СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире рынок недвижимости характеризуется высокой динамичностью и сложностью процессов управления. Агентства недвижимости сталкиваются с необходимостью координации множества объектов недвижимости различных типов, управления базой клиентов с учетом их потребностей и предпочтений, отслеживания сделок на всех этапах их жизненного цикла и организации аукционных торгов. Ручное управление этими процессами становится неэффективным и подверженным ошибкам, особенно при росте масштаба агентства и увеличении количества объектов и клиентов.

Разработка автоматизированной системы управления агентством недвижимости позволяет решить следующие актуальные задачи:

1 Централизованное управление каталогом недвижимости. Система обеспечивает единое хранилище информации об объектах недвижимости различных типов (квартиры, дома, коммерческая недвижимость) с возможностью быстрого поиска по адресу, цене и другим параметрам, что позволяет эффективно подбирать объекты под требования клиента.

2 Управление клиентской базой. Система обеспечивает ведение базы клиентов с валидацией контактных данных, поиск клиентов по имени или телефону, а также отслеживание истории взаимодействия с каждым клиентом, что способствует повышению качества обслуживания.

3 Контроль сделок и их статусов. Система обеспечивает создание и отслеживание сделок между клиентами и объектами недвижимости, контроль статусов сделок (в ожидании, завершена, отменена), а также ведение полной истории всех операций, что позволяет анализировать эффективность работы агентства.

4 Организация аукционных торгов. Система поддерживает создание аукционов для объектов недвижимости, что расширяет возможности продажи недвижимости.

Управление агентством недвижимости является активно развивающейся областью, которая находится на стыке информационных технологий, менеджмента и маркетинга. В настоящее время существует множество подходов и инструментов для решения задач управления недвижимостью.

Предлагаемая система управления компанией решает эти задачи, предоставляя решение для автоматизации процессов управления каталогом недвижимости различных типов, ведения клиентской базы, контроля сделок на всех этапах их жизненного цикла, организации аукционных торгов.

1 ОБЗОР ЛИТЕРАТУРЫ

1.1 Обзор аналогов

На рынке представлено множество систем управления агентствами недвижимости и CRM-решений для работы с недвижимостью. Рассмотрим основные решения и их особенности.

Bitrix24 является популярной CRM-системой с модулем управления недвижимостью. Основные функции включают управление клиентской базой, каталог объектов недвижимости, отслеживание сделок и воронку продаж, интеграции с телефонией и почтой, отчетность и аналитику, а также мобильные приложения. Преимущества Bitrix24 заключаются в широком функционале, облачном хранении данных, интеграциях с внешними сервисами и масштабируемости. Недостатки включают зависимость от интернета, высокую стоимость лицензий для расширенного функционала, избыточность функций для небольших агентств и сложность настройки под специфику работы агентства. Для нашего проекта учтена важность управления клиентами и сделками, но упрощен интерфейс и обеспечена работа без интернета.

amoCRM представляет собой облачную CRM-систему с возможностью настройки под недвижимость. Основные функции включают управление клиентами и лидами, воронку продаж, автоматизацию процессов, интеграции с сайтами и социальными сетями, а также аналитику и отчеты. Преимущества amoCRM заключаются в удобном интерфейсе, гибкой настройке воронок продаж и хорошей интеграции с веб-сайтами. Недостатки включают зависимость от интернета, ограничения в бесплатной версии, недостаточную специализацию для недвижимости и отсутствие встроенной поддержки аукционов. Для нашего проекта учтена важность управления сделками и клиентами, добавлена специализация для недвижимости и поддержка аукционов.

Excel/Google Sheets являются простыми инструментами для ведения учета недвижимости. Основные функции включают таблицы для объектов и клиентов, базовые формулы для расчетов, фильтрацию и сортировку данных, а также возможность совместной работы. Преимущества Excel/Google Sheets заключаются в простоте, доступности, отсутствии необходимости обучения и низкой стоимости. Недостатки включают отсутствие автоматизации процессов, риск потери данных, отсутствие валидации данных, невозможность эффективного управления связями между объектами, клиентами и сделками, а также отсутствие поддержки аукционов. Для нашего проекта учтена простота использования, добавлены автоматизация, валидация данных и специализированные функции для недвижимости.

Salesforce Real Estate Cloud представляет собой корпоративное решение для управления недвижимостью на базе платформы Salesforce. Основные функции включают управление портфелем недвижимости, клиентскими отношениями,

сделками и контрактами, аналитику и прогнозирование, а также интеграции с внешними системами. Преимущества Salesforce заключаются в мощном функционале, масштабируемости и профессиональных инструментах аналитики. Недостатки включают очень высокую стоимость, сложность настройки и освоения, избыточность для небольших и средних агентств и зависимость от интернета. Для нашего проекта учтена важность управления недвижимостью и сделками, упрощен интерфейс.

Анализ существующих решений показывает, что большинство систем ориентированы на крупные агентства и имеют избыточный функционал для небольших и средних компаний. Общие недостатки включают зависимость от интернета у облачных решений, высокую стоимость лицензий для полнофункциональных версий, недостаточную поддержку аукционных торгов, необходимость длительного обучения персонала.

Что будет учтено и улучшено в нашем проекте. Эффективное управление клиентской базой, где осуществляется регистрация клиентов с валидацией контактных данных, поиск по имени или телефону, отслеживание истории взаимодействия и автоматическое связывание клиентов со сделками и объектами. Контроль сделок и их статусов, включающий создание сделок между клиентами и объектами недвижимости, отслеживание статусов (в ожидании, завершена, отменена), ведение полной истории операций и автоматическое обновление статусов объектов при завершении сделок. Простота использования, обеспечивается нативным desktop-приложением с понятным интерфейсом, не требующим длительного обучения, интуитивной навигацией и специализированными формами для работы с недвижимостью. Организация аукционных торгов, где обеспечивается создание аукционов для объектов недвижимости, управление ставками участников, отслеживание текущей максимальной ставки, автоматическое определение победителя и контроль статусов аукционов. Работа без интернета обеспечивается локальным хранением данных и независимостью от внешних сервисов.

1.2 Обзор информационных технологий

В проекте применяется язык программирования C++ стандарта C++23. К преимуществам C++ относятся отличная производительность выполнения программ, полный контроль над системными ресурсами, обширная стандартная библиотека STL, возможности объектно-ориентированной парадигмы, умные указатели (shared_ptr, unique_ptr) для безопасного управления памятью, шаблоны для реализации обобщенного кода и механизм исключений для надежной обработки ошибочных ситуаций.

Применяется фреймворк Qt6, представляющий собой кроссплатформенный инструментарий для создания приложений с графическим пользовательским интерфейсом. Ключевые компоненты включают QtCore для

базовой функциональности, работы со строками и датами, QtWidgets для создания виджетов desktop-приложений, механизм сигналов и слотов для реализации событийной модели, классы QString для удобной работы со строками. Преимущества Qt6 состоят в кроссплатформенности для операционных систем Windows, Linux и macOS, богатом наборе готовых виджетов, высокой производительности интерфейса, зрелости платформы и качественной документации, а также возможности создания нативных приложений. Выбор фреймворка обоснован необходимостью разработки desktop-приложения с графическим интерфейсом, требованием кроссплатформенности и широкими возможностями для работы с данными и построения пользовательского интерфейса.

Используется текстовый формат с разделителями для хранения данных. Структура формата предусматривает разделители секций в виде отдельных файлов: properties.txt для объектов недвижимости, clients.txt для клиентов, transactions.txt для сделок и auctions.txt для аукционов, маркеры типов недвижимости в виде префиксов [APARTMENT], [HOUSE], [COMMERCIAL], разделители полей в виде символа '|', построчную запись данных, где каждое поле отделено разделителем. Выбор формата обоснован необходимостью хранения информации об объектах недвижимости, клиентах, сделках и аукционах в структурированном виде, удобством отладки и ручного редактирования при необходимости, а также простотой реализации парсера на основе стандартных потоков ввода-вывода C++.

CMake применяется в качестве системы сборки для кроссплатформенной компиляции проекта. Преимущества CMake состоят в поддержке различных компиляторов и платформ, интеграции с Qt6 через find_package и удобстве управления зависимостями, а также автоматической генерации файлов сборки для различных сред разработки. Обоснование заключается в необходимости кроссплатформенной сборки проекта и интеграции с Qt6.

Объектно-ориентированное программирование включает применение классов и наследования для моделирования предметной области, где Property и его производные классы (Apartment, House, CommercialProperty) реализуют полиморфизм, инкапсуляцию данных и методов. Паттерны проектирования включают использование умных указателей для управления памятью, контейнеров STL для хранения коллекций объектов, исключений для обработки ошибок и потоковой работы с файлами через стандартные потоки ввода-вывода (ifstream, ofstream). Обоснование заключается в обеспечении надежности, расширяемости и поддерживаемости кода.

1.3 Постановка задачи

Целью проекта является создание автоматизированной информационной системы управления агентством недвижимости на основе Qt6 и C++,

обеспечивающей эффективное управление каталогом объектов недвижимости, ведение клиентской базы, контроль сделок и организацию аукционных торгов.

Основные задачи проекта:

1 Разработать модель данных для представления объектов недвижимости, клиентов, сделок и аукционов. Реализовать базовый класс Property и производные классы (Apartment, House, CommercialProperty) для различных типов недвижимости, класс Client для представления клиентов агентства, класс Transaction для сделок купли-продажи с различными статусами и классы Auction и Bid для организации аукционных торгов.

2 Реализовать систему управления объектами недвижимости. Обеспечить добавление, редактирование и удаление объектов различных типов, поиск объектов по адресу (город, улица, дом), отслеживание статуса доступности объектов и автоматическое обновление статуса при завершении сделок.

3 Реализовать систему управления клиентской базой. Обеспечить добавление новых клиентов с валидацией контактных данных редактирование и удаление информации о клиентах, поиск клиентов по имени или номеру телефона, отслеживание истории взаимодействия с недвижимостью.

4 Реализовать систему управления аукционными торгами. Разработать механизм создания аукционов для объектов недвижимости, управление ставками участников аукциона, отслеживание истории ставок, автоматическое определение победителя аукциона, контроль статусов аукционов и поддержку функции выкупа.

5 Реализовать систему управления сделками. Обеспечить создание сделок между клиентами и объектами недвижимости, управление статусами сделок отслеживание финальной цены сделки и даты совершения, связывание сделок с объектами и клиентами, а также проверку доступности объектов для новых сделок.

6 Разработать графический пользовательский интерфейс. Создать главное окно приложения с навигацией по разделам системы, интерфейс для управления объектами недвижимости интерфейс для управления клиентами интерфейс для управления сделками, интерфейс для управления аукционами отображение статистики и аналитики на дашборде, а также поиск данных во всех разделах.

7 Реализовать систему хранения данных. Обеспечить сохранение информации об объектах недвижимости, клиентах, сделках и аукционах в текстовые файлы, автоматическую загрузку данных при запуске приложения.

8 Обеспечить валидацию данных и обработку ошибок. Реализовать проверку корректности вводимых данных.

В результате выполнения проекта создана информационная система управления агентством недвижимости, обеспечивающая комплексное управление каталогом объектов различных типов, ведение клиентской базы, контроль сделок и организацию аукционных торгов. Система имеет интуитивный графический интерфейс, обеспечивает надежное хранение данных.

2 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

Приложение для управления агентством недвижимости обеспечивает ведение каталога объектов недвижимости, управление клиентской базой, контроль сделок, организацию аукционных торгов, а также отображение статистики. Основные функции системы направлены на автоматизацию процессов управления объектами различных типов, ведение базы клиентов, отслеживание сделок и эффективное использование механизмов продажи недвижимости. Программа реализует следующие ключевые возможности:

– создание, редактирование и удаление объектов недвижимости различных типов (квартиры, дома, коммерческая недвижимость) с отслеживанием их статуса доступности, цены, площади и специфических характеристик для каждого типа;

– управление клиентской базой с возможностью добавления, изменения и удаления информации о клиентах, валидацией контактных данных и отслеживанием даты добавления;

– создание сделок между клиентами и объектами недвижимости с указанием финальной цены, статуса сделки и даты совершения для структурирования процесса продажи;

– управление статусами сделок (в ожидании, завершена, отменена) с автоматическим обновлением статуса доступности объектов недвижимости при изменении состояния сделок;

– контроль доступности объектов недвижимости с проверкой отсутствия активных сделок для объекта, предотвращением создания конфликтующих сделок и автоматическим обновлением статуса при изменении состояния сделок;

– организация аукционных торгов с созданием аукционов для объектов недвижимости, управлением ставками участников, отслеживанием текущей максимальной ставки, автоматическим завершением при достижении цены выкупа и автоматическим созданием сделки при завершении аукциона;

– автоматическое сохранение данных в текстовом формате при закрытии приложения и загрузка данных при запуске для обеспечения сохранности информации;

– отображение статистики по агентству, включающей общее количество объектов недвижимости, клиентов, сделок, аукционов и количество доступных объектов;

– поиск объектов недвижимости по идентификатору и адресу, клиентов по идентификатору и имени, сделок и аукционов по идентификатору для быстрого доступа к информации;

– управление несколькими компаниями с возможностью создания, переключения и удаления компаний для работы с различными организациями;

– использование пользовательских исключений для обработки ошибок, связанных с операциями над сотрудниками, проектами и задачами, что обеспечивает надежность работы системы.

Для более детального понимания архитектуры проекта используется диаграмма классов (Приложение А), которая визуализирует структуру системы и взаимосвязи между основными компонентами.

2.1 Описание функции управления объектами недвижимости

Функция обеспечивает управление каталогом объектов недвижимости агентства. Реализованы операции добавления, редактирования, удаления и поиска объектов. Система поддерживает три типа объектов недвижимости (Apartment, House, CommercialProperty), каждый со специфическими характеристиками. Для квартир отслеживается количество комнат, этаж, наличие балкона и лифта. Для домов учитывается количество этажей, количество комнат, площадь участка, наличие гаража и сада. Для коммерческой недвижимости фиксируется тип коммерческого использования, наличие парковки, количество парковочных мест и видимость с улицы. Для каждого объекта отслеживается статус доступности, цена, площадь, адрес (город, улица, дом) и описание. При добавлении проверяется уникальность идентификатора объекта и валидность всех параметров (цена должна быть от 10000 до 1000000000 рублей, площадь должна быть положительным числом, адрес не должен быть пустым). Реализован поиск по идентификатору (точное совпадение, если введены только цифры длиной от 6 до 8 символов) и по адресу (частичное совпадение по городу, улице или дому). При выборе объекта в таблице отображается детальная информация об объекте, включая все характеристики в зависимости от типа, а также список всех связанных сделок с указанием клиента, цены и статуса. Система автоматически обновляет статус доступности объектов при завершении или отмене сделок, обеспечивая актуальность информации о доступных объектах.

2.2 Описание функции управления клиентской базой

Функция обеспечивает управление клиентской базой агентства. Реализованы операции добавления, редактирования, удаления и поиска клиентов. Каждый клиент содержит информацию об идентификаторе, имени, телефоне, email и дате добавления в систему. Система автоматически устанавливает дату добавления клиента при создании записи в формате ISO 8601. Система выполняет валидацию контактных данных: проверку формата телефона и проверку формата email с использованием регулярных выражений. При добавлении проверяется уникальность идентификатора клиента и корректность всех полей. Реализован поиск клиентов по идентификатору (точное совпадение, если введены только цифры длиной от 6 до 8 символов) и по имени (частичное

совпадение, поиск по подстроке). При выборе клиента в таблице отображается детальная информация о клиенте, включая все контактные данные и дату добавления, а также список всех связанных сделок с указанием объекта недвижимости, цены и статуса. Система автоматически связывает клиентов со сделками и ставками в аукционах, что позволяет анализировать активность клиентов и историю их взаимодействия с агентством.

2.3 Описание функции управления сделками

Функция обеспечивает управление сделками. Реализованы операции создания, редактирования, удаления и поиска сделок. При создании сделки система требует выбора объекта недвижимости из списка доступных объектов и клиента из списка клиентов. Система автоматически устанавливает дату создания сделки в формате ISO 8601 и связывает сделку с объектом недвижимости и клиентом. Каждая сделка имеет уникальный идентификатор (от 6 до 8 цифр). При создании или редактировании сделки выполняется валидация: проверка доступности объекта недвижимости, проверка цены. При завершении сделки со статусом система автоматически обновляет статус доступности объекта недвижимости. Реализован поиск сделок по идентификатору (точное совпадение, если введены только цифры длиной от 6 до 8 символов). При выборе сделки в таблице отображается детальная информация о сделке, включая данные об объекте недвижимости и клиенте. Система отслеживает финальную цену сделки, которая может отличаться от цены объекта, и позволяет добавлять примечания к сделкам.

2.4 Описание функции управления аукционными торгами

Функция обеспечивает организацию и управление аукционными торгами для объектов недвижимости. Реализованы операции создания, просмотра, удаления и поиска аукционов. При создании аукциона система требует выбора объекта недвижимости из списка доступных объектов. Система автоматически устанавливает дату создания аукциона в формате ISO 8601 и начальный статус «Активен». Цена выкупа автоматически рассчитывается как начальная цена, умноженная на 1.7. Для каждого аукциона система управляет ставками участников: клиенты могут делать ставки через диалоговое окно просмотра аукциона. При добавлении ставки выполняется проверка, что ставка превышает текущую максимальную ставку или равна начальной цене, если ставок еще нет. Если ставка достигает или превышает цену выкупа, аукцион автоматически завершается и создается сделка со статусом «Завершена» для клиента, сделавшего ставку. Система отслеживает историю всех ставок с указанием клиента, суммы ставки и времени размещения. При просмотре аукциона отображается список всех ставок, отсортированный по убыванию суммы, и

информация об объекте недвижимости. Аукционы могут быть завершены вручную через кнопку «Завершить аукцион» в диалоговом окне. При ручном завершении аукциона создается сделка для клиента с максимальной ставкой. Если на аукционе нет ставок, аукцион может быть отменен. Система обеспечивает связь между аукционами и объектами недвижимости. При создании сделки из аукциона система автоматически генерирует уникальный идентификатор сделки. Реализован поиск аукционов по идентификатору (точное совпадение, если введены только цифры длиной от 6 до 8 символов).

2.5 Описание функции поиска данных

Функция обеспечивает быстрый поиск данных во всех разделах системы. Для объектов недвижимости реализован поиск по идентификатору (точное совпадение, если поисковый запрос содержит только цифры длиной от 6 до 8 символов) и по адресу (частичное совпадение, если поисковый запрос содержит не только цифры). Поиск по адресу выполняется по городу, улице или дому с использованием частичного совпадения (без учета регистра), что позволяет находить объекты даже при неполном указании адреса. Для клиентов реализован поиск по идентификатору (точное совпадение, если поисковый запрос содержит только цифры длиной от 6 до 8 символов) и по имени (частичное совпадение, если поисковый запрос содержит не только цифры). Поиск по имени выполняется с использованием частичного совпадения, что позволяет находить клиентов по части имени. Для сделок реализован поиск по идентификатору (точное совпадение, если поисковый запрос содержит только цифры длиной от 6 до 8 символов). Для аукционов реализован поиск по идентификатору (точное совпадение, если поисковый запрос содержит только цифры длиной от 6 до 8 символов). Все результаты поиска отображаются в таблицах с возможностью просмотра детальной информации при выборе записи. Система обеспечивает обновление результатов поиска при нажатии кнопки «Найти» или при очистке поля поиска (отображаются все записи).

2.6 Описание функции контроля доступности объектов

Функция обеспечивает контроль доступности объектов недвижимости при создании сделок и аукционов. При создании сделки со статусом «в ожидании» или «завершена» система проверяет, что объект недвижимости не участвует в других активных сделках, что предотвращает создание конфликтующих сделок. При создании аукциона система проверяет, что объект недвижимости не участвует в активных сделках и не участвует в других активных аукционах, что гарантирует корректность организации торгов. При завершении сделки со статусом «завершена» система автоматически обновляет статус доступности объекта на «недоступен», что предотвращает создание новых сделок для уже

проданного объекта. При отмене сделки со статусом «отменена» система автоматически возвращает объект в статус «доступен». При удалении сделки система проверяет наличие других активных сделок для объекта и при отсутствии таковых возвращает объект в статус "доступен". При завершении аукциона система также обновляет статус доступности объекта на "недоступен" через создание сделки. Система отслеживает текущий статус каждого объекта и обеспечивает автоматическую синхронизацию статусов при изменении состояния связанных сделок и аукционов, что гарантирует актуальность информации о доступных объектах

2.7 Описание функции сохранения и загрузки данных

Функция обеспечивает сохранение и загрузку данных агентства в текстовом формате с разделителями. При сохранении данные записываются в отдельные файлы для объектов недвижимости, клиентов, сделок и аукционов с использованием разделителя полей '|'. Маркеры типов недвижимости записываются как [APARTMENT], [HOUSE], [COMMERCIAL]. Каждое поле данных записывается в формате с разделителем, где поля отделены символом '|'. Для объектов недвижимости сохраняется информация о типе, идентификаторе, адресе (город, улица, дом), цене, площади, описании, статусе доступности (1 для доступных, 0 для недоступных) и специфических полях для каждого типа (для квартир: количество комнат, этаж, наличие балкона и лифта; для домов: количество этажей, количество комнат, площадь участка, наличие гаража и сада; для коммерческой недвижимости: тип бизнеса, наличие парковки, количество парковочных мест, видимость с улицы). Для клиентов сохраняется идентификатор, имя, телефон, email и дата добавления. Для сделок сохраняется идентификатор, идентификатор объекта недвижимости, идентификатор клиента, дата, финальная цена, статус и примечания. Для аукционов сохраняется идентификатор, идентификатор объекта недвижимости, адрес объекта, стартовая цена, цена выкупа, статус, даты создания и завершения, а также все ставки, где каждая ставка записывается на отдельной строке с префиксом BID:. При загрузке система последовательно читает данные из файлов, парсит разделители и восстанавливает структуру данных. Реализована обработка ошибок при чтении файлов: если файл не существует, система продолжает работу с пустыми данными. Система автоматически сохраняет данные при закрытии приложения через метод destroyInstance класса EstateAgency и загружает их при запуске через метод loadAllData, обеспечивая сохранность информации.

2.8 Описание функции отображения статистики

Функция обеспечивает отображение статистической информации об агентстве недвижимости на дашборде. Система рассчитывает и отображает

общее количество объектов недвижимости, общее количество клиентов, общее количество сделок, общее количество аукционов и количество доступных объектов недвижимости. Статистика отображается на дашборде в виде карточек с крупными цифрами и цветовой индикацией (разные цвета для разных метрик). Дашборд также содержит кнопки для сохранения данных, загрузки данных и обновления всех виджетов. Система обеспечивает обновление статистики в реальном времени при изменении данных через метод updateStats виджета DashboardWidget, что позволяет отслеживать актуальное состояние агентства. Статистика обновляется автоматически при переключении на вкладку дашборда и при выполнении операций с данными через сигналы dataChanged от виджетов.

2.9 Описание функции обработки ошибок и исключений

Функция обеспечивает надежную обработку ошибок и исключительных ситуаций на всех уровнях системы. Система использует специализированные классы исключений для различных компонентов: PropertyManagerException для ошибок управления недвижимостью, ClientManagerException для ошибок управления клиентами, TransactionManagerException для ошибок управления сделками, AuctionManagerException для ошибок управления аукционами и FileManagerException для ошибок работы с файлами. При возникновении ошибок система предоставляет информативные сообщения пользователю через диалоговые окна QMessageBox с указанием типа ошибки и деталей проблемы. Система обрабатывает стандартные исключения C++ (std::invalid_argument, std::bad_alloc, std::filesystem_error и другие) и преобразует их в понятные сообщения для пользователя. При ошибках валидации данных система предотвращает создание некорректных записей и информирует пользователя о причинах ошибки через предупреждающие диалоги. При ошибках работы с файлами система обеспечивает корректную обработку ситуаций отсутствия файлов или ошибок чтения, позволяя приложению продолжать работу с пустыми данными. При инициализации приложения все исключения обрабатываются с отображением критических сообщений об ошибках. Это обеспечивает надежность работы системы и предотвращает потерю данных при возникновении исключительных ситуаций.

2.10 Описание функции автоматического обновления статусов

Функция обеспечивает автоматическое обновление статусов объектов недвижимости при изменении состояния связанных сделок и аукционов. При завершении сделки со статусом "завершена" система автоматически обновляет статус доступности объекта недвижимости на «недоступен» через метод setAvailable(false), что предотвращает создание новых сделок для уже проданного объекта. При отмене сделки со статусом «отменена» система

автоматически возвращает объект в статус «доступен» через метод setAvailable(true), если объект не участвует в других активных сделках или аукционах (проверка выполняется через метод hasActiveTransactions). При удалении сделки система проверяет наличие других активных сделок для объекта и при отсутствии таковых обновляет статус объекта на "доступен". При завершении аукциона (автоматически при достижении цены выкупа или вручную) система создает сделку со статусом «завершена», что автоматически обновляет статус доступности объекта на «недоступен». Система проверяет состояние всех связанных сделок и аукционов перед изменением статуса объекта, что гарантирует корректность обновления. При создании новой сделки со статусом «в ожидании» или «завершена» система также обновляет статус объекта на «недоступен». Это обеспечивает консистентность данных в системе и автоматически поддерживает актуальность информации о доступных объектах.

3 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

3.1 Описание программных модулей

1 Класс `Property`:

Базовый абстрактный класс для объектов недвижимости: идентификатор, адрес, цена, площадь, описание и доступность.

Поля:

- `std::string id`: идентификатор объекта недвижимости;
- `std::string city`: город;
- `std::string street`: улица;
- `std::string house`: номер дома;
- `double price`: цена объекта;
- `double area`: площадь объекта;
- `std::string description`: описание объекта;
- `bool isAvailable`: флаг доступности объекта.

Методы:

- `Property(const std::string& id, const std::string& city, const std::string& street, const std::string& house, double price, double area, const std::string& description)`: конструктор базового класса;
- `virtual ~Property()`: виртуальный деструктор;
- `virtual std::string getType() const = 0`: получить тип недвижимости (чисто виртуальный);
- `virtual void displayInfo() const = 0`: вывести информацию об объекте (чисто виртуальный);
- `virtual std::string toFileString() const = 0`: сериализация в строку для файла (чисто виртуальный);
- `virtual Property* clone() const = 0`: клонирование объекта (чисто виртуальный);
- `bool operator==(const Property& other) const`: проверка равенства по идентификатору;
- `std::partial_ordering operator<=>(const Property& other) const`: сравнение объектов;
- `std::string getId() const`: получить идентификатор;
- `std::string getCity() const`: получить город;
- `std::string getStreet() const`: получить улицу;
- `std::string getHouse() const`: получить номер дома;

– `std::string getAddress() const`: получить полный адрес;
– `double getPrice() const`: получить цену;
– `double getArea() const`: получить площадь;
– `std::string getDescription() const`: получить описание;
– `bool getIsAvailable() const`: проверить доступность;
– `void setPrice(double newPrice)`: установить цену;
– `void setArea(double newArea)`: установить площадь;
– `void setAddress(const std::string& city, const std::string& street, const std::string& house)`: установить адрес;
– `void setDescription(std::string_view newDesc)`: установить описание;
– `void setAvailable(bool available)`: установить доступность;
– `static bool validatePrice(double price)`: валидация цены;
– `static bool validateArea(double area)`: валидация площади;
– `static bool validateId(std::string_view id)`: валидация идентификатора;
– `static bool validateAddressPart(std::string_view part)`: валидация части адреса;
– `friend std::ostream& operator<<(std::ostream& os, const Property& prop)`: вывод объекта в поток.

2 Класс `Apartment`:

Класс квартиры, наследуется от Property: количество комнат, этаж, наличие балкона и лифта.

Поля:

– `int rooms`: количество комнат;
– `int floor`: этаж;
– `bool hasBalcony`: наличие балкона;
– `bool hasElevator`: наличие лифта.

Методы:

– `explicit Apartment(const ApartmentParams& params)`: конструктор из параметров;
– `std::string getType() const override`: получить тип ("Apartment");

– `void displayInfo() const override`: вывести информацию о квартире;
– `std::string toFileString() const override`: сериализация в строку для файла;
– `Property* clone() const override`: клонирование квартиры;
– `int getRooms() const`: получить количество комнат;
– `int getFloor() const`: получить этаж;
– `bool getHasBalcony() const`: проверить наличие балкона;
– `bool getHasElevator() const`: проверить наличие лифта;
– `void setRooms(int rooms)`: установить количество комнат;
– `void setFloor(int floor)`: установить этаж.

3 Класс `House`:

Класс дома, наследуется от Property: количество этажей, комнат, площадь участка, наличие гаража и сада.

Поля:

– `int floors`: количество этажей;
– `int rooms`: количество комнат;
– `double landArea`: площадь участка;
– `bool hasGarage`: наличие гаража;
– `bool hasGarden`: наличие сада.

Методы:

– `explicit House(const HouseParams& params)`: конструктор из параметров;
– `std::string getType() const override`: получить тип ("House");
– `void displayInfo() const override`: вывести информацию о доме;
– `std::string toFileString() const override`: сериализация в строку для файла;
– `Property* clone() const override`: клонирование дома;
– `int getFloors() const`: получить количество этажей;
– `int getRooms() const`: получить количество комнат;
– `double getLandArea() const`: получить площадь участка;
– `bool getHasGarage() const`: проверить наличие гаража;
– `bool getHasGarden() const`: проверить наличие сада;
– `void setFloors(int floors)`: установить количество этажей;
– `void setRooms(int rooms)`: установить количество комнат;

– `void setLandArea(double landArea)`: установить площадь участка.

4 Класс `CommercialProperty`:

Класс коммерческой недвижимости, наследуется от Property: тип бизнеса, наличие парковки, количество парковочных мест, видимость с улицы.

Поля:

– `std::string businessType`: тип бизнеса;
– `bool hasParking`: наличие парковки;
– `int parkingSpaces`: количество парковочных мест;
– `bool isVisibleFromStreet`: видимость с улицы.

Методы:

– `explicit CommercialProperty(const CommercialPropertyParams& params)`: конструктор из параметров;
– `std::string getType() const override`: получить тип ("Commercial");
– `void displayInfo() const override`: вывести информацию о коммерческой недвижимости;
– `std::string toFileString() const override`: сериализация в строку для файла;
– `Property* clone() const override`: клонирование коммерческой недвижимости;
– `std::string getBusinessType() const`: получить тип бизнеса;
– `bool getHasParking() const`: проверить наличие парковки;
– `int getParkingSpaces() const`: получить количество парковочных мест;
– `bool getIsVisibleFromStreet() const`: проверить видимость с улицы;
– `void setBusinessType(std::string_view type)`: установить тип бизнеса;
– `void setParkingSpaces(int spaces)`: установить количество парковочных мест.

5 Класс `Client`:

Класс клиента: идентификатор, имя, телефон, email и дата регистрации.

Поля:

– `std::string id`: идентификатор клиента;
– `std::string name`: имя клиента;
– `std::string phone`: телефон клиента;
– `std::string email`: email клиента;

– std::string registrationDate: дата регистрации в формате ISO 8601.

Методы:

– Client(const std::string& id, const std::string& name, const std::string& phone, const std::string& email): конструктор клиента;
– bool operator==(const Client& other) const: проверка равенства по идентификатору;
– std::strong_ordering operator<=>(const Client& other) const: сравнение клиентов;
– std::string getId() const: получить идентификатор;
– std::string getName() const: получить имя;
– std::string getPhone() const: получить телефон;
– std::string getEmail() const: получить email;
– std::string getRegistrationDate() const: получить дату регистрации;
– void setName(std::string_view name): установить имя;
– void setPhone(std::string_view phone): установить телефон;
– void setEmail(std::string_view email): установить email;
– static bool validateId(std::string_view id): валидация идентификатора;
– static bool validatePhone(std::string_view phone): валидация телефона;
– static bool validateEmail(std::string_view email): валидация email;
– std::string toString() const: преобразование в строку;
– friend std::ostream& operator<<(std::ostream& os, const Client& client): вывод клиента в поток.

6 Класс Transaction:

Класс сделки: идентификатор, идентификатор объекта недвижимости, идентификатор клиента, дата, финальная цена, статус и примечания.

Поля:

– std::string id: идентификатор сделки;
– std::string propertyId: идентификатор объекта недвижимости;
– std::string clientId: идентификатор клиента;
– std::string date: дата сделки в формате ISO 8601;
– double finalPrice: финальная цена сделки;

– `std::string status`: статус сделки (pending, completed, cancelled);
– `std::string notes`: примечания к сделке.

Методы:

– `Transaction(const std::string& id, const std::string& propertyId, const std::string& clientId, double finalPrice, const std::string& status = "pending", const std::string& notes = "")`: конструктор сделки;
– `bool operator==(const Transaction& other) const`: проверка равенства по идентификатору;
– `std::strong_ordering operator<=>(const Transaction& other) const`: сравнение сделок;
– `std::string getId() const`: получить идентификатор;
– `std::string getPropertyId() const`: получить идентификатор объекта недвижимости;
– `std::string getClientId() const`: получить идентификатор клиента;
– `std::string getDate() const`: получить дату сделки;
– `double getFinalPrice() const`: получить финальную цену;
– `std::string getStatus() const`: получить статус;
– `std::string getNotes() const`: получить примечания;
– `void setStatus(std::string_view status)`: установить статус;
– `void setFinalPrice(double price)`: установить финальную цену;
– `void setNotes(std::string_view notes)`: установить примечания;
– `static bool validateId(std::string_view id)`: валидация идентификатора;
– `std::string toString() const`: преобразование в строку;
– `std::string toFileString() const`: сериализация в строку для файла;
– `friend std::ostream& operator<<(std::ostream& os, const Transaction& trans)`: вывод сделки в поток.

7 Класс `Auction`:

Класс аукциона: идентификатор, идентификатор объекта недвижимости, адрес объекта, начальная цена, цена выкупа, список ставок, статус и даты создания и завершения.

Поля:

- `std::string id`: идентификатор аукциона;
- `std::string propertyId`: идентификатор объекта недвижимости;
- `std::string propertyAddress`: адрес объекта недвижимости;
- `double startingPrice`: начальная цена;
- `double buyoutPrice`: цена выкупа;
- `std::vector<std::shared_ptr<Bid>> bids`: список ставок;
- `std::string status`: статус аукциона (active, completed, cancelled);
- `std::string createdAt`: дата создания в формате ISO 8601;
- `std::string completedAt`: дата завершения в формате ISO 8601.

Методы:

- `Auction(const std::string& id, const std::string& propertyId, const std::string& propertyAddress, double startingPrice)`: конструктор аукциона;
- `bool operator==(const Auction& other) const`: проверка равенства по идентификатору;
- `std::strong_ordering operator<=>(const Auction& other) const`: сравнение аукционов;
- `bool addBid(std::shared_ptr<Bid> bid)`: добавить ставку с проверкой;
- `void addBidDirect(std::shared_ptr<Bid> bid)`: добавить ставку напрямую;
- `double getCurrentHighestBid() const`: получить текущую максимальную ставку;
- `const Bid* getHighestBid() const`: получить указатель на максимальную ставку;
- `void complete()`: завершить аукцион;
- `void cancel()`: отменить аукцион;
- `std::string getId() const`: получить идентификатор;
- `std::string getPropertyId() const`: получить идентификатор объекта недвижимости;
- `std::string getPropertyAddress() const`: получить адрес объекта;

– `double getStartingPrice() const`: получить начальную цену;
– `double getBuyoutPrice() const`: получить цену выкупа;
– `std::vector<std::shared_ptr<Bid>> getBids() const`: получить список ставок;
– `std::string getStatus() const`: получить статус;
– `std::string getCreatedAt() const`: получить дату создания;
– `std::string getCompletedAt() const`: получить дату завершения;
– `bool isActive() const`: проверить, активен ли аукцион;
– `bool isCompleted() const`: проверить, завершен ли аукцион;
– `bool wasBuyout() const`: проверить, был ли выкуп;
– `std::string toString() const`: преобразование в строку;
– `std::string toFileString() const`: сериализация в строку для файла;
– `friend std::ostream& operator<<(std::ostream& os, const Auction& auction)`: вывод аукциона в поток.

8 Класс `Bid`:

Класс ставки в аукционе: идентификатор клиента, имя клиента, сумма ставки и временная метка.

Поля:

– `std::string clientId`: идентификатор клиента;
– `std::string clientName`: имя клиента;
– `double amount`: сумма ставки;
– `std::string timestamp`: временная метка в формате ISO 8601.

Методы:

– `Bid(const std::string& clientId, const std::string& clientName, double amount)`: конструктор ставки;
– `bool operator==(const Bid& other) const`: проверка равенства ставок;
– `std::partial_ordering operator<=>(const Bid& other) const`: сравнение ставок;
– `std::string getClientId() const`: получить идентификатор клиента;
– `std::string getClientName() const`: получить имя клиента;

– double getAmount() const: получить сумму ставки;
– std::string getTimestamp() const: получить временную метку;
– std::string toString() const: преобразование в строку;
– std::string toFileString() const: сериализация в строку для файла;
– friend std::ostream& operator<<(std::ostream& os, const Bid& bid): вывод ставки в поток.

9 Структура PropertyBaseParams:

Базовые параметры объекта недвижимости для создания.

Поля:

– std::string id: идентификатор объекта;
– std::string city: город;
– std::string street: улица;
– std::string house: номер дома;
– double price: цена;
– double area: площадь;
– std::string description: описание.

Методы:

– (нет): структура только для хранения параметров.

10 Структура ApartmentParams:

Параметры квартиры для создания, включает базовые параметры и специфичные для квартиры.

Поля:

– PropertyBaseParams base: базовые параметры объекта недвижимости;
– int rooms: количество комнат;
– int floor: этаж;
– bool hasBalcony: наличие балкона;
– bool hasElevator: наличие лифта.

Методы:

– (нет): структура только для хранения параметров.

11 Структура HouseParams:

Параметры дома для создания, включает базовые параметры и специфичные для дома.

Поля:

– PropertyBaseParams base: базовые параметры объекта недвижимости;
– int floors: количество этажей;
– int rooms: количество комнат;

– `double landArea`: площадь участка;
– `bool hasGarage`: наличие гаража;
– `bool hasGarden`: наличие сада.

Методы:

– `(нет)`: структура только для хранения параметров.

12 Структура `CommercialPropertyParams`:

Параметры коммерческой недвижимости для создания, включает базовые параметры и специфичные для коммерческой недвижимости.

Поля:

– `PropertyBaseParams base`: базовые параметры объекта недвижимости;
– `std::string businessType`: тип бизнеса;
– `bool hasParking`: наличие парковки;
– `int parkingSpaces`: количество парковочных мест;
– `bool isVisibleFromStreet`: видимость с улицы.

Методы:

– `(нет)`: структура только для хранения параметров.

13 Класс `PropertyManager`:

Менеджер объектов недвижимости: управление коллекцией объектов, добавление, удаление, поиск и фильтрация.

Поля:

– `std::vector<std::unique_ptr<Property>> properties`: коллекция объектов недвижимости.

Методы:

– `PropertyManager()`: конструктор менеджера;
– `void addProperty(std::unique_ptr<Property> property)`: добавить объект недвижимости;
– `void addApartment(const ApartmentParams& params)`: добавить квартиру;
– `void addHouse(const HouseParams& params)`: добавить дом;
– `void addCommercialProperty(const CommercialPropertyParams& params)`: добавить коммерческую недвижимость;
– `bool removeProperty(const std::string& id)`: удалить объект недвижимости по идентификатору;
– `Property* findProperty(const std::string& id) const`: найти объект по идентификатору;
– `std::vector<Property*> getAllProperties() const`: получить все объекты недвижимости;

– `std::vector<Property*> getAvailableProperties() const`: получить доступные объекты недвижимости;
– `std::vector<Property*> searchByPriceRange(double minPrice, double maxPrice) const`: поиск по диапазону цен;
– `std::vector<Property*> searchByAddress(const std::string& city, const std::string& street = "", const std::string& house = "") const`: поиск по адресу;
– `const std::vector<std::unique_ptr<Property>>& getProperties() const`: получить коллекцию объектов для FileManager;
– `void setProperties(std::vector<std::unique_ptr<Property>>&& props)`: установить коллекцию объектов из FileManager;
– `size_t getCount() const`: получить количество объектов.

14 Класс `ClientManager`:

Менеджер клиентов: управление коллекцией клиентов, добавление, удаление и поиск.

Поля:

– `std::vector<std::shared_ptr<Client>> clients`: коллекция клиентов.

Методы:

– `ClientManager()`: конструктор менеджера;
– `void addClient(std::shared_ptr<Client> client)`: добавить клиента;
– `bool removeClient(const std::string& id)`: удалить клиента по идентификатору;
– `Client* findClient(const std::string& id) const`: найти клиента по идентификатору;
– `std::vector<Client*> getAllClients() const`: получить всех клиентов;
– `std::vector<Client*> searchByName(std::string_view name) const`: поиск по имени;
– `std::vector<Client*> searchByPhone(std::string_view phone) const`: поиск по телефону;

– const std::vector<std::shared_ptr<Client>>& getClients() const: получить коллекцию клиентов для FileManager;
– void setClients(std::vector<std::shared_ptr<Client>>&& newClients): установить коллекцию клиентов из FileManager;
– size_t getCount() const: получить количество клиентов.

15 Класс TransactionManager:

Менеджер сделок: управление коллекцией сделок, добавление, удаление и поиск по различным критериям.

Поля:

– std::vector<std::shared_ptr<Transaction>> transactions: коллекция сделок.

Методы:

– TransactionManager(): конструктор менеджера;
– void addTransaction(std::shared_ptr<Transaction> transaction): добавить сделку;
– bool removeTransaction(const std::string& id): удалить сделку по идентификатору;
– Transaction* findTransaction(const std::string& id) const: найти сделку по идентификатору;
– std::vector<Transaction*> getAllTransactions() const: получить все сделки;
– std::vector<Transaction*> getTransactionsByClient(std::string_view clientId) const: получить сделки клиента;
– std::vector<Transaction*> getTransactionsByProperty(std::string_view propertyId) const: получить сделки по объекту недвижимости;
– std::vector<Transaction*> getTransactionsByStatus(std::string_view status) const: получить сделки по статусу;
– const std::vector<std::shared_ptr<Transaction>>& getTransactions() const: получить коллекцию сделок для FileManager;
– void setTransactions(std::vector<std::shared_ptr<Trans

action>>&& newTransactions): установить коллекцию сделок из FileManager;
– size_t getCount() const: получить количество сделок.

16 Класс AuctionManager:

Менеджер аукционов: управление коллекцией аукционов, добавление, удаление и поиск.

Поля:

– std::vector<std::shared_ptr<Auction>> auctions: коллекция аукционов.

Методы:

– AuctionManager(): конструктор менеджера;
– void addAuction(std::shared_ptr<Auction> auction): добавить аукцион;
– bool removeAuction(const std::string& id): удалить аукцион по идентификатору;
– Auction* findAuction(const std::string& id) const: найти аукцион по идентификатору;
– std::vector<Auction*> getAllAuctions() const: получить все аукционы;
– std::vector<Auction*> getActiveAuctions() const: получить активные аукционы;
– std::vector<Auction*> getCompletedAuctions() const: получить завершенные аукционы;
– std::vector<Auction*> getAuctionsByProperty(std::string_view propertyId) const: получить аукционы по объекту недвижимости;
– const std::vector<std::shared_ptr<Auction>>& getAuctions() const: получить коллекцию аукционов для FileManager;
– void setAuctions(std::vector<std::shared_ptr<Auction>> && newAuctions): установить коллекцию аукционов из FileManager;
– size_t getCount() const: получить количество аукционов.

17 Класс FileManager:

Менеджер файлов: сохранение и загрузка данных в текстовый формат с разметкой через менеджеры.

Поля:

– `static constexpr char FILE_DELIMITER`: разделитель полей в файле;
– `static constexpr char AVAILABLE_CHAR`: символ доступности;
– `static constexpr char UNAVAILABLE_CHAR`: символ недоступности;
– `static constexpr size_t BID_PREFIX_LENGTH`: длина префикса ставки.

Методы:

– `static void saveProperties(const PropertyManager& manager, const std::string& filename)`: сохранить объекты недвижимости в файл;
– `static void saveClients(const ClientManager& manager, const std::string& filename)`: сохранить клиентов в файл;
– `static void saveTransactions(const TransactionManager& manager, const std::string& filename)`: сохранить сделки в файл;
– `static void saveAuctions(const AuctionManager& manager, const std::string& filename)`: сохранить аукционы в файл;
– `static void loadProperties(PropertyManager& manager, const std::string& filename)`: загрузить объекты недвижимости из файла;
– `static void loadClients(ClientManager& manager, const std::string& filename)`: загрузить клиентов из файла;
– `static void loadTransactions(TransactionManager& manager, const std::string& filename)`: загрузить сделки из файла;
– `static void loadAuctions(AuctionManager& manager, const std::string& filename)`: загрузить аукционы из файла;
– `static void parseBidLine(std::string_view line, Auction* currentAuction)`: парсинг строки ставки из файла;
– `static std::shared_ptr<Auction> parseAuctionLine(std::string_view line)`: парсинг строки аукциона из файла.

18 Класс `EstateAgency`:

Центральный класс-синглтон агентства недвижимости: управление всеми менеджерами и координация работы системы.

Поля:

- `static EstateAgency* instance`: единственный экземпляр синглтона;
- `PropertyManager propertyManager`: менеджер объектов недвижимости;
- `ClientManager clientManager`: менеджер клиентов;
- `TransactionManager transactionManager`: менеджер сделок;
- `AuctionManager auctionManager`: менеджер аукционов;
- `std::string dataDirectory`: директория для хранения данных.

Методы:

- `EstateAgency()`: приватный конструктор синглтона;
- `~EstateAgency()`: деструктор синглтона;
- `static EstateAgency* getInstance()`: получить экземпляр синглтона;
- `static void destroyInstance()`: уничтожить экземпляр синглтона;
- `PropertyManager& getPropertyManager()`: получить менеджер объектов недвижимости;
- `ClientManager& getClientManager()`: получить менеджер клиентов;
- `TransactionManager& getTransactionManager()`: получить менеджер сделок;
- `AuctionManager& getAuctionManager()`: получить менеджер аукционов;
- `void saveAllData() const`: сохранить все данные через FileManager;
- `void loadAllData()`: загрузить все данные через FileManager;
- `void setDataDirectory(std::string_view dir)`: установить директорию для данных;
- `std::string getDataDirectory() const`: получить директорию для данных.

19 Класс `PropertyManagerException`:

Исключение для ошибок PropertyManager, наследуется от std::exception.

Поля:

- `std::string message`: сообщение об ошибке.

Методы:

– `explicit PropertyManagerException(const std::string& msg)`: конструктор исключения;
– `const char* what() const noexcept override`: получить сообщение об ошибке.

20 Класс `ClientManagerException`:

Исключение для ошибок ClientManager, наследуется от std::exception.

Поля:

– `std::string message`: сообщение об ошибке.

Методы:

– `explicit ClientManagerException(const std::string& msg)`: конструктор исключения;
– `const char* what() const noexcept override`: получить сообщение об ошибке.

21 Класс `TransactionManagerException`:

Исключение для ошибок TransactionManager, наследуется от std::exception.

Поля:

– `std::string message`: сообщение об ошибке.

Методы:

– `explicit TransactionManagerException(const std::string& msg)`: конструктор исключения;
– `const char* what() const noexcept override`: получить сообщение об ошибке.

22 Класс `AuctionManagerException`:

Исключение для ошибок AuctionManager, наследуется от std::exception.

Поля:

– `std::string message`: сообщение об ошибке.

Методы:

– `explicit AuctionManagerException(const std::string& msg)`: конструктор исключения;
– `const char* what() const noexcept override`: получить сообщение об ошибке.

23 Класс `FileManagerException`:

Исключение для ошибок FileManager, наследуется от std::exception.

Поля:

– `std::string message`: сообщение об ошибке.

Методы:

– `explicit FileManagerException(const std::string& msg)`: конструктор исключения;

– `const char* what() const noexcept override`: получить сообщение об ошибке.

24 Класс `ContainerManager`:

Шаблонный класс-контейнер для управления коллекциями объектов с идентификаторами.

Поля:

– `std::vector<std::shared_ptr<T>> items`: коллекция элементов.

Методы:

– `void add(std::shared_ptr<T> item)`: добавить элемент;
– `void remove(const std::string& id)`: удалить элемент по идентификатору;
– `std::shared_ptr<T> findById(const std::string& id) const`: найти элемент по идентификатору;
– `size_t size() const`: получить количество элементов;
– `typename std::vector<std::shared_ptr<T>>::const_iterator begin() const`: итератор на начало;
– `typename std::vector<std::shared_ptr<T>>::const_iterator end() const`: итератор на конец.

25 Класс `MainWindow`:

Главное окно приложения: навигация, виджеты для работы с данными, обработка исключений и сохранение/загрузка данных.

Поля:

– `EstateAgency* agency`: указатель на экземпляр EstateAgency;
– `QListWidget* navigationList`: виджет навигации;
– `QStackedWidget* contentStack`: стек виджетов для переключения разделов;
– `DashboardWidget* dashboardWidget`: виджет панели управления;
– `PropertiesWidget* propertiesWidget`: виджет управления объектами недвижимости;
– `ClientsWidget* clientsWidget`: виджет управления клиентами;
– `TransactionsWidget* transactionsWidget`: виджет управления сделками;
– `AuctionsWidget* auctionsWidget`: виджет управления аукционами.

Методы:

– `explicit MainWindow(QWidget* parent = nullptr)`: конструктор главного окна;
– `~MainWindow() override`: деструктор главного окна;
– `void setupUI()`: настройка интерфейса;
– `void setupMenuBar()`: настройка меню;
– `void setupNewUI()`: настройка нового интерфейса;
– `void applyStyles()`: применение стилей;
– `void updateDashboardStats()`: обновление статистики на панели;
– `void showStatusMessage(const QString& message, int timeout = 3000) const`: показать сообщение в статусной строке;
– `void handleException(const std::exception& e)`: обработка исключений;
– `template<typename Func> void executeWithExceptionHandling(Func&& operation)`: выполнение операции с обработкой исключений;
– `void saveAllData()`: сохранить все данные;
– `void loadAllData()`: загрузить все данные;
– `void refreshAllData()`: обновить все данные;
– `void onNavigationChanged(int index)`: обработчик изменения навигации;
– `void onDataChanged()`: обработчик изменения данных.

26 Класс `PropertyDialog`:

Диалог добавления и редактирования объекта недвижимости: форма с общими полями и динамическими полями в зависимости от типа.

Поля:

– `enum class PropertyType`: тип недвижимости (TypeApartment, TypeHouse, TypeCommercial);
– `struct CommonFields`: общие поля для всех типов недвижимости;
– `struct ApartmentFields`: поля специфичные для квартиры;
– `struct HouseFields`: поля специфичные для дома;
– `struct CommercialFields`: поля специфичные для коммерческой недвижимости;
– `QDialogButtonBox* buttonBox`: кнопки диалога.

Методы:

– `explicit PropertyDialog(QWidget* parent = nullptr, const Property* editProperty = nullptr)`: конструктор диалога;

– `~PropertyDialog() override`: деструктор диалога;
– `PropertyType getPropertyType() const`: получить тип недвижимости;
– `QString getId() const`: получить идентификатор;
– `QString getCity() const`: получить город;
– `QString getStreet() const`: получить улицу;
– `QString getHouse() const`: получить номер дома;
– `double getPrice() const`: получить цену;
– `double getArea() const`: получить площадь;
– `QString getDescription() const`: получить описание;
– `bool getIsAvailable() const`: проверить доступность;
– `int getRooms() const`: получить количество комнат (квартира);
– `int getFloor() const`: получить этаж (квартира);
– `bool getHasBalcony() const`: проверить наличие балкона (квартира);
– `bool getHasElevator() const`: проверить наличие лифта (квартира);
– `int getFloors() const`: получить количество этажей (дом);
– `double getLandArea() const`: получить площадь участка (дом);
– `bool getHasGarage() const`: проверить наличие гаража (дом);
– `bool getHasGarden() const`: проверить наличие сада (дом);
– `QString getBusinessType() const`: получить тип бизнеса (коммерческая);
– `int getParkingSpaces() const`: получить количество парковочных мест (коммерческая);
– `bool getIsVisibleFromStreet() const`: проверить видимость с улицы (коммерческая);
– `bool getHasParking() const`: проверить наличие парковки (коммерческая);
– `void propertyTypeChanged(int index)`: обработчик изменения типа недвижимости;
– `void validateAndAccept()`: валидация и принятие данных;
– `void setupUI()`: настройка интерфейса;
– `void loadPropertyData(const Property* prop)`: загрузка данных объекта для редактирования;
– `void updateTypeSpecificFields()`: обновление полей в зависимости от типа.

27 Класс `ClientDialog`:

Диалог добавления и редактирования клиента: форма с полями идентификатора, имени, телефона и email.

Поля:

- `QLineEdit* idEdit`: поле ввода идентификатора;
- `QLineEdit* nameEdit`: поле ввода имени;
- `QLineEdit* phoneEdit`: поле ввода телефона;
- `QLineEdit* emailEdit`: поле ввода email;
- `QDialogButtonBox* buttonBox`: кнопки диалога.

Методы:

- `explicit ClientDialog(QWidget* parent = nullptr, const Client* editClient = nullptr)`: конструктор диалога;
- `~ClientDialog() override`: деструктор диалога;
- `QString getId() const`: получить идентификатор;
- `QString getName() const`: получить имя;
- `QString getPhone() const`: получить телефон;
- `QString getEmail() const`: получить email;
- `void validateAndAccept()`: валидация и принятие данных;
- `void setupUI()`: настройка интерфейса;
- `void loadClientData(const Client* client)`: загрузка данных клиента для редактирования.

28 Класс `TransactionDialog`:

Диалог добавления и редактирования сделки: форма с выбором объекта недвижимости и клиента, ценой, статусом и примечаниями.

Поля:

- `QLineEdit* idEdit`: поле ввода идентификатора сделки;
- `QComboBox* propertyCombo`: выбор объекта недвижимости;
- `QComboBox* clientCombo`: выбор клиента;
- `QDoubleSpinBox* priceSpin`: поле ввода финальной цены;
- `QComboBox* statusCombo`: выбор статуса сделки;
- `QTextEdit* notesEdit`: поле ввода примечаний;
- `QDialogButtonBox* buttonBox`: кнопки диалога;
- `QLabel* propertyPriceLabel`: метка с ценой объекта;
- `QLabel* differenceLabel`: метка с разницей цен;
- `QStringList propertyIds`: список идентификаторов объектов;
- `QStringList clientIds`: список идентификаторов клиентов;
- `bool isUpdatingFromProperty`: флаг обновления из объекта.

Методы:

- `explicit TransactionDialog(QWidget* parent = nullptr, const Transaction* editTransaction = nullptr, const QStringList& propertyIds = QStringList(), const QStringList& clientIds = QStringList())`: конструктор диалога;
- `~TransactionDialog() override`: деструктор диалога;
- `QString getId() const`: получить идентификатор;
- `QString getPropertyId() const`: получить идентификатор объекта недвижимости;
- `QString getClientId() const`: получить идентификатор клиента;
- `double getFinalPrice() const`: получить финальную цену;
- `QString getStatus() const`: получить статус;
- `QString getNotes() const`: получить примечания;
- `void validateAndAccept()`: валидация и принятие данных;
- `void onPropertyChanged(int index)`: обработчик изменения объекта недвижимости;
- `void setupUI()`: настройка интерфейса;
- `void loadTransactionData(const Transaction* trans)`: загрузка данных сделки для редактирования;
- `void updatePriceFromProperty()`: обновление цены из объекта недвижимости;
- `void updatePriceDifference()`: обновление разницы цен.

29 Класс `AuctionDialog`:

Диалог создания и просмотра аукциона: форма с выбором объекта недвижимости, начальной ценой, таблицей ставок и управлением аукционом.

Поля:

- `bool isViewMode`: режим просмотра (только чтение);
- `Auction* currentAuction`: текущий редактируемый аукцион;
- `EstateAgency* agency`: указатель на EstateAgency;
- `QLineEdit* idEdit`: поле ввода идентификатора;
- `QComboBox* propertyCombo`: выбор объекта недвижимости;
- `QDoubleSpinBox* priceSpin`: поле ввода начальной цены;
- `QLabel* propertyPriceLabel`: метка с ценой объекта;
- `QLabel* buyoutPriceLabel`: метка с ценой выкупа;
- `QLabel* currentHighestBidLabel`: метка с текущей максимальной ставкой;

– `QLabel* statusLabel`: метка со статусом;
– `QTableWidget* bidsTable`: таблица ставок;
– `QPushButton* addBidBtn`: кнопка добавления ставки;
– `QPushButton* completeAuctionBtn`: кнопка завершения аукциона;
– `QComboBox* clientCombo`: выбор клиента для ставки;
– `QDoubleSpinBox* bidAmountSpin`: поле ввода суммы ставки;
– `QDialogButtonBox* buttonBox`: кнопки диалога;
– `QStringList propertyIds`: список идентификаторов объектов.

Методы:

– `explicit AuctionDialog(QWidget* parent = nullptr, Auction* editAuction = nullptr, const QStringList& propertyIds = QStringList())`: конструктор диалога;
– `~AuctionDialog() override`: деструктор диалога;
– `QString getId() const`: получить идентификатор;
– `QString getPropertyId() const`: получить идентификатор объекта недвижимости;
– `double getStartingPrice() const`: получить начальную цену;
– `void refreshBids()`: обновить таблицу ставок;
– `void updateAuctionInfo()`: обновить информацию об аукционе;
– `void validateAndAccept()`: валидация и принятие данных;
– `void onPropertyChanged(int index)`: обработчик изменения объекта недвижимости;
– `void addBid()`: добавить ставку;
– `void completeAuction()`: завершить аукцион;
– `void refreshAuctionInfo()`: обновить информацию об аукционе;
– `void setupUI()`: настройка интерфейса;
– `void loadAuctionData(const Auction* auction)`: загрузка данных аукциона для просмотра;
– `void updatePropertyInfo()`: обновление информации об объекте недвижимости;
– `void createTransactionFromAuction()`: создание сделки из завершенного аукциона.

30 Класс `PropertiesWidget`:

Виджет управления объектами недвижимости: таблица объектов, добавление, редактирование, удаление, поиск и отображение деталей.

Поля:

- `EstateAgency* agency`: указатель на EstateAgency;
- `QTableWidget* propertiesTable`: таблица объектов недвижимости;
- `QPushButton* addPropertyBtn`: кнопка добавления объекта;
- `QPushButton* refreshPropertyBtn`: кнопка обновления;
- `QPushButton* searchPropertyBtn`: кнопка поиска;
- `QLineEdit* searchPropertyEdit`: поле ввода поискового запроса;
- `QTextEdit* propertyDetailsText`: текстовое поле для деталей объекта.

Методы:

- `explicit PropertiesWidget(EstateAgency* agency, QWidget* parent = nullptr)`: конструктор виджета;
- `void refresh()`: обновить данные;
- `void updateTable()`: обновить таблицу;
- `void addProperty()`: добавить объект недвижимости;
- `void editProperty()`: редактировать объект недвижимости;
- `void deleteProperty()`: удалить объект недвижимости;
- `void searchProperties()`: поиск объектов недвижимости;
- `void propertySelectionChanged()`: обработчик изменения выбора в таблице;
- `void setupUI()`: настройка интерфейса;
- `void showPropertyDetails(const Property* prop)`: показать детали объекта;
- `void showPropertyTransactions(const std::string& propertyId)`: показать сделки по объекту;
- `void addPropertyToTable(const Property* prop)`: добавить объект в таблицу;
- `void selectRowById(QTableWidget* table, const QString& id) const`: выбрать строку по идентификатору;
- `QString getSelectedIdFromTable(const QTableWidget* table) const`: получить идентификатор выбранной строки;
- `bool checkTableSelection(const QTableWidget* table, const QString& errorMessage)`: проверить выбор в таблице;

– `bool isNumericId(const QString& text) const`: проверить, является ли текст числовым идентификатором.

31 Класс `ClientsWidget`:

Виджет управления клиентами: таблица клиентов, добавление, редактирование, удаление, поиск и отображение деталей.

Поля:

– `EstateAgency* agency`: указатель на EstateAgency;
– `QTableWidget* clientsTable`: таблица клиентов;
– `QPushButton* addClientBtn`: кнопка добавления клиента;
– `QPushButton* refreshClientBtn`: кнопка обновления;
– `QPushButton* searchClientBtn`: кнопка поиска;
– `QLineEdit* searchClientEdit`: поле ввода поискового запроса;
– `QTextEdit* clientDetailsText`: текстовое поле для деталей клиента.

Методы:

– `explicit ClientsWidget(EstateAgency* agency, QWidget* parent = nullptr)`: конструктор виджета;
– `void refresh()`: обновить данные;
– `void updateTable()`: обновить таблицу;
– `void addClient()`: добавить клиента;
– `void editClient()`: редактировать клиента;
– `void deleteClient()`: удалить клиента;
– `void searchClients()`: поиск клиентов;
– `void clientSelectionChanged()`: обработчик изменения выбора в таблице;
– `void setupUI()`: настройка интерфейса;
– `void showClientDetails(const Client* client)`: показать детали клиента;
– `void showClientTransactions(const std::string& clientId)`: показать сделки клиента;
– `void addClientToTable(const Client* client)`: добавить клиента в таблицу;
– `void selectRowById(QTableWidget* table, const QString& id) const`: выбрать строку по идентификатору;
– `QString getSelectedIdFromTable(const QTableWidget* table) const`: получить идентификатор выбранной строки;

– `bool checkTableSelection(const QTableWidget* table, const QString& errorMessage)`: проверить выбор в таблице.

32 Класс `TransactionsWidget`:

Виджет управления сделками: таблица сделок, добавление, редактирование, удаление, поиск и отображение деталей.

Поля:

– `EstateAgency* agency`: указатель на EstateAgency;
– `QTableWidget* transactionsTable`: таблица сделок;
– `QPushButton* addTransactionBtn`: кнопка добавления сделки;
– `QPushButton* refreshTransactionBtn`: кнопка обновления;
– `QPushButton* searchTransactionBtn`: кнопка поиска;
– `QLineEdit* searchTransactionEdit`: поле ввода поискового запроса;
– `QTextEdit* transactionDetailsText`: текстовое поле для деталей сделки.

Методы:

– `explicit TransactionsWidget(EstateAgency* agency, QWidget* parent = nullptr)`: конструктор виджета;
– `void refresh()`: обновить данные;
– `void updateTable()`: обновить таблицу;
– `void addTransaction()`: добавить сделку;
– `void editTransaction()`: редактировать сделку;
– `void deleteTransaction()`: удалить сделку;
– `void searchTransactions()`: поиск сделок;
– `void transactionSelectionChanged()`: обработчик изменения выбора в таблице;
– `void setupUI()`: настройка интерфейса;
– `void showTransactionDetails(const Transaction* trans)`: показать детали сделки;
– `void addTransactionToTable(const Transaction* trans)`: добавить сделку в таблицу;
– `bool validateTransaction(std::string_view propertyId, std::string_view clientId, std::string_view status, std::string_view excludeId = "")`: валидация сделки;

– bool hasActiveTransactions(const std::string& propertyId): проверить наличие активных сделок по объекту;
– void selectRowById(QTableWidget* table, const QString& id) const: выбрать строку по идентификатору;
– QString getSelectedIdFromTable(const QTableWidget* table) const: получить идентификатор выбранной строки;
– bool checkTableSelection(const QTableWidget* table, const QString& errorMessage): проверить выбор в таблице.

33 Класс AuctionsWidget:

Виджет управления аукционами: таблица аукционов, создание, просмотр, удаление, поиск и отображение деталей.

Поля:

– EstateAgency* agency: указатель на EstateAgency;
– QTableWidget* auctionsTable: таблица аукционов;
– QPushButton* addAuctionBtn: кнопка создания аукциона;
– QPushButton* refreshAuctionBtn: кнопка обновления;
– QPushButton* searchAuctionBtn: кнопка поиска;
– QLineEdit* searchAuctionEdit: поле ввода поискового запроса;
– QTextEdit* auctionDetailsText: текстовое поле для деталей аукциона.

Методы:

– explicit AuctionsWidget(EstateAgency* agency, QWidget* parent = nullptr): конструктор виджета;
– void refresh(): обновить данные;
– void updateTable(): обновить таблицу;
– void addAuction(): создать аукцион;
– void viewAuction(): просмотреть аукцион;
– void deleteAuction(): удалить аукцион;
– void searchAuctions(): поиск аукционов;
– void auctionSelectionChanged(): обработчик изменения выбора в таблице;
– void setupUI(): настройка интерфейса;
– void showAuctionDetails(const Auction* auction): показать детали аукциона;
– void addAuctionToTable(const Auction* auction): добавить аукцион в таблицу;

– `bool hasActiveTransactions(const std::string& propertyId)`: проверить наличие активных сделок по объекту;
– `QWidget* createActionButtons(QTableWidget* table, const QString& id, const std::function<void()>& viewAction, const std::function<void()>& deleteAction, bool isView = false)`: создать кнопки действий в таблице;
– `void selectRowById(QTableWidget* table, const QString& id) const`: выбрать строку по идентификатору;
– `QString getSelectedIdFromTable(const QTableWidget* table) const`: получить идентификатор выбранной строки;
– `bool checkTableSelection(const QTableWidget* table, const QString& errorMessage)`: проверить выбор в таблице.

34 Класс `DashboardWidget`:

Виджет панели управления: отображение статистики по объектам недвижимости, клиентам, сделкам и аукционам.

Поля:

– `EstateAgency* agency`: указатель на EstateAgency;
– `QLabel* statsPropertiesLabel`: метка со статистикой по объектам недвижимости;
– `QLabel* statsClientsLabel`: метка со статистикой по клиентам;
– `QLabel* statsTransactionsLabel`: метка со статистикой по сделкам;
– `QLabel* statsAvailableLabel`: метка со статистикой по доступным объектам;
– `QLabel* statsAuctionsLabel`: метка со статистикой по аукционам;
– `QPushButton* saveBtn`: кнопка сохранения данных;
– `QPushButton* loadBtn`: кнопка загрузки данных;
– `QPushButton* refreshBtn`: кнопка обновления статистики.

Методы:

– `explicit DashboardWidget(EstateAgency* agency, QWidget* parent = nullptr)`: конструктор виджета;
– `void updateStats()`: обновить статистику;
– `void setupUI()`: настройка интерфейса.

3.2 Примеры кода

3.2.1 Функция FileManager::loadProperties

Реализована с учетом следующих принципов:

– безопасное чтение файлов. Проверяется существование файла и обрабатываются ошибки чтения;

– обработка ошибок парсинга. При некорректных данных строка пропускается, загрузка продолжается;

– поддержка множественных типов. Система распознает и загружает различные типы недвижимости (квартиры, дома, коммерческая недвижимость);

– восстановление состояния. При загрузке восстанавливается полное состояние объектов, включая статус доступности.

Ключевые фрагменты кода и их описание:

1 Валидация существования файла.

```
void FileManager::loadProperties(PropertyManager
&manager, const std::string &filename)
{
    std::vector<std::unique_ptr<Property>> properties;
    std::ifstream file(filename);
    if (!file.is_open())
    {
        return;
    }
```

Перед чтением проверяется возможность открытия файла. Если файл не существует или недоступен, функция завершается без ошибки, что позволяет системе работать с пустыми данными при первом запуске.

2 Парсинг строк с обработкой пустых строк.

```
while (std::getline(file, line))
    {
        if (line.empty())
        {
            continue;
        }

        std::istringstream iss(line);
        std::string type;
        std::getline(iss, type, FILE_DELIMITER);
```

Строки читаются построчно с пропуском пустых строк. Тип объекта определяется по первому полю до разделителя. Это обеспечивает надежную обработку файлов с различным форматированием.

3 Обработка различных типов недвижимости.

```
try
        {
            if (type == "APARTMENT")
            {
                ApartmentParams params{
                    {id, city, street, house, price,
area, desc},
                    rooms, floor, balcony == 1, elevator
== 1};
                auto apartment =
std::make_unique<Apartment>(params);
                apartment->setAvailable(avail ==
std::string(1, AVAILABLE_CHAR));

properties.push_back(std::move(apartment));
            }
            else if (type == "HOUSE")
            {

            }
            else if (type == "COMMERCIAL")
            {

            }
        }
        catch (const std::invalid_argument &)
        {
            continue;
        }
        catch (const PropertyManagerException &)
        {
            continue;
        }
```

Для каждого типа недвижимости выполняется специфический парсинг полей (в данном случае показан один из вариантов). При ошибках парсинга или валидации некорректная строка пропускается, но загрузка продолжается для остальных записей. Это обеспечивает максимальную устойчивость к поврежденным данным.

4 Восстановление статуса доступности.

```
apartment->setAvailable(avail == std::string(1,
AVAILABLE_CHAR));
```

Статус доступности восстанавливается из файла на основе сохраненного символа. Это гарантирует корректное восстановление состояния объектов после перезапуска приложения.

5 Применение загруженных данных.

```
    file.close();
    manager.setProperties(std::move(properties));
}
```

После успешной загрузки всех объектов данные передаются в менеджер через перемещение, что обеспечивает эффективность и атомарность операции обновления.

3.2.2 Функция Auction::addBid

Реализована с учетом следующих принципов:

– проверка статуса аукциона. Ставки принимаются только для активных аукционов;

– валидация суммы ставки. Ставка должна превышать текущую максимальную на минимальный шаг или достигать цены выкупа;

– автоматическое завершение. При достижении цены выкупа аукцион автоматически завершается;

Ключевые фрагменты кода и их описание:

1 Проверка статуса аукциона.

```
bool Auction::addBid(std::shared_ptr<Bid> bid)
{
    if (status != Constants::AuctionStatus::ACTIVE)
    {
        return false;
    }

    double currentHighest = getCurrentHighestBid();
```

Перед обработкой ставки проверяется, что аукцион находится в активном состоянии. Это предотвращает добавление ставок в завершенные или отмененные аукционы.

2 Автоматическое завершение при достижении цены выкупа.

```
    if (bid->getAmount() >= buyoutPrice)
    {
        bids.push_back(bid);
        complete();
        return true;
    }
```

Если ставка достигает или превышает цену выкупа, ставка принимается и аукцион немедленно завершается. Это реализует механизм мгновенной покупки без ожидания окончания торгов.

3 Валидация минимальной ставки.

```
    if (double minBid = (currentHighest > NO_BID) ? 
currentHighest + MIN_BID_INCREMENT : startingPrice;
            bid->getAmount() < minBid)
        {
            return false;
        }

        bids.push_back(bid);
        return true;
```

Минимальная ставка рассчитывается как текущая максимальная плюс минимальный шаг (0.01 рубля) или начальная цена, если ставок еще нет. Ставка добавляется только если она соответствует требованиям, что гарантирует корректность торгов.

3.2.3 Функция AuctionDialog::createTransactionFromAuction

Реализована с учетом следующих принципов:

– генерация уникального идентификатора. Идентификатор сделки генерируется на основе времени с обеспечением уникальности;

– проверка дубликатов. Перед созданием проверяется отсутствие аналогичной сделки;

– автоматическое обновление статуса. После создания сделки объект недвижимости помечается как недоступный;

– обработка ошибок. При ошибках создания отображаются информативные сообщения пользователю.

Ключевые фрагменты кода и их описание:

1 Проверка наличия победителя.

```
void AuctionDialog::createTransactionFromAuction()
{
    if (!currentAuction || !agency)
        return;

    const Bid *winner = currentAuction->getHighestBid();
    if (!winner)
        return;
    if (hasExistingTransaction(agency, currentAuction-
>getPropertyId(), winner->getClientId(), winner-
>getAmount()))
```

```
        return;
```

Перед созданием сделки проверяется наличие победителя аукциона и отсутствие дублирующей сделки. Это предотвращает создание повторных сделок при повторных вызовах функции.

2 Генерация уникального идентификатора сделки.

```
std::string generateBaseTransactionId()
{
    auto now = std::chrono::system_clock::now();
    auto time =
std::chrono::system_clock::to_time_t(now);
    auto tm = Utils::getLocalTime(time);
    std::ostringstream oss;
    oss << std::put_time(&tm, "%H%M%S");
    std::string transactionId = oss.str();

    if (transactionId.length() < 8)
    {
        transactionId += std::to_string(tm.tm_mday %
100);
    }

    if (transactionId.length() > 8)
    {
        transactionId = transactionId.substr(0, 8);
    }
    if (transactionId.length() < 6)
    {
        transactionId = transactionId + std::string(6 -
transactionId.length(), '0');
    }

    return transactionId;
}
```

Базовый идентификатор генерируется на основе текущего времени (часы, минуты, секунды, день месяца). Идентификатор нормализуется до длины от 6 до 8 символов, что соответствует требованиям валидации.

3 Обеспечение уникальности идентификатора.

```
std::string ensureUniqueTransactionId(EstateAgency
*agency, const std::string &baseId, const std::string
&auctionId)
{
```

```
    std::string transactionId = baseId;
    std::string originalId = baseId;
    int suffix = 1;

    while (agency-
>getTransactionManager().findTransaction(transactionId)
!= nullptr)
    {
        if (transactionId.length() < 8)
        {
            transactionId = std::format("{}{}",
originalId, suffix % 10);
            if (transactionId.length() > 8)
            {
                transactionId = transactionId.substr(0,
8);
            }
        }
        else
        {
            transactionId = std::format("{}{}",
originalId.substr(0, 6), suffix % 100);
        }
        suffix++;

        if (suffix > 999)
        {
            transactionId =
generateFallbackTransactionId(auctionId);
            suffix = 1;
        }
    }

    return transactionId;
}
```

Если сгенерированный идентификатор уже существует, к нему добавляется числовой суффикс. При превышении лимита попыток используется резервный алгоритм генерации на основе идентификатора аукциона. Это гарантирует создание уникальной сделки даже при высокой нагрузке.

4 Создание сделки и обновление состояния.

```
    try
        {
```

```
            double finalPrice = winner->getAmount();
            std::string notes = "Продажа через аукцион. 
Аукцион ID: " + currentAuction->getId();

            auto transaction = 
std::make_shared<Transaction>(transactionId, 
currentAuction->getPropertyId(),

winner->getClientId(), finalPrice, "completed", notes);

            agency-
>getTransactionManager().addTransaction(transaction);

            Property *prop = agency-
>getPropertyManager().findProperty(currentAuction-
>getPropertyId());
            if (prop)
            {
                prop->setAvailable(false);
            }
        }
        catch (const std::invalid_argument &e)
        {
            QMessageBox::warning(this, "Ошибка", 
QString("Ошибка создания сделки: %1").arg(e.what()));
        }
```

Сделка создается со статусом «completed» и примечанием об источнике. После успешного добавления объект недвижимости помечается как недоступный, что предотвращает создание конфликтующих сделок.

3.2.4 Функция PropertyManager::searchByAddress

Реализована с учетом следующих принципов:

– нормализация строк. Поиск выполняется без учета регистра для удобства пользователя;

– частичное совпадение. Поиск работает по подстрокам, что позволяет находить объекты при неполном указании адреса;

– комбинированный поиск. Поиск выполняется по городу, улице и дому одновременно с возможностью указания любого количества параметров;

– эффективная фильтрация. Результаты формируются за один проход по коллекции.

Ключевые фрагменты кода и их описание:

1 Нормализация входных данных.

```
std::vector<Property                                    *>
PropertyManager::searchByAddress(const std::string &city,
const std::string &street,

const std::string &house) const
{
    std::vector<Property *> result;

    std::string lowerCity = city;
    std::string lowerStreet = street;
    std::string lowerHouse = house;
    std::ranges::transform(lowerCity,  lowerCity.begin(),
[](unsigned char c) { return std::tolower(c); });
    std::ranges::transform(lowerStreet,
lowerStreet.begin(),  [](unsigned  char  c)  {  return
std::tolower(c); });
    std::ranges::transform(lowerHouse,
lowerHouse.begin(),  [](unsigned  char  c)  {  return
std::tolower(c); });
```

Все входные строки преобразуются в нижний регистр для обеспечения поиска без учета регистра. Это улучшает пользовательский опыт, позволяя находить объекты независимо от регистра ввода.

2 Нормализация данных объектов.

```
    for (const auto &prop : properties)
    {
        std::string propCity = prop->getCity();
        std::string propStreet = prop->getStreet();
        std::string propHouse = prop->getHouse();

        std::ranges::transform(propCity,
propCity.begin(), [](unsigned char c) { return
std::tolower(c); });
        std::ranges::transform(propStreet,
propStreet.begin(), [](unsigned char c) { return
std::tolower(c); });
        std::ranges::transform(propHouse,
propHouse.begin(), [](unsigned char c) { return
std::tolower(c); });
```

Адресные данные каждого объекта также нормализуются для сравнения. Это обеспечивает корректное сопоставление независимо от того, как данные были сохранены в системе.

3 Проверка совпадения по всем параметрам.

```
bool matches = true;
        if (!lowerCity.empty() &&
!Utils::stringContains(propCity, lowerCity))
        {
            matches = false;
        }
        if (!lowerStreet.empty() &&
!Utils::stringContains(propStreet, lowerStreet))
        {
            matches = false;
        }
        if (!lowerHouse.empty() &&
!Utils::stringContains(propHouse, lowerHouse))
        {
            matches = false;
        }

        if (matches && (!lowerCity.empty() ||
!lowerStreet.empty() || !lowerHouse.empty()))
        {
            result.push_back(prop.get());
        }
    }
    return result;
```

Проверка выполняется только для непустых параметров поиска. Объект включается в результат, если все указанные параметры совпадают (частичное совпадение) и хотя бы один параметр был указан. Это позволяет гибко искать как по полному адресу, так и по отдельным компонентам.

3.2.5 Функция FileManager::loadAuctions

Реализована с учетом следующих принципов:

– последовательная обработка строк. Аукционы и ставки обрабатываются в порядке появления в файле;

– связывание ставок с аукционами. Ставки автоматически привязываются к текущему аукциону;

– восстановление статусов. При загрузке восстанавливаются статусы аукционов (активен, завершен, отменен);

– обработка ошибок. Некорректные строки пропускаются без прерывания загрузки.

Ключевые фрагменты кода и их описание:

1 Разделение обработки аукционов и ставок.

```
void FileManager::loadAuctions(AuctionManager &manager,
const std::string &filename)
{
    std::vector<std::shared_ptr<Auction>> auctions;
    std::ifstream file(filename);
    if (!file.is_open())
    {
        return;
    }

    auctions.clear();
    std::string line;
    Auction *currentAuction = nullptr;
    const std::string BID_PREFIX = "BID|";

    while (std::getline(file, line))
    {
        if (line.empty())
        {
            continue;
        }

        if (line.starts_with(BID_PREFIX))
        {
            parseBidLine(line, currentAuction);
        }
        else
        {
            auto auction = parseAuctionLine(line);
            if (auction != nullptr)
            {
                auctions.push_back(auction);
                currentAuction = auction.get();
            }
        }
    }
```

Система различает строки аукционов и ставок по префиксу "BID|". Ставки привязываются к последнему загруженному аукциону, что соответствует структуре файла, где ставки следуют сразу после описания аукциона.

2 Парсинг строки аукциона.

```
        std::shared_ptr<Auction>
FileManager::parseAuctionLine(std::string_view line)
{
    std::string lineStr(line);
    std::istringstream iss(lineStr);
    std::string id;
    std::string propertyId;
    std::string propertyAddress;
    std::string startingPriceStr;
    std::string buyoutPriceStr;
    std::string status;
    std::string createdAt;
    std::string completedAt;

    std::getline(iss, id, FILE_DELIMITER);
    std::getline(iss, propertyId, FILE_DELIMITER);
    ...

    try
    {
        double startingPrice =
std::stod(startingPriceStr);
        auto auction = std::make_shared<Auction>(id,
propertyId, propertyAddress, startingPrice);

        if (status ==
Constants::AuctionStatus::COMPLETED)
        {
            auction->complete();
        }
        else if (status ==
Constants::AuctionStatus::CANCELLED)
        {
            auction->cancel();
        }
        return auction;
    }
    catch (const std::invalid_argument &e)
    {
        return nullptr;
    }
}
```

Парсинг выполняется с извлечением всех полей через разделитель. Статус аукциона восстанавливается через вызов соответствующих методов, что гарантирует корректное состояние объекта.

3 Парсинг строки ставки.

```
void FileManager::parseBidLine(std::string_view line,
Auction *currentAuction)
{
    if (currentAuction == nullptr)
    {
        return;
    }

    try
    {
        std::string lineStr(line);
        std::istringstream
iss(lineStr.substr(BID_PREFIX_LENGTH));
        std::string auctionId;
        std::string clientId;
        std::string clientName;
        std::string amountStr;
        std::string timestamp;


        if (!std::getline(iss, auctionId, FILE_DELIMITER)
|| auctionId.empty())
        {
            return;
        }
        ...

        double amount = std::stod(amountStr);
        if (amount <= 0.0)
        {
            return;
        }

        auto bid = std::make_shared<Bid>(clientId,
clientName, amount);
        currentAuction->addBidDirect(bid);
    }
    catch (const std::invalid_argument &e)
```

```
        {

        }
}
```

Ставка добавляется напрямую в аукцион без повторной валидации, так как данные уже были проверены при сохранении. Это обеспечивает быстрое восстановление состояния без дублирования бизнес-логики.

3.3 Разработка алгоритмов

Схема алгоритма создания сделки из завершенного аукциона представлена в Приложении Б, схема алгоритма поиска объекта недвижимости по адресу представлена в Приложении В.

3.3.1 Алгоритм создания сделки из завершенного аукциона

Шаг 1: Начало.

Шаг 2: Проверка наличия текущего аукциона и агентства.

Шаг 3: Если проверка не пройдена – завершение функции.

Шаг 4: Получение победителя аукциона (ставка с максимальной суммой).

Шаг 5: Если победитель не найден – завершение функции.

Шаг 6: Проверка наличия существующей сделки с теми же параметрами (недвижимость, клиент, сумма).

Шаг 7: Если сделка существует – завершение функции (предотвращение дубликатов).

Шаг 8: Генерация базового идентификатора на основе текущего времени и нормализация до 6-8 символов.

Шаг 9: Если идентификатор уже существует – добавление числового суффикса к идентификатору (при необходимости – резервный ID). Повтор проверки до получения уникального идентификатора.

Шаг 10: Создание объекта Transaction с параметрами победителя, добавление в менеджер сделок, получение объекта недвижимости и установка статуса доступности в false (недоступен).

Шаг 11: Конец.

3.3.2 Алгоритм поиска объекта недвижимости по адресу

Шаг 1: Начало.

Шаг 2: Создание вектора для хранения результатов поиска и нормализация входных параметров поиска (город, улица, дом) в нижний регистр.

Шаг 3: Получение следующего объекта недвижимости из коллекции.

Шаг 4: Если объект получен – переход к шагу 5, иначе к шагу 8.

Шаг 5: Получение и нормализация адресных данных объекта (город, улица, дом) в нижний регистр.

Шаг 6: Проверка совпадения по всем параметрам (город, улица, дом) с установкой флага совпадения.

Шаг 7: Если флаг совпадения равен true и хотя бы один параметр не пустой – добавление объекта в результаты. Переход к шагу 3.

Шаг 8: Возврат вектора результатов поиска.

Шаг 9: Конец.

3.4 Особенности работы с используемыми библиотеками

В проекте используются Qt6 (модуль `Widgets`) и стандартная библиотека C++23 для хранения данных в текстовых файлах. Qt обеспечивает кроссплатформенный графический интерфейс, а файловые репозитории заменяют базу данных: все данные о недвижимости, клиентах, сделках и аукционах читаются и записываются в .txt файлы с простым текстовым форматом с разделителями.

1 Графический интерфейс на `Qt Widgets`

Управление окнами и виджетами выполнено на основе `QMainWindow`, `QStackedWidget`, `QListWidget`, `QTableWidget`, `QTextEdit`, `QLineEdit`, `QComboBox`, `QPushButton`, `QLabel`, `QDoubleSpinBox`, QCheckBox. Приложение использует навигационный список (`QListWidget`) для переключения между разделами и стек виджетов (`QStackedWidget`) для отображения соответствующего контента: управление недвижимостью, управление клиентами, управление сделками, управление аукционами, панель управления (дашборд). Таблицы поддерживают сортировку и отображение данных о недвижимости, клиентах, сделках и аукционах с использованием `QTableWidget` с настраиваемыми заголовками и форматированием ячеек. Для удобства восприятия числовых значений реализована валидация через статические методы классов сущностей (`Property::validatePrice`, `Property::validateArea`, `Client::validateId`, `Client::validatePhone`, `Client::validateEmail`) с проверкой диапазонов и форматов. Обновление интерфейса централизовано: после каждой операции вызываются методы `refresh()` соответствующих виджетов и `updateDashboardStats()` для обновления статистики на дашборде. Стилизация интерфейса выполнена через QSS (Qt Style Sheets) с темной цветовой схемой и современным дизайном.

2 Диалоги, валидаторы и взаимодействие с пользователем

Диалоги оформлены через `QDialog` с `QDialogButtonBox` для стандартных кнопок (OK, Cancel, Close). Для ввода данных используются `QLineEdit` (идентификаторы, адреса, описания), `QComboBox` (выбор

недвижимости и клиентов), QDoubleSpinBox (цены, площади), QTextEdit (описания, примечания), QCheckBox (булевы значения: доступность, балкон, лифт, гараж, сад, парковка, видимость). Валидация выполняется через статические методы классов: проверка непустых строк (Property::validateAddressPart), целых чисел в диапазоне (Property::validateId для 6-8 символов), вещественных чисел в диапазоне (Property::validatePrice для 10000-1000000000, Property::validateArea для положительных значений), корректности форматов (Client::validatePhone для форматов +375XXXXXXXXX, Client::validateEmail с использованием регулярных выражений). Сообщения об успехе/ошибках отображаются через QMessageBox::information / warning / critical с детальными описаниями на русском языке. Сигналы и слоты Qt связывают действия кнопок с обработчиками без плотной связности между слоями: виджеты взаимодействуют с менеджерами через указатель на EstateAgency (singleton), что обеспечивает слабую связанность компонентов.

3 Формирование детальной информации в формате HTML

Детальная информация о недвижимости, клиентах, сделках и аукционах генерируется в формате HTML и отображается в QTextEdit с поддержкой стилей CSS. HTML включает заголовки с форматированием (h2, h3), параграфы с выделением ключевых полей жирным шрифтом (теги <b>), таблицы для отображения связанных данных (списки сделок, списки ставок), стилизованные блоки для группировки информации. Денежные значения форматируются с фиксированной точностью (2 знака после запятой) через QString::number с параметром 'f' и добавлением единицы измерения "руб.". Площади форматируются аналогично с единицей измерения "м²". Детальная информация включает все характеристики объекта: для недвижимости отображаются базовые поля (ID, тип, адрес, цена, площадь, описание, доступность) и специфические поля в зависимости от типа (для квартир: комнаты, этаж, балкон, лифт; для домов: этажи, комнаты, площадь участка, гараж, сад; для коммерческой недвижимости: тип бизнеса, парковка, места парковки, видимость). Для сделок отображается информация о недвижимости и клиенте, для аукционов – информация о недвижимости и список всех ставок с указанием клиента, суммы и времени. Логика формирования HTML инкапсулирована в методах showPropertyDetails, showClientDetails,showTransactionDetails, showAuctionDetails соответствующих виджетов, что обеспечивает модульность и переиспользуемость кода.

4 Хранение данных в текстовых файлах

Данные читаются и записываются через std::ifstream / std::ofstream с использованием текстового формата с разделителем '|'

(FILE_DELIMITER). Для объектов недвижимости используется формат: тип|идентификатор|город|улица|дом|цена|площадь|описание|доступность|специфические_поля, где тип может быть [APARTMENT], [HOUSE] или [COMMERCIAL]; маркеры типов записываются как строковые константы; каждое поле записывается через разделитель; специфические поля для каждого типа записываются после базовых полей (для квартир: комнаты, этаж, балкон, лифт; для домов: этажи, комнаты, площадь участка, гараж, сад; для коммерческой недвижимости: тип бизнеса, парковка, места парковки, видимость); статус доступности записывается как символ '1' (AVAILABLE_CHAR) для доступных и '0' (UNAVAILABLE_CHAR) для недоступных. Для клиентов используется формат: идентификатор|имя|телефон|email|дата_регистрации, где все поля разделены разделителем. Для сделок используется формат: идентификатор|идентификатор_недвижимости|идентификатор_клиента|дата|финальная_цена|статус|примечания. Для аукционов используется формат: идентификатор|идентификатор_недвижимости|адрес|начальная_цена|цена_выкупа|статус|дата_создания|дата_завершения; ставки записываются на отдельных строках с префиксом «BID|» в формате: BID|идентификатор_аукциона|идентификатор_клиента|имя_клиента|сумма|время. Пути к файлам и каталоги создаются через std::filesystem::create_directories, что исключает ошибки из-за отсутствующих директорий. При загрузке выполняется проверка возможности открытия файла: если файл не существует, функция завершается без ошибки, что позволяет системе работать с пустыми данными при первом запуске. При парсинге выполняется обработка пустых строк (пропуск) и некорректных данных (пропуск строки с продолжением загрузки остальных записей).

5 Нормализация чисел, точность денег и валидация

Денежные значения обрабатываются в double с принудительным округлением до сотых (копеек) при отображении через QString::number с параметром 'f' и точностью 2. Для устойчивости к пользовательскому вводу выполняется валидация через статические методы классов: проверка диапазонов (Property::validatePrice для 10000-1000000000, Property::validateArea для положительных значений), форматов чисел (Property::validateId для 6-8 цифр), непустых строк (Property::validateAddressPart для адресных компонентов). При парсинге числовых значений из файлов используется обработка исключений std::invalid_argument и std::out_of_range для корректной обработки некорректных данных: при ошибке парсинга строка пропускается, но загрузка продолжается для остальных записей. Валидация идентификаторов выполняется через Property::validateId и Client::validateId с проверкой длины (6-8 символов) и содержания только цифр через std::ranges::all_of. Валидация адресов выполняется через

Property::validateAddressPart с проверкой непустоты и максимальной длины (100 символов) и наличия печатаемых символов через std::ranges::any_of.

6 Работа со временем и форматами

Даты создания сделок и аукционов хранятся в std::string в формате ISO 8601 (yyyy-MM-dd HH:mm:ss) и генерируются через Utils::getCurrentTimeString с использованием std::chrono::system_clock и std::put_time. Для валидации форматов используется проверка через регулярные выражения (Client::validateEmail) и строковые операции (Client::validatePhone для формата +375XXXXXXXXX). При сохранении даты записываются в текстовом формате как строки, при загрузке парсятся как строки без преобразования. Для форматирования чисел используется QString::number с указанием формата ('f' для вещественных с точностью 2, без параметров для целых) и добавлением единиц измерения ("руб.", "м²"). Для форматирования цен используется утилита Utils::formatPrice, которая добавляет форматирование с двумя знаками после запятой и единицей измерения.

7 Организация кода и разделение ответственности

Менеджеры (PropertyManager, ClientManager, TransactionManager, AuctionManager) инкапсулируют бизнес-логику управления коллекциями объектов, предоставляют методы поиска, фильтрации и доступа к данным, не зависят от деталей UI. Сервисы (FileManager) отвечают за файловые операции: сохранение и загрузку данных в текстовом формате с обработкой ошибок и пропуском некорректных записей. Утилиты (Utils) предоставляют функции для работы со строками (toString, toQString, stringContains), форматирования (formatPrice, formatNumber), работы со временем (getCurrentTimeString, getLocalTime), валидации (isNumericId). Хелперы (TableHelper) предоставляют утилиты для работы с таблицами: получение выбранной строки, очистка таблицы, форматирование типов и статусов для отображения. UI-слой (MainWindow, PropertiesWidget, ClientsWidget, TransactionsWidget, AuctionsWidget, DashboardWidget, PropertyDialog, ClientDialog, TransactionDialog, AuctionDialog) занимается сбором ввода и визуализацией результата, вызывая менеджеры и сервисы через четко определенные методы EstateAgency. Сущности (Property, Apartment, House, CommercialProperty, Client, Transaction, Auction, Bid) инкапсулируют данные и базовую логику работы с ними, включая валидацию через статические методы и форматирование для сохранения через toFileString. Ядро системы (EstateAgency) реализует паттерн Singleton и

координирует работу всех менеджеров, обеспечивает централизованное сохранение и загрузку данных через `FileManager`. Такое разделение упрощает тестирование и модификацию каждого слоя, обеспечивает слабую связанность компонентов и высокую переиспользуемость кода.

4 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ

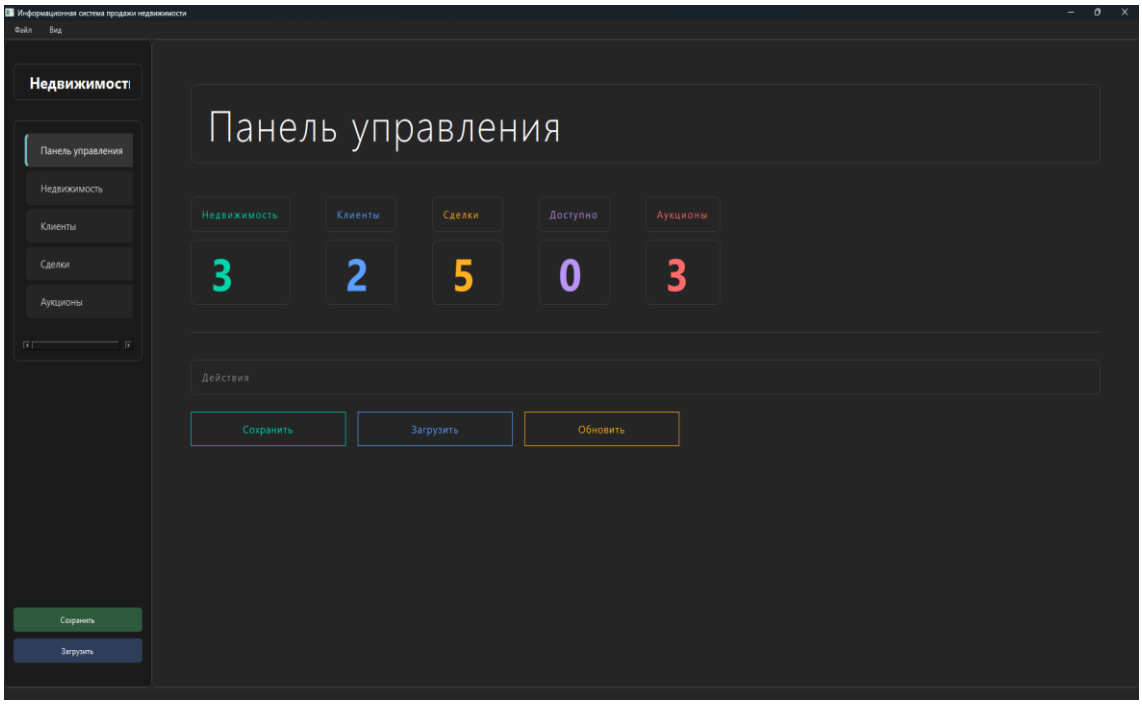

Рисунок 4.1 – Главное окно

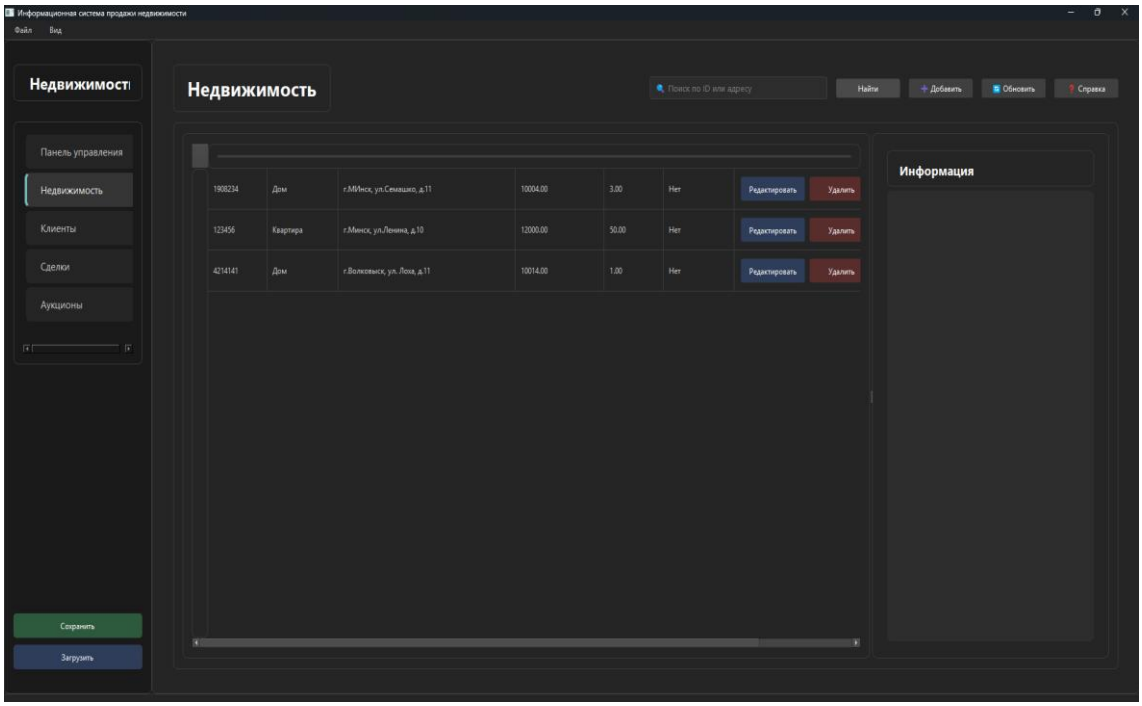

Рисунок 4.2 – Окно недвижимости

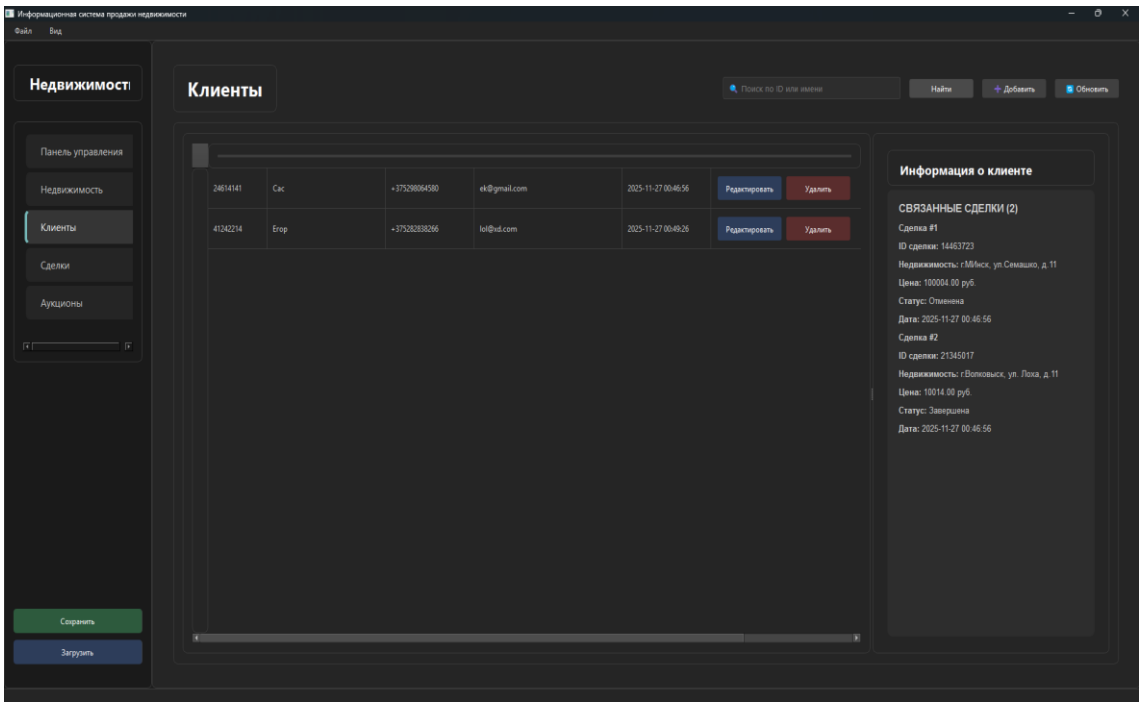

Рисунок 4.3 – Окно клиентов

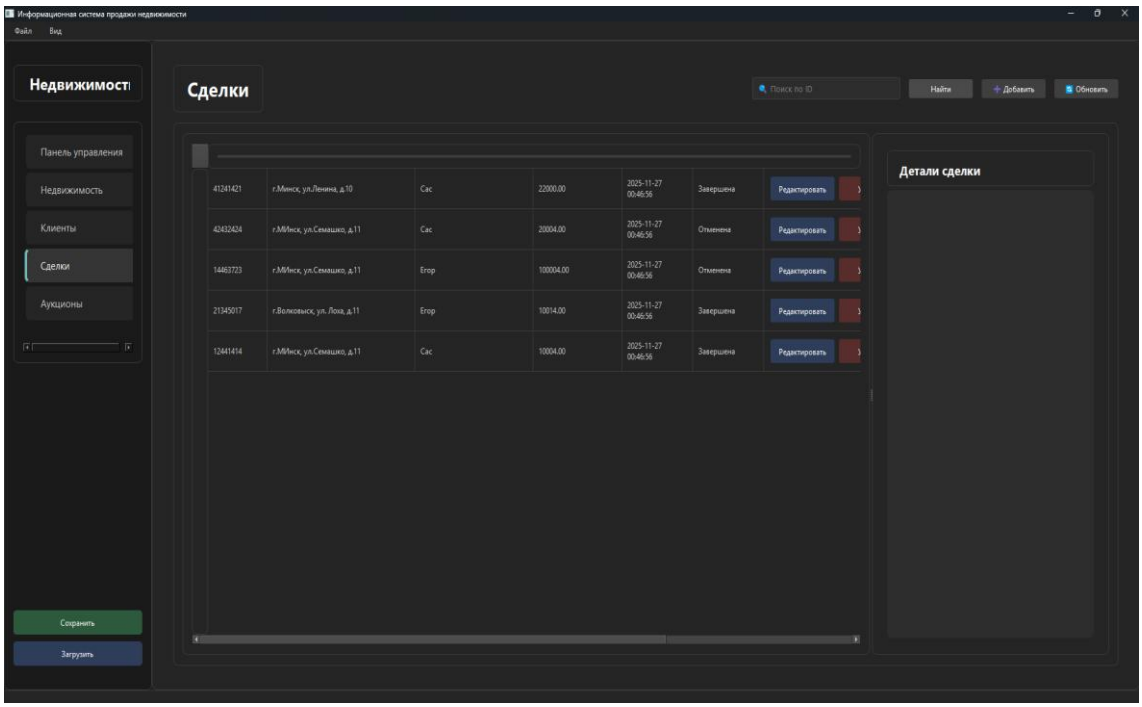

Рисунок 4.4 – Окно сделок

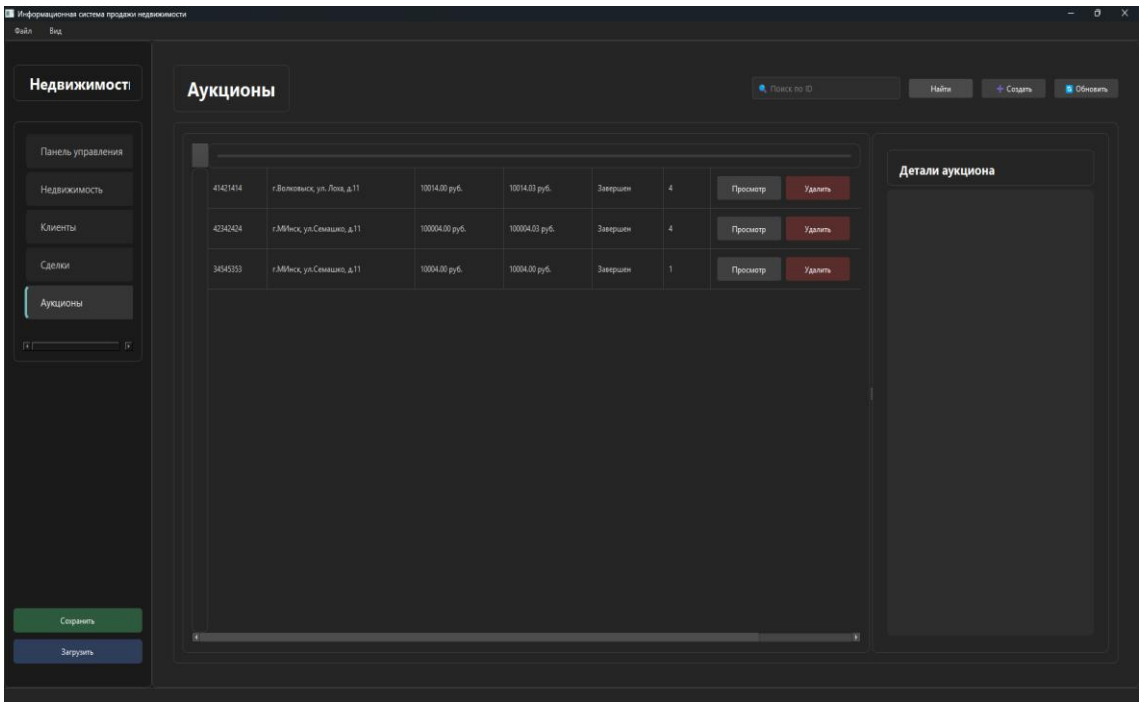

Рисунок 4.4 – Окно сделок

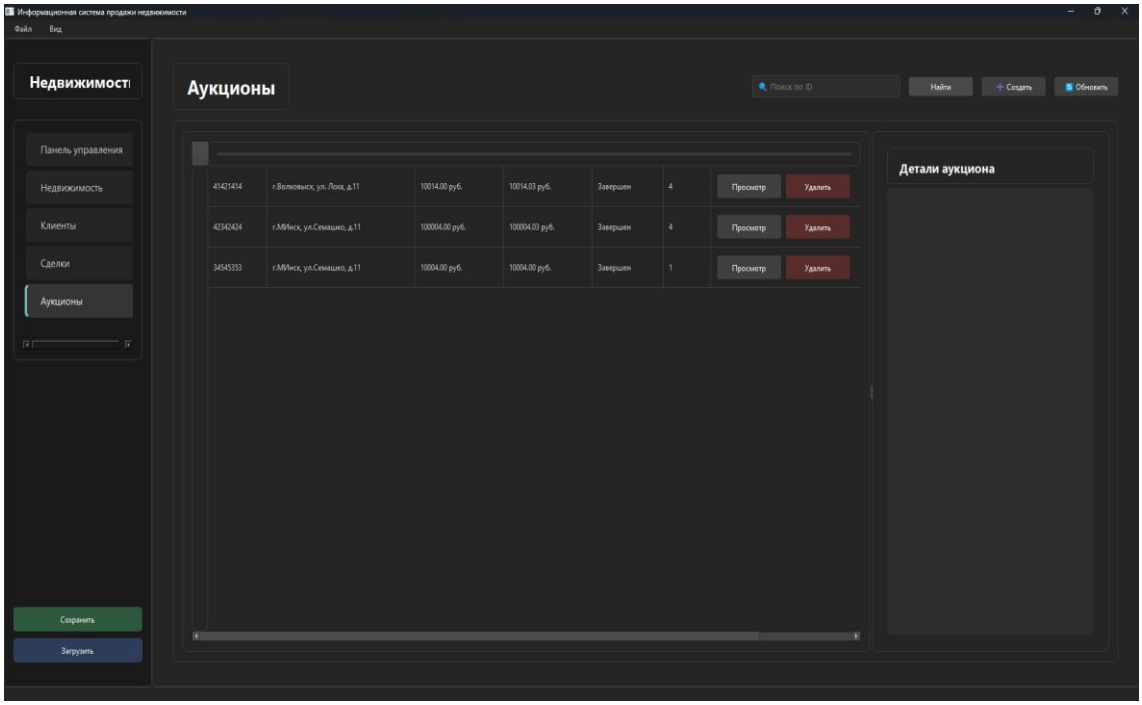

Рисунок 4.5 – Окно аукционов

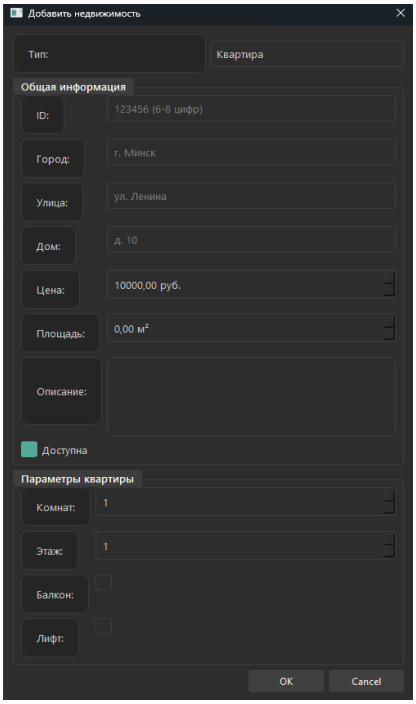

Рисунок 4.6 – Окно добавления недвижимости

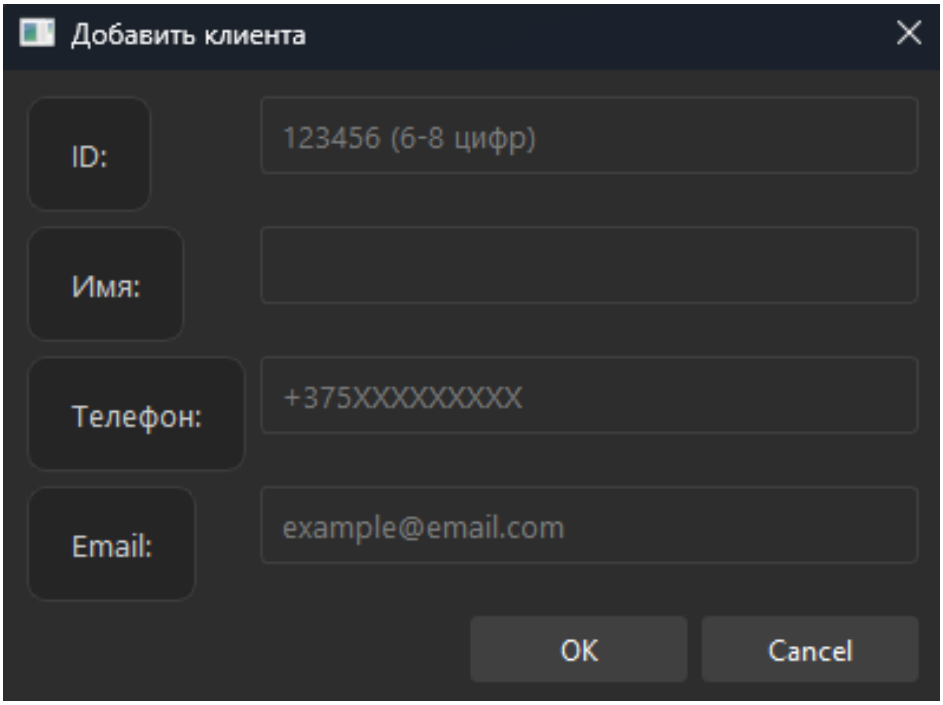

Рисунок 4.7 – Окно добавления клиента

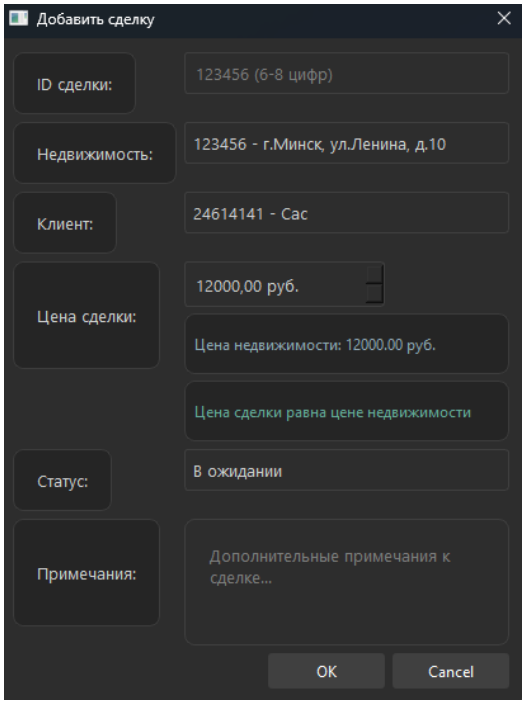

Рисунок 4.8 – Окно добавления сделки

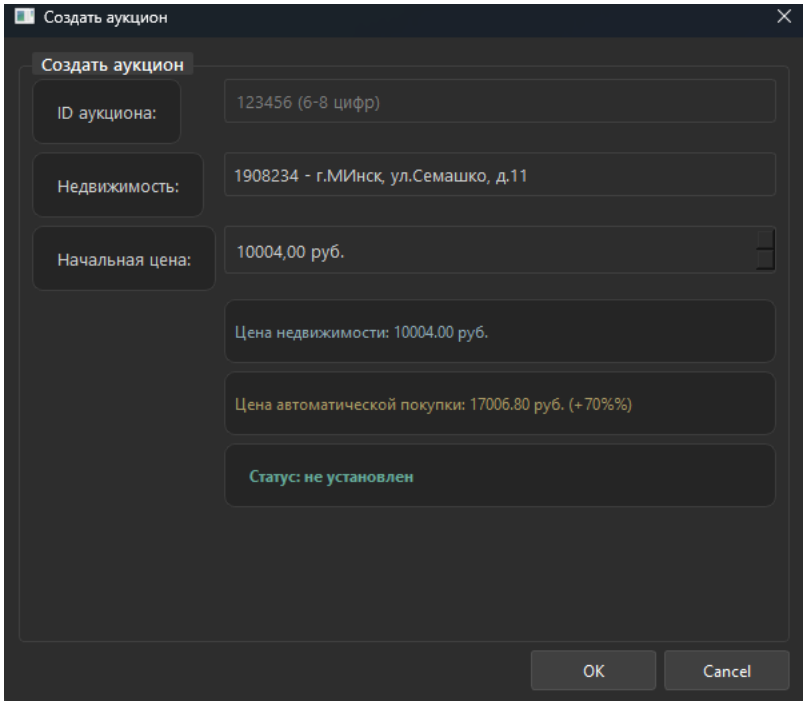

Рисунок 4.9 – Окно добавления аукциона

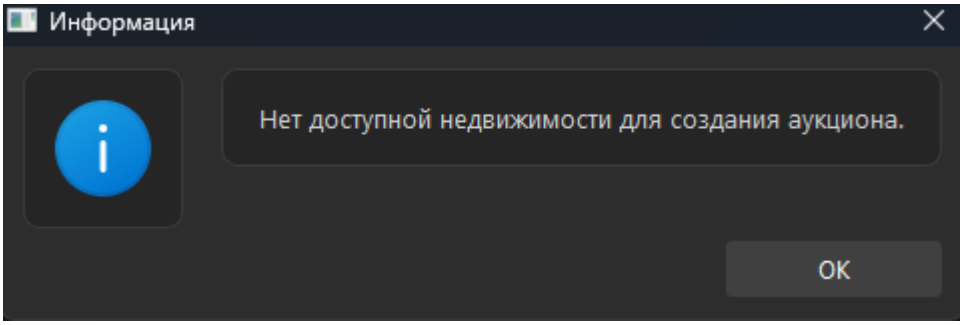

Рисунок 4.10 – Демонстрация работы валидации

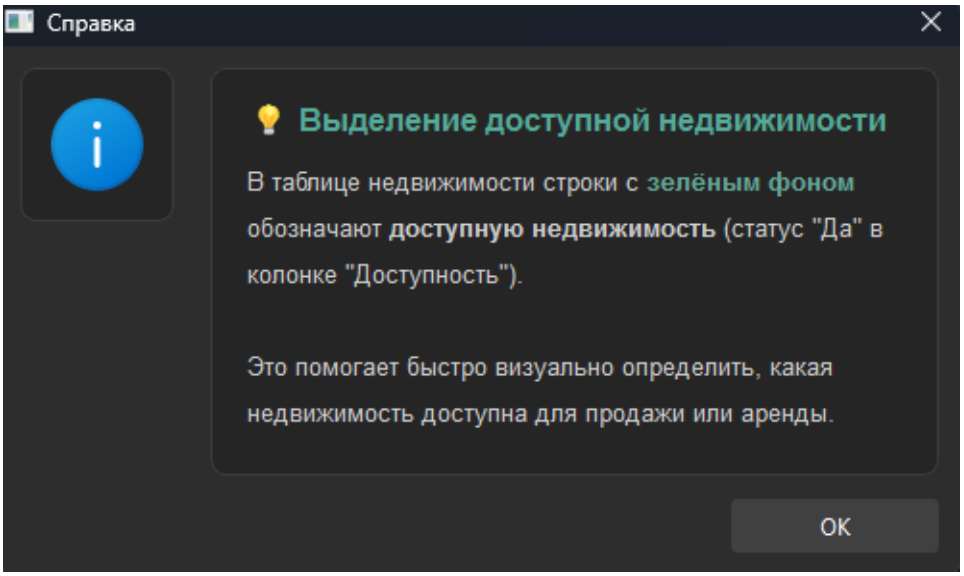

Рисунок 4.11 – Справка о недвижимости

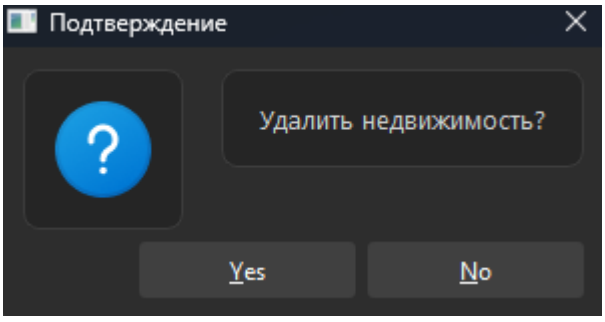

Рисунок 4.12 – Окно удаления недвижимости

ЗАКЛЮЧЕНИЕ

В работе разработано настольное приложение для управления агентством недвижимости, автоматизирующее ведение каталога объектов недвижимости, управление клиентской базой, контроль сделок купли-продажи и организацию аукционных торгов. Приложение реализовано на C++23 с использованием Qt6 для графического интерфейса.

Архитектура построена на принципах объектно-ориентированного программирования с четким разделением ответственности между компонентами. Сущности (Property, Apartment, House, CommercialProperty, Client, Transaction, Auction, Bid) моделируют основные объекты предметной области и инкапсулируют данные с валидацией через статические методы. Менеджеры (PropertyManager, ClientManager, TransactionManager, AuctionManager) обеспечивают управление коллекциями объектов, предоставляют методы поиска и фильтрации данных. Сервисы (FileManager) реализуют операции сохранения и загрузки данных в текстовом формате с обработкой ошибок. UI-компоненты отвечают за взаимодействие с пользователем и визуализацию данных через виджеты и диалоги.

Ключевая функциональность включает управление объектами недвижимости различных типов (квартиры, дома, коммерческая недвижимость) с отслеживанием их статуса доступности и специфических характеристик. Реализована система управления клиентской базой с валидацией контактных данных (телефон, email) и автоматическим отслеживанием даты регистрации. Система контроля доступности объектов предотвращает создание конфликтующих сделок и автоматически обновляет статусы при изменении состояния связанных операций. Организация аукционных торгов включает управление ставками участников, автоматическое завершение при достижении цены выкупа и создание сделок из завершенных аукционов.

Хранение данных организовано через текстовые файлы с разделителями, что обеспечивает человекочитаемость, простоту отладки и независимость от внешних систем. Данные сохраняются в отдельных файлах для каждого типа сущностей (properties.txt, clients.txt, transactions.txt, auctions.txt) с использованием разделителя '|' для полей. При загрузке выполняется обработка пустых строк и некорректных данных: некорректные записи пропускаются, но загрузка продолжается для остальных записей, что обеспечивает устойчивость к поврежденным данным. Обработка ошибок реализована через иерархию специализированных исключений (PropertyManagerException, ClientManagerException, TransactionManagerException, AuctionManagerException, FileManagerException) с информативными сообщениями для пользователя.

Графический интерфейс построен на базе Qt Widgets с использованием навигационного списка и стека виджетов для разделения функциональности. Детальная информация отображается в HTML-формате с визуализацией данных

через форматированные заголовки, таблицы и стилизованные блоки. Валидация входных данных выполняется на уровне сущностей через статические методы классов, обеспечивающие проверку диапазонов (цены от 10000 до 1000000000, площади положительные), форматов (идентификаторы 6-8 цифр, телефоны, email через регулярные выражения) и корректности адресных данных.

Результатом работы является функциональная система управления агентством недвижимости, готовая к практическому применению. Система демонстрирует эффективность применения современных возможностей C++23, объектно-ориентированного подхода с использованием наследования, полиморфизма и шаблонов, а также фреймворка Qt для создания нативных desktop-приложений. Архитектура обеспечивает расширяемость за счет четкого разделения слоев (сущности, менеджеры, сервисы, UI), надежность через многоуровневую валидацию и обработку исключений, а также удобство сопровождения кода благодаря модульной структуре и инкапсуляции функциональности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Луцик Ю. А. Объектно-ориентированное программирование на языке С++:учеб. пособие /Ю. А. Луцик, В. Н. Комличенко. – Минск: БГУИР, 2008.–266 с.

[2] Software Development Life Cycle (SDLC): Overview and Models – [Электронный ресурс]. – Режим доступа: https://www.ibm.com/topics/software-development-life-cycle – Дата доступа 20.10.2025

[3] Лафоре Р. Объектно-ориентированное программирование с С++/ 4-е издание М.:Питер, 2004. – 923 с.

[4] Qt 6 Widgets: перечень и описание классов виджетов – [Электронный ресурс]. – Адрес ресурса: https://doc.qt.io/qt-6/widget-classes.html – Дата доступа 20.11.2025

[5] cppreference: C++ Standard Library (справочник по стандартной библиотеке С++) – [Электронный ресурс]. – Адрес ресурса: https://en.cppreference.com/w/cpp/standard_library.html – Дата доступа 20.10.2025

ПРИЛОЖЕНИЕ А

Диаграмма классов

ПРИЛОЖЕНИЕ Б

Схема алгоритма создания сделки из завершенного аукциона

ПРИЛОЖЕНИЕ В

Схема алгоритма поиска объекта недвижимости по адресу

ПРИЛОЖЕНИЕ Г

Листинг кода

ПРИЛОЖЕНИЕ Д

Ведомость документов