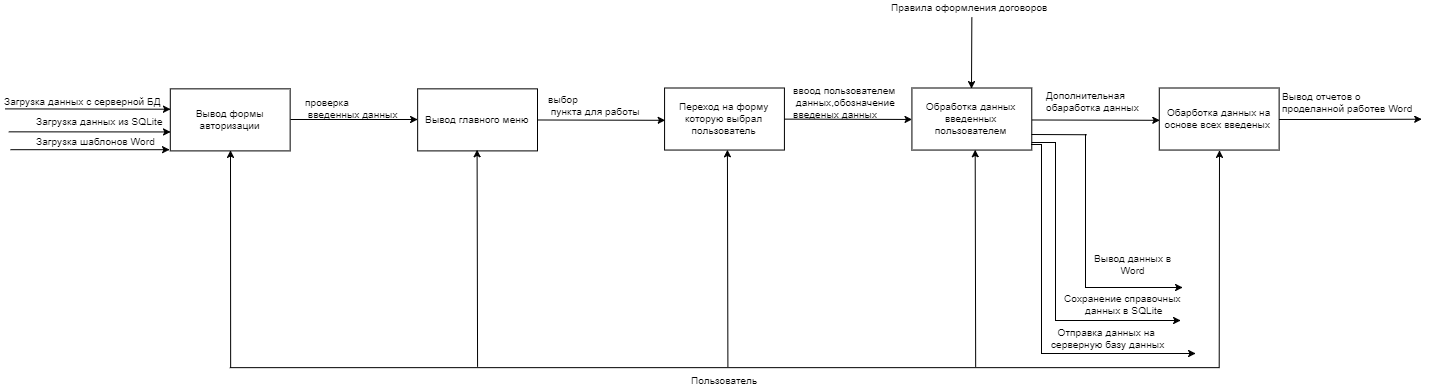


Рис. – Детализированная функциональная диаграмма

## Проектирование программного обеспечения

### Проектирование клиент серверной архитектуры

Разработка программного обеспечения выполнено посредством клиент-серверной технологии, которая предоставляет возможность работы нескольким клиентским программам одновременно на разных машинах с одними исходными данными.

Клиент серверная схема работы программного продукта представлена на рисунке 20.

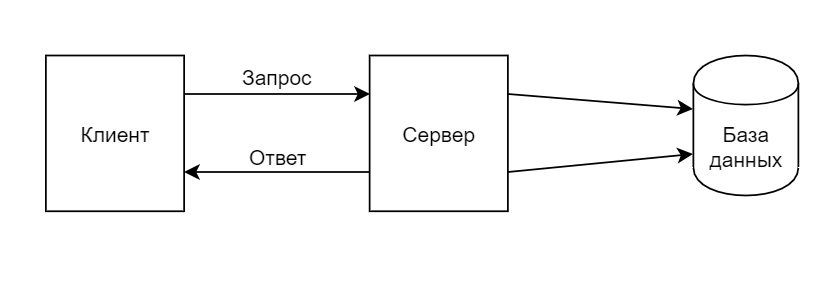


Рис. – Схема работы программного комлекса

Для взаимодействия клиента и сервера была выбрана технология REST API, которая предполагает отсылку запроса на клиент, когда это необходимо клиенту.

### Структурная схема

При разработке программного продукта была создана структурная схема клиентского приложения, которая представлена на рисунке 21. Данная схема демонстрирует возможности, предоставляемые при работе клиентского приложения, посредствам использования встроенного меню.

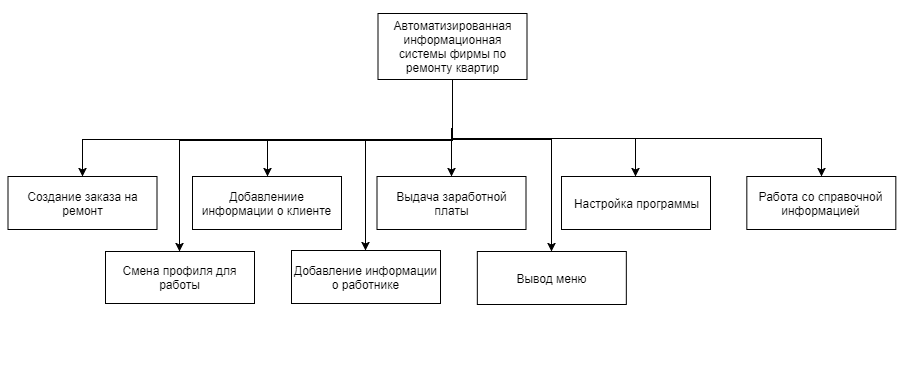


Рис. – Структурная схема клиентского приложения

### Проектирование баз данных

В ходе разработки проекта появилась необходимость в использовании базы данных для хранения информации для работы программы, для чего были спроектированы и созданы 2 базы данных:

* основная база данных, используется на сервере «Microsoft SQL Server» в качестве основной базы данных;
* в качестве дополнительной базы данных на компьютере клиента.

Рассмотрим обе базы, представленные в качестве диаграмм сущность-связь.

Начнем с локальной базы данных, схема которой представлена на рисунке 22.

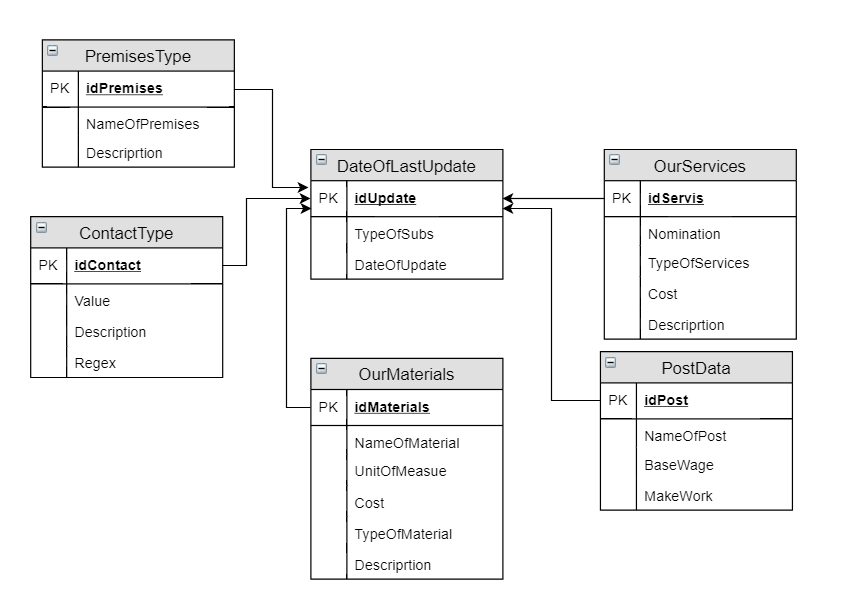


Рис. – Схема локальной базы данных

Данная база данных разработана на языке SQLite, для отображения данных на тестировании используется редактор «DB Browser for SQLite».

На данной схеме представлены 6 таблиц.

Стоит отметить, что данные таблицы создаются на основе JSON файла, который прикреплён в ресурсах клиентской части продукта и при желании может быть изменен.

Как уже говорилось выше, база данных на момент написания данной документации состоит из 6 таблиц:

* ContactType – Тип контактов – содержит данные о типах контактах в качестве справочной информации;
* DateOfLastUpdate – Дата последнего обновления – содержит данные о последнем обновлении данных с сервера и типе объектов обновления;
* OurMaterials – Наши материалы – содержит данные о материалах, которые используются в качестве справочных данных в программе;
* OurServices – Наши услуги – содержит данные об услугах, которые используются в качестве справочных данных в программе;
* PostData – Данные о должностях – содержит информацию о должностях и используется, когда происходит работа кадрового отдела;
* PremisesType – Данные о типах помещений – содержит информацию о типах помещений и используется в качестве справочных данных.

Для проектирования основной базы данных был выбран способ проектирования с использованием специализированного проекта в среде разработки Visual Studio 2019. Данные проект называется проект базы данных, с помощью данного проекта возможно спроектировать всю базу данных после чего загрузить с использованием специальных средств на сервер «Microsoft SQL Server». Схема сущность-связь разработанной базы данных представлена на рисунке 23.

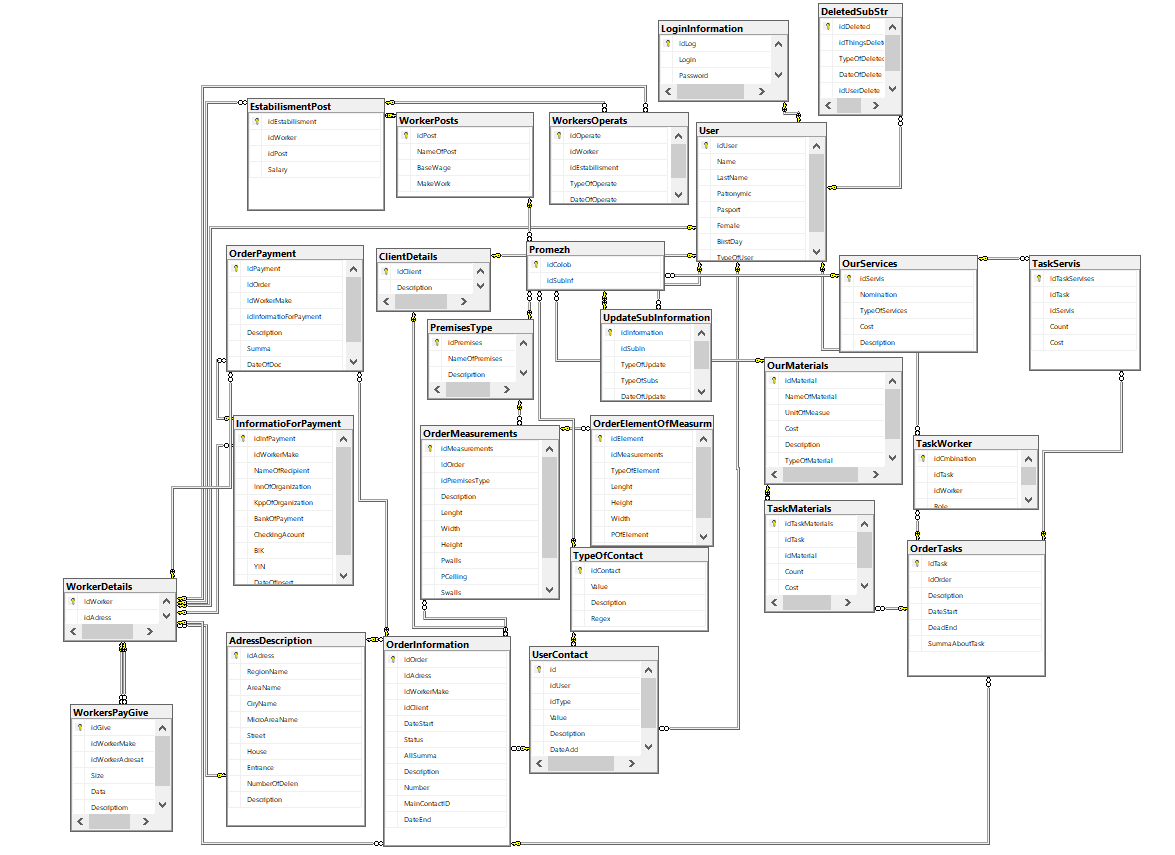


Рис. – Схема основной базы данных

Основная база данных состоит из 26 таблиц:

* AdressDescription – данные о адресе – данные об адресе как работника, так и адрес выполнения заказа;
* ClientDetails – данные о клиенте – детальная информация о клиенте данной фирмы;
* вDeletedSubStr – удаленная справочная информация – данные о том, какая справочная информация была удалена и когда это произошло;
* EstabilismentPost – установление должности за работником – информация о том, что за определенным работником была установлена определенная заработная плата и определенный должностной пост;
* InformatioForPayment –данные для оплаты – содержит реквизиты для оплаты клиентами заказов;
* LoginInformation – данные для авторизации – содержит информацию для авторизации, в том числе и клиентов при необходимости;
* OrderElementOfMeasurments – информация об элементах помещения – содержит данные о том, какие элементы помещения содержаться в данном помещении;
* OrderInformation – информация о заказе – содержит основную информацию о заказе, который поступил от клиента;
* OrderMeasurements – информация о помещениях для заказа – содержит данные о помещениях для заказа;
* OrderPayment – данные об оплате заказов – информация о том, какую оплату произвел клиент;
* OrderTasks – данные о задания для заказа – информация о заданиях для заказов;
* TaskWorker – данные о работниках, которые работают над заданием – информация о том какие рабочие выполняют работу над определенным заданием;
* OurMaterials – справочные данные о материалах – данные о том, какие материалы доступны для работы;
* OurServices – справочные данные об услугах – данные о том, какие услуги доступны для работы;
* PremisesType – тип помещений – данные о типах помещений;
* TaskMaterials – материалы для заданий – данные о материалах, которые используются в заданиях;
* TaskServis – услуги для заданий – информация об услугах, которые используются в задании;
* TypeOfContact – тип контактов – информация о типах контактов;
* User – пользователь – информация о пользователях, как о сотрудниках, так и о клиентах;
* UserContact – контактная информация пользователя – информация о контактах определенного пользователя;
* WorkerDetails – детальная информация о работнике – информация необходимая для работы с кадрами;
* WorkerPosts – доступные должности работников – информация о должностях доступных работникам;
* WorkersOperats – операции над должностями работников – определенные операции над должностями работников;
* Promezh – промежуточные данные – таблица для связи разных справочных данных в одну таблицу обновления;
* UpdateSubInformation – информация об обновлении справочных данных – данные о том, когда было произведено действие над определенным типом подставляемых данных;
* WorkersPayGive – данные о выдаче заработной платы работникам – информация о выдаче заработной платы работникам, принятым на работу.

# Глава 2. Практическая часть



## Разработка пользовательских интерфейсов программного обеспечения

Разработка пользовательского интерфейса являлась одной из основных стадий работы над проектом по причине необходимости работы приложения в адаптивном режиме.

Для реализации данного требования со стороны заказчиков программного обеспечения была выполнена разработка основного окна, которое повторяло бы все стандартные функции окна WPF проекта C#.

Среди основных функций можно считать следующие:

* возможность перемещать окно путем зажатия шапки окна;
* возможность сворачивать окно путем нажатия на соответствующую кнопку в шапке окна;
* возможность разворачивать окно на весь экран путем нажатия на соответствующую кнопку в шапке окна;
* возможность разворачивать окно путем двойного нажатия на его шапку.

Все вышеперечисленные требования удалось решить путем использования следующих библиотек:

* Microsoft.Windows.Shell – предоставляет возможность использовать стандартные функции окна Windows там, где это не предусмотрено изначально, что позволяет использовать данную библиотеку как средство работы с окном;
* Fody – необходима для синхронизации работы окна и внутренних свойств некоторых классов, которые будут использоваться в качестве классов для расчета некоторых свойств окна.

При использовании данного набора библиотек было создано адаптивное окно, представленное на рисунке 24.



Рис. – Базовое окно приложения

Пример адаптивности окна представлен на рисунке 25-26.

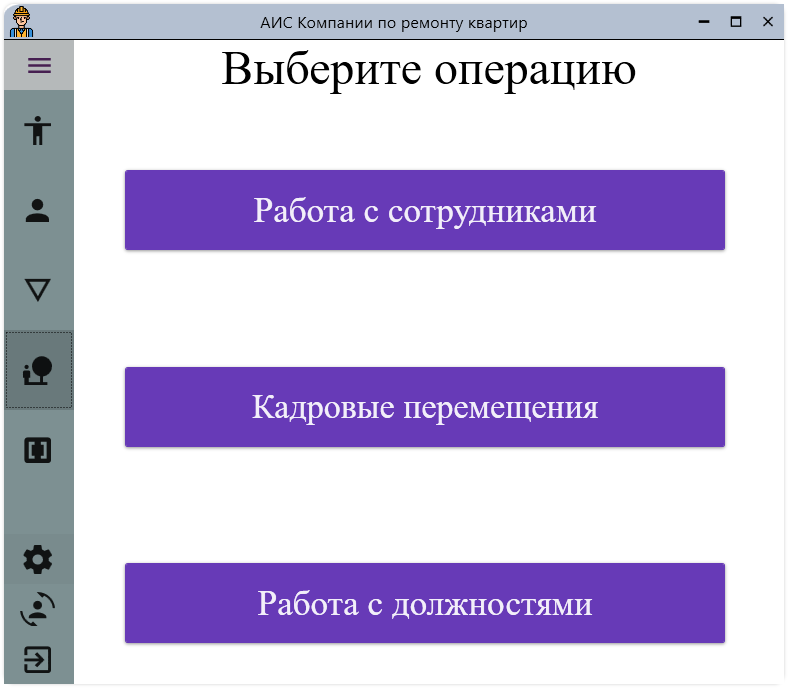


Рис. – Пример адаптивности окна

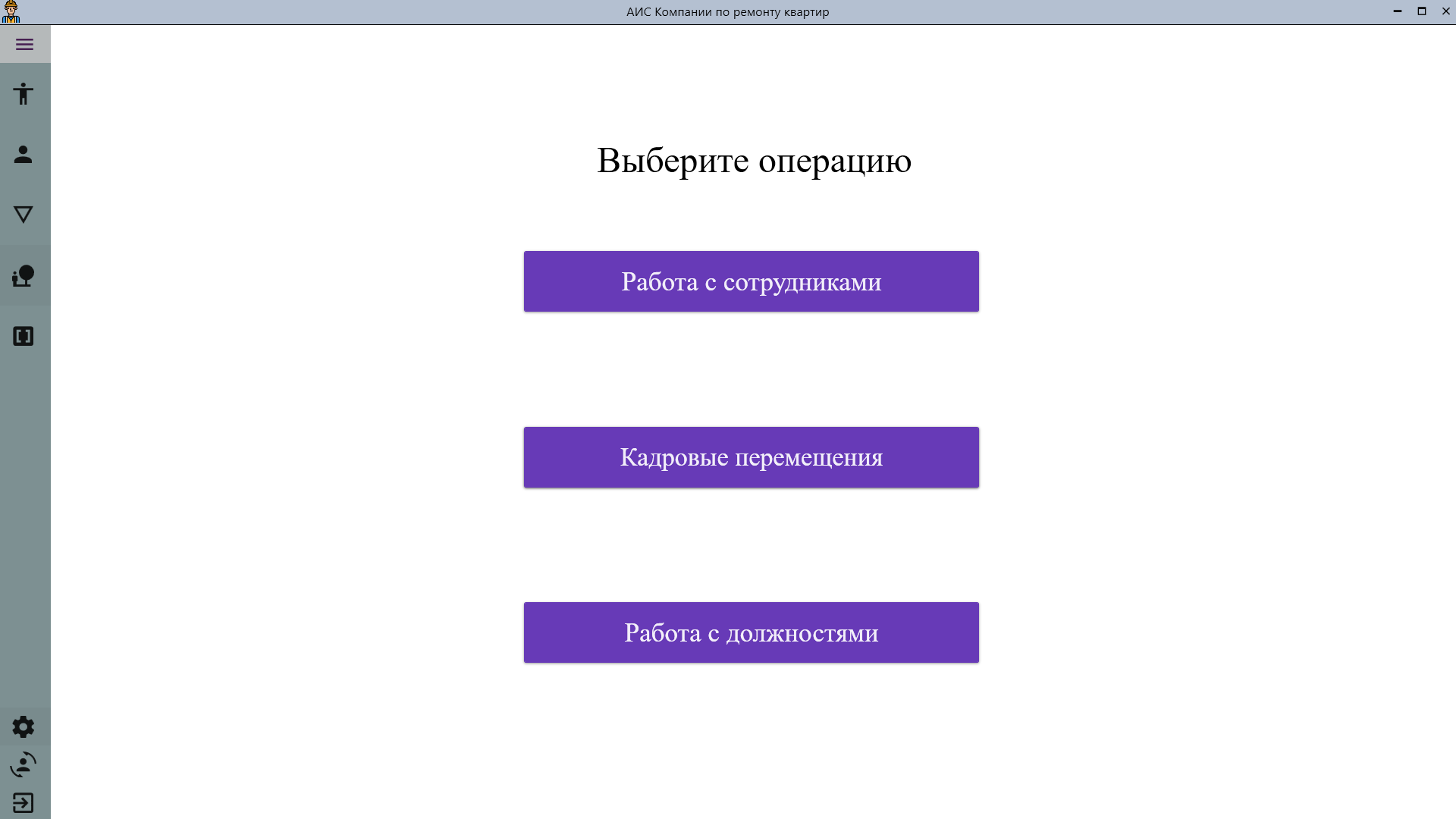


Рис. – Пример адаптивности окна

В качестве элемента для отображения содержимого окна был использован стандартный элемент WPF User Control.

Данный элемент управления представляет из себя некий «скелет», в который можно встроить различные элементы управления, что позволяет сделать вообще любой интерфейс для работы.

Для отображения данного элемента управления можно использовать стандартный элемент управления Grid. В который можно вставить User Control. В итоге пользователь увидит не 2 элемента управления, а один единственный, который будет отображать содержимое User control.

С помощью User Control можно сделать элементы управления адаптивными, что позволит сделать интерфейс изменяемым. Пример отображения User control в редакторе элементов управления WPF представлен на рисунке 27. При этом при отображении данного элемента управления на форме пользователь будет видеть примерно то, что представлено на рисунке 28.

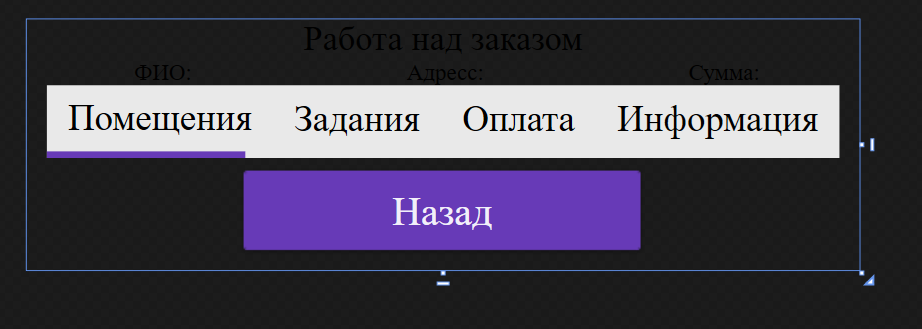


Рис. – Элемент управления User Control при отображении в редакторе содержимого

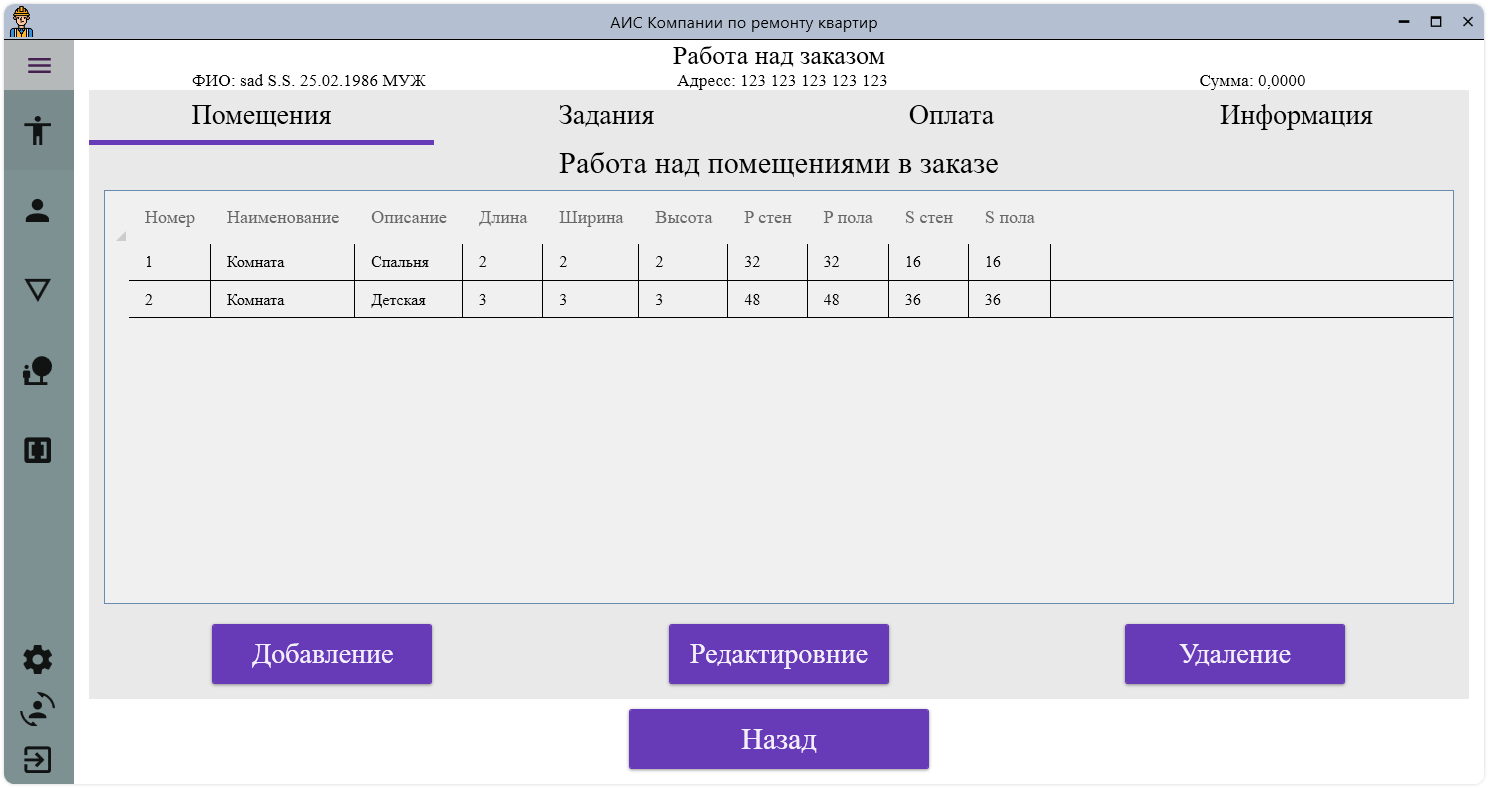


Рис. – Элемент управления User Control при отображении в окне

Для работы пользователей был создан целый набор элементов User Controlо, которые подставляются в зависимости от необходимости работы с определенными операциями.

Также по причине необходимости работы разных типов людей была создана система авторизации, пример которой представлен на рисунке 29.

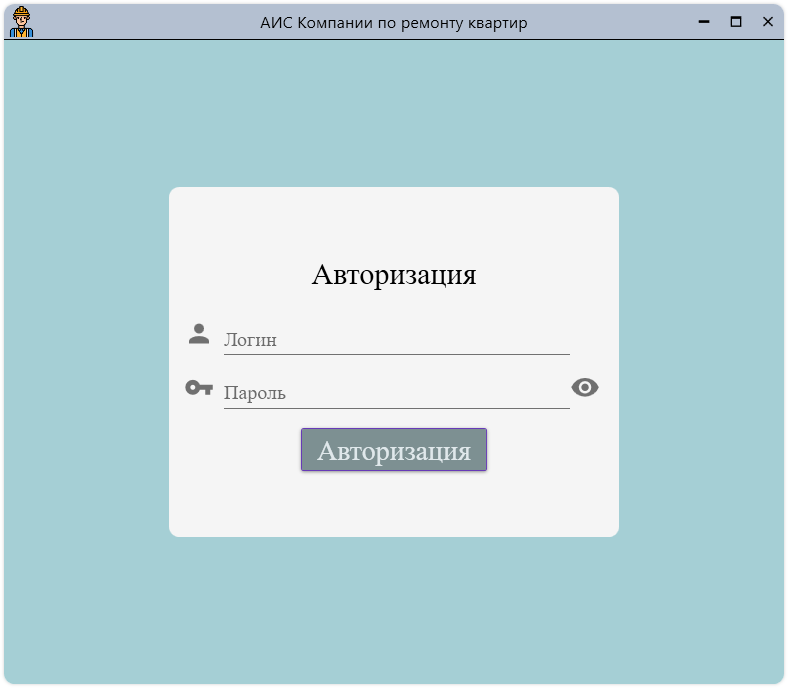


Рис. – Окно авторизации в приложении

После авторизации перед пользователем появляется окно загрузки, которое отображает степень загрузки определенных данных с сервера для ускорения работы при работе с этими данными, по причине того, что эти данные подставляются автоматически из локальной базы данных, которая создается автоматически. Окно загрузки представлено на рисунке 30.

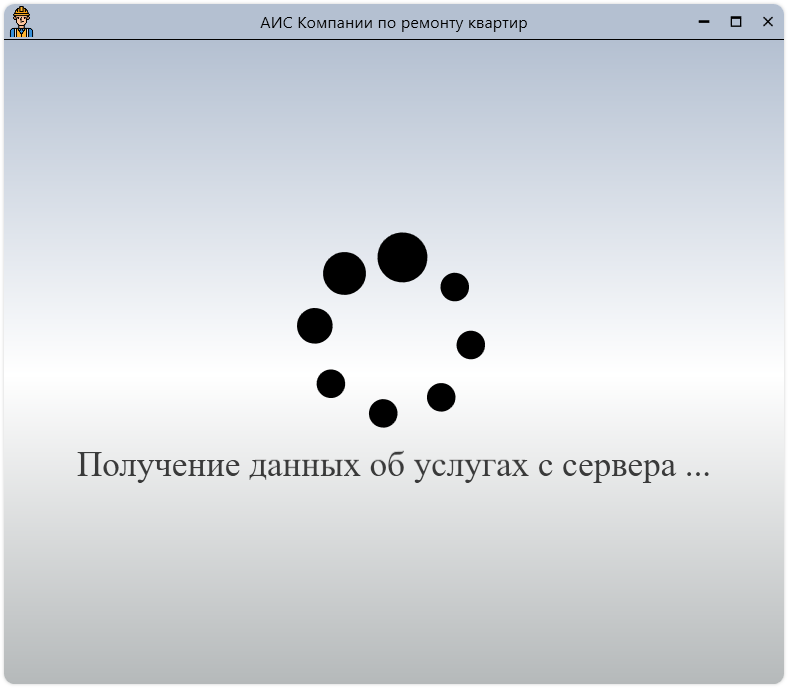


Рис. – Окно загрузки

Меню данного программного обеспечения представлено на рисунке 31. И представляет из себя боковую панель, которая при нажатии на кнопку будет увеличиваться.



Рис. – Меню приложения

Также приведу пример некоторый интерфейс приложения, который чем-либо примечателен. Пример данных окон представлен на рисунках 32-34.

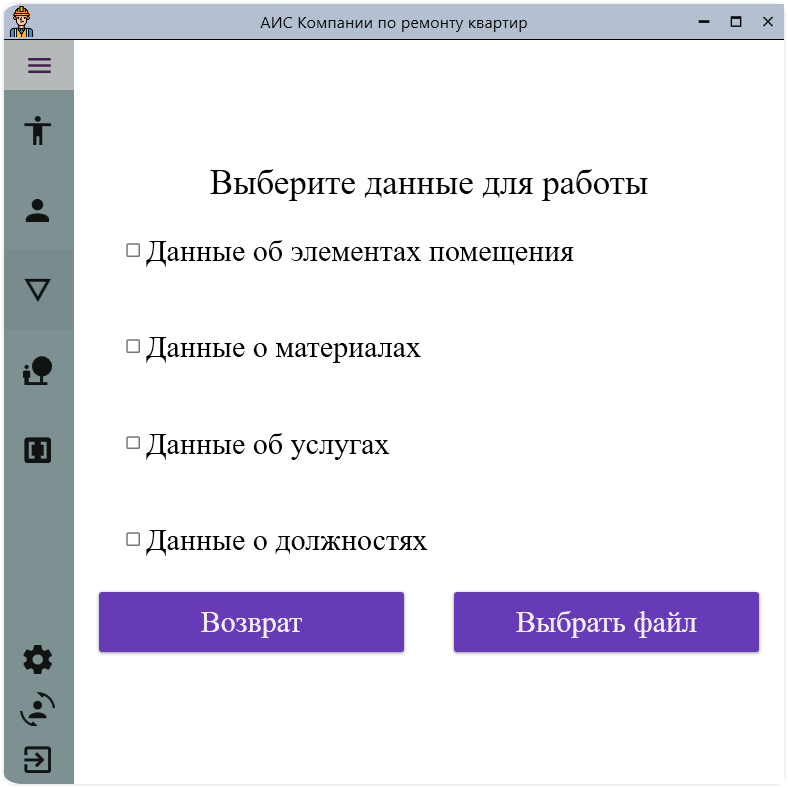


Рис. – Интерфейс приложения

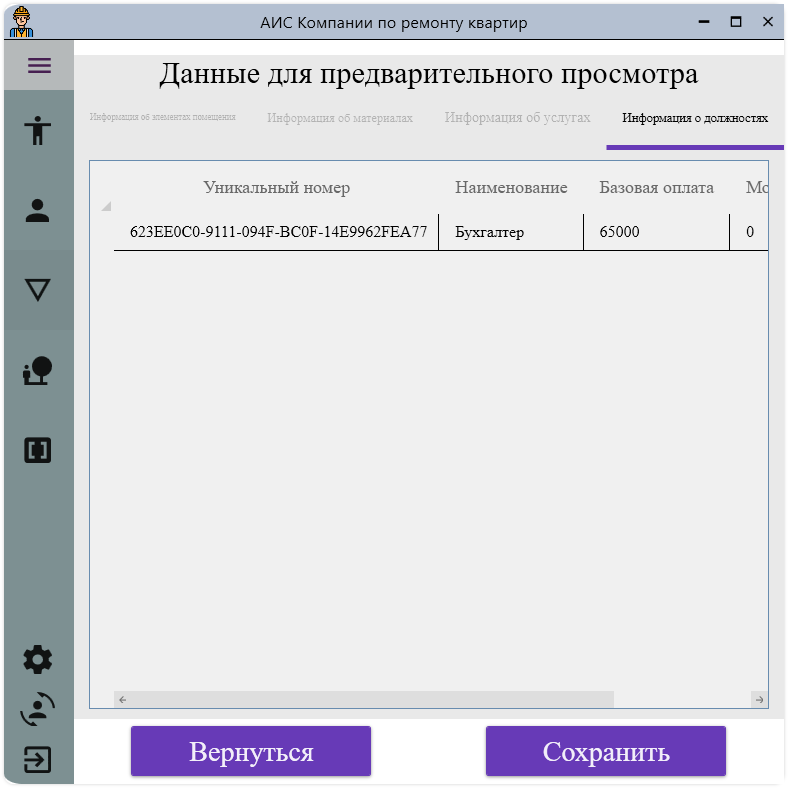


Рис. – Интерфейс приложения

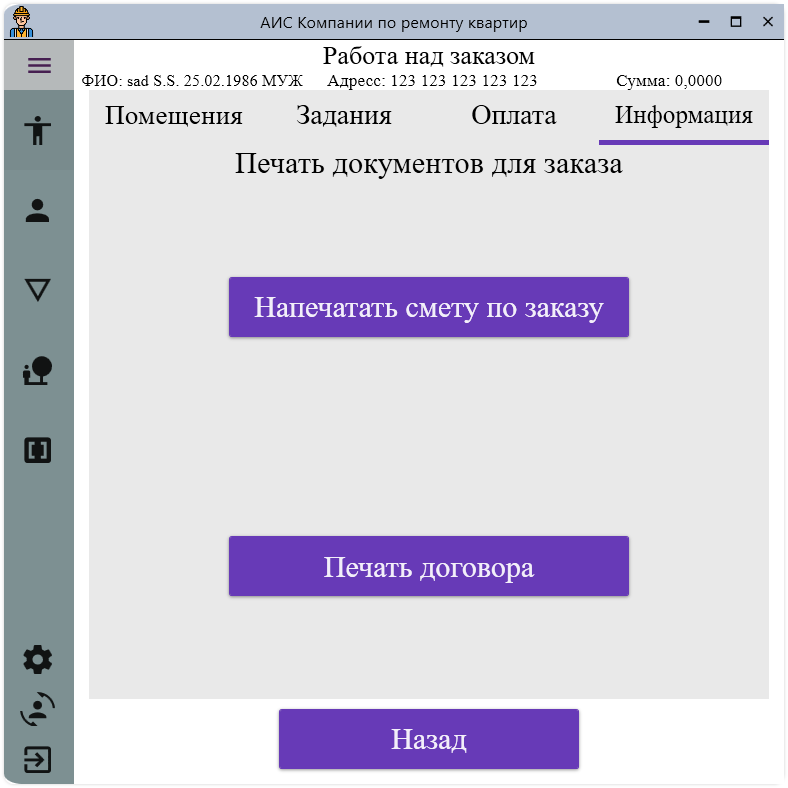


Рис. – Интерфейс приложения

## Тестирование и отладка программного обеспечения

Тестирование и отладка серверной части программного обеспечения происходило посредством программы Postman, подробности которой описаны в разделе 1.2.1 Характеристика среды программирования.

Данная программа позволяет отправлять запросы на сервер, который в данном случае представлен ASP.Net проектом на языке C#, который был разработан в «Visual Studio 2019», описание которого также приведено в разделе 1.2.1.

Visual Studio предоставляет собой хороший инструмент для тестирования и отладки программного обеспечения. Основным инструментом для этого можно считать исползование «точек останова».

Основным механизмом тестирования серверной части можно считать следующий:

* отправка запросов программой Postman;
* проверка на отсутствие ошибок при работе серверной части программы;
* при возникновении неточностей ставится точка останова в месте, где начинается работа определенного метода, что помогает выявить причину данной ошибки.

После создания и отладки серверной части программного продукта настало время клиентской программы. Отладка данной части программного продукта также проходила в «Visual Studio 2019».

Основной проблемой при разработке было то, что при разработке моделей, которые преобразуются в JSON возникали опечатки, которые препятствовали получению корректной информации, либо вообще ошибки при работе со стороны серверной части.

При наличии такого количества объектов при разработке избежать ошибок крайне сложно и не редко установление причин данных ошибок может занять большое количество времени и сил.

В качестве результатов тестировании можно привести выходные документы, созданием которых и предназначено заниматься данной программе.

Данные документы представлены на рисунке 35-37.

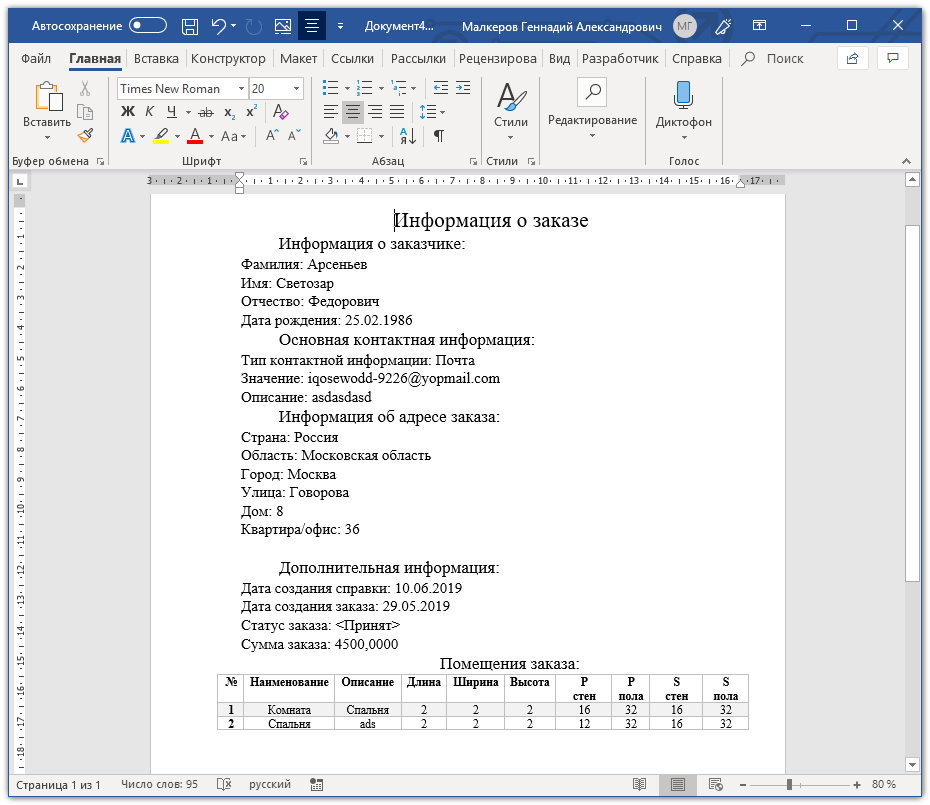


Рис. – Выходной справочный документ о заказе

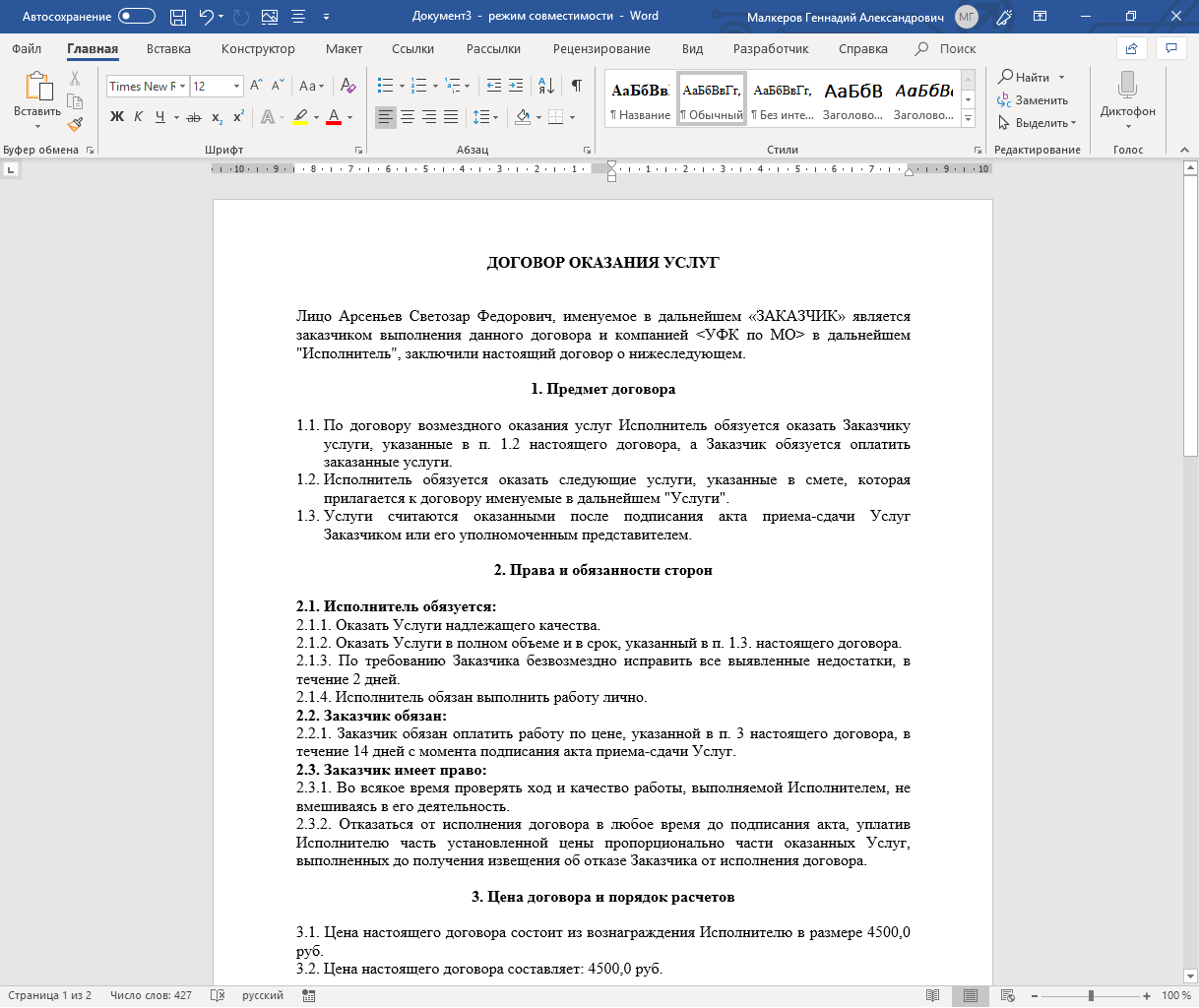


Рис. – Выходной договор на оказание услуг

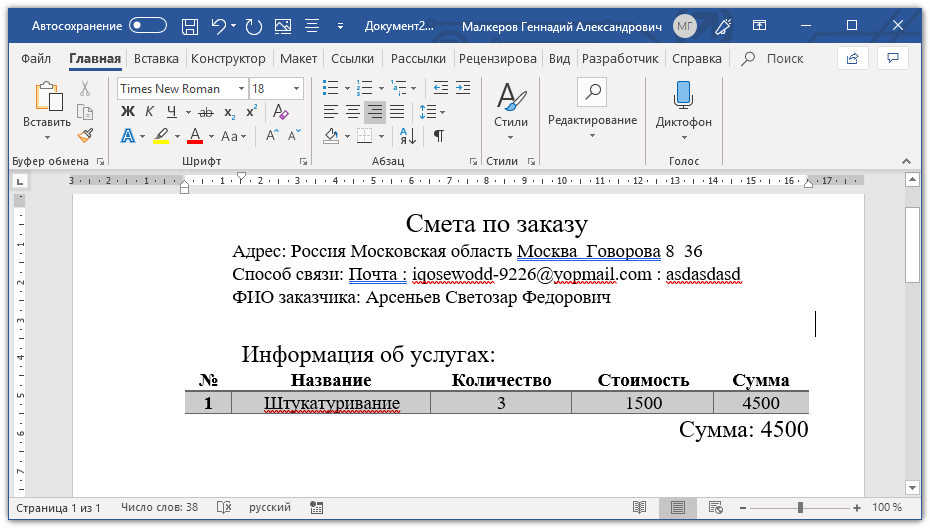


Рис. – Смета по заказу



## Руководства по использованию программного обеспечения

### Руководство системного программиста

* + - 1. Общие сведения о программе

Данная программа предназначена для автоматизации работы фирмы по ремонту квартир и позволяет автоматизировать некоторые рутинные процессы, которые выполняются в типичной компании по данной тематике.

Программа позволяет следующие процессы:

* составление сметы на материалы и услуги по определенным заданиям, которые были созданы;
* составление договора на оказание услуг;
* расчет площади элементов помещений с учетом элементов данных помещений;
* генерация отчетов в Word о различных хозяйственных операциях, происходящих в данной фирме;
* возможность загрузки справочной информации из файла Excel;
* разделенный режим доступа к программе;
* возможность работы с сотрудниками.

#### Проверка программы

Проверка работоспособности программы заключается в процессе отображения на форме авторизации сообщения, о том, что связь с сервером была установлена.

#### Требование к исполняемому серверу

Для работы данного программного комплекса необходимо наличие некоторого удаленного сервера, на котором предполагается выполнение серверной части программного обеспечения.

Среди требований к исполняемому серверу можно отметить следующие:

* наличие .Net Framework версии не ниже 4.6;
* наличие не менее 6 гб свободного пространства на диске;
* наличие не менее 2 гб оперативной памяти;
* частота процессора не менее 1,4 ГГц;
* тип процессора x64: AMD Opteron, AMD Athlon 64, Intel Xeon с поддержкой Intel EM64T, Intel Pentium IV с поддержкой EM64T.

#### Сообщения системному администратору

Сообщения системному администратору не предусмотрены в данной программе.

#### Настройка программы

Перед началом работы с программой её необходимо установить. Для того, чтобы это сделать необходимо запустить установщик, который идет в комплекте «RepairFlat.exe».

После запуска программы перед пользователем появиться окно, представленное на рисунке 38.

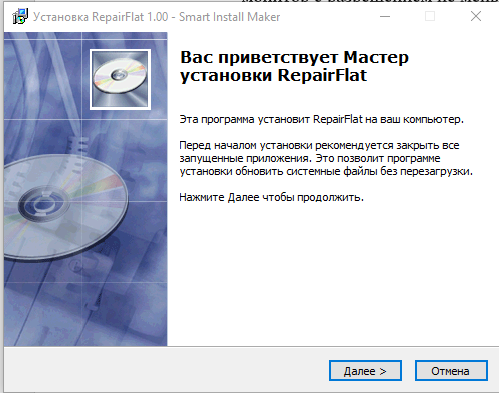


Рис. – Начальное окно установщика программного продукта

После нажатия кнопки далее перед пользователем появляется окно выбора пути расположения файлов установки, пример отображения данного окна представлен на рисунке 39.

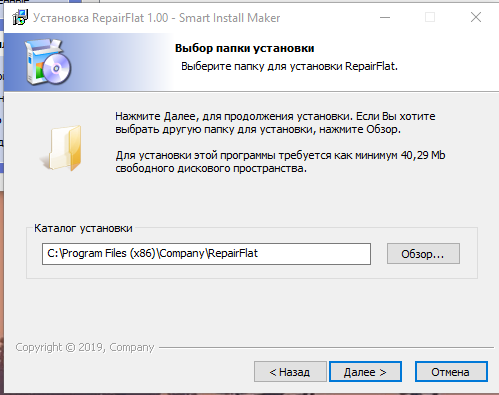


Рис. – Этап выбора пути расположения файлов

После указания необходимой рабочей папки и перехода на следующий этап, перед пользователем появляется окно, представленное на рисунке 40, на котором предполагается выбрать необходимость ярлыка на рабочем столе установленной программы.

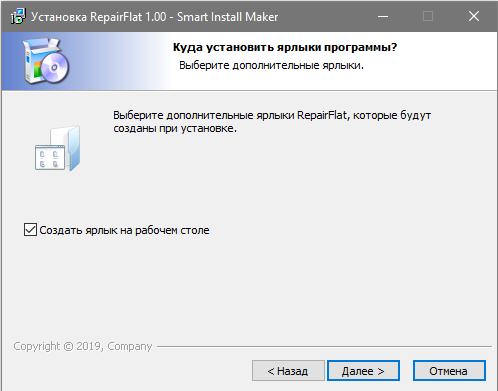


Рис. – Этап подтверждения создания ссылки на программу на рабочем столе

Данный этап был последним, после чего при нажатии кнопки далее перед пользователем появляется окно установки продукта, которое представлено на рисунке 41.

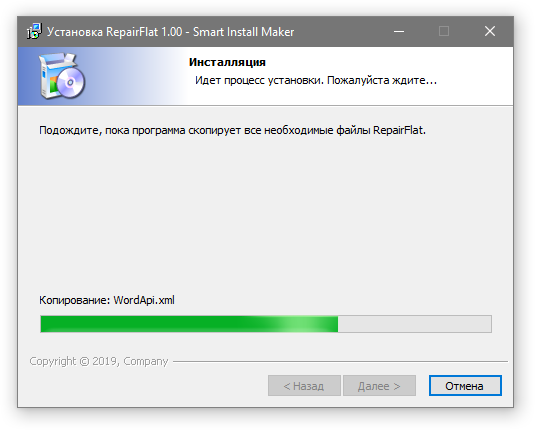


Рис. – Этап установки на диск необходимых библиотек

Теперь программу можно запускать посредством нажатия на соответствующую иконку на рабочем столе, пример отображения иконки представлен на рисунке 42.



Рис. – Отображение иконки на рабочем столе

После установки программы необходимо задать базовый сервер. Для того, чтобы это сделать необходимо указать в файле «RepairFlatWPF.exe.config» значение параметра BaseAdress, который отвечает за значение сервера для авторизации.

#### Структура программы

Структура программы представлена в качестве схемы на рисунке 43.

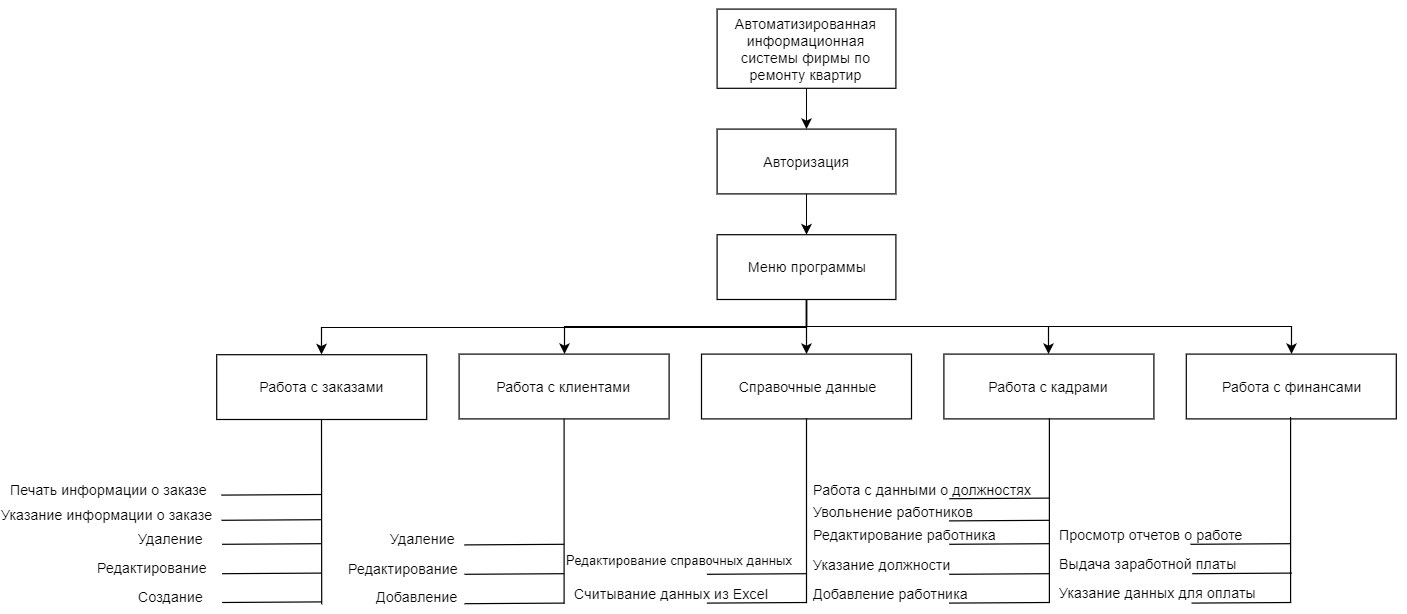


Рис. –Структурка программы

### Руководство программиста

#### Назначения и условия выполнения программы

Данная программа предназначена для автоматизации работы фирмы по ремонту квартир и позволяет автоматизировать некоторые рутинные процессы, которые выполняются в типичной компании по данной тематике.

Программа автоматизирует следующие процессы:

* составление сметы на материалы и услуги по определенным заданиям, которые были созданы;
* составление договора на оказание услуг;
* расчет площади элементов помещений с учетом элементов данных помещений;
* генерация отчетов в Word о различных хозяйственных операциях, происходящих в данной фирме;
* возможность загрузки справочной информации из файла Excel;
* разделенный режим доступа к программе;
* возможность работы с сотрудниками.

Необходимыми программными средствами для работы будут являться:

* операционная система Windows 10;
* приложение Microsoft Word и Microsoft Excel;
* наличие .Net Framework 4.7 версии и выше.

Необходимыми аппаратными средствами является:

* процессор с тактовой частотой 2,7 ГГц или выше. Рекомендуется использовать двухъядерный процессор;
* 1 ГБ ОЗУ (3 ГБ при выполнении в виртуальной машине);
* 10 ГБ доступного пространства на жестком диске;
* видеоадаптер с поддержкой DirectX 9 (разрешение 1600 x 900 или больше);
* монитор с разрешением не меньше, чем (1600 x 900);
* наличие интернет-соединение;
* клавиатура и мышь.

Стоит отметить, что программу следует запускать от имени администратора при установке данной программы в корень диска «C», в противном случае программа перестанет работать.

#### Обращение к программе

Обращение к программе происходит по средствам раннее установленной на компьютер программы «RepairFlat», которая при стандартной установке будет на рабочем столе в качестве ссылки.

#### Входные и выходные данные

Входные и выходные данные для программы представлены в предпроектном исследований предметной области.

#### Сообщения

Сообщения программисту в данной автоматизированной информационной системе не предусмотрено.

#### Характеристики программы

При разработке программного комплекса используются язык программирования C# версии 7.0. NET Framework 4.7.1 и с использованием технологий:

* технологии WPF для клиентского приложения;
* технология ASP.Net для серверной составляющей данной программы.

### Руководство пользователя

#### Назначение программы

Данная программа предназначена для автоматизации работы фирмы по ремонту квартир и позволяет автоматизировать некоторые рутинные процессы, которые выполняются в типичной компании по данной тематике.

Программа автоматизирует следующие процессы:

* составление сметы на материалы и услуги по определенным заданиям, которые были созданы;
* составление договора на оказание услуг;
* расчет площади элементов помещений с учетом элементов данных помещений;
* генерация отчетов в Word о различных хозяйственных операциях, происходящих в данной фирме;
* возможность загрузки справочной информации из файла Excel;
* разделенный режим доступа к программе;
* возможность работы с сотрудниками.

#### Условия выполнения программы

Необходимыми программными средствами для работы будут являться:

* операционная система Windows 10;
* приложение Microsoft Word и Microsoft Excel;
* наличие .Net Framework 4.7 версии и выше.

Необходимыми аппаратными средствами является:

* процессор с тактовой частотой 2,7 ГГц или выше. Рекомендуется использовать двухъядерный процессор;
* 1 ГБ ОЗУ (3 ГБ при выполнении в виртуальной машине);
* 10 ГБ доступного пространства на жестком диске;
* видеоадаптер с поддержкой DirectX 9 (разрешение 1600 x 900 или больше);
* монитор с разрешением не меньше, чем (1600 x 900);
* наличие интернет-соединение;
* клавиатура и мышь.

#### Выполнение программы

После запуска программы на выполнение перед пользователем демонстрируется окно, представленное на рисунке 44, на котором необходимо будет ввести логин и пароль работника.

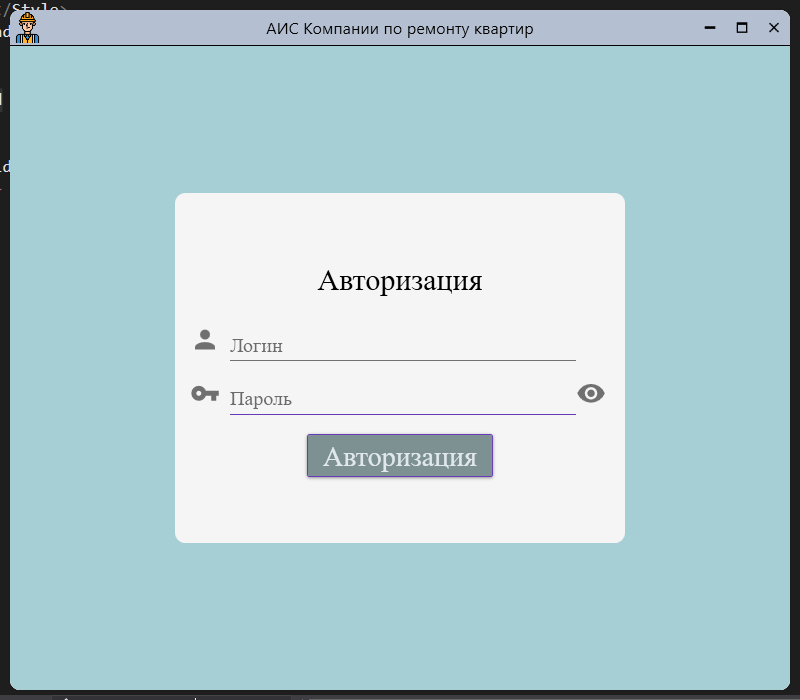


Рис. – Окно авторизации в системе

При неверном логине либо пароле пользователь получит сообщение, представленное на рисунке 45.

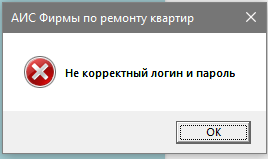


Рис. – Сообщение о неверном логине либо пароле

При работе с программой возможно использование в многопользовательском режиме с использованием следующего списка ролей:

* работник отдела кадров;
* менеджер;
* администратор;
* работник финансового отдела
* директор.

Рассмотрим особенности каждой роли на примере работы программы.

Работник отдела кадров имеет возможность работы только с кадрами фирмы. После авторизации под данной ролью перед пользователем представляется окно, представленное на рисунке 46.

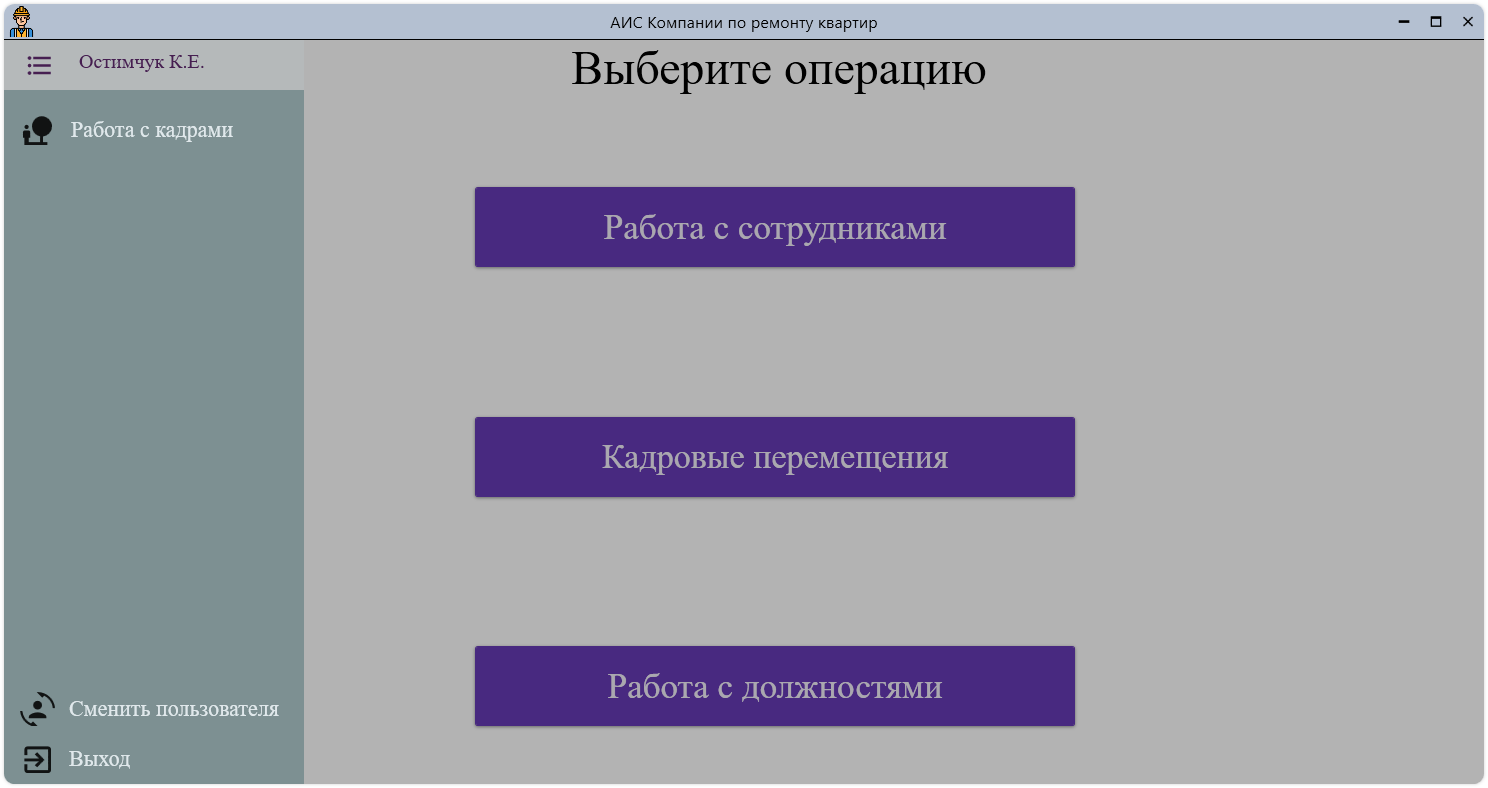


Рис. – Главное меню приложения при авторизации под правами кадрового работника

После закрытия бокового меню пользователь может выполнить следующие действия:

* создание нового работника;
* редактирования данных о работниках;
* принятие на работу нового сотрудника;
* изменение должности работника;
* указание данных о логине и пароле для работника.

Для примера создадим нового работника. Все совпадения с реальными людьми являются случайностью.

Чтобы добавить нового работника в данную программу необходимо зайти в основное окно работы с пользователем, после чего выбрать пункт «Работа с сотрудниками». Далее необходимо нажать на кнопку «Добавить нового работника», появиться окно, представленное на рисунке 47.

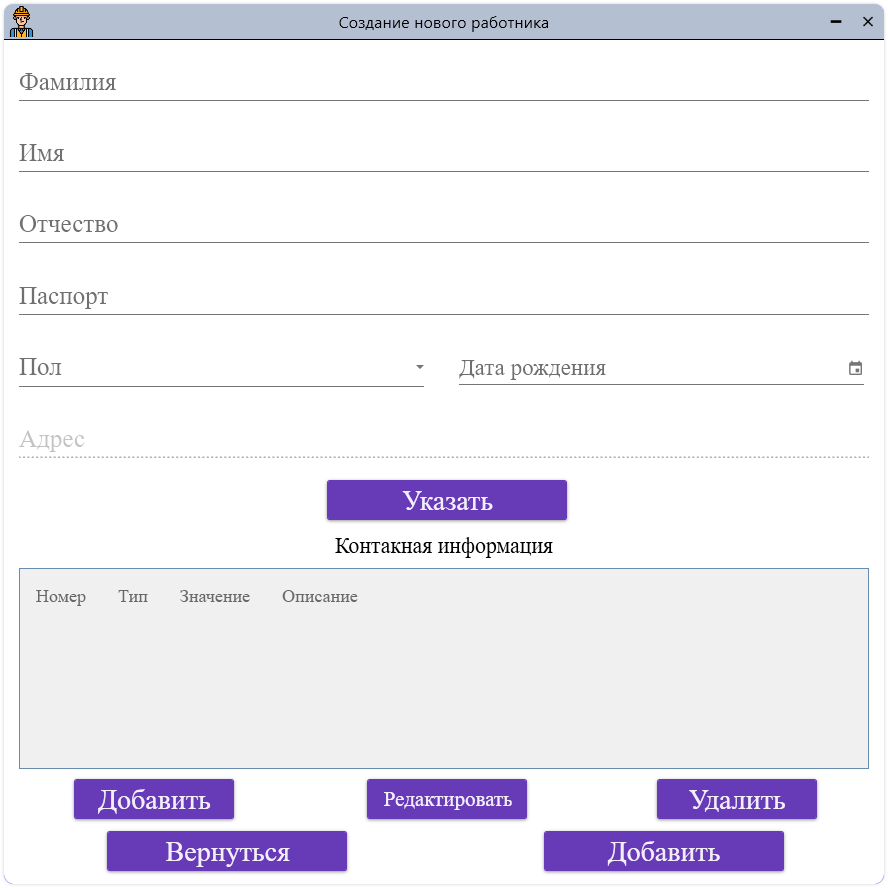


Рис. – Создание нового работника

При создании работника либо клиента появляется возможность добавления данных о контактной информации. Пример окна добавления контактной информации представлен на рисунке 48.

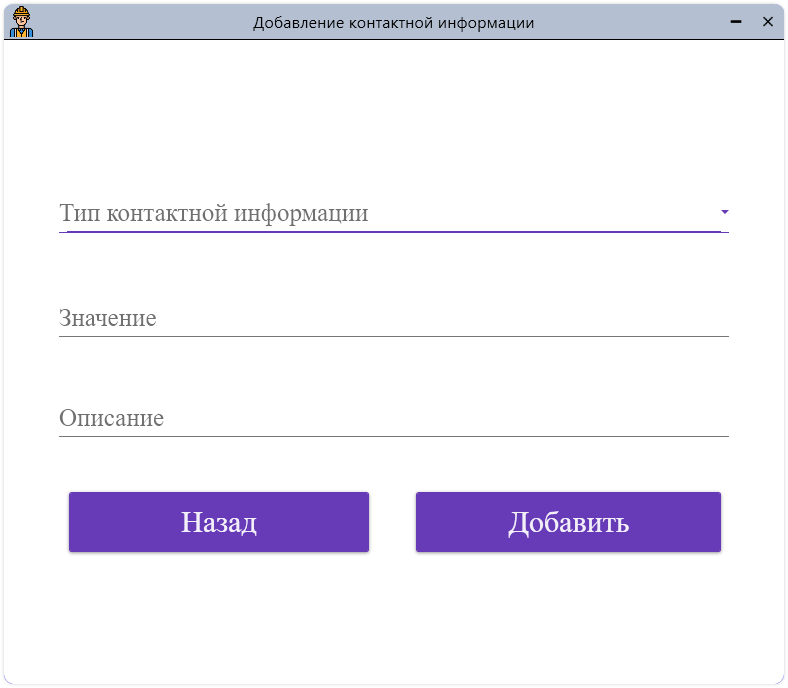


Рис. – Создание нового контакта

Также, при создании работника добавляется информация об адресе. Окно добавления/ редактирования адреса представлено на рисунке 49.

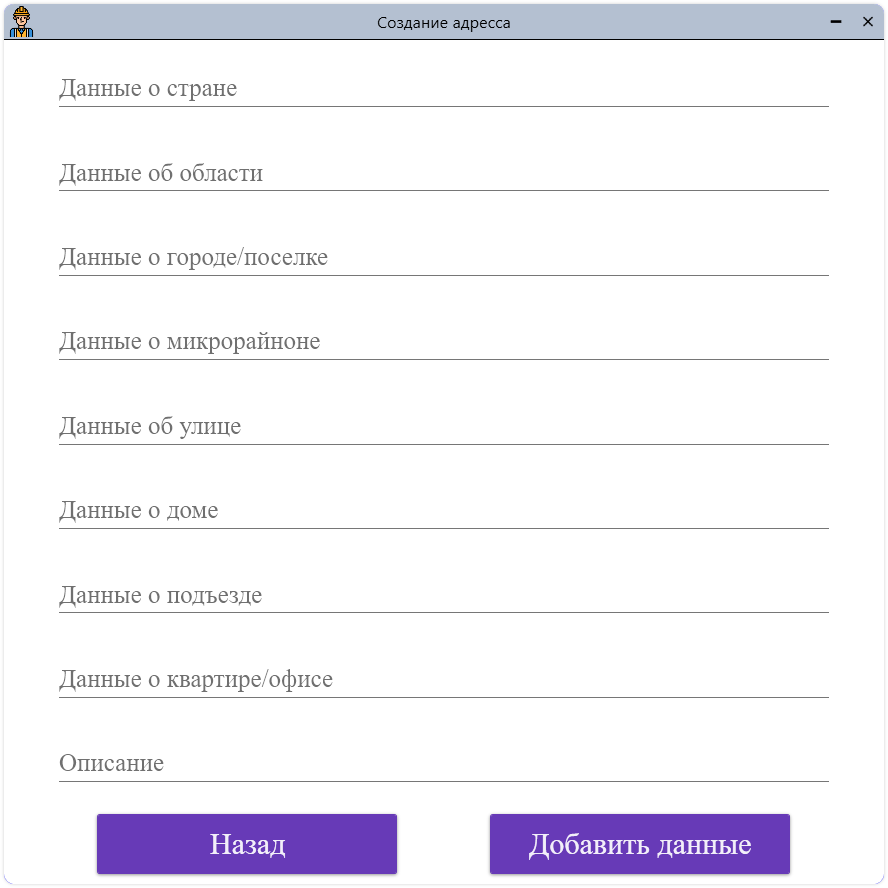


Рис. – Создание нового контакта

После заполнения данных о работнике (пример заполнения данных о работнике представлен на рисунке 50), появляется сообщение, представленное на рисунке 51.

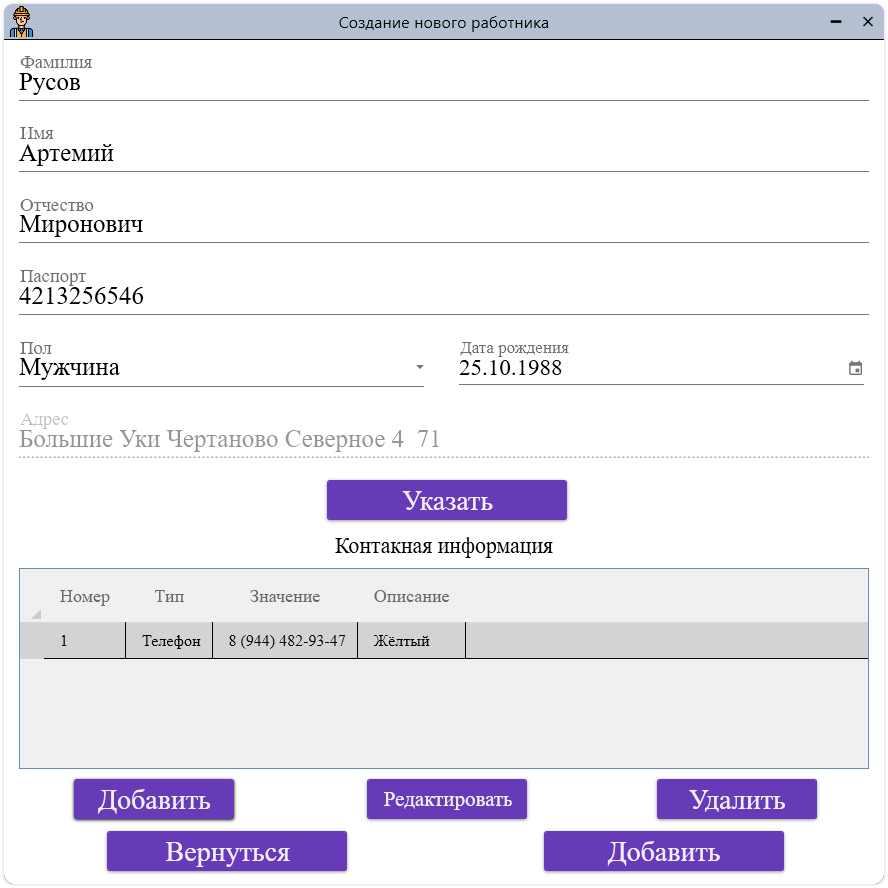


Рис. – Пример добавления данных о работнике

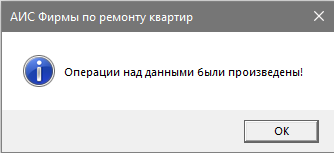


Рис. – Сообщение после добавления нового работника

Окно с выбором работников представлено на рисунке 52.

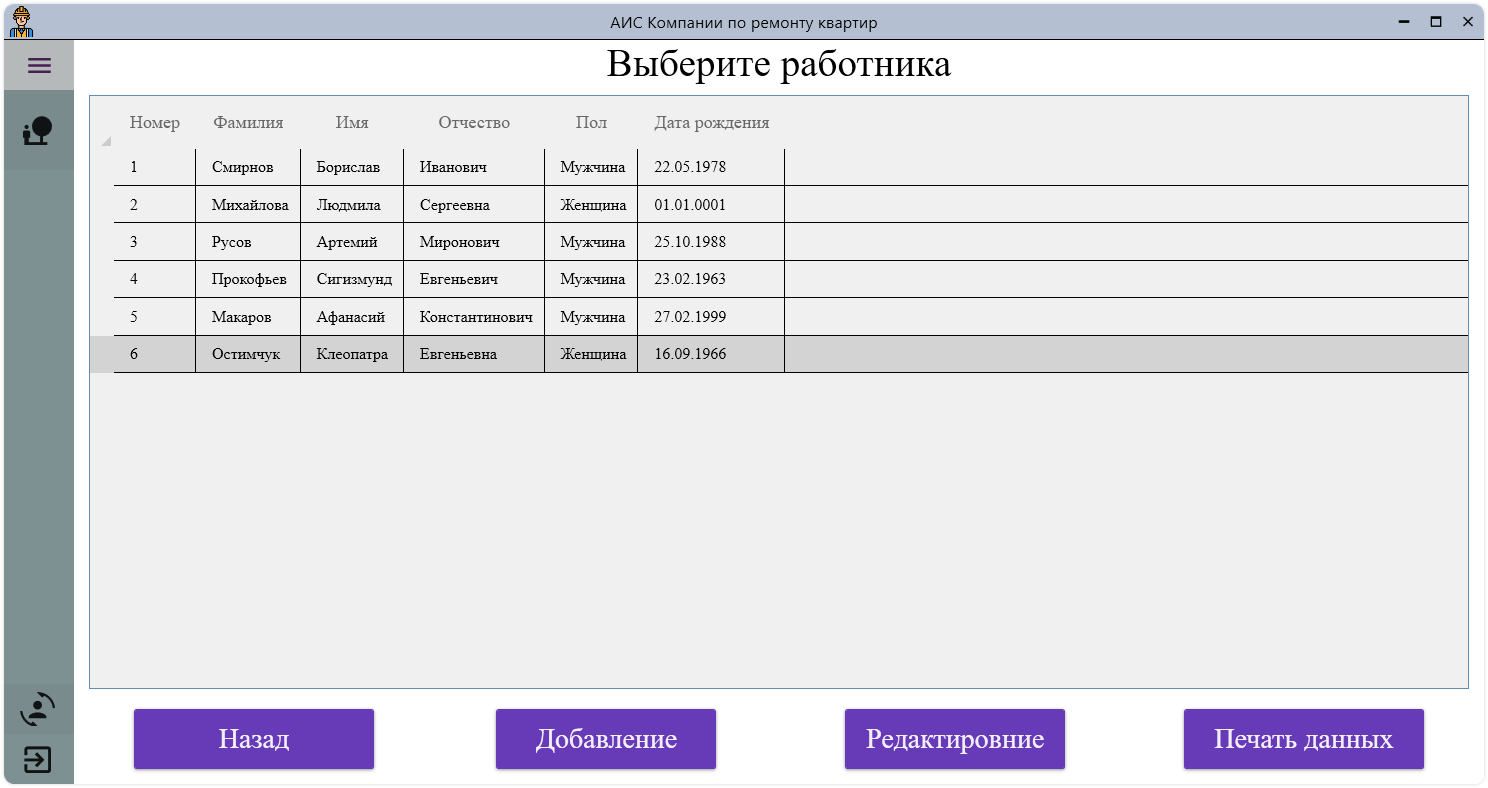


Рис. – Выбор работников для работы

Также на данном окне можно распечатать данные о работнике в документ WORD. Пример вывода представлен на рисунке 53.

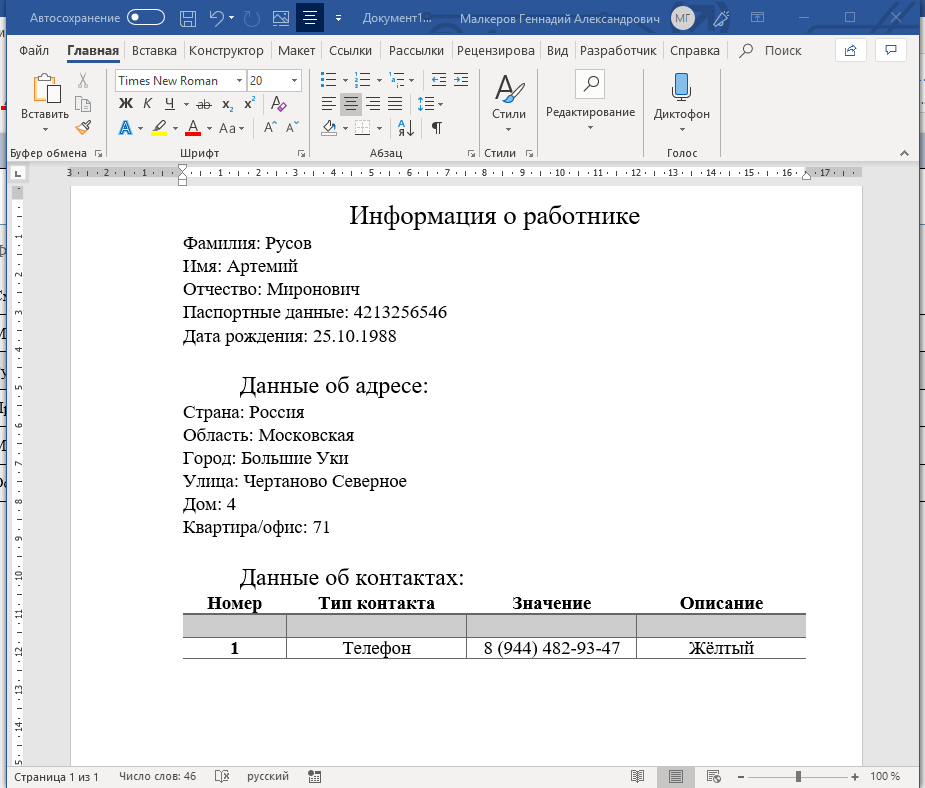


Рис. – Данные о работнике в WORD

При возврате назад путем нажатия на кнопку назад и выбора пункта «Кадровые перемещения», перед пользователем представляется окно, представленное на рисунке 54.

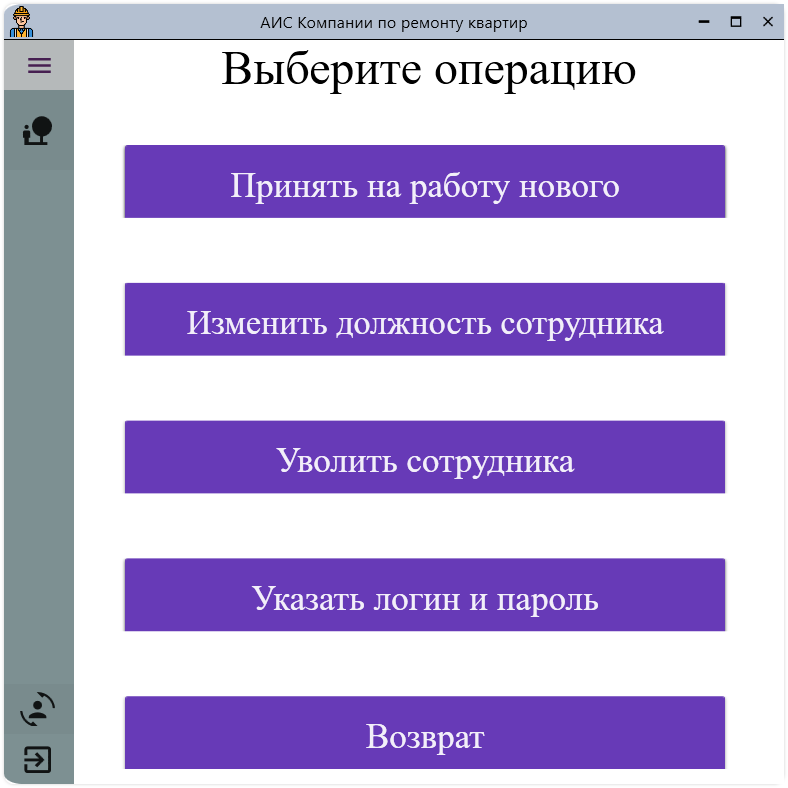


Рис. – Пункт кадровые перемещения

При выборе пункта «Принятие на работу» перед пользователем представлена информация, пример отображения данного окна представлен на рисунке 55.

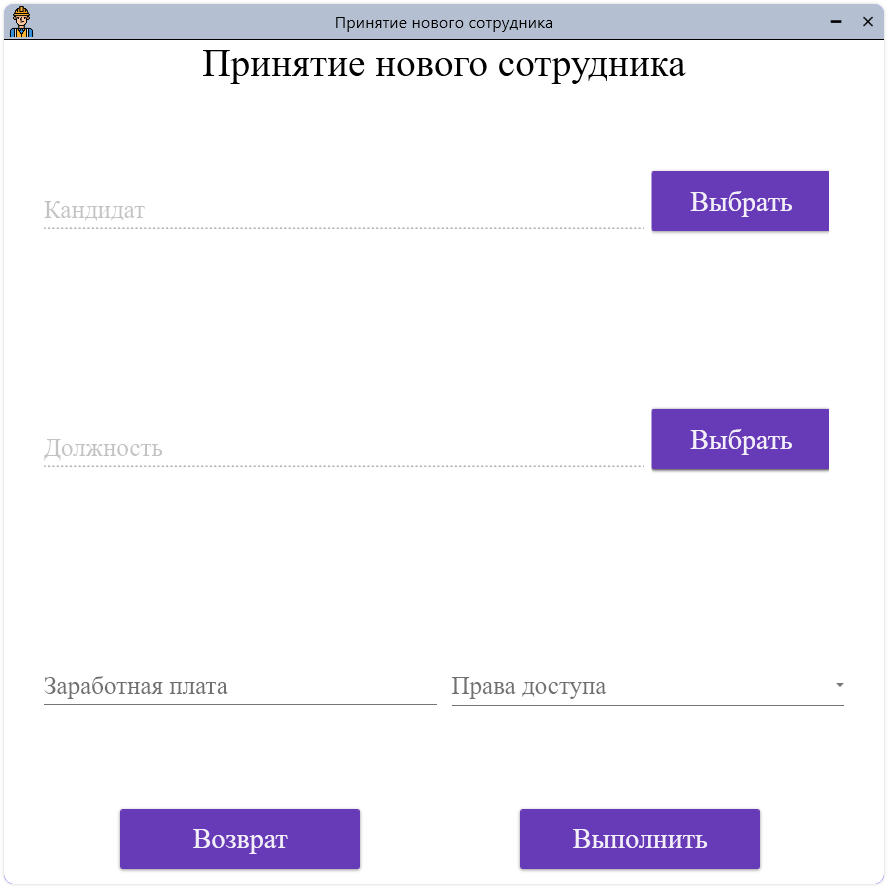


Рис. – Пункт принятие на работу

При необходимости выбора данных о должностях, перед пользователем представлено появляется окно, представленное на рисунке 56.

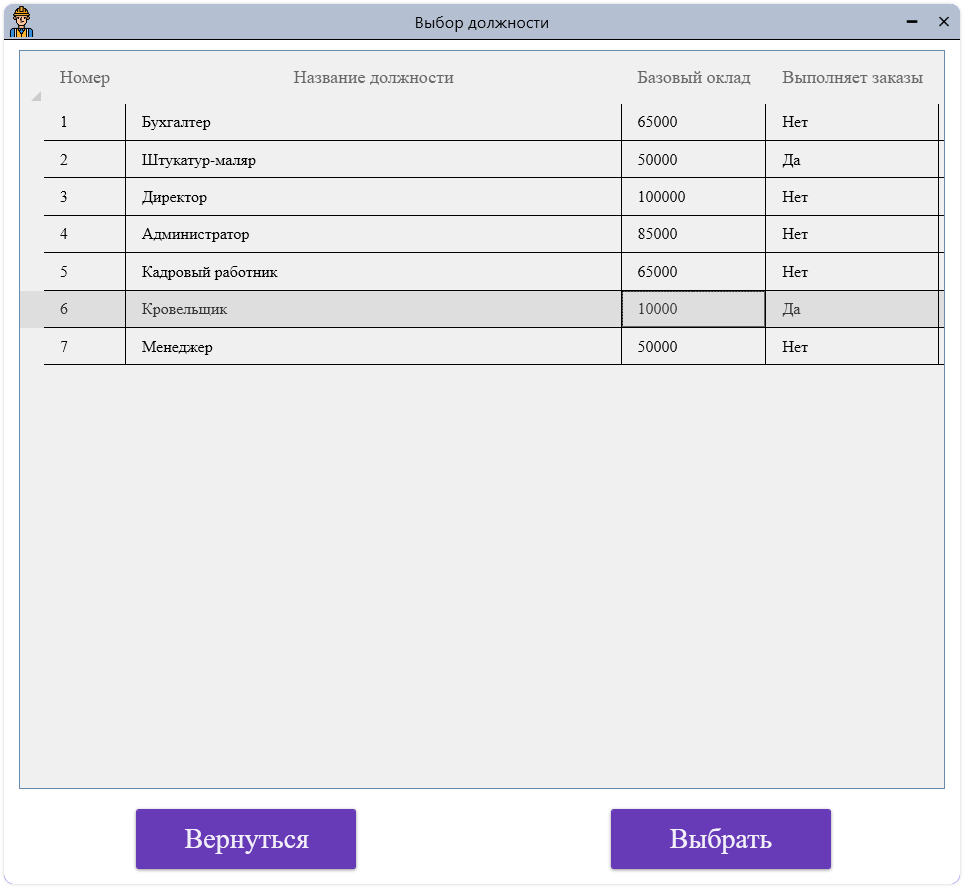


Рис. – Окно выбора должности

После выбора должности и работника, а также указания данных о заработной плате и указания сферы влияния в программу и выбора пункта «Выполнить» перед пользователем появляется сообщение, представленное на рисунке 49.

Также в меню, представленном на рисунке 50, можно сделать следующие действия:

* принятие на работу нового сотрудника;
* изменение должности для работающего сотрудника;
* увольнение сотрудника;
* указание логина и пароля для авторизации;

Также помимо выполнения описанных выше пунктов кадровый работник может редактировать данные о должностях и дополнять их.

Для выполнения данных операций пользователь должен перейти в пункт «Работа над должностями» меню кадрового работника, перед пользователем появиться окно, представленное на рисунке 57. При выборе пункта «Добавление» перед пользователем появиться окно, представленное на рисунке 58.

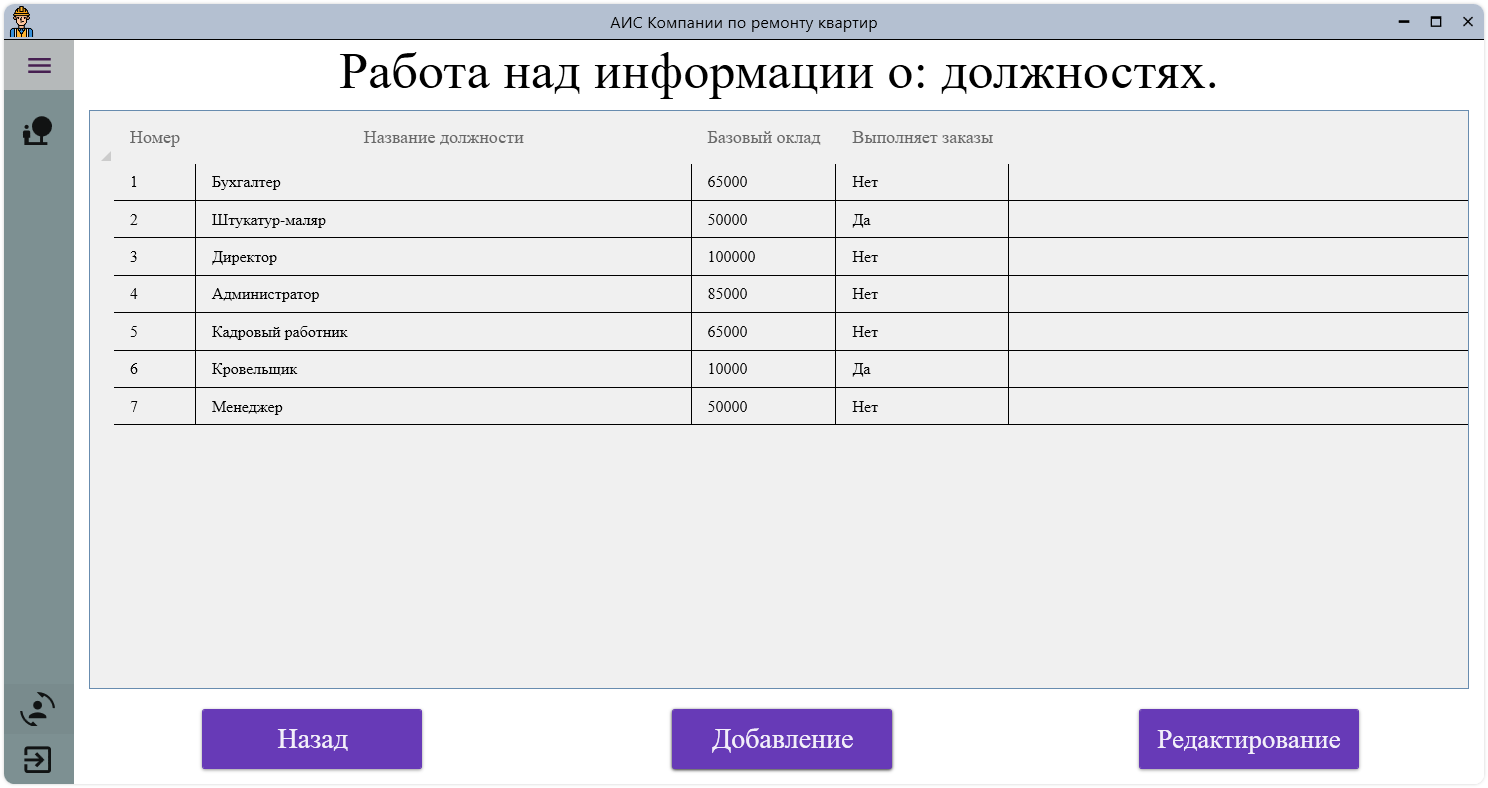


Рис. – Окно просмотра должностей

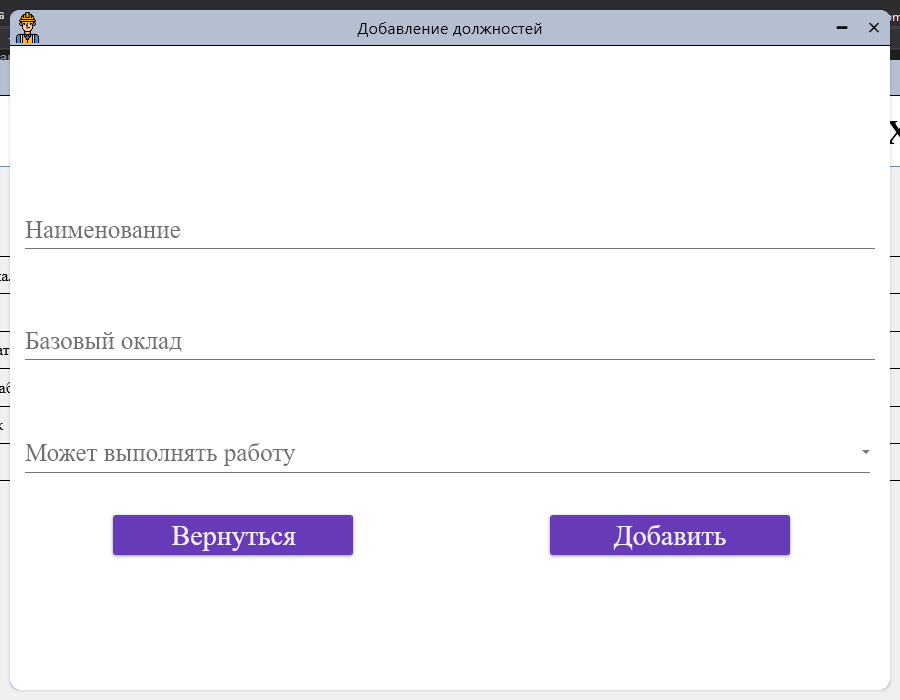


Рис. – Окно добавления данных о должностях

Роль администратора предполагает, что человек, имеющий данные права, сможет настраивать данную программу при работе в некоторых важных аспектах.

При авторизации в данной роли перед пользователем демонстрируется меню, представленное на рисунке 59. Данному пользователю (типу пользователей) предоставлен доступ к пунктам:

* настройки;
* справочные данные.

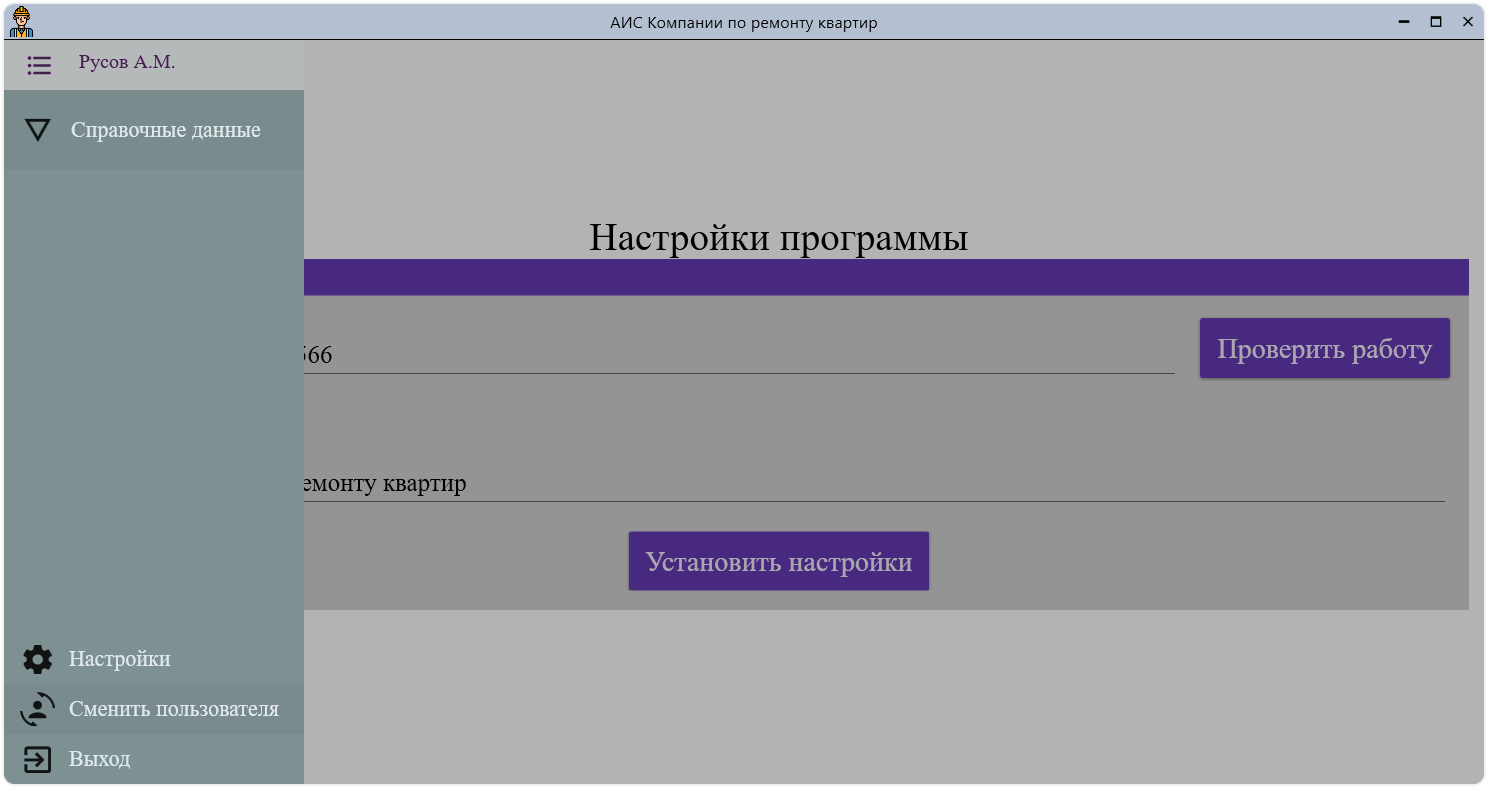


Рис. – Главное меню приложения при авторизации под правами администратора

Раздел настроек данной программы включает в себя следующие настройки:

* адреса базового сервера для подключения к нему;
* текста заголовка при открытии диалогового окна по типу MessageBox.

Пример отображения данного окна редактирования данных представлен на рисунке 60.

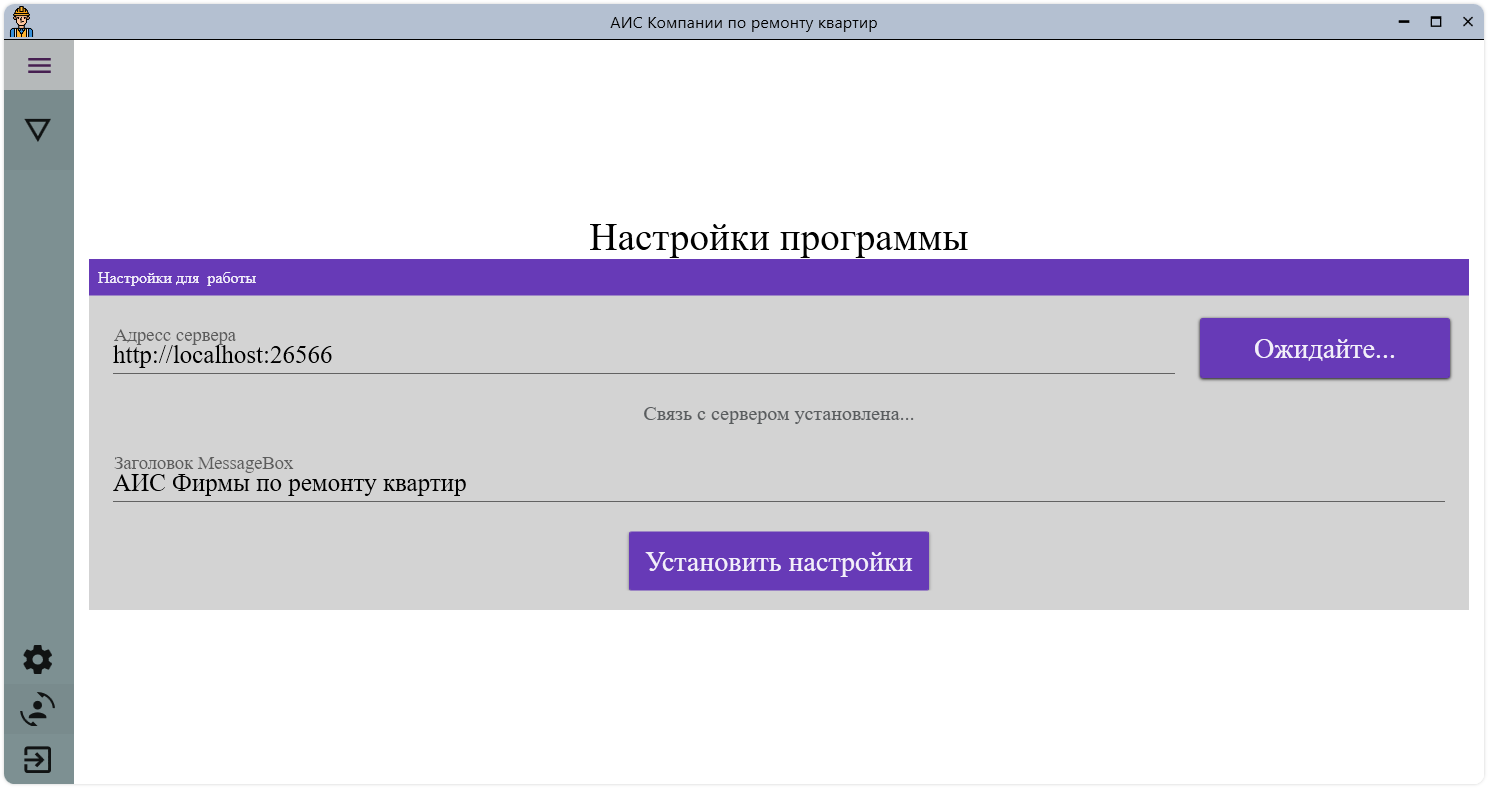


Рис. – Окно настроек программы

Для сохранения указанных настроек необходимо указать актуальный сервер, для проверки данного факта также имеется соответствующая функция.

Также администратор может открыть окно работы со справочными данными, пример отображения данного окна представлен на рисунке 61. Данная возможность необходима для контроля работы локальной базы данных.

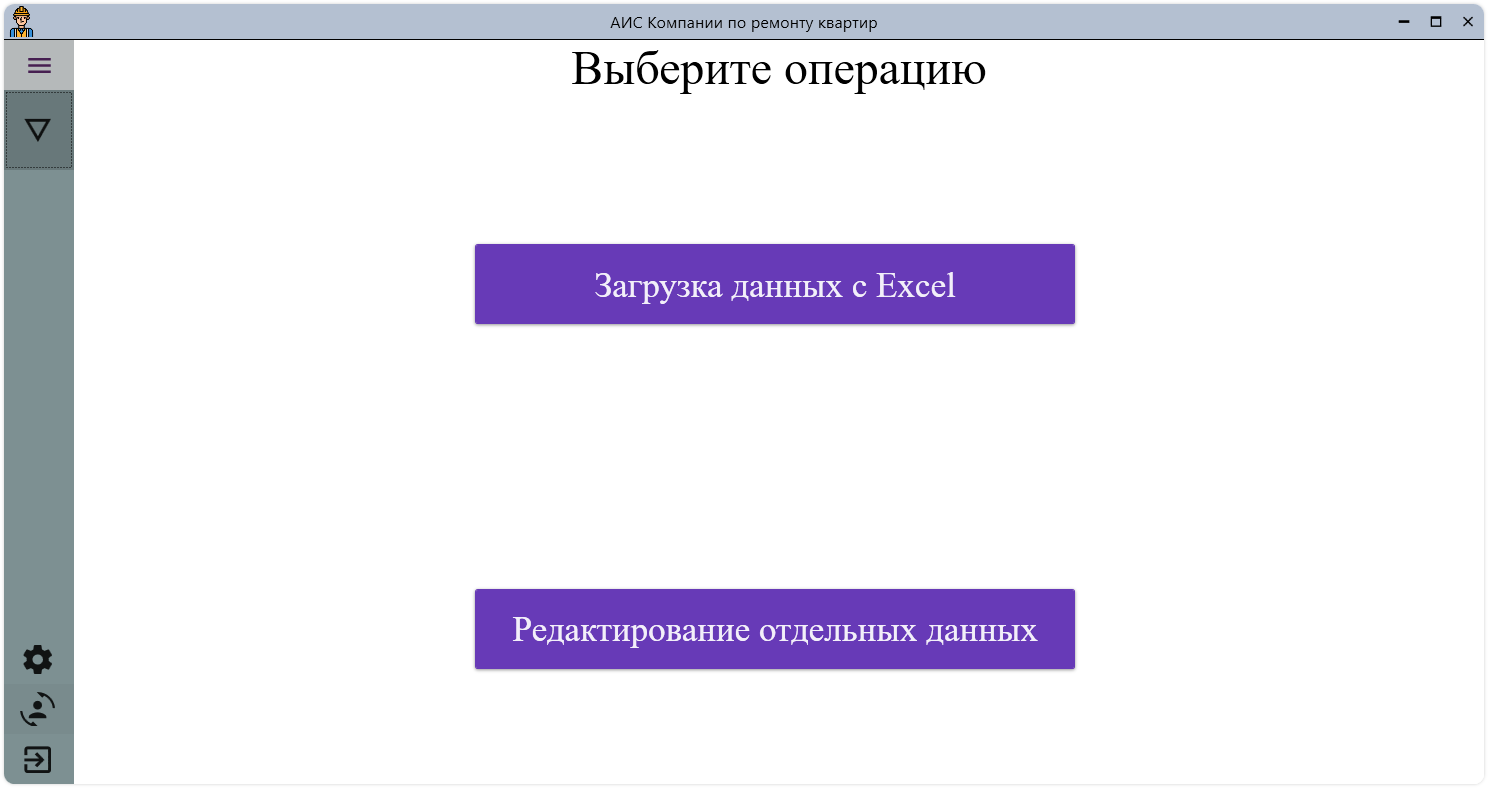


Рис. – Справочные данные

Далее рассмотрим роль менеджера. Данная роль предусматривает возможность работы:

* с данными клиентов;
* с данными о заказах;
* со справочными данными.

После авторизации пользователя с правами доступа менеджера перед пользователем представляется окно, представленное на рисунке 62.

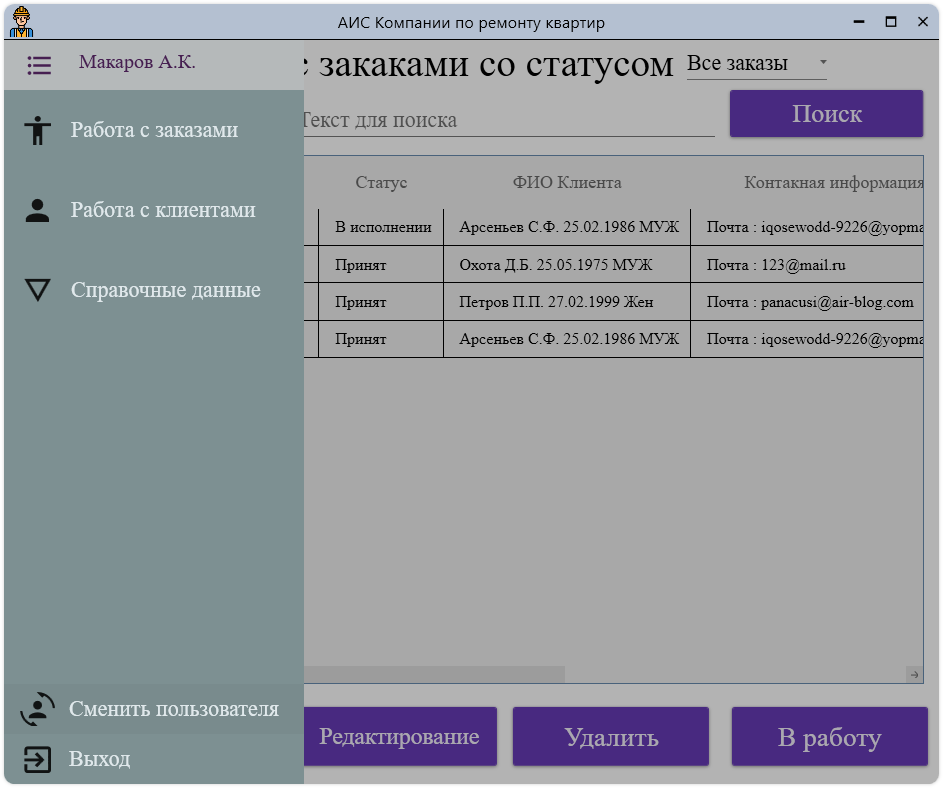


Рис. – Вид программы при авторизации с правами менеджера

При выборе пункта работа со справочными данными перед пользователем появляется окно, представленное на рисунке 63. При выборе пункта «Загрузка данных с Excel» перед пользователем появляется окно, представленное на рисунке 64, в котором необходимо выбрать тип данных для загрузки. После выбора пунктов пользователь должен выбрать файл для загрузки, после чего произойдет считывание данных и представление этих данных перед пользователем. Пример считывания данных представлен на рисунке 65. Пример исходного файла представлен на рисунке 66.

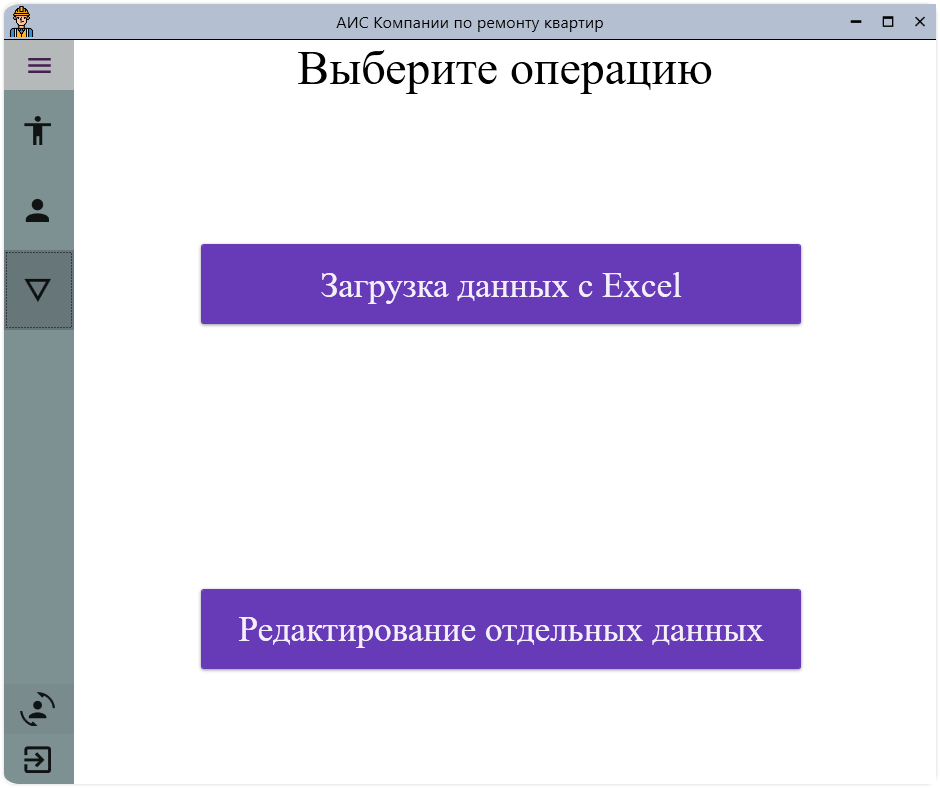


Рис. – Окно справочные данные

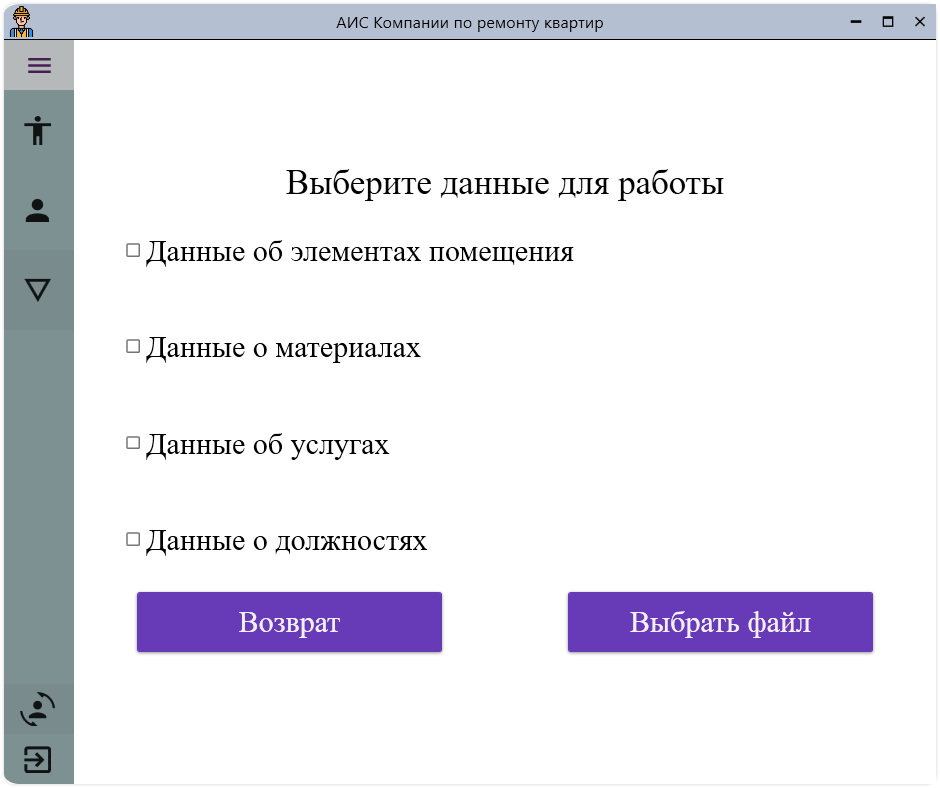


Рис. – Окно выбора типа справочных данных

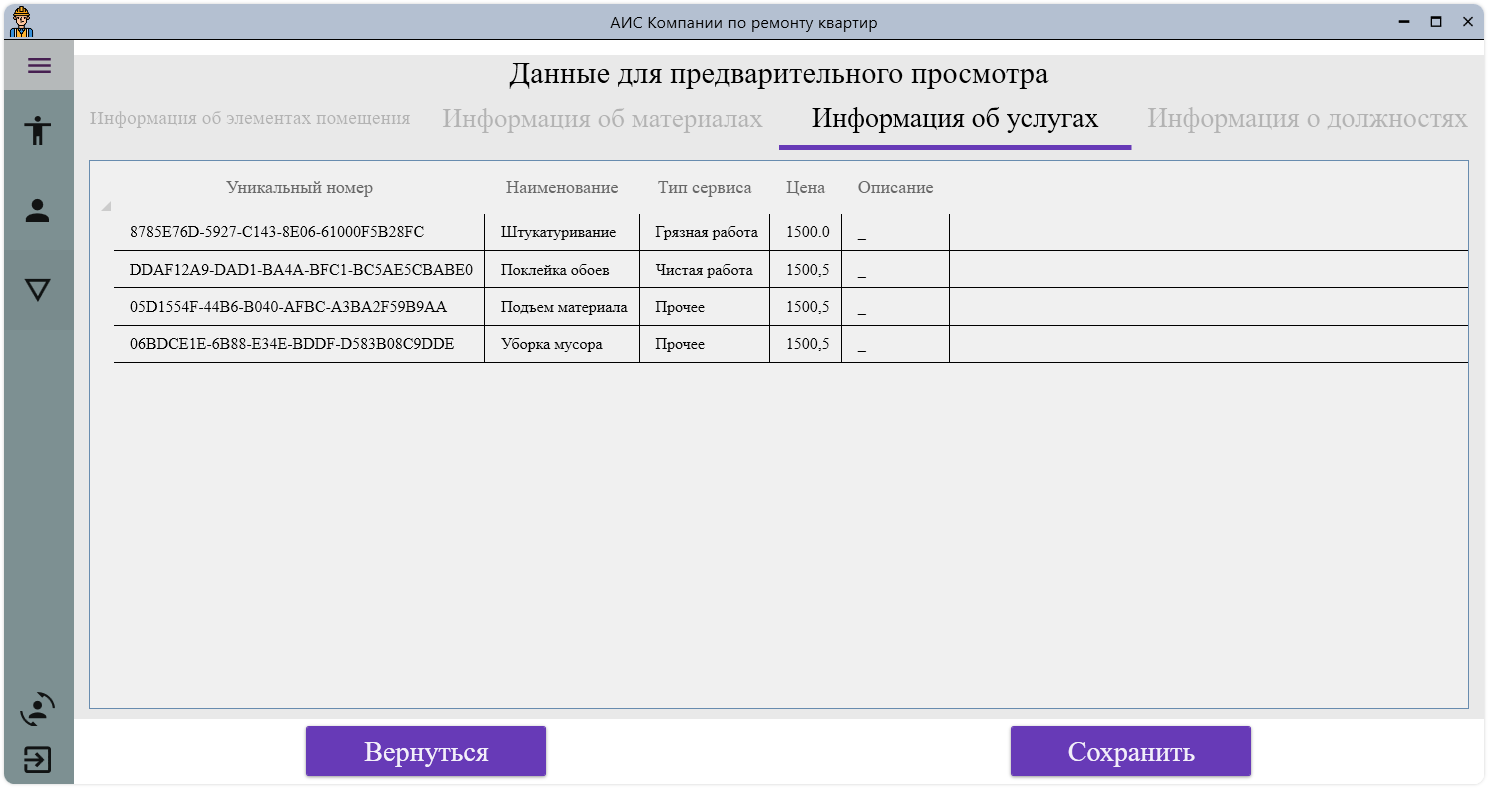


Рис. – Окно предварительного просмотра данных

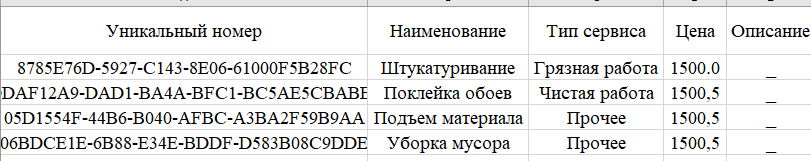


Рис. – Исходный файл Excel

После проверки данных на клиенте их можно загрузить на сервер с помощью кнопки «Сохранить», после чего появиться окно загрузки аналогичное окну при старте программы.

При необходимости работы с клиентами можно перейти в соответствующий пункт бокового меню «Работа с клиентами». Пример отображения данного пункта меню представлен на рисунке 67

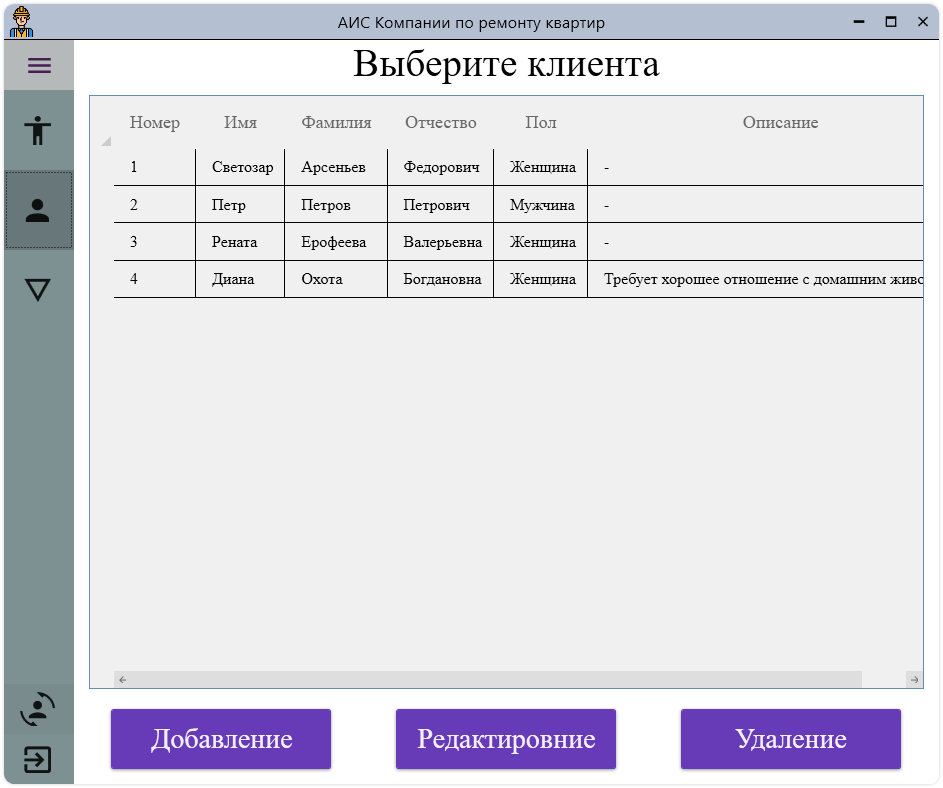


Рис. – Пункт работы с клиентами

С помощью данного меню пользователь может работать с данными о клиентах.

Пример формы создания нового клиента представлен на рисунке 68.

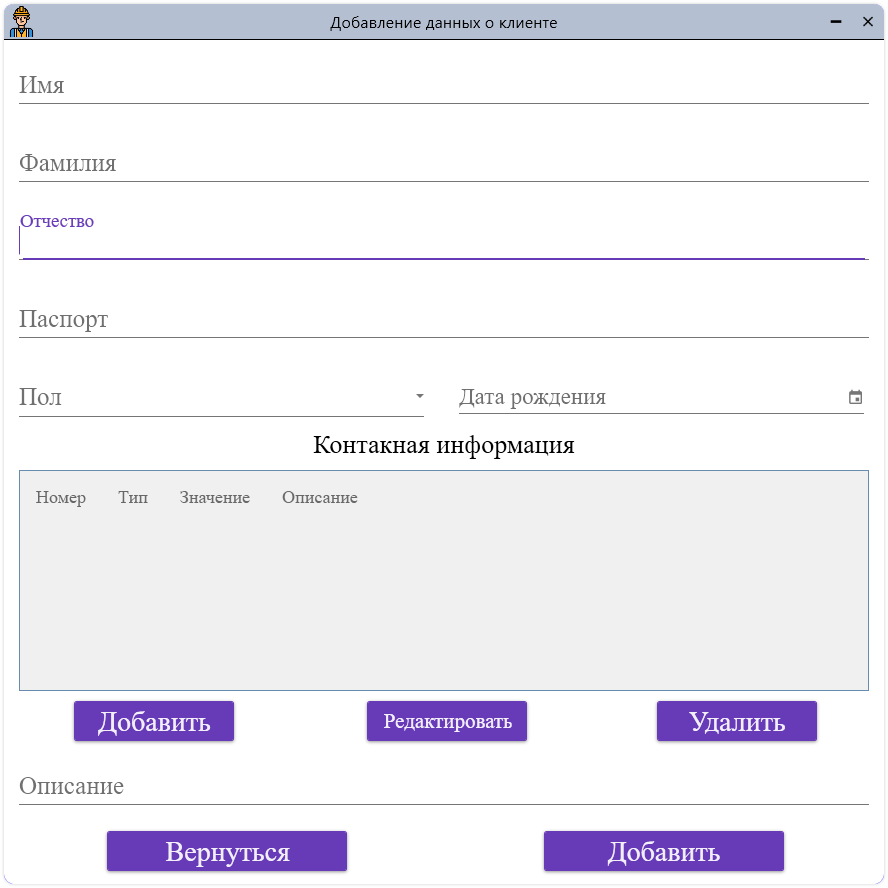


Рис. – Пример окна по созданию нового пользователя

Для редактирования данных о клиенте необходимо выбрать данные о клиенте из таблицы на рисунке 62, после чего нажать кнопку редактировать и появиться окно аналогичное окну, представленному на рисунке 63, только с данными клиента для редактирования. Пример отображения окна для редактирования, представлен на рисунке 69.

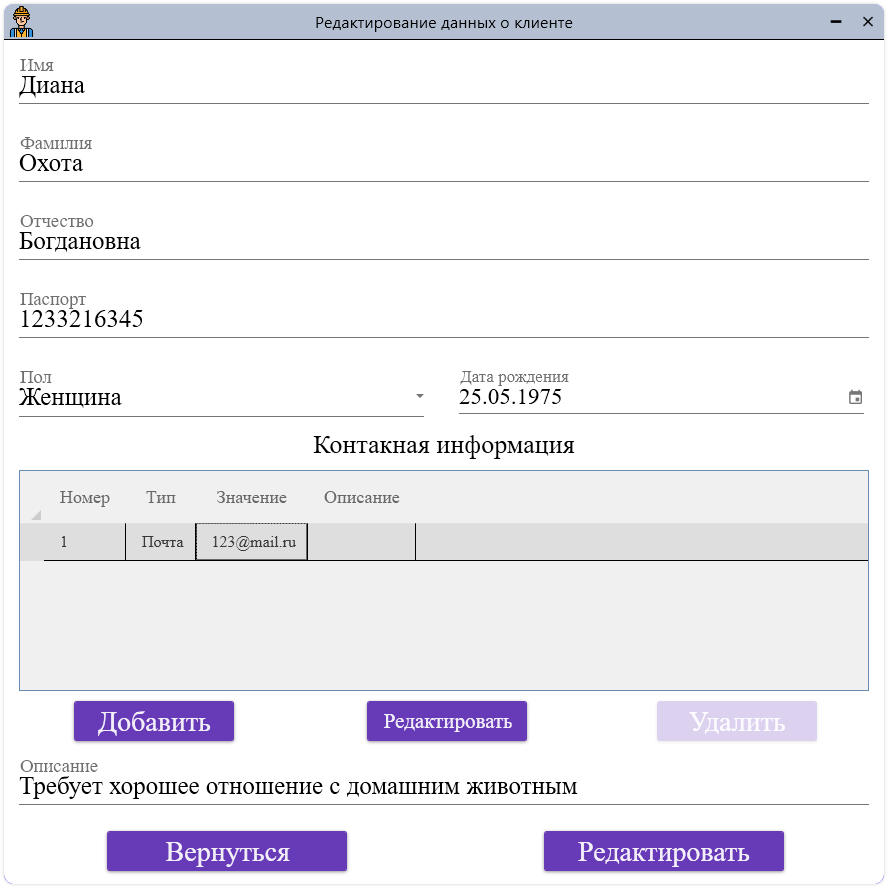


Рис. – Пример окна по редактированию пользователя

При указании данных о пользователях контактах необходимо вводить корректные данные. Например, на рисунке 70 продемонстрирована почта без специального значка собаки. Выдается сообщение о некорректности данных.

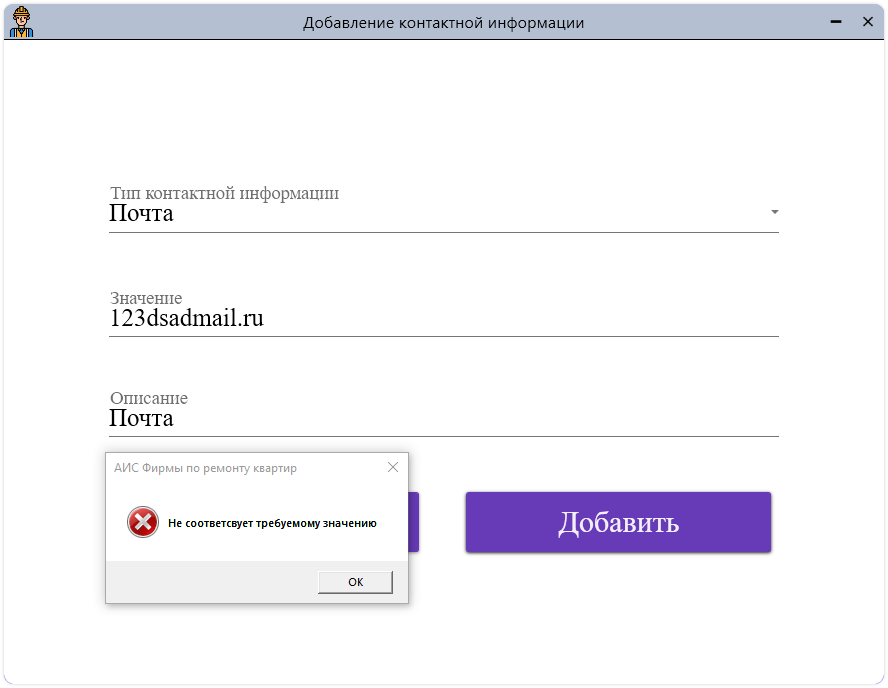


Рис. – Пример окна по указанию контактной информации

После указания корректной информации о типе контактов он добавится в информацию о контактах клиента, которого необходимо отредактировать. Пример представлен на рисунке 71.

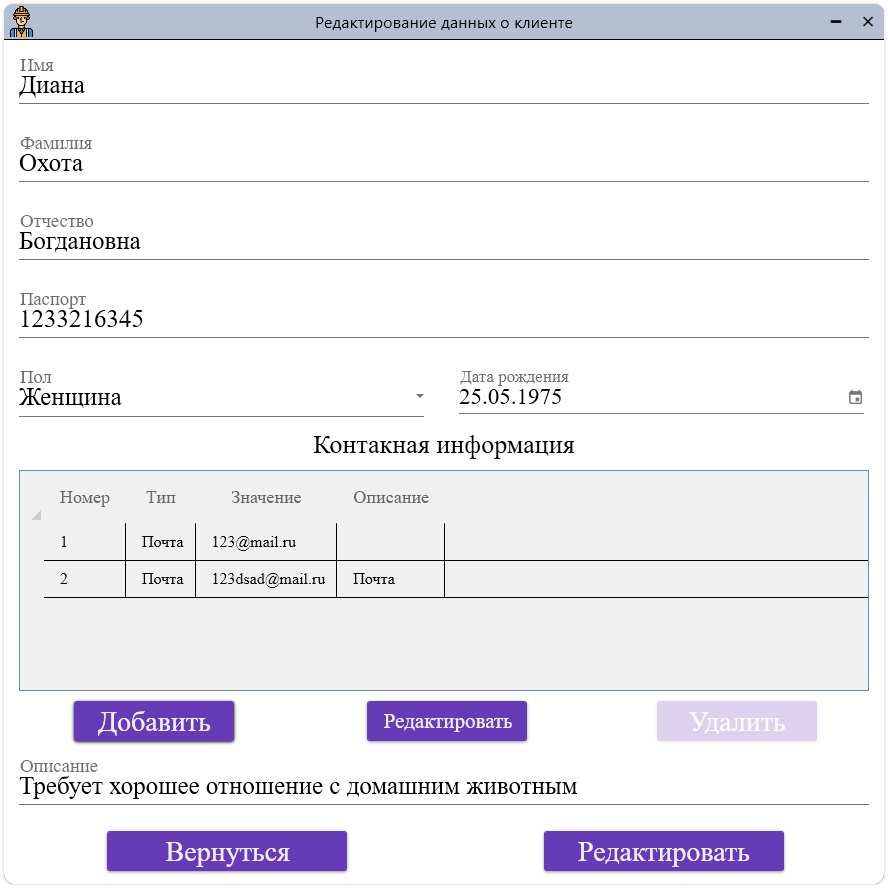


Рис. – Информация о данном контакте добавлена

После подтверждения факта редактирования появляется сообщение, представленное на рисунке 72.

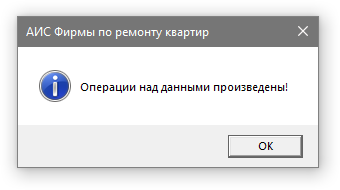


Рис. – Сообщение об обновление информации о клиенте

Для создания заказа необходимо перейти в пункт «Работа с заказами» бокового меню, представленного на рисунке 62 в пунктах, доступных данному типу работников.

Перед пользователем представляется окно, представленное на рисунке 73, на котором продемонстрированы некоторые возможности для поиска и сортировки.

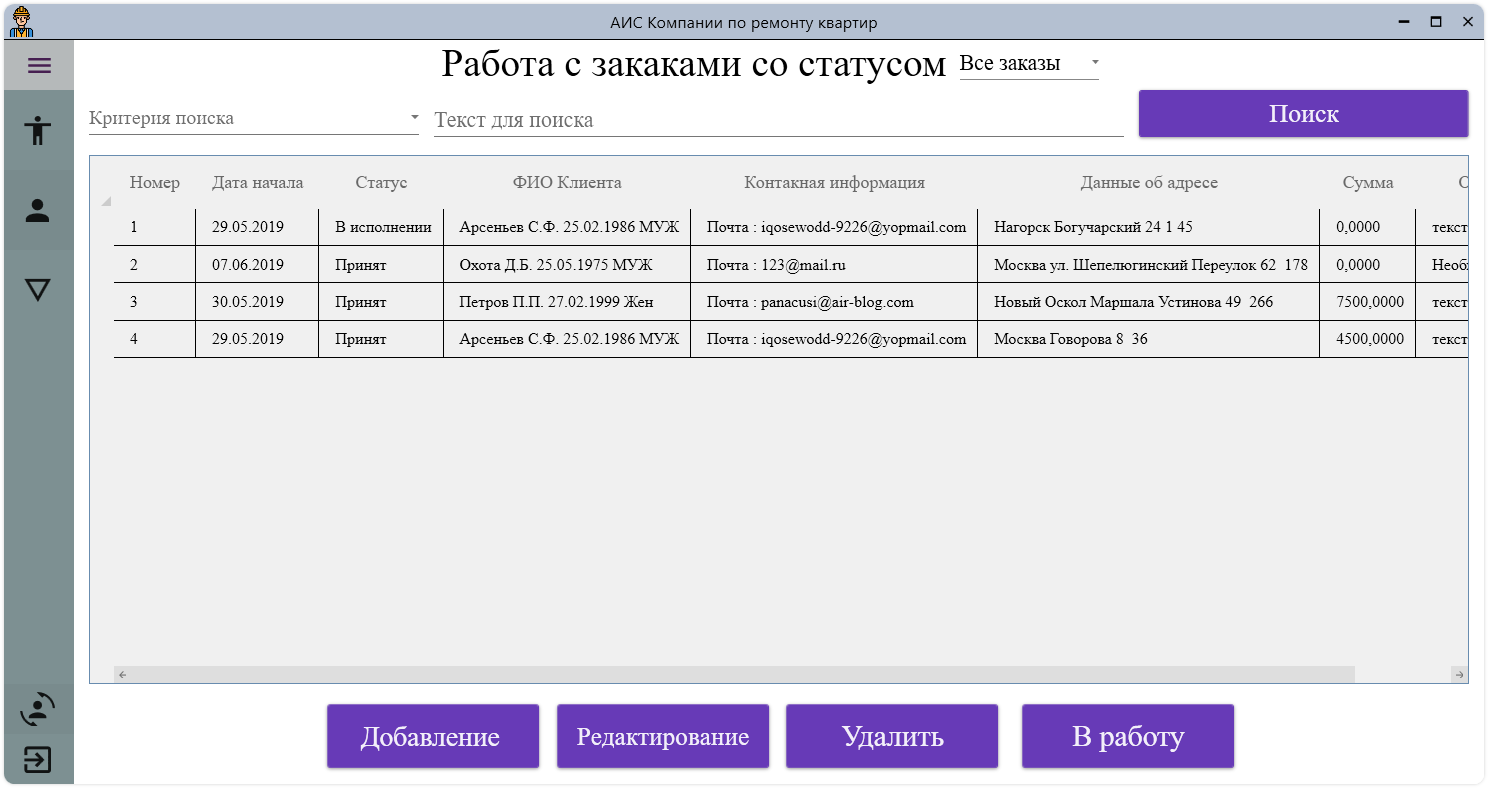


Рис. – Окно выбора заказа для дальнейшей работы

Для сортировки в данном окне можно использовать поиск и сортировку по статусу заказа.

Для создания заказа необходимо нажать на кнопку «Добавление», после чего откроется окно, представленное на рисунке 74.

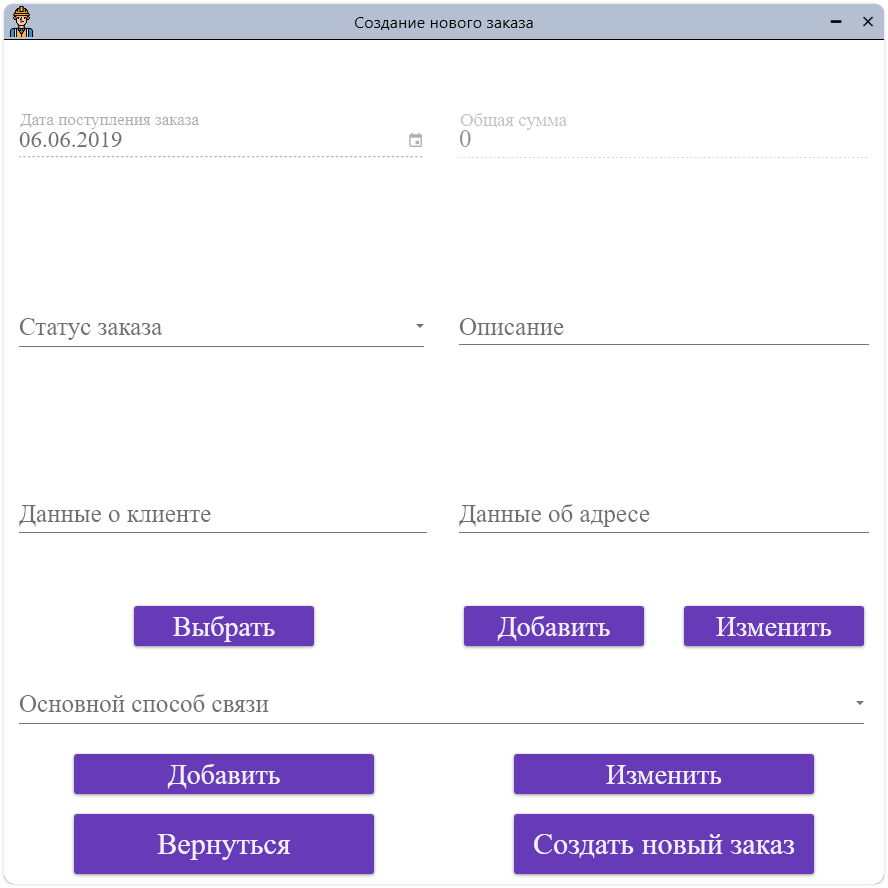


Рис. – Окно созданию заказа

Пример редактирования данных о заказе представлен на рисунке 75.

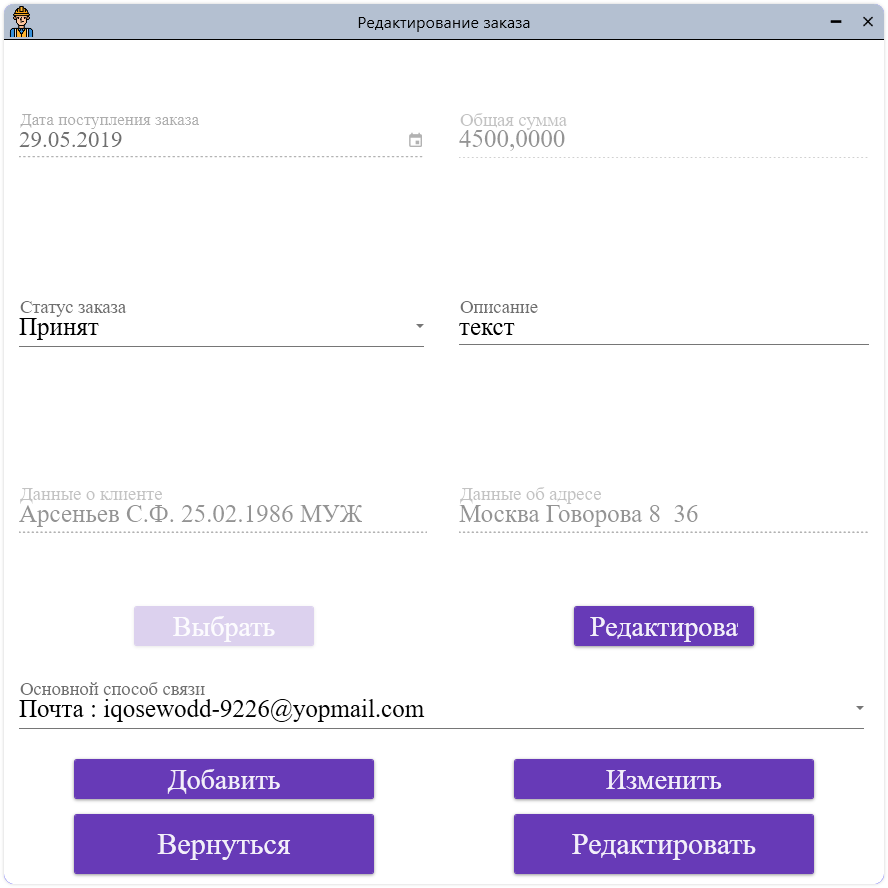


Рис. – Окно редактирования заказа

Для выбора заказа, для работы необходимо выбрать заказ из таблицы, после чего нажать на пункт «В работу», после чего появиться окно, представленное на рисунке 76, на котором отображены основные данные о заказе, а также данные о необходимости оплаты.

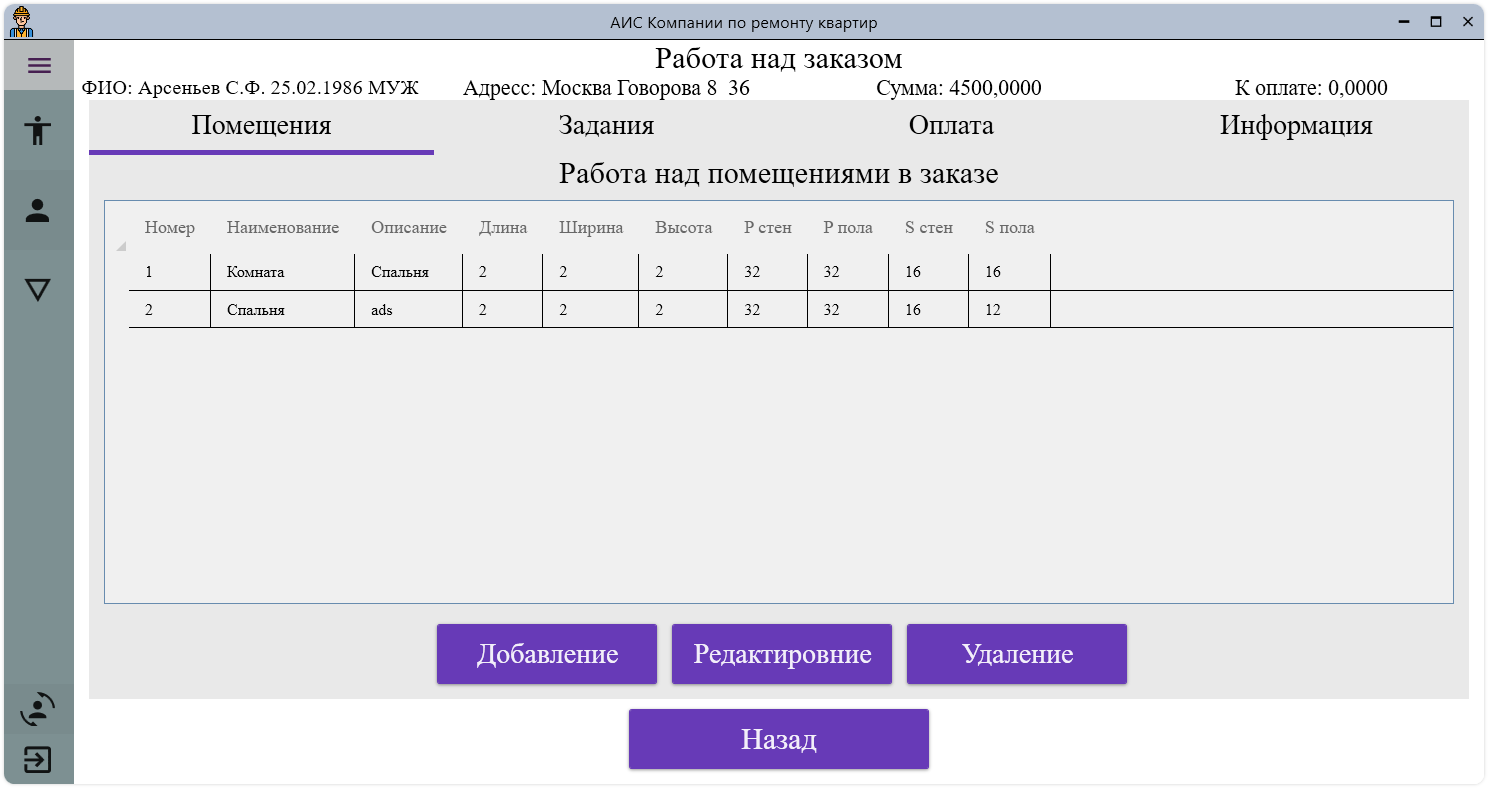


Рис. – Окно работы с заказами

При необходимости добавлении данных о помещениях необходимо перейти на вкладку «Помещения», после чего нажать на кнопку «Добавить» и откроется, окно, представленное, на рисунке 77. Также есть возможность добавления данных об элементах помещений, информацию о которых можно добавить путем нажатия кнопки «Добавить». После чего можно указать данные об элементах помещений, пример окна представлен на рисунке 78.

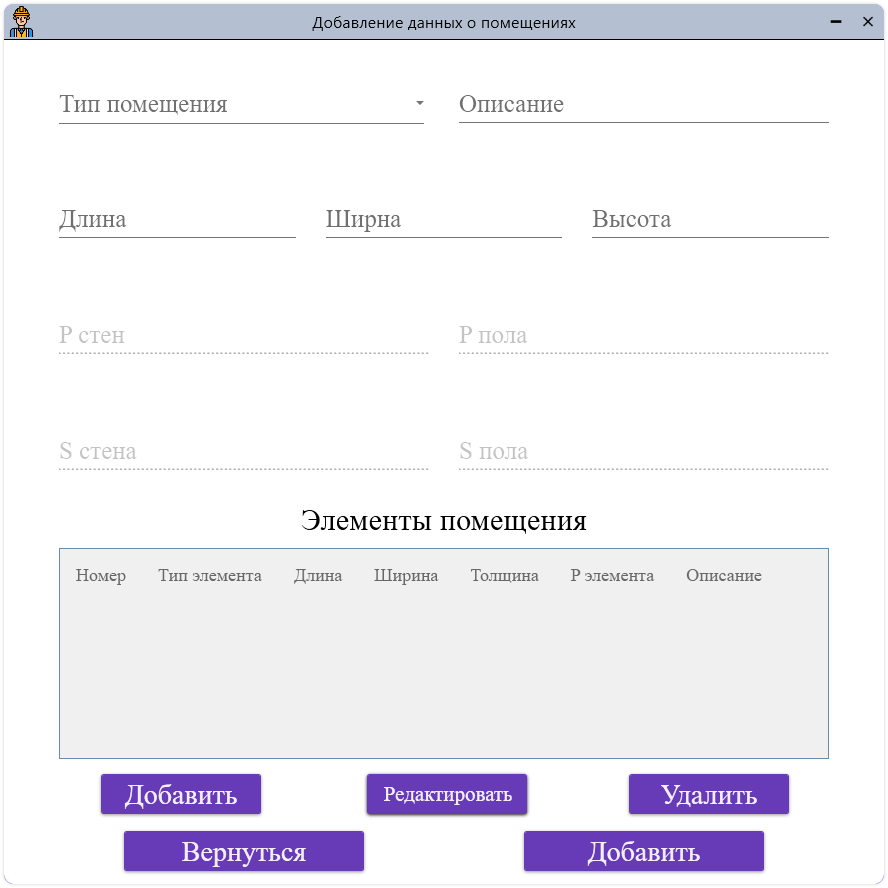


Рис. – Окно создания помещения

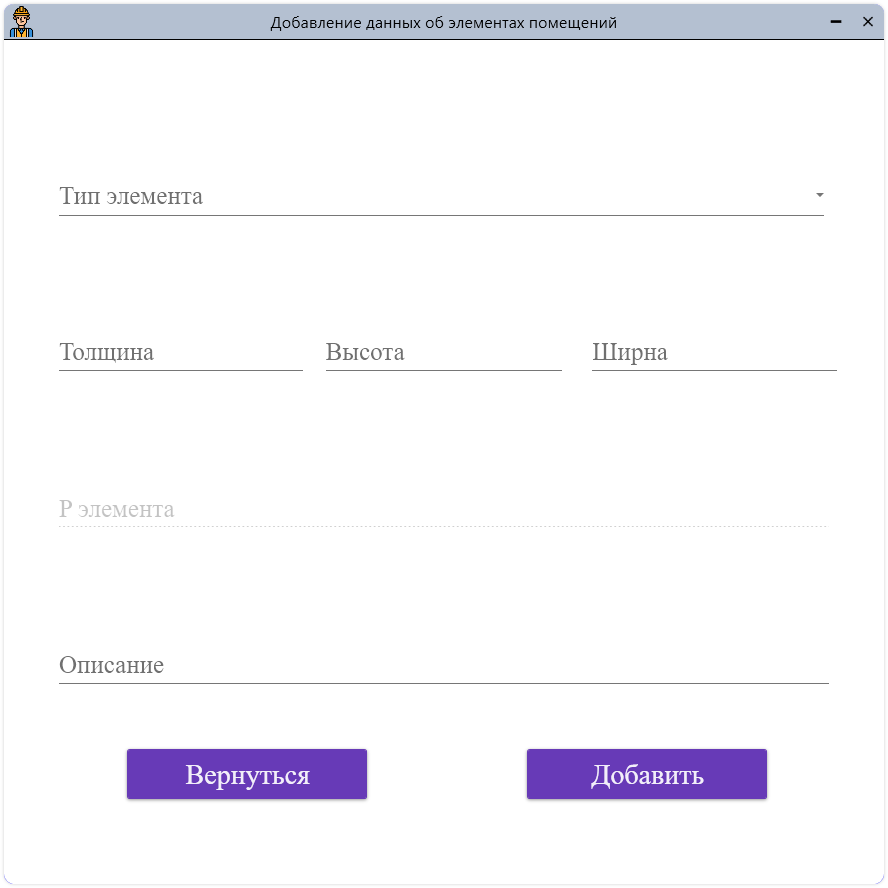


Рис. – Окно создания элемента помещения

Пример заполнения данных о помещениях, можно посмотреть на рисунке 79, на котором отображена информация о редактируемом помещении.

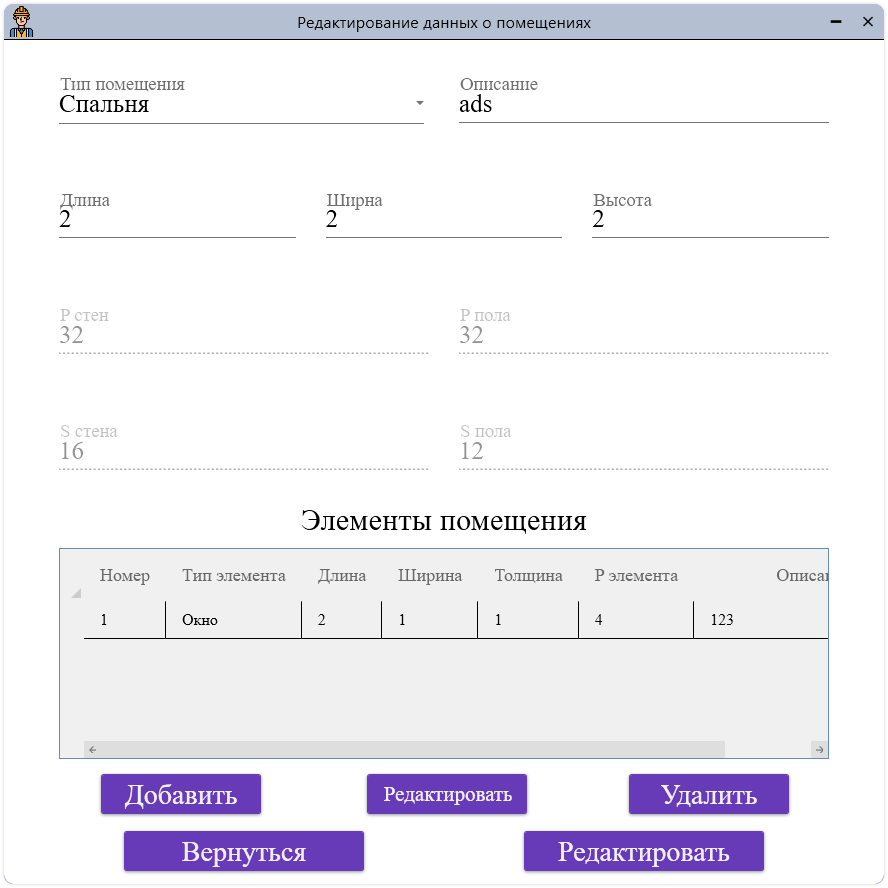


Рис. – Пример заполнения данных о помещении

Основной информацией о заказе в данной конфигурации является функция создания заданий, которую можно создать на вкладке задания. Пример отображения данной вкладки представлен на рисунке 80.

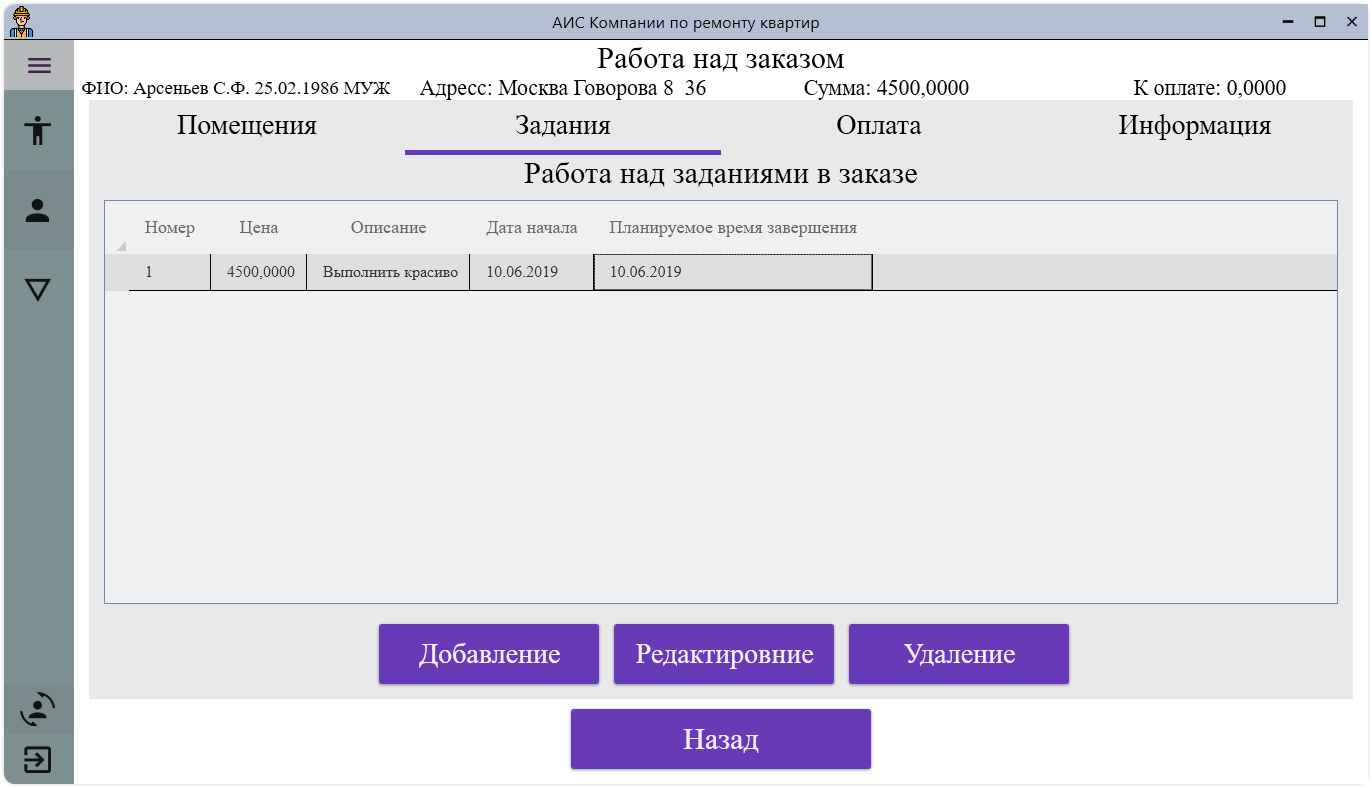


Рис. – Вкладка создания заказа

Для создания заказа необходимо нажать на кнопку «Добавление», после чего откроется окно создания задания, представленного на рисунке 81. Для примера приведу информацию с уже созданного задания.

Для задания можно задать информацию об услуге перейдя на вкладку «Информация об услугах» представленной на рисунке 82, после чего нажать на кнопку «Добавления», затем откроется окно создания информации об услугах, представленное на рисунке 83.

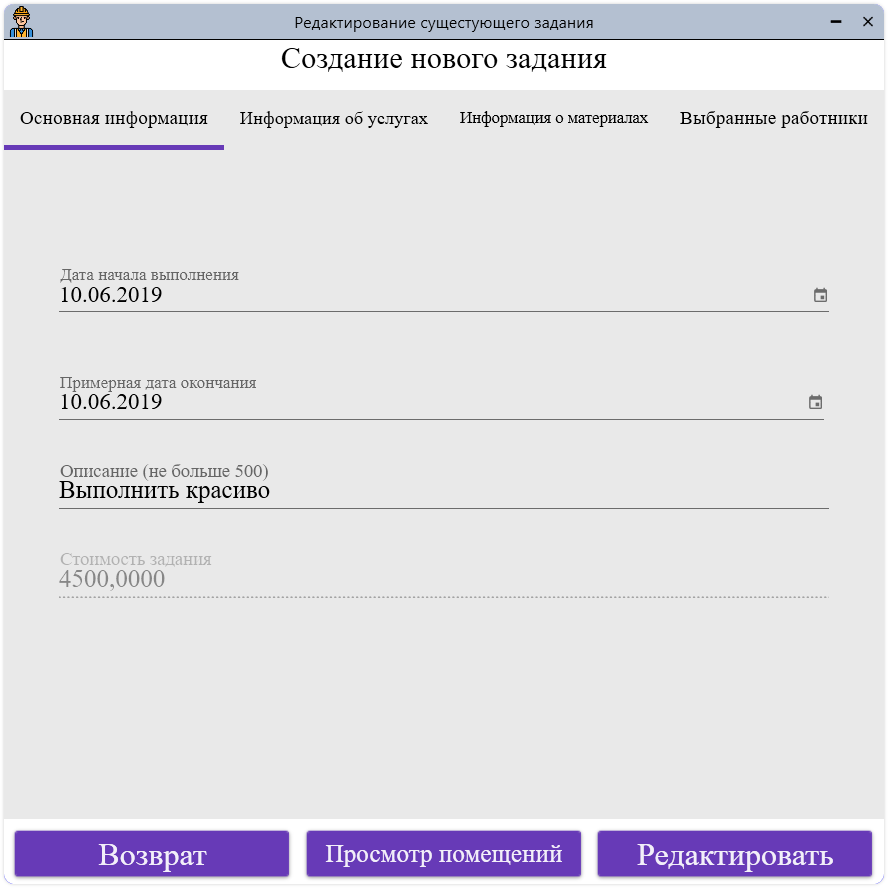


Рис. – Задание об основной информации о заданиях

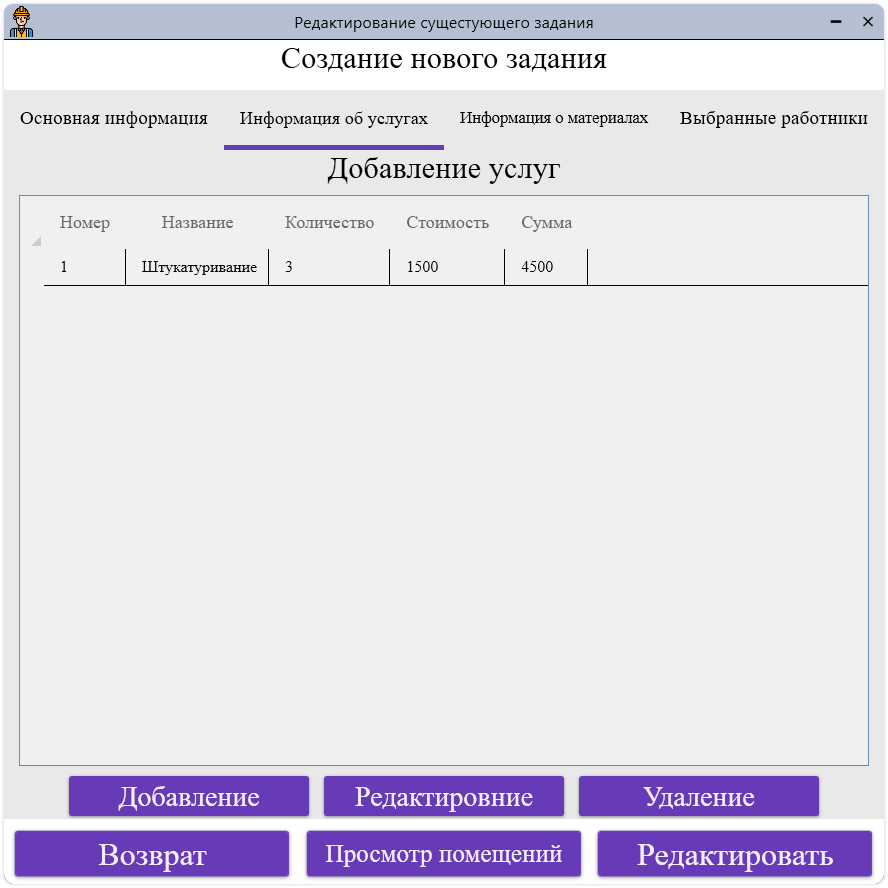


Рис. – Вкладка создания информации об услугах

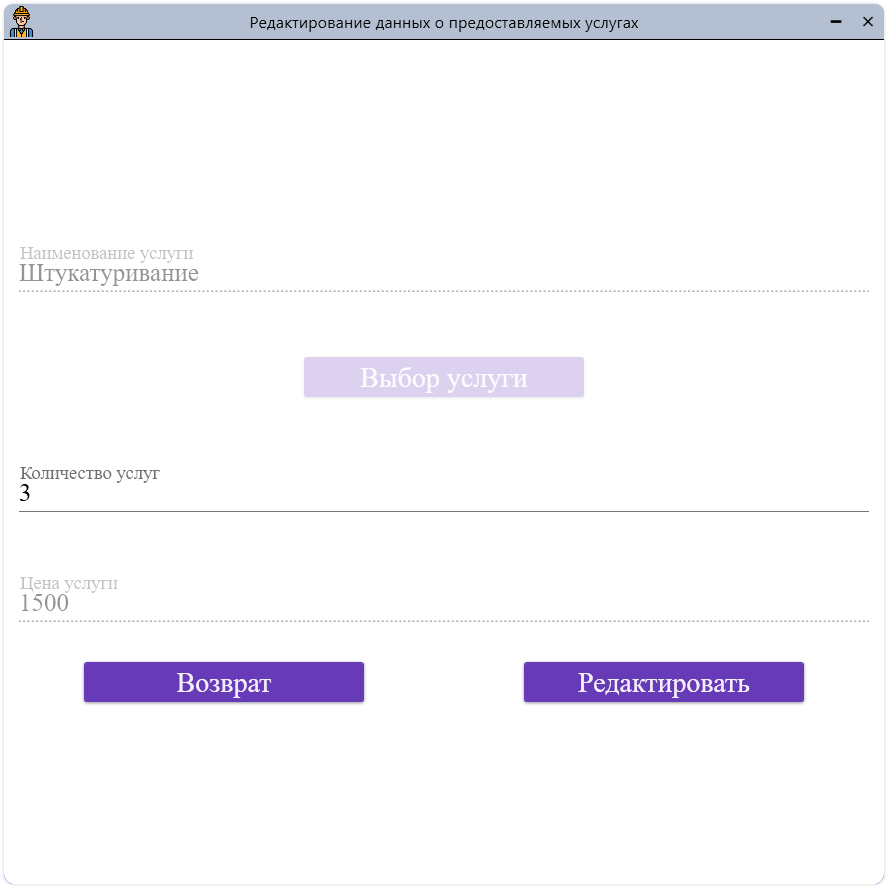


Рис. – Создание новой информации об услугах

Также для задания можно указать информацию о материалах перейдя на вкладку «Информация о материалах», пример отображения которой представлен на рисунке 84. Для добавления необходимо нажать на кнопку «Добавление» после чего откроется окно создания информации о материалах для задания, пример которого представлен на рисунке 85.

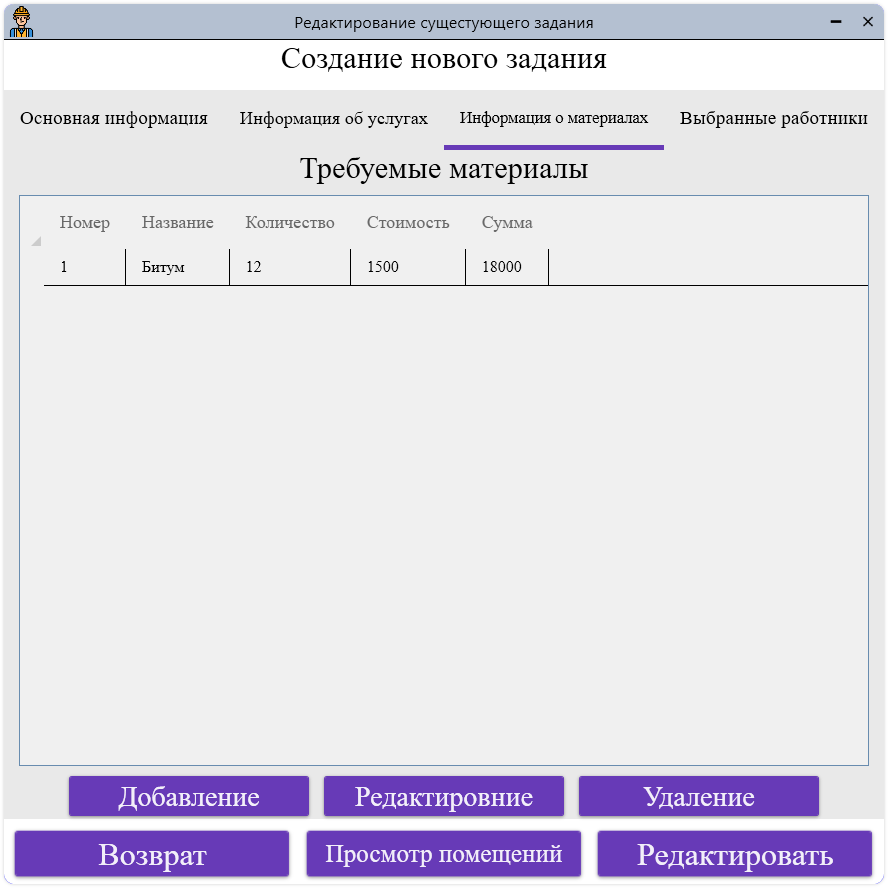


Рис. – Информация о материалах в задании

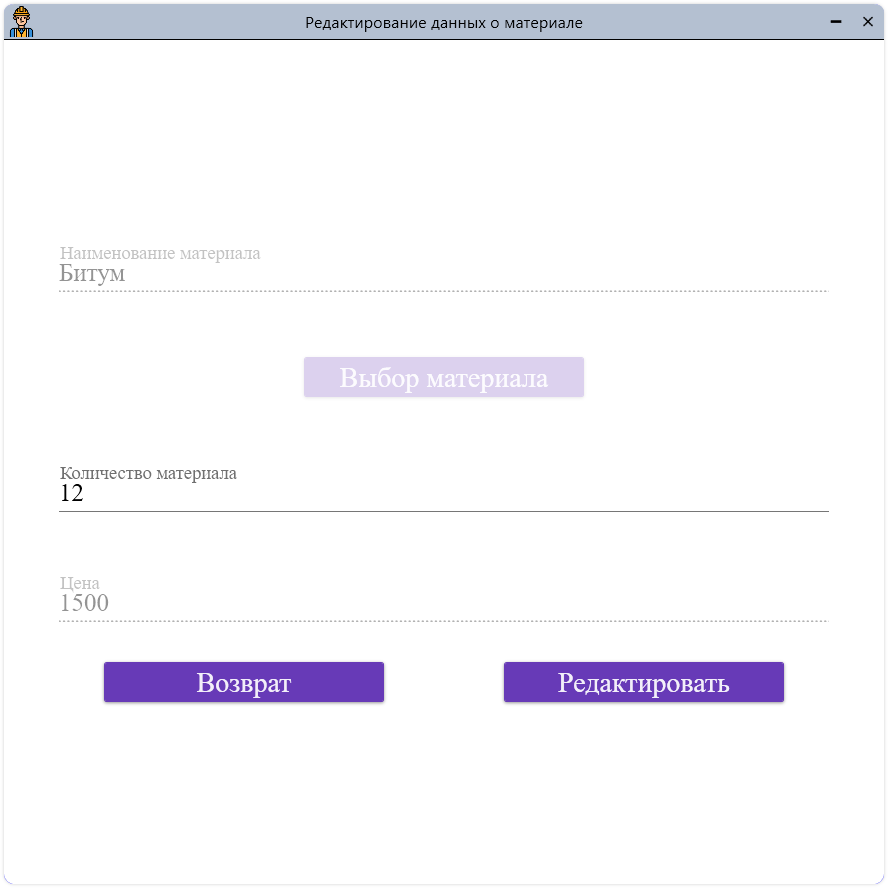


Рис. – Создание информации о материалах

Для каждого задания необходимо указывать информацию о работниках, которые будут отвечать за заказ. Данную информацию можно добавить на вкладке «Выбранные работники», пример отображения которой представлен на рисунке 86, пример добавления данных о работниках в задание представлен на рисунке 87.

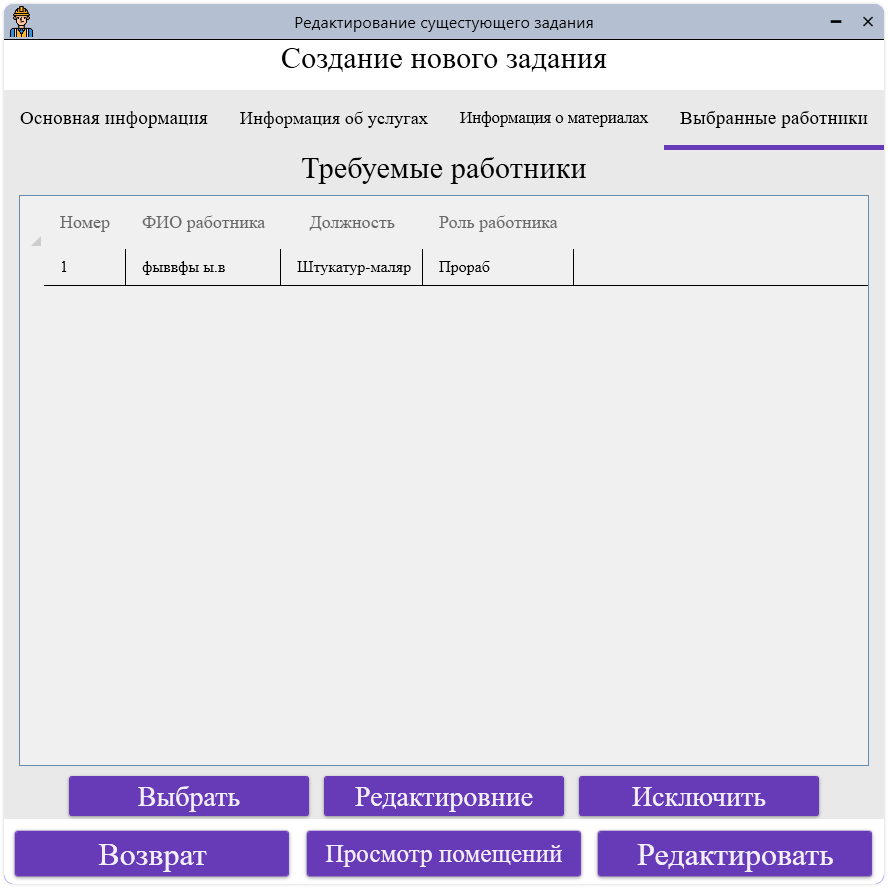


Рис. – Вкладка «Выбранные работники»

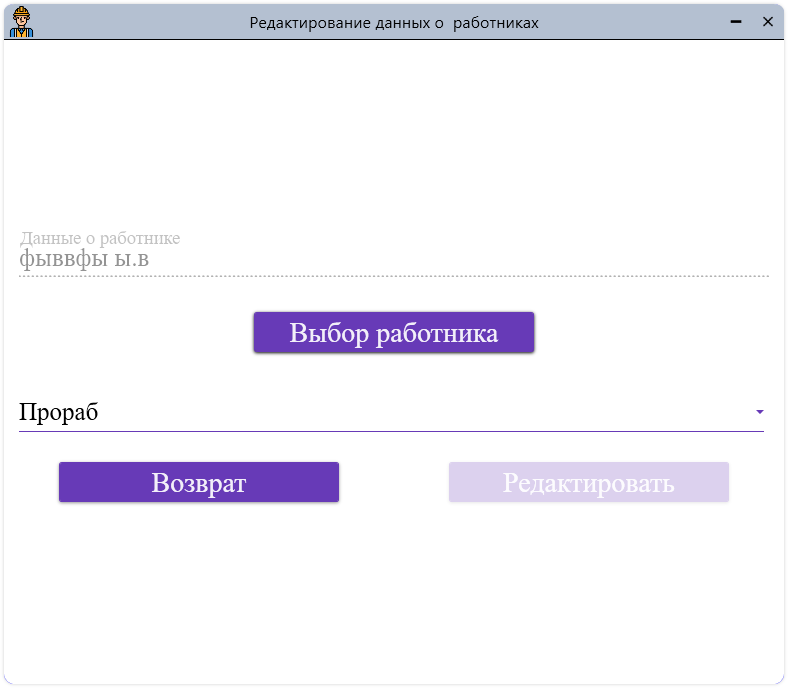


Рис. – Выбор нового работника

Вкладка «Оплата» при создании информации о заказе, представленная на рисунке 88, предоставляет возможность добавления данных об оплате. После создания информации об оплате форма создания, которой представлена на рисунке 89 можно напечатать квитанцию, пример создания которой представлен на рисунке 90.

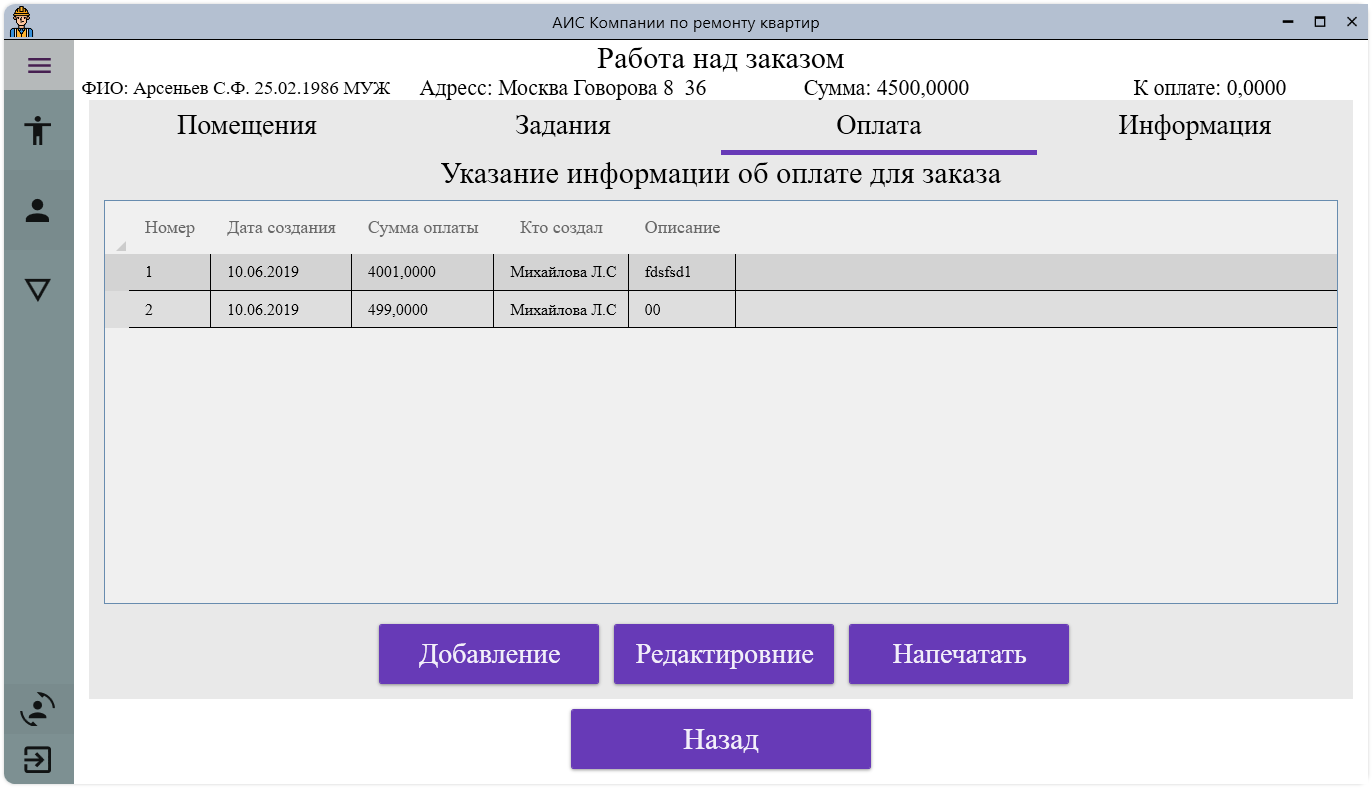


Рис. – Вкладка создания информации об оплате

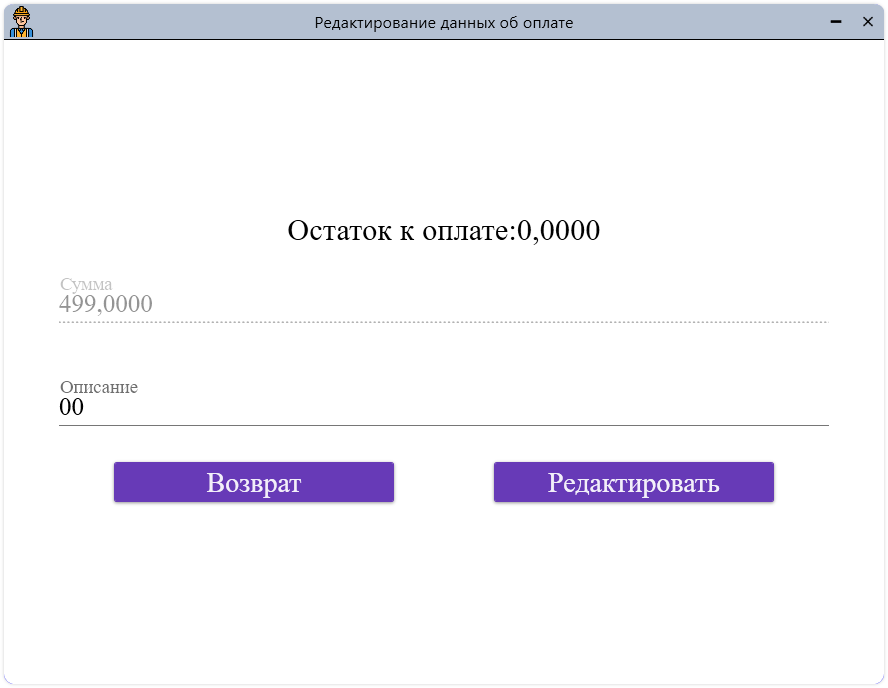


Рис. – Форма создания данных об оплате

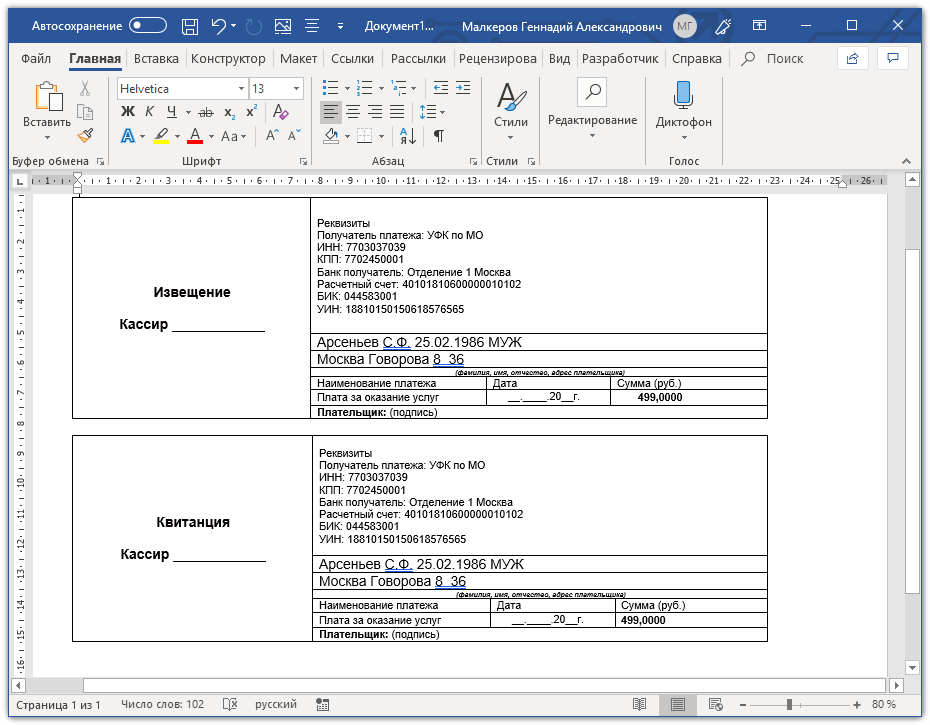


Рис. – Пример создания квитанции на основе данных об оплате

Для печати сметы и договора необходимо перейти на вкладку «Информация» пример отображения, которой представлен на рисунке 91. Пример создания каждого из документов представлен на рисунке 92-94.



Рис. – Вкладка «Информация» при работе с заказами

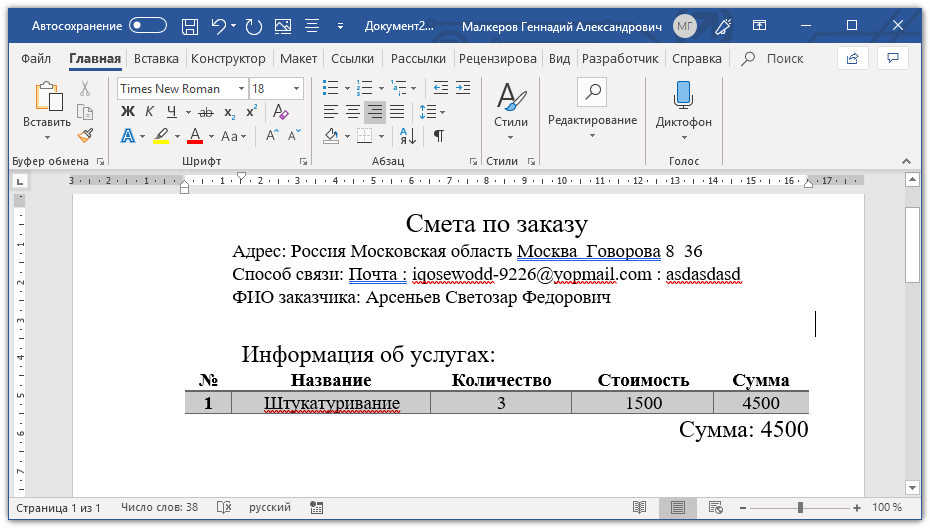


Рис. – Документ «Смета по заказу»

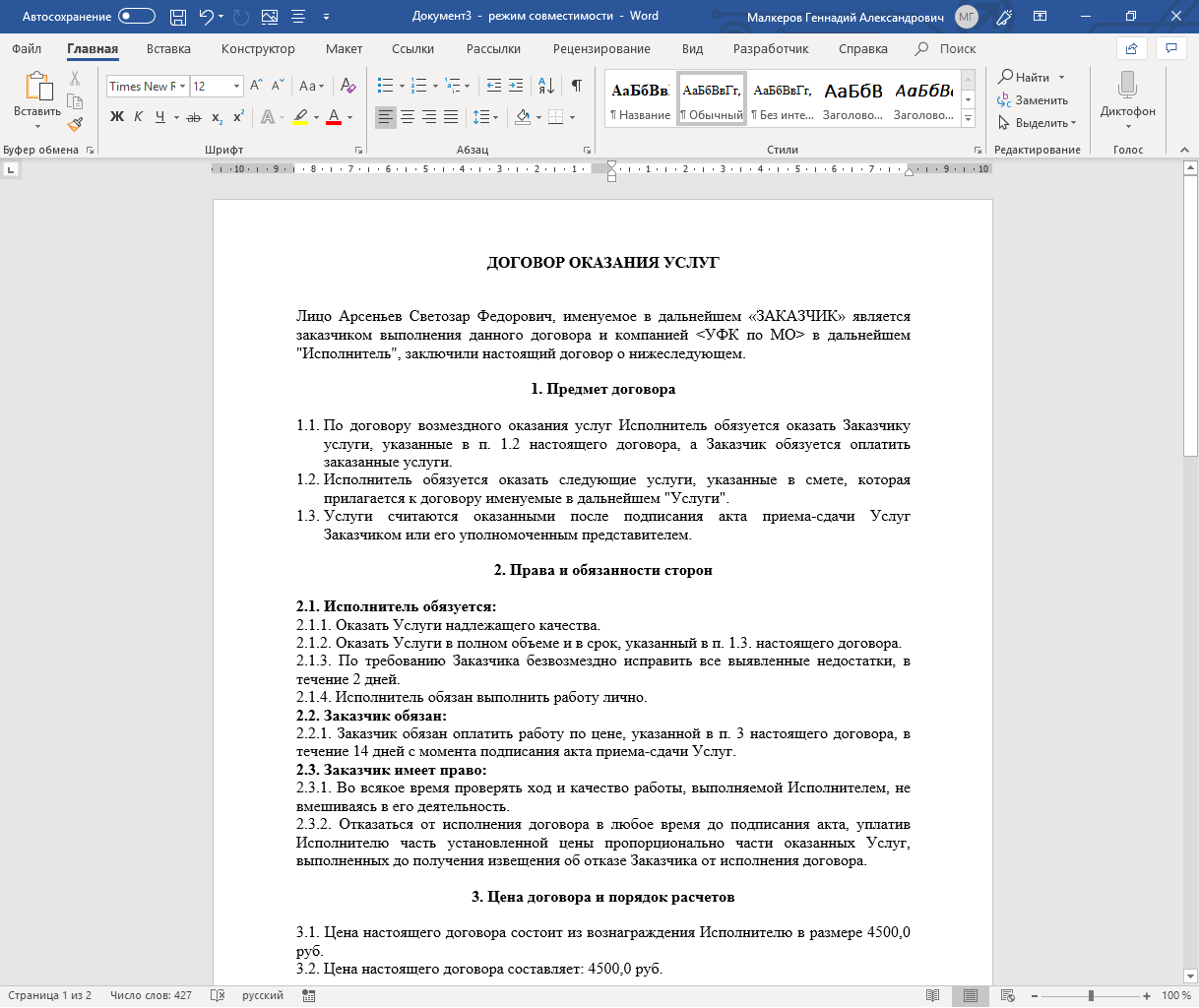


Рис. – Документ «Договор по заказу»

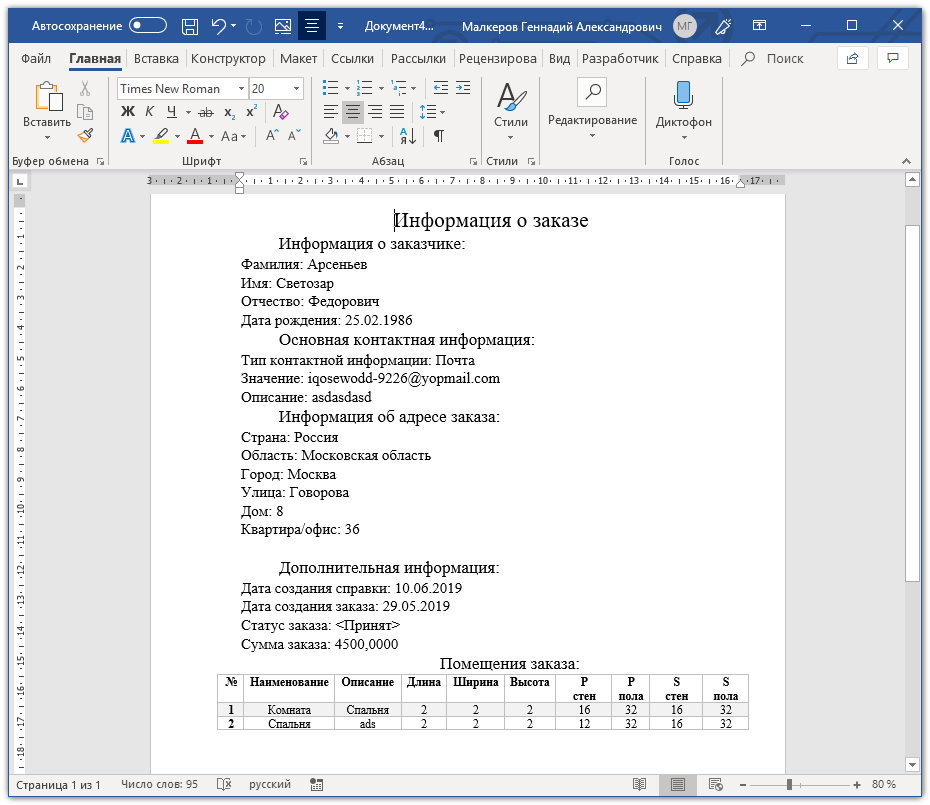


Рис. – Документ «Справочная информация о заказе»

#### 2.3.3.4 Сообщения пользователю

Пример сообщения пользователю представлен на рисунке 95-100.

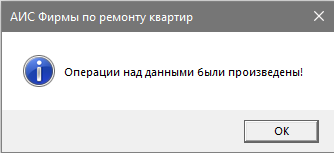


Рис. – Сообщение пользователю

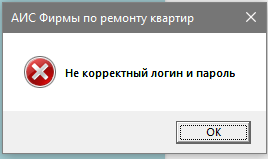


Рис. – Сообщение пользователю

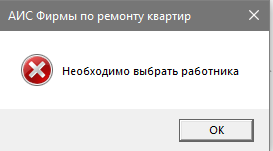


Рис. – Сообщение пользователю

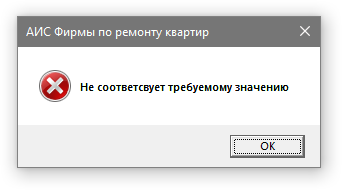


Рис. – Сообщение пользователю



Рис. – Сообщение пользователю

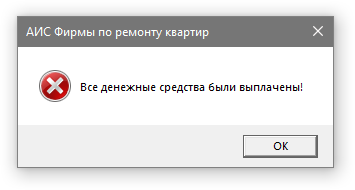


Рис. – Сообщение пользователю

# Заключение

В ходе разработки выпускной квалификационной работы было написано клиент-серверное приложение «Автоматизированная информационная система фирмы по ремонту квартир». При подготовке к написанию выпускной квалификационной работы была изучена предметная область разрабатываемого проекта, а также изучена специализированная литература по работе клиент-серверной схемы работы программного обеспечения на основе технологии Rest API.

Были составлены схемы баз данных, которые используются в качестве способа хранения информации. При разработке баз данных были использованы специализированные средства, которые также были изучены.

После составления схем баз данных была начата разработка серверной части продукта, которая использует основную базу данных.

Параллельно с разработкой серверной части было начато создание интерфейсов клиентского приложения, с которым взаимодействует пользователь.

Вместе с разработкой было выполнено тестирование программного продукта, что позволило выявить большое количество недостатков и недочетов, которые были исправлены.

Среди достоинств программы можно отметить следующие:

* удобный и понятный интерфейс;
* настройка добавленных опций в заказ;
* вывод отчетов в Word;
* импорт данных пользователя из Excel;
* простота дальнейшего расширения данной программы;
* клиент-серверная технология работы программы.

Тесты данной программы показали достаточную надежность и корректность работы программы, что позволяет использовать данное программно обеспечение в компании, занимающейся ремонтом квартир.

Дальнейшая доработка программы может заключаться в добавлении в данный программный комплекс мобильного приложения, которое поможет ускорить работу на объектах, а также увеличить степень взаимодействия с клиентами.

# Список литературы

Стандарты

1. ГОСТ 2.316-2008 Правила нанесения надписей, технических требований и таблиц на графических документах.
2. ГОСТ 7.1. – 2003. Библиографическая запись. Библиографическое описание. Общие требования и правила составления. – М.: ИПК Издательство стандартов, 2004. – 169 с.
3. ГОСТ 7.1-2003 Библиографическая запись. Библиографическое описание.
4. ГОСТ 7.32 – 2017. Система стандартов по информацию, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления. – М.: Москва СтандартИнформ, 2017. – 27 с.
5. ГОСТ\_2.105-95 Общие требования к текстовым документам
6. Единая система программной документации. – М.: Стандартинформ, 2005. – 128 с.

Интернет-ресурсы

1. Рынок ремонтно-строительных работ: тенденции и прогнозы – [Электронный ресурс]. – Режим доступа: https://www.vira.ru/exp/news/detail.php?ID=48005
2. Простой Софт - разработка программ– [Электронный ресурс]. – Режим доступа: http://www.prostoysoft.ru/index.htm
3. Конфигурация "Ремонт квартир" – [Электронный ресурс]. – Режим доступа: http://www.prostoysoft.ru/FlatRepair.htm
4. Draw.io – [Электронный ресурс]. – Режим доступа: https://www.draw.io/
5. DB Browser for SQLite – [Электронный ресурс]. – Режим доступа: https://sqlitebrowser.org/
6. SQL Server 2017 — Выпуски – [Электронный ресурс]. – Режим доступа: https://www.microsoft.com/ru-ru/sql-server/sql-server-2017-editions
7. Скачивание SQL Server Management Studio (SSMS) – [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-2017
8. Описание среды программирования VS – [Электронный ресурс]. – Режим доступа: https://studfiles.net/preview/4114690/
9. Скачать приложение Postman – [Электронный ресурс]. – Режим доступа: https://www.getpostman.com/downloads/
10. C# Язык программирования – [Электронный ресурс]. – Режим доступа: http://progopedia.ru/language/csharp/
11. SQL Язык программирования – [Электронный ресурс]. – Режим доступа: http://progopedia.ru/language/sql/
12. Введение в Entity Framework 6– [Электронный ресурс]. – Режим доступа: https://metanit.com/sharp/entityframework/1.1.php

# Приложение А

Исходный код программы

Модуль RepairFlatRestApi. AdressController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Web;

using System.Web.Http;

namespace RepairFlatRestApi.Controllers.MainConroller

{

[RoutePrefix("api/Adress")]

public class AdressController:BaseController

{

[HttpPost, Route("create")]

public HttpResponseMessage CreateNewAdress([FromBody] Models.AdressModel.AdressDesc NewAdress)

{

return CatchError(() =>

{

return DBController.CreaNewAdress(NewAdress);

}, nameof(SubStringController), nameof(CreateNewAdress));

}

[HttpGet, Route("data")]

public HttpResponseMessage MakeDataAbout([FromUri] Guid idAdress)

{

return CatchError(() =>

{

return DBController.GetDataAboutContact(idAdress);

}, nameof(SubStringController), nameof(CreateNewAdress));

}

[HttpPost, Route("update")]

public HttpResponseMessage UpdateAdress([FromBody] Models.AdressModel.AdressDesc UpdatedAdress)

{

return CatchError(() =>

{

return DBController.UpdateDataAboutAdress(UpdatedAdress);

}, nameof(SubStringController), nameof(CreateNewAdress));

}

}

}

Модуль RepairFlatRestApi. BaseController.cs

using Newtonsoft.Json;

using RepairFlatRestApi.Models.DescriptionJSON;

using System;

using System.Net.Http;

using System.Text;

using System.Web.Http;

namespace RepairFlatRestApi.Controllers

{

/// <summary>

/// Базовый класс для работы контроллеров

/// </summary>

public class BaseController : ApiController

{

protected HttpResponseMessage CatchError(Func<object> action, string NameOfClass, string nameOfMethod)

{

try

{

var resp = action();

var response = new HttpResponseMessage();

response.Content = new StringContent(JsonConvert.SerializeObject(resp), Encoding.UTF8, "application/json");

return response;

}

catch (Exception ex)

{

Logger.WriteToLog(Logger.TypeOfRecord.Exception, NameOfClass, nameOfMethod, ex.ToString().Replace(Environment.NewLine, ""));

return new HttpResponseMessage()

{

Content = new StringContent(JsonConvert.SerializeObject(new BaseResult() { success = false, description = ex.ToString() }), System.Text.Encoding.UTF8, "application/json"),

};

}

}

}

}

Модуль RepairFlatRestApi. ContactController.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Web;

using System.Web.Http;

namespace RepairFlatRestApi.Controllers.MainConroller

{

[RoutePrefix("api/Contact")]

public class ContactController:BaseController

{

[HttpPost, Route("create")]

public HttpResponseMessage CreateNewContact([FromBody] Models.ContactModel.InformationAboutContact NewContact)

{

return CatchError(() =>

{

return OtherController.WorkWithContactDBController.CreaNewContact(NewContact);

}, nameof(SubStringController), nameof(CreateNewContact));

}

[HttpPost, Route("update")]

public HttpResponseMessage UpdateDataAboutContact([FromBody] Models.ContactModel.InformationAboutContact UpdateContact)

{

return CatchError(() =>

{

return OtherController.WorkWithContactDBController.UpdateContactData(UpdateContact);

}, nameof(SubStringController), nameof(UpdateDataAboutContact));

}

[HttpGet, Route("getusercontact")]

public HttpResponseMessage GetDataAboutContact([FromUri] Guid idUser)

{

return CatchError(() =>

{

return OtherController.WorkWithContactDBController.CreateListOfContactUser(idUser);

}, nameof(SubStringController), nameof(GetDataAboutContact));

}

[HttpGet, Route("delete")]

public HttpResponseMessage DeleteContact([FromUri] Guid idContact)

{

return CatchError(() =>

{

return OtherController.WorkWithContactDBController.DeleteContact(idContact);

}, nameof(SubStringController), nameof(GetDataAboutContact));

}

}

}

Модуль RepairFlatRestApi. MeasurmentController.cs

using RepairFlatRestApi.Controllers.OtherController;

using RepairFlatRestApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Web;

using System.Web.Http;

namespace RepairFlatRestApi.Controllers.MainConroller

{

[RoutePrefix("api/Measurment")]

public class MeasurmentController:BaseController

{

[HttpGet, Route("allmeastbl")]

public HttpResponseMessage SelectDataAbOrderByID([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OtherController.MeasurmentDBController.MakeDataAbouMeas(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderByID));

}

[HttpGet, Route("delete")]

public HttpResponseMessage DeletePremmises([FromUri] Guid idPremises)

{

return CatchError(() =>

{

return OrderDBController.DeletePremises(idPremises);

}, nameof(SubStringController), nameof(SelectDataAbOrderByID));

}

[HttpGet, Route("infbyid")]

public HttpResponseMessage SelectMainInfAbMeas([FromUri] Guid idMeas)

{

return CatchError(() =>

{

return OtherController.MeasurmentDBController.SelectDataAboutMeasurment(idMeas);

}, nameof(SubStringController), nameof(SelectDataAbOrderByID));

}

[HttpPost, Route("create")]

public HttpResponseMessage MakeDataAboutMeasurment([FromBody] MeasurmentModel.DataAboutMeassFromDB newMeas)

{

return CatchError(() =>

{

return OtherController.MeasurmentDBController.MakeNewDataAboutMeas(newMeas);

}, nameof(SubStringController), nameof(SelectDataAbOrderByID));

}

[HttpPost, Route("update")]

public HttpResponseMessage UpdateDataAboutMeas([FromBody] MeasurmentModel.DataAboutMeassFromDB updatedMeas)

{

return CatchError(() =>

{

return OtherController.MeasurmentDBController.UpdateDataAboutMeas(updatedMeas);

}, nameof(SubStringController), nameof(SelectDataAbOrderByID));

}

}

}

Модуль RepairFlatRestApi. OrderController.cs

using RepairFlatRestApi.Controllers.OtherController;

using RepairFlatRestApi.Models.DescriptionJSON;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Net.Http;

using System.Web;

using System.Web.Http;

namespace RepairFlatRestApi.Controllers.MainConroller

{

[RoutePrefix("api/Order")]

public class OrderController:BaseController

{

#region Контроллеры самого заказа

[HttpGet, Route("alldata")]

public HttpResponseMessage SelectDataAbOrderByID([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return DBController.SelectAllDataAboutOrder(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderByID));

}

[HttpGet, Route("allorders")]

public HttpResponseMessage SelectAllOrder()

{

return CatchError(() =>

{

return OtherController.OrderDBController.SelectDataAboutAllOrder();

}, nameof(SubStringController), nameof(SelectAllOrder));

}

[HttpGet, Route("dataforupdate")]

public HttpResponseMessage GetDataAboutOrderToUpdate([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OtherController.OrderDBController.SelectDataForUpdate(idOrder);

}, nameof(SubStringController), nameof(GetDataAboutOrderToUpdate));

}

[HttpPost, Route("create")]

public HttpResponseMessage CreateNewOrders([FromBody] WorkWithOrder.BaseOrderInformation NewOrderData)

{

return CatchError(() =>

{

return DBController.CreateNewOrder(NewOrderData);

}, nameof(SubStringController), nameof(CreateNewOrders));

}

[HttpPost, Route("update")]

public HttpResponseMessage UpdateOrderInformation([FromBody] WorkWithOrder.BaseOrderInformation UpdateDataAbOrder)

{

return CatchError(() =>

{

return DBController.UpdateDataAboutOrder(UpdateDataAbOrder);

}, nameof(SubStringController), nameof(UpdateOrderInformation));

}

#endregion

#region Контроллер оплаты

[HttpGet, Route("get/payment")]

public HttpResponseMessage SelectDataAbOrderPaymentByIDorder([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OrderDBController.MakeDataAboutAllOrderPayment(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderPaymentByIDorder));

}

#endregion

#region Работа с заданиями

[HttpGet, Route("get/task")]

public HttpResponseMessage SelectDataAbOrderTaskByIDOrder([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OrderDBController.MakeDataAboutAllOrderTask(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

[HttpPost, Route("create/task")]

public HttpResponseMessage CreateTaskInOrder([FromBody] WorkWithOrder.InformationAboutTask DataAboutTask)

{

return CatchError(() =>

{

return OrderDBController.CreateNewTask(DataAboutTask);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

[HttpGet, Route("get/task")]

public HttpResponseMessage CreateTaskInOrder([FromUri] Guid idTask)

{

return CatchError(() =>

{

return OrderDBController.GetDataAboutTask(idTask);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

[HttpGet, Route("delete/task")]

public HttpResponseMessage DeleteTask([FromUri] Guid idTask,[FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OrderDBController.DeleteTask(idTask,idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

#endregion

#region Для отчетов

[HttpGet, Route("doc/sprav")]

public HttpResponseMessage DataForSpravka([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OrderDBController.MakeDataForSprav(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

[HttpGet, Route("doc/dogovor")]

public HttpResponseMessage DataForDogovor([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OrderDBController.MakeDataForDogovor(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

[HttpGet, Route("doc/smeta1")]

public HttpResponseMessage DataAbSmeta([FromUri] Guid idOrder)

{

return CatchError(() =>

{

return OrderDBController.MakeSmetaAll(idOrder);

}, nameof(SubStringController), nameof(SelectDataAbOrderTaskByIDOrder));

}

#endregion

}

}

}

Модуль RepairFlatRestApi. UserController.cs

using RepairFlatRestApi.Models.DescriptionJSON;

using System;

using System.Net.Http;

using System.Web.Http;

namespace RepairFlatRestApi.Controllers

{

[RoutePrefix("api/User")]

public class UserController : BaseController

{

[HttpPost, Route("create")]

public HttpResponseMessage CreateNewPerson([FromBody] PersonDesctiption.CreateNewClient DescriptionPerson)

{

return CatchError(() =>

{

return OtherController.ClientDBWorker.CreateNewClient(DescriptionPerson);

}, nameof(LoginController), nameof(CreateNewPerson));

}

[HttpPost, Route("update")]

public HttpResponseMessage UpdateDataAboutClient([FromBody] PersonDesctiption.CreateNewClient DescriptionPerson)

{

return CatchError(() =>

{

return OtherController.ClientDBWorker.UpdateDataAboutClient(DescriptionPerson);

}, nameof(LoginController), nameof(CreateNewPerson));

}

[HttpGet, Route("selectallClient")]

public HttpResponseMessage CreateListOf()

{

return CatchError(() =>

{

return OtherController.ClientDBWorker.CreateListOfClient();

}, nameof(LoginController), nameof(CreateNewPerson));

}

[HttpGet, Route("get")]

public HttpResponseMessage DataAboutUser([FromUri] Guid idUser)

{

return CatchError(() =>

{

return OtherController.ClientDBWorker.GetDataAboutUser(idUser);

}, nameof(LoginController), nameof(CreateNewPerson));

}

}

}

Модуль RepairFlatRestApi. ClientDBWorker.cs

using RepairFlatRestApi.Models;

using RepairFlatRestApi.Models.DescriptionJSON;

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Linq;

namespace RepairFlatRestApi.Controllers.OtherController

{

public class ClientDBWorker : DBController

{

internal static object CreateNewClient(PersonDesctiption.CreateNewClient descriptionPerson)

{

return Run((db) =>

{

try

{

db.User.AddOrUpdate(new User

{

idUser = descriptionPerson.idUser,

Name = descriptionPerson.Name,

LastName = descriptionPerson.Lastname,

Patronymic = descriptionPerson.Patronymic,

Pasport = descriptionPerson.Pasport,

Female = descriptionPerson.Female,

BirstDay = descriptionPerson.Birstday,

TypeOfUser = descriptionPerson.TypeOfUser

});

db.ClientDetails.AddOrUpdate(new ClientDetails

{

IdClient = descriptionPerson.idUser,

Description = descriptionPerson.Desc,

});

if (descriptionPerson.ListOfContact != null)

{

if (descriptionPerson.ListOfContact.Any())

{

foreach (var contact in descriptionPerson.ListOfContact)

{

db.UserContact.AddOrUpdate(new UserContact

{

DateAdd = contact.DateAdd,

Description = contact.Desctription,

id = contact.idContact,

idType = contact.idTypeOfContact,

idUser = contact.idUser,

Value = contact.Value

});

}

}

}

db.SaveChanges();

return new BaseResult() { success = true };

}

catch (Exception ex)

{

return new BaseResult()

{

success = false,

description = ex.Message

};

}

});

}

internal static object GetDataAboutUser(Guid idUser)

{

return Run((db) =>

{

var data = db.ClientDetails.Where(ee => ee.IdClient == idUser).FirstOrDefault();

if (data != null)

{

PersonDesctiption.CreateNewClient client = new PersonDesctiption.CreateNewClient

{

idUser = data.IdClient,

Birstday = data.User.BirstDay,

Desc = data.Description,

Female = data.User.Female,

Lastname = data.User.LastName,

Name = data.User.Name,

Pasport = data.User.Pasport,

Patronymic = data.User.Patronymic

};

return client;

}

else

{

return new PersonDesctiption.CreateNewClient { idUser = new Guid() };

}

});

}

internal static object UpdateDataAboutClient(PersonDesctiption.CreateNewClient descriptionPerson)

{

return Run((db) =>

{

try

{

var DataToUpdate = db.User.Where(e1 => e1.idUser == descriptionPerson.idUser).First();

DataToUpdate.LastName = descriptionPerson.Lastname;

DataToUpdate.Name = descriptionPerson.Name;

DataToUpdate.Patronymic = descriptionPerson.Patronymic;

DataToUpdate.Pasport = descriptionPerson.Pasport;

DataToUpdate.ClientDetails.Description = descriptionPerson.Desc;

DataToUpdate.BirstDay = descriptionPerson.Birstday;

DataToUpdate.Female = descriptionPerson.Female;

if (descriptionPerson.ListOfContact != null)

{

foreach (var contact in descriptionPerson.ListOfContact)

{

db.UserContact.AddOrUpdate(new UserContact

{

DateAdd = contact.DateAdd,

Description = contact.Desctription,

id = contact.idContact,

idType = contact.idTypeOfContact,

idUser = contact.idUser,

Value = contact.Value

});

}

}

db.SaveChanges();

return new BaseResult()

{

success = true

};

}

catch (Exception ex)

{

return new BaseResult()

{

success = false,

description = ex.Message

};

}

});

}

internal static PersonDesctiption.ListOfClient CreateListOfClient()

{

return Run((db) =>

{

try

{

var DataFromServer = db.User.Where(ee => ee.TypeOfUser == SomeEnums.TypeOfUser.Cl.ToString()).AsEnumerable();

if (DataFromServer != null)

{

List<PersonDesctiption.DataAboutClient> dataAboutClient = new List<PersonDesctiption.DataAboutClient>();

foreach (var clientInformation in DataFromServer)

{

PersonDesctiption.DataAboutClient clientInf = new PersonDesctiption.DataAboutClient

{

Birstday = clientInformation.BirstDay,

Name = clientInformation.Name,

idUser = clientInformation.idUser,

Female = clientInformation.Female,

Description = clientInformation.ClientDetails.Description,

Lastname = clientInformation.LastName,

Pasport = clientInformation.Pasport,

Patronymic = clientInformation.Patronymic

};

dataAboutClient.Add(clientInf);

}

return new PersonDesctiption.ListOfClient()

{

success = true,

description = "",

listOfClient = dataAboutClient

};

}

else

{

return new PersonDesctiption.ListOfClient()

{

success = false

};

}

}

catch (Exception ex)

{

return new PersonDesctiption.ListOfClient()

{

success = false,

description = ex.Message

};

}

});

}

}

}

Модуль RepairFlatRestApi. DBController.cs

using RepairFlatRestApi.Models;

using RepairFlatRestApi.Models.DescriptionJSON;

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Globalization;

using System.Linq;

using static RepairFlatRestApi.Models.AdressModel;

namespace RepairFlatRestApi.Controllers

{

/// <summary>

///

/// </summary>

public class DBController

{

#region Обработки при работе с данными пользователя

internal static WorkWithOrder.DataForRedact SelectAllDataAboutOrder(Guid idOrder)

{

return Run((db) =>

{

var informationAbOrder = db.OrderInformation.Where(e => e.IdOrder == idOrder).First();

var selectuser = db.User.Where(e => e.idUser == informationAbOrder.idClient).FirstOrDefault();

var selectContact = db.UserContact.Where(e => e.idUser == informationAbOrder.idClient).First();

var adressDesc = db.AdressDescription.Where(e => e.idAdress == informationAbOrder.IdAdress).First();

WorkWithOrder.DataForRedact dataForRedact = new WorkWithOrder.DataForRedact();

dataForRedact.FioClient = $"{selectuser.LastName} {selectuser.Name.Substring(0, 1)}.{selectuser.Patronymic.Substring(0, 1)}.";

dataForRedact.MainSS = $"{selectContact.TypeOfContact.Value} : {selectContact.Value}";

dataForRedact.DescAboutAdress = $"{adressDesc.RegionName} {adressDesc.CiryName} {adressDesc.MicroAreaName} {adressDesc.Street} {adressDesc.House} {adressDesc.Entrance} {adressDesc.NumberOfDelen}";

dataForRedact.IformationAboutOrder = new WorkWithOrder.BaseOrderInformation

{

idAdress = informationAbOrder.IdAdress,

Allsumma = informationAbOrder.AllSumma,

DataStart = informationAbOrder.DateStart,

Desc = informationAbOrder.Description,

idClient = informationAbOrder.idClient,

idOrder = informationAbOrder.IdOrder,

idWorkerMake = informationAbOrder.IdWorkerMake,

MainContactID = informationAbOrder.MainContactID,

Status = informationAbOrder.Status

};

return dataForRedact;

}, nameof(DBController), nameof(SelectAllDataAboutOrder));

}

internal static object GetDataAboutContact(Guid idAdress)

{

return Run((db) =>

{

var select = db.AdressDescription.Where(ee => ee.idAdress == idAdress).FirstOrDefault();

return new AdressDesc

{

AreaName= select.AreaName,

Desc= select.Description,

CityName = select.CiryName,

description = select.Description,

Entrance = select.Entrance,

House = select.House,

idAdress = select.idAdress,

MicroAreaName = select.MicroAreaName,

NumberOfDelen = select.NumberOfDelen,

RegionName = select.RegionName,

Street = select.Street,

success = true

};

});

}

#endregion

#region Работа с адресом

internal static object CreaNewAdress(AdressModel.AdressDesc newAdress)

{

return Run((db) =>

{

try

{

db.AdressDescription.Add(new AdressDescription

{

AreaName = newAdress.AreaName,

CiryName = newAdress.CityName,

Description = newAdress.description,

Entrance = newAdress.Entrance,

House = newAdress.House,

idAdress = newAdress.idAdress,

MicroAreaName = newAdress.MicroAreaName,

NumberOfDelen = newAdress.NumberOfDelen,

RegionName = newAdress.RegionName,

Street = newAdress.Street

});

db.SaveChanges();

return new BaseResult { success = true };

}

catch (Exception ex)

{

return new BaseResult { success = false, description = ex.ToString() };

}

}, nameof(DBController), nameof(CreaNewAdress));

}

internal static object UpdateDataAboutAdress(AdressModel.AdressDesc newAdress)

{

return Run((db) =>

{

try

{

var UpdatedAdress = db.AdressDescription.Where(e => e.idAdress == newAdress.idAdress).FirstOrDefault();

UpdatedAdress.CiryName = newAdress.CityName;

UpdatedAdress.Description = newAdress.Desc;

UpdatedAdress.Entrance = newAdress.Entrance;

UpdatedAdress.House = newAdress.House;

UpdatedAdress.idAdress = newAdress.idAdress;

UpdatedAdress.MicroAreaName = newAdress.MicroAreaName;

UpdatedAdress.NumberOfDelen = newAdress.NumberOfDelen;

UpdatedAdress.RegionName = newAdress.RegionName;

UpdatedAdress.Street = newAdress.Street;

UpdatedAdress.AreaName = newAdress.AreaName;

db.SaveChanges();

return new BaseResult { success = true };

}

catch (Exception ex)

{

return new BaseResult { success = false, description = ex.ToString() };

}

}, nameof(DBController), nameof(UpdateDataAboutAdress));

}

internal static object MakeUpdatePost(MakeSubs.MakeUpdOrInsPost listOfPost)

{

return Run((db) =>

{

try

{

List<Guid?> ListOfDeleteCodes = new List<Guid?>();

DateTime DateOfInsert = new DateTime();

if (!DateTime.TryParseExact(listOfPost.DateOfMake, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfInsert))

return new MakeSubs.ServisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

if (listOfPost.ListOfPostInsert != null)

{

foreach (var WhatInsert in listOfPost.ListOfPostInsert)

{

if(db.WorkerPosts.Where(ee=>ee.idPost== WhatInsert.idPost).FirstOrDefault()==null)

{

var NewPost = new WorkerPosts

{

BaseWage = WhatInsert.BaseWage,

NameOfPost = WhatInsert.NameOfPost,

idPost = WhatInsert.idPost,

MakeWork= WhatInsert.MakeWork

};

var InformationAboutIsert = new UpdateSubInformation

{

DateOfUpdate = DateOfInsert,

idSubIn = WhatInsert.idPost,

idUserMake = listOfPost.idUser,

idInformation = Guid.NewGuid(),

TypeOfSubs= SomeEnums.TypeOfSubs.Post.ToString(),

TypeOfUpdate = SomeEnums.TypeOfAction.AddOrUpdate.ToString()

};

db.WorkerPosts.Add(NewPost).UpdateSubInformation.Add(InformationAboutIsert);

}

}

}

if (listOfPost.listOfPostUpdate != null)

{

foreach (var WhatUpdate in listOfPost.listOfPostUpdate)

{

var UpdatedServis = db.WorkerPosts.Where(e => e.idPost == WhatUpdate.idPost).FirstOrDefault();

if (UpdatedServis != null)

{

UpdatedServis.NameOfPost = WhatUpdate.NameOfPost;

UpdatedServis.BaseWage = WhatUpdate.BaseWage;

UpdatedServis.MakeWork = WhatUpdate.MakeWork;

var InformationAboutUpdate = new UpdateSubInformation

{

DateOfUpdate = DateOfInsert,

idSubIn = WhatUpdate.idPost,

idUserMake = listOfPost.idUser,

idInformation = Guid.NewGuid(),

TypeOfSubs = SomeEnums.TypeOfSubs.Post.ToString(),

TypeOfUpdate = SomeEnums.TypeOfAction.Update.ToString()

};

UpdatedServis.UpdateSubInformation.Add(InformationAboutUpdate);

}

}

}

if (listOfPost.ListOfDeletePost != null)

{

foreach (var WhatDelete in listOfPost.ListOfDeletePost)

{

var DeleteThings = db.WorkerPosts.Where(e => e.idPost == WhatDelete.idGuid).FirstOrDefault();

if (DeleteThings != null)

{

db.Entry(DeleteThings).Collection(c => c.UpdateSubInformation).Load();

db.WorkerPosts.Remove(DeleteThings);

foreach(var delThink in DeleteThings.UpdateSubInformation)

{

DeleteThings.UpdateSubInformation.Remove(delThink);

}

ListOfDeleteCodes.Add(WhatDelete.idGuid);

}

}

}

db.SaveChanges();

if (ListOfDeleteCodes.Count != 0)

{

db.DeletedSubStr.AddRange(MakeListAboutDelete(ListOfDeleteCodes, DateOfInsert, listOfPost.idUser, SomeEnums.TypeOfSubs.Servises));

db.SaveChanges();

}

}

catch (Exception ex)

{

return new BaseResult

{

success = false,

description = $"Ошибка при работе с данными {ex.ToString()}!"

};

}

return new BaseResult

{

success = true,

description = "Операции над данными были произведены!"

};

}, nameof(DBController), nameof(MakeInserAndUpdateServises));

}

internal static MakeSubs.PostMake MakeDataAboutPost(string dateofclientlastupdate)

{

return Run((db) =>

{

if (string.IsNullOrEmpty(dateofclientlastupdate))

{//Если строка пустая возвращаем все

return AllPostHave();

}

else

{

DateTime DateOfLastUpdate = new DateTime();

if (DateTime.TryParseExact(dateofclientlastupdate, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfLastUpdate))

{//Если дату удалось распознать вернуть в соответствии с датой

var QueryWithOutDelete = db.WorkerPosts.Where((e) => e.UpdateSubInformation.FirstOrDefault().DateOfUpdate > DateOfLastUpdate && e.UpdateSubInformation.FirstOrDefault().TypeOfSubs== SomeEnums.TypeOfSubs.Post.ToString());

var ListOfPost = QueryWithOutDelete.Select(e => new MakeSubs.ListOfPostUpd

{

idPost=e.idPost,

BaseWage=e.BaseWage,

NameOfPost=e.NameOfPost,

TypeOfUpdate=e.UpdateSubInformation.FirstOrDefault().TypeOfUpdate,

MakeWork=e.MakeWork

}).ToList();

var QueryForDelete = db.DeletedSubStr.Where(e => e.DateOfDelete > DateOfLastUpdate && e.TypeOfDeleted == SomeEnums.TypeOfSubs.Post.ToString());

var ListOfDelete = QueryForDelete.Select(e => new MakeSubs.ListOfGuid

{

idGuid = e.idThingsDelete

}).ToList();

return new MakeSubs.PostMake

{

success = true,

kol = ListOfPost.Count + ListOfDelete.Count,

DateOfMakeAnswer = DateTime.Now,

listOfPost = ListOfPost,

ListOfDeletePost = ListOfDelete

};

}

else

{//Если дату не удалось распознать

return new MakeSubs.PostMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

}

}

}, nameof(DBController), nameof(UpdateDataAboutAdress));

}

private static MakeSubs.PostMake AllPostHave()

{

return Run((db) =>

{

var listOfPost = db.WorkerPosts.Select(e => new MakeSubs.ListOfPostUpd

{

idPost=e.idPost,

BaseWage=e.BaseWage,

NameOfPost=e.NameOfPost,

MakeWork=e.MakeWork

}).ToList();

return new MakeSubs.PostMake

{

success = true,

kol = listOfPost.Count,

listOfPost=listOfPost,

DateOfMakeAnswer = DateTime.Now

};

});

}

public static AuthDescription.ResultOfInformation Logining(AuthDescription.AskedInformation asked)

{

return Run((db) =>

{

var InfrormationAboutLogin = db.LoginInformation.Where(e => e.Login == asked.login && e.Password == asked.password);

if (InfrormationAboutLogin.FirstOrDefault() == null)

{

return new AuthDescription.ResultOfInformation

{

success = false,

description = "Не корректный логин и пароль"

};

}

else

{

var data = InfrormationAboutLogin.FirstOrDefault();

string dd = $"{data.User.LastName.Trim()} {data.User.Name.Substring(0, 1).ToUpper()}.{data.User.Patronymic.Substring(0, 1)}.";

return new AuthDescription.ResultOfInformation

{

success = true,

idUser = data.IdLog,

typeofpolz = data.User.TypeOfUser?.Trim(),

LastNameAndIni = dd

};

}

}, nameof(DBController), nameof(Logining));

}

/// <summary>

internal static AuthDescription.SucessNewUser CreateLoginPerson(AuthDescription.RegisterLoginPerson informationAboutNewPerson)

{

return Run((db) =>

{

Guid idUser = informationAboutNewPerson.idUser;

if (db.LoginInformation.Where(e => e.IdLog == idUser).FirstOrDefault() != null)

{

return new AuthDescription.SucessNewUser

{

success = false,

description = "Для данного пользователя уже есть информация о логине и пароле!"

};

}

var InforMationAboutLoginInTable = db.LoginInformation.Select(e => e.Login == informationAboutNewPerson.login);

if (db.LoginInformation.Where(e => e.Login == informationAboutNewPerson.login).FirstOrDefault() != null)

{

return new AuthDescription.SucessNewUser

{

success = false,

description = "Логин уже используется"

};

}

var newLogInformation = new LoginInformation

{

IdLog = idUser,

Login = informationAboutNewPerson.login,

Password = informationAboutNewPerson.password

};

db.LoginInformation.Add(newLogInformation);

db.SaveChanges();

return new AuthDescription.SucessNewUser

{

success = true,

idUser = idUser,

description = "Данные о логине и пароле добавлены"

};

}, nameof(DBController), nameof(CreateLoginPerson));

}

#endregion

#region Что-то общее

internal static DeletedSubStr[] MakeListAboutDelete(List<Guid?> ListOfDeleteCodes, DateTime DateOfDelete, Guid idUser, SomeEnums.TypeOfSubs typeOfSubs)

{

DeletedSubStr[] result = new DeletedSubStr[ListOfDeleteCodes.Count];

int l = 0;

foreach (var codes in ListOfDeleteCodes)

{

var InformationAboutDelete = new DeletedSubStr

{

DateOfDelete = DateOfDelete,

idDeleted = Guid.NewGuid(),

idThingsDelete = codes,

idUserDelete = idUser,

TypeOfDeleted = typeOfSubs.ToString()

};

result[l] = InformationAboutDelete;

l++;

}

return result;

}

#endregion

#region Обработки при работе с подстовляемыми данными об услугах

internal static MakeSubs.ServisesMake MakeDataAboutUpdateServises(string dateofclientlastupdate)

{

return Run((db) =>

{

if (string.IsNullOrEmpty(dateofclientlastupdate))

{//Если строка пустая возвращаем все

return AllServisesHave();

}

else

{

DateTime DateOfLastUpdate = new DateTime();

if (DateTime.TryParseExact(dateofclientlastupdate, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfLastUpdate))

{//Если дату удалось распознать вернуть в соответствии с датой

var QueryWithOutUpdate = db.OurServices.Where((e) => e.UpdateSubInformation.FirstOrDefault().DateOfUpdate > DateOfLastUpdate && e.UpdateSubInformation.FirstOrDefault().TypeOfSubs==SomeEnums.TypeOfSubs.Servises.ToString());

var ListOfServises = QueryWithOutUpdate.Select(e => new MakeSubs.ListOfUpdatedServises

{

idServises = e.idServis,

Nomination = e.Nomination,

TypeOfServises = e.TypeOfServices,

Description = e.Description,

Cost = e.Cost,

TypeOfUpdate = e.UpdateSubInformation.FirstOrDefault().TypeOfUpdate,

}).ToArray();

var QueryForDelete = db.DeletedSubStr.Where(e => e.DateOfDelete > DateOfLastUpdate && e.TypeOfDeleted == SomeEnums.TypeOfSubs.Servises.ToString());

var ListOfDelete = QueryForDelete.Select(e => new MakeSubs.ListOfGuid

{

idGuid = e.idThingsDelete

}).ToArray();

return new MakeSubs.ServisesMake

{

success = true,

kol = ListOfServises.Length + ListOfDelete.Length,

DateOfMakeAnswer = DateTime.Now,

ListOfServises = ListOfServises,

ListOfDeleteServises = ListOfDelete

};

}

else

{//Если дату не удалось распознать

return new MakeSubs.ServisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

}

}

}, nameof(DBController), nameof(MakeDataAboutUpdateServises));

}

internal static object MakeInserAndUpdateServises(MakeSubs.MakeUpdOrInsServises makeUpdateOrInserNew)

{

return Run((db) =>

{

try

{

List<Guid?> ListOfDeleteCodes = new List<Guid?>();

DateTime DateOfInsert = new DateTime();

if (!DateTime.TryParseExact(makeUpdateOrInserNew.DateOfMake, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfInsert))

return new MakeSubs.ServisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

if (makeUpdateOrInserNew.ListOfServises != null)

{

foreach (var WhatInsert in makeUpdateOrInserNew.ListOfServises)

{

var NewServises = new OurServices

{

Cost = WhatInsert.Cost,

Description = WhatInsert.Description,

idServis = WhatInsert.idServises,

Nomination = WhatInsert.Nomination,

TypeOfServices = WhatInsert.TypeOfServises,

};

var InformationAboutIsert = new UpdateSubInformation

{

idSubIn = WhatInsert.idServises,

TypeOfSubs = SomeEnums.TypeOfSubs.Servises.ToString(),

idInformation = Guid.NewGuid(),

idUserMake= makeUpdateOrInserNew.idUser,

DateOfUpdate = DateOfInsert,

TypeOfUpdate = SomeEnums.TypeOfAction.AddOrUpdate.ToString()

};

db.OurServices.AddOrUpdate(NewServises);

db.UpdateSubInformation.Add(InformationAboutIsert);

db.SaveChanges();

}

}

if (makeUpdateOrInserNew.ListOfDeleteServises != null)

{

foreach (var WhatDelete in makeUpdateOrInserNew.ListOfDeleteServises)

{

var DeleteThings = db.OurServices.Where(e => e.idServis == WhatDelete.idGuid).FirstOrDefault();

if (DeleteThings != null)

{

db.Entry(DeleteThings).Collection(c => c.UpdateSubInformation).Load();

db.OurServices.Remove(DeleteThings);

db.UpdateSubInformation.RemoveRange(db.UpdateSubInformation.Where(c => c.idSubIn == WhatDelete.idGuid).ToArray());

ListOfDeleteCodes.Add(WhatDelete.idGuid);

}

}

}

db.SaveChanges();

if (ListOfDeleteCodes.Count != 0)

{

db.DeletedSubStr.AddRange(MakeListAboutDelete(ListOfDeleteCodes, DateOfInsert, makeUpdateOrInserNew.idUser, SomeEnums.TypeOfSubs.Servises));

db.SaveChanges();

}

}

catch (Exception ex)

{

return new BaseResult

{

success = false,

description = $"Ошибка при работе с данными {ex.ToString()}!"

};

}

return new BaseResult

{

success = true,

description = "Операции над данными были произведены!"

};

}, nameof(DBController), nameof(MakeInserAndUpdateServises));

}

internal static MakeSubs.ServisesMake AllServisesHave()

{

return Run((db) =>

{

var listOfServise = db.OurServices.Select(e => new MakeSubs.ListOfUpdatedServises

{

idServises = e.idServis,

Nomination = e.Nomination,

TypeOfServises = e.TypeOfServices,

Description = e.Description,

Cost = e.Cost,

}).ToArray();

return new MakeSubs.ServisesMake

{

success = true,

kol = listOfServise.Length,

ListOfServises = listOfServise,

DateOfMakeAnswer = DateTime.Now

};

}, nameof(DBController), nameof(AllServisesHave));

}

#endregion

#region Обработки при работе с подстовляемыми данными об типах помещений

internal static object MakeDataAboutUpdatePremises(MakeSubs.MakeUpdOrInsPremises ListOfNewPremises)

{

return Run((db) =>

{

List<Guid?> ListOfDeleteCodes = new List<Guid?>();

DateTime DateOfAction = new DateTime();

if (!DateTime.TryParseExact(ListOfNewPremises.DateOfMake, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfAction))

return new MakeSubs.ServisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

try

{

if (ListOfNewPremises.ListOfPremises != null)

{

foreach (var InsertThing in ListOfNewPremises.ListOfPremises)

{

var newPremise = new PremisesType

{

Descriprtion = InsertThing.Description,

idPremises = InsertThing.idPremises,

NameOfPremises = InsertThing.Name

};

db.PremisesType.AddOrUpdate(newPremise);

var NewPremisesHistory = new UpdateSubInformation

{

idInformation = Guid.NewGuid(),

idUserMake= ListOfNewPremises.idUser,

idSubIn= InsertThing.idPremises,

TypeOfSubs=SomeEnums.TypeOfSubs.Premises.ToString(),

DateOfUpdate = DateOfAction,

TypeOfUpdate = SomeEnums.TypeOfAction.AddOrUpdate.ToString(),

};

db.UpdateSubInformation.AddOrUpdate(NewPremisesHistory);

}

}

if (ListOfNewPremises.ListOfDeletePremises != null)

{

foreach (var whatDelete in ListOfNewPremises.ListOfDeletePremises)

{

var deleteThings = db.PremisesType.Where(e => e.idPremises == whatDelete.idGuid).FirstOrDefault();

if (deleteThings != null)

{

db.Entry(deleteThings).Collection(c => c.UpdateSubInformation).Load();

db.PremisesType.Remove(deleteThings);

db.UpdateSubInformation.RemoveRange(db.UpdateSubInformation.Where(c => c.idSubIn == whatDelete.idGuid).ToArray());

ListOfDeleteCodes.Add(whatDelete.idGuid);

}

}

}

db.SaveChanges();

if (ListOfDeleteCodes.Count != 0)

{

db.DeletedSubStr.AddRange(MakeListAboutDelete(ListOfDeleteCodes, DateOfAction, ListOfNewPremises.idUser, SomeEnums.TypeOfSubs.Premises));

db.SaveChanges();

}

}

catch (Exception ex)

{

return new BaseResult

{

success = false,

description = $"Ошибка при работе с данными {ex.ToString()}!"

};

}

return new BaseResult

{

success = true,

description = "Операции над данными были произведены!"

};

}, nameof(DBController), nameof(MakeDataAboutUpdatePremises));

}

internal static MakeSubs.PremisesMake GetAllPremises(string dateofclientlastupdate)

{

return Run((db) =>

{

if (string.IsNullOrEmpty(dateofclientlastupdate))

{//Если строка пустая возвращаем все

return AllPremisesHave();

}

else

{

DateTime DateOfLastUpdate = new DateTime();

if (DateTime.TryParseExact(dateofclientlastupdate, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfLastUpdate))

{//Если дату удалось распознать вернуть в соответствии с датой

var QueryWithOutDelete = db.PremisesType.Where((e) => e.UpdateSubInformation.FirstOrDefault().DateOfUpdate > DateOfLastUpdate && e.UpdateSubInformation.FirstOrDefault().TypeOfSubs==SomeEnums.TypeOfSubs.Premises.ToString());

var ListPremises = QueryWithOutDelete.Select(e => new MakeSubs.ListOfPremisesUpd

{

idPremises = e.idPremises,

Name = e.NameOfPremises,

Description = e.Descriprtion,

TypeOfUpdate = e.UpdateSubInformation.FirstOrDefault().TypeOfUpdate,

}).ToArray();

var QueryForDelete = db.DeletedSubStr.Where(e => e.DateOfDelete > DateOfLastUpdate && e.TypeOfDeleted == SomeEnums.TypeOfSubs.Premises.ToString());

var ListOfDelete = QueryForDelete.Select(e => new MakeSubs.ListOfGuid

{

idGuid = e.idThingsDelete

}).ToArray();

return new MakeSubs.PremisesMake

{

success = true,

kol = ListPremises.Length + ListOfDelete.Length,

DateOfMakeAnswer = DateTime.Now,

listOfPremises = ListPremises,

ListOfDeletePremises = ListOfDelete

};

}

else

{//Если дату не удалось распознать

return new MakeSubs.PremisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

}

}

}, nameof(DBController), nameof(GetAllPremises));

}

private static MakeSubs.PremisesMake AllPremisesHave()

{

return Run((db) =>

{

var listOfServise = db.PremisesType.Select(e => new MakeSubs.ListOfPremisesUpd

{

idPremises = e.idPremises,

Name = e.NameOfPremises,

Description = e.Descriprtion

}).ToArray();

return new MakeSubs.PremisesMake

{

success = true,

kol = listOfServise.Length,

listOfPremises = listOfServise,

DateOfMakeAnswer = DateTime.Now

};

}, nameof(DBController), nameof(AllServisesHave));

}

#endregion

#region Обработки при работе с подстовляемыми данными о материалах

internal static MakeSubs.MaterialsMake MakeAllDataAboutContackt()

{

return Run((db) =>

{

var ListOfMaterial = db.OurMaterials.Select(e => new MakeSubs.ListOfMaterialsUpd

{

idMaterials = e.idMaterial,

NameOfMaterial = e.NameOfMaterial,

UnitOfMeasue = e.UnitOfMeasue,

Cost = e.Cost,

Description = e.Description

}).ToArray();

return new MakeSubs.MaterialsMake

{

success = true,

kol = ListOfMaterial.Length,

listOfMaterials = ListOfMaterial,

DateOfMakeAnswer = DateTime.Now

};

}, nameof(DBController), nameof(MakeAllDataAboutContackt));

}

internal static object MakeUpdateMaterials(MakeSubs.MakeUpdOrInsMaterials ListOfMaterials)

{

return Run((db) =>

{

List<Guid?> ListOfDeleteCodes = new List<Guid?>();

DateTime DateOfAction = new DateTime();

if (!DateTime.TryParseExact(ListOfMaterials.DateOfMake, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfAction))

return new MakeSubs.ServisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

try

{

if (ListOfMaterials.ListOfMaterials != null)

{

foreach (var NewMaterial in ListOfMaterials.ListOfMaterials)

{

var NMat = new OurMaterials

{

Cost = NewMaterial.Cost,

Description = NewMaterial.Description,

idMaterial = NewMaterial.idMaterials,

NameOfMaterial = NewMaterial.NameOfMaterial,

TypeOfMaterial = NewMaterial.TypeOfMaterial,

UnitOfMeasue = NewMaterial.UnitOfMeasue

};

var dd = new Promezh

{

idColob = Guid.NewGuid(),

idSubInf = NewMaterial.idMaterials

};

var NMatHistory = new UpdateSubInformation

{

DateOfUpdate = DateOfAction,

TypeOfUpdate = SomeEnums.TypeOfAction.AddOrUpdate.ToString(),

TypeOfSubs = SomeEnums.TypeOfSubs.Materials.ToString(),

idSubIn = NewMaterial.idMaterials,

idInformation = Guid.NewGuid(),

idUserMake = ListOfMaterials.idUser

};

db.OurMaterials.AddOrUpdate(NMat);

db.UpdateSubInformation.AddOrUpdate(NMatHistory);

}

}

if (ListOfMaterials.ListOfDeleteMaterials != null)

{

foreach (var DelMaterila in ListOfMaterials.ListOfDeleteMaterials)

{

var DeleteThing = db.OurMaterials.Where(e => e.idMaterial == DelMaterila.idGuid).FirstOrDefault();

if (DeleteThing != null)

{

db.Entry(DeleteThing).Collection(c => c.UpdateSubInformation).Load();

db.OurMaterials.Remove(DeleteThing);

db.UpdateSubInformation.RemoveRange(db.UpdateSubInformation.Where(c => c.idSubIn == DelMaterila.idGuid).ToArray());

ListOfDeleteCodes.Add(DelMaterila.idGuid);

}

}

}

db.SaveChanges();

if (ListOfDeleteCodes.Count != 0)

{

db.DeletedSubStr.AddRange(MakeListAboutDelete(ListOfDeleteCodes, DateOfAction, ListOfMaterials.idUser, SomeEnums.TypeOfSubs.Materials));

db.SaveChanges();

}

}

catch (Exception ex)

{

return new BaseResult

{

success = false,

description = $"Ошибка при работе с данными {ex.ToString()}!"

};

}

return new BaseResult

{

success = true,

description = "Операции над данными были произведены!"

};

}, nameof(DBController), nameof(MakeUpdateMaterials));

}

internal static object MakeListOfMaterials(string dateofclientlastupdate)

{

return Run((db) =>

{

if (string.IsNullOrEmpty(dateofclientlastupdate))

{//Если строка пустая возвращаем все

return MakeAllDataAboutContackt();

}

else

{

DateTime DateOfLastAction = new DateTime();

if (DateTime.TryParseExact(dateofclientlastupdate, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfLastAction))

{//Если дату удалось распознать вернуть в соответствии с датой

var QueryWithOutDelete = db.OurMaterials.Where((e) => e.UpdateSubInformation.FirstOrDefault().DateOfUpdate > DateOfLastAction && e.UpdateSubInformation.FirstOrDefault().TypeOfSubs==SomeEnums.TypeOfSubs.Materials.ToString());

var ListPremises = QueryWithOutDelete.Select(e => new MakeSubs.ListOfMaterialsUpd

{

idMaterials = e.idMaterial,

NameOfMaterial = e.NameOfMaterial,

UnitOfMeasue = e.UnitOfMeasue,

Cost = e.Cost,

Description = e.Description,

TypeOfUpdate = e.UpdateSubInformation.FirstOrDefault().TypeOfUpdate

}).ToArray();

var QueryForDelete = db.DeletedSubStr.Where(e => e.DateOfDelete > DateOfLastAction && e.TypeOfDeleted == SomeEnums.TypeOfSubs.Materials.ToString());

var ListOfDelete = QueryForDelete.Select(e => new MakeSubs.ListOfGuid

{

idGuid = e.idThingsDelete

}).ToArray();

return new MakeSubs.MaterialsMake

{

success = true,

kol = ListPremises.Length + ListOfDelete.Length,

DateOfMakeAnswer = DateTime.Now,

listOfMaterials = ListPremises,

ListOfDeleteMaterials = ListOfDelete

};

}

else

{//Если дату не удалось распознать

return new MakeSubs.MaterialsMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

}

}

}, nameof(DBController), nameof(MakeListOfMaterials));

}

#endregion

#region Обработки при работе с подстовляемыми данными о контактах

internal static MakeSubs.ContactsMake MakeAllDataAboutContacts()

{

return Run((db) =>

{

var ListOfContacts = db.TypeOfContact.Select(e => new MakeSubs.ListOfContactsUpd

{

Description = e.Description,

Value = e.Value,

idContact = e.idContact

}).ToList();

return new MakeSubs.ContactsMake

{

success = true,

kol = ListOfContacts.Count,

listOfContacts = ListOfContacts,

DateOfMakeAnswer = DateTime.Now

};

}, nameof(DBController), nameof(MakeAllDataAboutContackt));

}

internal static object MakeUpdateContacts(MakeSubs.MakeUpdOrInsContacts ListOfContacts)

{

return Run((db) =>

{

List<Guid?> ListOfDeleteCodes = new List<Guid?>();

DateTime DateOfAction = new DateTime();

if (!DateTime.TryParseExact(ListOfContacts.DateOfMake, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfAction))

return new MakeSubs.ServisesMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

try

{

if (ListOfContacts.ListOfContacts != null)

{

foreach (var ListOfInsContacts in ListOfContacts.ListOfContacts)

{

var NTContact = new TypeOfContact

{

idContact = ListOfInsContacts.idContact ?? default(Guid),

Description = ListOfInsContacts.Description,

Regex= ListOfInsContacts.Regex,

Value = ListOfInsContacts.Value

};

var HistOfInsContact = new UpdateSubInformation

{

TypeOfSubs=SomeEnums.TypeOfSubs.Contact.ToString(),

idSubIn= ListOfInsContacts.idContact,

idInformation=Guid.NewGuid(),

idUserMake= ListOfContacts.idUser,

DateOfUpdate = DateOfAction,

TypeOfUpdate = SomeEnums.TypeOfAction.AddOrUpdate.ToString()

};

db.TypeOfContact.AddOrUpdate(NTContact);

db.UpdateSubInformation.AddOrUpdate(HistOfInsContact);

}

}

if (ListOfContacts.ListOfDeleteContacts != null)

{

foreach (var DeleteContact in ListOfContacts.ListOfDeleteContacts)

{

var Query = db.TypeOfContact.Where(e => e.idContact == DeleteContact.idGuid).FirstOrDefault();

if (Query != null)

{

db.Entry(Query).Collection(c => c.UpdateSubInformation).Load();

db.TypeOfContact.Remove(Query);

db.UpdateSubInformation.RemoveRange(db.UpdateSubInformation.Where(c => c.idSubIn == DeleteContact.idGuid).ToArray());

ListOfDeleteCodes.Add(DeleteContact.idGuid);

}

}

}

db.SaveChanges();

if (ListOfDeleteCodes.Count != 0)

{

db.DeletedSubStr.AddRange(MakeListAboutDelete(ListOfDeleteCodes, DateOfAction, ListOfContacts.idUser, SomeEnums.TypeOfSubs.Contact));

db.SaveChanges();

}

}

catch (Exception ex)

{

return new BaseResult

{

success = false,

description = $"Ошибка при работе с данными {ex.ToString()}!"

};

}

return new BaseResult

{

success = true,

description = "Операции над данными были произведены!"

};

}, nameof(DBController), nameof(MakeUpdateContacts));

}

internal static object MakeListOfContacts(string dateOfLastClientAction)

{

return Run((db) =>

{

if (string.IsNullOrEmpty(dateOfLastClientAction))

{//Если строка пустая возвращаем все

return MakeAllDataAboutContacts();

}

else

{

DateTime DateOfLastAction = new DateTime();

if (DateTime.TryParseExact(dateOfLastClientAction, "dd.MM.yyyy HH:mm", CultureInfo.GetCultureInfo("ru-RU"), DateTimeStyles.None, out DateOfLastAction))

{//Если дату удалось распознать вернуть в соответствии с датой

var QueryWithOutDelete = db.UpdateSubInformation.Where((e) => e.DateOfUpdate > DateOfLastAction && e.TypeOfSubs==SomeEnums.TypeOfSubs.Contact.ToString());

var ListPremises = QueryWithOutDelete.Select(e => new MakeSubs.ListOfContactsUpd

{

Description = e.TypeOfContact.Description,

idContact = e.TypeOfContact.idContact,

Value = e.TypeOfContact.Value,

TypeOfUpdate = e.TypeOfUpdate,

Regex=e.TypeOfContact.Regex

}).ToList();

var QueryForDelete = db.DeletedSubStr.Where(e => e.DateOfDelete > DateOfLastAction && e.TypeOfDeleted == SomeEnums.TypeOfSubs.Contact.ToString());

var ListOfDelete = QueryForDelete.Select(e => new MakeSubs.ListOfGuid

{

idGuid = e.idThingsDelete

}).ToList();

return new MakeSubs.ContactsMake

{

success = true,

kol = ListPremises.Count + ListOfDelete.Count,

DateOfMakeAnswer = DateTime.Now,

listOfContacts = ListPremises,

ListOfDeleteContacts = ListOfDelete

};

}

else

{//Если дату не удалось распознать

return new MakeSubs.ContactsMake

{

success = false,

description = "Ошибка при преобразовании данных о дате"

};

}

}

}, nameof(DBController), nameof(MakeUpdateContacts));

}

#endregion

#region Работа с заказами

internal static WorkWithOrder.DataAboutAllOrder MakeDataAboutAllOrder()

{

return Run((db) =>

{

List<WorkWithOrder.OrderInTable> orders = new List<WorkWithOrder.OrderInTable>();

foreach (var order in db.OrderInformation.AsEnumerable())

{//TODO Оптимизировать

string F = db.User.Where(e => e.idUser == order.idClient).Select(e => e.LastName).FirstOrDefault();

string I = db.User.Where(e => e.idUser == order.idClient).Select(e => e.Name.Substring(0, 1)).FirstOrDefault();

string O = db.User.Where(e => e.idUser == order.idClient).Select(e => e.Patronymic.Substring(0, 1)).FirstOrDefault();

string NameOfCleint = $"{F} {I}.{O}.";

string TypeOfContact = db.TypeOfContact.Where((e) => e.UserContact.First().id == order.MainContactID).Select(e => e.Value).FirstOrDefault();

string ValuesOfContact = db.UserContact.Where(e => e.id == order.MainContactID).Select(e => e.Value).First();

string ContactInformatiom = $"{TypeOfContact} : {ValuesOfContact}";

var informationAboutAdress = db.AdressDescription.Where(e => e.idAdress == order.IdAdress).Select(e => new AdressDescription { idAdress = e.idAdress, AreaName = e.AreaName, Entrance = e.Entrance, CiryName = e.CiryName, Description = e.Description, House = e.House, MicroAreaName = e.MicroAreaName, NumberOfDelen = e.NumberOfDelen, RegionName = e.RegionName, Street = e.Street }).First();

string Adressinformation = $"{informationAboutAdress.RegionName} {informationAboutAdress.CiryName} {informationAboutAdress.MicroAreaName} {informationAboutAdress.Street} {informationAboutAdress.House} {informationAboutAdress.Entrance} {informationAboutAdress.NumberOfDelen}";

orders.Add(new WorkWithOrder.OrderInTable

{

DataAboutContact = ContactInformatiom,

DataStart = order.DateStart,

FIOCLient = NameOfCleint,

idOrder = order.IdOrder,

InformationAboutAdress = Adressinformation,

money = order.AllSumma,

number = order.Number,

Status = order.Status

});

}

return new WorkWithOrder.DataAboutAllOrder() { success = true, orders = orders, Count = orders.Count };

}, nameof(DBController), nameof(MakeDataAboutAllOrder));

}

internal static object CreateNewOrder(WorkWithOrder.BaseOrderInformation newOrderData)

{

return Run((db) =>

{

try

{

db.OrderInformation.Add(new OrderInformation

{

Status = newOrderData.Status,

AllSumma = newOrderData.Allsumma,

IdAdress = newOrderData.idAdress,

idClient = newOrderData.idClient,

MainContactID = newOrderData.MainContactID,

IdOrder = newOrderData.idOrder,

DateStart = newOrderData.DataStart,

IdWorkerMake = newOrderData.idWorkerMake,

Description = newOrderData.Desc,

});

db.SaveChanges();

return new BaseResult { success = true };

}

catch (Exception ex)

{

return new BaseResult { success = false, description = ex.Message };

}

}, nameof(DBController), nameof(CreateNewOrder));

}

internal static object UpdateDataAboutOrder(WorkWithOrder.BaseOrderInformation updateDataAbOrder)

{

return Run((db) =>

{

try

{

var selectWorkersOrder = db.OrderInformation.Where(e1 => e1.IdOrder == updateDataAbOrder.idOrder).First();

if (selectWorkersOrder != null)

{

selectWorkersOrder.AllSumma = updateDataAbOrder.Allsumma;

selectWorkersOrder.IdAdress = updateDataAbOrder.idAdress;

selectWorkersOrder.idClient = updateDataAbOrder.idClient;

selectWorkersOrder.MainContactID = updateDataAbOrder.MainContactID;

selectWorkersOrder.Description = updateDataAbOrder.Desc;

selectWorkersOrder.Status = updateDataAbOrder.Status;

db.SaveChanges();

return new BaseResult { success = true };

}

else

{

return new BaseResult { success = false, description = "Заказ не найден"};

}

}

catch (Exception ex)

{

return new BaseResult { success = false, description = ex.Message };

}

}, nameof(DBController), nameof(CreateNewOrder));

}

#endregion

#region Базовые вещи

public static TResult Run<TResult>(Func<RepaFlat, TResult> dbFunction, string NameOfClass, string nameOfMethod)

{

using (var db = new RepaFlat())

{

try

{

return dbFunction(db);

}

catch (Exception ex)

{

throw new Exception($"В методе {NameOfClass}::{nameOfMethod} произошла ошибка: <{ex.ToString()}>");

}

}

}

public static TResult Run<TResult>(Func<RepaFlat, TResult> dbFunction)

{

using (var db = new RepaFlat())

{

try

{

return dbFunction(db);

}

catch (Exception ex)

{

throw new Exception($"В методе произошла ошибка: <{ex.ToString()}>");

}

}

}

#endregion

}

}

Модуль RepairFlatRestApi. PaymnetDBController.cs

using System;

using System.Collections.Generic;

using System.Data.Entity.Migrations;

using System.Linq;

using System.Web;

using RepairFlatRestApi.Models.DescriptionJSON;

using static RepairFlatRestApi.Models.DescriptionJSON.WorkWithOrder;

using static RepairFlatRestApi.Models.OrderData;

namespace RepairFlatRestApi.Controllers.OtherController

{

public class OrderDBController : DBController

{

internal static object SelectDataAboutAllOrder()

{

return Run((db) =>

{

AllOrder allOrder = new AllOrder();

allOrder.listOfOrders = new List<AllDataAboutOrder>();

var ListOfOrder = db.OrderInformation.AsEnumerable();

foreach (var order in ListOfOrder)

{

string female = order.ClientDetails.User.Female == 1 ? "МУЖ" : "Жен";

AllDataAboutOrder allDataAbout = new AllDataAboutOrder();

allDataAbout.idOrder = order.IdOrder;

allDataAbout.DataStart = order.DateStart;

allDataAbout.Status = order.Status;

allDataAbout.Desc = order.Description?.Trim();

allDataAbout.DataAboutAdress = $"{order.AdressDescription.CiryName?.Trim()} {order.AdressDescription.Street?.Trim()} {order.AdressDescription.House?.Trim()} {order.AdressDescription.Entrance?.Trim()} {order.AdressDescription.NumberOfDelen?.Trim()}";

allDataAbout.ContactData = $"{order.UserContact.TypeOfContact.Value.Trim()} : {order.UserContact.Value.Trim()}";

allDataAbout.AllSumma = order.AllSumma;

allDataAbout.FIOClient = $"{order.ClientDetails.User.LastName?.Trim()} {order.ClientDetails.User.Name?.Substring(0, 1).ToUpper()}.{order.ClientDetails.User.Patronymic?.Substring(0, 1).ToUpper()}. {order.ClientDetails.User.BirstDay.Value.ToString("dd.MM.yyyy") } {female}";

allOrder.listOfOrders.Add(allDataAbout);

}

allOrder.success = true;

return allOrder;

});

}

internal static object DeletePremises(Guid idPremises)

{

return Run((db) =>

{

try

{

var dat = db.OrderMeasurements.Where(ee => ee.idMeasurements == idPremises).First();

if (dat != null)

{

db.OrderMeasurements.Remove(dat);

}

var dd = db.OrderElementOfMeasurments.Where(ee => ee.idMeasurements == idPremises).AsEnumerable();

if (dd != null)

{

if (dd.Any())

{

db.OrderElementOfMeasurments.RemoveRange(dd);

}

}

db.SaveChanges();

return new BaseResult { success = true };

}

catch (Exception ex)

{

return new BaseResult { success = false, description = ex.ToString() };

}

});

}

internal static object SelectDataForUpdate(Guid idOrder)

{

return Run((db) =>

{

var selectedOrder = db.OrderInformation.Where(ee => ee.IdOrder == idOrder).FirstOrDefault();

string female = selectedOrder.ClientDetails.User.Female == 1 ? "МУЖ" : "Жен";

return new Models.OrderData.DataForUpdate

{

AllSumma = selectedOrder.AllSumma,

idOrder = selectedOrder.IdOrder,

idAdress = selectedOrder.IdAdress,

idContact = selectedOrder.MainContactID,

Desc = selectedOrder.Description?.Trim(),

idUser = selectedOrder.idClient,

ContactData = $"{selectedOrder.UserContact.TypeOfContact.Value.Trim()} : {selectedOrder.UserContact.Value.Trim()}",

FIOClient = $"{selectedOrder.ClientDetails.User.LastName?.Trim()} {selectedOrder.ClientDetails.User.Name?.Substring(0, 1).ToUpper()}.{selectedOrder.ClientDetails.User.Patronymic?.Substring(0, 1).ToUpper()}. {selectedOrder.ClientDetails.User.BirstDay.Value.ToString("dd.MM.yyyy") } {female}",

DataStart = selectedOrder.DateStart,

DataAboutAdress = $"{selectedOrder.AdressDescription.CiryName?.Trim()} {selectedOrder.AdressDescription.Street?.Trim()} {selectedOrder.AdressDescription.House?.Trim()} {selectedOrder.AdressDescription.Entrance?.Trim()} {selectedOrder.AdressDescription.NumberOfDelen?.Trim()}",

Status = selectedOrder.Status

};

});

}

internal static object MakeDataForDogovor(Guid idOrder)

{

return Run((db) =>

{

var dd = db.OrderInformation.Where(ee => ee.IdOrder == idOrder).FirstOrDefault();

if (dd != null)

{

return new MakeDogovor

{

Summa = Convert.ToDouble(dd.AllSumma),

success = true,

Adress = $"{dd.AdressDescription.RegionName?.Trim()} {dd.AdressDescription.AreaName?.Trim()} {dd.AdressDescription.CiryName?.Trim()} {dd.AdressDescription.MicroAreaName?.Trim()} {dd.AdressDescription.Street?.Trim()} {dd.AdressDescription.House?.Trim()} {dd.AdressDescription.Entrance?.Trim()} {dd.AdressDescription.NumberOfDelen?.Trim()}",

ContactInf = $"{dd.UserContact.TypeOfContact.Value?.Trim()} : {dd.UserContact.Value?.Trim()} : {dd.UserContact.Description?.Trim()}",

FIOSmall = $"{dd.ClientDetails.User.LastName?.Trim()} {dd.ClientDetails.User.Name?.Substring(0, 1)}.{dd.ClientDetails.User.Patronymic?.Substring(0, 1)}.",

FullFIO = $"{dd.ClientDetails.User.LastName?.Trim()} {dd.ClientDetails.User.Name?.Trim()} {dd.ClientDetails.User.Patronymic?.Trim()} {dd.ClientDetails.User.Pasport?.Trim()}",

Inn = dd.OrderPayment.First().InformatioForPayment.InnOfOrganization,

KPP = dd.OrderPayment.First().InformatioForPayment.KppOfOrganization,

NameOfOrganization = $"<{dd.OrderPayment.First().InformatioForPayment.NameOfRecipient}>"

};

}

else

{

return new MakeDogovor { success = false };

}

});

}

internal static object MakeSmetaAll(Guid idOrder)

{

return Run((db) =>

{

var sme = db.OrderInformation.Where(ee => ee.IdOrder == idOrder).First();

if (sme != null)

{

MakeSmetaAll smet = new MakeSmetaAll();

var dataAbMat = db.TaskMaterials.Where(ee => ee.OrderTasks.IdOrder == idOrder).AsEnumerable();

var DataAbServ = db.TaskServis.Where(ee => ee.OrderTasks.IdOrder == idOrder).AsEnumerable();

if (dataAbMat != null)

{

if (dataAbMat.Any())

{

int numb = 1;

smet.materialsInf = new List<TaskMaterial>();

foreach (var dd in dataAbMat)

{

bool need = true;

if (smet.materialsInf.Any())

{

var data = smet.materialsInf.Where(ee => ee.idMaterial == dd.idMaterial).First();

if (data != null)

{

data.count = data.count + dd.Count;

data.summa = Convert.ToDecimal(data.count) \* data.cost;

need = false;

}

}

if(need)

{

smet.materialsInf.Add(new TaskMaterial

{

cost = dd.Cost,

count = dd.Count,

NameOfMaterials = dd.OurMaterials.NameOfMaterial?.Trim(),

numb = numb,

summa = Convert.ToDecimal(dd.Count) \* dd.Cost

});

numb++;

}

need = true;

}

smet.SummaMat = Convert.ToDouble(smet.materialsInf.Sum(ee => ee.summa));

}

}

if (DataAbServ != null)

{

if (DataAbServ.Any())

{

smet.ServisInf = new List<TaskServises>();

int numb = 1;

foreach (var dd in DataAbServ)

{

bool need = true;

if (smet.ServisInf.Any())

{

var data = smet.ServisInf.Where(ee => ee.idServis == dd.idServis).First();

if (data != null)

{

data.count = data.count + dd.Count;

data.summa = Convert.ToDecimal(data.count) \* data.cost;

need = false;

}

}

if(need)

{

smet.ServisInf.Add(new TaskServises

{

cost = dd.Cost,

count = dd.Count,

NameOfServises = dd.OurServices.Nomination?.Trim(),

summa= Convert.ToDecimal(dd.Count) \* dd.Cost,

numb=numb

});

numb++;

}

need = true;

smet.SummaServ = Convert.ToDouble(smet.ServisInf.Sum(ee => ee.summa));

}

}

}

var Adress = db.OrderInformation.Where(ee => ee.IdOrder == idOrder).First();

smet.AdressOfWork = $"{Adress.AdressDescription.RegionName?.Trim()} {Adress.AdressDescription.AreaName?.Trim()} {Adress.AdressDescription.CiryName?.Trim()} {Adress.AdressDescription.MicroAreaName?.Trim()} {Adress.AdressDescription.Street?.Trim()} {Adress.AdressDescription.House?.Trim()} {Adress.AdressDescription.Entrance?.Trim()} {Adress.AdressDescription.NumberOfDelen?.Trim()}";

smet.Contact = $"{Adress.UserContact.TypeOfContact.Value?.Trim()} : {Adress.UserContact.Value?.Trim()} : {Adress.UserContact.Description?.Trim()}";

smet.FIO = $"{Adress.ClientDetails.User.LastName?.Trim()} {Adress.ClientDetails.User.Name?.Trim()} {Adress.ClientDetails.User.Patronymic?.Trim()} ";

return smet;

}

else

{

return new MakeSmetaAll { success = false };

}

});

}

internal static object MakeDataForSprav(Guid idOrder)

{

return Run((db) =>

{

var dd = db.OrderInformation.Where(ee => ee.IdOrder == idOrder).FirstOrDefault();

if (dd != null)

{

int num = 1;

List<AllPremises> Premises = new List<AllPremises>();

foreach (var prem in dd.OrderMeasurements)

{

Premises.Add(new AllPremises

{

Desc = prem.Description,

Height = prem.Height,

idMeasurment = prem.idMeasurements,

lenght = prem.Lenght,

NameOf = prem.PremisesType.NameOfPremises,

number = num,

PCelling = prem.PCelling,

SFloor = prem.Pwalls,

SWalls = prem.Swalls,

Width = prem.Width,

PWalls = prem.Sfloor

});

num++;

}

MakeDataForSpravka makeDataForSpravka = new MakeDataForSpravka

{

AreaName = dd.AdressDescription.AreaName,

CityName = dd.AdressDescription.CiryName,

DateMakeOrder = dd.DateStart,

DateRozd = dd.ClientDetails.User.BirstDay,

DescContact = dd.UserContact.Description,

Description = dd.AdressDescription.Description,

Entrance = dd.AdressDescription.Entrance,

House = dd.AdressDescription.House,

LastName = dd.ClientDetails.User.LastName,

MicroAreaName = dd.AdressDescription.MicroAreaName,

Name = dd.ClientDetails.User.Name,

NumberOfDelen = dd.AdressDescription.NumberOfDelen,

Patronymic = dd.ClientDetails.User.Patronymic,

RegionName = dd.AdressDescription.RegionName,

StatusOfOrder = dd.Status,

Street = dd.AdressDescription.Street,

SummaOfOrder = dd.AllSumma,

TypeOfcontact = dd.UserContact.TypeOfContact.Value,

Value = dd.UserContact.Value,

success = true,

Premises = Premises

};

return makeDataForSpravka;

}

else

{

return new MakeDataForSpravka { success = false };

}

});

}

internal static object DeleteTask(Guid idTask, Guid idOrder)

{

return Run((db) =>

{

try

{

var task = db.OrderTasks.Where(ee => ee.IdTask == idTask).First();

if (task != null)

{

db.OrderTasks.Remove(task);

var mat = db.TaskMaterials.Where(ee => ee.idTask == idTask).AsEnumerable();

if (mat != null)

{

if (mat.Any())

{

db.TaskMaterials.RemoveRange(mat);

}

}

var Serv = db.TaskServis.Where(ee => ee.idTask == idTask).AsEnumerable();

if (Serv != null)

{

if (Serv.Any())

{

db.TaskServis.RemoveRange(Serv);

}

}

var Work = db.TaskWorker.Where(ee => ee.idTask == idTask).AsEnumerable();

if (Work != null)

{

if (Work.Any())

{

db.TaskWorker.RemoveRange(Work);

}

}

db.SaveChanges();

decimal summa = db.OrderTasks.Where(ee => ee.IdOrder == idOrder).Sum(ee => ee.SummaAboutTask) ?? default;

decimal Opl = db.OrderPayment.Where(ee => ee.IdOrder == idOrder).Sum(ee => ee.Summa) ?? default;

decimal Needd = summa - Opl;

return new SummaOfOrder { success = true, IdOrder = idOrder, summaOfOrder = summa, NeedPay = Needd };

}

else

{

return new SummaOfOrder { success = false };

}

}

catch (Exception ex)

{

return new SummaOfOrder { success = false, description = ex.ToString() };

}

});

}

internal static object MakeDataAboutAllOrderTask(Guid idOrder)

{

return Run((db) =>

{

var DataAboutTasks = db.OrderTasks.Where(e => e.IdOrder == idOrder).AsEnumerable();

if (DataAboutTasks.Any())

{

DataAboutTaskInOrder TasksData = new DataAboutTaskInOrder();

TasksData.InfTask = new List<TaskInf>();

foreach (var Task in DataAboutTasks)

{

TaskInf taskInf = new TaskInf();

taskInf.DateEnd = Task.DeadEnd;

taskInf.DateStart = Task.DateStart;

taskInf.Description = Task.Description;

taskInf.idTask = Task.IdTask;

taskInf.Summa = Task.SummaAboutTask;

TasksData.InfTask.Add(taskInf);

}

TasksData.success = true;

decimal summa = db.OrderTasks.Where(ee => ee.IdOrder == idOrder).Sum(ee => ee.SummaAboutTask) ?? default;

decimal Opl = db.OrderPayment.Where(ee => ee.IdOrder == idOrder).Sum(ee => ee.Summa) ?? default;

decimal Needd = summa - Opl;

TasksData.Ostatok = Needd;

return TasksData;

}

else

{

return new DataAboutTaskInOrder { success = false };

}

});

}

internal static object GetDataAboutTask(Guid idTask)

{

return Run((db) =>

{

var data = db.OrderTasks.Where(ee => ee.IdTask == idTask).FirstOrDefault();

if (data != null)

{

InformationAboutTask aboutTask = new InformationAboutTask();

aboutTask.success = true;

aboutTask.idOrder = data.IdOrder ?? default;

aboutTask.idTask = data.IdTask;

aboutTask.DateEnd = data.DeadEnd;

aboutTask.DateStart = data.DateStart;

aboutTask.Description = data.Description;

aboutTask.Summa = data.SummaAboutTask;

if (data.TaskMaterials != null)

{

if (data.TaskMaterials.Any())

{

aboutTask.InfAbMat = new MaterialInfTask();

aboutTask.InfAbMat.materialsInf = new List<TaskMaterial>();

foreach (var Mat in data.TaskMaterials)

{

TaskMaterial taskMaterial = new TaskMaterial

{

cost = Mat.Cost,

count = Mat.Count,

idMaterial = Mat.idMaterial ?? default,

idTaskMaterial = Mat.idMaterial ?? default,

NameOfMaterials = Mat.OurMaterials.NameOfMaterial,

};

aboutTask.InfAbMat.materialsInf.Add(taskMaterial);

}

}

}

if (data.TaskServis != null)

{

if (data.TaskServis.Any())

{

aboutTask.InfAbServ = new ServisesInfTask();

aboutTask.InfAbServ.ServisInf = new List<TaskServises>();

foreach (var Serv in data.TaskServis)

{

TaskServises taskServis = new TaskServises

{

cost = Serv.Cost,

count = Serv.Count,

idServis = Serv.idServis ?? default,

idTaskServises = Serv.IdTaskServises,

NameOfServises = Serv.OurServices.Nomination,

};

aboutTask.InfAbServ.ServisInf.Add(taskServis);

}

}

}

if (data.TaskWorker != null)

{

if (data.TaskWorker.Any())

{

aboutTask.InfAbWorkers = new WorkersInfTask();

aboutTask.InfAbWorkers.WorkerInf = new List<TaskWorker>();

foreach (var Worker in data.TaskWorker)

{

TaskWorker TaskWorker = new TaskWorker

{

FioOfWorker = $"{Worker.User.LastName?.Trim()} {Worker.User.Name?.Substring(0, 1)}.{Worker.User.Patronymic?.Substring(0, 1)}",

idTaskWorker = Worker.idCmbination,

idWorker = Worker.idWorker ?? default,

NameOfPost = Worker.User.WorkerDetails.EstabilismentPost.FirstOrDefault().WorkerPosts.NameOfPost,

Role = Worker.Role

};

aboutTask.InfAbWorkers.WorkerInf.Add(TaskWorker);

}

}

}

return aboutTask;

}

else

{

return new InformationAboutTask

{

success = false,

};

}

});

}

internal static object CreateNewTask(InformationAboutTask dataAboutTask)

{

return Run((db) =>

{

try

{

db.OrderTasks.AddOrUpdate(new Models.OrderTasks

{

DateStart = dataAboutTask.DateStart,

DeadEnd = dataAboutTask.DateEnd,

Description = dataAboutTask.Description,

IdOrder = dataAboutTask.idOrder,

IdTask = dataAboutTask.idTask,

SummaAboutTask = dataAboutTask.Summa

});

if (dataAboutTask.InfAbServ.ServisInf != null)

if (dataAboutTask.InfAbServ.ServisInf.Any())

{

foreach (var servInf in dataAboutTask.InfAbServ.ServisInf)

{

db.TaskServis.AddOrUpdate(new Models.TaskServis

{

idTask = dataAboutTask.idTask,

Cost = servInf.cost,

Count = servInf.count,

idServis = servInf.idServis,

IdTaskServises = servInf.idTaskServises

});

}

}

if (dataAboutTask.InfAbServ.DeleteServises != null)

if (dataAboutTask.InfAbServ.DeleteServises.Any())

{

foreach (var servInf in dataAboutTask.InfAbServ.DeleteServises)

{

var ss = db.TaskServis.Where(ee => ee.IdTaskServises == servInf).First();

if (ss != null)

db.TaskServis.Remove(ss);

}

}

if (dataAboutTask.InfAbMat.materialsInf != null)

if (dataAboutTask.InfAbMat.materialsInf.Any())

{

foreach (var MatInf in dataAboutTask.InfAbMat.materialsInf)

{

db.TaskMaterials.AddOrUpdate(new Models.TaskMaterials

{

Cost = MatInf.cost,

idMaterial = MatInf.idMaterial,

idTask = dataAboutTask.idTask,

idTaskMaterials = MatInf.idTaskMaterial,

Count = MatInf.count,

});

}

}

if (dataAboutTask.InfAbMat.DeleteMaterials != null)

if (dataAboutTask.InfAbMat.DeleteMaterials.Any())

{

foreach (var MatInf in dataAboutTask.InfAbMat.DeleteMaterials)

{

var ss = db.TaskMaterials.Where(ee => ee.idTaskMaterials == MatInf).First();

if (ss != null)

db.TaskMaterials.Remove(ss);

}

}

if (dataAboutTask.InfAbWorkers.WorkerInf != null)

if (dataAboutTask.InfAbWorkers.WorkerInf.Any())

{

foreach (var WorkInf in dataAboutTask.InfAbWorkers.WorkerInf)

{

db.TaskWorker.AddOrUpdate(new Models.TaskWorker

{

idTask = dataAboutTask.idTask,

idCmbination = WorkInf.idTaskWorker,

idWorker = WorkInf.idWorker,

Role = WorkInf.Role

});

}

}

if (dataAboutTask.InfAbWorkers.DeleteWorkers != null)

if (dataAboutTask.InfAbWorkers.DeleteWorkers.Any())

{

foreach (var WorkerInf in dataAboutTask.InfAbWorkers.DeleteWorkers)

{

var ss = db.TaskWorker.Where(ee => ee.idCmbination == WorkerInf).First();

if (ss != null)

db.TaskWorker.Remove(ss);

}

}

db.SaveChanges();

decimal summa = db.OrderTasks.Where(ee => ee.IdOrder == dataAboutTask.idOrder).Sum(ee => ee.SummaAboutTask) ?? default;

decimal Opl = db.OrderPayment.Where(ee => ee.IdOrder == dataAboutTask.idOrder).Sum(ee => ee.Summa) ?? default;

decimal Needd = summa - Opl;

db.OrderInformation.Where(ee => ee.IdOrder == dataAboutTask.idOrder).First().AllSumma = summa;

db.SaveChanges();

return new SummaOfOrder { IdOrder = dataAboutTask.idOrder, summaOfOrder = summa, success = true, NeedPay = Needd };

}

catch (Exception ex)

{

string ex1 = ex.ToString();

return new SummaOfOrder

{

success = false,

description = ex.ToString()

};

}

});

}

internal static object MakeDataAboutAllOrderPayment(Guid idOrder)

{

return Run((db) =>

{

var DataAboutAllPayment = db.OrderPayment.Where(e => e.IdOrder == idOrder).AsEnumerable();

if (DataAboutAllPayment.Any())

{

DataAboutPaymentInOrder dataAboutPaymentInOrder = new DataAboutPaymentInOrder();

dataAboutPaymentInOrder.InfPayment = new List<PaymentInf>();

foreach (var Payment in DataAboutAllPayment)

{

PaymentInf inf = new PaymentInf();

inf.Summa = Payment.Summa;

inf.Description = Payment.Description;

inf.idPayment = Payment.IdPayment;

inf.FioMake = $"{Payment.WorkerDetails.User.LastName?.Trim()} {Payment.WorkerDetails.User.Name?.Substring(0, 1)}.{Payment.WorkerDetails.User.Patronymic?.Substring(0, 1)}";

inf.DateOfMake = Payment.DateOfDoc;

dataAboutPaymentInOrder.InfPayment.Add(inf);

}

dataAboutPaymentInOrder.success = true;

decimal summa = db.OrderTasks.Where(ee => ee.IdOrder == idOrder).Sum(ee => ee.SummaAboutTask) ?? default;

decimal Opl = db.OrderPayment.Where(ee => ee.IdOrder == idOrder).Sum(ee => ee.Summa) ?? default;

decimal Needd = summa - Opl;

db.OrderInformation.Where(ee => ee.IdOrder == idOrder).First().AllSumma = summa;

dataAboutPaymentInOrder.NeedPay = Needd;

dataAboutPaymentInOrder.summaOfOrder = summa;

return dataAboutPaymentInOrder;

}

else

{

return new DataAboutPaymentInOrder { success = false };

}

});

}

}

}

Модуль RepairFlatRestApi. WorkWithContactDBController.cs

using RepairFlatRestApi.Models;

using RepairFlatRestApi.Models.DescriptionJSON;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using static RepairFlatRestApi.Models.ContactModel;

namespace RepairFlatRestApi.Controllers.OtherController

{

public class WorkWithContactDBController: DBController

{

internal static BaseResult CreaNewContact(InformationAboutContact newContact)

{

return Run((db) =>

{

try

{

db.UserContact.Add(new UserContact

{

DateAdd= newContact.DateAdd,

Description= newContact.Desctription,

id= newContact.idContact,

idType= newContact.idTypeOfContact,

idUser= newContact.idUser,

Value= newContact.Value

});

db.SaveChanges();

return new BaseResult()

{

success = true

};

}

catch (Exception ex)

{

return new BaseResult()

{

success = false,

description = ex.ToString()

};

}

});

}

internal static object UpdateContactData(InformationAboutContact updateContact)

{

return Run((db) =>

{

try

{

var updatedContact = db.UserContact.Where(e => e.id == updateContact.idContact).First();

if (updateContact != null)

{

updatedContact.idType = updateContact.idTypeOfContact;

updatedContact.idUser = updateContact.idUser;

updatedContact.Value = updateContact.Value;

updatedContact.Description = updateContact.Desctription;

}

db.SaveChanges();

return new BaseResult()

{

success = true

};

}

catch (Exception ex)

{

return new BaseResult()

{

success = false,

description = ex.ToString()

};

}

});

}

internal static object DeleteContact(Guid idContact)

{

return Run((db) =>

{

try

{

var contactInf = db.UserContact.Where(e => e.id == idContact).First();

var select = db.OrderInformation.Where(e => e.MainContactID == idContact).First();

if (select == null)

{

db.UserContact.Remove(contactInf);

return new BaseResult { success = true };

}

else

{

return new BaseResult { success = false,description="Необходимо удалить связь с заказами данного контакта!" };

}

}

catch(Exception ex)

{

return new BaseResult { success = false, description = $"Ошибка при работе {ex.Message}" };

}

});

}

internal static ListOfUserContactInf CreateListOfContactUser(Guid idUser)

{

return Run((db) =>

{

var que = db.UserContact.Where(e => e.idUser == idUser).AsEnumerable();

if (que != null)

{

List<ListOfContactUser> ListOfContact = new List<ListOfContactUser>();

foreach (var Contact in que)

{

ListOfContactUser contactUser = new ListOfContactUser

{

DateAdd = Contact.DateAdd,

idUser = Contact.idUser,

idContact = Contact.id,

Desctription = Contact.Description,

Value = Contact.Value,

idTypeOfContact = Contact.idType,

ValueTypeOfContact = Contact.TypeOfContact.Value

};

ListOfContact.Add(contactUser);

}

return new ListOfUserContactInf

{

success = true,

idUser = idUser,

listOfContact = ListOfContact

};

}

else

{

return new ListOfUserContactInf

{

success = false,

idUser = idUser,

description = "Данные о контактах не найдены"

};

}

});

}

}

}

Модуль RepairFlatRestApi. SomeEnums.cs

namespace RepairFlatRestApi.Models

{

public class SomeEnums

{

public static string[] FemaleType = new string[] { "Мужчина", "Женщина" };

public enum TypeOfAction

{

AddOrUpdate,

Update,

Delete

}

public enum TypeOfUser

{

/// <summary>

/// Клиент. Необходимо для мобильных приложений

/// </summary>

Cl,

/// <summary>

/// Администратор. Доступ ко всем только просмотр

/// </summary>

AD,

/// <summary>

/// Менеджер лезет везде кроме работы с финансами иди кадрами

/// </summary>

MG,

/// <summary>

/// Работник отдела кадров доступ только к работе с кадрами и все

/// </summary>

KW,

/// <summary>

/// Доступ к финансам и справочным данным

/// </summary>

BW,

/// <summary>

/// Видит все и делает все типо бог

/// </summary>

BB,

/// <summary>

/// Необходимо для мобилки если она будет но это не точно

/// </summary>

SW,

/// <summary>

/// Кандидат либо уволенный

/// </summary>

KD

}

public enum TypeOfSubs

{

Servises,

Premises,

Materials,

Contact,

Post

}

public enum TypeOfOperate

{

Adoption,

Permutation,

Firing

}

}

}

Модуль RepairFlatWPF. MakeSomeHelp.cs

using Newtonsoft.Json;

using RepairFlat.Model;

using RepairFlatWPF.Controller;

using RepairFlatWPF.Properties;

using RepairFlatWPF.UserControls;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Net.NetworkInformation;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

namespace RepairFlatWPF

{

/// <summary>

/// Выполнение некоторых повторяющихся событий

/// </summary>

public static class MakeSomeHelp

{

public static MessageBoxResult MSG(string Message, MessageBoxButton MsgBoxButton = MessageBoxButton.OK, MessageBoxImage MsgBoxImage = MessageBoxImage.None)

{

return MessageBox.Show(Message, Settings.Default.DefaultHeaderOfMessageBox, MsgBoxButton, MsgBoxImage);

}

public static object MakeDownloadByLink(string UrlOfDownload)

{

return BaseWorkWithServer.CatchErrorWithGet(UrlOfDownload, "GET", nameof(MakeLoading), nameof(MakeDownloadByLink));

}

public static bool MakePingToServer(string ServerAdress)

{

try

{

Ping ping = new Ping();

PingReply pingReply = null;

pingReply = ping.Send("ya.ru");

if (pingReply.Status != IPStatus.TimedOut)

{

return true;

}

else

{

return false;

}

}

catch (Exception ex)

{

MessageBox.Show(ex.ToString());

return false;

}

}

public static WorkWithDB.Tableses ReturnJsonOfTable()

{

var Resourses = Properties.Resources.ResourceManager.GetObject("DescriprionOfDB") as byte[];

string json = Encoding.UTF8.GetString(Resourses);

var ListOFTablesDescription = JsonConvert.DeserializeObject<WorkWithDB.Tableses>(json);

return ListOFTablesDescription;

}

public static void ChengeGridInMainWindow(UserControl controls)

{

((MainWindow)Application.Current.MainWindow).MainGrid.Children.Clear();

if (controls != null)

((MainWindow)Application.Current.MainWindow).MainGrid.Children.Add(controls);

}

public static void MakeShowLogining()

{

((MainWindow)Application.Current.MainWindow).ForLogin.Visibility = Visibility.Visible;

((MainWindow)Application.Current.MainWindow).ForLogin.Children.Clear();

((MainWindow)Application.Current.MainWindow).ForLogin.Children.Add(new LoginUserControl());

((MainWindow)Application.Current.MainWindow).ForLogin.Background = (System.Windows.Media.Brush)Application.Current.Resources["BackLogAndLoadColor"];

}

public static void MakeLoading(bool forWindow = true)

{

((MainWindow)Application.Current.MainWindow).ForLogin.Children.Clear();

((MainWindow)Application.Current.MainWindow).ForLogin.Children.Add(new MakeLoading(forWindow));

if (!forWindow)

((MainWindow)Application.Current.MainWindow).ForLogin.Visibility = Visibility.Visible;

if (forWindow)

((MainWindow)Application.Current.MainWindow).ForLogin.Background = (System.Windows.Media.Brush)Application.Current.Resources["GradientForLoading"];

}

public static void ShowMainGrid()

{

((MainWindow)Application.Current.MainWindow).ForLogin.Visibility = Visibility.Collapsed;

((MainWindow)Application.Current.MainWindow).ForLogin.Children.Clear();

}

public static void ChengeGridBackGroundStyle(string NameOfStyle)

{

((MainWindow)Application.Current.MainWindow).MainGrid.Background = (System.Windows.Media.Brush)Application.Current.Resources[NameOfStyle];

}

public static void GridChengeGridBackGroundStyle(string NameOfStyle)

{

((MainWindow)Application.Current.MainWindow).ForLogin.Background = (System.Windows.Media.Brush)Application.Current.Resources[NameOfStyle];

}

public static void DataGridMakeWork(UserControl controls)

{

((MainWindow)Application.Current.MainWindow).GridForContent.Children.Clear();

if (controls != null)

((MainWindow)Application.Current.MainWindow).GridForContent.Children.Add(controls);

}

public static void ShowOrCloseMenu(bool Open)

{

if (Open)

((MainWindow)Application.Current.MainWindow).GridMenu.Width = 300;

else

((MainWindow)Application.Current.MainWindow).GridMenu.Width = 0;

}

public static int SelectMaxValueinColumn(ref DataTable dataTable, string NameOfColumn)

{

int tempVariable = 0;

int highestNumber = dataTable.AsEnumerable()

.Where(x => int.TryParse(x.Field<string>(NameOfColumn), out tempVariable))

.Max(m => int.Parse(m.Field<string>(NameOfColumn)));

return highestNumber + 1;

}

public static object SelectedRowsInDataGrid(ref DataGrid DataGrid, int rowsNumber, int column = 0)

{

var ci = new DataGridCellInfo(DataGrid.Items[rowsNumber], DataGrid.Columns[column]);

var content = ci.Column.GetCellContent(ci.Item) as TextBlock;

if (content.Text != "")

{

if (content != null)

{

return content.Text;

}

else

{

return null;

}

}

else

{

return null;

}

}

public async static void UpdloadDataToServer(string Url, string Json)

{

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(Url, "POST", Json, nameof(BaseWorkWithServer), nameof(UpdloadDataToServer)));

var deserializedProduct = JsonConvert.DeserializeObject<BaseResult>(task.ToString());

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошикбка при работе {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

MakeSomeHelp.MSG("Операции над данными были произведены!", MsgBoxImage: MessageBoxImage.Information);

}

}

public static List<Tuple<int, Guid>> ListofId;

public static DataTable DataTableFromDataBase(SomeEnums.TypeOfSubs typeOfSubs)

{

DataTable DataAboutSomeSubInf = new DataTable();

ListofId = new List<Tuple<int, Guid>>();

if (typeOfSubs == SomeEnums.TypeOfSubs.Materials)

{//Материалы

foreach (string NameOfColumn in SomeEnums.MaterialSubs)

{

DataAboutSomeSubInf.Columns.Add(NameOfColumn);

}

string query = "select \* from OurMaterials";

var datafr = MakeWorkWirthDataBase.MakeSomeQueryWork(query, WorkWithTables: true);

if (datafr != null)

{

DataTable dataTable = new DataTable();

dataTable = datafr as DataTable;

for (int i = 0; i < dataTable.Rows.Count; i++)

{

DataRow rowsForInsert = DataAboutSomeSubInf.NewRow();

rowsForInsert[0] = i + 1;

rowsForInsert[1] = dataTable.Rows[i][1].ToString()?.Trim();

rowsForInsert[2] = dataTable.Rows[i][2].ToString()?.Trim();

rowsForInsert[3] = dataTable.Rows[i][4].ToString()?.Trim();

rowsForInsert[4] = dataTable.Rows[i][3].ToString()?.Trim();

rowsForInsert[5] = dataTable.Rows[i][5].ToString()?.Trim();

DataAboutSomeSubInf.Rows.Add(rowsForInsert);

ListofId.Add(new Tuple<int, Guid>(i + 1, Guid.Parse(dataTable.Rows[i][0].ToString())));

}

}

}

else if (typeOfSubs == SomeEnums.TypeOfSubs.Post)

{//Дожности

foreach (string NameOfColumn in SomeEnums.PostSubs)

{

DataAboutSomeSubInf.Columns.Add(NameOfColumn);

}

string query = "select \* from PostData";

var datafr = MakeWorkWirthDataBase.MakeSomeQueryWork(query, WorkWithTables: true);

if (datafr != null)

{

DataTable dataTable = new DataTable();

dataTable = datafr as DataTable;

for (int i = 0; i < dataTable.Rows.Count; i++)

{

DataRow rowsForInsert = DataAboutSomeSubInf.NewRow();

string MakeWork = "";

if (!string.IsNullOrEmpty(dataTable.Rows[i][3].ToString()))

{

MakeWork = Convert.ToInt32(dataTable.Rows[i][3].ToString()) == 1 ? "Да" : "Нет";

}

else

{

MakeWork = "Нет данных";

}

rowsForInsert[0] = i + 1;

rowsForInsert[1] = dataTable.Rows[i][1].ToString()?.Trim();

rowsForInsert[2] = dataTable.Rows[i][2].ToString()?.Trim();

rowsForInsert[3] = MakeWork;

DataAboutSomeSubInf.Rows.Add(rowsForInsert);

ListofId.Add(new Tuple<int, Guid>(i + 1, Guid.Parse(dataTable.Rows[i][0].ToString())));

}

}

}

else if (typeOfSubs == SomeEnums.TypeOfSubs.Servises)

{//Услуги

foreach (string NameOfColumn in SomeEnums.ServisesSubs)

{

DataAboutSomeSubInf.Columns.Add(NameOfColumn);

}

string query = "select \* from OurServices";

var datafr = MakeWorkWirthDataBase.MakeSomeQueryWork(query, WorkWithTables: true);

if (datafr != null)

{

DataTable dataTable = new DataTable();

dataTable = datafr as DataTable;

for (int i = 0; i < dataTable.Rows.Count; i++)

{

DataRow rowsForInsert = DataAboutSomeSubInf.NewRow();

rowsForInsert[0] = i + 1;

rowsForInsert[1] = dataTable.Rows[i][1].ToString()?.Trim();

rowsForInsert[2] = dataTable.Rows[i][2].ToString()?.Trim();

rowsForInsert[3] = dataTable.Rows[i][3].ToString()?.Trim();

rowsForInsert[4] = dataTable.Rows[i][4].ToString()?.Trim();

DataAboutSomeSubInf.Rows.Add(rowsForInsert);

ListofId.Add(new Tuple<int, Guid>(i + 1, Guid.Parse(dataTable.Rows[i][0].ToString())));

}

}

}

else if (typeOfSubs == SomeEnums.TypeOfSubs.Premises)

{//Помещения

foreach (string NameOfColumn in SomeEnums.PremisesSubs)

{

DataAboutSomeSubInf.Columns.Add(NameOfColumn);

}

string query = "select \* from PremisesType";

var datafr = MakeWorkWirthDataBase.MakeSomeQueryWork(query, WorkWithTables: true);

if (datafr != null)

{

DataTable dataTable = new DataTable();

dataTable = datafr as DataTable;

for (int i = 0; i < dataTable.Rows.Count; i++)

{

DataRow rowsForInsert = DataAboutSomeSubInf.NewRow();

rowsForInsert[0] = i + 1;

rowsForInsert[1] = dataTable.Rows[i][1].ToString()?.Trim();

rowsForInsert[2] = dataTable.Rows[i][2].ToString()?.Trim();

DataAboutSomeSubInf.Rows.Add(rowsForInsert);

ListofId.Add(new Tuple<int, Guid>(i + 1, Guid.Parse(dataTable.Rows[i][0].ToString())));

}

}

}

else if (typeOfSubs == SomeEnums.TypeOfSubs.Contact)

{//Типы контактов

foreach (string NameOfColumn in SomeEnums.ContactSubs)

{

DataAboutSomeSubInf.Columns.Add(NameOfColumn);

}

string query = "select \* from ContactType";

var datafr = MakeWorkWirthDataBase.MakeSomeQueryWork(query, WorkWithTables: true);

if (datafr != null)

{

DataTable dataTable = new DataTable();

dataTable = datafr as DataTable;

for (int i = 0; i < dataTable.Rows.Count; i++)

{

DataRow rowsForInsert = DataAboutSomeSubInf.NewRow();

rowsForInsert[0] = i + 1;

rowsForInsert[1] = dataTable.Rows[i][1].ToString()?.Trim();

rowsForInsert[2] = dataTable.Rows[i][2].ToString()?.Trim();

rowsForInsert[3] = dataTable.Rows[i][3].ToString()?.Trim();

DataAboutSomeSubInf.Rows.Add(rowsForInsert);

ListofId.Add(new Tuple<int, Guid>(i + 1, Guid.Parse(dataTable.Rows[i][0].ToString())));

}

}

}

return DataAboutSomeSubInf;

}

}

}

Модуль RepairFlatWPF. MakeWorkWirthDataBase.cs

using System;

using System.Collections.Generic;

using System.Data;

using System.Data.SQLite;

using System.IO;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Windows;

namespace RepairFlatWPF.Controller

{

public class MakeWorkWirthDataBase

{

public static string NameOfSqlFile = "MakeHelp.sqlite";

public static string PathToDB = "";

public static void MakeFilePathAndCheck()

{

string PathToDataBase = Path.Combine(Path.GetDirectoryName(Assembly.GetExecutingAssembly().Location), NameOfSqlFile);

if (!File.Exists(PathToDataBase))

{

SQLiteConnection.CreateFile(PathToDataBase);

}

PathToDB = $"Data Source = {PathToDataBase}";

CheckAndMakeTableInDB();

}

public static void CheckAndMakeTableInDB()

{

var ListOFTablesDescription = MakeSomeHelp.ReturnJsonOfTable();

foreach (var Table in ListOFTablesDescription.Tables)

{

CreateTableIfNotExist(Table.NameOfTable, Table.ColumnOfTable);

}

}

public static void CreateTableIfNotExist(string NameOfTable, List<WorkWithDB.ColumnOfTable> columns)

{

if (!IsTableExist(NameOfTable))

{

string NameOfPK = "";

StringBuilder CreateTableQuery = new StringBuilder($"CREATE TABLE {NameOfTable} (");

WorkWithDB.ColumnOfTable columnOfTableLast = columns.Last();

foreach (var colum in columns)

{

if (colum.Equals(columnOfTableLast))

{

CreateTableQuery.Append($" '{colum.NameOfCol}' {colum.TypeOfCol}");

}

else

{

CreateTableQuery.Append($" '{colum.NameOfCol}' {colum.TypeOfCol},");

}

if (colum.IsPk)

{

NameOfPK = colum.NameOfCol;

}

}

string EndOfCreate = !string.IsNullOrEmpty(NameOfPK) ? $", PRIMARY KEY(`{NameOfPK}`));" : $");";

CreateTableQuery.Append(EndOfCreate);

MakeSomeQueryWork(CreateTableQuery.ToString());

}

}

public static bool IsTableExist(string NameOfTable)

{

string query = "SELECT name FROM sqlite\_master WHERE type='table' AND name=@tableName;";

SQLiteParameter[] parameters = new SQLiteParameter[1];

parameters[0] = new SQLiteParameter("@tableName", NameOfTable);

string[] whatSelect = new string[] { "name" };

var makeWork = MakeSomeQueryWork(query, parameters: parameters, SelectedColumns: whatSelect);

if (makeWork != null)

{

string[] dd = makeWork as string[];

if (dd.Length != 0)

{

try

{

if (dd[0] == NameOfTable)

{

return true;

}

}

catch

{

return false;

}

}

return true;

}

return false;

}

public static object MakeSomeQueryWork(string query, string[] SelectedColumns = null, bool WorkWithTables = false, SQLiteParameter[] parameters = null)

{

try

{

using (SQLiteConnection connection = new SQLiteConnection(PathToDB))

{

connection.Open();

using (SQLiteCommand command = new SQLiteCommand(query, connection))

{

if (parameters != null)

{

command.Parameters.AddRange(parameters);

}

if (!WorkWithTables)

{

using (SQLiteDataReader reader = command.ExecuteReader())

{

if (SelectedColumns != null)

{

if (reader.HasRows)

{

string[] result = new string[reader.FieldCount];

if (reader.Read())

{

for (int i = 0; i < reader.FieldCount; i++)

{

if (reader.GetName(i) == SelectedColumns[i])

{

result[i] = reader[reader.GetName(i)].ToString();

}

}

}

else

{

return null;

}

return result;

}

else

{

return null;

}

}

else

{

return true;

}

}

}

else

{

DataTable ResultTable = new DataTable("Result");

using (SQLiteDataAdapter dataAdapter = new SQLiteDataAdapter(command))

{

dataAdapter.Fill(ResultTable);

if (ResultTable.Rows.Count != 0)

{

return ResultTable;

}

else

{

return null;

}

}

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Произошла ошибка при работе с базой данных <{ex.Message}>");

}

return null;

}

public static void Run(Action<SQLiteCommand> dbAction)

{

try

{

using (SQLiteConnection connection = new SQLiteConnection(PathToDB))

{

connection.Open();

using (SQLiteCommand command = new SQLiteCommand(connection))

{

using (var transaction = connection.BeginTransaction())

{

dbAction(command);

transaction.Commit();

}

}

}

}

catch (Exception ex)

{

MessageBox.Show($"Произошла ошибка при работе с базой данных <{ex.Message}>");

}

}

}

}

}

Модуль RepairFlatWPF. SomeEnums.cs

namespace RepairFlatWPF

{

public class SomeEnums

{

#region Место описани пунктов Combobox

public static string[] StatusOfOrder = new string[] { "Принят", "В исполнении", "Выполнен", "Отмен" };

public static string[] RypeOfSearchOrder = new string[] { "Все данные", "Дата начала", "Фамилия клиента" };

public static string[] TypeOfElementOfPremises = new string[] { "Оконо", "Дверь" };

public static string[] FemaleType = new string[] { "Мужчина", "Женщина" };

public static string[] TypeOfElement = new string[] { "Окно", "Дверь" };

public static string[] RoleOfWorker = new string[] { "Прораб", "Рабочий" };

public static string[] ContactTableDesc = new string[] { "Номер", "Тип", "Значение", "Описание" };

public static string[] ClientTables = new string[] { "Номер", "Имя", "Фамилия", "Отчество", "Пол", "Описание" };

public static string[] OrderMainTable = new string[] { "Номер", "Дата начала", "Статус", "ФИО Клиента", "Контакная информация", "Данные об адресе", "Сумма", "Описание" };

public static string[] MeasurmentMainTable = new string[] { "Номер", "Наименование", "Описание", "Длина", "Ширина", "Высота", "P стен", "P пола", "S стен", "S пола" };

public static string[] DataAboutElement = new string[] { "Номер", "Тип элемента", "Длина", "Ширина", "Толщина", "P элемента", "Описание" };

public static string[] WorkerTables = new string[] { "Номер", "Фамилия", "Имя", "Отчество", "Пол", "Дата рождения" };

public static string[] WorkerTablesRedact = new string[] { "Номер", "Фамилия", "Имя", "Отчество", "Пол", "Дата рождения", "Должность", "Оклад" };

public static string[] PostSubs = new string[] { "Номер", "Название должности", "Базовый оклад", "Выполняет заказы" };

public static string[] MaterialSubs = new string[] { "Номер", "Название материала", "Еденица измерения", "Тип материала", "Стоимость", "Описание" };

public static string[] ServisesSubs = new string[] { "Номер", "Название услуги", "Тип сервиса", "Цена", "Описание" };

public static string[] ContactSubs = new string[] { "Номер", "Название", "Описание", "Регулярное выражение" };

public static string[] PremisesSubs = new string[] { "Номер", "Название", "Описание" };

public static string[] WorkerTask = new string[] { "Номер", "ФИО работника", "Роль" };

public static string[] ServisesTask = new string[] { "Номер", "Наименование услуги", "Количество", "Стоимость", "Описание" };

public static string[] ServisesMaterials = new string[] { "Номер", "Наименование услуги", "Количество", "Стоимость", "Описание" };

public static string[] PayInf = new string[] { "Номер", "Дата создания", "Сумма оплаты", "Кто создал", "Описание" };

#region Описание задания

public static string[] TaskTable = new string[] { "Номер", "Цена", "Описание", "Дата начала", "Планируемое время завершения", };

public static string[] TaskMaterialTable = new string[] { "Номер", "Название", "Количество", "Стоимость", "Сумма" };

public static string[] TaskServisTable = new string[] { "Номер", "Название", "Количество", "Стоимость", "Сумма" };

public static string[] TaskWorkerTable = new string[] { "Номер", "ФИО работника", "Должность", "Роль работника" };

public enum TypeOfAction

{

AddOrUpdate,

Update,

Delete

}

public enum TypeOfConrols

{

/// <summary>

/// Работа с окном

/// </summary>

Window,

/// <summary>

/// Работа в пределах главного окна

/// </summary>

UserControl

}

public enum TypeOfUser

{

/// <summary>

/// Клиент. Необходимо для мобильных приложений

/// </summary>

Cl,

/// <summary>

/// Администратор. Доступ ко всем только просмотр

/// </summary>

AD,

/// <summary>

/// Менеджер лезет везде кроме работы с финансами иди кадрами

/// </summary>

MG,

/// <summary>

/// Работник отдела кадров доступ только к работе с кадрами и все

/// </summary>

KW,

/// <summary>

/// Доступ к финансам и справочным данным

/// </summary>

BW,

/// <summary>

/// Видит все и делает все типо бог

/// </summary>

BB,

/// <summary>

/// Необходимо для мобилки если она будет но это не точно

/// </summary>

SW,

/// <summary>

/// Кандидат либо уволенный

/// </summary>

KD

}

public enum TypeOfSubs

{

Servises,

Premises,

Materials,

Contact,

Post

}

public enum TypeOfSelect

{

Servises,

Premises

}

public enum TypeOfUserNeed

{

KD,

All,

ForOrder,

ForRedact

}

public enum TypeOfOperate

{

Adoption,

Permutation,

Firing

}

}

}

Модуль RepairFlatWPF. MakeLoading.xaml.cs

using Newtonsoft.Json;

using RepairFlat.Model;

using RepairFlatWPF.Controller;

using System;

using System.Data;

using System.Data.SQLite;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Controls;

using System.Windows.Media.Animation;

namespace RepairFlatWPF.UserControls

{

/// <summary>

/// Interaction logic for MakeLoading.xaml

/// </summary>

public partial class MakeLoading : UserControl

{

bool forAll = true;

public MakeLoading(bool ForAllWindow = true)

{

forAll = ForAllWindow;

InitializeComponent();

((Storyboard)FindResource("WaitStoryboard")).Begin();

DescriptionOfWork.Content = "Определение даты последнего обновления ...";

MakeDownload();

}

public async void MakeDownload()

{

await Task.Run(() => MakeWorkWirthDataBase.MakeFilePathAndCheck());

string CheckLastDate = "Select Max(idUpdate), DateOfUpdate from DateOfLastUpdate;";

var MaxDate = Task.Run(() => MakeWorkWirthDataBase.MakeSomeQueryWork(CheckLastDate, WorkWithTables: true));

object ServisesDownload = null;

object ContaktsDownload = null;

object PremisesDownload = null;

object MaterialDownload = null;

object PostDownload = null;

bool withOutData = false;

if (MaxDate.Result != null)

{

DataTable infAbDate = MaxDate.Result as DataTable;

if (infAbDate.Rows.Count != 0)

{//Берем с датой

DescriptionOfWork.Content = "Получение данных об услугах с сервера ...";

string someDate = infAbDate.Rows[0]["DateOfUpdate"].ToString();

if (someDate.Trim() != "") {

if (DateTime.TryParse(someDate, out DateTime buf))

someDate = buf.AddHours(-1).ToString("dd.MM.yyyy HH:mm");

#region Получение данных от сервера

DescriptionOfWork.Content = "Получение данных об услугах с сервера ...";

ServisesDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/servises/get?dateofclientlastupdate={someDate}")); //Скачивание данных о услугах

ContaktsDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/contact/get?dateofclientlastupdate={someDate}")); //Скачивание данных о контактах

DescriptionOfWork.Content = "Получение данных о типах помещений с сервера ...";

PremisesDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/premises/get?dateofclientlastupdate={someDate}")); //Скачивание данных о типах помещений

DescriptionOfWork.Content = "Получение данных о материалах с сервера ...";

MaterialDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/material/get?dateofclientlastupdate={someDate}")); //Скачивание данных о материалах

PostDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/post/get?dateofclientlastupdate={someDate}")); //Скачивание данных о материалах

#endregion

}

else

{

withOutData = true;

}

}

else

{

withOutData = true;

}

}

else

{

withOutData = true;

}

if (withOutData)

{

#region Получение данных от сервера

DescriptionOfWork.Content = "Получение данных об услугах с сервера ...";

ServisesDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/servises/get")); //Скачивание данных о услугах

ContaktsDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/contact/get")); //Скачивание данных о контактах

DescriptionOfWork.Content = "Получение данных о типах помещений с сервера ...";

PremisesDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/premises/get")); //Скачивание данных о типах помещений

DescriptionOfWork.Content = "Получение данных о материалах с сервера ...";

MaterialDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/material/get")); //Скачивание данных о материалах

PostDownload = await Task.Run(() => MakeDownloadByLink($"api/substring/post/get")); //Скачивание данных о должностях

#endregion

}

#region Преобразование данных в читаемое состояние

DescriptionOfWork.Content = "Преобразование над данными ...";

MakeSubs.ServisesMake ServisesMake = JsonConvert.DeserializeObject<MakeSubs.ServisesMake>(ServisesDownload.ToString());

MakeSubs.ContactsMake ContactsMake = JsonConvert.DeserializeObject<MakeSubs.ContactsMake>(ContaktsDownload.ToString());

MakeSubs.PremisesMake PremisesMake = JsonConvert.DeserializeObject<MakeSubs.PremisesMake>(PremisesDownload.ToString());

MakeSubs.MaterialsMake MaterialsMake = JsonConvert.DeserializeObject<MakeSubs.MaterialsMake>(MaterialDownload.ToString());

MakeSubs.PostMake PostMake = JsonConvert.DeserializeObject<MakeSubs.PostMake>(PostDownload.ToString());

#endregion

DescriptionOfWork.Content = "Запись данных в локальную базу данных ...";

#region Загрузка данных в локальную БД

if (ServisesMake.success)

await Task.Run(() => ServisesUpdlocToDB(ServisesMake));

if (MaterialsMake.success)

await Task.Run(() => MaterialsUpdlocToDB(MaterialsMake));

if (PremisesMake.success)

await Task.Run(() => PremisesUpdlocToDB(PremisesMake));

if (ContactsMake.success)

await Task.Run(() => ContactUpdlocToDB(ContactsMake));

if (PostMake.success)

await Task.Run(() => PostUpdlocToDB(PostMake));

#endregion

((Storyboard)FindResource("WaitStoryboard")).Stop();

DescriptionOfWork.Content = "Данные обновлены";

MakeSomeHelp.ShowMainGrid();

}

private void PostUpdlocToDB(MakeSubs.PostMake postMake)

{

string MakeQuery = "Insert into PostData (idPost,NameOfPost,BaseWage,MakeWork) Values (@idPost,@NameOfPost,@BaseWage,@MakeWork) On CONFLICT(idPost) DO UPDATE SET idPost=@idPost,NameOfPost=@NameOfPost,BaseWage=@BaseWage,MakeWork=@MakeWork;";

MakeWorkWirthDataBase.Run((command) =>

{

try

{

bool MakeSmF = false;

if (postMake.listOfPost != null)

if (postMake.listOfPost.Count() != 0)

{

MakeSmF = true;

foreach (var PostUpdate in postMake.listOfPost)

{

SQLiteParameter[] parameters = new SQLiteParameter[4];

parameters[0] = new SQLiteParameter("@idPost", PostUpdate.idPost.ToString());

parameters[1] = new SQLiteParameter("@NameOfPost", PostUpdate.NameOfPost);

parameters[2] = new SQLiteParameter("@BaseWage", PostUpdate.BaseWage);

parameters[3] = new SQLiteParameter("@MakeWork", Convert.ToInt32(PostUpdate.MakeWork));

command.Parameters.AddRange(parameters);

command.CommandText = MakeQuery;

command.ExecuteNonQuery();

}

}

if (postMake.ListOfDeletePost != null)

if (postMake.ListOfDeletePost.Count() != 0)

{

MakeSmF = true;

string QueryForDelete = "Delete From PostData where idPost=@idPost;";

foreach (var DeleteThings in postMake.ListOfDeletePost)

{

SQLiteParameter[] parameters = new SQLiteParameter[1];

parameters[0] = new SQLiteParameter("@idPost", DeleteThings.idGuid.ToString());

command.Parameters.AddRange(parameters);

command.CommandText = QueryForDelete;

command.ExecuteNonQuery();

}

}

if (MakeSmF)

{

string QueryForHistory = "Insert into DateOfLastUpdate (TypeOfSubs,DateOfUpdate) Values (@TypeOfSubs,@DateOfUpdate)";

SQLiteParameter[] parameters = new SQLiteParameter[2];

parameters[0] = new SQLiteParameter("@TypeOfSubs", SomeEnums.TypeOfSubs.Post.ToString());

parameters[1] = new SQLiteParameter("@DateOfUpdate", postMake.DateOfMakeAnswer.ToString("dd.MM.yyyy HH:mm"));

command.Parameters.AddRange(parameters);

command.CommandText = QueryForHistory;

command.ExecuteNonQuery();

}

}

catch (Exception ex)

{

MakeSomeHelp.MSG(ex.ToString());

}

});

}

public object MakeDownloadByLink(string UrlOfDownload)

{

return BaseWorkWithServer.CatchErrorWithGet(UrlOfDownload, "GET", nameof(MakeLoading), nameof(MakeDownloadByLink));

}

}

}

Модуль RepairFlatWPF. AddUserControl.xaml.cs

using Newtonsoft.Json;

using RepairFlat.Model;

using RepairFlatWPF.Model;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using static RepairFlat.Model.PersonDesctiption;

namespace RepairFlatWPF.UserControls

{

public partial class AddUserControl : UserControl

{

#region Переменные

Guid? idUser;

bool NewData = true;

DataTable TableOfContactInformation;

BaseWindow window;

List<Tuple<int, Guid, Guid?>> IdContactAndTypeInTable = new List<Tuple<int, Guid, Guid?>>();

#endregion

#region Констурктор и обработчик

public AddUserControl(Guid? idUser, ref BaseWindow baseWindow, bool redact = false)

{

InitializeComponent();

this.idUser = idUser;

window = baseWindow;

foreach (string Type in SomeEnums.FemaleType)

{

Female.Items.Add(Type);

}

TableOfContactInformation = new DataTable();

foreach (string NameOfColumn in SomeEnums.ContactTableDesc)

{

TableOfContactInformation.Columns.Add(NameOfColumn);

}

DataGrid.ItemsSource = TableOfContactInformation.DefaultView;

if (redact)

{

AddBtn.Content = "Редактировать";

NewData = false;

makeloadingListOfContact();

DeleteElement.IsEnabled = false;

makeLoadingDataAboutContact();

}

else

{

idUser = Guid.NewGuid();

}

}

private async void makeLoadingDataAboutContact()

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/user/get?idUser={idUser}"));

CreateNewClient DataAboutClient = JsonConvert.DeserializeObject<CreateNewClient>(InformationFromServer.ToString());

if (DataAboutClient.idUser != new Guid())

{

Name.Text = DataAboutClient.Name?.Trim();

Famil.Text = DataAboutClient.Lastname?.Trim();

Patronymic.Text = DataAboutClient.Patronymic?.Trim();

Pasport.Text = DataAboutClient.Pasport?.Trim();

Female.SelectedItem = SomeEnums.FemaleType[DataAboutClient.Female ?? default];

DateOfBirsd.SelectedDate = DataAboutClient.Birstday;

Description.Text = DataAboutClient.Desc?.Trim();

}

}

public async void makeloadingListOfContact()

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/contact/getusercontact?idUser={idUser}"));

ContactModel.ListOfUserContactInf listOfUserContactInf = JsonConvert.DeserializeObject<ContactModel.ListOfUserContactInf>(InformationFromServer.ToString());

if (listOfUserContactInf != null)

{

int number = 1;

foreach (var contact in listOfUserContactInf.listOfContact)

{

DataRow NewContactData = TableOfContactInformation.NewRow();

NewContactData[0] = number;

NewContactData[1] = contact.ValueTypeOfContact?.Trim();

NewContactData[2] = contact.Value?.Trim();

NewContactData[3] = contact.Desctription?.Trim();

TableOfContactInformation.Rows.Add(NewContactData);

IdContactAndTypeInTable.Add(new Tuple<int, Guid, Guid?>(number, contact.idContact, contact.idTypeOfContact));

number++;

}

DataGrid.ItemsSource = TableOfContactInformation.DefaultView;

}

}

private void RetutnBTN\_Click(object sender, RoutedEventArgs e)

{

window.Close();

}

private async void AddBtn\_Click(object sender, RoutedEventArgs e)

{

if (CheckFields())

{

//Тут добавление

List<ContactModel.InformationAboutContact> ContactInformatio = new List<ContactModel.InformationAboutContact>();

if (TableOfContactInformation.Rows.Count != 0)

{

for (int i = 0; i < TableOfContactInformation.Rows.Count; i++)

{

Guid idContact = IdContactAndTypeInTable.Where(e2 => e2.Item1 == Convert.ToInt32(TableOfContactInformation.Rows[i][0])).Select(e1 => e1.Item2).First();

Guid? idType = IdContactAndTypeInTable.Where(e2 => e2.Item1 == Convert.ToInt32(TableOfContactInformation.Rows[i][0])).Select(e1 => e1.Item3).First();

ContactModel.InformationAboutContact contact = new ContactModel.InformationAboutContact()

{

DateAdd = DateTime.Now,

Desctription = TableOfContactInformation.Rows[i][3].ToString(),

idContact = idContact,

idTypeOfContact = idType,

idUser = idUser,

Value = TableOfContactInformation.Rows[i][2].ToString()

};

ContactInformatio.Add(contact);

}

}

else

{

ContactInformatio = null;

}

CreateNewClient createNewClient = new CreateNewClient

{

idUser = idUser,

Birstday = DateOfBirsd.SelectedDate.Value,

Desc = Description.Text.Trim(),

Female = Female.SelectedIndex,

Lastname = Famil.Text.Trim(),

Name = Name.Text.Trim(),

Pasport = Pasport.Text.Trim(),

Patronymic = Patronymic.Text.Trim(),

TypeOfUser = SomeEnums.TypeOfUser.Cl.ToString(),

ListOfContact = ContactInformatio

};

string Json = JsonConvert.SerializeObject(createNewClient);

string urlSend = "api/user/create";

AddBtn.Content = "Ожидайте...";

RetutnBTN.Content = "Ожидайте...";

AddBtn.IsEnabled = false;

RetutnBTN.IsEnabled = false;

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(urlSend, "POST", Json, nameof(BaseWorkWithServer), nameof(AddBtn\_Click)));

var deserializedProduct = JsonConvert.DeserializeObject<BaseResult>(task.ToString());

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошикбка при создании пользователя {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

MakeSomeHelp.MSG("Операции над данными произведены!", MsgBoxImage: MessageBoxImage.Information);

}

window.Close();

}

}

private void AddElement\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Добавление контактной информации");

baseWindow.MakeOpen(new AddContactUserConrol(idUser, ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

var contactInformation = SaveSomeData.SomeObject as ContactModel.InformationAboutContact;

SaveSomeData.SomeObject = null;

SaveSomeData.MakeSomeOperation = false;

int Number = TableOfContactInformation.Rows.Count == 0 ? 1 : MakeSomeHelp.SelectMaxValueinColumn(ref TableOfContactInformation, SomeEnums.ContactTableDesc[0]);

DataRow newContact = TableOfContactInformation.NewRow();

newContact[0] = Number;

newContact[1] = contactInformation.NameOfValue;

newContact[2] = contactInformation.Value;

newContact[3] = contactInformation.Desctription;

TableOfContactInformation.Rows.Add(newContact);

IdContactAndTypeInTable.Add(new Tuple<int, Guid, Guid?>(Number, contactInformation.idContact, contactInformation.idTypeOfContact));

}

}

private void RedactElement\_Click(object sender, RoutedEventArgs e)

{

int index = DataGrid.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

ContactModel.InformationAboutContact aboutContact = new ContactModel.InformationAboutContact();

for (int i = 0; i < TableOfContactInformation.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfContactInformation.Rows[i][0].ToString()) == numberOfRows)

{

Guid idContact = IdContactAndTypeInTable.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

Guid? idType = IdContactAndTypeInTable.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item3).First();

aboutContact = new ContactModel.InformationAboutContact

{

idContact = idContact,

idTypeOfContact = idType,

idUser = idUser,

Value = TableOfContactInformation.Rows[i][2].ToString(),

Desctription = TableOfContactInformation.Rows[i][3].ToString(),

NameOfValue = TableOfContactInformation.Rows[i][1].ToString(),

Number = numberOfRows

};

}

}

BaseWindow baseWindow = new BaseWindow("Редактирование контактной информации");

baseWindow.MakeOpen(new AddContactUserConrol(idUser, ref baseWindow, aboutContact));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

ContactModel.InformationAboutContact dataUpdated = SaveSomeData.SomeObject as ContactModel.InformationAboutContact;

SaveSomeData.SomeObject = null;

for (int i = 0; i < TableOfContactInformation.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfContactInformation.Rows[i][0].ToString()) == dataUpdated.Number)

{

TableOfContactInformation.Rows[i][1] = dataUpdated.NameOfValue;

TableOfContactInformation.Rows[i][2] = dataUpdated.Value;

TableOfContactInformation.Rows[i][3] = dataUpdated.Desctription;

var data = IdContactAndTypeInTable.Single(e1 => e1.Item1 == dataUpdated.Number);

IdContactAndTypeInTable.Remove(data);

IdContactAndTypeInTable.Add(new Tuple<int, Guid, Guid?>(dataUpdated.Number, dataUpdated.idContact, dataUpdated.idTypeOfContact));

}

}

}

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать строку для редактирования", MsgBoxImage: MessageBoxImage.Error);

}

}

private void DeleteElement\_Click(object sender, RoutedEventArgs e)

{

int index = DataGrid.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

for (int i = 0; i < TableOfContactInformation.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfContactInformation.Rows[i][0].ToString()) == numberOfRows)

{

TableOfContactInformation.Rows[i].Delete();

var data = IdContactAndTypeInTable.Single(e1 => e1.Item1 == numberOfRows);

IdContactAndTypeInTable.Remove(data);

}

}

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать строку для удаления", MsgBoxImage: MessageBoxImage.Error);

}

}

}

}

Модуль RepairFlatWPF. ShowDataForPayment.xaml.cs

using Newtonsoft.Json;

using RepairFlatWPF.Model;

using System;

using System.IO;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Documents;

using static RepairFlatWPF.Model.DescMakePayment;

namespace RepairFlatWPF.UserControls.MoneyInformation

{

/// <summary>

/// Interaction logic for ShowDataForPayment.xaml

/// </summary>

public partial class ShowDataForPayment : UserControl

{

#region Переменные

DataAboutPayment InfAboutPayment = new DataAboutPayment();

#endregion

#region Конструктор

public ShowDataForPayment()

{

InitializeComponent();

MakeDataFromPayment();

}

#endregion

#region Обработчики событий

private void ExtionPayment\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Работа с данными для оплаты");

baseWindow.MakeOpen(new MakeDataForPayment(ref baseWindow, InfAboutPayment));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

MakeDataFromPayment();

}

}

private void Print\_Click(object sender, RoutedEventArgs e)

{

if (MakeSomeHelp.MSG("Вы дейсвительно хотите создать шаблон квитанции для оплаты ", MsgBoxImage: MessageBoxImage.Question, MsgBoxButton: MessageBoxButton.OKCancel) == MessageBoxResult.OK)

{

var Director = System.IO.Path.Combine(Environment.CurrentDirectory, "Temp");

if (!Directory.Exists(Director))

{

Directory.CreateDirectory(Director);

}

string NameOfFile = System.IO.Path.Combine(Director, "oplata.dotx");

if (!File.Exists(NameOfFile))

File.WriteAllBytes(NameOfFile, RepairFlatWPF.Properties.Resources.OplataDoc);

using (var application = new NetOffice.WordApi.Application { Visible = true })

{

using (var document = application.Documents.Add(NameOfFile))

{

string text = $"Получатель платежа: {InfAboutPayment.NameOfRecipient?.Trim()} {Environment.NewLine}ИНН: {InfAboutPayment.InnOfOrganization?.Trim()} {Environment.NewLine}КПП: {InfAboutPayment.KppOfOrganization?.Trim()} {Environment.NewLine}Банк получатель: {InfAboutPayment.BankOfPayment?.Trim()} {Environment.NewLine}Расчетный счет: {InfAboutPayment.CheckingAcount?.Trim()} {Environment.NewLine}БИК: {InfAboutPayment.BIK?.Trim()} {Environment.NewLine}УИН: {InfAboutPayment.YIN?.Trim()}";

var InfrormationForPayment = document.Bookmarks["InfrormationForPayment"].Range;

InfrormationForPayment.Text = text;

var InfrormationForPayment1 = document.Bookmarks["InfrormationForPayment1"].Range;

InfrormationForPayment1.Text = text;

}

application.Activate();

}

}

}

private void Return\_Click(object sender, RoutedEventArgs e)

{

MakeSomeHelp.DataGridMakeWork(new FinansInformation());

}

#endregion

#region Прочие обработчики

private async void MakeDataFromPayment()

{

var DataDle = await Task.Run(() => MakeSomeHelp.MakeDownloadByLink($"api/payment/getdata"));

var DataAbInf = JsonConvert.DeserializeObject<DataAboutPayment>(DataDle.ToString());

if (DataAbInf.success)

{

InfAboutPayment = DataAbInf;

ExtionPayment.Content = "Редактировать";

TextRange doc = new TextRange(IformationAb.Document.ContentStart, IformationAb.Document.ContentEnd);

doc.Text = $"Текущие данные:{Environment.NewLine}";

doc.Text += $"Были созданы: <{DataAbInf.NameOfWorkerMake?.Trim()}> {Environment.NewLine}";

doc.Text += $"Наименование получателя: <{DataAbInf.NameOfRecipient?.Trim()}> {Environment.NewLine}";

doc.Text += $"ИНН организации: <{DataAbInf.InnOfOrganization?.Trim()}> {Environment.NewLine}";

doc.Text += $"КПП организации: <{DataAbInf.KppOfOrganization?.Trim()}> {Environment.NewLine}";

doc.Text += $"Банк получатель: <{DataAbInf.BankOfPayment?.Trim()}> {Environment.NewLine}";

doc.Text += $"Расчетный счет: <{DataAbInf.CheckingAcount?.Trim()}> {Environment.NewLine}";

doc.Text += $"БИК: <{DataAbInf.BIK}> {Environment.NewLine}";

doc.Text += $"УИН: <{DataAbInf.YIN}> {Environment.NewLine}";

doc.Text += $"Дата создания/последнего обновления: <{DataAbInf.DateOfMake.Value.ToString("dd.MM.yyyy")}> {Environment.NewLine}";

}

else

{

InfAboutPayment = null;

ExtionPayment.Content = "Указать";

TextRange doc = new TextRange(IformationAb.Document.ContentStart, IformationAb.Document.ContentEnd);

doc.Text = "Необходимо указать данные для оплаты клиентам";

}

}

#endregion

}

}

Модуль RepairFlatWPF. AddNewTaskInOrderUserControl.xaml.cs

using Newtonsoft.Json;

using RepairFlatWPF.Model;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using static RepairFlatWPF.Model.OrderDesc;

namespace RepairFlatWPF.UserControls.OrderWork.AddInfromationUserControl

{

public partial class AddNewTaskInOrderUserControl : UserControl

{

#region Переменные

Guid idOrder;

Guid idTask;

bool NewTask = true;

BaseWindow Window;

DataTable TableOfMaterials, TableOfServises, TableOfWorker;

int MaxServis = 1, MaxMaterial = 1, MaxWorker = 1;

List<Tuple<int, Guid, Guid>> ListOfMateriaslId, ListOfServisesId, ListOfWorkerId;

List<Guid> ListOfDeleteServ, ListOfDeleteMat, ListOfDeleteWorker;

TextBlock textBoxSumma, textBoxost;

#endregion

#region Конструктор

public AddNewTaskInOrderUserControl(ref BaseWindow baseWindow, ref TextBlock textBoxSumma, ref TextBlock textBoxost, Guid idOrder, Guid idTask = new Guid())

{

InitializeComponent();

this.textBoxSumma = textBoxSumma;

this.textBoxost = textBoxost;

this.idOrder = idOrder;

ShowNeededPage(ForMainData);

Window = baseWindow;

MakePreparationTableColumnName();

if (idTask != new Guid())

{

this.idTask = idTask;

NewTask = false;

MakeDataByid();

ExtionButton.Content = "Редактировать";

}

else

{

this.idTask = Guid.NewGuid();

}

}

#endregion

#region Обработки основной части формы

private void SelectTabsClick(object sender, RoutedEventArgs e)

{

int index = int.Parse(((Button)e.Source).Uid);

switch (index)

{

case 0:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowNeededPage(ForMainData);

break;

case 1:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowNeededPage(ForServises);

break;

case 2:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowNeededPage(ForMaterials);

break;

case 3:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowNeededPage(ForWorker);

break;

}

}

private void RetutnBTN\_Click(object sender, RoutedEventArgs e)

{

if (MakeSomeHelp.MSG("Вы действительно хотите выйти из создания задания?", MessageBoxButton.OKCancel, MessageBoxImage.Question) == MessageBoxResult.OK)

{

Window.Close();

}

}

private async void ExtionButton\_Click(object sender, RoutedEventArgs e)

{

if (MakeCheckData())

{

InformationAboutTask informationAboutTask = new InformationAboutTask();

informationAboutTask.idOrder = idOrder;

informationAboutTask.idTask = idTask;

informationAboutTask.InfAbMat = new MaterialInfTask();

informationAboutTask.InfAbWorkers = new WorkersInfTask();

informationAboutTask.InfAbServ = new ServisesInfTask();

informationAboutTask.InfAbMat.materialsInf = new List<TaskMaterial>();

informationAboutTask.InfAbWorkers.WorkerInf = new List<TaskWorker>();

informationAboutTask.InfAbServ.ServisInf = new List<TaskServises>();

if (!NewTask)

{

if (ListOfDeleteServ.Any())

{

informationAboutTask.InfAbServ.DeleteServises = new List<Guid>();

informationAboutTask.InfAbServ.DeleteServises = ListOfDeleteServ;

}

if (ListOfDeleteMat.Any())

{

informationAboutTask.InfAbMat.DeleteMaterials = new List<Guid>();

informationAboutTask.InfAbMat.DeleteMaterials = ListOfDeleteMat;

}

if (ListOfDeleteWorker.Any())

{

informationAboutTask.InfAbWorkers.DeleteWorkers = new List<Guid>();

informationAboutTask.InfAbWorkers.DeleteWorkers = ListOfDeleteWorker;

}

}

informationAboutTask.DateEnd = DateEnd.SelectedDate;

informationAboutTask.DateStart = DateStart.SelectedDate;

informationAboutTask.Description = Description.Text?.Trim();

for (int i = 0; i < TableOfMaterials.Rows.Count; i++)

{

var dd = ListOfMateriaslId.Where(ee => ee.Item1 == Convert.ToInt32(TableOfMaterials.Rows[i][0].ToString())).First();

TaskMaterial taskMaterial = new TaskMaterial

{

cost = Convert.ToDouble(TableOfMaterials.Rows[i][3].ToString()),

count = Convert.ToDouble(TableOfMaterials.Rows[i][2].ToString()),

idMaterial = dd.Item2,

idTaskMaterial = dd.Item3,

NameOfMaterials = TableOfMaterials.Rows[i][1].ToString()

};

informationAboutTask.InfAbMat.materialsInf.Add(taskMaterial);

}

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

var dd = ListOfServisesId.Where(ee => ee.Item1 == Convert.ToInt32(TableOfServises.Rows[i][0].ToString())).First();

TaskServises taskServises = new TaskServises

{

cost = Convert.ToDouble(TableOfServises.Rows[i][3].ToString()),

count = Convert.ToDouble(TableOfServises.Rows[i][2].ToString()),

idServis = dd.Item2,

idTaskServises = dd.Item3,

NameOfServises = TableOfServises.Rows[i][1].ToString()

};

informationAboutTask.InfAbServ.ServisInf.Add(taskServises);

}

for (int i = 0; i < TableOfWorker.Rows.Count; i++)

{

var dd = ListOfWorkerId.Where(ee => ee.Item1 == Convert.ToInt32(TableOfWorker.Rows[i][0].ToString())).First();

TaskWorker TaskWorker = new TaskWorker

{

FioOfWorker = TableOfWorker.Rows[i][1].ToString(),

idWorker = dd.Item2,

Role = TableOfWorker.Rows[i][3].ToString(),

idTaskWorker = dd.Item3,

};

informationAboutTask.InfAbWorkers.WorkerInf.Add(TaskWorker);

}

informationAboutTask.Summa = Convert.ToDecimal(AllSumma.Text.ToString());

string Json = JsonConvert.SerializeObject(informationAboutTask);

string urlSend = "api/order/create/task";

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(urlSend, "POST", Json, nameof(BaseWorkWithServer), nameof(ExtionButton\_Click)));

var deserializedProduct = JsonConvert.DeserializeObject<SummaOfOrder>(task.ToString());

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошикбка при работе {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

textBoxost.Text = deserializedProduct.NeedPay.ToString();

textBoxSumma.Text = deserializedProduct.summaOfOrder.ToString();

MakeSomeHelp.MSG("Операции над данными были произведены!", MsgBoxImage: MessageBoxImage.Information);

}

Window.Close();

}

}

#endregion

#region Обработки при работе с услугами

private void AddServises\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Добавление данных об услугах");

baseWindow.MakeOpen(new AddServisesInOrder(ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAbout = SaveSomeData.SomeObject as TaskServises;

SaveSomeData.SomeObject = null;

int numberOfRows = 0;

if (ListOfServisesId.Any())

{

var sel = ListOfServisesId.Where(e2 => e2.Item2 == dataAbout.idServis);

if (sel.Any())

{

numberOfRows = sel.Select(e1 => e1.Item1).First();

}

}

double cost = dataAbout.cost ?? default;

double count = dataAbout.count ?? default;

double summa = cost \* count;

if (numberOfRows != 0)

{//Обновление

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfServises.Rows[i][0].ToString()) == numberOfRows)

{

double countTek = Convert.ToDouble(TableOfServises.Rows[i][2].ToString());

TableOfServises.Rows[i][1] = dataAbout.NameOfServises;

TableOfServises.Rows[i][2] = dataAbout.count + countTek;

TableOfServises.Rows[i][3] = dataAbout.cost;

TableOfServises.Rows[i][4] = cost \* (dataAbout.count + countTek);

}

}

}

else

{//Добавление

DataRow dataRow = TableOfServises.NewRow();

dataRow[0] = MaxServis;

dataRow[1] = dataAbout.NameOfServises;

dataRow[2] = dataAbout.count;

dataRow[3] = dataAbout.cost;

dataRow[4] = summa;

TableOfServises.Rows.Add(dataRow);

ListOfServisesId.Add(new Tuple<int, Guid, Guid>(MaxServis, dataAbout.idServis, Guid.NewGuid()));

MaxServis++;

}

MakeSumma();

}

}

private void EditServises\_Click(object sender, RoutedEventArgs e)

{

int index = DataAboutServises.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataAboutServises, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

Guid idServis = ListOfServisesId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

TaskServises InfAbServ = null;

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfServises.Rows[i][0].ToString()) == numberOfRows)

{

InfAbServ = new TaskServises

{

idServis = idServis,

cost = Convert.ToDouble(TableOfServises.Rows[i][3].ToString()),

count = Convert.ToInt32(TableOfServises.Rows[i][2].ToString()),

NameOfServises = TableOfServises.Rows[i][1].ToString()

};

}

}

BaseWindow baseWindow = new BaseWindow("Редактирование данных о предоставляемых услугах");

baseWindow.MakeOpen(new AddServisesInOrder(ref baseWindow, InfAbServ));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var updatedData = SaveSomeData.SomeObject as TaskServises;

SaveSomeData.SomeObject = null;

double cost = updatedData.cost ?? default;

double count = updatedData.count ?? default;

double summa = cost \* count;

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfServises.Rows[i][0].ToString()) == numberOfRows)

{

TableOfServises.Rows[i][1] = updatedData.NameOfServises;

TableOfServises.Rows[i][2] = updatedData.count;

TableOfServises.Rows[i][3] = updatedData.cost;

TableOfServises.Rows[i][4] = summa;

}

}

}

MakeSumma();

}

}

else

{

MakeSomeHelp.MSG("Не выбрана услуга для редактирование!", MsgBoxImage: MessageBoxImage.Hand);

}

}

private void DeleteSerises\_Click(object sender, RoutedEventArgs e)

{

int index = DataAboutServises.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataAboutServises, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

Guid idServis = ListOfServisesId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

if (!NewTask)

{

ListOfDeleteServ.Add(ListOfServisesId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item3).First());

}

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfServises.Rows[i][0].ToString()) == numberOfRows)

{

TableOfServises.Rows.Remove(TableOfServises.Rows[i]);

}

}

var delete = ListOfServisesId.Where(e1 => e1.Item2 == idServis).First();

ListOfServisesId.Remove(delete);

MakeSumma();

}

}

else

{

MakeSomeHelp.MSG("Не выбрана услуга для удаления!", MsgBoxImage: MessageBoxImage.Hand);

}

}

#endregion

#region Обработки про материалы

private void AddMaterials\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Добавление данных о материалах");

baseWindow.MakeOpen(new AddInfromationAboutMaterials(ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAboutMaterials = SaveSomeData.SomeObject as TaskMaterial;

SaveSomeData.SomeObject = null;

int numberOfRows = 0;

if (ListOfMateriaslId.Any())

{

var sele = ListOfMateriaslId.Where(e2 => e2.Item2 == dataAboutMaterials.idMaterial);

if (sele.Any())

numberOfRows = sele.Select(e1 => e1.Item1).First();

}

double cost = dataAboutMaterials.cost ?? default;

double count = dataAboutMaterials.count ?? default;

double summa = cost \* count;

if (numberOfRows != 0)

{//Обновление

for (int i = 0; i < TableOfMaterials.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfMaterials.Rows[i][0].ToString()) == numberOfRows)

{

double countTek = Convert.ToDouble(TableOfMaterials.Rows[i][2].ToString());

TableOfMaterials.Rows[i][1] = dataAboutMaterials.NameOfMaterials;

TableOfMaterials.Rows[i][2] = dataAboutMaterials.count + countTek;

TableOfMaterials.Rows[i][3] = dataAboutMaterials.cost;

TableOfMaterials.Rows[i][4] = cost \* (dataAboutMaterials.count + countTek);

}

}

}

else

{//Добавление

DataRow dataRow = TableOfMaterials.NewRow();

dataRow[0] = MaxMaterial;

dataRow[1] = dataAboutMaterials.NameOfMaterials;

dataRow[2] = dataAboutMaterials.count;

dataRow[3] = dataAboutMaterials.cost;

dataRow[4] = summa;

TableOfMaterials.Rows.Add(dataRow);

ListOfMateriaslId.Add(new Tuple<int, Guid, Guid>(MaxMaterial, dataAboutMaterials.idMaterial, Guid.NewGuid()));

MaxMaterial++;

}

MakeSumma();

}

}

private void EditMaterials\_Click(object sender, RoutedEventArgs e)

{

int index = DataAboutMaterials.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataAboutMaterials, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

Guid idMaterial = ListOfMateriaslId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

TaskMaterial InfAbMat = null;

for (int i = 0; i < TableOfMaterials.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfMaterials.Rows[i][0].ToString()) == numberOfRows)

{

InfAbMat = new TaskMaterial

{

idMaterial = idMaterial,

cost = Convert.ToDouble(TableOfMaterials.Rows[i][3].ToString()),

count = Convert.ToInt32(TableOfMaterials.Rows[i][2].ToString()),

NameOfMaterials = TableOfMaterials.Rows[i][1].ToString()

};

}

}

BaseWindow baseWindow = new BaseWindow("Редактирование данных о материале");

baseWindow.MakeOpen(new AddInfromationAboutMaterials(ref baseWindow, InfAbMat));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var updatedData = SaveSomeData.SomeObject as TaskMaterial;

SaveSomeData.SomeObject = null;

double cost = updatedData.cost ?? default;

double count = updatedData.count ?? default;

double summa = cost \* count;

for (int i = 0; i < TableOfMaterials.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfMaterials.Rows[i][0].ToString()) == numberOfRows)

{

TableOfMaterials.Rows[i][1] = updatedData.NameOfMaterials;

TableOfMaterials.Rows[i][2] = updatedData.count;

TableOfMaterials.Rows[i][3] = updatedData.cost;

TableOfMaterials.Rows[i][4] = summa;

}

}

}

MakeSumma();

}

}

else

{

MakeSomeHelp.MSG("Не выбран материал для редактирования!", MsgBoxImage: MessageBoxImage.Hand);

}

}

private void DeleteMaterials\_Click(object sender, RoutedEventArgs e)

{

int index = DataAboutServises.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataAboutServises, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

Guid idMaterial = ListOfMateriaslId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

if (!NewTask)

{

ListOfDeleteMat.Add(ListOfMateriaslId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item3).First());

}

for (int i = 0; i < TableOfMaterials.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfMaterials.Rows[i][0].ToString()) == numberOfRows)

{

TableOfMaterials.Rows.Remove(TableOfMaterials.Rows[i]);

}

}

var delete = ListOfMateriaslId.Where(e1 => e1.Item2 == idMaterial).First();

ListOfMateriaslId.Remove(delete);

MakeSumma();

}

}

else

{

MakeSomeHelp.MSG("Не выбран материал для удаления!", MsgBoxImage: MessageBoxImage.Hand);

}

}

#endregion

#region Обработки при работе с работниками

private void SelectWorker\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Добавление новых работников в задание");

baseWindow.MakeOpen(new AddWorkerInTask(ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAbout = SaveSomeData.SomeObject as TaskWorker;

SaveSomeData.SomeObject = null;

int numberOfRows = 0;

if (ListOfWorkerId.Any())

{

var sele = ListOfWorkerId.Where(e2 => e2.Item2 == dataAbout.idWorker);

if (sele.Any())

numberOfRows = sele.Select(e1 => e1.Item1).First();

}

if (numberOfRows != 0)

{

MakeSomeHelp.MSG("Работник уже есть в задании", MsgBoxImage: MessageBoxImage.Hand);

}

else

{

bool err = false;

for (int i = 0; i < TableOfWorker.Rows.Count; i++)

{

if (TableOfWorker.Rows[i][3].ToString() == SomeEnums.RoleOfWorker[0] && SomeEnums.RoleOfWorker[0] == dataAbout.Role)

{

MakeSomeHelp.MSG("В заказе не может быть больше 1 прораба", MsgBoxImage: MessageBoxImage.Hand);

err = true;

}

}

if (!err)

{

DataRow dataRow = TableOfWorker.NewRow();

dataRow[0] = MaxWorker;

dataRow[1] = dataAbout.FioOfWorker;

dataRow[2] = dataAbout.NameOfPost;

dataRow[3] = dataAbout.Role;

TableOfWorker.Rows.Add(dataRow);

ListOfWorkerId.Add(new Tuple<int, Guid, Guid>(MaxWorker, dataAbout.idWorker, Guid.NewGuid()));

MaxWorker++;

}

}

}

}

private void EditSelectWorker\_Click(object sender, RoutedEventArgs e)

{

int index = DataAboutWorker.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataAboutWorker, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

Guid idWorker = ListOfWorkerId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

TaskWorker InfAbWorker = null;

for (int i = 0; i < TableOfWorker.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfWorker.Rows[i][0].ToString()) == numberOfRows)

{

InfAbWorker = new TaskWorker

{

idWorker = idWorker,

FioOfWorker = TableOfWorker.Rows[i][1].ToString(),

NameOfPost = TableOfWorker.Rows[i][2].ToString(),

Role = TableOfWorker.Rows[i][3].ToString(),

};

}

}

BaseWindow baseWindow = new BaseWindow("Редактирование данных о работниках");

baseWindow.MakeOpen(new AddWorkerInTask(ref baseWindow, InfAbWorker));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var updatedData = SaveSomeData.SomeObject as TaskWorker;

SaveSomeData.SomeObject = null;

bool isBrigadeHave = false;

for (int i = 0; i < TableOfWorker.Rows.Count; i++)

{

if (TableOfWorker.Rows[i][3].ToString() == SomeEnums.RoleOfWorker[0] && SomeEnums.RoleOfWorker[0] == updatedData.Role)

{

MakeSomeHelp.MSG("В заказе не может быть больше 1 прораба", MsgBoxImage: MessageBoxImage.Hand);

isBrigadeHave = true;

}

}

if (!isBrigadeHave)

{

var RowsHaveWorker = ListOfServisesId.Where(e2 => e2.Item2 == updatedData.idWorker).Select(e1 => e1.Item1);

if (RowsHaveWorker.Any())

{

MakeSomeHelp.MSG("Работник уже есть в задании!", MsgBoxImage: MessageBoxImage.Hand);

}

else

{

for (int i = 0; i < TableOfWorker.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfWorker.Rows[i][0].ToString()) == numberOfRows)

{

TableOfWorker.Rows[i][1] = updatedData.FioOfWorker;

TableOfWorker.Rows[i][2] = updatedData.NameOfPost;

TableOfWorker.Rows[i][3] = updatedData.Role;

}

}

}

}

}

}

}

else

{

MakeSomeHelp.MSG("Не выбрана услуга для редактирование!", MsgBoxImage: MessageBoxImage.Hand);

}

}

private void DeleteWorker\_Click(object sender, RoutedEventArgs e)

{

int index = DataAboutWorker.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataAboutWorker, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

Guid idWorker = ListOfWorkerId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

if (NewTask)

{

for (int i = 0; i < TableOfWorker.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfWorker.Rows[i][0].ToString()) == numberOfRows)

{

TableOfWorker.Rows.Remove(TableOfWorker.Rows[i]);

}

}

var delete = ListOfWorkerId.Where(e1 => e1.Item2 == idWorker).First();

ListOfWorkerId.Remove(delete);

}

else

{//TODO Удаление при редактировании задании

ListOfDeleteWorker.Add(ListOfWorkerId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item3).First());

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

if (Convert.ToInt32(TableOfWorker.Rows[i][0].ToString()) == numberOfRows)

{

TableOfWorker.Rows.Remove(TableOfWorker.Rows[i]);

}

}

var delete = ListOfWorkerId.Where(e1 => e1.Item2 == idWorker).First();

ListOfWorkerId.Remove(delete);

}

}

}

else

{

MakeSomeHelp.MSG("Не выбрана работник для исплючения!", MsgBoxImage: MessageBoxImage.Hand);

}

}

#endregion

#region Вспомогательные

#region При работе с основной формой

private void ShowNeededPage(Grid grid)

{

ForMainData.Visibility = Visibility.Collapsed;

ForServises.Visibility = Visibility.Collapsed;

ForMaterials.Visibility = Visibility.Collapsed;

ForWorker.Visibility = Visibility.Collapsed;

grid.Visibility = Visibility.Visible;

}

private async void MakeDataByid()

{

var DataDle = await Task.Run(() => MakeSomeHelp.MakeDownloadByLink($"api/order/get/task?idTask={idTask}"));

var DataAbTask = JsonConvert.DeserializeObject<InformationAboutTask>(DataDle.ToString());

if (DataAbTask.success)

{

DateStart.SelectedDate = DataAbTask.DateStart;

DateEnd.SelectedDate = DataAbTask.DateEnd;

Description.Text = DataAbTask.Description?.Trim();

AllSumma.Text = DataAbTask.Summa.ToString();

if (DataAbTask.InfAbWorkers != null)

{

if (DataAbTask.InfAbWorkers.WorkerInf.Any())

{

foreach (var Worker in DataAbTask.InfAbWorkers.WorkerInf)

{

DataRow dataRow = TableOfWorker.NewRow();

dataRow[0] = MaxWorker;

dataRow[1] = Worker.FioOfWorker?.Trim();

dataRow[2] = Worker.NameOfPost?.Trim();

dataRow[3] = Worker.Role?.Trim();

TableOfWorker.Rows.Add(dataRow);

ListOfWorkerId.Add(new Tuple<int, Guid, Guid>(MaxWorker, Worker.idWorker, Worker.idTaskWorker));

MaxWorker++;

}

}

}

if (DataAbTask.InfAbMat != null)

{

if (DataAbTask.InfAbMat.materialsInf.Any())

{

foreach (var Material in DataAbTask.InfAbMat.materialsInf)

{

double cost = Material.cost ?? default;

double count = Material.count ?? default;

double summa = cost \* count;

DataRow dataRow = TableOfMaterials.NewRow();

dataRow[0] = MaxMaterial;

dataRow[1] = Material.NameOfMaterials?.Trim();

dataRow[2] = Material.count;

dataRow[3] = Material.cost;

dataRow[4] = summa;

TableOfMaterials.Rows.Add(dataRow);

ListOfMateriaslId.Add(new Tuple<int, Guid, Guid>(MaxMaterial, Material.idMaterial, Material.idTaskMaterial));

MaxMaterial++;

}

}

}

if (DataAbTask.InfAbServ != null)

{

if (DataAbTask.InfAbServ.ServisInf.Any())

{

foreach (var Serv in DataAbTask.InfAbServ.ServisInf)

{

double cost = Serv.cost ?? default;

double count = Serv.count ?? default;

double summa = cost \* count;

DataRow dataRow = TableOfServises.NewRow();

dataRow[0] = MaxServis;

dataRow[1] = Serv.NameOfServises?.Trim();

dataRow[2] = Serv.count;

dataRow[3] = Serv.cost;

dataRow[4] = summa;

TableOfServises.Rows.Add(dataRow);

ListOfServisesId.Add(new Tuple<int, Guid, Guid>(MaxServis, Serv.idServis, Serv.idTaskServises));

MaxServis++;

}

}

}

}

}

#endregion

private void MakePreparationTableColumnName()

{

TableOfMaterials = new DataTable();

TableOfServises = new DataTable();

TableOfWorker = new DataTable();

ListOfMateriaslId = new List<Tuple<int, Guid, Guid>>();

ListOfServisesId = new List<Tuple<int, Guid, Guid>>();

ListOfWorkerId = new List<Tuple<int, Guid, Guid>>();

ListOfDeleteMat = new List<Guid>();

ListOfDeleteServ = new List<Guid>();

ListOfDeleteWorker = new List<Guid>();

foreach (string NameOfMat in SomeEnums.TaskMaterialTable)

{

TableOfMaterials.Columns.Add(NameOfMat);

}

DataAboutMaterials.ItemsSource = TableOfMaterials.DefaultView;

foreach (string NameOfMat in SomeEnums.TaskServisTable)

{

TableOfServises.Columns.Add(NameOfMat);

}

DataAboutServises.ItemsSource = TableOfServises.DefaultView;

foreach (string NameOfMat in SomeEnums.TaskWorkerTable)

{

TableOfWorker.Columns.Add(NameOfMat);

}

DataAboutWorker.ItemsSource = TableOfWorker.DefaultView;

}

private bool MakeCheckData()

{

if (!DateStart.SelectedDate.HasValue)

{

MakeSomeHelp.MSG("Необходимо указать дату начала", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (!DateEnd.SelectedDate.HasValue)

{

MakeSomeHelp.MSG("Необходимо указать планируемую дату окончания", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (DateStart.SelectedDate > DateEnd.SelectedDate)

{

MakeSomeHelp.MSG("Дата начала не может быть больше даты окончания", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (TableOfServises.Rows.Count == 0)

{

if (TableOfWorker.Rows.Count == 0)

{

MakeSomeHelp.MSG("Для создания задания необходимо иметь какие либо услуги и работников, которые должны выполнить", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

else

{

MakeSomeHelp.MSG("Для создания задания необходимо иметь какие либо услуги", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

}

if (TableOfWorker.Rows.Count == 0)

{

MakeSomeHelp.MSG("Укажите работников для выполнения задания", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

return true;

}

private void MakeSumma()

{

double summa = 0;

for (int i = 0; i < TableOfMaterials.Rows.Count; i++)

{

summa = summa + Convert.ToDouble(TableOfMaterials.Rows[i][4].ToString());

}

for (int i = 0; i < TableOfServises.Rows.Count; i++)

{

summa = summa + Convert.ToDouble(TableOfServises.Rows[i][4].ToString());

}

AllSumma.Text = summa.ToString();

}

#endregion

}

}

Модуль RepairFlatWPF. AddServisesInOrder.xaml.cs

using RepairFlatWPF.Model;

using System;

using System.Data;

using System.Windows;

using System.Windows.Controls;

using static RepairFlatWPF.Model.OrderDesc;

namespace RepairFlatWPF.UserControls.OrderWork.AddInfromationUserControl

{

/// <summary>

/// Interaction logic for AddServisesInOrder.xaml

/// </summary>

public partial class AddServisesInOrder : UserControl

{

#region Переменные

Guid idServis;

bool IsSelect = false;

int Count;

decimal cost;

BaseWindow window;

#endregion

#region Обработка событий

public AddServisesInOrder(ref BaseWindow baseWindow, object InfromationAboutServis = null)

{

InitializeComponent();

window = baseWindow;

if (InfromationAboutServis != null)

{

AddServis.Content = "Редактировать";

var dataAbout = InfromationAboutServis as TaskServises;

idServis = dataAbout.idServis;

NameOfServis.Text = dataAbout.NameOfServises;

cost = Convert.ToDecimal(dataAbout.cost);

Cost.Text = cost.ToString();

Count = Convert.ToInt32(dataAbout.count);

CountOfServis.Text = Count.ToString();

IsSelect = true;

SelectServis.IsEnabled = false;

}

}

private void SelectServis\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Выбор услуги");

baseWindow.MakeOpen(new SettingsAndSubsInf.SelectSomeSubs(ref baseWindow, SomeEnums.TypeOfSubs.Servises));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

IsSelect = true;

idServis = SaveSomeData.idSubs;

SaveSomeData.idSubs = new Guid();

var row = SaveSomeData.SomeObject as DataRow;

SaveSomeData.SomeObject = null;

cost = decimal.Parse(row[3].ToString());

NameOfServis.Text = row[1].ToString()?.Trim();

Cost.Text = cost.ToString();

}

}

private void Return\_Click(object sender, RoutedEventArgs e)

{

window.Close();

}

private void AddServis\_Click(object sender, RoutedEventArgs e)

{

if (CheckFields())

{

TaskServises taskServises = new TaskServises

{

NameOfServises = NameOfServis.Text,

cost = Convert.ToDouble(cost),

count = Count,

idServis = idServis

};

SaveSomeData.SomeObject = taskServises;

SaveSomeData.MakeSomeOperation = true;

window.Close();

}

}

#endregion

#region Дополнительно

private bool CheckFields()

{

bool result = true;

if (!IsSelect)

{

MakeSomeHelp.MSG("Необходимо выбрать услугу");

result = false;

}

if (!int.TryParse(CountOfServis.Text.Trim(), out Count) && IsSelect)

{

MakeSomeHelp.MSG("Необходимо указать количество и оно должно быть целым числом", MsgBoxImage: MessageBoxImage.Hand);

result = false;

}

return result;

}

#endregion

}

}

Модуль RepairFlatWPF. AddWorkerInTask.xaml.cs

using RepairFlatWPF.Model;

using RepairFlatWPF.UserControls.WorkerInformation.KadrWork;

using System;

using System.Windows;

using System.Windows.Controls;

using static RepairFlatWPF.Model.OrderDesc;

namespace RepairFlatWPF.UserControls.OrderWork.AddInfromationUserControl

{

/// <summary>

/// Interaction logic for AddWorkerInTask.xaml

/// </summary>

public partial class AddWorkerInTask : UserControl

{

#region Переменные

Guid IdWorker;

bool WorkerSel = false;

string NameOfPost;

BaseWindow window;

#endregion

#region Конструктор

public AddWorkerInTask(ref BaseWindow baseWindow, object InfAbWorker = null)

{

InitializeComponent();

window = baseWindow;

foreach (string Role in SomeEnums.RoleOfWorker)

{

RoleOfWorker.Items.Add(Role);

}

if (InfAbWorker != null)

{

var dd = InfAbWorker as TaskWorker;

NameOfWorker.Text = dd.FioOfWorker?.Trim();

int ind = Array.IndexOf(SomeEnums.RoleOfWorker, dd.Role);

RoleOfWorker.SelectedIndex = ind;

IdWorker = dd.idWorker;

WorkerSel = true;

SetWorkers.Content = "Редактировать";

NameOfPost = dd.NameOfPost;

SetWorkers.IsEnabled = false;

}

}

#endregion

#region Обработчики событий

private void SetWorkers\_Click(object sender, RoutedEventArgs e)

{

if (MakeCheck())

{

SaveSomeData.MakeSomeOperation = true;

TaskWorker taskWorker = new TaskWorker

{

FioOfWorker = NameOfWorker.Text,

idWorker = IdWorker,

NameOfPost = NameOfPost,

Role = RoleOfWorker.Text,

};

SaveSomeData.SomeObject = taskWorker;

window.Close();

}

}

private void SelectWorker\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Выбор работника");

baseWindow.MakeOpen(new ShowAllWorkers(ref baseWindow, SomeEnums.TypeOfUserNeed.ForOrder));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

IdWorker = SaveSomeData.idSubs;

SaveSomeData.idSubs = new Guid();

var row = SaveSomeData.SomeObject as System.Data.DataRow;

SaveSomeData.SomeObject = null;

NameOfWorker.Text = $"{row[1]?.ToString().Trim()} {row[2]?.ToString().Substring(0, 1)}.{row[3]?.ToString().Substring(0, 1)}.";

NameOfPost = row[6]?.ToString();

WorkerSel = true;

}

}

private void Return\_Click(object sender, RoutedEventArgs e)

{

window.Close();

}

#endregion

#region Прочие обработки

private bool MakeCheck()

{

if (!WorkerSel)

{

MakeSomeHelp.MSG("Необходимо указать работника", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (RoleOfWorker.SelectedIndex == -1)

{

MakeSomeHelp.MSG("Необходимо указать роль работника", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

return true;

}

#endregion

}

}

Модуль RepairFlatWPF. AditionalTablesUserControl.xaml.cs

using Newtonsoft.Json;

using System;

using System.IO;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using static RepairFlatWPF.Model.OrderDesc;

namespace RepairFlatWPF.UserControls.OrderWork

{

/// <summary>

/// Interaction logic for AditionalTablesUserControl.xaml

/// </summary>

public partial class AditionalTablesUserControl : UserControl

{

Guid idOrder;

public AditionalTablesUserControl(Guid idOrder)

{

InitializeComponent();

this.idOrder = idOrder;

WhatPrint.Items.Add("Отдельно каждое задание");

WhatPrint.Items.Add("Вместе все данные ");

}

private async void MakeSmeta\_Click(object sender, RoutedEventArgs e)

{

if (WhatPrint.SelectedIndex != -1)

{

if (WhatPrint.SelectedIndex == 0)

{//Отдельно по заданиям

}

else if (WhatPrint.SelectedIndex == 1)

{//Вместе по заданиям

var DataDle = await Task.Run(() => MakeSomeHelp.MakeDownloadByLink($"api/Order/doc/smeta1?idOrder={idOrder}"));

var InfForSmeta = JsonConvert.DeserializeObject<MakeSmetaAll>(DataDle.ToString());

var Director = System.IO.Path.Combine(Environment.CurrentDirectory, "Temp");

if (!Directory.Exists(Director))

{

Directory.CreateDirectory(Director);

}

string NameOfFile = System.IO.Path.Combine(Director, "smeta1.dotx");

if (!File.Exists(NameOfFile))

File.WriteAllBytes(NameOfFile, RepairFlatWPF.Properties.Resources.Smeta);

using (var application = new NetOffice.WordApi.Application { Visible = true })

{

using (var document = application.Documents.Add(NameOfFile))

{

var AdressOfWork = document.Bookmarks["AdressOfWork"].Range;

AdressOfWork.Text = $"{InfForSmeta.AdressOfWork}";

var Contact = document.Bookmarks["Contact"].Range;

Contact.Text = $"{InfForSmeta.Contact}";

var FIO = document.Bookmarks["FIO"].Range;

FIO.Text = $"{InfForSmeta.FIO}";

var SummaMat = document.Bookmarks["SummaMat"].Range;

SummaMat.Text = $"{InfForSmeta.SummaMat}";

var SummaServ = document.Bookmarks["SummaServ"].Range;

SummaServ.Text = $"{InfForSmeta.SummaServ}";

if (InfForSmeta.materialsInf != null)

{

var InfAbMat = document.Bookmarks["InfAbMat"].Range.Tables[1];

InfAbMat.Rows[2].Cells.Delete();

foreach (var Mat in InfForSmeta.materialsInf)

{

var rows = InfAbMat.Rows.Add();

rows.Cells[1].Range.Text = Mat.numb.ToString();

rows.Cells[2].Range.Text = Mat.NameOfMaterials.ToString();

rows.Cells[3].Range.Text = Mat.count.ToString();

rows.Cells[4].Range.Text = Mat.cost.ToString();

rows.Cells[5].Range.Text = Mat.summa.ToString();

}

}

if (InfForSmeta.ServisInf != null)

{

var InfAbServ = document.Bookmarks["InfAbServ"].Range.Tables[1];

InfAbServ.Rows[2].Cells.Delete();

foreach (var Mat in InfForSmeta.ServisInf)

{

var rows = InfAbServ.Rows.Add();

rows.Cells[1].Range.Text = Mat.numb.ToString();

rows.Cells[2].Range.Text = Mat.NameOfServises.ToString();

rows.Cells[3].Range.Text = Mat.count.ToString();

rows.Cells[4].Range.Text = Mat.cost.ToString();

rows.Cells[5].Range.Text = Mat.summa.ToString();

}

}

}

application.Activate();

}

}

}

else

{

MakeSomeHelp.MSG("Выберите тип печати сметы", MsgBoxImage: MessageBoxImage.Hand);

}

}

private async void MakeDogovor\_Click(object sender, RoutedEventArgs e)

{

if (MakeSomeHelp.MSG("Вы дейсвительно хотите напечатать договор для текущего заказа?", MsgBoxImage: MessageBoxImage.Question, MsgBoxButton: MessageBoxButton.OKCancel) == MessageBoxResult.OK)

{

var DataDle = await Task.Run(() => MakeSomeHelp.MakeDownloadByLink($"api/Order/doc/dogovor?idOrder={idOrder}"));

var InfForDog = JsonConvert.DeserializeObject<MakeDogovor>(DataDle.ToString());

var Director = System.IO.Path.Combine(Environment.CurrentDirectory, "Temp");

if (!Directory.Exists(Director))

{

Directory.CreateDirectory(Director);

}

string NameOfFile = System.IO.Path.Combine(Director, "dogovor.dotx");

if (!File.Exists(NameOfFile))

File.WriteAllBytes(NameOfFile, RepairFlatWPF.Properties.Resources.Dogovor);

using (var application = new NetOffice.WordApi.Application { Visible = true })

{

using (var document = application.Documents.Add(NameOfFile))

{

var FIOSmall = document.Bookmarks["FIOSmall"].Range;

FIOSmall.Text = $"{InfForDog.FIOSmall}";

var Adress = document.Bookmarks["Adress"].Range;

Adress.Text = $"{InfForDog.Adress}";

var ContactInf = document.Bookmarks["ContactInf"].Range;

ContactInf.Text = $"{InfForDog.ContactInf}";

var FullFIO = document.Bookmarks["FullFIO"].Range;

FullFIO.Text = $"{InfForDog.FullFIO}";

var FullFIO1 = document.Bookmarks["FullFIO1"].Range;

FullFIO1.Text = $"{InfForDog.FullFIO}";

var Inn = document.Bookmarks["Inn"].Range;

Inn.Text = $"{InfForDog.Inn}";

var KPP = document.Bookmarks["KPP"].Range;

KPP.Text = $"{InfForDog.KPP}";

var NameOfOrganization = document.Bookmarks["NameOfOrganization"].Range;

NameOfOrganization.Text = $"{InfForDog.NameOfOrganization}";

var NameOfOrganization1 = document.Bookmarks["NameOfOrganization1"].Range;

NameOfOrganization1.Text = $"{InfForDog.NameOfOrganization}";

var Summa = document.Bookmarks["Summa"].Range;

Summa.Text = $"{InfForDog.Summa}";

var Summa1 = document.Bookmarks["Summa1"].Range;

Summa1.Text = $"{InfForDog.Summa}";

}

application.Activate();

}

}

}

private async void MakeSpavk\_Click(object sender, RoutedEventArgs e)

{

if (MakeSomeHelp.MSG("Вы дейсвительно хотите напечатать справочную информацию о текущем заказе?", MsgBoxImage: MessageBoxImage.Question, MsgBoxButton: MessageBoxButton.OKCancel) == MessageBoxResult.OK)

{

var DataDle = await Task.Run(() => MakeSomeHelp.MakeDownloadByLink($"api/Order/doc/sprav?idOrder={idOrder}"));

var InfForSpr = JsonConvert.DeserializeObject<MakeDataForSpravka>(DataDle.ToString());

var Director = System.IO.Path.Combine(Environment.CurrentDirectory, "Temp");

if (!Directory.Exists(Director))

{

Directory.CreateDirectory(Director);

}

string NameOfFile = System.IO.Path.Combine(Director, "SpavkaAboutOrder.dotx");

if (!File.Exists(NameOfFile))

File.WriteAllBytes(NameOfFile, RepairFlatWPF.Properties.Resources.SpavkaAboutOrder);

using (var application = new NetOffice.WordApi.Application { Visible = true })

{

using (var document = application.Documents.Add(NameOfFile))

{

if (!String.IsNullOrEmpty(InfForSpr.description?.Trim()))

{

var Description = document.Bookmarks["Description"].Range;

Description.Text = $"Описание: {InfForSpr.Description?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.NumberOfDelen?.Trim()))

{

var NumberOfDelen = document.Bookmarks["NumberOfDelen"].Range;

NumberOfDelen.Text = $"Квартира/офис: {InfForSpr.NumberOfDelen?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.Entrance?.Trim()))

{

var Entrance = document.Bookmarks["Entrance"].Range;

Entrance.Text = $"Подъезд: {InfForSpr.Entrance?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.House?.Trim()))

{

var House = document.Bookmarks["House"].Range;

House.Text = $"Дом: {InfForSpr.House?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.Street?.Trim()))

{

var Street = document.Bookmarks["Street"].Range;

Street.Text = $"Улица: {InfForSpr.Street?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.MicroAreaName?.Trim()))

{

var MicroAreaName = document.Bookmarks["MicroAreaName"].Range;

MicroAreaName.Text = $"Микрорайон: {InfForSpr.MicroAreaName?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.CityName?.Trim()))

{

var CityName = document.Bookmarks["CityName"].Range;

CityName.Text = $"Город: {InfForSpr.CityName?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.AreaName?.Trim()))

{

var AreaName = document.Bookmarks["AreaName"].Range;

AreaName.Text = $"Область: {InfForSpr.AreaName?.Trim()}{Environment.NewLine}";

}

if (!String.IsNullOrEmpty(InfForSpr.RegionName?.Trim()))

{

var RegionName = document.Bookmarks["RegionName"].Range;

RegionName.Text = $"Страна: {InfForSpr.RegionName?.Trim()}{Environment.NewLine}";

}

var DateMake = document.Bookmarks["DateMake"].Range;

DateMake.Text = $"{DateTime.Now.ToString("dd.MM.yyyy")}";

var DateMakeOrder = document.Bookmarks["DateMakeOrder"].Range;

DateMakeOrder.Text = $"{InfForSpr.DateMakeOrder.Value.ToString("dd.MM.yyyy")}";

var DateRozd = document.Bookmarks["DateRozd"].Range;

DateRozd.Text = $"{InfForSpr.DateRozd.Value.ToString("dd.MM.yyyy")}";

var DescContact = document.Bookmarks["DescContact"].Range;

DescContact.Text = $"{InfForSpr.DescContact}";

var LastName = document.Bookmarks["LastName"].Range;

LastName.Text = $"{InfForSpr.LastName}";

var Name = document.Bookmarks["Name"].Range;

Name.Text = $" {InfForSpr.Name}";

var Patronymic = document.Bookmarks["Patronymic"].Range;

Patronymic.Text = $"{InfForSpr.Patronymic}";

var StatusOfOrder = document.Bookmarks["StatusOfOrder"].Range;

StatusOfOrder.Text = $" <{SomeEnums.StatusOfOrder[InfForSpr.StatusOfOrder ?? default]}>";

var SummaOfOrder = document.Bookmarks["SummaOfOrder"].Range;

SummaOfOrder.Text = $"{InfForSpr.SummaOfOrder}";

var TypeOfcontact = document.Bookmarks["TypeOfcontact"].Range;

TypeOfcontact.Text = $"{InfForSpr.TypeOfcontact}";

var Value = document.Bookmarks["Value"].Range;

Value.Text = $"{InfForSpr.Value}";

if (InfForSpr.Premises != null)

{

var TablePremis = document.Bookmarks["PremisesInf"].Range.Tables[1];

TablePremis.Rows[2].Cells.Delete();

foreach (var conact in InfForSpr.Premises)

{

var rows = TablePremis.Rows.Add();

rows.Cells[1].Range.Text = conact.number.ToString();

rows.Cells[2].Range.Text = conact.NameOf.ToString();

rows.Cells[3].Range.Text = conact.Desc.ToString();

rows.Cells[4].Range.Text = conact.lenght.ToString();

rows.Cells[5].Range.Text = conact.Width.ToString();

rows.Cells[6].Range.Text = conact.Height.ToString();

rows.Cells[7].Range.Text = conact.PWalls.ToString();

rows.Cells[8].Range.Text = conact.PCelling.ToString();

rows.Cells[9].Range.Text = conact.SWalls.ToString();

rows.Cells[10].Range.Text = conact.SFloor.ToString();

}

}

}

application.Activate();

}

}

}

}

}

Модуль RepairFlatWPF. CreateNewOrder.xaml.cs

using Newtonsoft.Json;

using RepairFlat.Model;

using RepairFlatWPF.Model;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using static RepairFlat.Model.PersonDesctiption;

using static RepairFlatWPF.WorkWithOrder;

namespace RepairFlatWPF.UserControls.OrderWork

{

public partial class CreateNewOrder : UserControl

{

#region Переменные

public bool NewOrder = true;

Guid? IdUser;

Guid idAdress;

Guid idContact;

Guid TempidContact;

Guid idOrder;

List<Tuple<int, Guid>> ContactId = new List<Tuple<int, Guid>>();

BaseWindow Window;

#endregion

#region Конструктор

public CreateNewOrder(ref BaseWindow baseWindow, bool NewOrder = true, Guid idOrder = new Guid())

{

InitializeComponent();

DateOfOrder.SelectedDate = DateTime.Now;

Window = baseWindow;

this.NewOrder = NewOrder;

if (!NewOrder)

{

SelectClient.IsEnabled = false;

OperationBTN.Content = "Редактировать";

MakeOperationWithAdress.Content = "Редактировать";

LoadingDataAboutContact(idOrder);

this.idOrder = idOrder;

}

else

{

AllSumma.Text = "0";

}

foreach (var StatOfOrder in SomeEnums.StatusOfOrder)

{

StatusOfOrders.Items.Add(StatOfOrder);

}

}

#endregion

#region Обработка событий

private void SelectClient\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Выберите клиента");

baseWindow.MakeOpen(new ClientWork.SelectClientUserControl(SomeEnums.TypeOfConrols.Window, ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

if (SaveSomeData.SomeObject != null)

{

var DataAboutUser = SaveSomeData.SomeObject as DescriptionOfUser;

SaveSomeData.SomeObject = null;

IdUser = DataAboutUser.idUser;

string female = DataAboutUser.Female == 1 ? "МУЖ" : "Жен";

ClientFIO.Text = $"{DataAboutUser.Lastname?.Trim()} {DataAboutUser.Name?.Substring(0, 1).ToUpper()}.{DataAboutUser.Patronymic?.Substring(0, 1).ToUpper()}. {DataAboutUser.Birstday.Value.ToString("dd.MM.yyyy") } {female}";

List<Guid> ContactId = new List<Guid>();

ConctactType.Items.Clear();

makeloadingListOfContact(IdUser);

}

}

}

public async void makeloadingListOfContact(Guid? idUser, Guid idTempContact = new Guid())

{

try

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/contact/getusercontact?idUser={idUser}"));

ContactModel.ListOfUserContactInf listOfUserContactInf = JsonConvert.DeserializeObject<ContactModel.ListOfUserContactInf>(InformationFromServer.ToString());

if (listOfUserContactInf.listOfContact != null)

{

int num = 0;

foreach (var contact in listOfUserContactInf.listOfContact)

{

ComboBoxItem NewItem = new ComboBoxItem();

NewItem.Content = $"{contact.ValueTypeOfContact.Trim()} : {contact.Value.Trim()}";

NewItem.ToolTip = contact.Desctription;

ConctactType.Items.Add(NewItem);

ContactId.Add(Tuple.Create(num, contact.idContact));

num++;

}

if (idTempContact != null)

{

ConctactType.SelectedIndex = ContactId.Where(ee => ee.Item2 == idTempContact).Select(ee => ee.Item1).FirstOrDefault();

}

}

if (ConctactType.Items.Count == 0)

{

ComboBoxItem NewItem = new ComboBoxItem();

NewItem.Content = $"Необходимо добавить данные о контатах";

ConctactType.Items.Add(NewItem);

}

}

catch (Exception ex)

{

MakeSomeHelp.MSG(ex.ToString());

}

}

public object MakeDownloadByLink(string UrlOfDownload)

{

return BaseWorkWithServer.CatchErrorWithGet(UrlOfDownload, "GET", nameof(MakeLoading), nameof(MakeDownloadByLink));

}

private void ReturnBtn\_Click(object sender, RoutedEventArgs e)

{

Window.Close();

}

private void ConctactType\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

int ind = ConctactType.SelectedIndex;

if (ind != -1)

{

idContact = ContactId.Where(ww => ww.Item1 == ind).Select(ee => ee.Item2).FirstOrDefault();

}

}

private async void AddContactData\_Click(object sender, RoutedEventArgs e)

{

if (IdUser != null)

{

BaseWindow baseWindow = new BaseWindow("Добавление контактной информации");

baseWindow.MakeOpen(new AddContactUserConrol(IdUser, ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAboutNewContact = SaveSomeData.SomeObject as ContactModel.InformationAboutContact;

SaveSomeData.SomeObject = null;

string Json = JsonConvert.SerializeObject(dataAboutNewContact);

string urlSend = "api/contact/create";

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(urlSend, "POST", Json, nameof(BaseWorkWithServer), nameof(OperationBTN\_Click)));

var deserializedProduct = JsonConvert.DeserializeObject<BaseResult>(task.ToString());

ContactId.Clear();

ConctactType.Items.Clear();

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошибка при работе {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

MakeSomeHelp.MSG("Данные о контакте добавлены!", MsgBoxImage: MessageBoxImage.Information);

}

makeloadingListOfContact(IdUser);

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать клиента для добавления данных о контакте!", MsgBoxImage: MessageBoxImage.Error);

}

}

private async void RedactContactData\_Click(object sender, RoutedEventArgs e)

{

if (IdUser != null)

{

if (ConctactType.SelectedIndex != -1)

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/contact/getusercontact?idUser={IdUser}"));

ContactModel.ListOfUserContactInf listOfUserContactInf = JsonConvert.DeserializeObject<ContactModel.ListOfUserContactInf>(InformationFromServer.ToString());

ContactModel.InformationAboutContact aboutContact = new ContactModel.InformationAboutContact();

if (listOfUserContactInf.listOfContact != null)

{

foreach (var contact in listOfUserContactInf.listOfContact)

{

if (contact.idContact == idContact)

{

aboutContact.idContact = contact.idContact;

aboutContact.idTypeOfContact = contact.idTypeOfContact;

aboutContact.idUser = contact.idUser;

aboutContact.Value = contact.Value.Trim();

aboutContact.Desctription = contact.Desctription.Trim();

}

}

}

BaseWindow baseWindow = new BaseWindow("Редактирование контактной информации");

baseWindow.MakeOpen(new AddContactUserConrol(IdUser, ref baseWindow, aboutContact));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAboutNewContact = SaveSomeData.SomeObject as ContactModel.InformationAboutContact;

SaveSomeData.SomeObject = null;

string Json = JsonConvert.SerializeObject(dataAboutNewContact);

string urlSend = "api/contact/update";

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(urlSend, "POST", Json, nameof(BaseWorkWithServer), nameof(OperationBTN\_Click)));

var deserializedProduct = JsonConvert.DeserializeObject<BaseResult>(task.ToString());

ContactId.Clear();

ConctactType.Items.Clear();

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошикбка при работе {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

MakeSomeHelp.MSG("Данные о контакте обновлены!", MsgBoxImage: MessageBoxImage.Information);

}

makeloadingListOfContact(IdUser);

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать контакт для редактирования!", MsgBoxImage: MessageBoxImage.Error);

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать клиента для редактирования данных о номере контакте!", MsgBoxImage: MessageBoxImage.Error);

}

}

private async void OperationBTN\_Click(object sender, RoutedEventArgs e)

{

if (CheckField())

{

if (NewOrder)

{

idOrder = Guid.NewGuid();

BaseOrderInformation infOrder = new BaseOrderInformation

{

idAdress = idAdress,

Allsumma = 0,

DataStart = DateTime.Now,

Desc = Description.Text.Trim(),

idClient = IdUser,

idOrder = idOrder,

idWorkerMake = SaveSomeData.IdUser,

Status = StatusOfOrders.SelectedIndex,

MainContactID = idContact,

};

string Json = JsonConvert.SerializeObject(infOrder);

string urlSend = "api/order/create";

OperationBTN.Content = "Ожидайте...";

ReturnBtn.Content = "Ожидайте...";

ReturnBtn.IsEnabled = false;

OperationBTN.IsEnabled = false;

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(urlSend, "POST", Json, nameof(BaseWorkWithServer), nameof(OperationBTN\_Click)));

var deserializedProduct = JsonConvert.DeserializeObject<BaseResult>(task.ToString());

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошикбка при работе {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

MakeSomeHelp.MSG("Данные добавлены!", MsgBoxImage: MessageBoxImage.Information);

}

Window.Close();

}

else

{

BaseOrderInformation infOrder = new BaseOrderInformation

{

idAdress = idAdress,

Allsumma = 0,

Desc = Description.Text.Trim(),

idClient = IdUser,

idOrder = idOrder,

idWorkerMake = SaveSomeData.IdUser,

Status = StatusOfOrders.SelectedIndex,

MainContactID = idContact,

};

string Json = JsonConvert.SerializeObject(infOrder);

string urlSend = "api/order/update";

OperationBTN.Content = "Ожидайте...";

ReturnBtn.Content = "Ожидайте...";

var task = await Task.Run(() => BaseWorkWithServer.CatchErrorWithPost(urlSend, "POST", Json, nameof(BaseWorkWithServer), nameof(OperationBTN\_Click)));

var deserializedProduct = JsonConvert.DeserializeObject<BaseResult>(task.ToString());

if (!deserializedProduct.success)

{

MakeSomeHelp.MSG($"Произошла ошикбка при работе {deserializedProduct.description}", MsgBoxImage: MessageBoxImage.Error);

}

else

{

MakeSomeHelp.MSG("Данные отредактированы!", MsgBoxImage: MessageBoxImage.Information);

}

Window.Close();

}

}

}

private void MakeOperationWithAdress\_Click(object sender, RoutedEventArgs e)

{

if (NewOrder && idAdress == new Guid())

{//Тут добавление данных об адресе

BaseWindow baseWindow = new BaseWindow("Создание адресса");

baseWindow.MakeOpen(new AditinalControl.AddInformationAboutAdress(Guid.NewGuid(), ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

MakeOperationWithAdress.Content = "Редактировать";

SaveSomeData.MakeSomeOperation = false;

var dataAboutAdress = SaveSomeData.SomeObject as ModelAdress.DataAboutAdress;

SaveSomeData.SomeObject = null;

if (dataAboutAdress != null)

{

idAdress = dataAboutAdress.idAdress;

FullAdress.Text = $"{dataAboutAdress.CityName} {dataAboutAdress.Street} {dataAboutAdress.House} {dataAboutAdress.Entrance} {dataAboutAdress.NumberOfDelen}";

}

}

}

else

{//Тут редактирование данных об адресе

BaseWindow baseWindow = new BaseWindow("Обновление адресса");

baseWindow.MakeOpen(new AditinalControl.AddInformationAboutAdress(idAdress, ref baseWindow, Redact: true));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAboutAdress = SaveSomeData.SomeObject as ModelAdress.DataAboutAdress;

SaveSomeData.SomeObject = null;

if (dataAboutAdress != null)

{

FullAdress.Text = $"{dataAboutAdress.CityName} {dataAboutAdress.Street} {dataAboutAdress.House} {dataAboutAdress.Entrance} {dataAboutAdress.NumberOfDelen}";

}

}

}

}

#endregion

#region Прочие обработчики

private async void LoadingDataAboutContact(Guid idOrder)

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/order/dataforupdate?idOrder={idOrder}"));

OrderDesc.DataForUpdate DataAboutOrder = JsonConvert.DeserializeObject<OrderDesc.DataForUpdate>(InformationFromServer.ToString());

IdUser = DataAboutOrder.idUser;

idAdress = DataAboutOrder.idAdress ?? default(Guid);

TempidContact = DataAboutOrder.idContact ?? default(Guid);

idContact = DataAboutOrder.idContact ?? default(Guid);

DateOfOrder.SelectedDate = DataAboutOrder.DataStart;

StatusOfOrders.SelectedIndex = DataAboutOrder.Status ?? default(int);

AllSumma.Text = DataAboutOrder.AllSumma.ToString();

Description.Text = DataAboutOrder.Desc;

FullAdress.Text = DataAboutOrder.DataAboutAdress;

ClientFIO.Text = DataAboutOrder.FIOClient;

makeloadingListOfContact(IdUser, TempidContact);

}

private bool CheckField()

{

if (StatusOfOrders.SelectedIndex == -1)

{

MakeSomeHelp.MSG("Необходимо указать статус заказа!", MsgBoxImage: MessageBoxImage.Error);

return false;

}

if (IdUser == new Guid())

{

MakeSomeHelp.MSG("Необходимо указать данные о клиенте!", MsgBoxImage: MessageBoxImage.Error);

return false;

}

if (idAdress == new Guid())

{

MakeSomeHelp.MSG("Необходимо указать данные об адресе!", MsgBoxImage: MessageBoxImage.Error);

return false;

}

if (ConctactType.SelectedIndex == -1)

{

MakeSomeHelp.MSG("Необходимо выбрать основной способ связи!", MsgBoxImage: MessageBoxImage.Error);

return false;

}

return true;

}

#endregion

}

}

Модуль RepairFlatWPF. MainWorkWithOrderUserControl.xaml.cs

using System;

using System.Windows;

using System.Windows.Controls;

namespace RepairFlatWPF.UserControls.OrderWork

{

public partial class MainWorkWithOrderUserControl : UserControl

{

Guid idOrder;

public MainWorkWithOrderUserControl(Guid idOrder, object AllDataAboutOrder = null, string FioClient = "", string Adress = "", string AllSumma = "")

{

InitializeComponent();

this.idOrder = idOrder;

FIOClient.Text += FioClient;

this.Adress.Text += Adress;

SummaOfOrder.Text += AllSumma;

#region Загружаем данные обо всем заказе на соответсвующие элементы управления

//Данные о помещениях

ForPermisent.Children.Clear();

ForPermisent.Children.Add(new WorkWithMeasurment(idOrder));

//данные о заданиях

ForTasks.Children.Clear();

ForTasks.Children.Add(new WorkWithTasksUserControl(idOrder, ref SummaOfOrder, ref Ostatok));

//Данные об оплате

ForPayment.Children.Clear();

ForPayment.Children.Add(new InformationAboutOrderPay(idOrder, ref Ostatok, FioClient, Adress));

//При загрузке полностью данные о заказе

ForPrint.Children.Clear();

ForPrint.Children.Add(new AditionalTablesUserControl(idOrder));

#endregion

}

private void ReturnBtn\_Click(object sender, RoutedEventArgs e)

{

MakeSomeHelp.DataGridMakeWork(new SelectOrderToWork());

}

private void SelectTabsClick(object sender, RoutedEventArgs e)

{

int index = int.Parse(((Button)e.Source).Uid);

switch (index)

{

case 0:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowInformationAboutPresunt();

break;

case 1:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowInformationAboutTask();

break;

case 2:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowInformationAboutPayment();

break;

case 3:

GridCursor.SetValue(Grid.ColumnProperty, index);

ShowInformationAboutPrint();

break;

}

}

#region Показать соответсвующую "вкладку" (Говно код)

private void ShowInformationAboutPresunt()

{

ForTasks.Visibility = Visibility.Collapsed;

ForPermisent.Visibility = Visibility.Visible;

ForPayment.Visibility = Visibility.Collapsed;

ForPrint.Visibility = Visibility.Collapsed;

}

private void ShowInformationAboutPayment()

{

ForTasks.Visibility = Visibility.Collapsed;

ForPayment.Visibility = Visibility.Visible;

ForPermisent.Visibility = Visibility.Collapsed;

ForPrint.Visibility = Visibility.Collapsed;

}

private void ShowInformationAboutPrint()

{

ForTasks.Visibility = Visibility.Collapsed;

ForPrint.Visibility = Visibility.Visible;

ForPermisent.Visibility = Visibility.Collapsed;

ForPayment.Visibility = Visibility.Collapsed;

}

private void ShowInformationAboutTask()

{

ForTasks.Visibility = Visibility.Visible;

ForPrint.Visibility = Visibility.Collapsed;

ForPermisent.Visibility = Visibility.Collapsed;

ForPayment.Visibility = Visibility.Collapsed;

}

#endregion

}

}

Модуль RepairFlatWPF. PremisesRedactUC.xaml.cs

using Newtonsoft.Json;

using RepairFlatWPF.Controller;

using RepairFlatWPF.Model;

using System;

using System.Collections.Generic;

using System.Data.SQLite;

using System.Windows;

using System.Windows.Controls;

using static RepairFlat.Model.MakeSubs;

namespace RepairFlatWPF.UserControls.SettingsAndSubsInf.ControlForRedact

{

public partial class PremisesRedactUC : UserControl

{

BaseWindow window;

Guid idPremises;

bool Redact = false;

public PremisesRedactUC(ref BaseWindow baseWindow, object DataAboutPremises = null)

{

InitializeComponent();

window = baseWindow;

if (DataAboutPremises != null)

{

Redact = true;

var dat = DataAboutPremises as RepairFlat.Model.MakeSubs.ListOfPremises;

NameOfPremises.Text = dat.Name.Trim();

Description.Text = dat.Description.Trim();

idPremises = dat.idPremises;

AddBtn.Content = "Редактировать";

}

else

{

idPremises = Guid.NewGuid();

}

}

private void RetutnBTN\_Click(object sender, RoutedEventArgs e)

{

window.Close();

}

private void AddBtn\_Click(object sender, RoutedEventArgs e)

{

if (CheckData())

{

string query = "";

if (!Redact)

{

query = "Insert into PremisesType (idPremises,NameOfPremises,Descriprtion) values (@idPremises,@NameOfPremises,@Descriprtion)";

}

else

{

query = "Update PremisesType set NameOfPremises=@NameOfPremises, Descriprtion=@Descriprtion where idPremises=@idPremises;";

}

SQLiteParameter[] sQLiteParameter = new SQLiteParameter[3];

sQLiteParameter[0] = new SQLiteParameter("@idPremises", idPremises.ToString());

sQLiteParameter[1] = new SQLiteParameter("@NameOfPremises", NameOfPremises.Text.Trim());

sQLiteParameter[2] = new SQLiteParameter("@Descriprtion", Description.Text.Trim());

MakeWorkWirthDataBase.MakeSomeQueryWork(query, parameters: sQLiteParameter);

MakeUpdateServer();

}

}

private void MakeUpdateServer()

{

MakeUpdOrInsPremises makeUpdOrInsPremises = new MakeUpdOrInsPremises();

makeUpdOrInsPremises.idUser = SaveSomeData.IdUser ?? default;

makeUpdOrInsPremises.DateOfMake = DateTime.Now.ToString("dd.MM.yyyy HH:mm");

ListOfPremises listOfPremises = new ListOfPremises { idPremises = idPremises, Description = Description.Text.Trim(), Name = Description.Text.Trim() };

makeUpdOrInsPremises.ListOfPremises = new List<ListOfPremises>();

makeUpdOrInsPremises.ListOfPremises.Add(listOfPremises);

string Json = JsonConvert.SerializeObject(makeUpdOrInsPremises);

string urlSend = "api/substring/premises/update";

MakeSomeHelp.UpdloadDataToServer(urlSend, Json);

window.Close();

}

bool CheckData()

{

if (string.IsNullOrEmpty(NameOfPremises.Text.Trim()))

{

MakeSomeHelp.MSG("Необходимо указать название помещения", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (string.IsNullOrEmpty(Description.Text.Trim()))

{

MakeSomeHelp.MSG("Необходимо указать описание помещения", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

return true;

}

}

}

Модуль RepairFlatWPF. RedactSomeSubs.xaml.cs

using RepairFlatWPF.Model;

using RepairFlatWPF.UserControls.KadrWork;

using RepairFlatWPF.UserControls.SettingsAndSubsInf.ControlForRedact;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace RepairFlatWPF.UserControls.SettingsAndSubsInf

{

public partial class RedactSomeSubs : UserControl

{

SomeEnums.TypeOfSubs typeOfSubs;

DataTable DataAboutSomeSubInf;

List<Tuple<int, Guid>> ListofId;

bool whenPost = false;

public RedactSomeSubs(SomeEnums.TypeOfSubs typeOfSubs, bool whenPost = false)

{

InitializeComponent();

this.whenPost = whenPost;

this.typeOfSubs = typeOfSubs;

makeSelectFromDB();

}

private void MakePreparateData()

{

DataAboutSomeSubInf = new DataTable();

ListofId = new List<Tuple<int, Guid>>();

}

private void makeSelectFromDB()

{

MakePreparateData();

TypeOfWork.Text = "Работа над информации о:";

DataAboutSomeSubInf = MakeSomeHelp.DataTableFromDataBase(typeOfSubs);

switch (typeOfSubs)

{

case SomeEnums.TypeOfSubs.Contact: { TypeOfWork.Text += " типе контактов."; break; }

case SomeEnums.TypeOfSubs.Materials: { TypeOfWork.Text += " материалах."; break; }

case SomeEnums.TypeOfSubs.Post: { TypeOfWork.Text += " должностях."; break; }

case SomeEnums.TypeOfSubs.Premises: { TypeOfWork.Text += " типах помещений."; break; }

case SomeEnums.TypeOfSubs.Servises: { TypeOfWork.Text += " услугах"; break; }

}

ListofId = MakeSomeHelp.ListofId;

DataGrid.ItemsSource = DataAboutSomeSubInf.DefaultView;

}

private void RetunBtn\_Click(object sender, RoutedEventArgs e)

{

if (!whenPost)

MakeSomeHelp.DataGridMakeWork(new UserControls.SettingsAndSubsInf.SelectDataForRedactUC());

else

MakeSomeHelp.DataGridMakeWork(new MenuKadrWork());

}

private void AddElement\_Click(object sender, RoutedEventArgs e)

{

switch (typeOfSubs)

{

case SomeEnums.TypeOfSubs.Contact:

{//Работа с контакной информацией

BaseWindow baseWindow = new BaseWindow("Добавление типа контакной информации");

baseWindow.MakeOpen(new ContactTypeRedactUC(ref baseWindow));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Materials:

{//Работа с материалами

BaseWindow baseWindow = new BaseWindow("Добавление материала");

baseWindow.MakeOpen(new MaterialsRedactUC(ref baseWindow));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Post:

{//Работа с должностями

BaseWindow baseWindow = new BaseWindow("Добавление должностей");

baseWindow.MakeOpen(new PostRedactUC(ref baseWindow));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Premises:

{//Работа с типами помещений

BaseWindow baseWindow = new BaseWindow("Добавление типов помещений");

baseWindow.MakeOpen(new PremisesRedactUC(ref baseWindow));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Servises:

{//Работа с услугами

BaseWindow baseWindow = new BaseWindow("Добавление новых услуг");

baseWindow.MakeOpen(new ServisesRedactUC(ref baseWindow));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

}

}

private void EditElement\_Click(object sender, RoutedEventArgs e)

{

int index = DataGrid.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

for (int i = 0; i < DataAboutSomeSubInf.Rows.Count; i++)

{

if (Convert.ToInt32(DataAboutSomeSubInf.Rows[i][0].ToString()) == numberOfRows)

{

Guid idSubs = ListofId.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

SaveSomeData.MakeSomeOperation = false;

SaveSomeData.SomeObject = null;

switch (typeOfSubs)

{

case SomeEnums.TypeOfSubs.Contact:

{//Работа с контакной информацией

string value = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 1)?.ToString();

string desc = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 2)?.ToString();

string regex = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 3)?.ToString();

BaseWindow baseWindow = new BaseWindow("Редактирование контакной информации");

baseWindow.MakeOpen(new ContactTypeRedactUC(ref baseWindow, idSubs, value, desc, regex));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Materials:

{//Работа с материалами

RepairFlat.Model.MakeSubs.ListOfMaterials listOfMaterials = new RepairFlat.Model.MakeSubs.ListOfMaterials();

listOfMaterials.NameOfMaterial = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 1)?.ToString();

listOfMaterials.UnitOfMeasue = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 2)?.ToString();

listOfMaterials.TypeOfMaterial = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 3)?.ToString();

if (Decimal.TryParse(MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 4)?.ToString(), out decimal temp))

{

listOfMaterials.Cost = temp;

}

else

{

listOfMaterials.Cost = null;

}

listOfMaterials.idMaterials = idSubs;

listOfMaterials.Description = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 5)?.ToString();

BaseWindow baseWindow = new BaseWindow("Редактирование материала");

baseWindow.MakeOpen(new MaterialsRedactUC(ref baseWindow, listOfMaterials));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Post:

{//Работа с должностями

string dd = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 3).ToString();

bool? MakeWoe = default;

if (dd == "Да")

{

MakeWoe = true;

}

else if (dd == "Нет")

{

MakeWoe = false;

}

else

{

MakeWoe = null;

}

RepairFlat.Model.MakeSubs.ListOfPost listOfPost = new RepairFlat.Model.MakeSubs.ListOfPost();

listOfPost.NameOfPost = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 1)?.ToString();

listOfPost.MakeWork = MakeWoe;

listOfPost.idPost = idSubs;

if (decimal.TryParse(MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 2)?.ToString(), out decimal temp))

{

listOfPost.BaseWage = temp;

}

else

{

listOfPost.BaseWage = null;

}

BaseWindow baseWindow = new BaseWindow("Редактирование данных о должностях");

baseWindow.MakeOpen(new PostRedactUC(ref baseWindow, listOfPost));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Premises:

{//Работа с типами помещений

RepairFlat.Model.MakeSubs.ListOfPremises listOfPremises = new RepairFlat.Model.MakeSubs.ListOfPremises();

listOfPremises.idPremises = idSubs;

listOfPremises.Name = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 1)?.ToString();

listOfPremises.Description = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 2)?.ToString();

BaseWindow baseWindow = new BaseWindow("Обновление данных о типах помещений");

baseWindow.MakeOpen(new PremisesRedactUC(ref baseWindow, listOfPremises));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

case SomeEnums.TypeOfSubs.Servises:

{//Работа с услугами

RepairFlat.Model.MakeSubs.ListOfServises listOfServises = new RepairFlat.Model.MakeSubs.ListOfServises();

listOfServises.idServises = idSubs;

listOfServises.Nomination = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 1).ToString();

listOfServises.TypeOfServises = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 2).ToString();

listOfServises.Cost = Convert.ToDecimal(MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 3)?.ToString());

listOfServises.Description = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index, 4)?.ToString();

BaseWindow baseWindow = new BaseWindow("Обновление данных об услугах");

baseWindow.MakeOpen(new ServisesRedactUC(ref baseWindow, listOfServises));

baseWindow.ShowDialog();

makeSelectFromDB();

break;

}

}

}

}

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать данные для работы!", MsgBoxImage: MessageBoxImage.Hand);

}

}

}

}

Модуль RepairFlatWPF. SelectTypeForUpdateAndFile.xaml.cs

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

namespace RepairFlatWPF.UserControls.SettingsAndSubsInf

{

/// <summary>

/// Interaction logic for SelectTypeForUpdateAndFile.xaml

/// </summary>

public partial class SelectTypeForUpdateAndFile : UserControl

{

List<int> IndexOfSheet = new List<int>();

public SelectTypeForUpdateAndFile()

{

InitializeComponent();

}

private void Return\_Click(object sender, RoutedEventArgs e)

{

MakeSomeHelp.DataGridMakeWork(new UserControls.SettingsAndSubsInf.WorkWithSubInfromation());

}

private void ReadData\_Click(object sender, RoutedEventArgs e)

{

MakeSomeHelp.DataGridMakeWork(new UserControls.SettingsAndSubsInf.ShowDataAboutInsert(IndexOfSheet));

}

private void Checbox\_Checked(object sender, RoutedEventArgs e)

{

int index = int.Parse(((CheckBox)e.Source).Uid);

if (!IndexOfSheet.Contains(index))

{

IndexOfSheet.Add(index);

}

}

private void Checbox\_Unchecked(object sender, RoutedEventArgs e)

{

int index = int.Parse(((CheckBox)e.Source).Uid);

if (IndexOfSheet.Contains(index))

{

IndexOfSheet.Remove(index);

}

}

}

}

Модуль RepairFlatWPF. CreateNewWorker.xaml.cs

using Newtonsoft.Json;

using RepairFlat.Model;

using RepairFlatWPF.Model;

using System;

using System.Collections.Generic;

using System.Data;

using System.Linq;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using static RepairFlatWPF.Model.WorkerDescriptiom;

namespace RepairFlatWPF.UserControls.WorkerInformation.KadrWork

{

public partial class CreateNewWorker : UserControl

{

string TypeOfUser;

BaseWindow window;

bool NewWorker = true;

Guid idUser = Guid.NewGuid();

Guid idAdress;

DataTable DataAboutContact = new DataTable("ContactTable");

List<Tuple<int, Guid, Guid>> ContactDataGuidByNamber;

List<Guid?> idOfDelete = new List<Guid?>();

public CreateNewWorker(ref BaseWindow baseWindow, Guid idUser = new Guid())

{

InitializeComponent();

window = baseWindow;

MakeAllPreparationWithData();

foreach (string Type in SomeEnums.FemaleType)

{

Female.Items.Add(Type);

}

if (idUser != new Guid())

{

this.NewWorker = false;

AddBtn.Content = "Редактировать";

this.idUser = idUser;

makeloadingListOfContact();

WorkWithAdress.Content = "Редактировать";

makeotherSelectDataAsync();

}

}

private void MakeAllPreparationWithData()

{

DataAboutContact = new DataTable("ContactTable");

DataGrid.ItemsSource = DataAboutContact.DefaultView;

foreach (string NameOfColumn in SomeEnums.ContactTableDesc)

{

DataAboutContact.Columns.Add(NameOfColumn);

}

ContactDataGuidByNamber = new List<Tuple<int, Guid, Guid>>();

}

#region Работа с контактами

private void AddContact\_Click(object sender, RoutedEventArgs e)

{

BaseWindow baseWindow = new BaseWindow("Добавление контактной информации");

baseWindow.MakeOpen(new AddContactUserConrol(idUser, ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

var contactInformation = SaveSomeData.SomeObject as ContactModel.InformationAboutContact;

SaveSomeData.SomeObject = null;

SaveSomeData.MakeSomeOperation = false;

int Number = DataAboutContact.Rows.Count == 0 ? 1 : MakeSomeHelp.SelectMaxValueinColumn(ref DataAboutContact, SomeEnums.ContactTableDesc[0]);

DataRow newContact = DataAboutContact.NewRow();

newContact[0] = Number;

newContact[1] = contactInformation.NameOfValue;

newContact[2] = contactInformation.Value;

newContact[3] = contactInformation.Desctription;

DataAboutContact.Rows.Add(newContact);

ContactDataGuidByNamber.Add(new Tuple<int, Guid, Guid>(Number, contactInformation.idContact, contactInformation.idTypeOfContact ?? default(Guid)));

}

}

private void RedactContact\_Click(object sender, RoutedEventArgs e)

{

int index = DataGrid.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

ContactModel.InformationAboutContact aboutContact = new ContactModel.InformationAboutContact();

for (int i = 0; i < DataAboutContact.Rows.Count; i++)

{

if (Convert.ToInt32(DataAboutContact.Rows[i][0].ToString()) == numberOfRows)

{

Guid idContact = ContactDataGuidByNamber.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item2).First();

Guid idType = ContactDataGuidByNamber.Where(e2 => e2.Item1 == numberOfRows).Select(e1 => e1.Item3).First();

aboutContact = new ContactModel.InformationAboutContact

{

idContact = idContact,

idTypeOfContact = idType,

idUser = idUser,

Value = DataAboutContact.Rows[i][2].ToString(),

Desctription = DataAboutContact.Rows[i][3].ToString(),

NameOfValue = DataAboutContact.Rows[i][1].ToString(),

Number = numberOfRows

};

}

}

BaseWindow baseWindow = new BaseWindow("Редактирование контактной информации");

baseWindow.MakeOpen(new AddContactUserConrol(idUser, ref baseWindow, aboutContact));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

ContactModel.InformationAboutContact dataUpdated = SaveSomeData.SomeObject as ContactModel.InformationAboutContact;

SaveSomeData.SomeObject = null;

for (int i = 0; i < DataAboutContact.Rows.Count; i++)

{

if (Convert.ToInt32(DataAboutContact.Rows[i][0].ToString()) == dataUpdated.Number)

{

DataAboutContact.Rows[i][1] = dataUpdated.NameOfValue;

DataAboutContact.Rows[i][2] = dataUpdated.Value;

DataAboutContact.Rows[i][3] = dataUpdated.Desctription;

var data = ContactDataGuidByNamber.Single(e1 => e1.Item1 == dataUpdated.Number);

ContactDataGuidByNamber.Remove(data);

ContactDataGuidByNamber.Add(new Tuple<int, Guid, Guid>(dataUpdated.Number, dataUpdated.idContact, dataUpdated.idTypeOfContact ?? default));

}

}

}

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать контакт для редактирования", MsgBoxImage: MessageBoxImage.Error);

}

}

private void DeleteContact\_Click(object sender, RoutedEventArgs e)

{

int index = DataGrid.SelectedIndex;

if (index != -1)

{

var indexOfSelectedRows = MakeSomeHelp.SelectedRowsInDataGrid(ref DataGrid, index);

int numberOfRows = 0;

if (int.TryParse(indexOfSelectedRows.ToString(), out numberOfRows))

{

if (NewWorker)

{

for (int i = 0; i < DataAboutContact.Rows.Count; i++)

{

if (Convert.ToInt32(DataAboutContact.Rows[i][0].ToString()) == numberOfRows)

{

DataAboutContact.Rows[i].Delete();

var data = ContactDataGuidByNamber.Single(e1 => e1.Item1 == numberOfRows);

ContactDataGuidByNamber.Remove(data);

}

}

}

else

{

for (int i = 0; i < DataAboutContact.Rows.Count; i++)

{

if (Convert.ToInt32(DataAboutContact.Rows[i][0].ToString()) == numberOfRows)

{

DataAboutContact.Rows[i].Delete();

var data = ContactDataGuidByNamber.Single(e1 => e1.Item1 == numberOfRows);

idOfDelete.Add(data.Item2);

ContactDataGuidByNamber.Remove(data);

}

}

}

}

}

else

{

MakeSomeHelp.MSG("Необходимо выбрать строку для удаления", MsgBoxImage: MessageBoxImage.Error);

}

}

private async void makeotherSelectDataAsync()

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/worker/getData?idWorker={idUser}"));

var dataAbWorker= JsonConvert.DeserializeObject<MakeNewWorker>(InformationFromServer.ToString());

if (dataAbWorker != null)

{

Name.Text = dataAbWorker.Name;

Famil.Text = dataAbWorker.Lastname;

Patronymic.Text = dataAbWorker.Patronymic;

Pasport.Text = dataAbWorker.Pasport;

Female.SelectedItem = SomeEnums.FemaleType[dataAbWorker.Female?? default];

DateOfBirsd.SelectedDate = dataAbWorker.Birstday;

Adress.Text = dataAbWorker.DescOfAdress;

idAdress = dataAbWorker.idAdress;

TypeOfUser = dataAbWorker.TypeOfUser;

}

}

//Загрузка данных о контактах

public async void makeloadingListOfContact()

{

var InformationFromServer = await Task.Run(() => MakeDownloadByLink($"api/contact/getusercontact?idUser={idUser}"));

ContactModel.ListOfUserContactInf listOfUserContactInf = JsonConvert.DeserializeObject<ContactModel.ListOfUserContactInf>(InformationFromServer.ToString());

if (listOfUserContactInf != null)

{

int number = 1;

foreach (var contact in listOfUserContactInf.listOfContact)

{

DataRow NewContactData = DataAboutContact.NewRow();

NewContactData[0] = number;

NewContactData[1] = contact.ValueTypeOfContact?.Trim();

NewContactData[2] = contact.Value?.Trim();

NewContactData[3] = contact.Desctription?.Trim();

DataAboutContact.Rows.Add(NewContactData);

ContactDataGuidByNamber.Add(new Tuple<int, Guid, Guid>(number, contact.idContact, contact.idTypeOfContact ?? default(Guid)));

number++;

}

}

DataGrid.ItemsSource = DataAboutContact.DefaultView;

}

#endregion

//Добавление работника

private void AddBtn\_Click(object sender, RoutedEventArgs e)

{

if (CheckData())

{

if (NewWorker)

{//Если добавление

MakeNewWorker newWorker = new MakeNewWorker();

newWorker.InformatioAboutContact = new PersonDesctiption.InformationAboutContact();

newWorker.InformatioAboutContact.ListOfContact = new List<ContactModel.InformationAboutContact>();

for (int i = 0; i < DataAboutContact.Rows.Count; i++)

{

var idOfCont = ContactDataGuidByNamber.Where(ee => ee.Item1 == Convert.ToInt32(DataAboutContact.Rows[i][0].ToString()));

ContactModel.InformationAboutContact contact = new ContactModel.InformationAboutContact();

contact.Desctription = DataAboutContact.Rows[i][3].ToString()?.Trim();

contact.Value = DataAboutContact.Rows[i][2].ToString()?.Trim();

contact.idUser = idUser;

contact.idTypeOfContact = idOfCont.First().Item3;

contact.idContact = idOfCont.First().Item2;

contact.DateAdd = DateTime.Now;

newWorker.InformatioAboutContact.ListOfContact.Add(contact);

}

newWorker.Birstday = DateOfBirsd.SelectedDate.Value;

newWorker.Female = Female.SelectedIndex;

newWorker.idAdress = idAdress;

newWorker.idUser = idUser;

newWorker.Lastname = Famil.Text?.Trim();

newWorker.Name = Name.Text?.Trim();

newWorker.Patronymic = Patronymic.Text?.Trim();

newWorker.Pasport = Pasport.Text?.Trim();

newWorker.TypeOfUser = SomeEnums.TypeOfUser.KD.ToString();

string Json = JsonConvert.SerializeObject(newWorker);

string urlSend = "api/worker/createorupdate/worker";

MakeSomeHelp.UpdloadDataToServer(urlSend, Json);

window.Close();

}

else

{//Если редактирование

MakeNewWorker newWorker = new MakeNewWorker();

newWorker.InformatioAboutContact = new PersonDesctiption.InformationAboutContact();

newWorker.InformatioAboutContact.ListOfContact = new List<ContactModel.InformationAboutContact>();

for (int i = 0; i < DataAboutContact.Rows.Count; i++)

{

var idOfCont = ContactDataGuidByNamber.Where(ee => ee.Item1 == Convert.ToInt32(DataAboutContact.Rows[i][0].ToString()));

ContactModel.InformationAboutContact contact = new ContactModel.InformationAboutContact();

contact.Desctription = DataAboutContact.Rows[i][3].ToString()?.Trim();

contact.Value = DataAboutContact.Rows[i][2].ToString()?.Trim();

contact.idUser = idUser;

contact.idTypeOfContact = idOfCont.First().Item3;

contact.idContact = idOfCont.First().Item2;

contact.DateAdd = DateTime.Now;

newWorker.InformatioAboutContact.ListOfContact.Add(contact);

}

newWorker.Birstday = DateOfBirsd.SelectedDate.Value;

newWorker.Female = Female.SelectedIndex;

newWorker.idAdress = idAdress;

newWorker.idUser = idUser;

newWorker.Lastname = Famil.Text?.Trim();

newWorker.Name = Name.Text?.Trim();

newWorker.Patronymic = Patronymic.Text?.Trim();

newWorker.Pasport = Pasport.Text?.Trim();

newWorker.TypeOfUser = TypeOfUser;

if (idOfDelete.Count != 0)

{

newWorker.InformatioAboutContact.ListForDelete = new List<Guid?>();

newWorker.InformatioAboutContact.ListForDelete = idOfDelete;

}

string Json = JsonConvert.SerializeObject(newWorker);

string urlSend = "api/worker/createorupdate/worker";

MakeSomeHelp.UpdloadDataToServer(urlSend, Json);

window.Close();

}

}

}

private void RetutnBTN\_Click(object sender, RoutedEventArgs e)

{

window.Close();

}

#region Работа с адресом

private void WorkWithAdress\_Click(object sender, RoutedEventArgs e)

{

if (NewWorker)

{

this.idAdress = Guid.NewGuid();

BaseWindow baseWindow = new BaseWindow("Создание адресса");

baseWindow.MakeOpen(new AditinalControl.AddInformationAboutAdress(idAdress, ref baseWindow));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAboutAdress = SaveSomeData.SomeObject as ModelAdress.DataAboutAdress;

SaveSomeData.SomeObject = null;

if (dataAboutAdress != null)

{

idAdress = dataAboutAdress.idAdress;

Adress.Text = $"{dataAboutAdress.CityName} {dataAboutAdress.Street} {dataAboutAdress.House} {dataAboutAdress.Entrance} {dataAboutAdress.NumberOfDelen}";

}

}

}

else

{

BaseWindow baseWindow = new BaseWindow("Обновление адресса");

baseWindow.MakeOpen(new AditinalControl.AddInformationAboutAdress(idAdress, ref baseWindow, Redact: true));

baseWindow.ShowDialog();

if (SaveSomeData.MakeSomeOperation)

{

SaveSomeData.MakeSomeOperation = false;

var dataAboutAdress = SaveSomeData.SomeObject as ModelAdress.DataAboutAdress;

SaveSomeData.SomeObject = null;

if (dataAboutAdress != null)

{

Adress.Text = $"{dataAboutAdress.CityName} {dataAboutAdress.Street} {dataAboutAdress.House} {dataAboutAdress.Entrance} {dataAboutAdress.NumberOfDelen}";

}

}

}

}

#endregion

public object MakeDownloadByLink(string UrlOfDownload)

{

return BaseWorkWithServer.CatchErrorWithGet(UrlOfDownload, "GET", nameof(MakeLoading), nameof(MakeDownloadByLink));

}

bool CheckData()

{

if (string.IsNullOrEmpty(Name.Text.Trim()))

{

MakeSomeHelp.MSG("Необходимо указать значение имени", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (string.IsNullOrEmpty(Famil.Text.Trim()))

{

MakeSomeHelp.MSG("Необходимо указать значение фамилии", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (string.IsNullOrEmpty(Patronymic.Text.Trim()))

{

MakeSomeHelp.MSG("Необходимо указать значение отчества", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (string.IsNullOrEmpty(Pasport.Text.Trim()))

{

MakeSomeHelp.MSG("Необходимо указать значение паспорта", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (Female.SelectedIndex==-1)

{

MakeSomeHelp.MSG("Необходимо указать значение пола работника", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (DateOfBirsd.SelectedDate == null)

{

MakeSomeHelp.MSG("Необходимо указать дату рождения", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (idAdress == new Guid())

{

MakeSomeHelp.MSG("Необходимо место проживания", MsgBoxImage: MessageBoxImage.Hand);

return false;

}

if (DataAboutContact.Rows.Count==0)

{

if (MakeSomeHelp.MSG("Не указана контакная информация для работника. Вы действительно не хотите ее указывать?", MsgBoxImage: MessageBoxImage.Question,MsgBoxButton:MessageBoxButton.OKCancel) == MessageBoxResult.OK)

{

return true;

}

else

{

return false;

}

}

return true;

}

}

}

Модуль RepairFlatWPF. BaseWindow.xaml.cs

using System.Windows;

using System.Windows.Controls;

namespace RepairFlatWPF.UserControls

{

public partial class BaseWindow : Window

{

public BaseWindow(string title)

{

InitializeComponent();

this.DataContext = new MainWindowViewModel(this, title);

}

public void MakeOpen(UserControl userControl)

{

GridForContent.Children.Clear();

GridForContent.Children.Add(userControl);

}

}

}

Модуль RepairFlatWPF. MainWindow.xaml.cs

using RepairFlatWPF.Model;

using RepairFlatWPF.UserControls;

using RepairFlatWPF.UserControls.KadrWork;

using RepairFlatWPF.UserControls.MoneyInformation;

using RepairFlatWPF.UserControls.SettingsAndSubsInf;

using System;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media.Animation;

namespace RepairFlatWPF

{

public partial class MainWindow : Window

{

bool open = true;

public MainWindow()

{

InitializeComponent();

this.DataContext = new MainWindowViewModel(this, "");

MakeSomeHelp.MakeShowLogining();

}

public void Makecheck()

{

WorkWithOrder.Visibility = Visibility.Collapsed;

ClientWork.Visibility = Visibility.Collapsed;

Spravoch.Visibility = Visibility.Collapsed;

KadrWork.Visibility = Visibility.Collapsed;

Finans.Visibility = Visibility.Collapsed;

Settings.Visibility = Visibility.Collapsed;

NameOfPolz.Text = SaveSomeData.LastNameAndIni;

if (SaveSomeData.TypeOfUser == SomeEnums.TypeOfUser.AD.ToString())

{//Администратор

Settings.Visibility = Visibility.Visible;

Spravoch.Visibility = Visibility.Visible;

MakeSomeHelp.DataGridMakeWork(new SettingsUserControls());

}

else if (SaveSomeData.TypeOfUser == SomeEnums.TypeOfUser.BW.ToString())

{//Работник бухгалтерии

Finans.Visibility = Visibility.Visible;

Spravoch.Visibility = Visibility.Visible;

MakeSomeHelp.DataGridMakeWork(new FinansInformation());

}

else if (SaveSomeData.TypeOfUser == SomeEnums.TypeOfUser.KW.ToString())

{//Работник отдела кадров

KadrWork.Visibility = Visibility.Visible;

MakeSomeHelp.DataGridMakeWork(new MenuKadrWork());

}

else if (SaveSomeData.TypeOfUser == SomeEnums.TypeOfUser.MG.ToString())

{//Менеджер

WorkWithOrder.Visibility = Visibility.Visible;

ClientWork.Visibility = Visibility.Visible;

Spravoch.Visibility = Visibility.Visible;

MakeSomeHelp.DataGridMakeWork(new SelectOrderToWork());

}

else if (SaveSomeData.TypeOfUser == SomeEnums.TypeOfUser.BB.ToString())

{//Босс

WorkWithOrder.Visibility = Visibility.Visible;

ClientWork.Visibility = Visibility.Visible;

Spravoch.Visibility = Visibility.Visible;

KadrWork.Visibility = Visibility.Visible;

Finans.Visibility = Visibility.Visible;

Settings.Visibility = Visibility.Visible;

MakeSomeHelp.DataGridMakeWork(new SelectOrderToWork());

}

}

private void ButtonOpenMenu\_Click(object sender, RoutedEventArgs e)

{

ButtonOpenMenu.Visibility = Visibility.Visible;

open = true;

ButtonCloseMenu.Visibility = Visibility.Collapsed;

Bluring.Visibility = Visibility.Visible;

}

private void ButtonMenuClose\_Click(object sender, RoutedEventArgs e)

{

ButtonCloseMenu.Visibility = Visibility.Visible;

open = false;

ButtonOpenMenu.Visibility = Visibility.Collapsed;

Bluring.Visibility = Visibility.Collapsed;

}

private void ListViewMenu\_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

if (open)

{

int index = ListViewMenu.SelectedIndex;

switch (index)

{

case 0:

//Работа с заказами

CloseMenu();

MakeSomeHelp.DataGridMakeWork(new UserControls.SelectOrderToWork());

break;

case 1:

//Работа с клиентами

CloseMenu();

BaseWindow baseWindow = new BaseWindow("sa");

MakeSomeHelp.DataGridMakeWork(new UserControls.ClientWork.SelectClientUserControl(SomeEnums.TypeOfConrols.UserControl, ref baseWindow));

break;

case 2:

//Справочные данные

CloseMenu();

MakeSomeHelp.DataGridMakeWork(new UserControls.SettingsAndSubsInf.WorkWithSubInfromation());

break;

case 3:

//Работа с кадрами

CloseMenu();

MakeSomeHelp.DataGridMakeWork(new UserControls.KadrWork.MenuKadrWork());

break;

case 4:

//Работа с финансами

CloseMenu();

MakeSomeHelp.DataGridMakeWork(new UserControls.MoneyInformation.FinansInformation());

break;

}

}

}

void CloseMenu()

{

var animation = (Storyboard)FindResource("CloseMenu");

animation.Begin();

ButtonCloseMenu.Visibility = Visibility.Visible;

open = false;

ButtonOpenMenu.Visibility = Visibility.Collapsed;

Bluring.Visibility = Visibility.Collapsed;

}

private void BottomListView\_PreviewMouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

//Проверка на то, что окно открыто или нет, сделано, для того, что бы не работа при закрытом

if (open)

{

int index = BottomListView.SelectedIndex;

switch (index)

{

case 0:

//Переход на отображение настроек

CloseMenu();

MakeSomeHelp.DataGridMakeWork(new UserControls.SettingsAndSubsInf.SettingsUserControls());

break;

case 1:

//Смена профиля

//bright.Visibility = Visibility.Collapsed;

CloseMenu();

if (MessageBox.Show("Вы действительно хотите сменить пользователя?", "АИС Фирмы по ремонту квартир", MessageBoxButton.OKCancel, MessageBoxImage.Question) == MessageBoxResult.OK)

{

GridMenu.Width = 0;

MakeSomeHelp.MakeShowLogining();

SaveSomeData.LastNameAndIni = "";

SaveSomeData.IdUser = new Guid();

SaveSomeData.TypeOfUser = "";

open = false;

}

break;

case 2:

//Выход из приложения

if (MessageBox.Show("Вы действительно хотите выйти из программы?", "АИС Ремонт квартир", MessageBoxButton.OKCancel, MessageBoxImage.Question) == MessageBoxResult.OK)

{

Application.Current.Shutdown();

}

break; }

}

}

public void setChildren(UserControl ss1)

{

MainGrid.Children.Clear();

MainGrid.Children.Add(ss1);

}

private void Bluring\_MouseLeftButtonUp(object sender, MouseButtonEventArgs e)

{

CloseMenu();

}

}

}

Модуль RepairFlatWPF. BaseWorkWithServer.cs

using RepairFlatWPF.Properties;

using System;

using System.IO;

using System.Net;

namespace RepairFlatWPF

{

public static class BaseWorkWithServer

{

public static object CatchErrorWithPost(string PosfixUrl, string typeOfMessage, string WhatSend, string NameOfClass, string nameOfMethod)

{

try

{

var request = (HttpWebRequest)WebRequest.Create(UrlSendMake(PosfixUrl));

request.ContentType = "application/json";

request.Method = typeOfMessage;

using (var streamWriter = new StreamWriter(request.GetRequestStream()))

{

streamWriter.WriteAsync(WhatSend);

}

var response = (HttpWebResponse)request.GetResponse();

object result = new object();

using (StreamReader streamReader = new StreamReader(response.GetResponseStream()))

{

result = streamReader.ReadToEnd();

}

return result;

}

catch (Exception ex)

{

MakeSomeHelp.MSG($"Произошла ошибка при работе с сервером {ex.Message}");

return null;

}

}

public static object CatchErrorWithGet(string PosfixUrl, string typeOfMessage, string NameOfClass, string nameOfMethod)

{

try

{

var request = (HttpWebRequest)WebRequest.Create(UrlSendMake(PosfixUrl));

request.ContentType = "application/json";

request.Method = typeOfMessage;

var response = (HttpWebResponse)request.GetResponse();

object result = new object();

using (StreamReader streamReader = new StreamReader(response.GetResponseStream()))

{

result = streamReader.ReadToEnd();

}

return result;

}

catch (Exception ex)

{

MakeSomeHelp.MSG($"Произошла ошибка при работе с сервером {ex.Message}");

return null;

}

}

private static string UrlSendMake(string urlSecondPart)

{

string baseAdress = Settings.Default.BaseAdress;

if (string.IsNullOrEmpty(baseAdress)) throw new Exception("Необходимо указать адрес для сервера для работы!");

Uri baseUri = new Uri(baseAdress);

Uri resultUrl = new Uri(baseUri, urlSecondPart);

return resultUrl.ToString();

}

}

}

# Приложение Б

Титульный лист презентации

