ДЕПАРТАМЕНТ ОБРАЗОВАНИЯ ТОМСКОЙ ОБЛАСТИ

ОБЛАСТНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ПРОФЕССИОНАЛЬНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

«ТОМСКИЙ ТЕХНИКУМ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ»

Специальность 09.02.07 Информационные системы и программирование

Разработка автоматизированной информационной системы учета риэлтерских операций «Realtor tool»

Пояснительная записка

к курсовому проекту

КП.25.09.02.07.631.11 ПЗ

|  |  |  |
| --- | --- | --- |
| Студент  «\_\_»\_\_\_\_\_\_\_\_ 2025 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Литвинчук Э. А. |
| Преподаватель  «\_\_»\_\_\_\_\_\_\_\_ 2025 г. | \_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Маюнова А. Ю. |

Томск 2025

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc215184125)

[1 ОБЩАЯ ЧАСТЬ 4](#_Toc215184126)

[1.1 Анализ предметной области 4](#_Toc215184127)

[1.1.1 Характеристика предметной области 4](#_Toc215184128)

[1.1.2 Анализ конкурентов 5](#_Toc215184129)

[1.1.3 Анализ целевой аудитории 6](#_Toc215184130)

[1.2 Выбор сред и средств разработки 6](#_Toc215184131)

[1.2.1 Выбор СУБД 6](#_Toc215184132)

[1.2.2 Выбор языка программирования 7](#_Toc215184133)

[1.2.3 Выбор фреймворка 8](#_Toc215184134)

[1.2.4 Выбор среды разработки 9](#_Toc215184135)

[2 СПЕЦИАЛЬНАЯ ЧАСТЬ 10](#_Toc215184136)

[2.1 Список терминов и их определения 10](#_Toc215184137)

[2.2 Требования к информационной системе 11](#_Toc215184138)

[2.3 Диаграммы вариантов использования 12](#_Toc215184139)

[2.4 Диаграммы состояний 14](#_Toc215184140)

[2.5 Схема данных 17](#_Toc215184141)

[2.6 Словарь данных 18](#_Toc215184142)

[2.7 Пользовательский сценарий 23](#_Toc215184143)

[2.8 Прототипы интерфейсов 24](#_Toc215184144)

[ЗАКЛЮЧЕНИЕ 29](#_Toc215184145)

[ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 30](#_Toc215184146)

[ПРИЛОЖЕНИЕ А 32](#_Toc215184147)

[ПРИЛОЖЕНИЕ Б 34](#_Toc215184148)

[ПРИЛОЖЕНИЕ В 38](#_Toc215184149)

[ПРИЛОЖЕНИЕ Г 158](#_Toc215184150)

# ВВЕДЕНИЕ

В современном мире информационные системы становятся неотъемлемой частью различных сфер деятельности, включая сферу недвижимости. Риэлторские агентства, занимающиеся продажей, арендой и управлением недвижимостью, являются популярными местами, где люди ищут жилье, инвестируют и заключают сделки. Однако, с ростом популярности услуг в сфере недвижимости, владельцы агентств сталкиваются с проблемами, связанными с управлением объектами, обслуживанием клиентов и эффективным учетом финансовых операций.

В рамках данного курсового проекта было разработано кроссплатформенное приложение для риэлторского агентства «Realtor tool». Настольное приложение представляет собой решение, которое позволяет эффективно управлять базой данных недвижимости, обрабатывать запросы клиентов и автоматизировать финансовые операции без необходимости бумажной волокиты. Это удобно и экономит время как для клиентов, так и для сотрудников агентства.

Целью данного проекта является улучшение процесса обслуживания клиентов в риэлторском агентстве, повышение эффективности работы персонала и увеличение общей прибыльности агентства путём ускорения обработки запросов и оптимизации роли агентов. Настольное приложение для агентства включает в себя модули для управления базой недвижимости, сотрудниками, акциями и учетом финансовых операций.

Ожидается, что разработанное настольное приложение для риэлторского агентства «Realtor tool» будет способствовать повышению уровня обслуживания клиентов и оптимизации работы персонала.

# 1 ОБЩАЯ ЧАСТЬ

## Анализ предметной области

### Характеристика предметной области

Риэлторская деятельность представляет собой профессиональную посредническую услугу на рынке недвижимости, включающую операции купли-продажи, аренды и обмена объектов недвижимости. Предметной областью данного проекта является автоматизация бизнес-процессов риэлторского агентства, специализирующегося на работе с жилой недвижимостью: квартирами, частными домами и земельными участками.

Ключевые процессы предметной области включают:

1. ведение базы клиентов и объектов недвижимости;
2. управление заявками на покупку, продажу, аренду и сдачу в аренду;
3. подбор соответствий между потребностями покупателей/арендаторов и предложениями продавцов/арендодателей;
4. оформление и сопровождение сделок;
5. расчет комиссионных вознаграждений;
6. управление персоналом и распределение обязанностей.

Особенностью данной предметной области является необходимость обработки большого объема структурированной информации, требующей точного сопоставления параметров и оперативного обновления статусов. Система должна обеспечивать надежное хранение данных, эффективный поиск и возможность анализа информации для принятия бизнес-решений.

### Анализ конкурентов

На рынке программного обеспечения для риэлторских агентств представлены как универсальные CRM-системы, так и специализированные решения. Основными конкурентами можно считать:

1. универсальные CRM-системы (Битрикс24, AMO CRM):
2. преимущества: широкий функционал, интеграция с другими бизнес-процессами;
3. недостатки: отсутствие специализации под риэлторскую деятельность, сложность адаптации;
4. специализированные риэлторские системы (РИЭЛТОР, RECRM):
5. преимущества: ориентированность на специфику работы с недвижимостью;
6. недостатки: высокая стоимость, избыточность функций для небольших агентств;
7. самодельные решения на базе Excel/Access:
8. преимущества: низкая стоимость внедрения;
9. недостатки: ограниченная функциональность, ошибки при ручном вводе данных.

Конкурентные преимущества разрабатываемой системы «Realtor tool»:

1. специализация на жилой недвижимости;
2. автоматизированный подбор совпадений между заявками;
3. интуитивно понятный интерфейс;
4. гибкая система ролей и прав доступа;
5. оптимизация для работы небольших и средних агентств.

### Анализ целевой аудитории

Целевой аудиторией проекта являются следующие группы людей:

1. риэлторы;
2. административные агентства;
3. менеджеры.

Ключевые потребности целевой аудитории:

1. сокращение времени на обработку заявок;
2. уменьшение количества ошибок при подборе объектов;
3. прозрачность бизнес-процессов;
4. возможность удаленной работы;
5. интеграция с другими системами (бухгалтерскими, отчетными);
6. соответствие законодательным требованиям.

Система "Realtor tool" ориентирована на удовлетворение этих потребностей через автоматизацию рутинных операций и предоставление инструментов для эффективного управления риэлторским бизнесом.

## Выбор сред и средств разработки

### Выбор СУБД

При выборе СУБД конкурировали две популярные системы управления реляционными базами данных: «PostgreSQL» и «MySQL».

«PostgreSQL» — это свободная объектно-реляционная система управления базами данных, имеющая открытый исходный код, строгую поддержку ACID, оптимизированная для обеспечения высокой производительности, поддерживающая сложные запросы и операции.

«MySQL» — это объектно-реляционная система управления базами данных, принадлежащая компании Oracle. Данная СУБД имеет меньше встроенных функций, что почти не играет роли в данном проекте, однако имеет лицензирование.

Разрабатываемый проект не имеет особых требований к СУБД, поэтому выбор между вышеперечисленными системами мог бы ограничиться рамками привычности команды разработчиков, однако предпочтение было отдано «PostgreSQL», ввиду невозможности использования лицензионной версии «MySQL» на территории России.

### Выбор языка программирования

Разрабатываемая система является клиент-серверное приложением, что подразумевает использования одного или нескольких языков программирования.

При выборе языка программирования для серверной части системы конкурировали два популярных языка программирования: «C#» и «Java».

«C#» — это язык высокого уровня, разработанный компанией Microsoft, который предлагает высокую производительность, имеющий отличный потенциал для использования больших проектов ввиду хорошей масштабируемости, а также имеющий обширную документацию и большое сообщество.

«Java» — это кроссплатформенный язык программирования, отличающийся высокой стабильностью работы и широким распространением в корпоративной среде, однако требующий большего объема кода для реализации аналогичного функционала и обладающий менее выразительным синтаксисом по сравнению с C#.

Сравнительный анализ показал следующие преимущества C# для данного проекта:

1. более высокая производительность благодаря оптимизированной работе с памятью и современным компиляторам;
2. богатый набор языковых возможностей (LINQ, async/await, свойства, делегаты), сокращающих объем boilerplate-кода;
3. мощная экосистема .NET с интегрированными решениями для веб-сервисов, работы с базами данных и безопасности;
4. лучшая эргономика разработки за счет строгой типизации и современных инструментов Visual Studio;
5. активное развитие языка и платформы с регулярным выходом новых версий.

### Выбор фреймворка

Разрабатываемая система является клиент-серверным приложением, что подразумевает использование одного или нескольких фреймворков для реализации пользовательского интерфейса.

При выборе технологии для клиентской части системы конкурировали два популярных фреймворка: «Avalonia UI» и «WPF».

«Avalonia UI» — это кроссплатформенный фреймворк для построения пользовательских интерфейсов, который предлагает единую кодную базу для запуска на Windows, Linux и macOS, имеющий современную архитектуру и активное сообщество разработчиков, а также поддерживающий последние стандарты разработки.

«WPF» — это технология для создания desktop-приложений от Microsoft, отличающаяся зрелостью и богатой экосистемой, однако ограниченная поддержкой только операционной системы Windows и имеющая менее активное развитие в последние годы.

Сравнительный анализ показал следующие преимущества Avalonia UI для данного проекта:

1. полная кроссплатформенность с нативной производительностью на всех основных ОС;
2. современная архитектура с поддержкой последних версий .NET и C#;
3. активное развитие и регулярные обновления с добавлением новых возможностей;
4. единая кодовая база для всех платформ, снижающая затраты на разработку и поддержку;
5. открытый исходный код с прозрачным процессом разработки.

### Выбор среды разработки

После выбора языка программирования ясно, что необходима среда разработки с функционалом, поддерживающим «C#» и платформу «.Net». Выбор стоял между двумя мощными средами разработки: «Rider» и «Visual Studio».

«Rider» — это кроссплатформенная IDE для .NET-разработчиков, принадлежащая компании «JetBrains», основанная на платформе IntelliJ и ReSharper, позволяющая не только разрабатывать код, но и работать с фреймворками и СУБД.

«Visual Studio» — это интегрированная среда разработки (IDE) от компании Microsoft, предназначенная для создания приложений, сервисов и инструментов для различных платформ. Данная среда разработки поддерживает множество языков программирования, что делает его универсальным инструментом для разработчиков.

В ходе анализа сред разработки был выбран «Rider», ввиду высшей производительности и удобств, связанных с работой с фреймворками и базами данных.

# 2 СПЕЦИАЛЬНАЯ ЧАСТЬ

## Список терминов и их определения

Информационная система — это программно-аппаратный комплекс, предназначенный для сбора, хранения, обработки и предоставления информации, поддерживающий бизнес-процессы риэлторского агентства.

Риэлтор – это сотрудник агентства недвижимости, имеющий право на совершение операций с недвижимостью (создание и редактирование заявок, сделок, подбор совпадений)

Администратор – это роль пользователя в системе, обладающая правами управления сотрудниками (просмотр, увольнение, восстановление, сброс пароля) и полным доступом к данным

Неавторизованный пользователь – это пользователь, не прошедший процедуру аутентификации, имеющий доступ только к функциям авторизации

Клиент – это физическое лицо, являющееся стороной в сделке с недвижимостью (покупатель, продавец, арендатор, арендодатель)

Заявка – это запрос от клиента на выполнение операции с недвижимостью (покупка, продажа, аренда, сдача в аренду), содержащий параметры поиска или данные об объекте

Сделка – это юридически оформленная операция с недвижимостью, связывающая покупателя и продавца, содержащая информацию о финальной цене, комиссии и статусе

Сервис сопоставления – это сервис, реализующий логику автоматического поиска и оценки совпадений между заявками на покупку/аренду и продажу/сдачу.

Словарь – это справочник системы, содержащий типы классификаторов (например, типы ремонта, виды санузлов, роли сотрудников).

Статус заявки – это этап жизненного цикла заявки: Новая, В процессе, Найдено совпадение, Завершена, Отменена, Истекла.

Тип заявки – это направление операции: аренда, сдача в аренду, покупка, продажа.

Участник сделки – это сторона, участвующая в сделке (покупатель/арендатор или продавец/арендодатель), связанная с конкретной заявкой.

Фотография – это изображение, привязанное к объекту системы (недвижимость, клиент, сотрудник), хранящееся в базе данных в бинарном формате.

Частный дом – это тип недвижимости, представляющий собой отдельно стоящее жилое здание, может иметь участок, гараж, подвал.

## Требования к информационной системе

Информационная система должна обладать следующей функциональностью, разделённую по ролям:

1. функциональность неавторизованного пользователя:
2. авторизация;
3. ограничение авторизации для уволенных и/или удалённых учётных записей;
4. функциональность для любой роли:
5. деаутентификация;
6. просмотр профиля;
7. смена пароля;
8. просмотр реестра заявок;
9. просмотр карточки заявки;
10. просмотр реестра сделок;
11. просмотр карточки сделки;
12. функциональность администратора:
13. просмотр реестра сотрудников;
14. просмотр карточки сотрудника;
15. перевод статуса сотрудника в «уволен»;
16. перевод статуса сотрудника в «нанят»;
17. удаление сотрудника;
18. восстановление сотрудника;
19. сброс пароля сотрудника;
20. функциональность для риэлтора:
21. создание карточки заявки;
22. редактирование карточки заявки;
23. просмотр предлагаемых системой заявок;
24. создание карточки сделки;
25. редактирование карточки сделки;
26. изменение статуса сделки (черновик, в процессе, отменена, завершена).

## Диаграммы вариантов использования

В соответствие с описанной ранее функциональностью были построены диаграммы вариантов использования для каждой из ролей (рисунок 2.3.1 – рисунок 2.3.4).



Рисунок 2.3.1 – Диаграмма вариантов использования неавторизованного пользователя

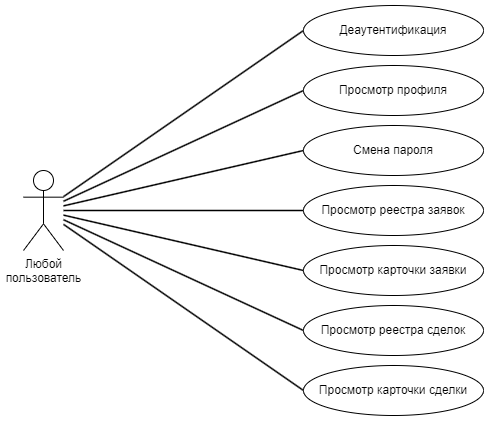


Рисунок 2.3.2 – Диаграмма вариантов использования для авторизованного пользователя



Рисунок 2.3.3 – Диаграмма вариантов использования администратора



Рисунок 2.3.4 – Диаграмма вариантов использования риэлтора

## Диаграммы состояний

Для демонстрации состояний объектов в системе были построены диаграммы состояний.

В системе присутствуют статусные модели данных о сотрудниках, сделках и заявках. Диаграммы состояний этих объектов представлены на рисунках 2.4.1 – 2.4.5.

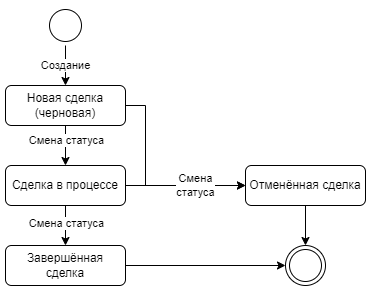


Рисунок 2.4.1 – Диаграмма состояний сделки

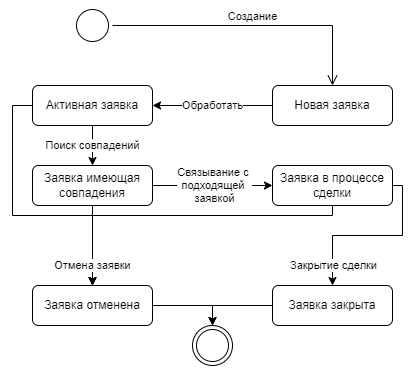


Рисунок 2.4.2 – Диаграмма состояний сделки

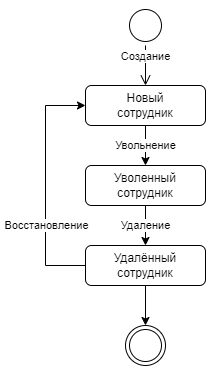


Рисунок 2.4.3 – Диаграмма состояний сотрудников

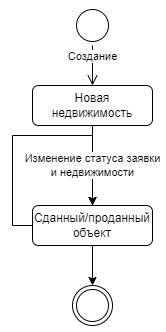


Рисунок 2.4.4 – Диаграмма состояний недвижимости

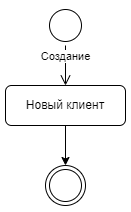


Рисунок 2.4.5 – Диаграмма состояний клиентов

## Схема данных

Для хранения данных была спроектирована и реализована база данных, логическая структура которой представлена на рисунке 2.5.1.



Рисунок 2.5.1 – Логическая модель базы данных

## Словарь данных

Таблица 2.6.1 – Словарь данных таблицы Addresses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | City | TEXT | N | Город |
|  | District | TEXT | N | Район |
|  | Street | TEXT | Y | Улица |
|  | HouseNumber | TEXT | N | Номер дома |
|  | BuildingNumber | TEXT | N | Номер корпуса/строения |
|  | PostalCode | TEXT | N | Почтовый индекс |
|  | IsDeleted | TEXT | Y | Флаг удаления (по умолчанию false) |

Таблица 2.6.2 – Словарь данных таблицы Dictionaries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | Type | TEXT | Y | Тип справочника |

Таблица 2.6.3 – Словарь данных таблицы DictionaryValues

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
| FK | DictionaryId | TEXT | Y | Ссылка на справочник |
|  | Value | TEXT | Y | Значение справочника |

Таблица 2.6.4 – Словарь данных таблицы Photos

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | EntityType | INTEGER | Y | Тип сущности |
| FK | EntityId | TEXT | Y | ID сущности |
|  | FileName | TEXT | Y | Имя файла |
|  | ContentType | TEXT | Y | MIME-тип файла |
|  | SortOrder | INTEGER | Y | Порядок сортировки |
|  | IsMain | BOOLEAN | Y | Флаг главного изображения |
|  | CreatedDate | TIMESTAMP | Y | Дата создания |
|  | FileData | BYTEA | Y | Данные файла (фотография) |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

Таблица 2.6.5 – Словарь данных таблицы Reviews

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | FirstName | TEXT | Y | Имя клиента |
|  | LastName | TEXT | Y | Фамилия клиента |
|  | MiddleName | TEXT | N | Отчество клиента |
|  | Phone | TEXT | Y | Номер телефона |
|  | PassportSeries | TEXT | N | Серия паспорта |
|  | PassportNumber | TEXT | N | Номер паспорта |
|  | RegistrationAddress | TEXT | N | Адрес регистрации |
| FK | PhotoId | TEXT | N | Ссылка на фото |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

Таблица 2.6.6 – Словарь данных таблицы Employees

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | Login | TEXT | Y | Логин |
|  | PasswordHash | TEXT | Y | Хеш пароля |
|  | Salt | TEXT | N | Соль для пароля |
|  | FirstName | TEXT | Y | Имя |
|  | LastName | TEXT | Y | Фамилия |
|  | MiddleName | TEXT | N | Отчество |
| FK | PhotoId | TEXT | N | Фото сотрудника |
| FK | RoleId | TEXT | N | Роль (из DictionaryValues) |
|  | LastAuthentication | TIMESTAMP | N | Дата последней аутентификации |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |
|  | Fired | BOOLEAN | Y | Флаг увольнения |

Таблица 2.6.7 – Словарь данных таблицы Realties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | Name | TEXT | Y | Название объекта |
|  | Description | TEXT | N | Описание |
| FK | AddressId | TEXT | Y | Адрес объекта |
| FK | ParentRealtyId | TEXT | N | Родительский объект недвижимости |
|  | IsActive | BOOLEAN | Y | Активен ли объект |
|  | RealtyType | INTEGER | Y | Тип недвижимости |
|  | TotalArea | NUMERIC | N | Общая площадь |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

Таблица 2.6.8 – Словарь данных таблицы Apartments

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ (наследуется от Realties) |
|  | ApartmentNumber | TEXT | N | Номер квартиры |
|  | Floor | INT | N | Этаж |
|  | RoomsCount | INT | N | Количество комнат |
|  | LivingArea | NUMERIC | N | Жилая площадь |
|  | KitchenArea | NUMERIC | N | Площадь кухни |
|  | HasBalcony | BOOLEAN | Y | Наличие балкона |
|  | HasLoggia | BOOLEAN | Y | Наличие лоджии |
| FK | RenovationTypeId | TEXT | N | Тип ремонта |
| FK | BathroomTypeId | TEXT | N | Тип санузла |

Таблица 2.6.9 – Словарь данных таблицы PrivateHouses

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ (наследуется от Realties) |
|  | RoomsCount | INTEGER | N | Количество комнат |
|  | FloorsCount | INTEGER | N | Количество этажей |
|  | HasGarage | BOOLEAN | N | Наличие гаража |
|  | HasBasement | BOOLEAN | N | Наличие подвала |
| FK | HeatingTypeId | TEXT | N | Тип отопления |
| FK | ConstructionMaterialId | TEXT | N | Материал постройки |
|  | YearBuilt | INTEGER | N | Год постройки |

Таблица 2.6.10 – Словарь данных таблицы Areas

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ (наследуется от Realties) |
|  | Square | NUMERIC | Y | Площадь участка |
|  | HasUtilities | BOOLEAN | Y | Наличие коммуникаций |
| FK | LandCategoryId | TEXT | N | Категория земли |

Таблица 2.6.11 – Словарь данных таблицы Listings

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
| FK | RealtyId | TEXT | Y | Объект недвижимости |
| FK | OwnerId | TEXT | Y | Владелец (клиент) |
| FK | ResponsibleEmployeeId | TEXT | N | Ответственный сотрудник |
|  | Price | NUMERIC | Y | Цена предложения |
| FK | CurrencyId | TEXT | N | Валюта |
| FK | ListingTypeId | TEXT | N | Тип предложения |
| FK | StatusId | TEXT | N | Статус предложения |
|  | Terms | TEXT | N | Условия сделки |
|  | CreatedDate | TIMESTAMP | Y | Дата создания |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

Таблица 2.6.12 – Словарь данных таблицы Deals

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
| FK | ListingId | TEXT | Y | Предложение |
| FK | BuyerId | TEXT | Y | Покупатель (клиент) |
| FK | EmployeeId | TEXT | N | Сотрудник, оформивший сделку |
|  | FinalPrice | NUMERIC | Y | Финальная цена |
|  | Commission | NUMERIC | Y | Комиссия агентства |
|  | DealDate | TIMESTAMP | Y | Дата сделки |
| FK | DealTypeId | TEXT | N | Тип сделки |
| FK | StatusId | TEXT | Y | Статус сделки |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

Таблица 2.6.13 – Словарь данных таблицы ClientRequests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
|  | Type | INTEGER | Y | Тип запроса |
|  | Status | INTEGER | N | Статус запроса |
|  | CreatedDate | TIMESTAMP | N | Дата создания |
|  | CompletedDate | TIMESTAMP | N | Дата завершения |
| FK | ClientId | TEXT | N | Клиент |
| FK | EmployeeId | TEXT | N | Ответственный сотрудник |
|  | MaxPrice | NUMERIC | N | Максимальная цена |
|  | MinRooms | INTEGER | N | Минимальное количество комнат |
|  | MinArea | NUMERIC | N | Минимальная площадь |
|  | MaxArea | NUMERIC | N | Максимальная площадь |
|  | DesiredLocation | TEXT | N | Желаемый район/адрес |
|  | AdditionalRequirements | TEXT | N | Дополнительные требования |
| FK | ListingId | TEXT | N | Подобранное предложение |
| FK | MatchedRequestId | TEXT | N | Сопоставленный запрос |
| FK | DealId | TEXT | N | Сделка (если состоялась) |
|  | DesiredRealtyType | INTEGER | N | Желаемый тип недвижимости |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

Таблица 2.6.14 – Словарь данных таблицы DealParticipants

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ключ | Имя поля | Тип данных | Обязательно? | Примечание |
| PK | Id | TEXT | Y | Первичный ключ |
| FK | DealId | TEXT | Y | Сделка |
| FK | ClientRequestId | TEXT | Y | Запрос клиента |
|  | Role | INTEGER | Y | Роль в сделке |
|  | IsDeleted | BOOLEAN | Y | Флаг удаления |

## Пользовательский сценарий

При запуске системы открывается окно авторизации, в котором обязательно заполняются поля «логин» и «пароль», после чего нажимается кнопка «войти». Если учётная запись не переведена в статус уволена или не была удалена, то при корректном вводе данных система авторизует учётную запись пользователя.

После успешной авторизации пользователя встречает главное окно с кнопками выбора страниц в верхней панели. Помимо этих кнопок на панели есть кнопка выхода из аккаунта. Для перехода в свой профиль, пользователь может нажать на кнопку «профиль» после чего откроется страница профиля, на которой можно отредактировать свои данные. Также в случае входа в учётную запись с ролю «администратор», будет доступна кнопка «сотрудники», по нажатии на которую откроется окно с реестром и добавлением сотрудников. Добавление сотрудника происходит после заполнения полей, добавления фото и указания роли, после нажатия на кнопку «добавить». Также на этой странице можно выбрать одного из сотрудников в реестре и перейти в карточку сотрудника. В карточке сотрудника помимо просмотра данных возможно перевести сотрудника в статус «уволенные», удалить, а также сбросить пароль.

Для автоматизации основного бизнес-процесса риелторского агентства, пользователь должен авторизоваться в роли риэлтора и нажать на кнопку «Создать заявку», после чего откроется соответствующая страница. На этой странице необходимо указать данные о клиенте, операции и объекте недвижимости, с которым происходят операции. В конце нужно нажать на кнопку «создать», после чего заявка сохранится и произойдёт автоматический подбор заявок с соответствующими по различным параметрам заявки от других клиентов.

Для создания сделки можно найти подходящую сделку либо сразу при создании заявки, либо через переход в карточку заявки, где предложения также будут показаны. На предложение имеется кнопка «создать сделку». В случае если две заявки подходят друг другу, то риэлтор создаёт сделку в статусе «черновик».

Для просмотра сделок пользователь переходит в реестр сделок, по нажатию на кнопку «сделки», после чего может выбрать любую карточку и перейти к её детальному описанию. В случае если пользователь является риэлтором, то пользователь может изменять статусы сделки через нажатие на кнопку «редактировать», после чего изменяется статус и нажимается кнопка «сохранить».

## Прототипы интерфейсов

Для реализации интерфейсов приложения в начале нужно построить прототипы интерфейсов, которые представлены на рисунках 2.8.1 – 2.8.8.

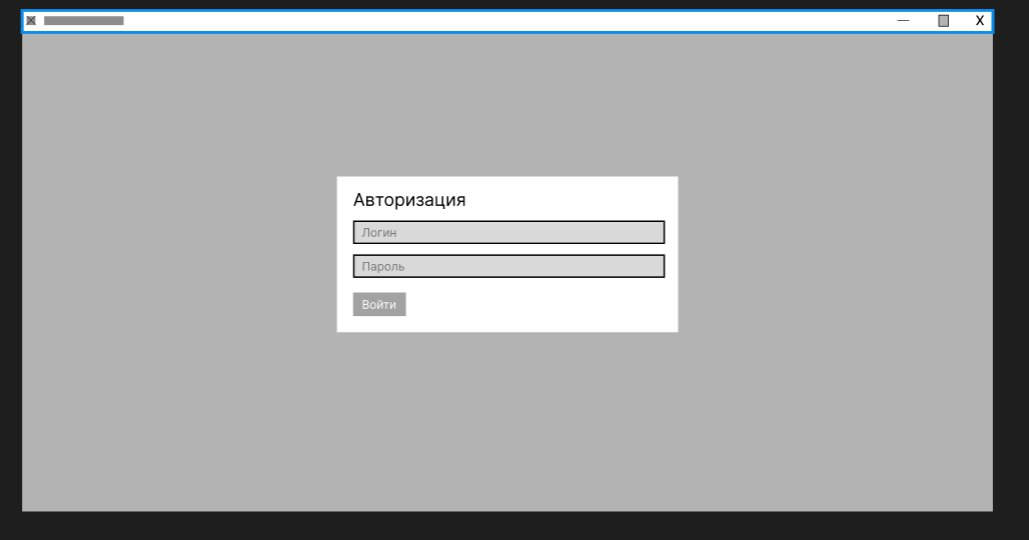


Рисунок 2.8.1 – Прототип окна авторизации

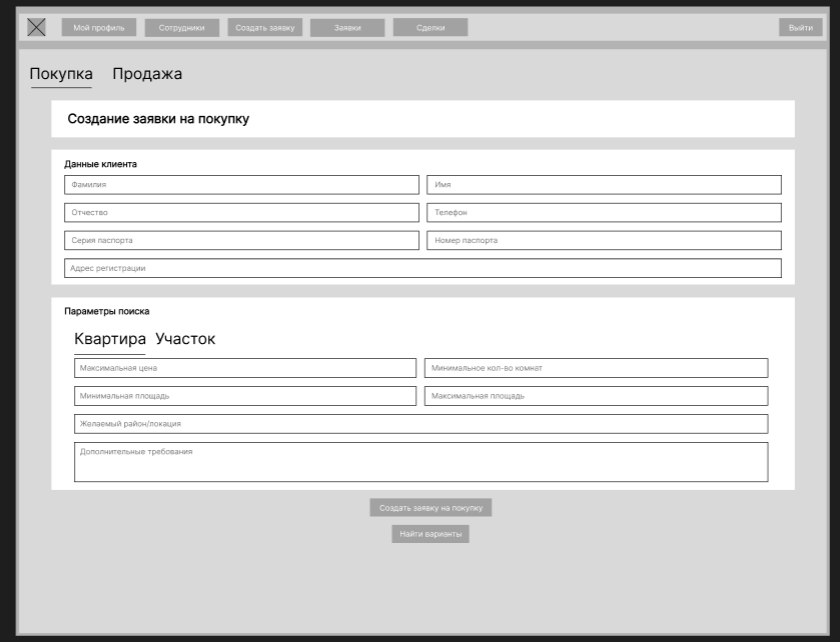


Рисунок 2.8.2 – Прототип окна создания заявки о покупке квартиры

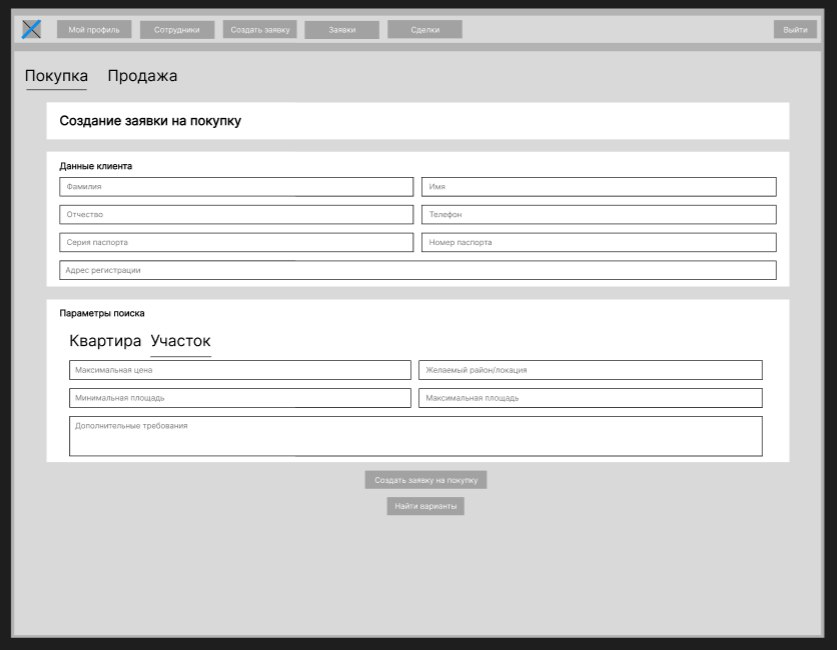


Рисунок 2.8.3 – Прототип окна создания заявки о покупке земельного участка

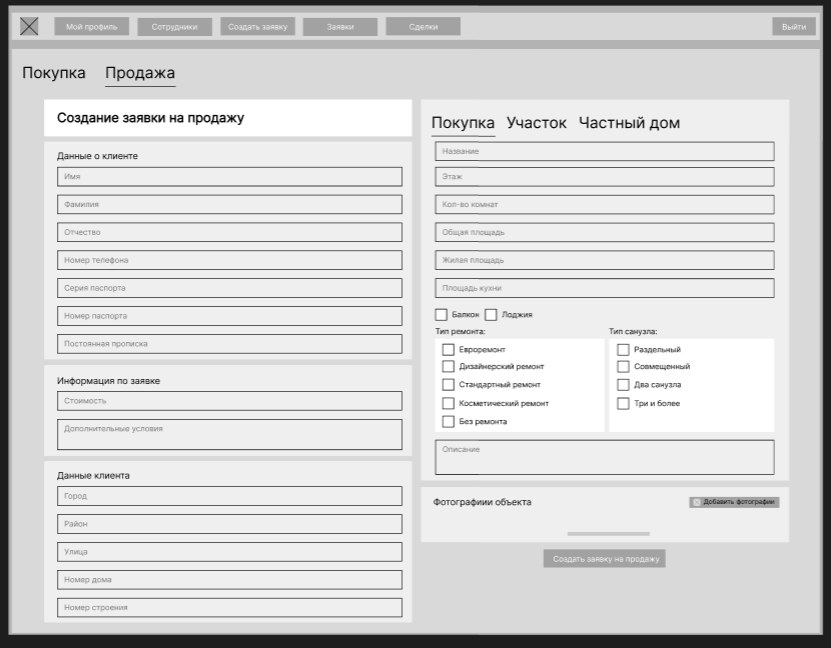


Рисунок 2.8.4 – Прототип окна создания заявки о продаже квартиры

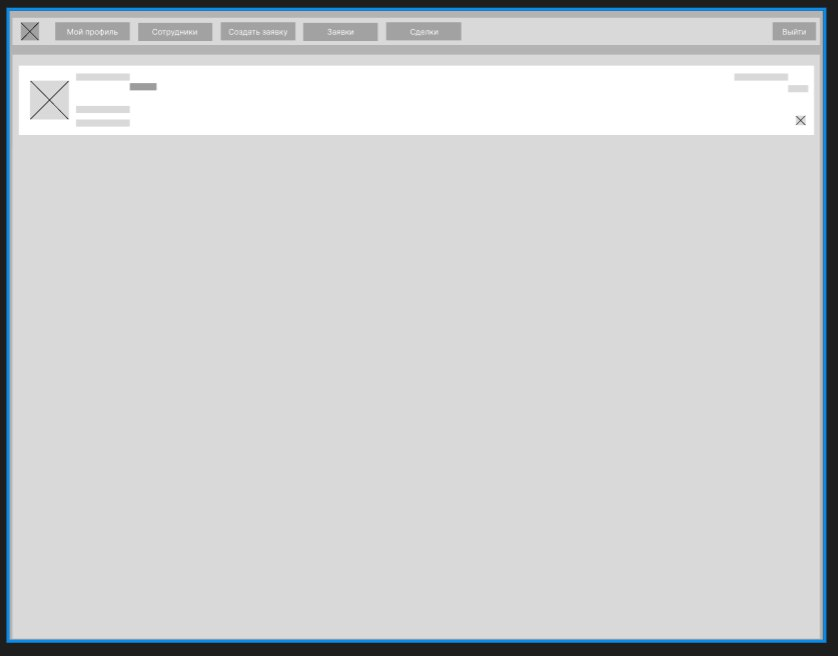


Рисунок 2.8.5 – Прототип окна реестра заявок



Рисунок 2.8.6 – Прототип окна реестра сделок

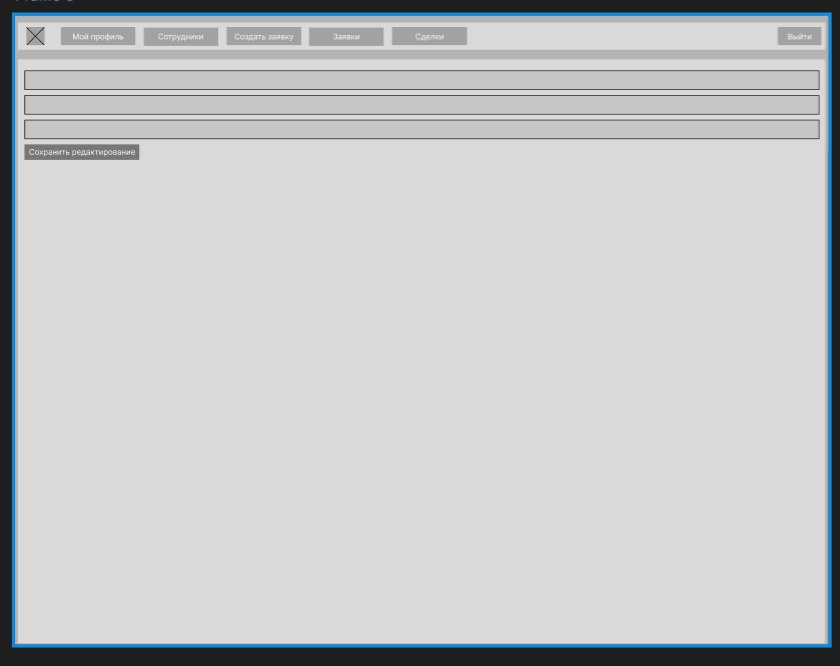


Рисунок 2.8.7 – Прототип окна реестра заявок

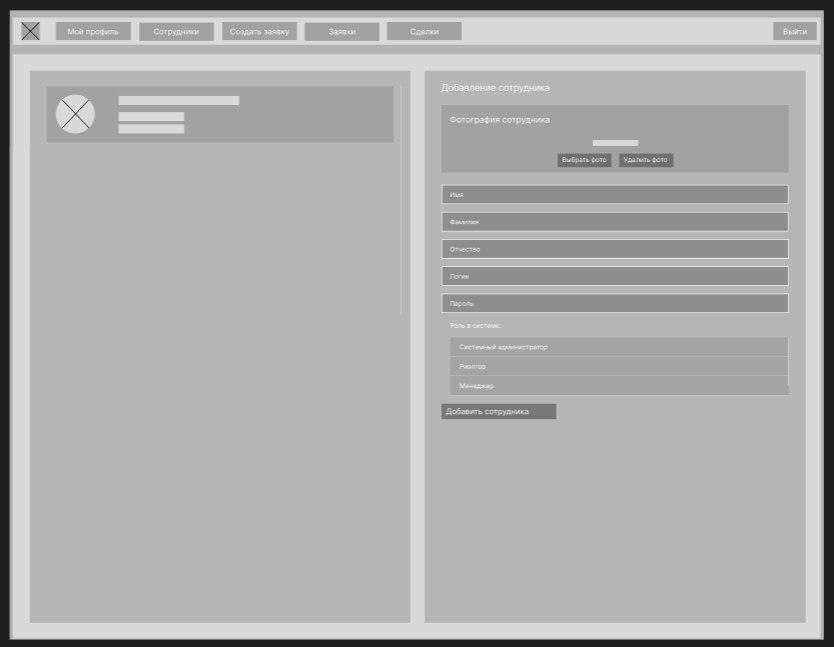


Рисунок 2.8.8 – Прототип окна реестра сотрудников

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данного курсового проекта была достигнута основная цель – разработано кроссплатформенное настольное приложение «Realtor tool», призванное комплексно решить актуальные проблемы риэлторских агентств в сфере управления данными, клиентским обслуживанием и финансовым учетом.

Проведенная работа подтвердила, что автоматизация ключевых бизнес-процессов является необходимым условием для повышения конкурентоспособности на современном рынке недвижимости. Разработанное приложение интегрирует в себе функциональные модули для управления базой объектов недвижимости, персоналом, акционными предложениями и финансовыми операциями, что позволяет отказаться от неэффективной бумажной работы.

Ожидаемые результаты от внедрения данного программного решения заключаются в значительном повышении качества обслуживания клиентов за счет ускорения обработки их запросов и повышения прозрачности сделок. Одновременно с этим оптимизация внутренних процессов, таких как учет и распределение задач между агентами, позволит повысить производительность труда сотрудников и, как следствие, общую прибыльность агентства.

Таким образом, реализация проекта «Realtor tool» демонстрирует, что использование специализированного программного обеспечения является мощным инструментом для трансформации работы риэлторского бизнеса, перевода его на качественно новый уровень эффективности и клиентоориентированности. В перспективе приложение может быть доработано за счет интеграции с онлайн-базами данных, внедрения систем аналитики и мобильного доступа для клиентов, что откроет дополнительные возможности для роста и развития агентства.

# ПЕРЕЧЕНЬ ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1) Литвиненко Иван. Руководство по использованию Avalonia UI в приложениях на C# / Литвиненко Иван. — [Онлайн]. — URL: https://avaloniaui.net/ (дата доступа: 29-08-2025).

2) Краткий обзор языка C# – Текст: Электронный // Microsoft – [сайт] URL:https://learn.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/ Дата обращения: 29-08-2025.

3) Открытие общего диалогового окна (WPF .NET) Текст: Электронный / Microsoft – [сайт] URL: https://learn.microsoft.com/ru-ru/dotnet/desktop/wpf/windows/how-to-open-common-system-dialog-box?view=netdesktop-8.0 Дата обращения: 29-08-2025.

4) Иэн Гриффитс. Программируем на С# 8.0 / Иэн Гриффитс – СПб: Питер, 2021. – 944 с. – Текст: непосредственный .

5) Джозеф Албахари, Бен Албахари. C# 9.0. Карманный справочник / Албахари Джозеф, Албахари Бен – СПб: Питер, 2021. – 256 c. – Текст : непосредственный.

6) Роб Майлз. The C# Programming Yellow Book / Роб Майлз – eBook, 2015. – 222 c. – Текст: непосредственный.

7) Статья «Основы программирования на C#» на портале C# Corner / Автор не указан. — [Онлайн]. — URL: https://www.c-sharpcorner.com/ (дата обращения: 29-08-2025).

8) Документация Avalonia UI на официальном сайте проекта / Avalonia UI Contributors. — [Онлайн]. — URL: https://avaloniaui.net/docs/ (дата обращения: 29-08-2025).

9) Статья «Разработка приложений с использованием Avalonia UI» на сайте «Medium» / Автор не указан. — [Онлайн]. — URL: https://medium.com/ (дата доступа: 29-08-2025).

10) Видеоуроки по Avalonia UI на платформе YouTube / Автор не указан. — [Онлайн]. — URL: https://www.youtube.com/ (дата обращения: 29-08-2025).

11) Форум Avalonia UI на GitHub / Avalonia UI Contributors. — [Онлайн]. — URL: https://github.com/AvaloniaUI/Avalonia/discussions (дата обращения: 29-08-2025).

12) Статья «Программирование на C#» на сайте Microsoft Docs / Microsoft Corporation. — [Онлайн]. — URL: https://docs.microsoft.com/ru-ru/dotnet/csharp/ (дата доступа: 29-08-2025).

13) Блог Avalonia UI на официальном сайте проекта / Avalonia UI Contributors. — [Онлайн]. — URL: https://avaloniaui.net/blog/ (дата обращения: 29-08-2025).

14) Вебинары по разработке на C# и Avalonia UI на платформе Webinar.ru / Автор не указан. — [Онлайн]. — URL: https://webinar.ru/ (дата обращения: 29-08-2025).

15) Книга «Разработка приложений на C# с использованием Avalonia UI» на сайте издательства «Лори» / Автор не указан. — [Онлайн]. — URL: https://lori.ru/ (дата обращения: 29-08-2025).

# Инструкция пользователя

При запуске системы открывается окно авторизации, в котором обязательно заполняются поля «логин» и «пароль», после чего нажимается кнопка «войти». Если учётная запись не переведена в статус уволена или не была удалена, то при корректном вводе данных система авторизует учётную запись пользователя.

После успешной авторизации пользователя встречает главное окно с кнопками выбора страниц в верхней панели. Помимо этих кнопок на панели есть кнопка выхода из аккаунта. Для перехода в свой профиль, пользователь может нажать на кнопку «профиль» после чего откроется страница профиля, на которой можно отредактировать свои данные. Также в случае входа в учётную запись с ролю «администратор», будет доступна кнопка «сотрудники», по нажатии на которую откроется окно с реестром и добавлением сотрудников. Добавление сотрудника происходит после заполнения полей, добавления фото и указания роли, после нажатия на кнопку «добавить». Также на этой странице можно выбрать одного из сотрудников в реестре и перейти в карточку сотрудника. В карточке сотрудника помимо просмотра данных возможно перевести сотрудника в статус «уволенные», удалить, а также сбросить пароль.

Для автоматизации основного бизнес-процесса риелторского агентства, пользователь должен авторизоваться в роли риэлтора и нажать на кнопку «Создать заявку», после чего откроется соответствующая страница. На этой странице необходимо указать данные о клиенте, операции и объекте недвижимости, с которым происходят операции. В конце нужно нажать на кнопку «создать», после чего заявка сохранится и произойдёт автоматический подбор заявок с соответствующими по различным параметрам заявки от других клиентов.

Для создания сделки можно найти подходящую сделку либо сразу при создании заявки, либо через переход в карточку заявки, где предложения также будут показаны. На предложение имеется кнопка «создать сделку». В случае если две заявки подходят друг другу, то риэлтор создаёт сделку в статусе «черновик».

Для просмотра сделок пользователь переходит в реестр сделок, по нажатию на кнопку «сделки», после чего может выбрать любую карточку и перейти к её детальному описанию. В случае если пользователь является риэлтором, то пользователь может изменять статусы сделки через нажатие на кнопку «редактировать», после чего изменяется статус и нажимается кнопка «сохранить».

# Тестирование

Таблица Б.1 – Тестовый пример №1

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_1 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Авторизация |
| Краткое изложение теста | Вход в приложение |
| Этапы теста | 1. Открыть окно авторизации 2. Ввести логин и пароль 3. Нажать кнопку «войти» |
| Тестовые данные | 1. Поле ввода «логин» 2. Поле ввода «пароль» |
| Ожидаемый результат | Вход в приложение |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | Приложение запущено |
| Постусловие | Открыта главная страница |
| Примечания/комментарии | Отсутствуют |

Таблица Б.2 – Тестовый пример №2

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_2 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Добавление сотрудника |
| Краткое изложение теста | Возможность добавлять сотрудника |
| Этапы теста | 1. Перейти на страницу сотрудники 2. Заполнить данные (по желанию добавить фото) 3. Указать роль в системе 4. Нажать «добавить сотрудника» |
| Тестовые данные | 1. Фотография сотрудника 2. Поле ввода «имя» 3. Поле ввода «фамилия» 4. Поле ввода «отчество» 5. Поле ввода «логин» 6. Поле ввода «пароль» 7. Роль в системе с возможностью выбора |
| Ожидаемый результат | Сотрудник добавлен |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | 1. Вход в приложение выполнен 2. Открыта главная страница |
| Постусловие | Открыта страница «сотрудники» |
| Примечания/комментарии | Отсутствуют |

Таблица Б.3 – Тестовый пример №3

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_3 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Создание заявки |
| Краткое изложение теста | Создание заявки |
| Этапы теста | 1. Перейти на страницу «Создать заявку» 2. Заполнить данные клиента 3. Заполнить данные квартира/участок 4. Нажимаем «создать заявку» |
| Тестовые данные | 1. Поля ввода с информацией клиента 2. Поля ввода с информацией квартиры/участка |
| Ожидаемый результат | Заявка создана и выведен список совпадений |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | 1. Выполнен вход в приложение |
| Постусловие | Открыта страница «создание заявки» и выведен список вариантов |
| Примечания/комментарии | Отсутствуют |

Таблица Б.4 – Тестовый пример №4

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_4 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Сохранность заявки |
| Краткое изложение теста | Проверка наличия созданной заявки и сохранность ее наполнения |
| Этапы теста | 1. Перейти на страницу «заявки» 2. Проверить наличие созданной заявки 3. Нажать на созданную заявку 4. Проверить наличие введенных данных |
| Тестовые данные | Заявка |
| Ожидаемый результат | Заявка сохранена со всеми данными |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | 1. Выполнен вход в приложение 2. Заявка создана |
| Постусловие | Открыта страница заявки |
| Примечания/комментарии | Отсутствуют |

Таблица Б.5 – Тестовый пример №5

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_5 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Создание сделки |
| Краткое изложение теста | Возможность создавать сделки по заявке |
| Этапы теста | 1. Перейти на страницу «заявки» 2. Выбрать заявку для сделки и перейти на нее 3. Нажать кнопку «создать сделку» 4. Подтвердить действие во всплывающем окне |
| Тестовые данные | 1. Создана заявка 2. Заявка актуальна |
| Ожидаемый результат | Сделка совершена |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | 1. Вход в приложение выполнен 2. Создана хоть одно заявка 3. Заявка актуальна |
| Постусловие | Открыта страница заявки |
| Примечания/комментарии | Отсутствуют |

Таблица Б.6 – Тестовый пример №6

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_6 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Наличие сделки |
| Краткое изложение теста | Проверка наличия созданной сделки |
| Этапы теста | 1. Перейти на страницу «сделки» 2. Найти созданную сделку и перейти в нее 3. Проверить наличие данных |
| Тестовые данные | Создана сделка |
| Ожидаемый результат | Сделка успешно создана со всеми данными |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | 1. Выполнен вход в приложение 2. Сделка создана |
| Постусловие | Открыта страница сделки |
| Примечания/комментарии | Отсутствуют |

Таблица Б.7 – Тестовый пример №7

|  |  |
| --- | --- |
| Тестовый пример # | TC\_UI\_7 |
| Приоритет тестирования | Высокий |
| Заголовок/название теста | Проверка профиля |
| Краткое изложение теста | Проверка профиля на редактирование данных |
| Этапы теста | 1. Перейти на страницу «мой профиль» 2. Ввести данные 3. Нажать на кнопку «сохранить редактирование» |
| Тестовые данные | 1. Поля ввода 2. Кнопка «сохранить редактирование» |
| Ожидаемый результат | Данные успешно отредактированы |
| Фактический результат | Соответствует ожидаемому результату |
| Статус | Зачет |
| Предварительное условие | 1. Выполнен вход в приложение |
| Постусловие | Открыто страница «мой профиль» |
| Примечания/комментарии | Отсутствуют |

# Листинг кода

1. Листинг кода доменной модели «Apartament»

namespace RealtorTool.Core.DbEntities.RealtyModels;

public class Apartment : Realty

{

public string? ApartmentNumber { get; set; }

public int? Floor { get; set; }

public int? RoomsCount { get; set; }

public decimal? TotalArea { get; set; }

public decimal? LivingArea { get; set; }

public decimal? KitchenArea { get; set; }

public bool HasBalcony { get; set; }

public bool HasLoggia { get; set; }

public string? RenovationTypeId { get; set; }

public DictionaryValue? RenovationType { get; set; }

public string? BathroomTypeId { get; set; }

public DictionaryValue? BathroomType { get; set; }

}

1. Листинг кода доменной модели «Area»

namespace RealtorTool.Core.DbEntities.RealtyModels;

public class Area : Realty

{

public decimal Square { get; set; }

public string? LandCategoryId { get; set; }

public DictionaryValue? LandCategory { get; set; }

public bool HasUtilities { get; set; }

}

1. Листинг кода доменной модели «PrivateHouse»

namespace RealtorTool.Core.DbEntities.RealtyModels;

public class PrivateHouse : Realty, ISoftDelete

{

public int? RoomsCount { get; set; }

public decimal? TotalArea { get; set; }

public int? FloorsCount { get; set; }

public bool? HasGarage { get; set; }

public bool? HasBasement { get; set; }

public string? HeatingTypeId { get; set; }

public DictionaryValue? HeatingType { get; set; }

public string? ConstructionMaterialId { get; set; }

public DictionaryValue? ConstructionMaterial { get; set; }

public int? YearBuilt { get; set; }

}

1. Листинг кода доменной модели «Realty»

using RealtorTool.Core.Enums;

namespace RealtorTool.Core.DbEntities;

public class Realty : BaseIdEntity, ISoftDelete

{

public string Name { get; set; } = null!;

public string? Description { get; set; }

public decimal? TotalArea { get; set; }

public string AddressId { get; set; } = null!;

public Address Address { get; set; } = null!;

public string? ParentRealtyId { get; set; }

public Realty? ParentRealty { get; set; }

public bool IsActive { get; set; } = true;

public RealtyType RealtyType { get; set; }

public ICollection<Realty>? ChildRealties { get; set; } = new List<Realty>();

public ICollection<Listing> Listings { get; set; } = new List<Listing>();

public ICollection<Photo> Photos { get; set; } = new List<Photo>();

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «PotentialMatch»

using RealtorTool.Core.Enums;

namespace RealtorTool.Core.DbEntities.Views;

public class PotentialMatch

{

public string BuyRequestId { get; set; }

public string SellRequestId { get; set; }

public string BuyerId { get; set; }

public string SellerId { get; set; }

public decimal? MaxPrice { get; set; }

public decimal ListingPrice { get; set; }

public string RealtyId { get; set; }

public string SellListingId { get; set; }

public RealtyType RealtyType { get; set; }

public decimal? TotalArea { get; set; }

public string RealtyName { get; set; }

public string City { get; set; }

public string District { get; set; }

public int? MinRooms { get; set; }

public decimal? MinArea { get; set; }

public decimal? MaxArea { get; set; }

public string DesiredLocation { get; set; }

public int? DesiredRealtyType { get; set; }

public int? ActualRoomsCount { get; set; }

public bool TypeMatch { get; set; }

public int MatchScore { get; set; }

public bool PriceMatch { get; set; }

public bool MinAreaMatch { get; set; }

public bool MaxAreaMatch { get; set; }

public bool RoomsMatch { get; set; }

public bool LocationMatch { get; set; }

public ApplicationType BuyType { get; set; }

public ApplicationType SellType { get; set; }

public bool IsGoodMatch => MatchScore >= 70;

public bool IsPerfectMatch => MatchScore >= 90;

public decimal? PriceDifference => MaxPrice.HasValue ? ListingPrice – MaxPrice.Value : null;

public bool IsWithinBudget => !MaxPrice.HasValue || ListingPrice <= MaxPrice.Value;

public string RealtyTypeDisplayName => RealtyType switch

{

RealtyType.Apartment => "Квартира",

RealtyType.PrivateHouse => "Частный дом",

RealtyType.Area => "Земельный участок",

\_ => "Неизвестный тип"

};

public string MatchDescription

{

get

{

if (MatchScore >= 90) return "★ Идеальное совпадение";

if (MatchScore >= 70) return "✓ Хорошее совпадение";

if (MatchScore >= 50) return "○ Возможное совпадение";

return "△ Слабое совпадение";

}

}

}

1. Листинг кода доменной модели «Address»

namespace RealtorTool.Core.DbEntities;

public class Address : BaseIdEntity, ISoftDelete

{

public string? City { get; set; }

public string? District { get; set; }

public string Street { get; set; } = null!;

public string? HouseNumber { get; set; }

public string? BuildingNumber { get; set; }

public string? PostalCode { get; set; }

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «BaseIdEntity»

namespace RealtorTool.Core.DbEntities;

public class BaseIdEntity

{

public string Id { get; set; } = Guid.NewGuid().ToString();

}

1. Листинг кода доменной модели «Client»

using System.ComponentModel.DataAnnotations.Schema;

namespace RealtorTool.Core.DbEntities;

public class Client : BaseIdEntity, ISoftDelete

{

public string FirstName { get; set; } = null!;

public string LastName { get; set; } = null!;

public string? MiddleName { get; set; }

public string Phone { get; set; } = null!;

public string? PassportSeries { get; set; }

public string? PassportNumber { get; set; }

public string? RegistrationAddress { get; set; }

public string? PhotoId { get; set; }

public Photo? Photo { get; set; }

public bool IsDeleted { get; set; }

[NotMapped]

public string FullName => $"{FirstName} {LastName}";

}

1. Листинг кода доменной модели «ClientRequest»

using System.ComponentModel.DataAnnotations.Schema;

using RealtorTool.Core.Enums;

namespace RealtorTool.Core.DbEntities;

/// <summary>

/// Запрос клиента на операции с недвижимостью

/// </summary>

public class ClientRequest : BaseIdEntity, ISoftDelete

{

/// <summary>

/// Тип запроса (аренда, сдача, покупка, продажа)

/// </summary>

public ApplicationType Type { get; set; }

/// <summary>

/// Текущий статус запроса

/// </summary>

public ApplicationStatus Status { get; set; }

/// <summary>

/// Дата создания запроса

/// </summary>

public DateTime CreatedDate { get; set; } = DateTime.UtcNow;

/// <summary>

/// Дата завершения запроса

/// </summary>

public DateTime? CompletedDate { get; set; }

/// <summary>

/// Идентификатор клиента-инициатора запроса

/// </summary>

public string ClientId { get; set; } = null!;

/// <summary>

/// Клиент-инициатор запроса

/// </summary>

public Client Client { get; set; } = null!;

/// <summary>

/// Идентификатор сотрудника, сопровождающего запрос

/// </summary>

public string EmployeeId { get; set; } = null!;

/// <summary>

/// Сотрудник, сопровождающий запрос

/// </summary>

public Employee Employee { get; set; } = null!;

// Параметры поиска для запросов на покупку/аренду

/// <summary>

/// Максимальная цена для запросов на покупку/аренду

/// </summary>

public decimal? MaxPrice { get; set; }

/// <summary>

/// Минимальное количество комнат

/// </summary>

public int? MinRooms { get; set; }

/// <summary>

/// Минимальная площадь

/// </summary>

public decimal? MinArea { get; set; }

/// <summary>

/// Максимальная площадь

/// </summary>

public decimal? MaxArea { get; set; }

/// <summary>

/// Желаемое расположение

/// </summary>

public string? DesiredLocation { get; set; }

/// <summary>

/// Дополнительные требования клиента

/// </summary>

public string? AdditionalRequirements { get; set; }

/// <summary>

/// Идентификатор объявления (для запросов на продажу/сдачу)

/// </summary>

public string? ListingId { get; set; }

/// <summary>

/// Объявление связанное с запросом

/// </summary>

public Listing? Listing { get; set; }

/// <summary>

/// Идентификатор встречного запроса при успешном matching

/// </summary>

public string? MatchedRequestId { get; set; }

/// <summary>

/// Встречный запрос

/// </summary>

public ClientRequest? MatchedRequest { get; set; }

/// <summary>

/// Идентификатор сделки

/// </summary>

public string? DealId { get; set; }

/// <summary>

/// Сделка по запросу

/// </summary>

public Deal? Deal { get; set; }

/// <summary>

/// Желаемый тип недвижимости (для запросов на покупку/аренду)

/// </summary>

public RealtyType? DesiredRealtyType { get; set; }

public bool IsDeleted { get; set; }

[NotMapped] public Realty? Realty => Listing?.Realty;

}

1. Листинг кода доменной модели «Deal»

namespace RealtorTool.Core.DbEntities;

public class Deal : BaseIdEntity, ISoftDelete

{

public string ListingId { get; set; } = null!;

public Listing Listing { get; set; } = null!;

public string BuyerId { get; set; } = null!;

public Client Buyer { get; set; } = null!;

public string? EmployeeId { get; set; }

public Employee? Employee { get; set; }

public decimal FinalPrice { get; set; }

public decimal Commission { get; set; }

public DateTime DealDate { get; set; }

public string? DealTypeId { get; set; }

public DictionaryValue? DealType { get; set; }

public string StatusId { get; set; } = null!;

public DictionaryValue Status { get; set; }

public ICollection<DealParticipant> Participants { get; set; } = new List<DealParticipant>();

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «DealParticipant»

using RealtorTool.Core.Enums;

namespace RealtorTool.Core.DbEntities;

public class DealParticipant : BaseIdEntity, ISoftDelete

{

public string DealId { get; set; } = null!;

public Deal Deal { get; set; } = null!;

public string ClientRequestId { get; set; } = null!;

public ClientRequest ClientRequest { get; set; } = null!;

public ParticipantRole Role { get; set; }

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «Dictionary»

namespace RealtorTool.Core.DbEntities;

public class Dictionary : BaseIdEntity

{

public string Type { get; set; } = null!;

}

1. Листинг кода доменной модели «DictionaryValue»

using System.ComponentModel.DataAnnotations.Schema;

namespace RealtorTool.Core.DbEntities;

public class DictionaryValue : BaseIdEntity

{

public string DictionaryId { get; set; } = null!;

public Dictionary? Dictionary { get; set; }

public string Value { get; set; } = null!;

[NotMapped] public bool IsSelected { get; set; }

}

1. Листинг кода доменной модели «Employee»

using System.ComponentModel.DataAnnotations.Schema;

namespace RealtorTool.Core.DbEntities;

public class Employee : BaseIdEntity, ISoftDelete

{

public string Login { get; set; } = null!;

public string PasswordHash { get; set; } = null!;

public string? Salt { get; set; }

public string FirstName { get; set; } = null!;

public string LastName { get; set; } = null!;

public string? MiddleName { get; set; }

public DateTime? LastAuthentication { get; set; }

public string? RoleId { get; set; }

public DictionaryValue? Role { get; set; }

public string? PhotoId { get; set; }

public Photo? Photo { get; set; }

public bool IsDeleted { get; set; }

public bool Fired { get; set; }

[NotMapped]

public bool HasPhoto => Photo != null;

[NotMapped]

public string FullName => $"{LastName} {FirstName} {MiddleName}";

}

1. Листинг кода доменной модели «ISoftDelete»

namespace RealtorTool.Core.DbEntities;

public interface ISoftDelete

{

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «Listing»

namespace RealtorTool.Core.DbEntities;

public class Listing : BaseIdEntity, ISoftDelete

{

public string RealtyId { get; set; } = null!;

public Realty Realty { get; set; } = null!;

public string OwnerId { get; set; } = null!;

public Client Owner { get; set; } = null!;

public string? ResponsibleEmployeeId { get; set; }

public Employee? ResponsibleEmployee { get; set; }

public decimal Price { get; set; }

public string? CurrencyId { get; set; }

public DictionaryValue? Currency { get; set; }

public string? ListingTypeId { get; set; }

public DictionaryValue ListingType { get; set; } = null!;

public string? StatusId { get; set; }

public DictionaryValue? Status { get; set; }

public string? Terms { get; set; }

public DateTime CreatedDate { get; set; } = DateTime.UtcNow;

public ICollection<ClientRequest> ClientRequests { get; set; } = new List<ClientRequest>();

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «Photo»

namespace RealtorTool.Core.DbEntities;

public class Listing : BaseIdEntity, ISoftDelete

{

public string RealtyId { get; set; } = null!;

public Realty Realty { get; set; } = null!;

public string OwnerId { get; set; } = null!;

public Client Owner { get; set; } = null!;

public string? ResponsibleEmployeeId { get; set; }

public Employee? ResponsibleEmployee { get; set; }

public decimal Price { get; set; }

public string? CurrencyId { get; set; }

public DictionaryValue? Currency { get; set; }

public string? ListingTypeId { get; set; }

public DictionaryValue ListingType { get; set; } = null!;

public string? StatusId { get; set; }

public DictionaryValue? Status { get; set; }

public string? Terms { get; set; }

public DateTime CreatedDate { get; set; } = DateTime.UtcNow;

public ICollection<ClientRequest> ClientRequests { get; set; } = new List<ClientRequest>();

public bool IsDeleted { get; set; }

}

1. Листинг кода доменной модели «ApplicationStatus»

namespace RealtorTool.Core.Enums;

public enum ApplicationStatus

{

New,

InProgress,

Matched,

Completed,

Canceled,

Expired

}

1. Листинг кода доменной модели «ApplicationType»

namespace RealtorTool.Core.Enums;

/// <summary>

/// Типы сделок заявки.

/// </summary>

public enum ApplicationType

{

/// <summary>

/// Аренда

/// </summary>

Rent,

/// <summary>

/// Сдача в аренду

/// </summary>

RentOut,

/// <summary>

/// Покупка

/// </summary>

Purchase,

/// <summary>

/// Продажа

/// </summary>

Sale

}

1. Листинг кода доменной модели «EmployeeRole»

namespace RealtorTool.Core.Enums;

public enum EmployeeRole

{

/// <summary>

/// Система

/// </summary>

SystemRole,

/// <summary>

/// Системный администратор

/// </summary>

SystemAdministrator,

/// <summary>

/// Риэлтор

/// </summary>

Realtor,

/// <summary>

/// Менеджер

/// </summary>

Manager

}

1. Листинг кода доменной модели «EntityTypeForPhoto»

namespace RealtorTool.Core.Enums;

public enum EntityTypeForPhoto

{

Realty,

Client,

Employee

}

1. Листинг кода доменной модели «ParticipantRole»

namespace RealtorTool.Core.Enums;

public enum ParticipantRole

{

Buyer, // Покупатель/Арендатор

Seller // Продавец/Арендодатель

}

1. Листинг кода доменной модели «RealtyType»

namespace RealtorTool.Core.Enums;

public enum RealtyType

{

Apartment,

Area,

PrivateHouse

}

1. Листинг кода «DataContex»

using Microsoft.EntityFrameworkCore;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.RealtyModels;

using RealtorTool.Core.DbEntities.Views;

namespace RealtorTool.Data.Context;

public class DataContext : DbContext

{

public DataContext()

{

}

public DataContext(DbContextOptions<DataContext> options)

: base(options)

{

}

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseNpgsql("Server=localhost;port=5415;user id=postgres;password=toor;database=RT;");

public DbSet<Address> Addresses { get; set; } = null!;

public DbSet<Client> Clients { get; set; } = null!;

public DbSet<Employee> Employees { get; set; } = null!;

public DbSet<Dictionary> Dictionaries { get; set; } = null!;

public DbSet<DictionaryValue> DictionaryValues { get; set; } = null!;

public DbSet<Photo> Photos { get; set; } = null!;

public DbSet<ClientRequest> ClientRequests { get; set; } = null!;

// Недвижимость и её наследники (TPT стратегия)

public DbSet<Realty> Realties { get; set; } = null!;

public DbSet<Apartment> Apartments { get; set; } = null!;

public DbSet<PrivateHouse> PrivateHouses { get; set; } = null!;

public DbSet<Area> Areas { get; set; } = null!;

// Предложения и сделки

public DbSet<Listing> Listings { get; set; } = null!;

public DbSet<Deal> Deals { get; set; } = null!;

public DbSet<DealParticipant> DealParticipants { get; set; } = null!;

public DbSet<PotentialMatch> PotentialMatches { get; set; } = null!;

/// <summary>

/// Конфигурация модели данных

/// </summary>

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

// Table Per Type (TPT) для наследования Realty

modelBuilder.Entity<Realty>().ToTable("Realties");

modelBuilder.Entity<Apartment>().ToTable("Apartments");

modelBuilder.Entity<PrivateHouse>().ToTable("PrivateHouses");

modelBuilder.Entity<Area>().ToTable("Areas");

// Конфигурация Address

modelBuilder.Entity<Address>(entity =>

{

entity.HasKey(a => a.Id);

entity.HasIndex(a => a.City);

entity.HasIndex(a => a.District);

entity.HasIndex(a => a.PostalCode);

// Уникальный индекс для предотвращения дубликатов

entity.HasIndex(a => new { a.City, a.District, a.Street, a.HouseNumber, a.BuildingNumber })

.IsUnique();

});

// Конфигурация Client

modelBuilder.Entity<Client>(entity =>

{

entity.HasKey(c => c.Id);

entity.HasIndex(c => c.Phone).IsUnique();

entity.HasOne(c => c.Photo)

.WithOne()

.HasForeignKey<Client>(c => c.PhotoId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasIndex(c => c.PhotoId);

});

// Конфигурация Employee

modelBuilder.Entity<Client>(entity =>

{

entity.HasKey(c => c.Id);

entity.HasIndex(c => c.Phone).IsUnique();

entity.HasOne(c => c.Photo)

.WithOne()

.HasForeignKey<Client>(c => c.PhotoId)

.OnDelete(DeleteBehavior.SetNull);

entity.HasIndex(c => c.PhotoId);

});

// Конфигурация Realty

modelBuilder.Entity<Realty>(entity =>

{

entity.HasKey(r => r.Id);

// Связь с Address

entity.HasOne(r => r.Address)

.WithMany()

.HasForeignKey(r => r.AddressId)

.OnDelete(DeleteBehavior.Restrict);

// Самореференциальная связь (Parent-Child)

entity.HasOne(r => r.ParentRealty)

.WithMany(r => r.ChildRealties)

.HasForeignKey(r => r.ParentRealtyId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasMany(r => r.Photos)

.WithOne()

.HasForeignKey(p => p.EntityId)

.OnDelete(DeleteBehavior.Cascade);

// Индексы

entity.HasIndex(r => r.IsActive);

entity.HasIndex(r => r.AddressId);

entity.HasIndex(r => r.ParentRealtyId);

entity.HasIndex(r => r.RealtyType);

});

// Конфигурация Apartment

modelBuilder.Entity<Apartment>(entity =>

{

entity.HasOne(a => a.RenovationType)

.WithMany()

.HasForeignKey(a => a.RenovationTypeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(a => a.BathroomType)

.WithMany()

.HasForeignKey(a => a.BathroomTypeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(a => a.RenovationTypeId);

entity.HasIndex(a => a.BathroomTypeId);

entity.HasIndex(a => a.Floor);

entity.HasIndex(a => a.RoomsCount);

entity.HasIndex(a => a.TotalArea);

});

// Конфигурация PrivateHouse

modelBuilder.Entity<PrivateHouse>(entity =>

{

entity.HasOne(ph => ph.HeatingType)

.WithMany()

.HasForeignKey(ph => ph.HeatingTypeId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasOne(ph => ph.ConstructionMaterial)

.WithMany()

.HasForeignKey(ph => ph.ConstructionMaterialId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(ph => ph.HeatingTypeId);

entity.HasIndex(ph => ph.ConstructionMaterialId);

entity.HasIndex(ph => ph.YearBuilt);

entity.HasIndex(ph => ph.FloorsCount);

entity.HasIndex(ph => ph.RoomsCount);

});

// Конфигурация Area

modelBuilder.Entity<Area>(entity =>

{

entity.HasOne(a => a.LandCategory)

.WithMany()

.HasForeignKey(a => a.LandCategoryId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(a => a.LandCategoryId);

entity.HasIndex(a => a.Square);

});

// Конфигурация ClientRequest (УБРАНА связь с Realty)

modelBuilder.Entity<ClientRequest>(entity =>

{

entity.HasKey(cr => cr.Id);

// Связь с Client

entity.HasOne(cr => cr.Client)

.WithMany()

.HasForeignKey(cr => cr.ClientId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Employee

entity.HasOne(cr => cr.Employee)

.WithMany()

.HasForeignKey(cr => cr.EmployeeId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Listing

entity.HasOne(cr => cr.Listing)

.WithMany(l => l.ClientRequests)

.HasForeignKey(cr => cr.ListingId)

.OnDelete(DeleteBehavior.Restrict);

// УБРАНА связь с Realty

// entity.HasOne(cr => cr.Realty)...

// Самореференциальная связь (MatchedRequest)

entity.HasOne(cr => cr.MatchedRequest)

.WithMany()

.HasForeignKey(cr => cr.MatchedRequestId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Deal через DealParticipant (обратная связь)

entity.HasMany<DealParticipant>()

.WithOne(dp => dp.ClientRequest)

.HasForeignKey(dp => dp.ClientRequestId)

.OnDelete(DeleteBehavior.Restrict);

// Индексы

entity.HasIndex(cr => cr.Type);

entity.HasIndex(cr => cr.Status);

entity.HasIndex(cr => cr.CreatedDate);

entity.HasIndex(cr => cr.ClientId);

entity.HasIndex(cr => cr.EmployeeId);

entity.HasIndex(cr => cr.ListingId);

entity.HasIndex(cr => cr.MatchedRequestId);

entity.HasIndex(cr => new { cr.Status, cr.Type });

});

// Конфигурация Listing

modelBuilder.Entity<Listing>(entity =>

{

entity.HasKey(l => l.Id);

// Связь с Realty

entity.HasOne(l => l.Realty)

.WithMany(r => r.Listings)

.HasForeignKey(l => l.RealtyId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Client (Owner)

entity.HasOne(l => l.Owner)

.WithMany()

.HasForeignKey(l => l.OwnerId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Employee (ResponsibleEmployee)

entity.HasOne(l => l.ResponsibleEmployee)

.WithMany()

.HasForeignKey(l => l.ResponsibleEmployeeId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с DictionaryValue (Currency)

entity.HasOne(l => l.Currency)

.WithMany()

.HasForeignKey(l => l.CurrencyId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с DictionaryValue (ListingType)

entity.HasOne(l => l.ListingType)

.WithMany()

.HasForeignKey(l => l.ListingTypeId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с DictionaryValue (Status)

entity.HasOne(l => l.Status)

.WithMany()

.HasForeignKey(l => l.StatusId)

.OnDelete(DeleteBehavior.Restrict);

// Индексы

entity.HasIndex(l => l.RealtyId);

entity.HasIndex(l => l.OwnerId);

entity.HasIndex(l => l.ResponsibleEmployeeId);

entity.HasIndex(l => l.CurrencyId);

entity.HasIndex(l => l.ListingTypeId);

entity.HasIndex(l => l.StatusId);

entity.HasIndex(l => l.Price);

entity.HasIndex(l => l.CreatedDate);

entity.HasIndex(l => new { l.StatusId, l.CreatedDate });

});

// Конфигурация Deal (ОБНОВЛЕНА - убрана связь с ClientRequest)

modelBuilder.Entity<Deal>(entity =>

{

entity.HasKey(d => d.Id);

// Связь с Listing

entity.HasOne(d => d.Listing)

.WithMany()

.HasForeignKey(d => d.ListingId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Client (Buyer) - оставляем для быстрого доступа к основному покупателю

entity.HasOne(d => d.Buyer)

.WithMany()

.HasForeignKey(d => d.BuyerId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с Employee

entity.HasOne(d => d.Employee)

.WithMany()

.HasForeignKey(d => d.EmployeeId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с DictionaryValue (DealType)

entity.HasOne(d => d.DealType)

.WithMany()

.HasForeignKey(d => d.DealTypeId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с DictionaryValue (Status)

entity.HasOne(d => d.Status)

.WithMany()

.HasForeignKey(d => d.StatusId)

.OnDelete(DeleteBehavior.Restrict);

// Связь с DealParticipant

entity.HasMany(d => d.Participants)

.WithOne(dp => dp.Deal)

.HasForeignKey(dp => dp.DealId)

.OnDelete(DeleteBehavior.Cascade);

// Индексы

entity.HasIndex(d => d.ListingId);

entity.HasIndex(d => d.BuyerId);

entity.HasIndex(d => d.EmployeeId);

entity.HasIndex(d => d.DealTypeId);

entity.HasIndex(d => d.StatusId);

entity.HasIndex(d => d.DealDate);

entity.HasIndex(d => d.FinalPrice);

entity.HasIndex(d => new { d.StatusId, d.DealDate });

});

// Конфигурация DealParticipant (НОВАЯ сущность)

modelBuilder.Entity<DealParticipant>(entity =>

{

entity.HasKey(dp => dp.Id);

// Связь с Deal

entity.HasOne(dp => dp.Deal)

.WithMany(d => d.Participants)

.HasForeignKey(dp => dp.DealId)

.OnDelete(DeleteBehavior.Cascade);

// Связь с ClientRequest

entity.HasOne(dp => dp.ClientRequest)

.WithMany()

.HasForeignKey(dp => dp.ClientRequestId)

.OnDelete(DeleteBehavior.Restrict);

// Уникальный индекс - одна заявка не может участвовать дважды в одной сделке

entity.HasIndex(dp => new { dp.DealId, dp.ClientRequestId })

.IsUnique();

// Индекс для поиска по роли

entity.HasIndex(dp => dp.Role);

// Индекс для поиска всех сделок заявки

entity.HasIndex(dp => dp.ClientRequestId);

});

// Конфигурация DictionaryValue

modelBuilder.Entity<DictionaryValue>(entity =>

{

entity.HasKey(dv => dv.Id);

entity.HasOne(dv => dv.Dictionary)

.WithMany()

.HasForeignKey(dv => dv.DictionaryId)

.OnDelete(DeleteBehavior.Restrict);

entity.HasIndex(dv => dv.DictionaryId);

entity.HasIndex(dv => dv.Value);

entity.HasIndex(dv => new { dv.DictionaryId, dv.Value }).IsUnique();

});

// Конфигурация Dictionary

modelBuilder.Entity<Dictionary>(entity =>

{

entity.HasKey(d => d.Id);

entity.HasIndex(d => d.Type).IsUnique();

});

// Конфигурация Photo (базовый класс)

modelBuilder.Entity<Photo>(entity =>

{

entity.HasKey(p => p.Id);

// Общие индексы для всех фото

entity.HasIndex(p => p.IsMain);

entity.HasIndex(p => p.SortOrder);

entity.HasIndex(p => p.CreatedDate);

entity.HasIndex(p => p.EntityType);

entity.HasIndex(p => p.EntityId);

// Ограничение на размер файла (20MB максимум)

entity.Property(p => p.FileData)

.HasMaxLength(20 \* 1024 \* 1024);

});

modelBuilder.Entity<PotentialMatch>(entity =>

{

entity.HasNoKey();

entity.ToView("PotentialMatches");

entity.Ignore(e => e.MatchScore);

entity.Ignore(e => e.IsGoodMatch);

entity.Ignore(e => e.IsPerfectMatch);

entity.Ignore(e => e.MatchDescription);

entity.Ignore(e => e.PriceDifference);

entity.Ignore(e => e.IsWithinBudget);

});

}

/// <summary>

/// Переопределение SaveChangesAsync для автоматической установки дат

/// </summary>

public override async Task<int> SaveChangesAsync(CancellationToken cancellationToken = default)

{

var entries = ChangeTracker.Entries()

.Where(e => e.Entity is BaseIdEntity &&

(e.State == EntityState.Added || e.State == EntityState.Modified));

foreach (var entityEntry in entries)

{

if (entityEntry.State == EntityState.Added)

{

// Для новых фото устанавливаем дату создания

if (entityEntry.Entity is Photo photo && photo.CreatedDate == default)

{

photo.CreatedDate = DateTime.UtcNow;

}

// Для новых сделок устанавливаем дату сделки

if (entityEntry.Entity is Deal deal && deal.DealDate == default)

{

deal.DealDate = DateTime.UtcNow;

}

// Для новых запросов устанавливаем дату создания

if (entityEntry.Entity is ClientRequest request && request.CreatedDate == default)

{

request.CreatedDate = DateTime.UtcNow;

}

// Для новых объявлений устанавливаем дату создания

if (entityEntry.Entity is Listing listing && listing.CreatedDate == default)

{

listing.CreatedDate = DateTime.UtcNow;

}

}

}

return await base.SaveChangesAsync(cancellationToken);

}

}

1. Листинг кода конвертора «BooleanToColorConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

using Avalonia.Media;

namespace RealtorTool.Desktop.Converters;

public class BooleanToColorConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is bool boolValue)

{

return boolValue ? Brushes.Green : Brushes.Red;

}

return Brushes.Black;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

1. Листинг кода конвертора «ByteArrayToImageConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

using Avalonia.Media.Imaging;

namespace RealtorTool.Desktop.Converters;

public class ByteArrayToImageConverter : IValueConverter

{

public object? Convert(object? value, Type targetType, object? parameter, CultureInfo culture)

{

if (value is byte[] bytes && bytes.Length > 0)

{

try

{

using var stream = new System.IO.MemoryStream(bytes);

return new Bitmap(stream);

}

catch

{

return GetDefaultImage();

}

}

return GetDefaultImage();ф

}

public object? ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)

{

throw new NotSupportedException();

}

private static Bitmap? GetDefaultImage()

{

try

{

// Можно вернуть дефолтное изображение из ресурсов

// или просто null для отображения плейсхолдера

return null;

}

catch

{

return null;

}

}

}

1. Листинг кода конвертора «CollectionCountToBoolConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

namespace RealtorTool.Desktop.Converters;

public class CollectionCountToBoolConverter : IValueConverter

{

public object Convert(object? value, Type targetType, object? parameter, CultureInfo? culture)

{

if (value is int count)

{

return count > 0;

}

return false;

}

public object ConvertBack(object? value, Type targetType, object? parameter, CultureInfo? culture)

{

throw new NotSupportedException();

}

}

1. Листинг кода конвертора «DateTimeFormatConverter»

using System;

using System.Globalization;

using Avalonia.Data;

using Avalonia.Data.Converters;

namespace RealtorTool.Desktop.Converters;

public class DateTimeFormatConverter : IValueConverter

{

public object? Convert(object? value, Type targetType, object? parameter, CultureInfo? culture)

{

if (value is DateTime dateTime)

{

return dateTime.ToString("dd-MM-yyyy HH-mm");

}

return null;

}

public object? ConvertBack(object? value, Type targetType, object? parameter, CultureInfo? culture)

{

if (value is string dateString && DateTime.TryParseExact(dateString, "dd-MM-yyyy HH-mm",

CultureInfo.InvariantCulture, DateTimeStyles.None, out DateTime result))

{

return result;

}

return BindingOperations.DoNothing;

}

}

1. Листинг кода конвертора «MatchesCountToColorConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

using Avalonia.Media;

namespace RealtorTool.Desktop.Converters;

public class MatchesCountToColorConverter : IValueConverter

{

public static MatchesCountToColorConverter Instance { get; } = new();

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is int count)

{

return count > 0 ? Brushes.Green : Brushes.Red;

}

return Brushes.Black;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

1. Листинг кода конвертора «MatchScoreToBackgroundConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

using Avalonia.Media;

namespace RealtorTool.Desktop.Converters;

public class MatchScoreToBackgroundConverter : IValueConverter

{

public static MatchScoreToBackgroundConverter Instance { get; } = new();

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is int score)

{

return score >= 90 ? new SolidColorBrush(Color.FromArgb(20, 0, 255, 0)) : // Светло-зеленый

score >= 70 ? new SolidColorBrush(Color.FromArgb(20, 255, 165, 0)) : // Светло-оранжевый

score >= 50 ? new SolidColorBrush(Color.FromArgb(20, 0, 0, 255)) : // Светло-синий

new SolidColorBrush(Color.FromArgb(10, 255, 0, 0)); // Светло-красный

}

return Brushes.Transparent;

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

1. Листинг кода конвертора «MatchScoreToColorConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

using Avalonia.Media;

namespace RealtorTool.Desktop.Converters;

public class MatchScoreToColorConverter : IValueConverter

{

public static MatchScoreToColorConverter Instance { get; } = new();

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is decimal score)

{

return score switch

{

>= 90 => new SolidColorBrush(Colors.Green), // Идеальное совпадение

>= 70 => new SolidColorBrush(Colors.Orange), // Хорошее совпадение

>= 50 => new SolidColorBrush(Colors.Yellow), // Возможное совпадение

\_ => new SolidColorBrush(Colors.LightGray) // Слабое совпадение

};

}

return new SolidColorBrush(Colors.LightGray);

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

1. Листинг кода конвертора «ObjectIsNotNullConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

namespace RealtorTool.Desktop.Converters;

public class ObjectIsNotNullConverter : IValueConverter

{

public static ObjectIsNotNullConverter Instance { get; } = new();

public object Convert(object? value, Type targetType, object? parameter, CultureInfo culture)

{

return value != null;

}

public object ConvertBack(object? value, Type targetType, object? parameter, CultureInfo culture)

{

// Обычно в односторонних конвертерах этот метод не используется

throw new NotSupportedException();

}

}

1. Листинг кода конвертора «StatusToColorConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

namespace RealtorTool.Desktop.Converters;

public class StatusToColorConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is string status)

{

return status switch

{

"Активно" => "#4CAF50",

"Продано" => "#F44336",

"Снято" => "#FF9800",

"Черновик" => "#9E9E9E",

"На проверке" => "#2196F3",

\_ => "#757575"

};

}

return "#757575";

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

1. Листинг кода конвертора «TrueToFalseConverter»

using System;

using System.Globalization;

using Avalonia.Data.Converters;

namespace RealtorTool.Desktop.Converters;

public class TrueToFalseConverter : IValueConverter

{

public object Convert(object value, Type targetType, object parameter, CultureInfo culture)

{

if (value is bool boolValue)

{

return boolValue ? "Поиск..." : "Обновить поиск";

}

return "Обновить поиск";

}

public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture)

{

throw new NotImplementedException();

}

}

1. Листинг кода сервиса «AccountingService»

using System;

using System.Security.Cryptography;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RealtorTool.Core.DbEntities;

using RealtorTool.Data.Context;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.Services.Implementations;

public class AccountingService : IAccountingService

{

private readonly DataContext \_context;

public AccountingService(DataContext context)

{

\_context = context;

}

/// <inheritdoc />

public async Task<Employee?> LoginAsync(string? login, string? password)

{

if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))

return null;

var employee = await \_context.Employees

.FirstOrDefaultAsync(a => a.Login == login);

if (employee == null)

return null;

if (!VerifyPassword(password, employee.PasswordHash, employee.Salt))

return null;

return employee;

}

public (string Hash, string Salt) HashPassword(string password)

{

byte[] saltBytes = new byte[16];

using (var rng = RandomNumberGenerator.Create())

{

rng.GetBytes(saltBytes);

}

string salt = Convert.ToBase64String(saltBytes);

string hash = ComputeHash(password, salt);

return (hash, salt);

}

private static bool VerifyPassword(string password, string storedHash, string storedSalt)

{

string computedHash = ComputeHash(password, storedSalt);

return storedHash == computedHash;

}

private static string ComputeHash(string password, string salt)

{

using var pbkdf2 = new Rfc2898DeriveBytes(

password,

Convert.FromBase64String(salt),

10000,

HashAlgorithmName.SHA256);

byte[] hashBytes = pbkdf2.GetBytes(32);

return Convert.ToBase64String(hashBytes);

}

}

1. Листинг кода сервиса «MatchingService»

using System;

using System.Collections.Generic;

using System.Diagnostics;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.Views;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.Services.Interfaces;

namespace RealtorTool.Desktop.Services.Implementations;

public class MatchingService : IMatchingService

{

private readonly DataContext \_context;

public MatchingService(DataContext context)

{

\_context = context;

}

public async Task<List<PotentialMatch>> FindPotentialMatchesAsync(string buyRequestId)

{

var matches = await \_context.PotentialMatches

.Where(x => x.BuyRequestId == buyRequestId)

.AsNoTracking()

.ToListAsync();

return matches.OrderByDescending(m => m.MatchScore).ToList();

}

public async Task<List<PotentialMatch>> FindPotentialMatchesForSellRequestAsync(string sellRequestId)

{

var matches = await \_context.PotentialMatches

.Where(x => x.SellRequestId == sellRequestId)

.AsNoTracking()

.ToListAsync();

return matches.OrderByDescending(m => m.MatchScore).ToList();

}

public async Task<List<PotentialMatch>> FindBestMatchesAsync(string buyRequestId, int limit = 10)

{

var allMatches = await FindPotentialMatchesAsync(buyRequestId);

return allMatches

.Where(m => m.IsGoodMatch)

.Take(limit)

.ToList();

}

public async Task<PotentialMatch?> FindBestMatchAsync(string buyRequestId)

{

var matches = await FindPotentialMatchesAsync(buyRequestId);

return matches.FirstOrDefault(m => m.IsPerfectMatch) ?? matches.FirstOrDefault();

}

// Метод для массового матчинга всех заявок

public async Task<Dictionary<string, List<PotentialMatch>>> FindAllPotentialMatchesAsync()

{

var allMatches = await \_context.PotentialMatches

.FromSqlRaw("SELECT \* FROM potential\_matches")

.AsNoTracking()

.ToListAsync();

return allMatches

.GroupBy(m => m.BuyRequestId)

.ToDictionary(g => g.Key, g => g.OrderByDescending(m => m.MatchScore).ToList());

}

public async Task<bool> CreateMatchAsync(string buyRequestId, string sellRequestId)

{

using var transaction = await \_context.Database.BeginTransactionAsync();

try

{

// Находим заявки с включением связанных данных

var buyRequest = await \_context.ClientRequests

.Include(cr => cr.Client)

.Include(cr => cr.Employee)

.FirstOrDefaultAsync(cr => cr.Id == buyRequestId);

var sellRequest = await \_context.ClientRequests

.Include(cr => cr.Listing)

.ThenInclude(l => l.Realty)

.Include(cr => cr.Client)

.FirstOrDefaultAsync(cr => cr.Id == sellRequestId);

if (buyRequest == null || sellRequest == null)

{

throw new ArgumentException("Одна из заявок не найдена");

}

if (sellRequest.Listing == null)

{

throw new InvalidOperationException("У заявки на продажу отсутствует объявление");

}

// Обновляем статусы заявок

buyRequest.Status = ApplicationStatus.Matched;

sellRequest.Status = ApplicationStatus.Matched;

buyRequest.MatchedRequestId = sellRequestId;

sellRequest.MatchedRequestId = buyRequestId;

// Определяем тип сделки

var dealTypeId = GetDealTypeId(buyRequest.Type);

// Создаем сделку

var deal = new Deal

{

ListingId = sellRequest.ListingId,

BuyerId = buyRequest.ClientId,

EmployeeId = buyRequest.EmployeeId,

FinalPrice = sellRequest.Listing.Price,

Commission = CalculateCommission(sellRequest.Listing.Price, buyRequest.Type),

DealDate = DateTime.UtcNow,

DealTypeId = dealTypeId,

StatusId = "deal\_in\_progress" // предполагаем, что такой ID есть в словаре

};

await \_context.Deals.AddAsync(deal);

await \_context.SaveChangesAsync(); // Сохраняем, чтобы получить ID сделки

// Создаем участников сделки

var participants = new List<DealParticipant>

{

new DealParticipant

{

DealId = deal.Id,

ClientRequestId = buyRequest.Id,

Role = ParticipantRole.Buyer

},

new DealParticipant

{

DealId = deal.Id,

ClientRequestId = sellRequest.Id,

Role = ParticipantRole.Seller

}

};

await \_context.DealParticipants.AddRangeAsync(participants);

await \_context.SaveChangesAsync();

// Фиксируем транзакцию

await transaction.CommitAsync();

return true;

}

catch (Exception ex)

{

await transaction.RollbackAsync();

// Логируем ошибку

Console.WriteLine($"Ошибка при создании сделки: {ex.Message}");

Debug.WriteLine($"Ошибка при создании сделки: {ex}");

return false;

}

}

// Вспомогательные методы

private string GetDealTypeId(ApplicationType requestType)

{

return requestType switch

{

ApplicationType.Purchase or ApplicationType.Sale => "deal\_sale",

ApplicationType.Rent or ApplicationType.RentOut => "deal\_rent",

\_ => "deal\_sale"

};

}

private decimal CalculateCommission(decimal price, ApplicationType requestType)

{

// Разная комиссия для продажи и аренды

var commissionRate = requestType switch

{

ApplicationType.Purchase or ApplicationType.Sale => 0.03m, // 3%

ApplicationType.Rent or ApplicationType.RentOut => 0.05m, // 5%

\_ => 0.03m

};

return price \* commissionRate;

}

}

1. Листинг кода сервиса «NavigationService»

using System;

using System.Collections.Generic;

using System.Threading.Tasks;

using Microsoft.Extensions.DependencyInjection;

using RealtorTool.Desktop.ViewModels;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.Services.Implementations;

public class NavigationService : INavigationService

{

private readonly IServiceProvider \_serviceProvider;

private readonly Stack<ViewModelBase> \_navigationStack = new();

// Добавьте событие для уведомления о смене страницы

public event Action<ViewModelBase>? CurrentPageChanged;

public NavigationService(IServiceProvider serviceProvider)

{

\_serviceProvider = serviceProvider;

}

public async Task NavigateToAsync<TViewModel>() where TViewModel : ViewModelBase

{

var viewModel = \_serviceProvider.GetRequiredService<TViewModel>();

await Navigate(viewModel);

}

public async Task NavigateToAsync<TViewModel>(object parameter) where TViewModel : ViewModelBase

{

var viewModel = \_serviceProvider.GetRequiredService<TViewModel>();

if (viewModel is IParameterReceiver parameterReceiver)

{

parameterReceiver.ReceiveParameter(parameter);

}

await Navigate(viewModel);

}

private async Task Navigate(ViewModelBase viewModel)

{

\_navigationStack.Push(viewModel);

// УВЕДОМИТЕ о смене страницы

CurrentPageChanged?.Invoke(viewModel);

await Task.CompletedTask;

}

public async Task GoBackAsync()

{

if (\_navigationStack.Count > 1)

{

\_navigationStack.Pop();

var previousViewModel = \_navigationStack.Peek();

CurrentPageChanged?.Invoke(previousViewModel);

}

await Task.CompletedTask;

}

public void GoBack()

{

if (\_navigationStack.Count > 1)

{

\_navigationStack.Pop();

var previousViewModel = \_navigationStack.Peek();

CurrentPageChanged?.Invoke(previousViewModel);

}

}

public event Action<PageViewModelBase> OnNavigationRequested;

public void NavigateTo(PageViewModelBase page)

{

OnNavigationRequested?.Invoke(page);

}

}

1. Листинг кода сервиса «PhotoService»

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Threading.Tasks;

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.ApplicationLifetimes;

using Avalonia.Media.Imaging;

using Microsoft.EntityFrameworkCore;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.DTOs;

using RealtorTool.Desktop.Services.Interfaces;

namespace RealtorTool.Desktop.Services.Implementations;

public class PhotoService : IPhotoService

{

private readonly DataContext \_context;

public PhotoService(

DataContext context)

{

\_context = context;

}

public async Task<List<UploadedPhoto>> SelectImagesAsync(Window ownerWindow = null)

{

var uploadedPhotos = new List<UploadedPhoto>();

try

{

var dialog = new OpenFileDialog

{

Title = "Выберите фотографии",

AllowMultiple = true,

Filters = new List<FileDialogFilter>

{

new()

{

Name = "Изображения",

Extensions = new List<string> { "jpg", "jpeg", "png", "bmp", "gif" }

}

}

};

// Если окно не передано, пытаемся найти его

ownerWindow ??= GetDialogOwnerWindow();

if (ownerWindow == null)

{

Console.WriteLine("Не удалось определить окно для диалога");

return uploadedPhotos;

}

var result = await dialog.ShowAsync(ownerWindow);

if (result != null && result.Any())

{

// Обработка выбранных файлов

foreach (var filePath in result)

{

if (IsImageFile(filePath))

{

var uploadedPhoto = await LoadImageAsync(filePath);

if (uploadedPhoto != null)

{

uploadedPhotos.Add(uploadedPhoto);

}

}

}

}

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при выборе файлов: {ex.Message}");

}

return uploadedPhotos;

}

private Window? GetDialogOwnerWindow()

{

try

{

if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktop)

{

// Возвращаем активное окно или главное окно

return desktop.Windows.FirstOrDefault(w => w.IsActive) ?? desktop.MainWindow;

}

return null;

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при получении окна: {ex.Message}");

return null;

}

}

public async Task<UploadedPhoto?> LoadImageAsync(string filePath)

{

try

{

var fileInfo = new FileInfo(filePath);

// Проверяем размер файла (максимум 10MB)

if (fileInfo.Length > 10 \* 1024 \* 1024)

{

Console.WriteLine($"Файл слишком большой: {filePath}");

return null;

}

// Читаем файл

var fileData = await File.ReadAllBytesAsync(filePath);

// Создаем превью

using var stream = new MemoryStream(fileData);

var bitmap = new Bitmap(stream);

// Создаем объект загруженного фото

return new UploadedPhoto

{

FileName = Path.GetFileName(filePath),

FilePath = filePath,

FileSize = fileInfo.Length,

FileSizeFormatted = FormatFileSize(fileInfo.Length),

FileData = fileData,

Preview = bitmap,

ContentType = GetContentType(filePath)

};

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка загрузки изображения {filePath}: {ex.Message}");

return null;

}

}

public void DisposeImagePreview(UploadedPhoto photo)

{

photo.Preview?.Dispose();

}

public async Task<string[]?> SavePhotosToDatabaseAsync(List<UploadedPhoto> uploadedPhotos, string entityId, EntityTypeForPhoto entityType)

{

if (uploadedPhotos == null || !uploadedPhotos.Any())

return null;

bool entityExists = await CheckEntityExistsAsync(entityId, entityType);

if (!entityExists)

throw new ArgumentException($"Entity with ID {entityId} and type {entityType} does not exist");

var photos = uploadedPhotos.Select((uploaded, index) => new Photo

{

Id = Guid.NewGuid().ToString(),

EntityType = entityType,

EntityId = entityId,

FileName = Path.GetFileName(uploaded.FileName),

ContentType = uploaded.ContentType,

FileData = uploaded.FileData ?? throw new ArgumentException("File data cannot be null"),

SortOrder = index,

IsMain = index == 0,

CreatedDate = DateTime.UtcNow

}).ToList();

await RemoveExistingPhotosAsync(entityId, entityType);

await \_context.Photos.AddRangeAsync(photos);

await \_context.SaveChangesAsync();

return photos.Select(x => x.Id).ToArray();

}

public async Task<List<UploadedPhoto>> LoadPhotosFromDatabaseAsync(string entityId, EntityTypeForPhoto entityType)

{

var photos = await \_context.Photos

.Where(p => p.EntityId == entityId && p.EntityType == entityType)

.OrderBy(p => p.SortOrder)

.ToListAsync();

var uploadedPhotos = new List<UploadedPhoto>();

foreach (var photo in photos)

{

try

{

using var stream = new MemoryStream(photo.FileData);

var bitmap = new Bitmap(stream);

uploadedPhotos.Add(new UploadedPhoto

{

Id = photo.Id,

FileName = photo.FileName,

FileSize = photo.FileData.Length,

FileSizeFormatted = FormatFileSize(photo.FileData.Length),

FileData = photo.FileData,

Preview = bitmap,

ContentType = photo.ContentType,

IsMain = photo.IsMain

});

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка загрузки фото {photo.FileName}: {ex.Message}");

}

}

return uploadedPhotos;

}

public async Task DeletePhotoAsync(string photoId)

{

var photo = await \_context.Photos.FindAsync(photoId);

if (photo != null)

{

\_context.Photos.Remove(photo);

await \_context.SaveChangesAsync();

}

}

private async Task<bool> CheckEntityExistsAsync(string entityId, EntityTypeForPhoto entityType)

{

return entityType switch

{

EntityTypeForPhoto.Realty => await \_context.Realties.AnyAsync(r => r.Id == entityId),

EntityTypeForPhoto.Client => await \_context.Clients.AnyAsync(c => c.Id == entityId),

EntityTypeForPhoto.Employee => await \_context.Employees.AnyAsync(e => e.Id == entityId),

\_ => false

};

}

private async Task RemoveExistingPhotosAsync(string entityId, EntityTypeForPhoto entityType)

{

var existingPhotos = await \_context.Photos

.Where(p => p.EntityId == entityId && p.EntityType == entityType)

.ToListAsync();

\_context.Photos.RemoveRange(existingPhotos);

await \_context.SaveChangesAsync();

}

public string FormatFileSize(long bytes)

{

string[] suffixes = { "B", "KB", "MB", "GB" };

int counter = 0;

decimal number = bytes;

while (Math.Round(number / 1024) >= 1)

{

number /= 1024;

counter++;

}

return $"{number:n1} {suffixes[counter]}";

}

private bool IsImageFile(string filePath)

{

var extensions = new[] { ".jpg", ".jpeg", ".png", ".bmp", ".gif" };

var extension = Path.GetExtension(filePath).ToLower();

return extensions.Contains(extension);

}

private string GetContentType(string filePath)

{

return Path.GetExtension(filePath).ToLower() switch

{

".jpg" or ".jpeg" => "image/jpeg",

".png" => "image/png",

".bmp" => "image/bmp",

".gif" => "image/gif",

\_ => "application/octet-stream"

};

}

}

1. Листинг кода сервиса «WindowService»

using System;

using System.Linq;

using Avalonia;

using Avalonia.Controls;

using Avalonia.Controls.ApplicationLifetimes;

using Microsoft.Extensions.DependencyInjection;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.Views.Windows;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.Services.Implementations;

/// <inheritdoc />

public class WindowService : IWindowService

{

private readonly IServiceProvider \_serviceProvider;

private Window? \_currentWindow;

/// <summary>

/// Конструктор.

/// </summary>

public WindowService(IServiceProvider serviceProvider)

{

\_serviceProvider = serviceProvider;

}

/// <inheritdoc />

public void ShowWindow<TWindow>() where TWindow : Window

{

var window = \_serviceProvider.GetRequiredService<TWindow>();

window.Show();

\_currentWindow = window;

}

/// <inheritdoc />

public void ShowDialog<TWindow>(Window owner) where TWindow : Window

{

var window = \_serviceProvider.GetRequiredService<TWindow>();

window.ShowDialog(owner);

}

/// <inheritdoc />

public void CloseWindow(Window window)

{

window.Close();

}

/// <inheritdoc />

public void CloseCurrentWindow()

{

\_currentWindow?.Close();

\_currentWindow = null;

}

/// <inheritdoc />

public void SetCurrentWindow(Window window)

{

\_currentWindow = window;

}

public Window? GetMainWindow()

{

if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktopLifetime)

{

return desktopLifetime.MainWindow ?? desktopLifetime.Windows.FirstOrDefault();

}

if (Application.Current?.ApplicationLifetime is ISingleViewApplicationLifetime singleViewLifetime)

{

return singleViewLifetime.MainView as Window;

}

return TopLevel.GetTopLevel(null) as Window;

}

public Window? GetActiveWindow()

{

if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktopLifetime)

{

return desktopLifetime.Windows.FirstOrDefault(w => w.IsActive)

?? desktopLifetime.MainWindow

?? desktopLifetime.Windows.FirstOrDefault();

}

return GetMainWindow();

}

public void Logout()

{

if (Application.Current?.ApplicationLifetime is IClassicDesktopStyleApplicationLifetime desktopLifetime)

{

// Скрываем все окна кроме авторизации

foreach (var window in desktopLifetime.Windows.ToList())

{

if (window is MainWindow mainWindow)

{

// Скрываем вместо закрытия

mainWindow.Hide();

}

else if (window is not AuthorizationWindow)

{

window.Close();

}

}

// Показываем окно авторизации

var authWindow = \_serviceProvider.GetRequiredService<AuthorizationWindow>();

authWindow.Show();

// Устанавливаем как главное окно

desktopLifetime.MainWindow = authWindow;

\_currentWindow = authWindow;

}

}

}

1. Листинг кода сервиса бизнес-логики «DealItemViewModel»

using System.Linq;

using RealtorTool.Core.DbEntities;

namespace RealtorTool.Desktop.ViewModels.Items;

public class DealItemViewModel : ViewModelBase

{

public Deal Deal { get; set; }

public DealItemViewModel(Deal deal)

{

Deal = deal;

}

// Главное изображение из связанного объекта недвижимости

public byte[]? MainImage => Deal.Listing?.Realty?.Photos

.OrderBy(p => p.SortOrder)

.FirstOrDefault()?.FileData;

public bool HasImage => MainImage != null;

// Форматированная дата сделки

public string FormattedDealDate => Deal.DealDate.ToString("dd.MM.yyyy");

// Форматированная финальная цена

public string FormattedFinalPrice => $"{Deal.FinalPrice:N0}";

// Форматированная комиссия

public string FormattedCommission => $"{Deal.Commission:N0}";

// Основной покупатель

public string BuyerName => Deal.Buyer != null

? $"{Deal.Buyer.LastName} {Deal.Buyer.FirstName} {Deal.Buyer.MiddleName}"

: "Не указан";

// Ответственный сотрудник

public string EmployeeName => Deal.Employee != null

? $"{Deal.Employee.LastName} {Deal.Employee.FirstName}"

: "Не назначен";

// Название объекта недвижимости

public string RealtyName => Deal.Listing?.Realty?.Name ?? "Объект не указан";

// Адрес объекта

public string Address => Deal.Listing?.Realty?.Address != null

? $"{Deal.Listing.Realty.Address.City}, {Deal.Listing.Realty.Address.Street}"

: "Адрес не указан";

// Количество участников сделки

public int ParticipantsCount => Deal.Participants?.Count ?? 0;

// Статус сделки

public string Status => Deal.Status?.Value ?? "Не указан";

// Тип сделки

public string DealType => Deal.DealType?.Value ?? "Не указан";

}

1. Листинг кода сервиса бизнес-логики «ListingItemViewModel»

using System;

using System.Linq;

using Avalonia.Media.Imaging;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

namespace RealtorTool.Desktop.ViewModels.Items;

public class ListingItemViewModel : ViewModelBase

{

public Listing Listing { get; }

[Reactive] public Bitmap? MainImage { get; set; }

[Reactive] public bool HasImage { get; set; }

public ListingItemViewModel(Listing listing)

{

Listing = listing;

LoadMainImage();

}

private void LoadMainImage()

{

var mainPhoto = Listing.Realty?.Photos?.FirstOrDefault();

if (mainPhoto?.FileData != null && mainPhoto.FileData.Length > 0)

{

try

{

using var stream = new System.IO.MemoryStream(mainPhoto.FileData);

MainImage = new Bitmap(stream);

HasImage = true;

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка загрузки изображения: {ex.Message}");

HasImage = false;

}

}

else

{

HasImage = false;

}

}

}

1. Листинг кода сервиса бизнес-логики «BuyApplicationPageViewModel»

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using System.Threading.Tasks;

using DynamicData;

using Microsoft.EntityFrameworkCore;

using MsBox.Avalonia;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.Views;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages.ApplicationPages;

public class BuyApplicationPageViewModel : PageViewModelBase

{

private readonly DataContext \_context;

private readonly IMatchingService \_matchingService;

public Employee \_authEmploee { get; private set; }

[Reactive] public Client NewClient { get; set; } = new();

[Reactive] public decimal? MaxPrice { get; set; }

[Reactive] public int? MinRooms { get; set; }

[Reactive] public decimal? MinArea { get; set; }

[Reactive] public decimal? MaxArea { get; set; }

[Reactive] public string DesiredLocation { get; set; }

[Reactive] public string AdditionalRequirements { get; set; }

[Reactive] public ObservableCollection<PotentialMatch> FoundMatches { get; set; } = new();

[Reactive] public bool HasMatches { get; set; }

[Reactive] public bool IsSearching { get; set; }

[Reactive] public PotentialMatch SelectedMatch { get; set; }

[Reactive] public bool IsMatchSelected { get; set; }

[Reactive] public bool IsCreatingRequest { get; set; }

private int \_selectedPropertyTypeIndex;

public int SelectedPropertyTypeIndex

{

get => \_selectedPropertyTypeIndex;

set => this.RaiseAndSetIfChanged(ref \_selectedPropertyTypeIndex, value);

}

public RealtyType CurrentRealtyType => SelectedPropertyTypeIndex switch

{

0 => RealtyType.Apartment,

1 => RealtyType.Area,

2 => RealtyType.PrivateHouse,

\_ => RealtyType.Apartment

};

public ReactiveCommand<Unit, Unit> CreateBuyRequestCommand { get; }

public ReactiveCommand<Unit, Unit> SearchMatchesCommand { get; }

public ReactiveCommand<PotentialMatch, Unit> SelectMatchButton { get; }

public ReactiveCommand<PotentialMatch, Unit> ViewMatchDetailsCommand { get; }

public BuyApplicationPageViewModel(DataContext context, IMatchingService matchingService)

{

\_context = context;

\_matchingService = matchingService;

GetDataFromMessageBus();

CreateBuyRequestCommand = ReactiveCommand.CreateFromTask(CreateBuyRequestAsync);

SearchMatchesCommand = ReactiveCommand.CreateFromTask(SearchMatchesAsync);

SelectMatchButton = ReactiveCommand.CreateFromTask<PotentialMatch>(SelectMatchAsync);

// ДОБАВИТЬ: Инициализация команды просмотра деталей

ViewMatchDetailsCommand = ReactiveCommand.Create<PotentialMatch>(match =>

{

SelectedMatch = match;

IsMatchSelected = true;

});

}

private void GetDataFromMessageBus()

{

MessageBus.Current

.Listen<Employee>("CurrentAuth")

.Subscribe(x =>

{

\_authEmploee = x;

});

if (\_authEmploee != null)

return;

\_authEmploee = \_context.Employees.Find("system\_empl")!;

}

private async Task CreateBuyRequestAsync()

{

if (IsCreatingRequest) return;

IsCreatingRequest = true;

await using var transaction = await \_context.Database.BeginTransactionAsync();

try

{

if (string.IsNullOrWhiteSpace(NewClient.FirstName) || string.IsNullOrWhiteSpace(NewClient.LastName))

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Заполните имя и фамилию клиента")

.ShowAsync();

return;

}

if (\_authEmploee == null)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Не удалось определить сотрудника для обработки заявки")

.ShowAsync();

return;

}

await \_context.Clients.AddAsync(NewClient);

await \_context.SaveChangesAsync();

var buyRequest = new ClientRequest

{

DesiredRealtyType = CurrentRealtyType,

Type = ApplicationType.Purchase,

Status = ApplicationStatus.New,

ClientId = NewClient.Id,

EmployeeId = \_authEmploee.Id,

MaxPrice = MaxPrice,

MinRooms = MinRooms,

MinArea = MinArea,

MaxArea = MaxArea,

DesiredLocation = DesiredLocation,

AdditionalRequirements = AdditionalRequirements,

CreatedDate = DateTime.UtcNow

};

await \_context.ClientRequests.AddAsync(buyRequest);

await \_context.SaveChangesAsync();

await transaction.CommitAsync();

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Заявка на покупку создана")

.ShowAsync();

await SearchMatchesAsync();

}

catch (DbUpdateException dbEx)

{

await transaction.RollbackAsync();

var errorMessage = $"Ошибка базы данных: {dbEx.InnerException?.Message ?? dbEx.Message}";

await MessageBoxManager.GetMessageBoxStandard("Ошибка базы данных", errorMessage).ShowAsync();

}

catch (Exception ex)

{

await transaction.RollbackAsync();

var errorMessage = $"Неожиданная ошибка: {ex.Message}";

await MessageBoxManager.GetMessageBoxStandard("Ошибка", errorMessage).ShowAsync();

}

finally

{

IsCreatingRequest = false;

}

}

private async Task SearchMatchesAsync()

{

if (NewClient?.Id == null)

{

await MessageBoxManager

.GetMessageBoxStandard("Информация", "Сначала создайте заявку клиента")

.ShowAsync();

return;

}

IsSearching = true;

try

{

// Находим созданную заявку

var buyRequest = await \_context.ClientRequests

.FirstOrDefaultAsync(cr => cr.ClientId == NewClient.Id && cr.Type == ApplicationType.Purchase);

if (buyRequest != null)

{

var matches = await \_matchingService.FindPotentialMatchesAsync(buyRequest.Id);

FoundMatches.Clear();

FoundMatches.AddRange(matches);

HasMatches = matches.Any();

// Сбрасываем выбранный матч

SelectedMatch = null;

IsMatchSelected = false;

}

else

{

await MessageBoxManager

.GetMessageBoxStandard("Информация", "Не найдена заявка для поиска совпадений")

.ShowAsync();

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка при поиске совпадений: {ex.Message}")

.ShowAsync();

}

finally

{

IsSearching = false;

}

}

public async Task SelectMatchAsync(PotentialMatch selectedMatch)

{

if (NewClient?.Id == null || selectedMatch == null) return;

var buyRequest = await \_context.ClientRequests

.FirstOrDefaultAsync(cr => cr.ClientId == NewClient.Id && cr.Type == ApplicationType.Purchase);

if (buyRequest != null)

{

var messageBox = MessageBoxManager.GetMessageBoxStandard(

"Подтверждение",

$"Вы уверены, что хотите создать сделку для этого объекта?\n\n" +

$"Цена: {selectedMatch.ListingPrice:N0} руб.\n" +

$"Тип: {GetRealtyTypeDisplayName(selectedMatch.RealtyType)}\n" +

$"Район: {selectedMatch.District}",

ButtonEnum.YesNo,

Icon.Question

);

var result = await messageBox.ShowAsync();

if (result == ButtonResult.Yes)

{

var createResult = await \_matchingService.CreateMatchAsync(buyRequest.Id, selectedMatch.SellRequestId);

if (createResult)

{

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сделка создана успешно!")

.ShowAsync();

ClearForm();

}

else

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Не удалось создать сделку. Попробуйте позже.")

.ShowAsync();

}

}

}

}

private string GetRealtyTypeDisplayName(RealtyType realtyType)

{

return realtyType switch

{

RealtyType.Apartment => "Квартира",

RealtyType.PrivateHouse => "Частный дом",

RealtyType.Area => "Земельный участок",

\_ => "Неизвестно"

};

}

private void ClearForm()

{

NewClient = new Client();

MaxPrice = null;

MinRooms = null;

MinArea = null;

MaxArea = null;

DesiredLocation = string.Empty;

AdditionalRequirements = string.Empty;

FoundMatches.Clear();

HasMatches = false;

SelectedMatch = null;

IsMatchSelected = false;

}

public string MatchesSummary => HasMatches

? $"Найдено {FoundMatches.Count} совпадений"

: "Совпадения не найдены";

public int GoodMatchesCount => FoundMatches.Count(m => m.IsGoodMatch);

public int PerfectMatchesCount => FoundMatches.Count(m => m.IsPerfectMatch);

}

1. Листинг кода сервиса бизнес-логики «SellApplicationPageViewModel»

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using System.Threading.Tasks;

using System.Windows.Input;

using DynamicData;

using Microsoft.EntityFrameworkCore;

using MsBox.Avalonia;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.RealtyModels;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.DTOs;

using RealtorTool.Desktop.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages.ApplicationPages;

public class SellApplicationPageViewModel : ViewModelBase

{

/// Сервисы.

private readonly DataContext \_context;

private readonly IPhotoService \_photoService;

// Поля.

[Reactive] public Address NewAddress { get; set; } = new();

[Reactive] public Listing NewListing { get; set; } = new();

[Reactive] public Client NewClient { get; set; } = new();

[Reactive] public Area NewArea { get; set; } = new();

[Reactive] public PrivateHouse NewPrivateHouse { get; set; } = new();

[Reactive] public Apartment NewApartament { get; set; } = new();

[Reactive] public ObservableCollection<UploadedPhoto> UploadedPhotos { get; set; } = new();

[Reactive] public string PhotosSummary { get; set; } = "Фотографии не добавлены";

// Коллекции.

public ObservableCollection<DictionaryValue> RenovationTypes { get; set; } = new();

public ObservableCollection<DictionaryValue> BathroomTypes { get; set; } = new();

public RealtyType CurrentRealtyType { get; private set; }

private int \_selectedPropertyTypeIndex;

public int SelectedPropertyTypeIndex

{

get => \_selectedPropertyTypeIndex;

set

{

this.RaiseAndSetIfChanged(ref \_selectedPropertyTypeIndex, value);

CurrentRealtyType = SelectedPropertyTypeIndex switch

{

0 => RealtyType.Apartment,

1 => RealtyType.Area,

2 => RealtyType.PrivateHouse,

\_ => CurrentRealtyType

};

}

}

// Кнопки.

public ICommand CreateSellRequest { get; }

public ReactiveCommand<Unit, Unit> SelectImagesCommand { get; }

public ReactiveCommand<UploadedPhoto, Unit> RemovePhotoCommand { get; }

/// <summary>

/// Конструктор.

/// </summary>

public SellApplicationPageViewModel(

DataContext context,

IPhotoService photoService)

{

\_context = context;

\_photoService = photoService;

CreateSellRequest = ReactiveCommand.CreateFromTask(CreateSellRequestAsync);

SelectImagesCommand = ReactiveCommand.CreateFromTask(SelectImagesAsync);

RemovePhotoCommand = ReactiveCommand.Create<UploadedPhoto>(RemovePhoto);

\_ = InitializeAsync();

}

private async Task InitializeAsync()

{

var renovationTypesList = await \_context.DictionaryValues

.Where(x => x.DictionaryId == "renovation\_type")

.ToListAsync();

RenovationTypes.AddRange(renovationTypesList);

var bathroomTypes = await \_context.DictionaryValues

.Where(x => x.DictionaryId == "bathroom\_type")

.ToListAsync();

BathroomTypes.AddRange(bathroomTypes);

}

private async Task CreateSellRequestAsync()

{

try

{

using var transaction = await \_context.Database.BeginTransactionAsync();

var employee = await \_context.Employees.FirstOrDefaultAsync(); // TODO Подключить MB

if (employee == null)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Не найден сотрудник для обработки заявки")

.ShowAsync();

return;

}

\_context.Clients.Attach(NewClient);

\_context.Addresses.Attach(NewAddress);

Realty createdRealty = null;

switch (CurrentRealtyType)

{

case RealtyType.Area:

NewArea.Address = NewAddress;

NewArea.RealtyType = RealtyType.Area;

\_context.Areas.Attach(NewArea);

createdRealty = NewArea;

break;

case RealtyType.Apartment:

NewApartament.RenovationType = RenovationTypes.FirstOrDefault(x => x.IsSelected);

NewApartament.BathroomType = BathroomTypes.FirstOrDefault(x => x.IsSelected);

NewApartament.Address = NewAddress;

NewApartament.RealtyType = RealtyType.Apartment;

\_context.Apartments.Attach(NewApartament);

createdRealty = NewApartament;

break;

case RealtyType.PrivateHouse:

NewPrivateHouse.Address = NewAddress;

NewPrivateHouse.RealtyType = RealtyType.PrivateHouse;

\_context.PrivateHouses.Attach(NewPrivateHouse);

createdRealty = NewPrivateHouse;

break;

}

NewListing.Realty = createdRealty;

NewListing.Owner = NewClient;

NewListing.CurrencyId = "currency\_rub";

NewListing.ListingTypeId = "listing\_sale";

NewListing.StatusId = "listing\_active";

\_context.Listings.Attach(NewListing);

var sellRequest = new ClientRequest

{

Type = ApplicationType.Sale,

Status = ApplicationStatus.New,

ClientId = NewClient.Id,

ListingId = NewListing.Id,

EmployeeId = employee.Id,

CreatedDate = DateTime.UtcNow

};

\_context.ClientRequests.Attach(sellRequest);

await \_context.SaveChangesAsync();

if (createdRealty != null && UploadedPhotos.Any())

{

await \_photoService.SavePhotosToDatabaseAsync(

UploadedPhotos.ToList(),

createdRealty.Id,

EntityTypeForPhoto.Realty);

}

await transaction.CommitAsync();

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Заявка на продажу успешно создана")

.ShowAsync();

ClearForm();

}

catch (Exception e)

{

var exceptionMessage = $"{e.Message}\n{(e.InnerException?.Message ?? "")}";

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", exceptionMessage)

.ShowAsync();

}

}

private async Task AddAddressDataToContextAsync()

{

await \_context.AddAsync(NewAddress);

}

private async Task AddPrivateHouseDataToContextAsync()

{

throw new NotImplementedException();

}

private async Task<Realty> AddApartmentDataToContextAsync()

{

NewApartament.RenovationType = RenovationTypes.FirstOrDefault(x => x.IsSelected);

NewApartament.BathroomType = BathroomTypes.FirstOrDefault(x => x.IsSelected);

NewApartament.Address = NewAddress;

NewApartament.RealtyType = RealtyType.Apartment;

await \_context.AddAsync(NewApartament);

await \_context.SaveChangesAsync();

return NewApartament;

}

private async Task AddAreaDataToContextAsync()

{

throw new NotImplementedException();

}

private async Task AddClientDataToContextAsync()

{

await \_context.AddAsync(NewClient);

}

private async Task SelectImagesAsync()

{

var newPhotos = await \_photoService.SelectImagesAsync();

if (newPhotos.Any())

{

foreach (var photo in newPhotos)

{

UploadedPhotos.Add(photo);

}

UpdatePhotosSummary();

}

}

public void RemovePhoto(UploadedPhoto photo)

{

UploadedPhotos.Remove(photo);

\_photoService.DisposeImagePreview(photo);

UpdatePhotosSummary();

}

private void UpdatePhotosSummary()

{

if (!UploadedPhotos.Any())

{

PhotosSummary = "Фотографии не добавлены";

return;

}

var totalSize = UploadedPhotos.Sum(p => p.FileSize);

PhotosSummary = $"{UploadedPhotos.Count} фото • {\_photoService.FormatFileSize(totalSize)}";

}

private void ClearForm()

{

NewAddress = new Address();

NewListing = new Listing();

NewClient = new Client();

NewArea = new Area();

NewApartament = new Apartment();

NewPrivateHouse = new PrivateHouse();

UploadedPhotos.Clear();

UpdatePhotosSummary();

// Сбрасываем выбранные типы ремонта и санузлов

foreach (var item in RenovationTypes)

{

item.IsSelected = false;

}

foreach (var item in BathroomTypes)

{

item.IsSelected = false;

}

}

}

1. Листинг кода сервиса бизнес-логики «ApartmentDetailPageViewModel»

using System;

using System.Reactive.Linq;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.RealtyModels;

namespace RealtorTool.Desktop.ViewModels.Pages.RealtyDetailPages;

public class ApartmentDetailPageViewModel : PageViewModelBase

{

[Reactive] public Apartment Apartment { get; set; }

public ApartmentDetailPageViewModel()

{

Title = "Квартира";

MessageBus.Current

.Listen<Apartment>()

.Take(1)

.ObserveOn(RxApp.MainThreadScheduler)

.Subscribe(a =>

{

Apartment = a;

});

}

}

1. Листинг кода сервиса бизнес-логики «AreaDetailPageViewModel»

using System;

using System.Reactive.Linq;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.RealtyModels;

namespace RealtorTool.Desktop.ViewModels.Pages.RealtyDetailPages;

public class AreaDetailPageViewModel : PageViewModelBase

{

[Reactive] public Area Area { get; set; }

public AreaDetailPageViewModel()

{

Title = "Район";

MessageBus.Current

.Listen<Area>()

.Take(1)

.ObserveOn(RxApp.MainThreadScheduler)

.Subscribe(a =>

{

Area = a;

});

}

}

1. Листинг кода сервиса бизнес-логики «PrivateHouseDetailPageViewModel»

using System;

using System.Reactive.Linq;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.RealtyModels;

namespace RealtorTool.Desktop.ViewModels.Pages.RealtyDetailPages;

public class PrivateHouseDetailPageViewModel : PageViewModelBase

{

[Reactive] public PrivateHouse House { get; set; }

public PrivateHouseDetailPageViewModel()

{

Title = "Частный дом";

MessageBus.Current

.Listen<PrivateHouse>()

.Take(1)

.ObserveOn(RxApp.MainThreadScheduler)

.Subscribe(h =>

{

House = h;

});

}

}

1. Листинг кода сервиса бизнес-логики «ApplicationsPageViewModel»

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using MsBox.Avalonia;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.ViewModels.Items;

using RealtorTool.Desktop.ViewModels.Windows;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class ApplicationsPageViewModel : PageViewModelBase

{

private readonly IServiceProvider \_serviceProvider;

private readonly DataContext \_dataContext;

private readonly INavigationService \_navigationService;

private ListingItemViewModel? \_selectedListing;

[Reactive] public ObservableCollection<ListingItemViewModel> Listings { get; set; } = new();

public ListingItemViewModel? SelectedListing

{

get => \_selectedListing;

set

{

if (string.IsNullOrEmpty(value?.Listing.Id))

MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Id заявки не найден")

.ShowAsync();

this.RaiseAndSetIfChanged(ref \_selectedListing, value);

OpenListingDetailAsync(value?.Listing.Id!);

}

}

public ApplicationsPageViewModel(

DataContext dataContext,

IServiceProvider serviceProvider,

INavigationService navigationService)

{

\_dataContext = dataContext;

\_serviceProvider = serviceProvider;

\_navigationService = navigationService;

\_ = LoadDataAsync();

}

public async Task LoadDataAsync()

{

var listings = await \_dataContext.Listings

.Include(l => l.Realty)

.ThenInclude(r => r.Photos.OrderBy(p => p.SortOrder).Take(1))

.Include(l => l.Currency)

.Include(l => l.ListingType)

.Include(l => l.Status)

.ToListAsync();

Listings.Clear();

foreach (var listing in listings)

{

Listings.Add(new ListingItemViewModel(listing));

}

}

private void OpenListingDetailAsync(string listingId)

{

var detailVm = \_serviceProvider.GetRequiredService<ListingDetailViewModel>();

if (detailVm is IParameterReceiver parameterReceiver)

parameterReceiver.ReceiveParameter(listingId);

// Отправляем через MessageBus с правильным токеном

MessageBus.Current.SendMessage(detailVm, "NavigateToPage");

// Используем сервис навигации

\_navigationService.NavigateTo(detailVm);

MessageBus.Current.SendMessage(listingId, "ListingDetailsId");

}

}

1. Листинг кода сервиса бизнес-логики «CreatingApplicationPageViewModel»

using System;

using System.Collections.ObjectModel;

using System.Reactive;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.Enums;

using RealtorTool.Desktop.ViewModels.Pages.ApplicationPages;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class CreatingApplicationPageViewModel : PageViewModelBase

{

[Reactive] public bool RentIsVisible { get; set; } = true;

[Reactive] public bool RentingOutIsVisible { get; set; }

[Reactive] public bool PurchaseIsVisible { get; set; }

[Reactive] public bool SaleIsVisible { get; set; }

private readonly Lazy<BuyApplicationPageViewModel> \_buyApplicationPageViewModel;

private readonly Lazy<SellApplicationPageViewModel> \_sellApplicationPageViewModel;

private readonly Lazy<LeaseApplicationPageViewModel> \_leaseApplicationPageViewModel;

private readonly Lazy<RentalApplicationPageViewModel> \_rentalApplicationPageViewModel;

public BuyApplicationPageViewModel BuyApplicationPageViewModel => \_buyApplicationPageViewModel.Value;

public SellApplicationPageViewModel SellApplicationPageViewModel => \_sellApplicationPageViewModel.Value;

public LeaseApplicationPageViewModel LeaseApplicationPageViewModel => \_leaseApplicationPageViewModel.Value;

public RentalApplicationPageViewModel RentalApplicationPageViewModel => \_rentalApplicationPageViewModel.Value;

public ReactiveCommand<ApplicationType, Unit> ChangeOperationType { private set; get; }

public CreatingApplicationPageViewModel(

Func<BuyApplicationPageViewModel> buyApplicationPageViewModelFactory,

Func<SellApplicationPageViewModel> sellApplicationPageViewModelFactory,

Func<LeaseApplicationPageViewModel> leaseApplicationPageViewModelFactory,

Func<RentalApplicationPageViewModel> rentalApplicationPageViewModelFactory)

{

InitialButtons();

\_buyApplicationPageViewModel = new Lazy<BuyApplicationPageViewModel>(buyApplicationPageViewModelFactory);

\_sellApplicationPageViewModel = new Lazy<SellApplicationPageViewModel>(sellApplicationPageViewModelFactory);

\_leaseApplicationPageViewModel = new Lazy<LeaseApplicationPageViewModel>(leaseApplicationPageViewModelFactory);

\_rentalApplicationPageViewModel = new Lazy<RentalApplicationPageViewModel>(rentalApplicationPageViewModelFactory);

}

private void InitialButtons()

{

ChangeOperationType = ReactiveCommand.Create<ApplicationType>(ChangeOperationTypeImpl);

}

private void ChangeOperationTypeImpl(ApplicationType type)

{

RentIsVisible = type == ApplicationType.Rent;

RentingOutIsVisible = type == ApplicationType.RentOut;

PurchaseIsVisible = type == ApplicationType.Purchase;

SaleIsVisible = type == ApplicationType.Sale;

}

}

1. Листинг кода сервиса бизнес-логики «DealDetailPageViewModel»

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.EntityFrameworkCore;

using MsBox.Avalonia;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Data.Context;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class DealDetailPageViewModel : PageViewModelBase, IParameterReceiver

{

private readonly DataContext \_dataContext;

private readonly IServiceProvider \_serviceProvider;

private readonly INavigationService \_navigationService;

[Reactive] public Deal? Deal { get; set; }

[Reactive] public ObservableCollection<DictionaryValue> AvailableStatuses { get; set; } = new();

[Reactive] public DictionaryValue? SelectedStatus { get; set; }

[Reactive] public bool IsLoading { get; set; }

[Reactive] public bool IsEditing { get; set; }

public ICommand SaveCommand { get; }

public ICommand CancelCommand { get; }

public ICommand EditCommand { get; }

public DealDetailPageViewModel(

DataContext dataContext,

IServiceProvider serviceProvider,

INavigationService navigationService)

{

\_dataContext = dataContext;

\_serviceProvider = serviceProvider;

\_navigationService = navigationService;

SaveCommand = ReactiveCommand.CreateFromTask(SaveDealAsync);

CancelCommand = ReactiveCommand.Create(CancelEdit);

EditCommand = ReactiveCommand.Create(StartEdit);

}

public void ReceiveParameter(object parameter)

{

if (parameter is string dealId)

{

\_ = LoadDealAsync(dealId);

}

}

private async Task LoadDealAsync(string dealId)

{

IsLoading = true;

try

{

// Загружаем сделку со всеми связанными данными

Deal = await \_dataContext.Deals

.Include(d => d.Listing)

.ThenInclude(l => l.Realty)

.ThenInclude(r => r.Address)

.Include(d => d.Listing)

.ThenInclude(l => l.Currency)

.Include(d => d.Listing)

.ThenInclude(l => l.Realty)

.ThenInclude(r => r.Photos.OrderBy(p => p.SortOrder))

.Include(d => d.Buyer)

.Include(d => d.Employee)

.Include(d => d.DealType)

.Include(d => d.Status)

.Include(d => d.Participants)

.ThenInclude(p => p.ClientRequest)

.ThenInclude(cr => cr.Client)

.FirstOrDefaultAsync(d => d.Id == dealId && !d.IsDeleted);

if (Deal == null)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Сделка не найдена")

.ShowAsync();

\_navigationService.GoBack();

return;

}

// Загружаем доступные статусы для сделок

await LoadAvailableStatusesAsync();

// Устанавливаем текущий статус

SelectedStatus = AvailableStatuses.FirstOrDefault(s => s.Id == Deal.StatusId);

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка загрузки сделки: {ex.Message}")

.ShowAsync();

}

finally

{

IsLoading = false;

}

}

private async Task LoadAvailableStatusesAsync()

{

try

{

var statuses = await \_dataContext.DictionaryValues

.Include(dv => dv.Dictionary)

.Where(dv => dv.Dictionary.Type == "deal\_status")

.OrderBy(dv => dv.Value)

.ToListAsync();

AvailableStatuses.Clear();

foreach (var status in statuses)

{

AvailableStatuses.Add(status);

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка загрузки статусов: {ex.Message}")

.ShowAsync();

}

}

private void StartEdit()

{

IsEditing = true;

}

private void CancelEdit()

{

IsEditing = false;

// Восстанавливаем исходный статус

SelectedStatus = AvailableStatuses.FirstOrDefault(s => s.Id == Deal?.StatusId);

}

private async Task SaveDealAsync()

{

if (Deal == null || SelectedStatus == null)

return;

try

{

// Обновляем статус сделки

Deal.StatusId = SelectedStatus.Id;

Deal.Status = SelectedStatus;

\_dataContext.Deals.Update(Deal);

await \_dataContext.SaveChangesAsync();

IsEditing = false;

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сделка успешно обновлена", ButtonEnum.Ok, Icon.Success)

.ShowAsync();

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка сохранения: {ex.Message}")

.ShowAsync();

}

}

// Вычисляемые свойства для удобства привязки

public string BuyerName => Deal?.Buyer != null

? $"{Deal.Buyer.LastName} {Deal.Buyer.FirstName} {Deal.Buyer.MiddleName}"

: "Не указан";

public string EmployeeName => Deal?.Employee != null

? $"{Deal.Employee.LastName} {Deal.Employee.FirstName} {Deal.Employee.MiddleName}"

: "Не назначен";

public string RealtyName => Deal?.Listing?.Realty?.Name ?? "Не указан";

public string Address => Deal?.Listing?.Realty?.Address != null

? $"{Deal.Listing.Realty.Address.City}, {Deal.Listing.Realty.Address.Street} {Deal.Listing.Realty.Address.HouseNumber}"

: "Адрес не указан";

public string FormattedFinalPrice => Deal != null ? $"{Deal.FinalPrice:N0} {Deal.Listing?.Currency?.Value}" : "";

public string FormattedCommission => Deal != null ? $"{Deal.Commission:N0} {Deal.Listing?.Currency?.Value}" : "";

public string FormattedDealDate => Deal?.DealDate.ToString("dd.MM.yyyy") ?? "";

public int ParticipantsCount => Deal?.Participants?.Count ?? 0;

public bool HasPhotos => Deal?.Listing?.Realty?.Photos?.Any() == true;

public ObservableCollection<Photo> Photos => new(Deal?.Listing?.Realty?.Photos?.OrderBy(p => p.SortOrder) ?? Enumerable.Empty<Photo>());

}

1. Листинг кода сервиса бизнес-логики «DealsListPageViewModel»

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using MsBox.Avalonia;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.Services.Implementations;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.ViewModels.Items;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class DealsListPageViewModel : PageViewModelBase

{

private readonly IServiceProvider \_serviceProvider;

private readonly DataContext \_dataContext;

private readonly INavigationService \_navigationService;

private DealItemViewModel? \_selectedDeal;

[Reactive] public ObservableCollection<DealItemViewModel> Deals { get; set; } = new();

public DealItemViewModel? SelectedDeal

{

get => \_selectedDeal;

set

{

if (string.IsNullOrEmpty(value?.Deal.Id))

MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Id сделки не найден")

.ShowAsync();

this.RaiseAndSetIfChanged(ref \_selectedDeal, value);

OpenDealDetailAsync(value?.Deal.Id!);

}

}

public DealsListPageViewModel(

IServiceProvider serviceProvider,

INavigationService navigationService,

DataContext dataContext)

{

\_serviceProvider = serviceProvider;

\_navigationService = navigationService;

\_dataContext = dataContext;

\_ = LoadDataAsync();

}

private async Task LoadDataAsync()

{

var deals = await \_dataContext.Deals

.Include(d => d.Listing)

.ThenInclude(l => l.Realty)

.ThenInclude(r => r.Photos.OrderBy(p => p.SortOrder).Take(1))

.Include(d => d.Listing)

.ThenInclude(l => l.Currency)

.Include(d => d.Buyer)

.Include(d => d.Employee)

.Include(d => d.DealType)

.Include(d => d.Status)

.Include(d => d.Participants)

.ThenInclude(p => p.ClientRequest)

.ThenInclude(cr => cr.Client)

.Where(d => !d.IsDeleted)

.ToListAsync();

Deals.Clear();

foreach (var deal in deals)

{

Deals.Add(new DealItemViewModel(deal));

}

}

private void OpenDealDetailAsync(string dealId)

{

var detailVm = \_serviceProvider.GetRequiredService<DealDetailPageViewModel>();

if (detailVm is IParameterReceiver parameterReceiver)

parameterReceiver.ReceiveParameter(dealId);

MessageBus.Current.SendMessage(detailVm, "NavigateToPage");

\_navigationService.NavigateTo(detailVm);

MessageBus.Current.SendMessage(dealId, "DealDetailsId");

}

// Команда для обновления списка сделок

public async Task RefreshDealsAsync()

{

await LoadDataAsync();

}

}

1. Листинг кода сервиса бизнес-логики «EmployeeDetailViewModel»

using System;

using System.Reactive;

using System.Reactive.Linq;

using System.Threading.Tasks;

using System.Windows.Input;

using Microsoft.EntityFrameworkCore;

using MsBox.Avalonia;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Data.Context;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class EmployeeDetailViewModel : PageViewModelBase, IParameterReceiver

{

private readonly DataContext \_context;

[Reactive] public Employee Employee { get; set; }

[Reactive] public bool CanFire { get; set; }

[Reactive] public bool CanRecover { get; set; }

public ICommand ResetPasswordCommand { get; }

public ICommand FireCommand { get; }

public ICommand DeleteCommand { get; }

public ICommand RecoverCommand { get; }

public EmployeeDetailViewModel(DataContext context)

{

\_context = context;

ResetPasswordCommand = ReactiveCommand.CreateFromTask(ResetPasswordImpl);

FireCommand = ReactiveCommand.CreateFromTask(FireImpl);

DeleteCommand = ReactiveCommand.CreateFromTask(DeleteImpl);

RecoverCommand = ReactiveCommand.CreateFromTask(RecoverImpl);

GetDataFromMessageBus();

}

public void ReceiveParameter(object parameter)

{

if (parameter is string listingId)

{

\_ = LoadEmployeeAsync(listingId);

}

}

private async Task ResetPasswordImpl()

{

throw new NotImplementedException();

}

private async Task FireImpl()

{

try

{

Employee.Fired = true;

CanFire = false;

CanRecover = true;

\_context.Attach(Employee);

await \_context.SaveChangesAsync();

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сотрудник уволен", ButtonEnum.Ok)

.ShowAsync();

}

catch (Exception e)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", e.InnerException.Message, ButtonEnum.Ok)

.ShowAsync();

}

}

private async Task RecoverImpl()

{

try

{

Employee.Fired = false;

CanFire = true;

CanRecover = false;

\_context.Attach(Employee);

await \_context.SaveChangesAsync();

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сотрудник восстановлен", ButtonEnum.Ok)

.ShowAsync();

}

catch (Exception e)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", e.InnerException.Message, ButtonEnum.Ok)

.ShowAsync();

}

}

private async Task DeleteImpl()

{

try

{

Employee.IsDeleted = true;

\_context.Attach(Employee);

await \_context.SaveChangesAsync();

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сотрудник удалён", ButtonEnum.Ok)

.ShowAsync();

}

catch (Exception e)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", e.InnerException.Message, ButtonEnum.Ok)

.ShowAsync();

}

}

private async Task LoadEmployeeAsync(string listingId)

{

try

{

Employee = await \_context.Employees

.Include(x => x.Photo)

.Include(x => x.Role)

.FirstAsync(l => l.Id == listingId);

if (Employee != null)

{

Title = $"Сотрудник №{Employee.Id} - {Employee.FullName}";

CanFire = !Employee.Fired;

CanRecover = Employee.Fired;

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Не удалось загрузить сотрудника: {ex.Message}")

.ShowAsync();

}

}

private void GetDataFromMessageBus()

{

MessageBus.Current

.Listen<string>("ListingDetailsId")

.SelectMany(async listingId =>

{

await LoadEmployeeAsync(listingId);

return Unit.Default;

})

.Subscribe();

}

}

1. Листинг кода сервиса бизнес-логики «EmployeesPageViewModel»

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using System.Threading.Tasks;

using System.Windows.Input;

using DynamicData;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.DependencyInjection;

using MsBox.Avalonia;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.DTOs;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class EmployeesPageViewModel : PageViewModelBase

{

private readonly DataContext \_context;

private readonly IAccountingService \_accountingService;

private readonly IPhotoService \_photoService;

private readonly IServiceProvider \_serviceProvider;

private readonly INavigationService \_navigationService;

private Employee? \_selectedEmployee;

[Reactive] public ObservableCollection<Employee> Employees { get; set; } = new();

[Reactive] public Employee NewEmployee { get; set; } = new();

[Reactive] public string Password { get; set; } = string.Empty;

[Reactive] public List<DictionaryValue> Roles { get; set; } = new();

[Reactive] public DictionaryValue SelectedRole { get; set; } = new();

// Заменяем коллекцию на одно свойство для фотографии

[Reactive] public UploadedPhoto? CurrentPhoto { get; set; }

[Reactive] public string PhotosSummary { get; set; } = "Фотография не добавлена";

public Employee SelectedEmployee

{

get => \_selectedEmployee;

set

{

this.RaiseAndSetIfChanged(ref \_selectedEmployee, value);

OpenEmployeeDetailAsync(\_selectedEmployee.Id);

\_selectedEmployee = null;

}

}

public bool HasPhoto => CurrentPhoto != null;

public ICommand CreateEmployeeCommand { get; }

public ReactiveCommand<Unit, Unit> SelectImageCommand { get; }

public ReactiveCommand<UploadedPhoto?, Unit> RemovePhotoCommand { get; }

public EmployeesPageViewModel(

DataContext context,

IAccountingService accountingService,

IPhotoService photoService, IServiceProvider serviceProvider, INavigationService navigationService)

{

\_context = context;

\_accountingService = accountingService;

\_photoService = photoService;

\_serviceProvider = serviceProvider;

\_navigationService = navigationService;

LoadData();

CreateEmployeeCommand = ReactiveCommand.CreateFromTask(CreateEmployeeAsync);

SelectImageCommand = ReactiveCommand.CreateFromTask(SelectImageAsync);

RemovePhotoCommand = ReactiveCommand.Create<UploadedPhoto?>(RemovePhoto);

}

private async void LoadData()

{

try

{

var employees = await \_context.Employees

.Include(x => x.Photo)

.Include(x => x.Role)

.ToListAsync();

Employees.Clear();

Employees.AddRange(employees);

if (!Roles.Any())

{

var roles = await \_context.DictionaryValues

.Where(x => x.DictionaryId == "employee\_role")

.ToListAsync();

Roles.AddRange(roles);

}

}

catch (System.Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Не удалось загрузить список сотрудников: {ex.Message}")

.ShowAsync();

}

}

private async Task CreateEmployeeAsync()

{

if (string.IsNullOrWhiteSpace(NewEmployee.FirstName) || string.IsNullOrWhiteSpace(NewEmployee.LastName))

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Заполните имя и фамилию сотрудника")

.ShowAsync();

return;

}

if (string.IsNullOrWhiteSpace(NewEmployee.Login) || string.IsNullOrWhiteSpace(Password))

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Заполните логин и пароль")

.ShowAsync();

return;

}

try

{

using var transaction = await \_context.Database.BeginTransactionAsync();

var passwordHashAndSalt = \_accountingService.HashPassword(Password);

NewEmployee.PasswordHash = passwordHashAndSalt.Hash;

NewEmployee.Salt = passwordHashAndSalt.Salt;

NewEmployee.RoleId = SelectedRole.Id;

await \_context.Employees.AddAsync(NewEmployee);

await \_context.SaveChangesAsync();

if (CurrentPhoto != null)

{

var photoIds = await \_photoService.SavePhotosToDatabaseAsync(

new List<UploadedPhoto> { CurrentPhoto },

NewEmployee.Id,

EntityTypeForPhoto.Employee);

NewEmployee.PhotoId = photoIds![0];

}

\_context.Attach(NewEmployee);

await \_context.SaveChangesAsync();

await transaction.CommitAsync();

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сотрудник успешно создан")

.ShowAsync();

ClearForm();

LoadData();

}

catch (DbUpdateException dbEx)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка базы данных", dbEx.InnerException?.Message ?? dbEx.Message)

.ShowAsync();

}

catch (System.Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Не удалось создать сотрудника: {ex.Message}")

.ShowAsync();

}

}

private async Task SelectImageAsync()

{

var newPhotos = await \_photoService.SelectImagesAsync();

if (newPhotos.Any())

{

CurrentPhoto = newPhotos.First();

UpdatePhotosSummary();

this.RaisePropertyChanged(nameof(HasPhoto));

}

}

private void RemovePhoto(UploadedPhoto? photo)

{

if (photo != null)

{

\_photoService.DisposeImagePreview(photo);

}

CurrentPhoto = null;

UpdatePhotosSummary();

this.RaisePropertyChanged(nameof(HasPhoto));

}

private void UpdatePhotosSummary()

{

if (CurrentPhoto == null)

{

PhotosSummary = "Фотография не добавлена";

return;

}

PhotosSummary = $"{CurrentPhoto.FileName} • {\_photoService.FormatFileSize(CurrentPhoto.FileSize)}";

}

private void ClearForm()

{

NewEmployee = new Employee();

Password = string.Empty;

CurrentPhoto = null;

UpdatePhotosSummary();

this.RaisePropertyChanged(nameof(HasPhoto));

}

private void OpenEmployeeDetailAsync(string selectedEmployeeId)

{

var detailVm = \_serviceProvider.GetRequiredService<EmployeeDetailViewModel>();

if (detailVm is IParameterReceiver parameterReceiver)

parameterReceiver.ReceiveParameter(selectedEmployeeId);

MessageBus.Current.SendMessage(detailVm, "NavigateToPage");

\_navigationService.NavigateTo(detailVm);

MessageBus.Current.SendMessage(selectedEmployeeId, "SelectedEmployeeId");

}

}

1. Листинг кода сервиса бизнес-логики «HomePageViewModel»

namespace RealtorTool.Desktop.ViewModels.Pages;

public class HomePageViewModel : PageViewModelBase

{

public HomePageViewModel()

{

}

}

1. Листинг кода сервиса бизнес-логики «ListingDetailViewModel»

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Reactive;

using System.Reactive.Linq;

using System.Threading.Tasks;

using DynamicData;

using Microsoft.EntityFrameworkCore;

using MsBox.Avalonia;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.RealtyModels;

using RealtorTool.Core.DbEntities.Views;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.ViewModels.Pages.RealtyDetailPages;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class ListingDetailViewModel : PageViewModelBase, IParameterReceiver

{

private readonly DataContext \_context;

private readonly IMatchingService \_matchingService;

private Employee \_authEmployee;

[Reactive] public Listing? Listing { get; set; }

[Reactive] public PageViewModelBase RealtyViewModel { get; set; }

[Reactive] public ClientRequest? SellRequest { get; set; }

[Reactive] public ObservableCollection<PotentialMatch> FoundMatches { get; set; } = new();

[Reactive] public bool HasMatches { get; set; }

[Reactive] public bool IsSearching { get; set; }

[Reactive] public PotentialMatch SelectedMatch { get; set; }

[Reactive] public bool IsMatchSelected { get; set; }

[Reactive] public bool IsCreatingDeal { get; set; }

public ReactiveCommand<Unit, Unit> SearchMatchesCommand { get; }

public ReactiveCommand<PotentialMatch, Unit> SelectMatchCommand { get; }

public ReactiveCommand<Unit, Unit> RefreshListingCommand { get; }

public string OwnerFullName => Listing?.Owner != null

? $"{Listing.Owner.FirstName} {Listing.Owner.LastName}".Trim()

: string.Empty;

public string EmployeeFullName => Listing?.ResponsibleEmployee != null

? $"{Listing.ResponsibleEmployee.FirstName} {Listing.ResponsibleEmployee.LastName}".Trim()

: string.Empty;

public ListingDetailViewModel(

DataContext context,

IMatchingService matchingService)

{

\_context = context;

\_matchingService = matchingService;

SearchMatchesCommand = ReactiveCommand.CreateFromTask(SearchMatchesAsync);

SelectMatchCommand = ReactiveCommand.Create<PotentialMatch>(SelectMatch);

RefreshListingCommand = ReactiveCommand.CreateFromTask(RefreshListingAsync);

GetDataFromMessageBus();

}

private void GetDataFromMessageBus()

{

MessageBus.Current

.Listen<string>("ListingDetailsId")

.SelectMany(async listingId =>

{

await LoadListingAsync(listingId);

await LoadSellRequestAsync();

Title = $"Заявка №{listingId}";

LoadRealtyView();

await SearchMatchesAsync();

return Unit.Default;

})

.Subscribe();

MessageBus.Current

.Listen<Employee>("CurrentAuth")

.Subscribe(x =>

{

\_authEmployee = x;

});

}

public void ReceiveParameter(object parameter)

{

if (parameter is string listingId)

{

\_ = LoadListingAsync(listingId);

}

}

private async Task LoadListingAsync(string listingId)

{

try

{

Listing = await \_context.Listings

.Include(l => l.Realty)

.Include(l => l.ListingType)

.Include(l => l.Status)

.Include(l => l.Currency)

.Include(l => l.Owner)

.Include(l => l.ResponsibleEmployee)

.Include(l => l.ClientRequests)

.FirstOrDefaultAsync(l => l.Id == listingId);

if (Listing != null)

{

Title = $"Заявка №{Listing.Id} - {Listing.ListingType?.Value}";

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Не удалось загрузить заявку: {ex.Message}")

.ShowAsync();

}

}

private async Task LoadSellRequestAsync()

{

if (Listing?.Id == null) return;

try

{

// Ищем заявку на продажу/сдачу для этого листинга

SellRequest = await \_context.ClientRequests

.Include(cr => cr.Client)

.Include(cr => cr.Employee)

.FirstOrDefaultAsync(cr =>

cr.ListingId == Listing.Id &&

(cr.Type == ApplicationType.Sale || cr.Type == ApplicationType.RentOut));

if (SellRequest == null)

{

// Можно создать заявку автоматически или показать сообщение

await MessageBoxManager

.GetMessageBoxStandard("Информация",

"Для этого объявления не найдена заявка на продажу/сдачу. " +

"Создайте заявку для поиска совпадений.")

.ShowAsync();

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка при загрузке заявки: {ex.Message}")

.ShowAsync();

}

}

private void LoadRealtyView()

{

var realty = Listing?.Realty;

if (realty == null)

{

RealtyViewModel = null;

return;

}

switch (realty)

{

case Apartment apt:

RealtyViewModel = new ApartmentDetailPageViewModel();

MessageBus.Current.SendMessage(apt);

break;

case PrivateHouse house:

RealtyViewModel = new PrivateHouseDetailPageViewModel();

MessageBus.Current.SendMessage(house);

break;

case Area area:

RealtyViewModel = new AreaDetailPageViewModel();

MessageBus.Current.SendMessage(area);

break;

default:

RealtyViewModel = null;

break;

}

}

private async Task SearchMatchesAsync()

{

if (SellRequest?.Id == null)

{

await MessageBoxManager

.GetMessageBoxStandard("Информация",

"Не найдена заявка на продажу/сдачу для поиска совпадений")

.ShowAsync();

return;

}

IsSearching = true;

try

{

var matches = await \_matchingService.FindPotentialMatchesForSellRequestAsync(SellRequest.Id);

FoundMatches.Clear();

FoundMatches.AddRange(matches);

HasMatches = matches.Any();

// Сбрасываем выбранный матч

SelectedMatch = null;

IsMatchSelected = false;

if (!HasMatches)

{

await MessageBoxManager

.GetMessageBoxStandard("Информация", "Подходящих совпадений не найдено")

.ShowAsync();

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка при поиске совпадений: {ex.Message}")

.ShowAsync();

}

finally

{

IsSearching = false;

}

}

private void SelectMatch(PotentialMatch match)

{

SelectedMatch = match;

IsMatchSelected = match != null;

}

public async Task CreateDealAsync(PotentialMatch match)

{

if (IsCreatingDeal || match == null || SellRequest?.Id == null) return;

IsCreatingDeal = true;

try

{

// Подтверждение создания сделки

var messageBox = MessageBoxManager.GetMessageBoxStandard(

"Создание сделки",

$"Вы уверены, что хотите создать сделку?\n\n" +

$"Объект: {match.RealtyName}\n" +

$"Тип: {match.RealtyTypeDisplayName}\n" +

$"Район: {match.District}\n" +

$"Цена: {match.ListingPrice:N0} руб.\n" +

$"Качество совпадения: {match.MatchScore}%",

ButtonEnum.YesNo,

Icon.Question

);

var result = await messageBox.ShowAsync();

if (result == ButtonResult.Yes)

{

// Создаем сделку через сервис matching

var createResult = await \_matchingService.CreateMatchAsync(match.BuyRequestId, match.SellRequestId);

if (createResult)

{

await MessageBoxManager

.GetMessageBoxStandard("Успех", "Сделка успешно создана!")

.ShowAsync();

// Обновляем статусы и перезагружаем данные

await UpdateRequestStatuses();

await RefreshListingAsync();

await SearchMatchesAsync();

// Отправляем сообщение об обновлении

MessageBus.Current.SendMessage("DealCreated", "GlobalNotification");

}

else

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Не удалось создать сделку")

.ShowAsync();

}

}

}

catch (Exception ex)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", $"Ошибка при создании сделки: {ex.Message}")

.ShowAsync();

}

finally

{

IsCreatingDeal = false;

}

}

private async Task UpdateRequestStatuses()

{

if (SellRequest?.Id == null) return;

try

{

// Обновляем статус заявки на продажу

var matchedStatus = ApplicationStatus.Matched;

SellRequest.Status = matchedStatus;

// Сохраняем изменения

await \_context.SaveChangesAsync();

}

catch (Exception ex)

{

Console.WriteLine($"Ошибка при обновлении статусов: {ex.Message}");

}

}

private async Task RefreshListingAsync()

{

if (Listing?.Id != null)

{

await LoadListingAsync(Listing.Id);

await LoadSellRequestAsync();

}

}

// Свойства для отображения статистики

public string MatchesSummary => HasMatches

? $"Найдено {FoundMatches.Count} совпадений"

: "Совпадения не найдены";

public int GoodMatchesCount => FoundMatches.Count(m => m.IsGoodMatch);

public int PerfectMatchesCount => FoundMatches.Count(m => m.IsPerfectMatch);

public string MatchQualityDescription

{

get

{

if (!HasMatches) return "Совпадения не найдены";

if (PerfectMatchesCount > 0) return $"{PerfectMatchesCount} идеальных совпадений";

if (GoodMatchesCount > 0) return $"{GoodMatchesCount} хороших совпадений";

return "Совпадения низкого качества";

}

}

public string ListingStatusDescription => Listing?.Status?.Value ?? "Неизвестно";

public bool CanCreateDeals => HasMatches && !IsCreatingDeal;

}

1. Листинг кода сервиса бизнес-логики «MyProfilePageViewModel»

using System;

using System.Threading.Tasks;

using System.Windows.Input;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Data.Context;

namespace RealtorTool.Desktop.ViewModels.Pages;

public class MyProfilePageViewModel : PageViewModelBase

{

private readonly DataContext \_context;

[Reactive] public Employee? CurrentPerson { get; set; }

public ICommand SaveChanges { get; }

public MyProfilePageViewModel(DataContext context)

{

\_context = context;

MessageBus.Current

.Listen<Employee>("CurrentAuth")

.Subscribe(x =>

{

CurrentPerson = x;

});

SaveChanges = ReactiveCommand.CreateFromTask(SaveChangesProfileDataAsync);

}

private async Task SaveChangesProfileDataAsync()

{

if (CurrentPerson == default)

return;

\_context.Update(CurrentPerson);

await \_context.SaveChangesAsync();

}

}

1. Листинг кода сервиса бизнес-логики «PersonProfilePageViewModel»

namespace RealtorTool.Desktop.ViewModels.Pages;

public class PersonProfilePageViewModel : PageViewModelBase

{

public PersonProfilePageViewModel()

{

Title = "Профиль пользователя";

}

}

1. Листинг кода сервиса бизнес-логики «AuthorizationWindowViewModel»

using System.Reactive;

using System.Threading.Tasks;

using Avalonia.Controls;

using Microsoft.EntityFrameworkCore;

using MsBox.Avalonia;

using MsBox.Avalonia.Enums;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Data;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.Views;

using RealtorTool.Services.Interfaces;

using MainWindow = RealtorTool.Desktop.Views.Windows.MainWindow;

namespace RealtorTool.Desktop.ViewModels;

/// <summary>

/// VM Окна авторизации.

/// </summary>

public class AuthorizationWindowViewModel : ViewModelBase

{

private readonly IWindowService \_windowService;

private readonly IAccountingService \_accountingService;

/// <summary>

/// Поле логина.

/// </summary>

[Reactive] public string? Login { get; set; }

/// <summary>

/// Поле пароля.

/// </summary>

[Reactive] public string? Password { get; set; }

/// <summary>

/// Бинд для кнопки авторизации.

/// </summary>

public ReactiveCommand<Window, Unit> Auth { get; }

/// <summary>

/// Конструктор.

/// </summary>

public AuthorizationWindowViewModel(

IWindowService windowService,

IAccountingService accountingService)

{

\_windowService = windowService;

\_accountingService = accountingService;

Auth = ReactiveCommand.CreateFromTask<Window>(AuthAsync);

}

/// <summary>

/// Конструктор по умолчанию

/// </summary>

public AuthorizationWindowViewModel(IAccountingService accountingService)

{

\_accountingService = accountingService;

}

/// <summary>

/// Метод проверки авторизации.

/// </summary>

private async Task AuthAsync(Window currentWindow)

{

var authPerson = await \_accountingService.LoginAsync(Login, Password);

if (authPerson == null)

{

await MessageBoxManager

.GetMessageBoxStandard("Ошибка", "Логин или пароль не совпадают", ButtonEnum.Ok)

.ShowAsync();

return;

}

\_windowService.ShowWindow<MainWindow>();

MessageBus.Current.SendMessage(authPerson, "CurrentAuth");

currentWindow.Close();

}

}

1. Листинг кода сервиса бизнес-логики «MainWindowViewModel»

using System;

using System.Collections.ObjectModel;

using System.Reactive.Linq;

using System.Windows.Input;

using ReactiveUI;

using ReactiveUI.Fody.Helpers;

using RealtorTool.Core.DbEntities;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.ViewModels.Pages;

using RealtorTool.Services.Interfaces;

namespace RealtorTool.Desktop.ViewModels.Windows;

/// <summary>

/// VM Главного окна.

/// </summary>

public class MainWindowViewModel : ViewModelBase

{

private readonly INavigationService \_navigationService;

private readonly IWindowService \_windowService;

private Employee CurrentAuth { get; set; }

public ObservableCollection<PageViewModelBase> PaneItems { get; set; }

[Reactive] public PageViewModelBase SelectedPageItem { get; set; }

public ICommand OpenMyProfilePage { get; private set; }

public ICommand OpenHomePage { get; private set; }

public ICommand OpenCreatingApplicationPage { get; private set; }

public ICommand OpenEmployeesPage { get; private set; }

public ICommand OpenApplicationsPage { get; private set; }

public ICommand LogoutCommand { get; private set; }

public ICommand OpenDealListPage { get; private set; }

/// <summary>

/// Конструктор.

/// </summary>

public MainWindowViewModel(

MyProfilePageViewModel myProfilePageViewModel,

PersonProfilePageViewModel personProfilePageViewModel,

HomePageViewModel homePageViewModel,

CreatingApplicationPageViewModel creatingApplicationPageViewModel,

EmployeesPageViewModel employeesPageViewModel,

ApplicationsPageViewModel applicationsPageViewModel,

DealsListPageViewModel dealsListPageViewModel,

INavigationService navigationService,

IWindowService windowService)

{

\_navigationService = navigationService;

\_windowService = windowService;

PaneItems =

[

myProfilePageViewModel,

personProfilePageViewModel,

homePageViewModel,

creatingApplicationPageViewModel,

employeesPageViewModel,

applicationsPageViewModel,

dealsListPageViewModel

];

SelectedPageItem = PaneItems[2];

\_navigationService.OnNavigationRequested += OnNavigationRequested;

InitialButtons();

GetDataFromMessageBus();

}

private void InitialButtons()

{

OpenMyProfilePage = ReactiveCommand.Create(OpenMyProfilePageImpl);

OpenHomePage = ReactiveCommand.Create(OpenHomePageImpl);

OpenCreatingApplicationPage = ReactiveCommand.Create(OpenCreatingApplicationPageImpl);

OpenEmployeesPage = ReactiveCommand.Create(OpenEmployeesPageImpl);

OpenApplicationsPage = ReactiveCommand.Create(OpenApplicationsPageImpl);

OpenDealListPage = ReactiveCommand.Create(OpenDealListPageImpl);

LogoutCommand = ReactiveCommand.Create(LogoutImpl);

}

private void OnNavigationRequested(PageViewModelBase page)

{

if (page != null)

{

SelectedPageItem = page;

}

}

private void LogoutImpl()

{

CurrentAuth = null;

MessageBus.Current.SendMessage<Employee?>(null, "CurrentAuth");

\_windowService.Logout();

}

private void OpenHomePageImpl() => SelectedPageItem = PaneItems[2];

private void OpenMyProfilePageImpl() => SelectedPageItem = PaneItems[0];

private void OpenCreatingApplicationPageImpl() => SelectedPageItem = PaneItems[3];

private void OpenEmployeesPageImpl() => SelectedPageItem = PaneItems[4];

private void OpenApplicationsPageImpl() => SelectedPageItem = PaneItems[5];

private void OpenDealListPageImpl() => SelectedPageItem = PaneItems[6];

private void GetDataFromMessageBus()

{

MessageBus.Current

.Listen<Employee>("CurrentAuth")

.Subscribe(x => { CurrentAuth = x; });

MessageBus.Current

.Listen<PageViewModelBase>("NavigateToPage")

.ObserveOn(RxApp.MainThreadScheduler) // Важно для UI

.Subscribe(page =>

{

if (page != null)

{

SelectedPageItem = page;

}

});

}

public void Dispose()

{

\_navigationService.OnNavigationRequested -= OnNavigationRequested;

}

}

1. Листинг кода сервиса бизнес-логики «PageViewModelBase»

namespace RealtorTool.Desktop.ViewModels;

public class PageViewModelBase : ViewModelBase

{

public string? Title { get; set; }

}

1. Листинг кода сервиса бизнес-логики «PotentialMatchViewModel»

using System.Threading.Tasks;

using Microsoft.EntityFrameworkCore;

using RealtorTool.Core.DbEntities;

using RealtorTool.Core.DbEntities.Views;

using RealtorTool.Core.Enums;

using RealtorTool.Data.Context;

namespace RealtorTool.Desktop.ViewModels;

public class PotentialMatchViewModel : ViewModelBase

{

private readonly PotentialMatch \_match;

private readonly DataContext \_context;

public PotentialMatchViewModel(PotentialMatch match, DataContext context)

{

\_match = match;

\_context = context;

}

public string BuyRequestId => \_match.BuyRequestId;

public string SellRequestId => \_match.SellRequestId;

public decimal MatchScore => \_match.MatchScore;

public string MatchDescription => \_match.MatchDescription;

public bool IsGoodMatch => \_match.IsGoodMatch;

public string RealtyTypeDisplay => \_match.RealtyType switch

{

RealtyType.Apartment => "Квартира",

RealtyType.PrivateHouse => "Частный дом",

RealtyType.Area => "Земельный участок",

\_ => "Неизвестно"

};

public string PriceInfo => \_match.ListingPrice.ToString("N0") + " руб.";

public async Task<Realty?> GetRealtyAsync()

{

var sellRequest = await \_context.ClientRequests

.Include(cr => cr.Listing)

.ThenInclude(l => l.Realty)

.FirstOrDefaultAsync(cr => cr.Id == \_match.SellRequestId);

return sellRequest?.Listing?.Realty;

}

public async Task<Client?> GetSellerAsync()

{

var sellRequest = await \_context.ClientRequests

.Include(cr => cr.Client)

.FirstOrDefaultAsync(cr => cr.Id == \_match.SellRequestId);

return sellRequest?.Client;

}

}

1. Листинг кода сервиса бизнес-логики «ViewModelBase»

using ReactiveUI;

namespace RealtorTool.Desktop.ViewModels;

public class ViewModelBase : ReactiveObject

{

}

1. Листинг кода «Program»

using Avalonia;

using Avalonia.ReactiveUI;

using System;

using System.IO;

using HotAvalonia;

using RealtorTool.Desktop.ViewModels;

using RealtorTool.Desktop.ViewModels.Pages;

using RealtorTool.Desktop.Views.Pages;

using RealtorTool.Desktop.Views.Windows;

using RealtorTool.Services.Interfaces;

using MainWindow = RealtorTool.Desktop.Views.Windows.MainWindow;

using Microsoft.EntityFrameworkCore;

using Microsoft.Extensions.Configuration;

using Microsoft.Extensions.DependencyInjection;

using RealtorTool.Data.Context;

using RealtorTool.Desktop.Converters;

using RealtorTool.Desktop.Services.Implementations;

using RealtorTool.Desktop.Services.Interfaces;

using RealtorTool.Desktop.ViewModels.Items;

using RealtorTool.Desktop.ViewModels.Pages.ApplicationPages;

using RealtorTool.Desktop.ViewModels.Pages.RealtyDetailPages;

using RealtorTool.Desktop.ViewModels.Windows;

using RealtorTool.Desktop.Views.Pages.ApplicationPages;

using RealtorTool.Desktop.Views.Pages.RealtyDetailPages;

namespace RealtorTool.Desktop;

sealed class Program

{

public static IServiceProvider ServiceProvider { get; private set; }

[STAThread]

public static void Main(string[] args) => BuildAvaloniaApp()

.StartWithClassicDesktopLifetime(args);

public static AppBuilder BuildAvaloniaApp()

{

var services = new ServiceCollection();

ConfigureServices(services);

ServiceProvider = services.BuildServiceProvider();

return AppBuilder.Configure<App>()

.UsePlatformDetect()

.WithInterFont()

.LogToTrace()

.UseReactiveUI();

}

private static void ConfigureServices(IServiceCollection services)

{

var configuration = new ConfigurationBuilder()

.SetBasePath(Directory.GetCurrentDirectory())

.AddJsonFile("appsettings.json", optional: false, reloadOnChange: true)

.Build();

var connectionString = configuration.GetConnectionString("DefaultConnection");

if (string.IsNullOrEmpty(connectionString))

throw new InvalidOperationException("Connection string 'DefaultConnection' not found.");

// Регистрация DataContext

services.AddDbContext<DataContext>(options =>

options.UseNpgsql(connectionString));

// Регистрация сервисов

services.AddSingleton<DateTimeFormatConverter>();

services.AddSingleton<StatusToColorConverter>();

services.AddSingleton<MatchScoreToColorConverter>();

services.AddSingleton<MatchScoreToBackgroundConverter>();

services.AddSingleton<ByteArrayToImageConverter>();

services.AddSingleton<BooleanToColorConverter>();

services.AddSingleton<MatchesCountToColorConverter>();

services.AddSingleton<TrueToFalseConverter>();

services.AddSingleton<ObjectIsNotNullConverter>();

services.AddSingleton<IWindowService, WindowService>();

services.AddSingleton<IAccountingService, AccountingService>();

services.AddSingleton<INavigationService, NavigationService>();

services.AddSingleton<IPhotoService, PhotoService>();

services.AddSingleton<IMatchingService, MatchingService>();

// Регистрация ViewModels

services.AddTransient<AuthorizationWindowViewModel>();

services.AddTransient<MainWindowViewModel>();

services.AddTransient<MyProfilePageViewModel>();

services.AddTransient<PersonProfilePageViewModel>();

services.AddTransient<HomePageViewModel>();

services.AddTransient<CreatingApplicationPageViewModel>();

services.AddTransient<EmployeesPageViewModel>();

services.AddTransient<ApplicationsPageViewModel>();

services.AddTransient<ListingDetailViewModel>();

services.AddTransient<ApartmentDetailPageViewModel>();

services.AddTransient<AreaDetailPageViewModel>();

services.AddTransient<PrivateHouseDetailPageViewModel>();

services.AddTransient<ListingItemViewModel>();

services.AddTransient<EmployeeDetailViewModel>();

services.AddTransient<DealDetailPageViewModel>();

// Регистрация страниц с ленивой загрузкой

services.AddTransient<BuyApplicationPageViewModel>();

services.AddTransient<Func<BuyApplicationPageViewModel>>(sp =>

() => sp.GetRequiredService<BuyApplicationPageViewModel>());

services.AddTransient<DealsListPageViewModel>();

services.AddTransient<Func<DealsListPageViewModel>>(sp =>

() => sp.GetRequiredService<DealsListPageViewModel>());

services.AddTransient<SellApplicationPageViewModel>();

services.AddTransient<Func<SellApplicationPageViewModel>>(sp =>

() => sp.GetRequiredService<SellApplicationPageViewModel>());

services.AddTransient<LeaseApplicationPageViewModel>();

services.AddTransient<Func<LeaseApplicationPageViewModel>>(sp =>

() => sp.GetRequiredService<LeaseApplicationPageViewModel>());

services.AddTransient<RentalApplicationPageViewModel>();

services.AddTransient<Func<RentalApplicationPageViewModel>>(sp =>

() => sp.GetRequiredService<RentalApplicationPageViewModel>());

// Регистрация View

services.AddTransient<AuthorizationWindow>();

services.AddTransient<MainWindow>();

services.AddTransient<MyProfilePageView>();

services.AddTransient<PersonProfilePageView>();

services.AddTransient<HomePageView>();

services.AddTransient<CreatingApplicationPageView>();

services.AddTransient<EmployeesPageView>();

services.AddTransient<BuyApplicationPageView>();

services.AddTransient<SellApplicationPageView>();

services.AddTransient<LeaseApplicationPageView>();

services.AddTransient<RentalApplicationPageView>();

services.AddTransient<ApplicationsPageView>();

services.AddTransient<ListingDetailView>();

services.AddTransient<ApartmentDetailPageView>();

services.AddTransient<AreaDetailPageView>();

services.AddTransient<PrivateHouseDetailPageView>();

services.AddTransient<EmployeeDetailView>();

services.AddTransient<DealsListPageView>();

services.AddTransient<DealDetailPageView>();

}

}

# Результаты работы программы

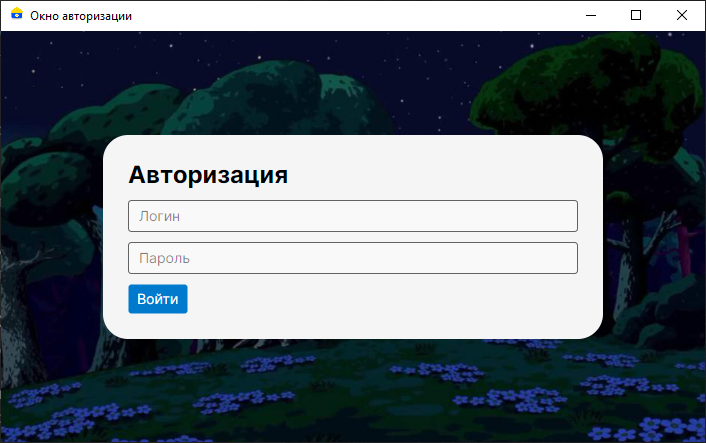


Рисунок Г.1 – Окно авторизации

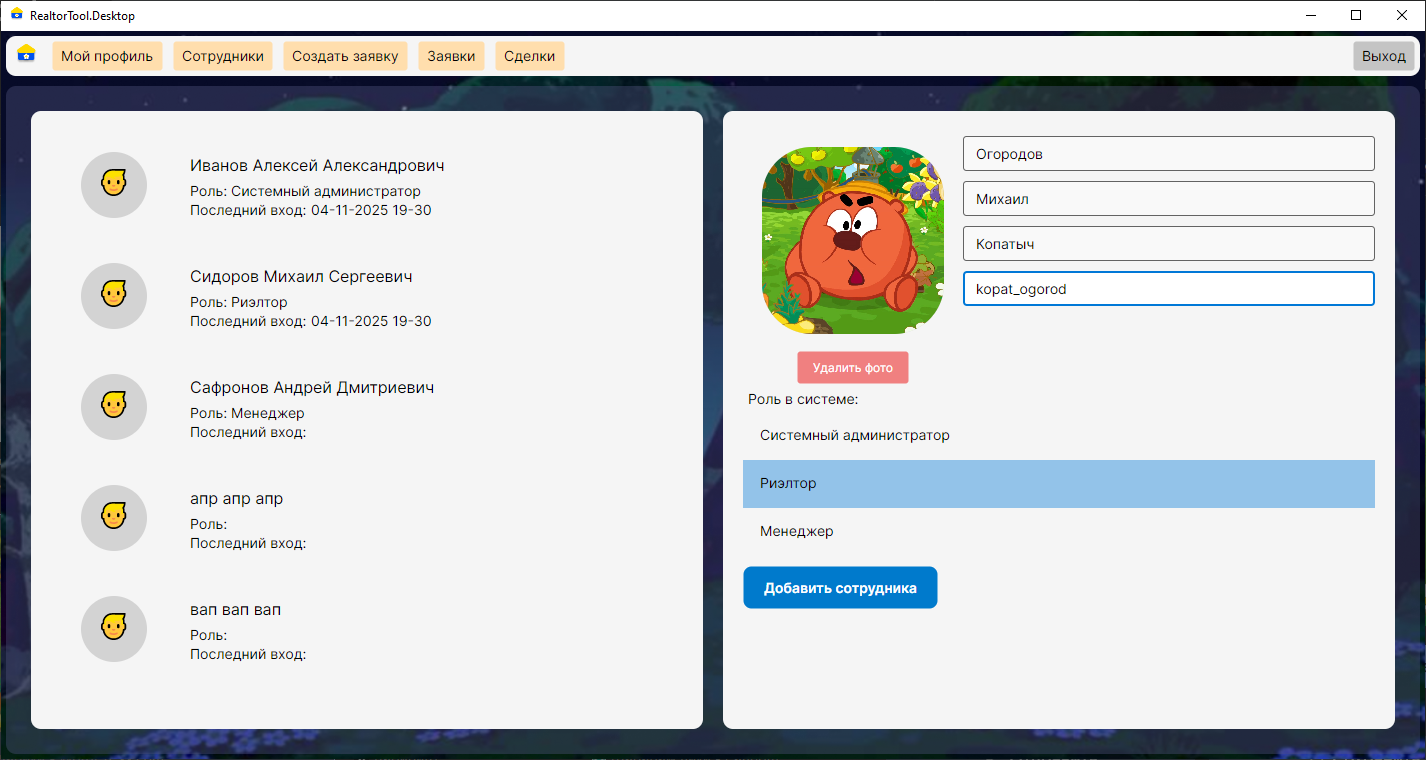


Рисунок Г.2 – Окно управления сотрудниками

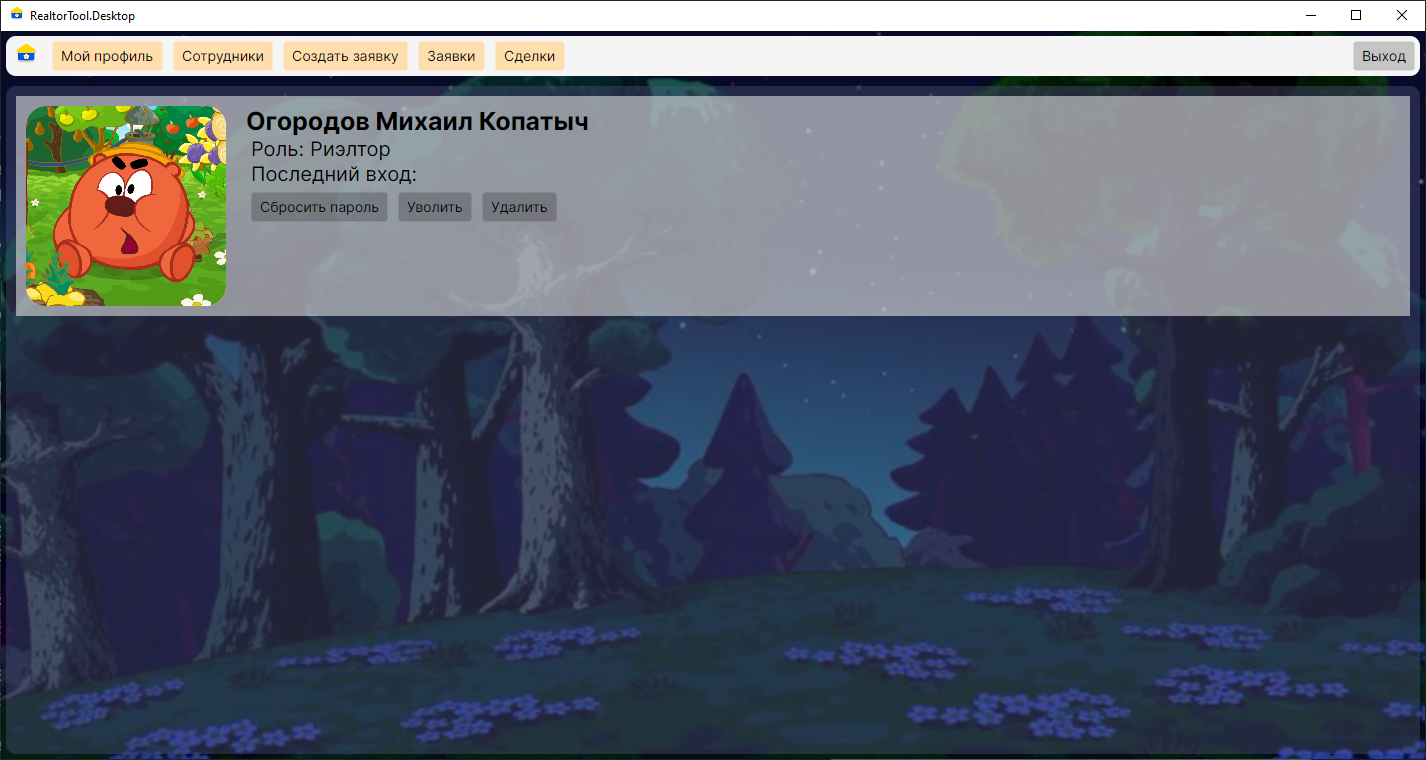


Рисунок Г.3 – Окно карточки сотрудника

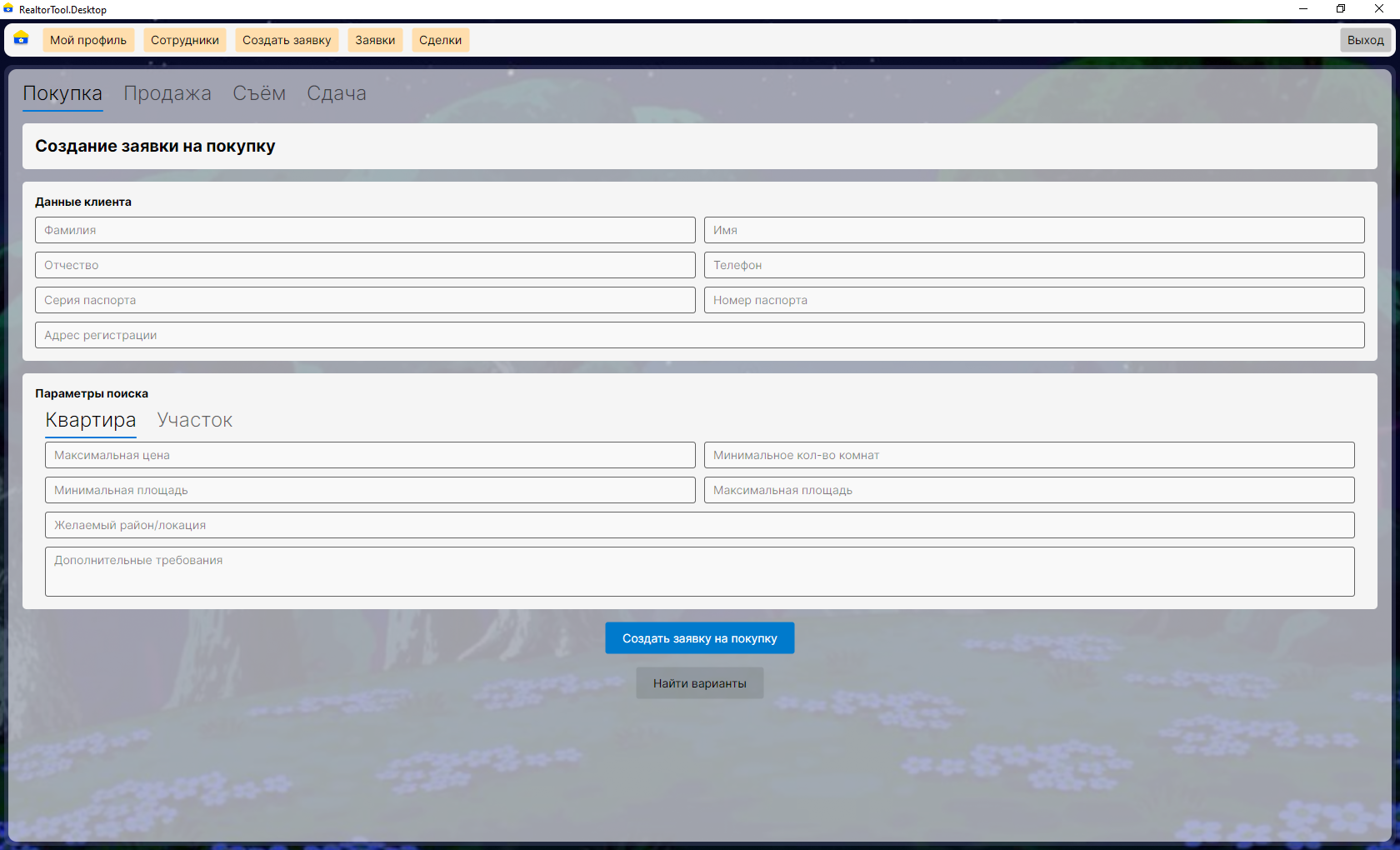


Рисунок Г.4 – Окно создания заявки на покупку квартиры

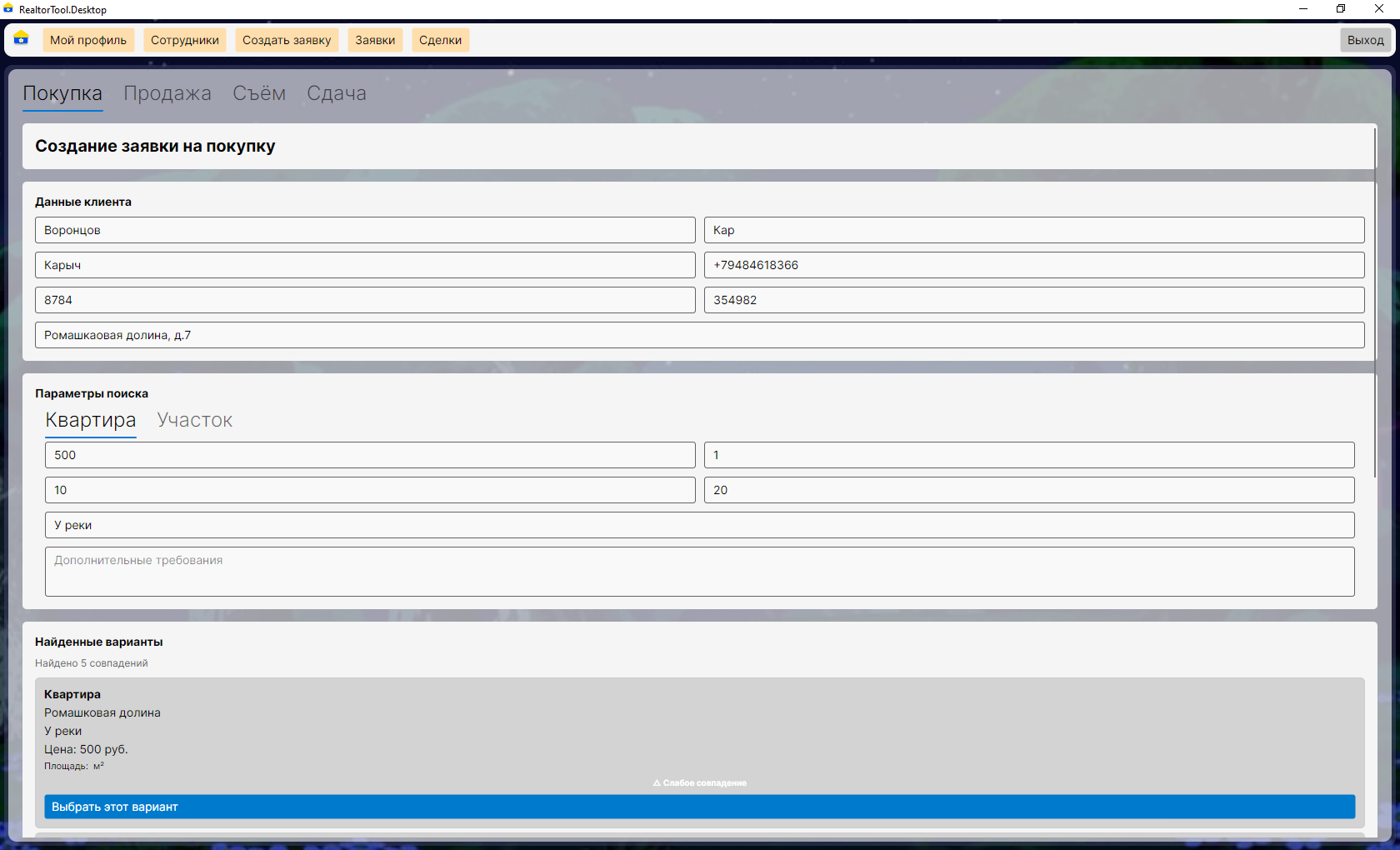


Рисунок Г.5 – Окно создания заявки на покупку участка

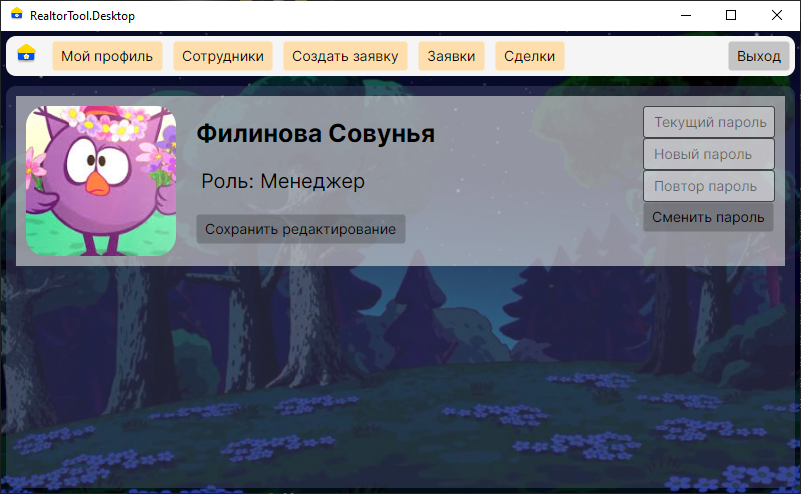


Рисунок Г.6 – Окно моего профиля

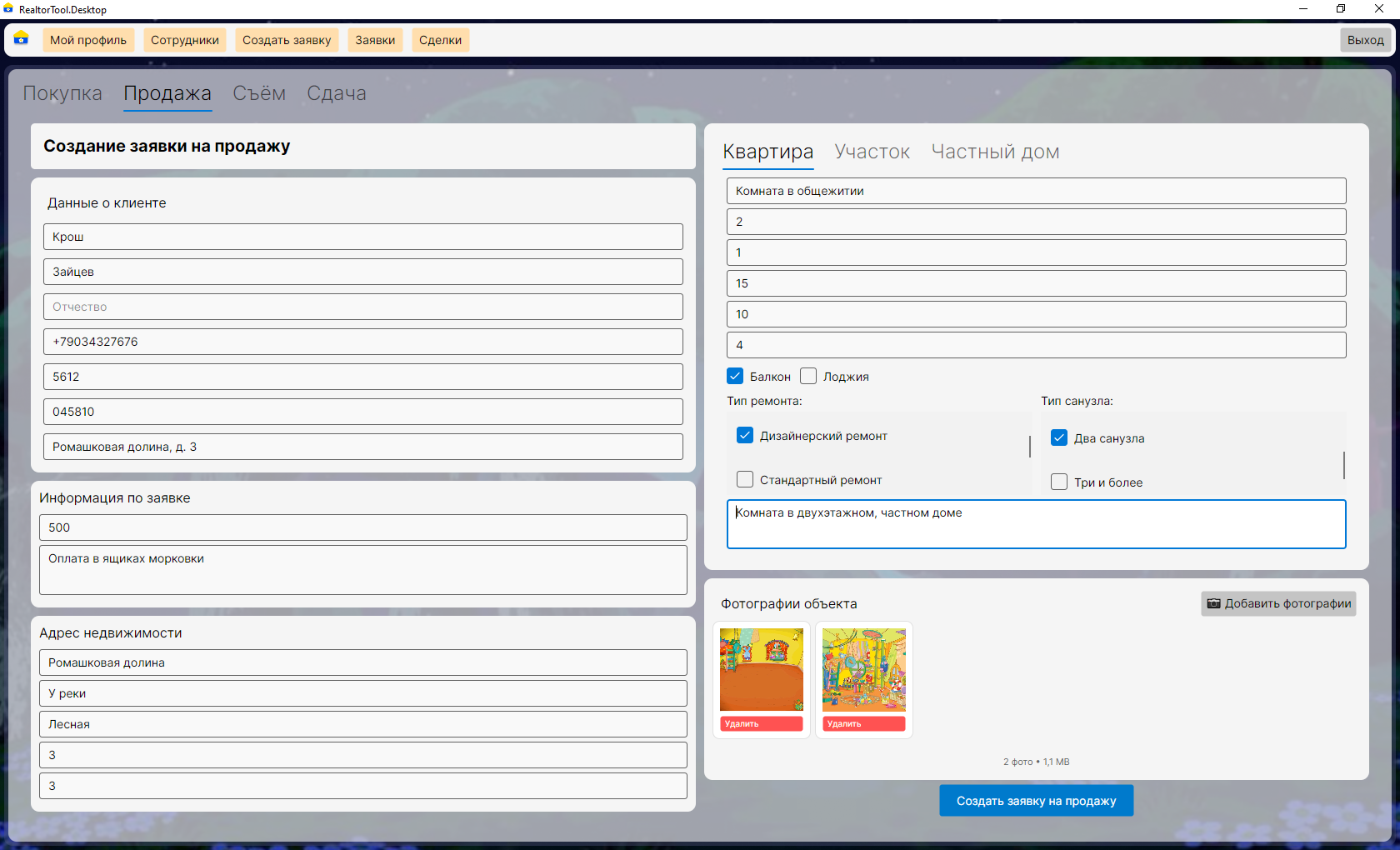


Рисунок Г.7 – Окно создания заявки на продажу квартиры

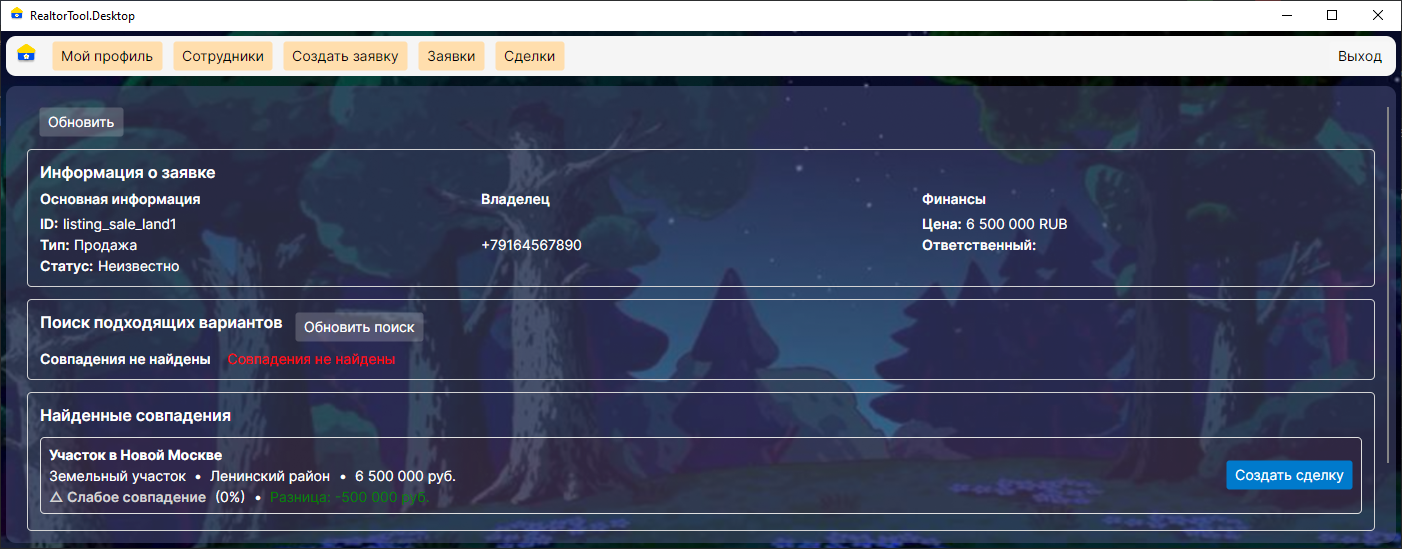


Рисунок Г.8 – Окно карточки заявки

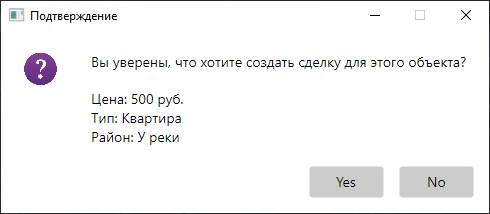


Рисунок Г.9 – Окно подтверждения связывания заявок

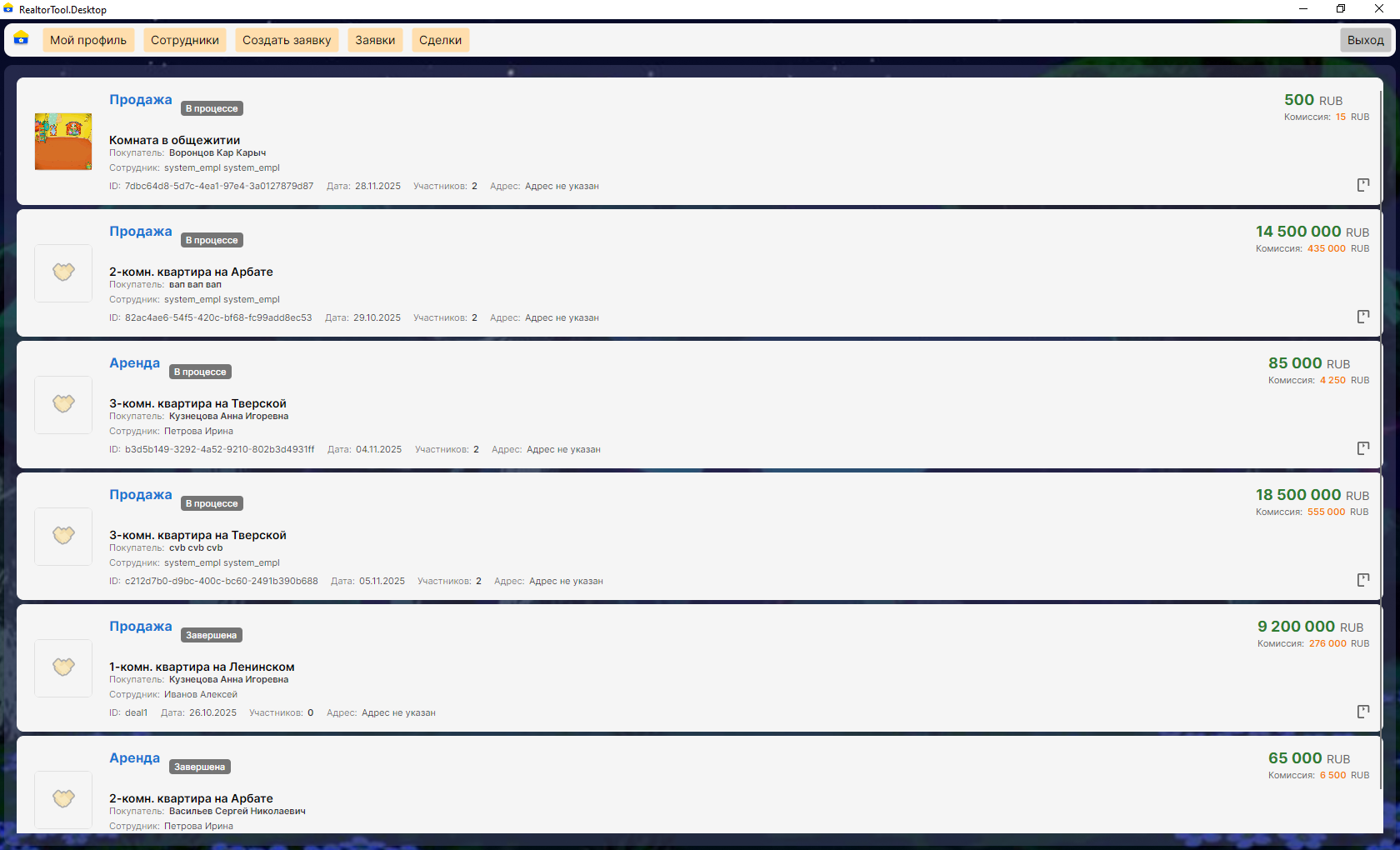


Рисунок Г.10 – Окно реестра сделок

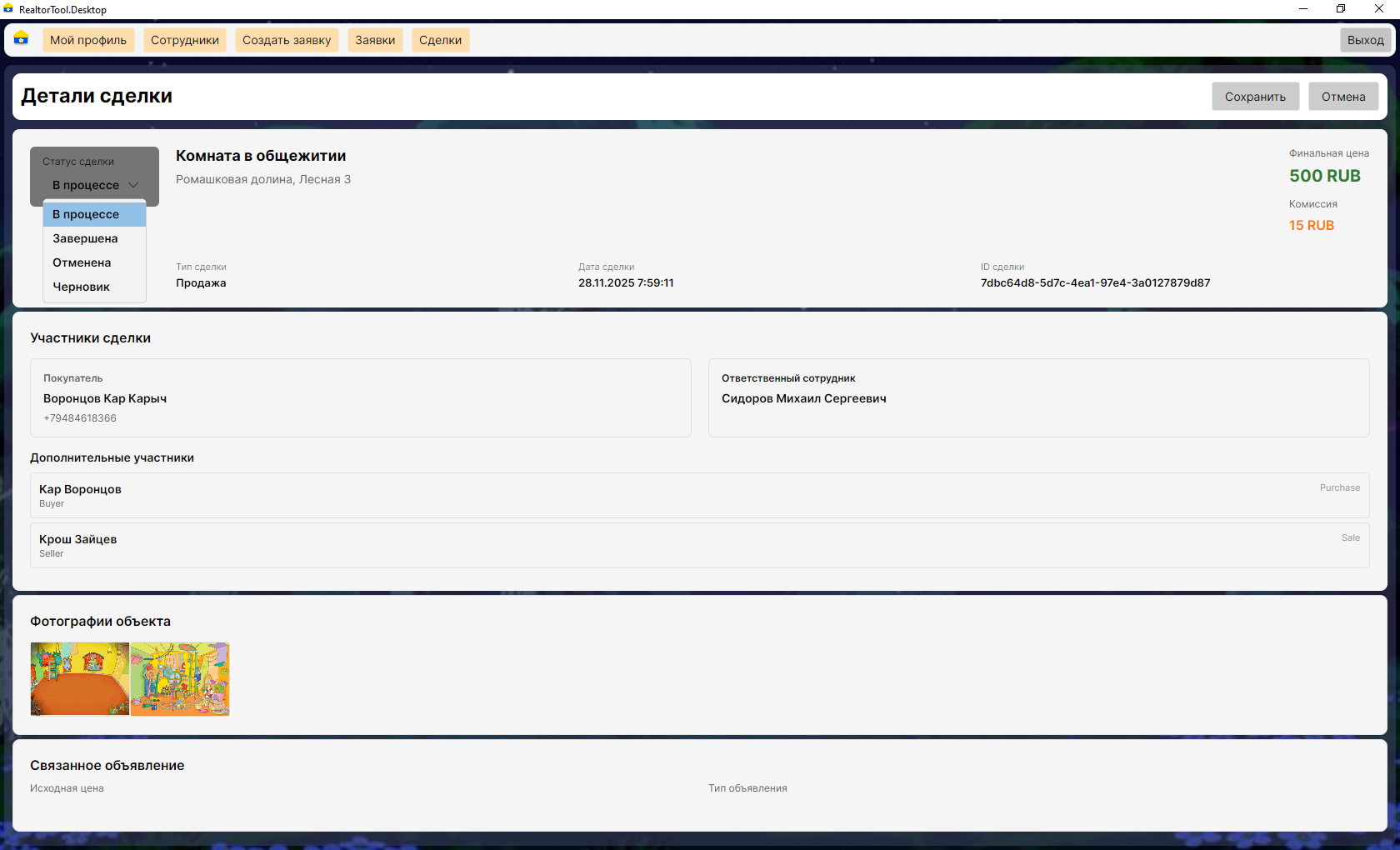


Рисунок Г.11 – Окно карточки сделки



Г.12 – Окно реестра заявок