МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки: 09.03.02 «Информационные системы и технологии»

Направленность: Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Приложение для аренды жилых помещений

Студент _____ Серскова Дарья Юрьевна
(подпись, дата) (ФИО полностью)

Руководитель _____ Кольчугина Е.А.
(подпись, дата) (фамилия, инициалы)

Нормоконтролёр _____ Такташкин Д.В.
(подпись, дата) (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от _____№_____)

Заведующий кафедрой _____ Козлов А.Ю.
(подпись) (фамилия, инициалы)

Работа защищена с отметкой _____ (протокол заседания ГЭК от _____№____)

Секретарь ГЭК _____ Попова Н.А.
(подпись) (фамилия, инициалы)

Пенза, 2025

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«20» декабря 2024 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент__Серскова Дарья Юрьевна_____гр._21ВИ1_факультета__ФВТ___
<Фамилия, имя, отчество полностью>
2.Тема работы__Приложение для аренды жилых помещений_____
_____
_____
_____
Тема утверждена приказом ПГУ №_____от "_____"_____20_____

2. Руководитель работы__д.т.н., доцент кафедры «МО и ПЭВМ» Кольчугина Е.А.____

3. Задание на работу (назначение разработки, исходные данные и т.п.)

1) *Разработать приложение для реализации сервиса аренды жилых помещений*
2) *Среда функционирования: web-браузер под управлением ОС Windows 10*
3) *Среда разработки: ОС Windows 10*
Языковые средства разработки: C#, HTML, CSS, JavaScript, SQL, UML
Инструментальные средства: MS Visual Studio 2022, Microsoft SQL Server NetBeans, Visual Paradigm 17.1.
Браузеры для тестирования: Google Chrome, Yandex, Mozilla Firefox
4) *Основные функции приложения:*
а) авторизация и регистрация пользователей с разграничением прав доступа;
б) просмотр и фильтрация объявлений по параметрам;
в) бронирование объектов недвижимости (пользователь);
г) добавление недвижимости в избранное (пользователь);
д) отправление заявки на размещение объявлений (пользователь);
е) удаление пользователей, назначение их администратором

ж) удаление, добавление, редактирование объявлений (администратор);

з) работа с комментариями (добавление пользователем, удаление тором)

5)Особенности: модуль бронирования с проверкой доступности и уведомлениями

4. Перечень подлежащих разработке вопросов

1) *Анализ предметной области и требований к системе;*
2) *Проектирование;*
3) *Реализация;*
4) *Отладка и тестирование;*
5) *Контроль качества приложения;*
6) *Оформление пояснительной записки.*

5. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	08.04.2025 – 14.04.2025	
2	Проектирование программных средств	20%	15.04.2025 – 25.04.2025	
3	Реализация программных средств	30%	26.04.2025 – 12.05.2025	
4	Тестирование программных средств	20%	13.05.2025 – 22.05.2025	
5	Оформление пояснительной записки	20%	23.05.2025 – 03.06.2025	

Дата выдачи задания «20» декабря 2024г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

Работу к защите допустить
Декан ФВТ _____ Л.Р. Фионова
(подпись, дата)

Реферат

Пояснительная записка содержит 162 листа, 34 рисунка, 14 таблиц, 17 листингов, 23 использованных источников и 7 приложений.

АРЕНДА ЖИЛЫХ ПОМЕЩЕНИЙ, МНОГОСЛОЙНАЯ МОНОЛИТНАЯ АРХИТЕКТУРА WEB-ПРИЛОЖЕНИЕ, UML, C#, CSS, HTML, JS, POSTGRESQL, REST, БАЗА ДАННЫХ.

Объект исследования – приложение для аренды жилых помещений, разработанное на AST.NET Core.

Цель выпускной квалификационной работы – разработать интуитивно понятное и стабильное приложение для аренды жилых помещений с функциями администрирования.

Технологии разработки – приложение построено с использованием HTML, CSS и JavaScript для клиентской части и C# с ASP.NET Core для серверной. Данные хранятся в SQL Server и передаются между клиентом и сервером через REST API в формате JSON. Структура приложения была спроектирована с использованием UML.

Среда исполнения – работает в веб-браузере под управлением Windows 10.

Результаты работы – в рамках выпускной квалификационной работы было создано приложение, предназначенное для сдачи жилых помещений в аренду. При реализации данного приложения были учтены и выполнены все предъявляемые требования.

Содержание

Введение 7

1 Анализ требований к поставленной задаче и предметной области 9

1.1 Постановка задачи 9

1.2 Анализ аналогов 10

1.3 Выбор методов и средств разработки 12

2 Проектирование приложения для аренды жилых помещений 16

2.1 Архитектура приложения 16

2.2 Проектирование структур данных 17

2.2.1 Концептуальная модель данных 17

2.2.2 Логическая модель базы данных 19

2.3 Проектирование алгоритмов и архитектуры приложения 23

2.4 Проектирование пользовательского интерфейса 28

2.5 Проектирование классов приложения для аренды жилых помещений… 43

3 Реализация приложения для аренды жилых помещений 46

3.1 Реализация базы данных 46

3.2 Реализация серверной части приложения 50

3.3 Реализация клиентской части приложения 56

4 Отладка и тестирование 62

4.1 Обоснование выбора режима тестирования 62

4.2 Тестирование приложения для аренды жилых помещений 63

4.2.1 Модульное тестирование 63

4.2.2 Функциональное тестирование 64

5 Контроль качества 66

5.1 Обоснование выбора метрик кода 66
5.2 Оценка качества программного кода..... 66
5.2.1 Метрики Холстеда..... 66
5.2.2 Метрика Чепина 68
6 Расчет бюджета разработки 70
6.1 Планирование разработки и оценка бюджета 70
Заключение 75
Приложение А (рекомендуемое) Графическая часть 79
Приложение Б (рекомендуемое) Спецификация атрибутов сущности и связи между ними 89
Приложение В (обязательное) Листинг серверной части 92
Приложение Г (обязательное) Листинг клиентской части 119
Приложение Д (рекомендуемое) Листинг модульных тестов 148
Приложение Е (рекомендуемое) Функциональное тестирование 154
Приложение Ж (рекомендуемое) Листинг функции CreateProperty..... 161

Введение

Современные технологические процессы информационных систем интенсивно внедряются в разные сферы жизни человека, включая рынок аренды жилых помещений. Уход от традиционных методов поиска жилища через объявления в печатных изданиях, на досках объявлений или по советам знакомых – стал шагом развития из-за стремительного распространения цифровых технологий.

По данным исследования PropTechBuzz, в 2024 году объем мирового рынка цифровых решений в недвижимости превысил 35 миллиардов долларов, а спрос на онлайн–сервисы для аренды не прекращает расти [1]. Это связано с тем, что пользователи ожидают:

– быстрый доступ к информации;
– поиск жилья по необходимым критериям;
– минимальная роль посредников.

Веб-сайты и мобильные прибавления для аренды жилища становятся главным каналом взаимодействия между арендодателями и арендаторами. Такие платформы дают возможность сберечь время, упростить выбор подходящего объекта, снизить опасности недостоверной информации и обеспечить комфортную коммуникацию между сторонами. Но почти многие существующие решения или перегружены функциональностью, или не имеют удобный интерфейс и гибких опций.

С ростом числа арендаторов – в особенности студенты, профессионалы на временных договорах и путешественники – вырастает потребность в быстрых и надежных инструментах для поиска жилья.

Цель дипломной работы заключается в разработке приложения для аренды жилых помещений, обеспечивающего интуитивно понятный интерфейс, размеренную работу и возможность администрирования контента.

В соответствии с целью были поставлены следующие задачи:

– провести анализ предметной области;

– спроектировать архитектуру приложения;

– реализовать серверную часть на языке C# с использованием ASP.NET и базы данных Microsoft SQL Server;

– разработать клиентскую часть с применением HTML, CSS и JavaScript;

– обеспечить взаимодействие компонентов;

– провести тестирование системы.

Для проектирования использовались диаграммы UML, реализация выполнялась в среде Visual Studio.

1 Анализ требований к поставленной задаче и предметной области

1.1 Постановка задачи

Рынок аренды жилых помещений интенсивно переходит в онлайн-формат [2]. Пользователи все чаще ищут предложения с помощью интернета и мобильных приложений, при этом они ждут удобства, простоты взаимодействия и доверенности информации.

В наше время в особенности нужны решения, позволяющие с легкостью публиковать и находить объявления, фильтровать их по данным параметрам.

Принимая это во внимание, необходима разработка приложения, предназначенного для размещения и поиска объявлений об аренде жилых помещений, включающие функции регистрации, авторизации, фильтрации, услуг бронирования и администрирования контента [3].

Целью дипломной работы является проектирование и реализация такого программного решения, обеспечивающего размеренную работу, простой интерфейс и гибкие возможности управления данными.

Для достижения поставленной цели необходимо выполнить следующие задачи:

– спроектировать клиент-серверную архитектуру системы, основанную на REST API [4];

– спроектировать архитектуру базы данных, содержащую информацию о пользователях, объектах недвижимости и ролях;

– разработать систему управления пользователями, включающую регистрацию, аутентификацию и контроль доступа;

– реализовать интерфейс, позволяющий пользователям просматривать, искать, бронировать объекты и управлять своими учетными записями;

– реализовать административную панель для управления контентом и пользователями;

– обеспечить взаимодействие между элементами системы;

– обеспечить тестирование для оценки стабильности и корректности работы приложения в различных условиях работы.

Для достижения поставленной цели нужно реализовать веб-приложение с использованием современных технологий программной разработки.

1.2 Анализ аналогов

Для более точного определения технических и пользовательских требований к разрабатываемому программному продукту был проведен анализ знаменитых веб-сервисов аренды жилых помещений, действующих на российской платформе. Цель анализа заключается в нахождении преимуществ и недочетов известных решений, а также определить, какие функции наиболее востребованы у конечных пользователей.

Большинство современных цифровых платформ выступают в роли посредников между арендодателями и арендаторами, предоставляя при всем этом инструменты поиска, фильтрации, публикации объявлений и коммуникации. Но не все решения нацелены на простоту использования или владеют интуитивно понятным интерфейсом. Поэтому в начале проектирования принципиально выявить те нюансы, которые нуждаются в улучшении при создании нового приложения.

В рамках анализа были изучены следующие онлайн-платформы:

– ЦИАН (clin.ru) – один из крупнейших интернет-сервисов с обширной базой недвижимостей по аренде и продажи, обладающее широким набором функций, включая фильтрацию, карту, поиск и личный кабинет [5];

– Авито Недвижимость (avito.ru) – популярная платформа объявлений, в том числе и по аренде жилья, благодаря широкому охвату аудитории и возможности размещать объявления [6];

– Яндекс.Недвижимость (realty.yandex.ru) – сервис от Яндекса для поиска недвижимости с картой, аналитикой цен и данными Росреестра [7];

– DomClick (domclick.ru) – платформа для сделок с недвижимостью от Сбербанка, предлагающая предварительное одобрение ипотеки и удобный интерфейс [8];

– СберАренда (sberarenda.ru) – это платформа для аренды недвижимости, которая предлагает удобные онлайн-инструменты, включая электронные сделки, цифровые договоры и проверку благонадежности арендодателей [9].

Для сравнения этих решений по основным параметрам была составлена таблица 1.

Таблица 1 – Сравнение функциональности существующих сервисов аренды

Сервис	**Основные функции**	**Особенности интерфейса**
ЦИАН	Поиск, фильтрация, личный кабинет, карта	Сложный, перегруженный
Авито Недвижимость	Универсальный поиск, фильтры, личные сообщения	Многофункциональный, запутанный
Яндекс.Недвижимость	Интеграция с картой, аналитика, официальные данные	Простой, но перегруженный данными
DomClick	Поиск, ипотека, сделки	Упрощенный, но фокус на покупку
СберАренда	Верификация, онлайн-договор, защищенная сделка	Современный, лаконичный

В таблице 2 проведено сопоставление перечисленных решений с разрабатываемым приложением. Приложение отличается минималистичным интерфейсом, ориентированным исключительно на аренду, расширенными административными возможностями и поддержкой предварительного бронирования жилья.

Таблица 2 – Сравнение разработанного приложения с существующими аналогами

Критерий	**ЦИАН**	**Авито Недвижимость**	**Яндекс.Недвижимость**	**Разрабатываемое приложение**
Упор на аренду	Частично	Нет	Частично	Да
Простота интерфейса	Средняя	Низкая	Средняя	Высокая
Поддержка бронирования	Нет	Нет	Нет	Да
Админ-панель	Нет	Нет	Нет	Да
Посредники	Частично	Да	Нет	Да

Результаты проведенного анализа показали, что существующие решения для аренды имеют такие недостатки: они либо сложны в использовании, либо не специализированы на аренде, либо не имеют административную панель. Это подтверждает необходимость и актуальность разработки специализированного приложения, которое будет интуитивно понятным, функциональным и ориентированным на прямое взаимодействия без посредников.

1.3 Выбор методов и средств разработки

На основе поставленных целей и проведенного сравнительного анализа с существующими аналогами были определены следующие функциональные требования:

– регистрация новых пользователей;

– авторизация с учетом различных прав и ролей;

– просмотр списка доступной недвижимости;

– поиск недвижимости с использованием фильтров по цене, типу, количеству комнат и местоположению;

– возможность сохранять объявления в избранное;

– бронирование объявлений недвижимости;

– подача заявок на публикацию объявлений;

– публикация комментариев к объявлениям;

– просмотр и редактирование личной информации в профиле;

– доступ администратора ко всем объявлениям;

– добавление, редактирование и удаление объявлений администратором;

– одобрение или отклонение заявок администратором на размещение объявления;

– удаление комментариев пользователей администратором;

– управление учетными записями пользователей администратором: назначение роли администратора, удаление учетных записей.

Нефункциональные требования:

– совместимость и стабильная работа на операционных системах Windows 10 и выше;

– интерфейс приложения должен корректно отображаться и функционировать во всех актуальных версиях браузеров;

– приложение должно поддерживать стабильную работу при подключении нескольких пользователей одновременно;

– обеспечение безопасности хранения пользовательских данных;

– интерфейс должен быть понятным и простым для восприятия;

– система должна обеспечивать разграничение прав доступа, исключающая доступ пользователей к административным функциям.

Для наглядного представления взаимодействия различных типов пользователей с системой была разработана диаграмма вариантов использования, представленная на рисунке 1 и в приложении А.

Диаграмма вариантов использования отражает структуру взаимодействия с системой в зависимости от роли пользователя. Каждый элемент диаграммы позволяет определить, какие действия доступны конкретному типу.

Рисунок 1 – Диаграмма вариантов использования

Диаграмма предоставляет функционал для двух типов ролей: администратора и пользователя. Обычный пользователь может: «Авторизоваться», «Зарегистрироваться», «Изменить данные в профиле», «Подать заявку на размещение объявления», «Просмотреть доступные объявления», «Забронировать недвижимость», «Оставить комментарий» и «Добавить объявление в избранное». Администратор может: «Авторизоваться», «Добавить объект недвижимости», «Редактировать объект недвижимости», «Рассмотреть заявку на размещение объявления», «Удалить комментарий», «Удалить объект недвижимости», «Просмотреть все объявления недвижимости», «Удалить пользователя», «Просмотреть всех пользователей», «Изменить данные в профиле» и «Назначить администратором пользователя».

Таким образом, диаграмма вариантов использования показывает функциональные возможности приложения. Наглядное представление сценариев упрощает выявление возможных несоответствий и помогает обеспечить полноту реализации всех предусмотренных функций.

Приложение разрабатывалось с учетом требований к безопасности, стабильности и удобству использования. Клиент-серверная архитектура с REST API обеспечивает независимость компонентов и модульность.

Использование ASP.NET Core, C# и Microsoft Visual Studio для серверной части позволило эффективно реализовать бизнес-логику и обеспечить качественное тестирование и отладку.

Клиентская часть, разработанная на HTML, CSS и JavaScript, обеспечила создание адаптивного и интуитивно понятного интерфейса, не требующая больших затрат ресурсов браузера.

Инструмент Visual Paradigm поддерживающий создание диаграмм UML, помог визуализировать структуру приложения и логику взаимодействия компонентов. Разработанные диаграммы с использованием данного инструмента легли в основу детального проектирования системы.

2 Проектирование приложения для аренды жилых помещений

2.1 Архитектура приложения

Согласно поставленным требованиям к программному обеспечению, была разработана диаграмма коммуникаций, отображающая процесс взаимодействия основных компонентов приложения. Разработанная диаграмма представлена на рисунке 2 и в приложении А.

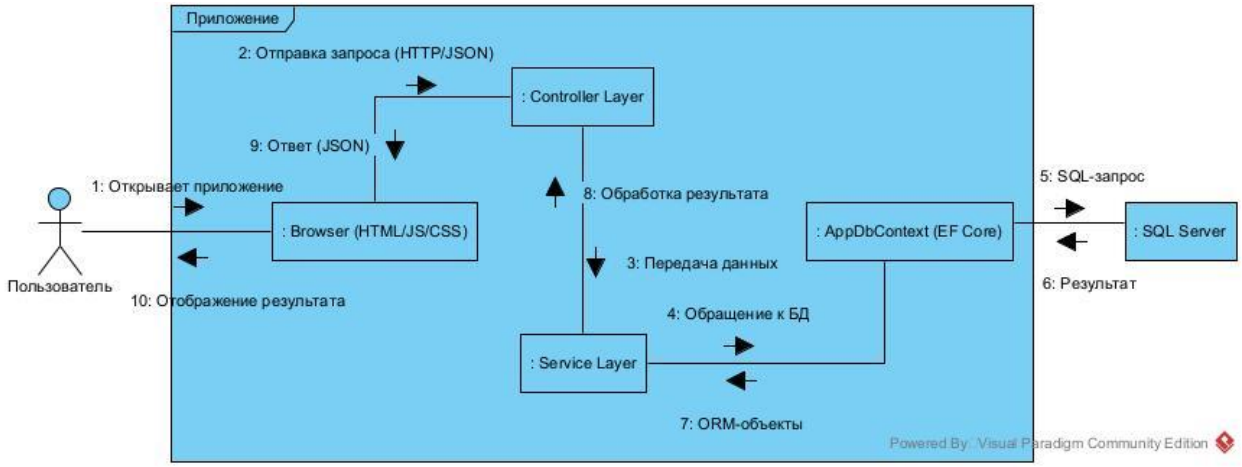

Рисунок 2 – Диаграмма коммуникаций

Данная диаграмма демонстрирует структуру приложения и отражает принципы организации взаимодействия между его ключевыми элементами. Пользователь взаимодействует с приложением через клиентскую часть. При выполнении действий, таких как авторизации, просмотр объявлений, добавление комментариев или отправка заявок на размещение объявления, клиент отправляет HTTP-запросы к серверу.

Серверная часть приложения реализована на платформе ASP.NET Core и построена по принципу многослойной монолитной архитектуры.

В процессе работы приложение взаимодействует с базой данных, где хранятся сведения о пользователях, объектах недвижимости, комментариях, заявках и другой информации. Взаимодействие между слоями организовано последовательно: контроллер вызывает сервис, сервис обращается к контексту данных, после чего результат передается обратно пользователю.

Такая архитектура обеспечивает модульность, упрощенное тестирование и масштабируемость приложения.

2.2 Проектирование структур данных

Проектирование базы данных является одним из основных этапов разработки приложения, потому что структура данных определяет логическую модель системы, обеспечивает связность, целостность и корректность хранения информации. В дипломной работе используется реляционная модель, реализованная средствами ORM – инструмента Entity Framework Core, что позволяет взаимодействовать с базой данных через объекты .Net.

2.2.1 Концептуальная модель данных

Концептуальная модель представляет собой отображение реальной области, для которой разрабатывается база данных [10]. Главная задача состоит в определении основных объектов системы, их свойства и взаимосвязи.

Концептуальная модель базы данных представлена на рисунке 3 и в приложении А.

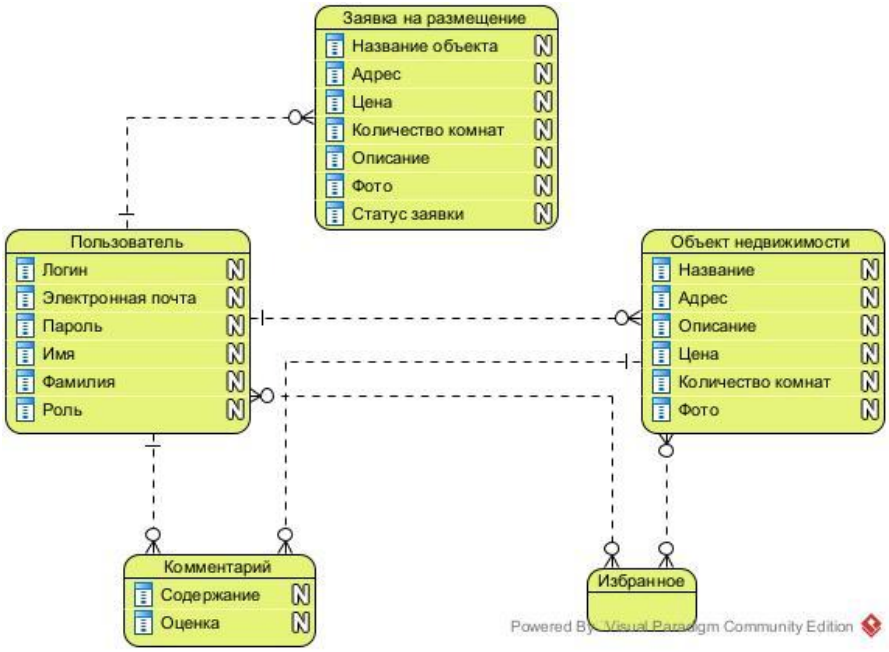

Рисунок 3 – Диаграмма концептуальной модели

В разрабатываемом приложении выделены следующие основные сущности:

– пользователь (User) – физическое лицо, проходящее регистрацию в системе для размещения или аренды объектов недвижимости, также он может выступать в роли администратора или обычного пользователя, что влияет на доступ к возможностям системы;

– недвижимость (Property) – дом, квартира или другая жилая площадь, доступная для аренды, объекты добавляются администратором и могут быть забронированы другими пользователями;

– комментарий (Comment) – текстовое мнение пользователя о конкретном объекте недвижимости, каждый отзыв привязан к определенному пользователю и объекту;

– избранное (Favorite) – механизм сохранения понравившихся пользователям объектов, что позволяет быстро находить ранее отмеченные варианты;

– запрос на публикацию (PropertyRequest) – запрос пользователя на добавление нового объявления в систему, это необходимо для отображения в общем списке объявлений требуется подтверждение администратором.

Между этими сущностями установлены следующие связи:

– один пользователь может подать множества запросов, оставить большое количество отзывов, добавить в избранное несколько объектов;

– один объект недвижимости может быть прокомментирован многими пользователями и добавлен в избранное много раз;

– комментарий всегда связан с одним пользователем и одним объектом;

– запрос на публикацию относится к одному пользователю и не связан с конкретным объектом до момента одобрения.

Представленная модель не включает в себя технических подробностей (типы данных, ключи) и служит для первичного представления архитектуры системы.

2.2.2 Логическая модель базы данных

Абстрактное представление структуры данных, используемое для проектирования баз данных, называется логической моделью. Она определяет организацию данных и их взаимосвязи, не привязываясь к конкретной системе управления базами данных [11].

Цель построения – создание надежной, целостной и масштабируемой системы хранения информации, оптимизированной для работы с Microsoft SQL Server и ORM Entity Framework Core.

Логическая модель базы данных представлена на рисунке 4 и в приложении А.

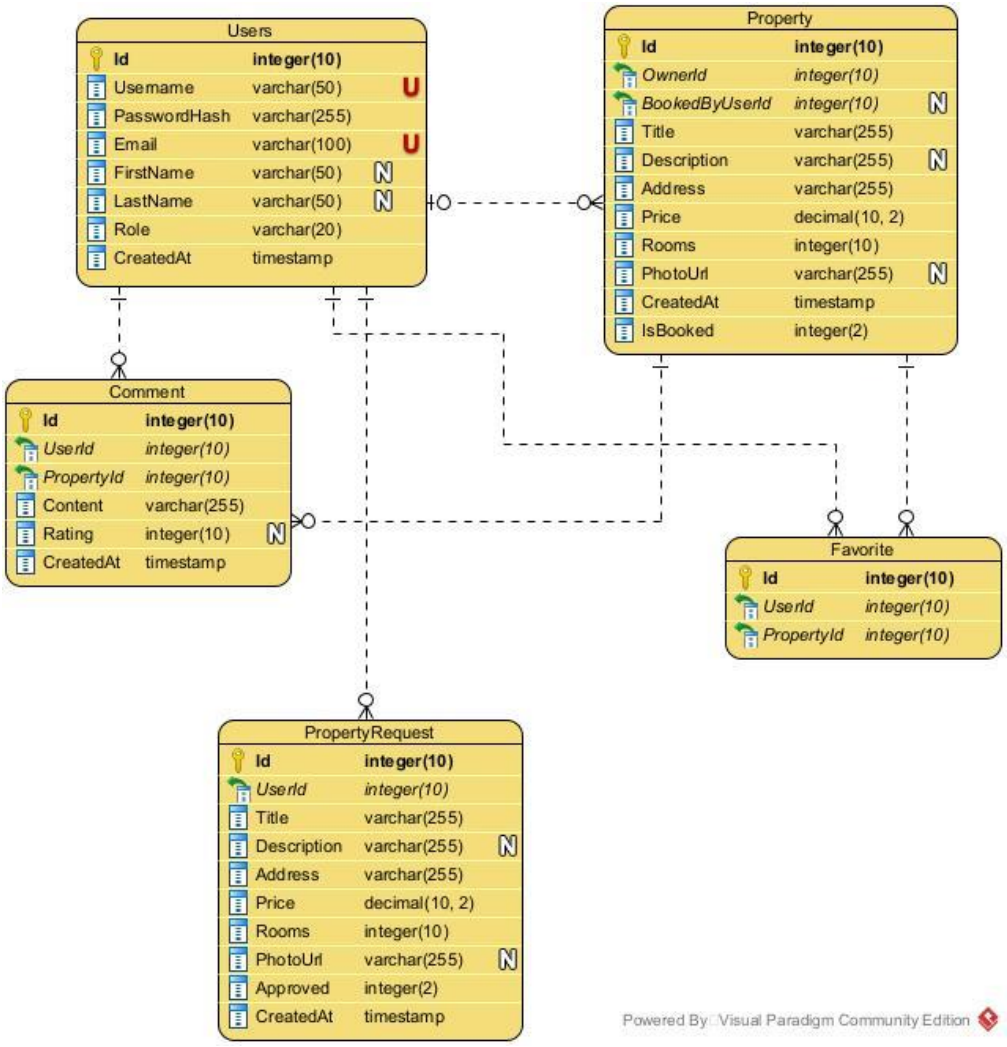

Рисунок 4 – Диаграмма логической модели

Архитектура базы данных организована с помощью пяти основных таблиц. Каждая таблица спроектирована для управления отдельной частью логики приложения. Детальное описание этих таблиц представлено в таблицах 3 – 7 и в приложении Б.

Каждая таблица в логической модели отражает конкретный аспект предметной области и спроектирована с учетом требований к функциональности приложения. Например, таблица Users отвечает за хранение информации о зарегистрированных пользователях, тогда как Properties описывает объекты недвижимости, доступные для аренды. Таблицы Comments, Favorites и PropertyRequests реализуют дополнительные функции, такие как комментарии, добавление в избранное и подача заявок. Наличие внешних ключей и продуманная структура атрибутов позволяют эффективно организовать связи между сущностями, обеспечивая как масштабируемость, так и удобство при выполнении запросов.

Таблица 3 – Спецификация атрибутов сущности Users

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID пользователя	Yes	No
Username	VARCHAR(50)	Yes	Имя пользователя	No	No
Email	VARCHAR(100)	Yes	Адрес электронной почты	No	No
Password	VARCHAR(255)	Yes	Пароль пользователя	No	No
FirstName	VARCHAR(50)	No	Имя пользователя	No	No
LastName	VARCHAR(50)	No	Фамилия пользователя	No	No
Role	VARCHAR(20)	Yes	Роль	No	No
CreatedAt	DATETIME2	Yes	Дата регистрации	No	No

Таблица 4 – Спецификация атрибутов сущности Properties

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID объекта недвижимости	Yes	No
Title	VARCHAR(255)	Yes	Заголовок объявления	No	No
Address	VARCHAR(255)	Yes	Адрес объекта	No	No
Description	VARCHAR(255)	No	Описание недвижимости	No	No
Price	DECIMAL(10,2)	Yes	Стоимость аренды	No	No
Rooms	INTEGER	Yes	Количество комнат	No	No

Продолжение таблицы 4

Name	Datatype	Required	Definition	Is PK	Is FK
PhotoUrl	VARCHAR(255)	No	Ссылка на изображение объекта	No	No
IsBooked	BOOLEAN	Yes	Флаг, указывающий, забронирован ли объект	No	No
OwnerId	INTEGER	Yes	Внешний ключ администратора, разместившего объект	No	Yes
BookedByUserId	INTEGER	No	Внешний ключ на пользователя, забронировавшего объект	No	Yes
CreatedAt	DATETIME2	Yes	Дата размещения объявления	No	No

Таблица 5 – Спецификация атрибутов сущности Comments

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID комментария	Yes	No
UserId	INTEGER	Yes	Внешний ключ на пользователя	No	Yes
PropertyId	INTEGER	Yes	Внешний ключ на объект недвижимости	No	Yes
Content	VARCHAR(255)	Yes	Текст комментария	No	No
Rating	INTEGER	No	Оценка от 1 до 5	No	No
CreatedAt	DATETIME2	Yes	Дата размещения объявления	No	No

Таблица 6 – Спецификация атрибутов сущности Favorites

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID записи избранного	Yes	No
UserId	INTEGER	Yes	Внешний ключ на пользователя	No	Yes
PropertyId	INTEGER	Yes	Внешний ключ на объект недвижимости	No	Yes

Таблица 7 – Спецификация атрибутов сущности PropertyRequests

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID записи избранного	Yes	No
UserId	INTEGER	Yes	Внешний ключ на пользователя	No	Yes
Title	VARCHAR(255)	Yes	Заголовок объявления	No	No
Address	VARCHAR(255)	Yes	Адрес объекта	No	No
Description	VARCHAR(255)	No	Описание недвижимости	No	No

Продолжение таблицы 7

Name	Datatype	Required	Definition	Is PK	Is FK
Price	DECIMAL(10,2)	Yes	Стоимость аренды	No	No
Rooms	INTEGER	Yes	Количество комнат	No	No
PhotoUrl	VARCHAR(255)	No	Ссылка на изображение объекта	No	No
Approved	BOOLEAN	Yes	Флаг утверждения заявки	No	No
CreatedAt	DATETIME2	Yes	Дата размещения объявления	No	No

Связи между таблицами организованы на основе внешних ключей и обеспечивают логическую целостность системы, они представлены в таблице 8 и в приложении Б. Такая структура позволяет обеспечить согласованность данных между таблицами. Также облегчается выполнение операций выборки и обновления данных, что повышает эффективность системы.

Таблица 8 – Связи между сущностями базы данных

From Table	To Table	FK Field	Cardinality	On Delete Behavior	Nullable
Properties	Users	OwnerId	Many to One	CASCADE	No
Properties	Users	BookedByUserId	Many to One	SET NULL / Custom	Yes
Comments	Users	UserId	Many to One	RESTRICT	No
Comments	Properties	PropertyId	Many to One	CASCADE	No
Favorites	Users	UserId	Many to One	RESTRICT	No
Favorites	Properties	PropertyId	Many to One	RESTRICT	No
PropertyRequests	Users	UserId	Many to One	CASCADE	No

Логическая модель играет важную роль при проектировании структуры базы данных, так как она позволяет заранее определить, какие объекты будут храниться в системе, каким образом они связаны между собой и какие атрибуты являются критически важными для функционирования приложения. Такой уровень абстракции удобен для согласования требований между разработчиками, аналитиками и заказчиком, а также служит основой для автоматической генерации базы данных средствами ORM. Благодаря использованию логической модели удаётся сократить количество потенциальных ошибок на этапе реализации и упростить дальнейшую поддержку системы. Кроме того, четко определённые связи между сущностями

обеспечивают целостность и непротиворечивость данных, особенно при выполнении сложных запросов или модификации информации в условиях многопользовательской среды.

2.3 Проектирование алгоритмов и архитектуры приложения

При разработке логики взаимодействия компонентов приложения была применена многослойная монолитная архитектура, обеспечивающая модульность, организованность и масштабируемость без излишней сложности [12]. Такое решение обеспечивает логическое разделение ответственности между слоями и упрощает сопровождение кода.

Приложение включает следующие основные слои:

– интерфейс пользователя (UI) – фронтенд, разработанный с использованием веб-технологий (HTML, CSS, JavaScript), отвечающий за визуализацию данных и отправку HTTP-запросов к серверной части;

– контроллеры (Controllers) – этот слой обрабатывает входящие HTTP-запросы, осуществляет проверку корректности передаваемых данных и делегирует дальнейшую обработку соответствующим сервисам;

– слой бизнес-логики (Services) – реализует основные бизнес-процессы приложения, такие как бронирование, управление контентом, фильтрация данных, контроль доступа.

Слой бизнес-логики содержит сервисные классы: UserService (управление пользователями) и PropertyService(управление объектами недвижимости).

В PropertyService реализованы шаблоны проектирования:

– Factory Method – используется для создания объектов недвижимости разных типов (HouseFactory, ApartmentFactory) [13];

– Strategy – применяется для инкапсуляции логики различных сценариев бронирования [14];

– Observer – применяется для передачи уведомлений между компонентами при изменении характеристик объекта недвижимости, когда данные об объекте обновляются, механизм уведомления PropertyChangeNotifier

сбрасывается, он оповещает заинтересованные элементы системы, такие как ConsolePropertyObserver [15];

– модели (Models) – включает в себя классы, описывающие сущности предметной области, а также конфигурацию контекста базы данных.

Вся система реализована в виде единого монолитного приложения с REST-архитектурой, что обеспечивает масштабируемость и производительность при умеренной нагрузке на сервер.

Сценарий работы механизма бронирования:

– пользователь нажимает кнопку «Забронировать»;

– отправляется POST-запрос на сервер по маршруту /api/properties/book/{id};

– контроллер обрабатывает запрос, проверяя права доступа и корректность идентификатора;

– сервис бронирования проверяет доступность объекта (IsBooked = false);

– в случае доступности вносятся изменения в базу данных: IsBooked = true, BookedByUserId = currentUser.Id;

– изменения сохраняются через DbContext;

– пользователь получает уведомление об успешном завершении бронирования.

Примеры реализованных алгоритмов:

– процесс регистрации с проверкой уникальности: проверяется, существует ли уже пользователь с заданным логином или электронной почтой, если совпадения отсутствуют, то создается учетная запись с ролью user;

– система фильтрации объявлений: на основе пользовательского запроса извлекаются необходимые параметры фильтрации и генерирует SQL-запрос с динамическими условиями, в результате пользователь получает список наиболее подходящих объектов;

– алгоритм модерации контента для администратора, позволяющий изменять роли пользователей, подтверждать или отклонять заявки на

публикацию объявлений, получать доступ к спискам объектов и комментариям, а также выполнять операции редактирования или удаления данных.

Архитектура серверной части представлена на рисунке 5 и в приложении А с помощью структурно-функциональной диаграммы, отображающей основные компоненты системы.

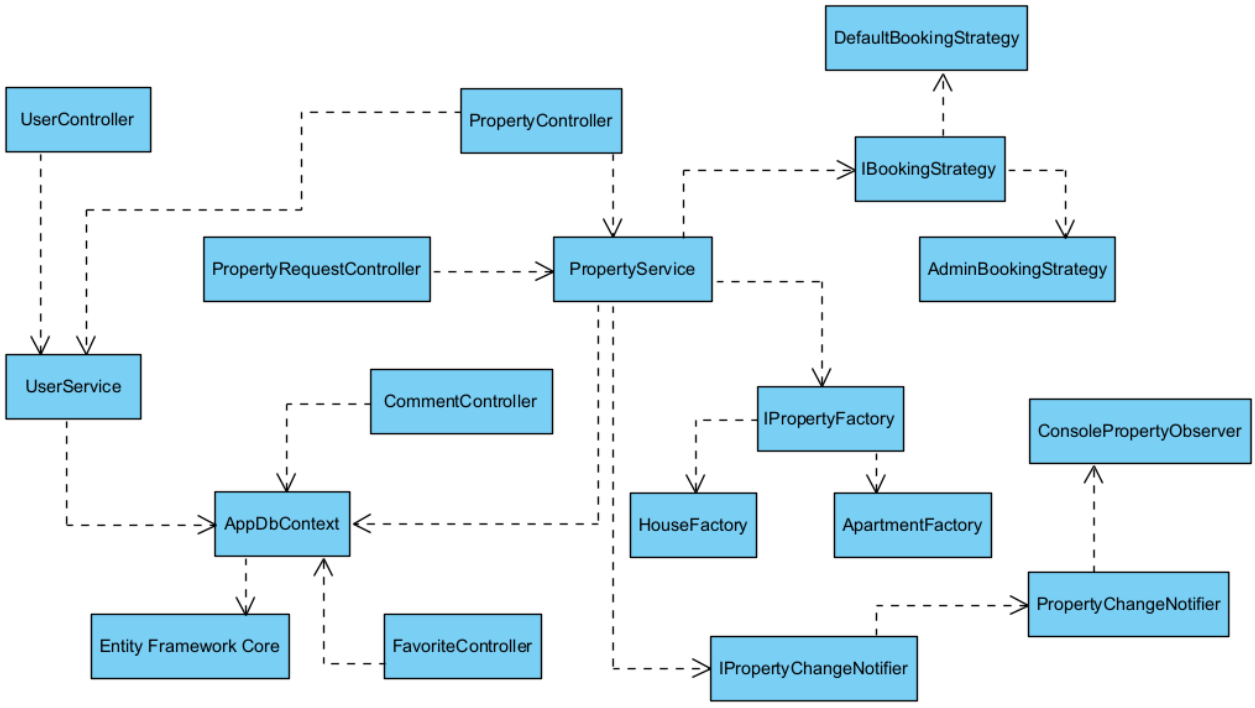

Рисунок 5 – Структурно-функциональная диаграмма архитектуры серверной части

В реализации архитектуре используются паттерны проектирования, такие как стратегия, фабрика и наблюдатель, а также принципы многоуровневой архитектуры, что обеспечило гибкость и масштабируемость системы. Каждый компонент отвечает за строго определенный набор функций, что упрощает модификацию и сопровождение кода.

Внедрение новых типов недвижимости реализуется через паттерн фабрика. Настройка различных сценариев бронирования – через паттерн стратегия. Отслеживание изменений свойств – через паттерн наблюдатель.

Это решение снижает связанность между модулями, что упрощает расширение функциональности без необходимости значительных изменений существующего кода.

Структурно-функциональная диаграмма позволила показать архитектуру, которая предоставляет устойчивую основу для дальнейшего развития системы, особенно в условиях активного пользовательского использования. Контроллеры взаимодействуют с сервисами, которые, в свою очередь, управляют доступом к данным через контекст базы данных. Это обеспечивает четкое разграничение ответственности и упрощает тестирование отдельных компонентов.

При разработке учитывались требования к безопасности, стабильности и производительности, что позволило реализовать устойчивую к ошибкам и отказам серверную часть. Логика обработки исключений и валидации данных встроена на уровне сервисов, что предотвращает некорректные операции. Такой подход важен при работе с пользовательскими действиями, где необходим строгий контроль над целостностью данных.

Диаграмма последовательности, показывающая взаимодействия между компонентами системы в процессе бронирования объекта недвижимости, представлена на рисунке 6 и в приложении А. На ней представлен динамический обмен сообщениями между объектами и помогает представить порядок вызовов между участниками процесса, что особенно важно при разработке и откладке.

Диаграмма последовательности позволяет выявить избыточные зависимости, ошибки в распределении функций и в работе системы. Так же способствует более точной реализации бизнес-логики и упрощает командную работу над системой. Она наглядно иллюстрирует пошаговый порядок взаимодействия между пользователем, контроллером, сервисом и базой данных в процессе бронирования. Диаграмма визуализирует логику обработки запросов, контроль состояния и уведомления, что облегчает понимание и проверку корректности бизнес-процессов.

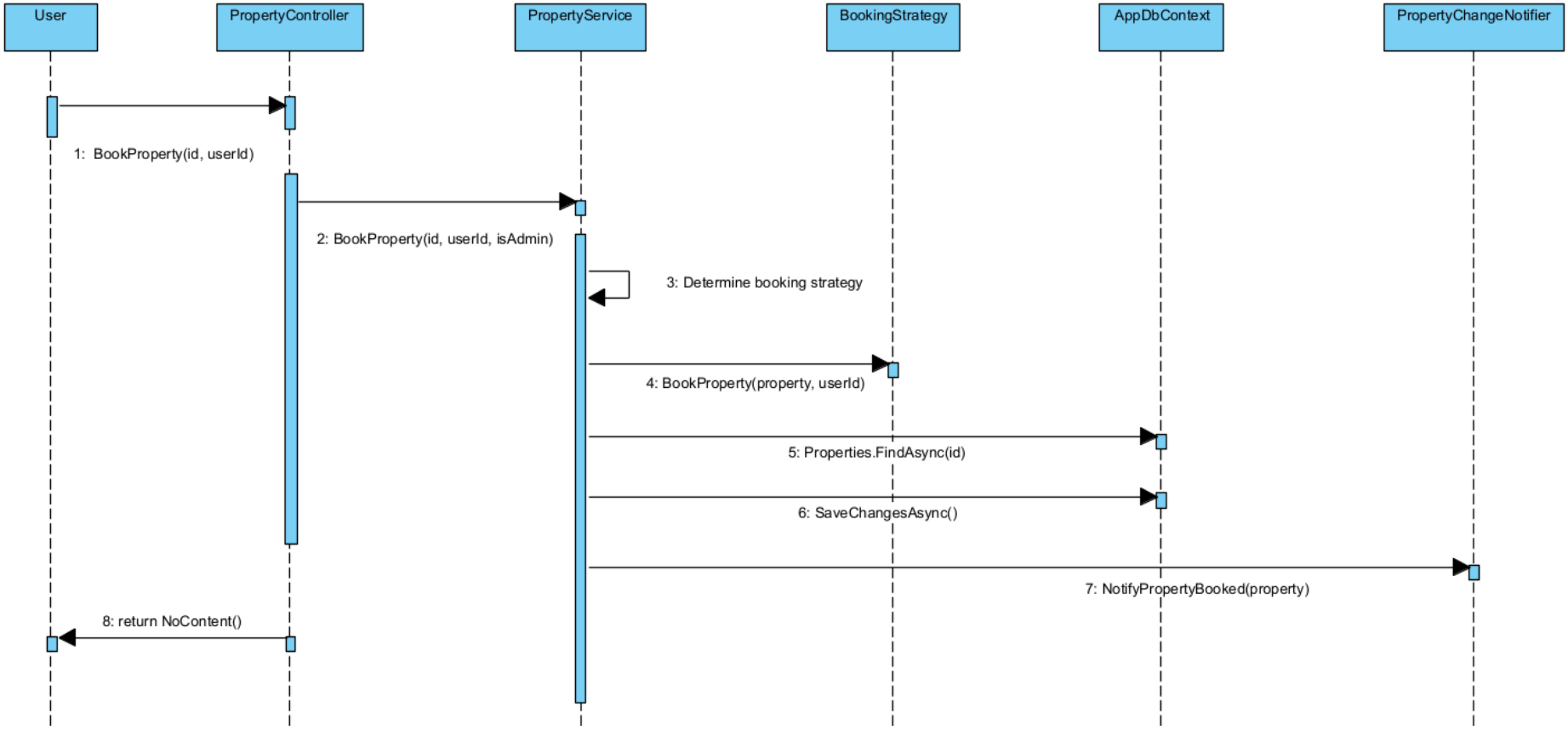

Рисунок 6 – Диаграмма последовательности

Диаграмма показывает последовательность: пользователь инициирует бронирование через интерфейс. Запрос поступает в PropertyController, где проходит первичная проверка. Затем PropertyService обрабатывает сценарий, выбирает подходящую стратегию бронирования (IBookingStrategy) и, при успешной проверке, вносит изменение в базу данных через DbContext. Завершающим этапом становится уведомление пользователя через NotificationService.

Таким образом, диаграмма коммуникаций дополняет представление о внутренней структуре приложения.

2.4 Проектирование пользовательского интерфейса

Этап проектирования пользовательского интерфейса играет ключевую роль в обеспечении доступности системы для конечного пользователя. Основная задача – реализовать понятную и удобную логику взаимодействия.

В соответствии с требованиями был спроектирован интерфейс клиентской части в виде веб-приложения.

Взаимодействие пользователя с приложением осуществляется через формы, кнопки и интерактивные элементы, реализованные с учетом стандартов доступности и пользовательского опыта. На рисунке 7 представлена форма авторизации.

Для обеспечения комфортного и интуитивно понятного взаимодействия с системой использовались современные подходы к дизайну интерфейсов. Все элементы управления выполнены таким образом, чтобы минимизировать количество действий пользователя и снизить вероятность ошибок при вводе данных. Особое внимание уделено адаптивности интерфейса, это позволяет корректно отображать страницы на устройствах с разными размерами экранов.

Кроме того, интерфейс сделан так, чтобы его было легко менять и добавлять новые функции в будущем. Это поможет системе развиваться и подстраиваться под нужды пользователей.

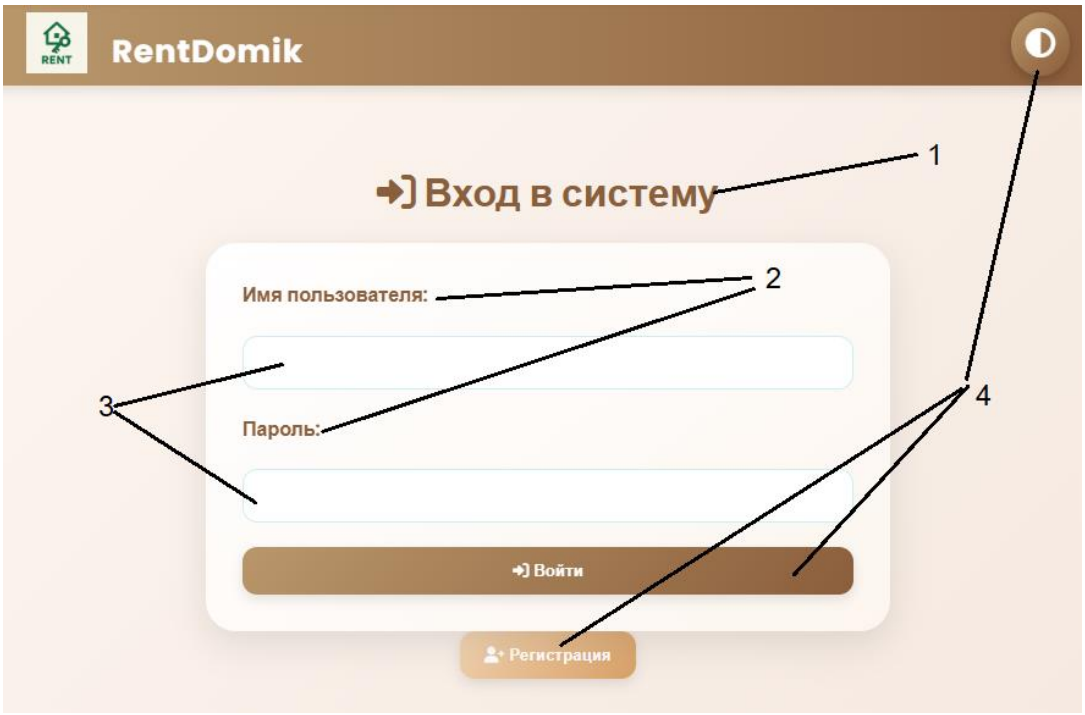

Рисунок 7 – Форма авторизации

Данная форма включает в себя:

1 – заголовок формы с текстом «Вход в систему», компонент <h1>;

2 – подписи к текстовым полям, поясняющие назначение каждого поля, компонент <label>;

3 – поля для ввода имени пользователя и пароля, компонент <input>;

4 – кнопки, позволяющие выполнить вход или перейти к регистрации, компонент <button>.

Иконка в правом верхнем углу отвечает за смену темы. Вид формы в темной теме представлен на рисунке 8.

Реализации функции переключения между светлой и темной темами позволяет пользователям выбирать наиболее комфортный для них визуальный режим работы с приложением. Это важно для снижения нагрузки на глаза при длительном использовании и в условиях разного освещения. Темная тема способствует уменьшению яркости экрана, что помогает избежать быстрой усталости зрения, а светлая – обеспечивает привычный большинству пользователей интерфейс.

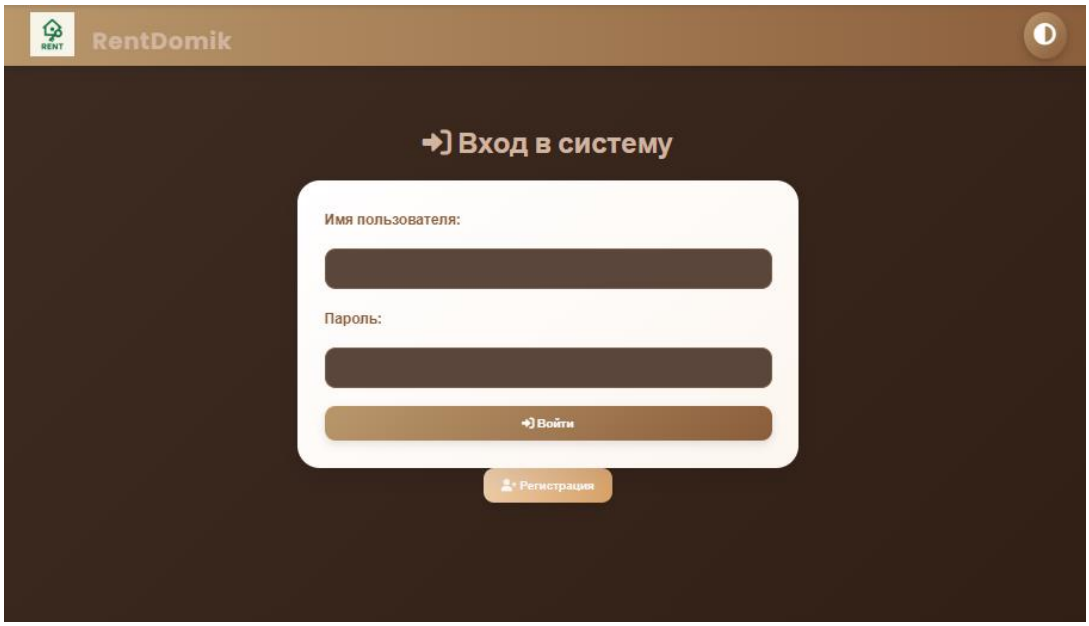

Рисунок 8 – Форма авторизации в темной теме

Форма регистрации представлена на рисунке 9.

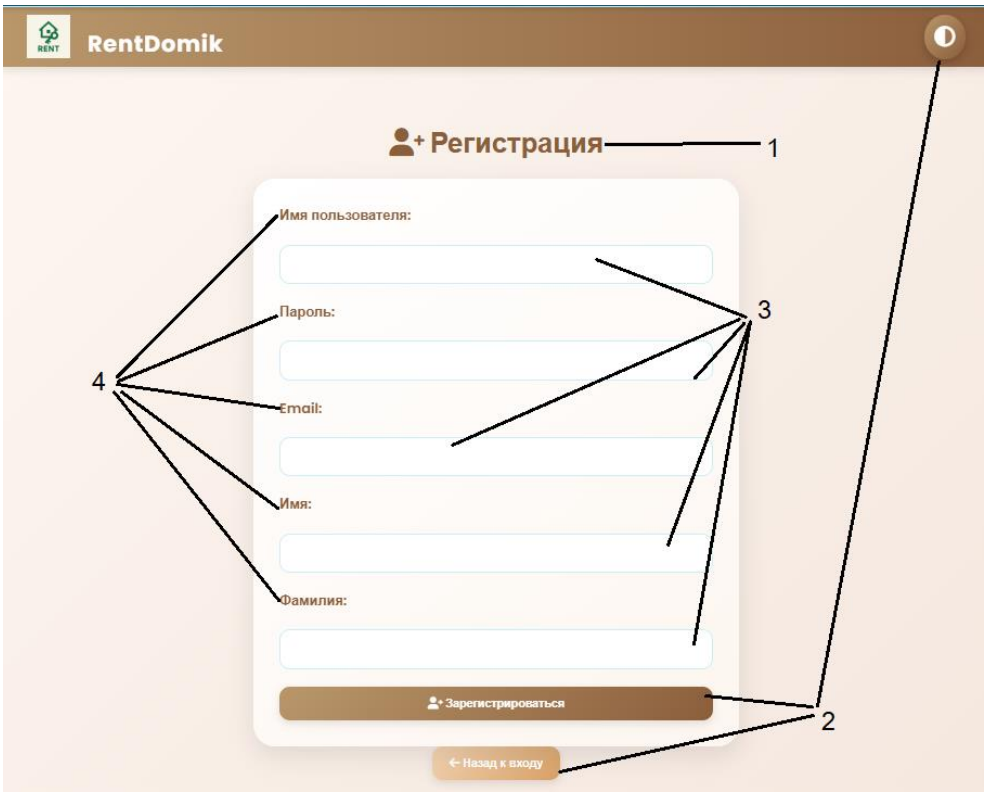

Рисунок 9 – Форма регистрации

Данная форма включает в себя:

1 – заголовок формы с текстом «Регистрация», компонент <h1>;

2 – кнопки для подтверждения регистрации и возврата на форму авторизации, компонент <button>;

3 – поля для ввода данных, компонент <input>;

4 – подписи к текстовым полям, поясняющие какую информацию надо ввести, компонент <label>.

После ввода данных открывается главная страница пользователя, представленная на рисунке 10.

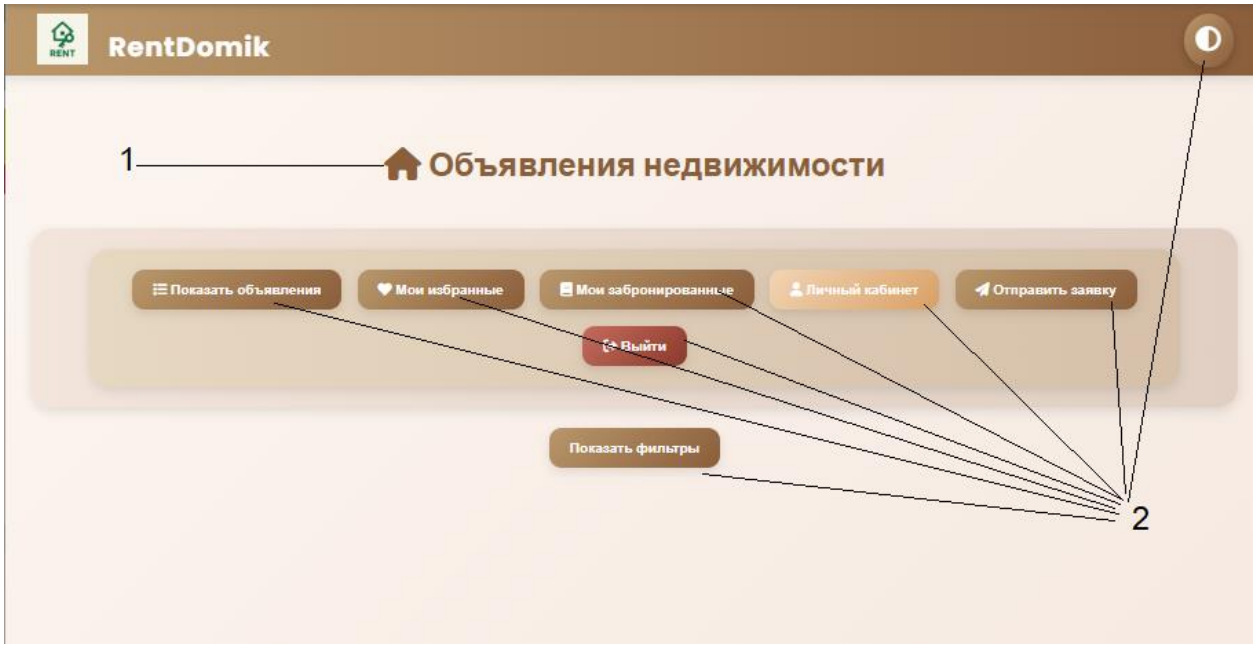

Рисунок 10 – Форма главной страницы пользователя

Данная форма включает в себя:

1 – заголовок страницы с объявлениями недвижимости для пользователей, компонент <h1>;

2 – кнопки управления: показать все доступные объявления, избранные, забронированные, личный кабинет, отправить заявку на размещение объявления, выход. Кнопка «показать фильтры» позволяет открыть панель для сортировки или поиска объявлений, компонент <button>.

Форма показа объявлений представлена на рисунке 11. На экране отображаются карточки недвижимости с основными характеристиками: изображение, адрес, цена, количество комнат. Пользователь может применить фильтры по цене, количеству комнат и ключевым словам для более точного

поиска. Для каждой карточки доступны действия: добавить в избранное, забронировать, оставить комментарий и оценку.

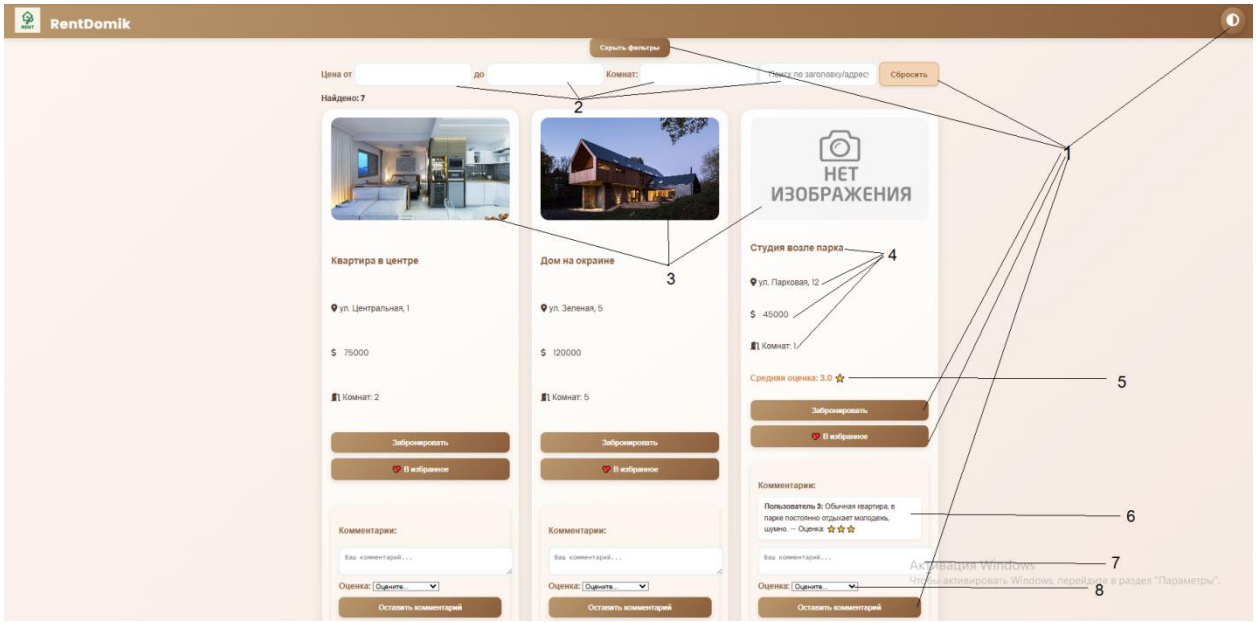

Рисунок 11 – Форма показа объявлений

Данная форма включает в себя:

1 – кнопки показа фильтрации, ее сброса, перехода на темную тему, бронировки объявления, добавления его в избранное, оставления комментария, компонент <button>;

2 – поля ввода информации для фильтрации объявлений, компонент <input>;

3 – изображения объявлений. Карточка без фотографии имеет заглушку «Нет изображения», компонент <img>;

4 – подписи к полям (адрес, цена, количество комнат), компонент <label>;

5 – средняя оценка объявления, компонент <label>;

6 – блок комментариев с текстом и оценкой в виде звезд, компонент <section>;

7 – поле ввода комментария, компонент <textarea>;

8 – выпадающий список, для выбора оценки, компонент <select>.

Раздел избранных объявлений представлен на рисунке 12.

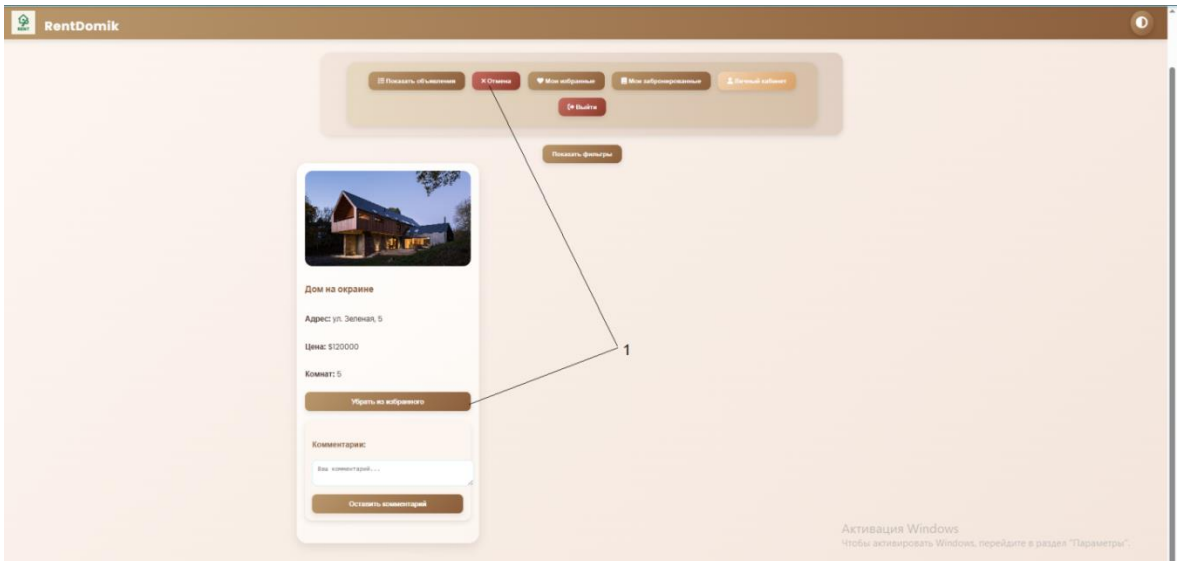

Рисунок 12 – Форма избранных объявлений

На форме добавлена исчезающая кнопка «отмена», она появляется при нажатии на «Показать объявления», «Мои избранные», «Мои забронированные», также карточка объявления с кнопкой «Убрать из избранного» и с возможностью комментирования, компонент <button>.

Страница забронированных недвижимостей представлена на рисунке 13.

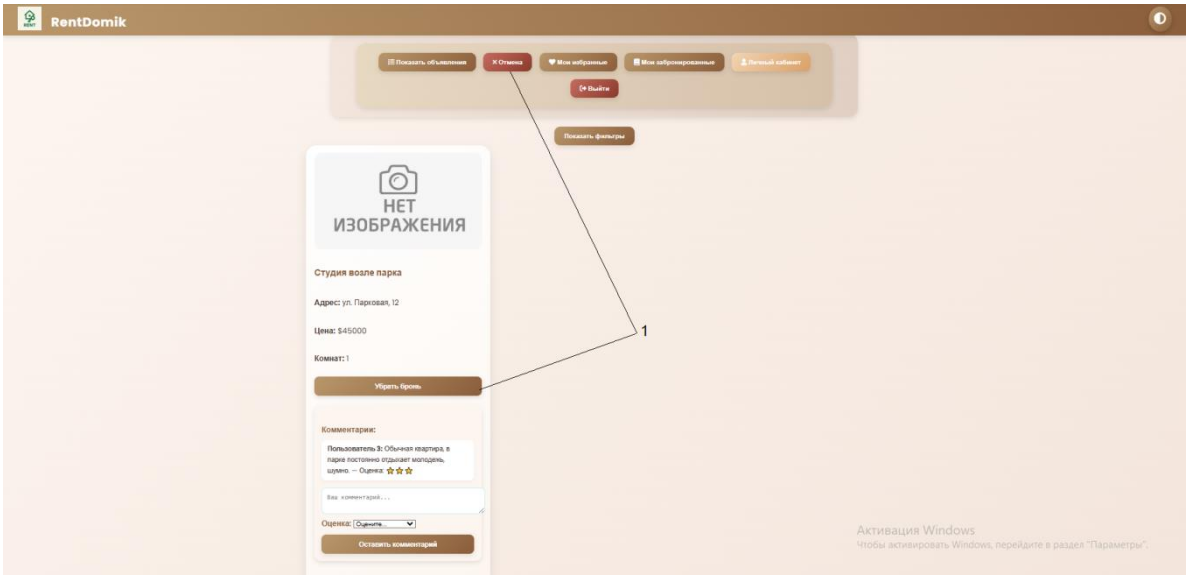

Рисунок 13 – Раздел «Мои забронированные»

В данном разделе отображается исчезающая кнопка «отмена», она появляется при нажатии на «Показать объявления», «Мои избранные», «Мои

забронированные», также карточка объявления с кнопкой «Убрать бронь» и возможностью комментирования и оценки объявления, компонент <button>.

Интерфейс личного кабинета представлен на рисунке 14. Пользователь может обновить информацию либо отменить внесенные изменения.

Рисунок 14 – Форма личного кабинета пользователя

В данной форме реализовано:

1 – заголовок формы «Личный кабинет», компонент <h1>;

2 – подписи к полям (имя, фамилия, email), компонент <label>;

3 – текстовые поля ввода и редактирования, компонент <input>;

4 – кнопки для обновления информации и отмены внесения изменений, компонент <button>.

Раздел заявки на размещение объявления представлен на рисунке 15. На данной форме пользователь может создать заявку на размещение объявления, заполнив необходимые поля с информацией о недвижимости. Интерфейс интуитивно понятен и включает много текстовых полей для ввода данных. После заполнения формы заявка отправляется администратору на рассмотрение.

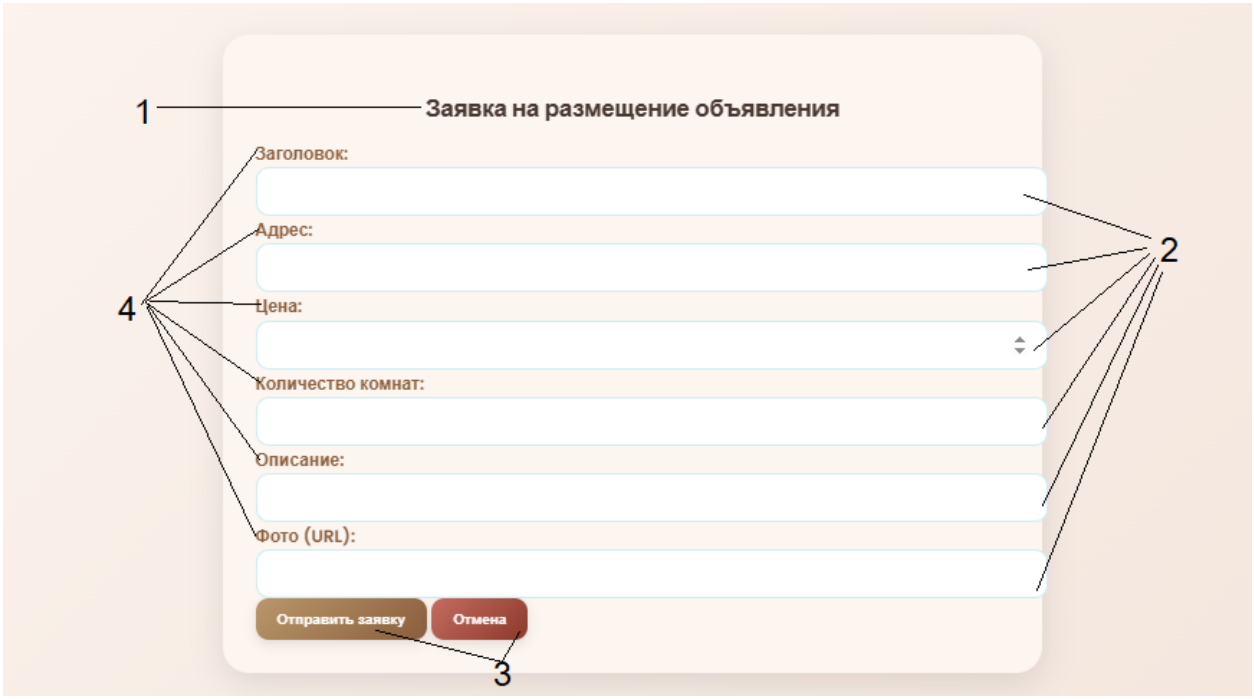

Рисунок 15 – Форма заявки на размещение объявления

В данной форме реализовано:

1 – заголовок формы с текстом «Заявка на размещение объявления», компонент <h1>;

2 – поля для ввода информации об объявлении, компонент <input>;

3 – кнопки, позволяющие отправить заявку и отменить это действие, компонент <button>;

4 – подписи к текстовым полям, поясняющие назначение каждого поля, компонент <label>.

После авторизации и взаимодействия с пользовательским интерфейсом, предусмотрена также административная панель, доступ к которой имеют только пользователи с соответствующими правами. Административная часть веб-приложения предназначена для управления контентом, пользователями и заявками, а также для контроля над размещением и модерацией объявлений. Функционал административной панели обеспечивает эффективное управление системой. Главная страница администратора представлена на рисунке 16.

Рисунок 16 – Форма главной страницы администратора

На данной форме реализовано:

1 – заголовок страницы с объявлениями недвижимости для администратора, компонент <h1>;

2 – кнопки управления: добавить объявление, редактировать и показать все объявления, личный кабинет администратора, показать пользователей, назначение администратора и просмотр заявок, также кнопка показать фильтры позволяет открыть панель для сортировки или поиска объявлений, компонент <button>.

Форма добавления объявления представлена на рисунке 17. Она предназначена для ввода новой информации об объекте недвижимости, с учетом удобства и простоты использования, чтобы пользователь мог быстро и без ошибок заполнить все необходимые поля. Для повышения качества данных предусмотрена проверка вводимой информации, что позволяет избежать некорректные или неполные записи. Такая организация интерфейса способствует эффективному и комфортному добавлению новых объектов в систему.

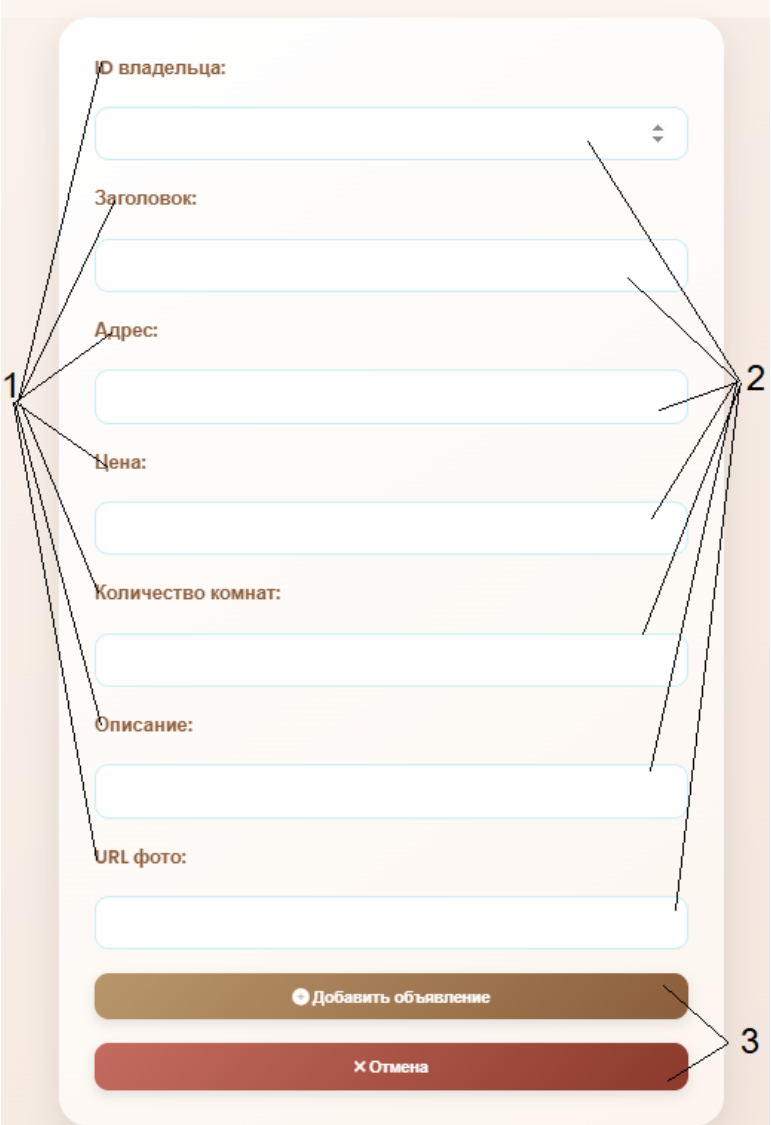

Рисунок 17 – Форма добавления объявления

В данной форме реализовано:

1 – подписи к текстовым полям, поясняющие назначение каждого поля, компонент <label>;

2 – поля для ввода информации об объявлении, компонент <input>;

3 – кнопки, позволяющие добавить объявление или отменить это действие, компонент <button>.

На рисунке 18 представлена форма редактирования объявления. Данный интерфейс используется для корректировки уже существующего объявления. Интуитивно понятный дизайн обеспечивает удобство внесения изменений и минимизирует вероятность ошибок.

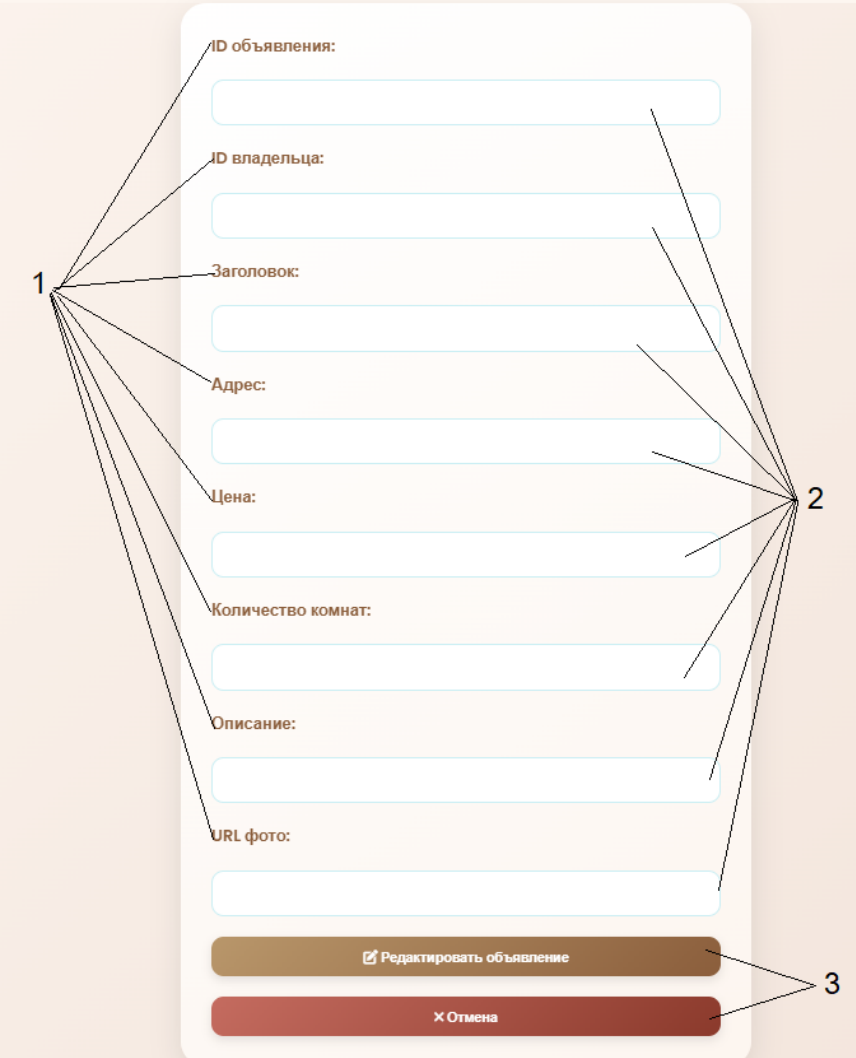

Рисунок 18 – Форма редактирования объявления

Данная форма включает в себя:

1 – подписи к текстовым полям, поясняющие назначение каждого поля, компонент <label>;

2 – поля для ввода информации об объявлении, компонент <input>;

3 – кнопки, позволяющие редактировать объявление или отменить это действие, компонент <button>.

Форма показа объявлений представлена на рисунке 19. Она позволяет администратору просматривать все размещенные объявления в виде карточек. Это способствует оперативному контролю и поддержанию актуальности размещенной информации на платформе.

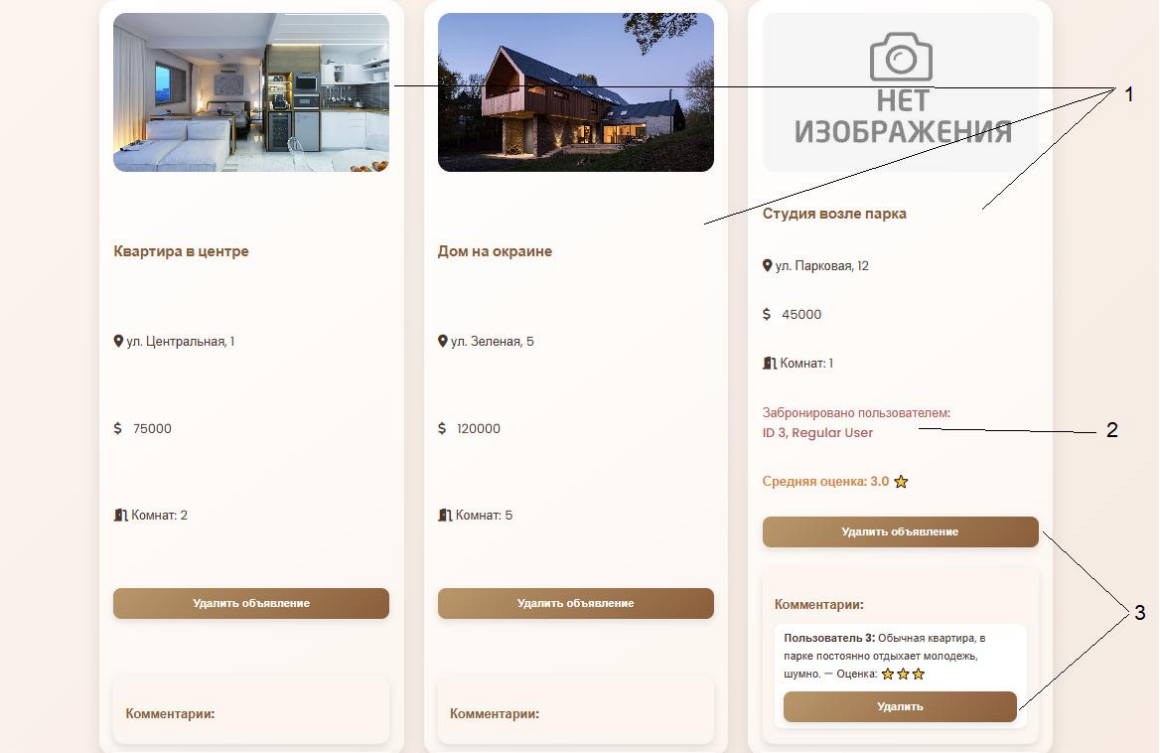

Рисунок 19 – Форма «Показать объявления»

На данной форме реализовано:

1 – все карточки объявлений, содержащие изображение и информацию о недвижимости;

2 – текстовая информация о бронировании, включая ID пользователя, компонент <p>;

3 – кнопки для удаления объявления и комментария, компонент <button>.

На рисунке 20 представлена форма показа пользователей. Она отображает список всех зарегистрированных пользователей системы.

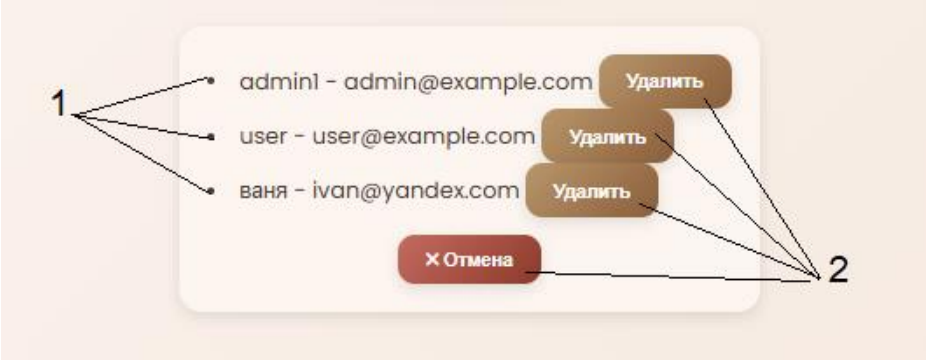

Рисунок 20 – Форма «Показать пользователей»

На данной форме реализовано:

1 – текстовая информация о пользователях, компонент <p>;

2 – кнопки для удаления пользователя и отмены, компонент <button>.

Форма личный кабинет представлен на рисунке 21. Интерфейс предназначен для редактирования персональной информации пользователя.

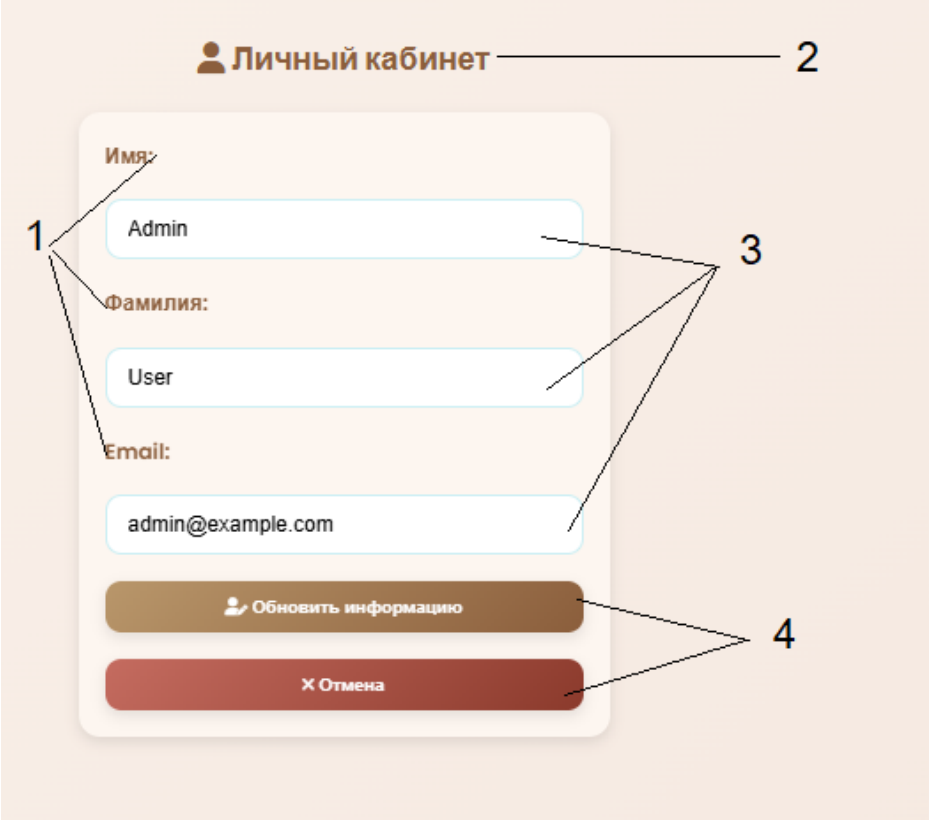

Рисунок 21 – Форма личного кабинета администратора

Данная форма включает в себя:

1 – подписи к текстовым полям, поясняющие назначение каждого поля, компонент <label>;

2 – заголовок страницы личного кабинета, компонент <h1>;

3 – поля для ввода новой информации, компонент <input>;

4 – кнопки, позволяющие обновить информацию и отменить это действие, компонент <button>.

На рисунке 22 представлена форма просмотра заявки. Она используется администратором для рассмотрения заявок на размещение новых объявлений.

Рисунок 22 – Форма заявки на объявления

Данная форма включает в себя:

1 – заголовок страницы заявки на объявления, компонент <h1>;

2 – текстовая информация об объявлении, компонент <p>;

3 – кнопки одобрения, отклонения заявки на размещение объявления и отмена действий, компонент <button>.

Также есть возможность назначить пользователя администратором. При нажатии на кнопку «Назначить администратора» появляется уведомление, изображенное на рисунке 23, куда вводится ID пользователя, для смены его роли.

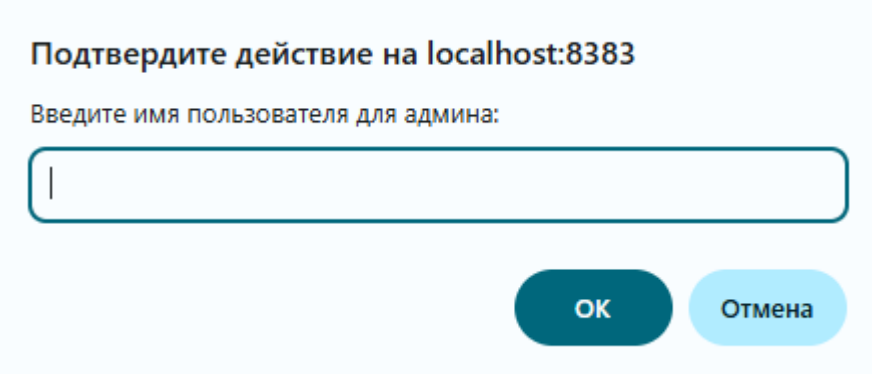

Рисунок 23 – Уведомление на назначение администратора

Клиентская часть приложения была реализована с помощью языка JavaScript, архитектура представлена на рисунке 24 и в приложении А в виде структурно-функциональной диаграммы.

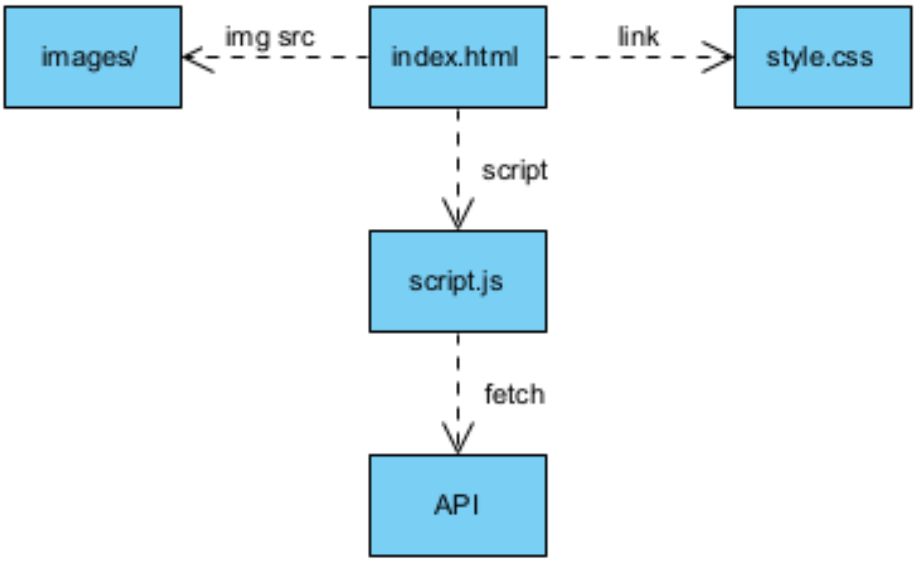

Рисунок 24 – Структурно-функциональная диаграмма клиентской части

Структура клиентской части состоит из следующих компонентов:

– index.html – это основной HTML-файл, содержащий структуру веб-страницы, имеющий ссылки на другие файлы, такие как style.css и script.js;

– style.css – это файл, определяющий визуальное оформление элементов интерфейса;

– script.js – это файл, реализующий логику работы на стороне клиента;

– images/ – каталог, содержащий используемые на веб-странице изображения;

– API – внешний элемент, отражающий взаимодействие клиентской части с серверной через REST API.

Каждый компонент клиентской части выполняет строго определённую функцию, что обеспечивает чёткую организацию кода и упрощает навигацию по проекту. Разделение структуры на отдельные функциональные элементы способствует снижению связности и повышает удобство при внесении изменений или добавлении нового функционала.

Особое внимание в архитектуре клиентской части уделено взаимодействию с серверной стороной через REST API. Это позволяет получать

актуальные данные, отправлять запросы и синхронизировать состояние интерфейса с бизнес-логикой приложения. Такая организация обеспечивает гибкость пользовательского взаимодействия, позволяет быстро реагировать на действия пользователя и поддерживает высокую интерактивность веб-интерфейса.

Структурно-функциональная схема облегчает понимание логики работы клиентской части и способствует ее дальнейшему сопровождению и развитию.

2.5 Проектирование классов приложения для аренды жилых помещений

Для обеспечения четкого представления архитектуры приложения, для аренды жилых помещений, была разработана диаграмма классов, представленная на рисунке 25 и в приложение А.

Диаграмма классов отражает ключевые компоненты и их взаимосвязи. В отличие от реляционной модели данных, она позволяет представить объекты как единицы, которые обладают не только состоянием, но и поведением. Такой подход упрощает понимание логики взаимодействия между слоями приложения и создает более полное представления о внутреннем устройстве системы [16].

На диаграмме отображены основные сущности, участвующие в бизнес-процессах. Особое внимание уделено структурным связям между объектами, включая композиции и зависимости [17]. Это позволяет наглядно представить, какие объекты являются частью других, а какие используются временно. Также определены сервисы, контроллеры и вспомогательные классы, обеспечивающие работу API, что делает схему максимально информативной.

Диаграмма классов формирует принципы построения бизнес-логики и взаимодействие между слоями. Визуальное отображение сущностей и их связей облегчает переход к практической реализации, снижая риск архитектурных ошибок и повышая качество программного обеспечения.

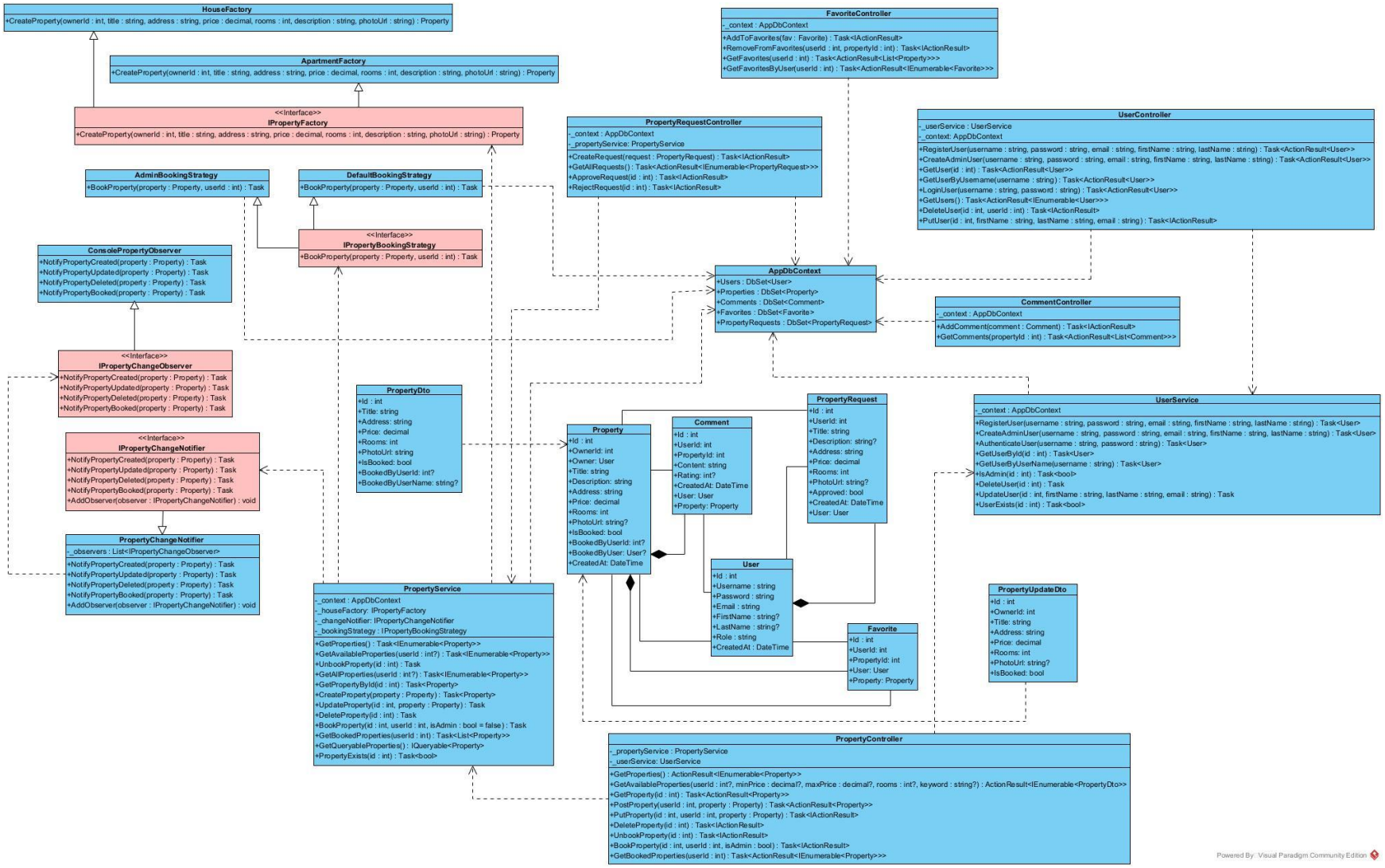

Рисунок 25 – Диаграмма классов

В таблице 9 представлено описание компонентов диаграммы классов.

Таблица 9 – Компоненты диаграммы классов

Компонент	**Назначение**
User	Модель, содержащая данные пользователя: имя, логин, пароль, дата регистрации
Property	Модель, представляющая объект недвижимости с описанием, адресом, статусов бронирования и фото
Comment	Модель, содержащая текст комментария, которое привязано к объекту недвижимости и пользователю
Favorite	Модель, связывающая пользователя с добавленным в избранное объектом недвижимости
PropertyRequest	Модель, представляющая запрос пользователя на публикацию нового объекта недвижимости
PropertyDto	Класс, содержащий данные для отображения информации об объекте
PropertyUpdateDto	Класс, используемый для передачи данных при обновлении информации об объекте недвижимости
PropertyService	Класс, предоставляющий бизнес-логику для управления данными объектов недвижимости
UserService	Класс, реализующий обработку запросов при регистрации, аутентификации и управлении пользователями
PropertyController	Класс, реализующий REST-интерфейс для взаимодействия с объектами недвижимости
UserController	Класс, обрабатывающий запросы пользователей: регистрация, получение данных, обновление
FavoriteController	Класс, предоставляющий API для управления избранным
CommentController	Класс, реализующий логику добавления и отображения комментариев к объектам недвижимости
PropertyRequestController	Класс, обрабатывающий запрос запросы на публикацию новых объектов недвижимости
HouseFactory, ApartmentFactory	Классы, реализующие создание объектов недвижимости типов «Дом» и «Квартира»
PropertyFactory	Абстрактный класс, определяющий общий интерфейс для создания объектов недвижимости
IPropertyBookingStrategy	Интерфейс, определяющий стратегию бронирования объектов недвижимости
AdminBookingStrategy, DefaultBookingStrategy	Класс, реализующий стратегии бронирования для обычного пользователя и администратора
IPropertyChangeObserver	Интерфейс наблюдателя, отслеживающий события: создание, удаление, изменение и бронирование объектов
IPropertyChangeNotifier	Интерфейс субъекта, отправляющего уведомления наблюдателям об изменениях объектов
PropertyChangeNotifier	Класс, реализующий механизм подписки и уведомления об изменениях состояния объектов

Таким образом, представленный набор классов охватывает все ключевые аспекты архитектуры приложения.

3 Реализация приложения для аренды жилых помещений

3.1 Реализация базы данных

Для хранения информации о пользователях, объектах недвижимости, комментариях, избранных записях и заявках на размещение в проекте используется реляционная база данных, реализованная средствами ORM-технологии Entity Framework Core на платформе .NET.

База данных создается по принципу Code First, при котором структура таблиц формируется на основании C# – классов, описывающих сущности предметной области. Миграции формируются автоматически при помощи встроенных инструментов EF Core.

Каждая сущность предметной области описывается в виде класс с соответствующими свойствами. На листинге 1 представлен пример модели пользователя.

Листинг 1 – Класс User

```
public class User
{
    public int Id { get; set; }
    public string Username { get; set; }
    public string Email { get; set; }
    public string PasswordHash { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Role { get; set; } = "user";
    public DateTime CreatedAt { get; set; } = DateTime.Now;
}
```

Подобным образом можно представить и другие сущности. Например, модель объекта недвижимости, продемонстрированная на листинге 2.

Листинг 2 – Класс Favorite

```
public class Property
{
    public int Id { get; set; }
    public string Title { get; set; }
    public string Address { get; set; }
    public decimal Price { get; set; }
    public int Rooms { get; set; }
    public string? Description { get; set; }
    public string? PhotoUrl { get; set; }
```

```
    public int OwnerId { get; set; }
    public User? Owner { get; set; }
    public bool IsBooked { get; set; }
    public int? BookedByUserId { get; set; }
    public User? BookedByUser { get; set; }
    public DateTime CreatedAt { get; set; } = DateTime.Now;
}
```

Эта модель характерна наличием отношения «один-ко-многим» с пользователями и допускает отсутствие арендатора, предоставляя возможность не указывать его, если объект свободен.

Контекст, представленный классом, унаследованным от DbContext, выполняет роль посредника между приложением и базой данных, что продемонстрированно в листинге 3.

Листинг 3 – Класс AppDbContext

```
public class AppDbContext : DbContext
{
    public DbSet<User> Users { get; set; }
    public DbSet<Property> Properties { get; set; }
    public DbSet<Comment> Comments { get; set; }
    public DbSet<Favorite> Favorites { get; set; }
    public DbSet<PropertyRequest> PropertyRequests { get; set; }

    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<Property>()
            .HasOne(p => p.Owner)
            .WithMany()
            .HasForeignKey(p => p.OwnerId)
            .OnDelete(DeleteBehavior.Cascade);

        modelBuilder.Entity<Comment>()
            .HasOne(c => c.Property)
            .WithMany()
            .HasForeignKey(c => c.PropertyId)
            .OnDelete(DeleteBehavior.Cascade);
    }
}
```

Через метод OnModelCreating задаются правила каскадного удаления и ограничения между таблицами.

EF Core предоставляет механизм миграций, позволяющий создавать и изменять структуру базы данных без прямого написания SQL-кода. Например,

при первом запуске проекта выполняются команды, представленные на листинге 4.

Листинг 4 – Команды для миграций

```
dotnet ef migrations add InitialCreate
dotnet ef database update
```

В результате EF Core генерирует SQL-скрипты, по которым создаются таблицы и связи.

Автоматически сгенерированная миграция для создания таблицы объектов недвижимости представлены на листинге 5.

Листинг 5 – Миграция для создания таблицы объектов недвижимости

```
migrationBuilder.CreateTable(
    name: "Properties",
    columns: table => new
    {
        Id = table.Column<int>(nullable: false)
            .Annotation("SqlServer:Identity", "1, 1"),
        Title = table.Column<string>(nullable: true),
        Address = table.Column<string>(nullable: true),
        Price = table.Column<decimal>(nullable: false),
        Rooms = table.Column<int>(nullable: false),
        Description = table.Column<string>(nullable: true),
        PhotoUrl = table.Column<string>(nullable: true),
        OwnerId = table.Column<int>(nullable: false),
        IsBooked = table.Column<bool>(nullable: false),
        BookedByUserId = table.Column<int>(nullable: true),
        CreatedAt = table.Column<DateTime>(nullable: false)
    },
    constraints: table =>
    {
        table.PrimaryKey("PK_Properties", x => x.Id);
        table.ForeignKey(
            name: "FK_Properties_Users_OwnerId",
            column: x => x.OwnerId,
            principalTable: "Users",
            principalColumn: "Id",
            onDelete: ReferentialAction.Cascade);
    });
```

После выполнения миграции формируются таблицы:

– Users – таблица пользователей;

– Properties – таблица объектов недвижимости;

– PropertyRequests – таблица заявок на размещение недвижимости;

– Comments – таблица отзывов пользователей;

– Favorites – таблица избранных объявлений.

Физические модели баз данных определяют методы организации данных в хранилище и механизмы доступа к ним на аппаратном уровне. Диаграмма, представленная на рисунке 26 и в приложении А, наглядно показывает таблицы, связи между ними и ключевые поля проекта.

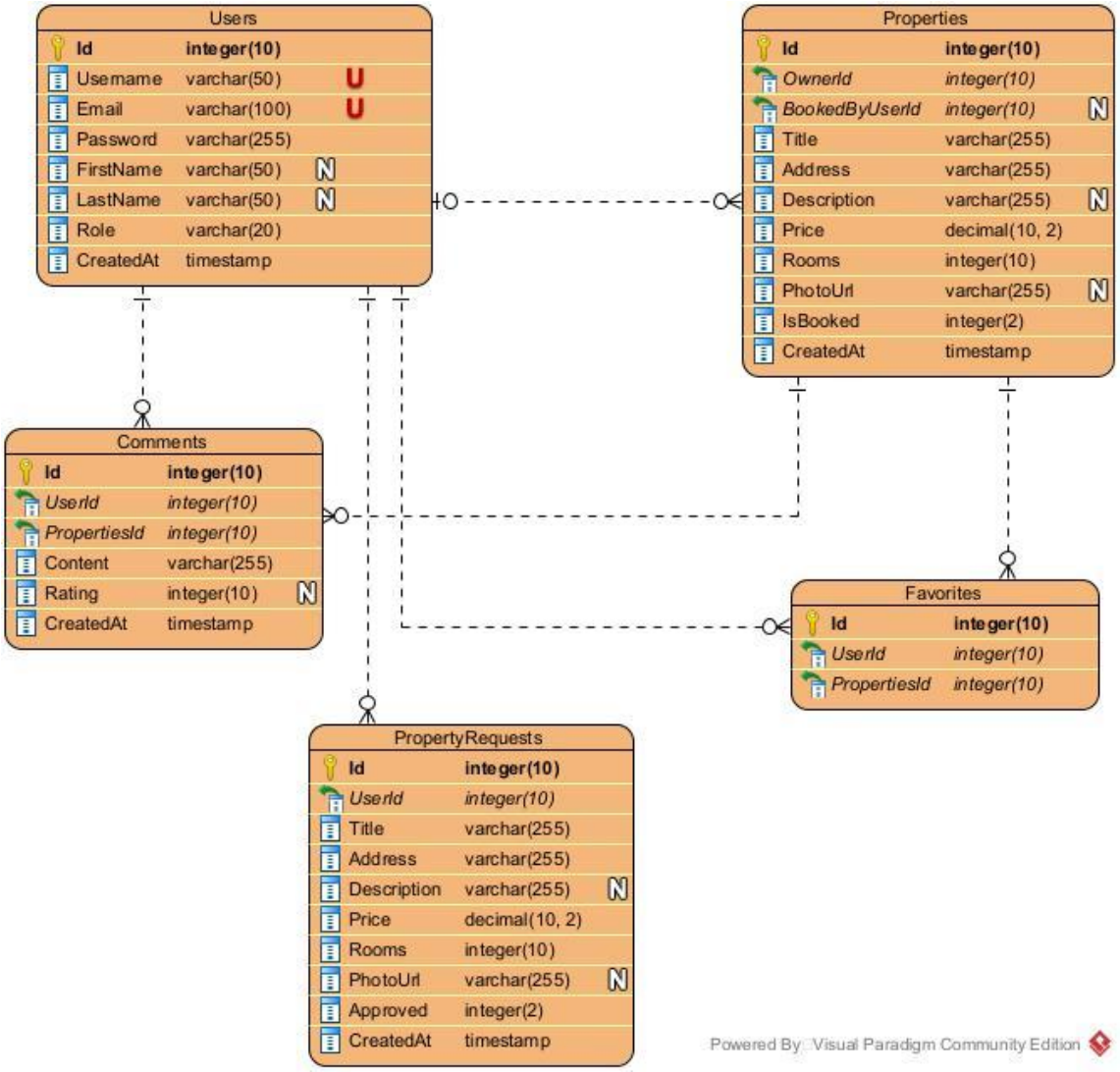

Рисунок 26 – Диаграмма физической модели

Таким образом, база данных реализована как совокупность логически взаимосвязанных таблиц, отвечающих требованиям, установленным на этапе проектирования. Применение Entity Framework Core обеспечило структурированное хранение данных и гибкость в изменении структуры.

3.2 Реализация серверной части приложения

Серверная часть информационной системы разработана на платформе ASP.NET Core, что способствовало высокой производительности и создание масштабируемых приложений. Основной принцип построения серверной части – многослойная монолитная архитектура, в рамках которой реализовано четкое разделение логики между слоями: контроллеров, сервисов и доступа к данным.

Приложение предоставляет доступ к функциональности через REST API, взаимодействие осуществляется по протоколу HTTP с использованием формата обмена данными JSON [18]. Такой подход обеспечивает совместимость с различными клиентскими приложениями и простоту интеграции.

Архитектура серверной части включает в себя следующие уровни:

– контроллеры – принимают входящие HTTP-запросы, выполняют базовую валидацию и передают управление на уровень бизнес-логики;

– сервисы – содержат бизнес-логику приложения и осуществляют обработку данных, поступающих от контроллеров или из базы данных;

– модели – представляют сущности предметной области, соответствующие структуре таблиц в базе данных;

– контекст базы данных – реализует доступ к данным посредством Entity Framework Core и выполняет роль промежуточного слоя между базой данных и логикой приложения.

Все классы сгруппированы по слоям в соответствующих пространствах имен, что облегчает поддержку и расширение проекта.

Использование ASP.NET Core обусловлено его возможностями:

– встроенная поддержка авторизации и аутентификации;

– интеграция с системами логирования через компонент «ILogger», что позволяет отслеживать ошибки, запросы и внутренние события приложения;

– расширяемость через middleware;

– высокая производительность и масштабируемость.

Контроллеры реализованы с использованием атрибутов ApiController и Route. На листинге 6 приведен фрагмент контроллера, обрабатывающего запросы, связанные с объектами недвижимости.

Листинг 6 – Фрагмент контроллера PropertyController

```
[ApiController]
[Route("api/[controller]")]
public class PropertyController : ControllerBase
{
    private readonly PropertyService _service;

    public PropertyController(PropertyService service)
    {
        _service = service;
    }
    [HttpGet]
    public async Task<IActionResult> GetAll()
    {
        var result = await _service.GetAllAsync();
        return Ok(result);
    }
    [HttpPost]
    public async Task<IActionResult> Create([FromBody] Property
property)
    {
        await _service.CreateAsync(property);
        return Ok(property);
    }
}
```

Контроллер взаимодействует с сервисом PropertySevice, внедренным через механизм внедрения зависимостей, что соответствует современным практикам разработки.

Бизнес-логика приложения инкапсулируется в отдельных сервисах. Это позволяет избежать дублирование кода и упрощает его сопровождение. На листинге 7 представлен сервис для работы с сущностью Property.

Листинг 7 – Сервис для работы с сущностью Property

```
public class PropertyService
{
    private readonly AppDbContext _context;
    public PropertyService(AppDbContext context)
    {
        _context = context;
    }
    public async Task<List<Property>> GetAllAsync()
```

```
    {
        return await _context.Properties.ToListAsync();
    }
    public async Task CreateAsync(Property property)
    {
        _context.Properties.Add(property);
        await _context.SaveChangesAsync();
    }
}
```

Все операции с базой данных осуществляются через контекст AppDbContext, настроенный с использованием ORM Entity Framework Core. Это гарантирует надежное и безопасное взаимодействие с хранилищем данных.

Одна из ключевых функций приложения – прием и обработка заявок пользователей на размещение объектов. После рассмотрения администратором заявка может быть одобрена или отклонена. В случае одобрения объект переносится в основную таблицу. Пример представлен на листинге 8.

Листинг 8 – Обработка одобрения заявки

```
[HttpPost("approve/{id}")]
public async Task<IActionResult> ApproveRequest(int id)
{
    var request = await _context.PropertyRequests.FindAsync(id);
    if (request == null) return NotFound();
    var property = new Property
    {
        OwnerId = request.UserId,
        Title = request.Title,
        Address = request.Address,
        Price = request.Price,
        Rooms = request.Rooms,
        Description = request.Description,
        PhotoUrl = request.PhotoUrl
    };
    _context.Properties.Add(property);
    request.Approved = true;

    await _context.SaveChangesAsync();
    return Ok(property);
}
```

Для отклонения заявки реализован отдельный маршрут, представленный на листинге 9.

Листинг 9 – Обработка отклонения заявки

```
[HttpDelete("reject/{id}")]
public async Task<IActionResult> Reject(int id)
```

```
{
    var request = await _context.PropertyRequests.FindAsync(id);
    if (request == null) return NotFound();

    _context.PropertyRequests.Remove(request);
    await _context.SaveChangesAsync();

    return NoContent();
}
```

Такая структура обеспечивает гибкость и безопасность при управлении пользовательским контентом.

Для валидации входных данных используются атрибуты аннотаций, такие как [Required], [MaxLength], [Range]. Это позволяет ASP.NET Core автоматически проверять корректность данных и возвращать ошибку при нарушении правил валидации. Пример представлен на листинге 10.

Листинг 10 – Атрибуты аннотаций

```
[Required]
[MaxLength(100)]
public string Title { get; set; }

[Range(1, 5)]
public int Rating { get; set; }
```

Безопасность приложения обеспечивается авторизацией по ролям. Пользователи системы делятся на обычных пользователей и администраторов. Доступ к административным маршрутам ограничен с помощью атрибута [Authorize(Roles = “admin”)].

Взаимодействие между клиентом и сервером осуществляется в формате JSON. Пример структуры запроса на создание объекта недвижимости представлен на листинге 11.

Листинг 11 – Структура запроса на создание объекта недвижимости

```
{
  "title": "Уютная квартира",
  "address": "ул. Ленина, 12",
  "price": 28000,
  "rooms": 2,
  "photoUrl": "https://example.com/photo.jpg"
}
```

Это делает API совместимым с современными фронтенд-фреймворками.

Серверная часть информационной системы реализована в виде многослойного монолитного REST-приложения на платформе ASP.NET Core. Архитектура обеспечивает логическое разделение компонентов, упрощает сопровождение и тестирование. Использование Entity Framework Core позволяет надежно и безопасно работать с базой данных. Отдельное внимание уделено безопасности, авторизации по ролям и валидации данных. Интеграция с клиентским приложением реализована через современный и универсальный формат JSON. Листинг реализации серверной части представлен в приложении В.

Физическая структура размещения компонентов приложения представлена на диаграмме развертывания на рисунке 27 и в приложении А. Она показывает, как взаимодействуют между собой клиент, сервер и база данных в процессе работы системы.

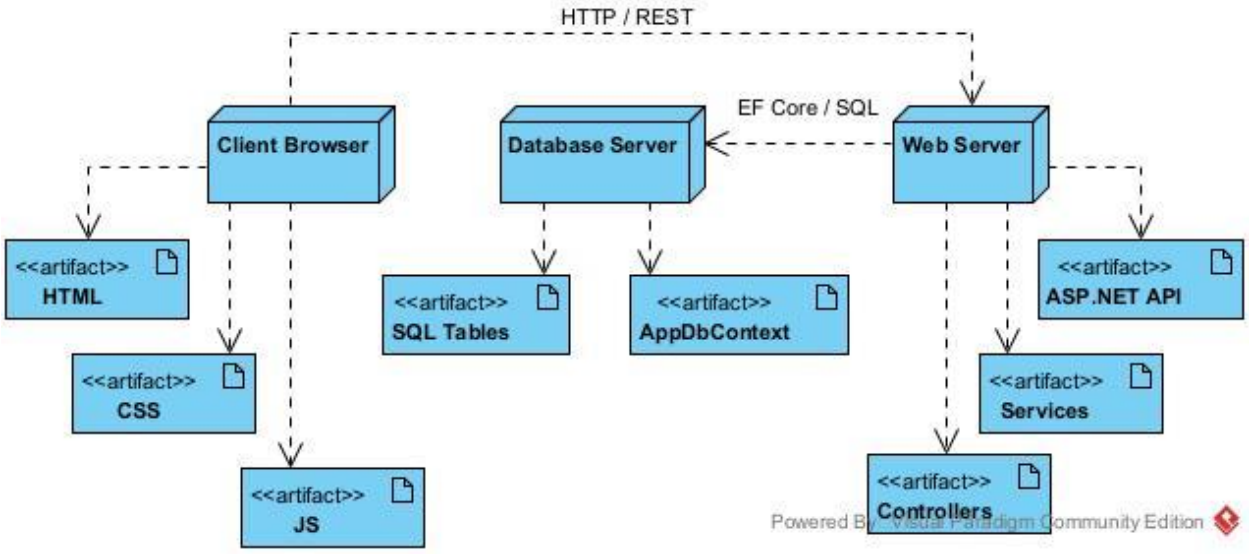

Рисунок 27 – Диаграмма развертывания

Диаграмма иллюстрирует работу трех основных узлов:

– клиентский браузер (Client Browser) – выполняет HTML, CSS и JavaScript, отображает интерфейс и инициирует запросы к серверу;

– веб-сервер (Web Server) – обрабатывает HTTP-запросы через ASP.NET Core, реализует бизнес-логику и взаимодействует с базой данных;

– сервер базы данных (Database Server) – отвечает за хранение информации в SQL-таблицах, к которым осуществляется доступ через Entity Framework Core.

Взаимодействие между узлами осуществляется по протоколу HTTP с передачей данных в формате JSON, а также посредством SQL-запросов на уровне сервера.

Архитектура серверной части представлена на рисунке 28 и в приложении А с помощью диаграммы компонентов, где отображены ключевые слои приложения и их взаимодействие.

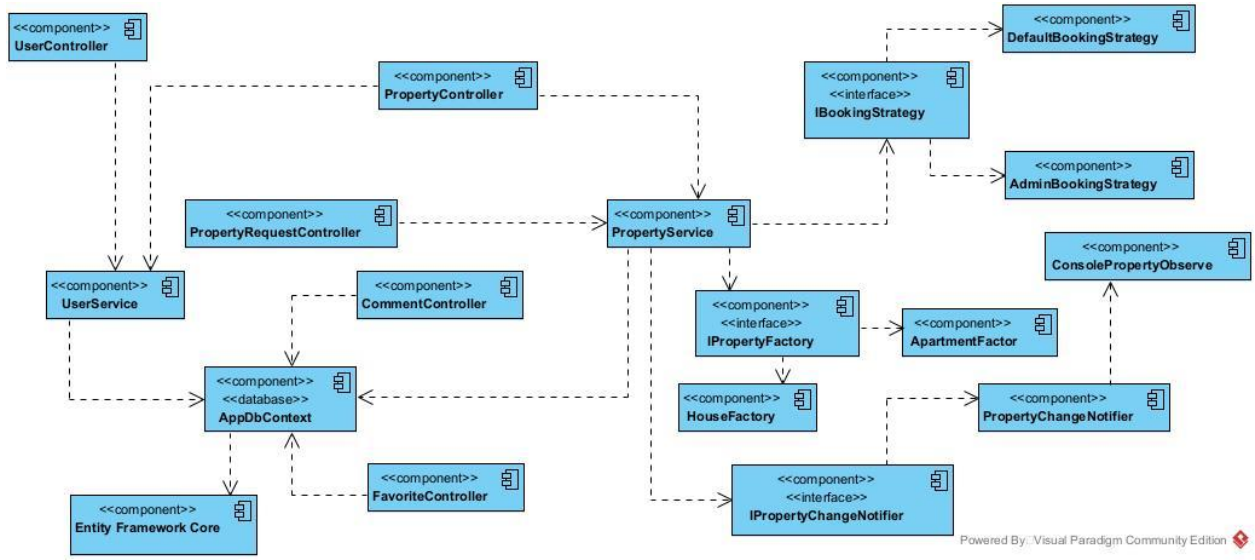

Рисунок 28 – Диаграмма компонентов серверной части

Использование шаблонов проектирования в сочетании с многоуровневой архитектурой позволило создать решение, обладающее необходимой гибкостью и возможностью расширения функционала. Четкое разделение обязанностей между слоями системы упрощает понимание логики работы компонентов при добавлении нового функционала или внедрении новых сценариев использования.

Применение паттернов проектирования способствует сокрытию изменяемых элементов бизнес-логики и уменьшению взаимозависимости между модулями. Это позволяет легко внедрять новые типы объектов недвижимости,

настраивать различные сценарии бронирования в зависимости от роли пользователя и обеспечивает адаптацию системы к изменениям ее состояния.

Такой подход актуален для информационных систем с активным пользовательским взаимодействием. В итоге, реализованная архитектура гарантирует не только соответствие заданным функциональным требованиям, но и создает надежную основу для дальнейшего развития приложения.

3.3 Реализация клиентской части приложения

Клиентская часть приложения обеспечивает визуальное представление данных и реализует механизмы взаимодействия пользователя с функциональностью системы. В качестве платформы был выбран формат одностраничного веб-приложения, реализованный на основе технологий HTML, CSS и JavaScript.

Приложение разделено на модули, соответствующие различным пользовательским сценариям: регистрация и вход, работа с объявлениями, отправка заявок, управление избранным и бронированием, а также доступ к административным возможностям. Такой подход позволяет пользователю работать в пределах одного интерфейса и не требует перезагрузку страниц, что значительно повышает удобство использования.

Для взаимодействия с серверной частью используются асинхронные HTTP-запросы, реализованные через fetch API. Ответы сервера обрабатываются на стороне клиента, после чего данные динамически отображаются в DOM-дереве.

Пример отправки запроса на получение списка доступных объявлений представлен на листинге 12.

Листинг 12 – Запрос на получение списка доступных объявлений

```
fetch(`/api/Property/available?userId=${userId}`)
  .then(response => response.json())
  .then(data => renderProperties(data))
  .catch(error => console.error("Ошибка загрузки данных: ",
error));
```

На стороне клиента осуществляется предварительная валидация данных с целью исключения некорректных запросов к серверу. Например, при регистрации пользователя осуществляется проверка на заполненность обязательных полей (username, email, password).

Ключевым элементом клиентской логики является отображение объектов недвижимости в виде карточек. Каждая карточка содержит основную информацию об объекте: адрес, стоимость аренды, количество комнат, изображение, а также кнопки взаимодействия (забронировать, добавить в избранное, оставить комментарий). Карточки формируются динамически на основе полученных с сервера данных, с последующей вставкой в DOM-дерево.

Визуализация списка объектов реализуется на листинге 13.

Листинг 13 – Визуализация списка объектов

```
function renderProperties(properties) {
  const container = document.getElementById("propertyList");
  container.innerHTML = "";
  properties.forEach(p => {
    const card = document.createElement("div");
    card.className = "property-card";
    card.innerHTML = `
      <img src="${p.photoUrl || 'images/no-image.png'}"
alt="Изображение">
      <h3>${p.title}</h3>
      <p>Адрес: ${p.address}</p>
      <p>Цена: ${p.price} ₽</p>
      <p>Комнат: ${p.rooms}</p>
      <button
onclick="bookProperty(${p.id})">Забронировать</button>
      <button onclick="addToFavorites(${p.id})">Избранное</button>
    `;
    container.appendChild(card);
  });
}
```

Для удобства работы пользователя реализован механизм фильтрации объявлений, позволяющий задать диапазон стоимости, количество комнат и ключевые слова, например, по адресу или название объекта. Данные фильтра передаются как параметры GET-запроса, представленного на листинге 14.

Листинг 14 – GET-запрос с данными фильтрации

```
const url =
`/api/Property/available?minPrice=${min}&maxPrice=${max}&rooms=${r
ooms}&keyword=${keyword}`;
```

Также реализован функционал управления избранными объектами. Пользователь может сохранить понравившиеся объявления в личный список. Для хранения используется соответствующий маршрут API:

– POST/api/Favorite – добавление объекта в избранное;

– DELETE/api/Favorite/{userId}/{propertyId} – удаление;

– GET/api/Favotite/{userId} – получение списка избранных объектов.

Пример клиентской функции представлен на листинге 15.

Листинг 15 – Функция добавления объявления в избранное

```
function addToFavorites(propertyId) {
  fetch(`/api/Favorite`, {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify({ userId: currentUser.id, propertyId })
  });
}
```

Клиентская часть также включает модуль бронирования объектов недвижимости. Если объект не занят, пользователь может отправить запрос, и сервер обновляет поля IsBooked и BookedByUserId. После успешного бронирования карточка объекта не видна в общем списке недвижимостей у обычных пользователей, а у администратора отображается имя арендующего.

Для пользователей реализована форма подачи заявки на размещение объявления. После отправки данные сохраняются в таблице PropertyRequests, где далее обрабатываются администратором.

Важным компонентом клиентской части является панель администратора, предоставляющая расширенные возможности по управлению контентом системы. Интерфейс администратора включает следующие модули:

– просмотр и модерация заявок на размещение объявлений;

– добавление, редактирование и удаление объектов недвижимости;

– управление списком пользователей (назначение роли администратора, удаление);

– управление комментариями (удаление).

Панель реализована в виде набора кнопок, отображаемых при подтверждение административных прав и представлена на листинге 16.

Листинг 16 – Администраторская панель

```
if (currentUser.role === "admin") {
  document.getElementById("adminPanel").style.display = "block";
}
```

Форма редактирования объекта недвижимости отправляется методом PUT по соответствующему маршруту. Для повышения надежности обновления реализована проверка соответствия идентификатора объекта, представленная на листинге 17.

Листинг 17 – Проверка соответствия идентификатора объекта

```
if (formId !== property.id) {
  alert("Ошибка: ID не совпадают");
  return;
}
```

Клиентская часть использует механизмы валидации и уведомлений пользователя. Все формы содержат HTML5-валидацию (required, type=”email”, min, max) и обработчики событий.

Для повышения качества пользовательского взаимодействия реализованы следующие функции:

– отображение индикатора загрузки (<I class =”fas fa-spinner fa-spin”></i>);

– визуальная смена темы с сохранением настроек в localStorage;

– подсказки и всплывающие сообщения при успешных или неудачных действиях.

Все функции клиентской части построены на архитектуре REST и взаимодействуют с API через асинхронные HTTP-запросы. Это обеспечивает четкое разделение клиентской и серверной логики.

Диаграмма компонентов, представленная на рисунке 29 и в приложении А, описывает ее структуру.

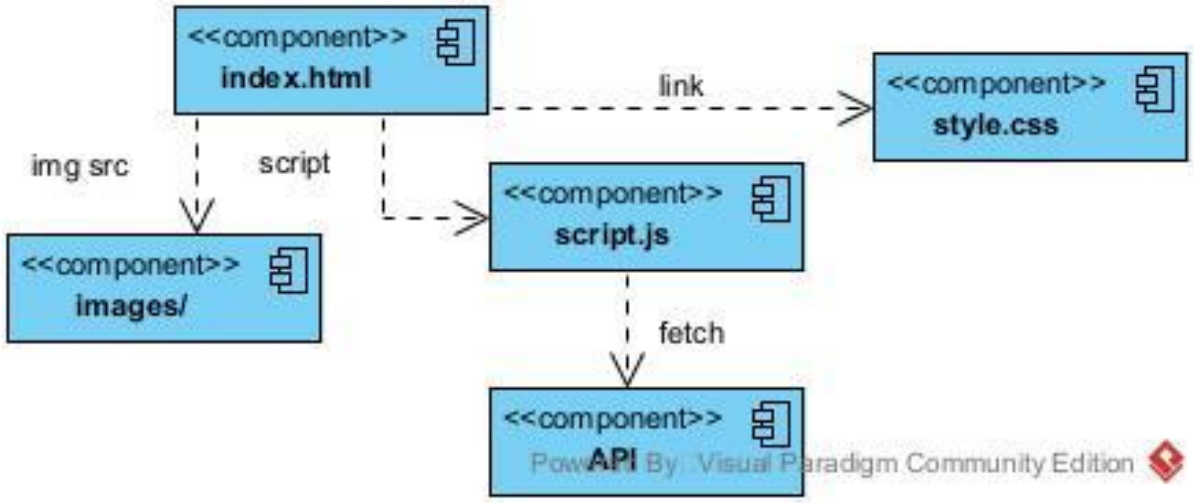

Рисунок 29 – Диаграмма компонентов клиентской части

Структура клиентской части состоит из следующих компонентов:

– index.html – это основной файл HTML, который представляет собой скелет веб-страницы, содержащий ссылки на другие файлы (style.css и script.js);

– style.css – это файл, в котором прописаны стили, определяющие, как элементы HTML должны отображаться на экране;

– script.js – это файл, в котором реализованы все основные функции приложения, так же работа с API для получения и отправки данных;

– images/ – каталог с изображениями;

– API – внешний компонент, обозначающий взаимодействие клиентской части с сервером через REST API.

Таким образом, клиентская часть приложения разработана с учетом современных стандартов веб-технологий, с использованием HTML, CSS и JavaScript. Благодаря внедрению REST-архитектуры и модульного подхода удалось достичь следующих результатов:

– создание понятного и удобного интерфейса для пользователя;

– высокая интерактивность и быстрая реакция на действия пользователя;

– простота расширения функционала в будущем;

– разграничение прав пользователей.

Интерфейс тесно связан с серверной логикой и системой управления данными, что позволило реализовать все требования, указанные в техническом задании. Листинг реализации клиентской части представлен в приложении Г.

4 Отладка и тестирование

4.1 Обоснование выбора режима тестирования

Для обеспечения высокого качества разработки и стабильной работы приложения для аренды жилых помещений применялся комплексный подход к тестированию, включающий использование различных методик. Тестирование позволяет своевременно выявлять ошибки, оценивать надежность архитектуры и корректность реализации, а также повысить удобство использования системы конечным пользователями.

На практике применяются различные виды тестирования, каждый из которых выполняет свои функции:

– модульное тестирование;
– функциональное тестирование;
– интеграционное тестирование;
– тестирование производительности.

Модульное тестирование представляет собой подход к проверке программного обеспечения, при котором отдельные части или компоненты приложения тестируются на правильность функционирования [19].

Главная задача модульного тестирования – подтвердить, что каждый отдельный модуль функционирует корректно и соответствует заданным требованиям. Модули могут включать функции, методы или классы, которые отвечают за выполнение конкретных задач.

Функциональное тестирование – это вид проверки программного обеспечения, при котором оценивается, насколько корректно работают все заявленные функции.

Цель такого тестирования – убедиться, что программа выполняет поставленные задачи и соответствует ожиданиями пользователей. При этом внимание уделяется именно тому, как работает система снаружи, без анализа внутренней структуры или кода. Такой подход помогает выявить несоответствия и ошибки в работе программы с точки зрения пользователя.

Интеграционное тестирование проверяет корректность взаимодействия между отдельными компонентами системы, такими как клиентская часть, сервер, база данных и сторонними сервисами через API [20].

Тестирование производительности используется для того, чтобы определить, насколько быстро и стабильно работает система, особенно в условиях повышенной нагрузки.

В качестве метода ручного тестирования программного обеспечения было выбрано функциональное тестирование, так как оно позволяет оценить, насколько поведение соответствует поставленным задачам, а также проверить корректность работы в условиях, приближенных к реальному использованию.

В качестве автоматизированного подхода применено модульное тестирование, что дает возможность эффективно контролировать отдельные элементы программы и существенно ускоряет процесс выявления и устранения локальных ошибок

4.2 Тестирование приложения для аренды жилых помещений

4.2.1 Модульное тестирование

Модульное тестирование проводилось с использованием среды Visual Studio и фреймворка xUnit. План модульного тестирования приведен в таблице 10. Код разработанных Unit-тестов приведен в приложении Д.

Таблица 10 – Модульное тестирование

Тестируемая функция	**Описание**	**Ожидаемый результат**	**Актуальный результат**
RegisterUser	Проверка правильной работы выполнения регистрации пользователя	Пройден	Пройден
AuthenticateUser	Проверка правильной работы авторизации пользователя, в результате которой возвращается роль пользователя	Пройден	Пройден
BookProperty	Проверка бронирования объекта – должно устанавливаться IsBooked = true и BookedByUserId	Пройден	Пройден

Продолжение таблицы 10

Тестируемая функция	Описание	Ожидаемый результат	Актуальный результат
CreateRequest	Проверка отправки заявки на размещение нового объекта	Пройден	Пройден
AddComment	Проверка добавления комментария к объекту	Пройден	Пройден

На рисунке 30 приведены результаты выполнения модульного тестирования.

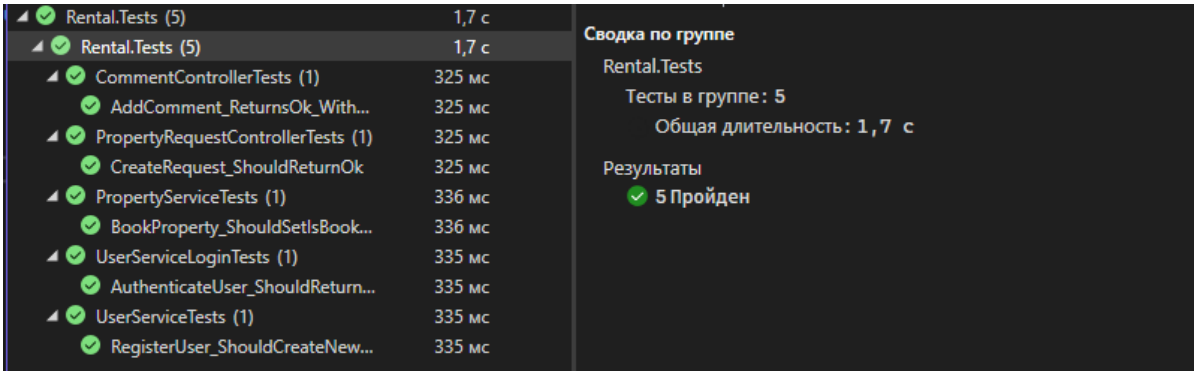

Рисунок 30 – Результаты выполнения модульных тестов

Из полученных результатов можно сделать вывод о корректной работе методов, так как тесты были успешно пройдены.

4.2.2 Функциональное тестирование

Для проверки корректности работы пользовательского интерфейса и серверной валидации полей формы было проведено функциональное тестирование методом черного ящика. Этот метод позволяет оценить работу приложения с точки зрения пользователя, без углубления во внутреннюю реализацию. В качестве тестирования выбрано поле «Адрес» при добавлении нового объекта недвижимости. Для этого были выделены классы эквивалентности данного поля, которые представлены в таблице 11.

Таблица 11 – Классы эквивалентности

Требование	Корректные значения	Некорректные значения
Длина строки	2 – 255 символов (1)	0 – 1 символ (2), более 255 символов (3)

Продолжение таблицы 11

Требование	Корректные значения	Некорректные значения
Начальный символ	Строка начинается с буквы (4)	Строка начинается не с буквы (5)
Допустимые символы	Кириллица (6), цифры (7), тире (8), точки (9)	Спецсимволы и иные недопустимые символы (10)

В соответствии с выделенными классами эквивалентности были разработаны тест-кейсы, представленные в таблице 12.

Таблица 12 – Тест-кейсы

№	КЭ	Входные данные	Выходные данные
1	1,4,6,7,8,9	ул.Карла Маркса 133	Успешно
2	2,4,6	й	Ошибка: длина меньше двух символов
3	3,4,6	Всывсвсывсвышвьлыьсввылдсь высвыывсывсвсвыскавеиретеие Мавимевыиаемуепмукмкупмук Укпукперепукпукпукпукрпкеык Епукпмукпмкпкупкпкеодшдзщг Керкердьфждьсфхзуьазйхьасэж Уацуаещдщккщкшлкокушоеогк Упурнуукпукпукпыщзвьсщдвыз Всывсзжфбсюзфюзюсбзжфюысвсзьфбщзсвбфзщбсьзщфьбсщфз	Ошибка: превышение допустимой длины
4	1,5,6,9	.Красная	Ошибка: строка не начинается с буквы
5	1,4,6,10	ул@	Ошибка: содержит недопустимые символы
6	1,5,7	3215	Ошибка: отсутствуют буквы
7	1,5,8	---	Ошибка: отсутствуют буквы
8	1,5,9	…	Ошибка: отсутствуют буквы
9	1,5,7,8,9	35 – 45.	Ошибка: отсутствуют буквы
10	1,4,6,7,	Ладожская 105	Успешно
11	1,4,6,8	Огурцова-Строительная	Успешно
12	1,4,6,9	ул.Тамбовская	Успешно

В ходе выполнения функционального тестирования черным ящиком полученные результаты не отличаются от ожидаемых, что свидетельствует о корректной работе приложения. Результаты функционального тестирования представлены в листинге Е.

5 Контроль качества

5.1 Обоснование выбора метрик кода

Для объективной оценки качества программного кода применяются количественные характеристики – метрики кода. С их помощью можно оценить сложность логики, удобство сопровождения, читаемость и уровень поддержки программного решения.

Среди часто используемых метрик в анализе программного кода можно выделить следующие:

– метрики Холстеда – базируется на подсчете числа уникальных и повторяющихся операторов и операндов в исходном коде, позволяющая рассчитать объем программы, ее логическую нагрузку и усилия, необходимые для понимания и работы с кодом [21];

– метрики Чепина – учитывает количество переменных, задействованных в модуле, и классифицирует их по функциям: входные, изменяемые, управляющие и вспомогательные, это помогает оценить структуру логики и уровень когнитивной сложности [22].

Использование таких метрик дает возможность проанализировать внешний функционал приложения, глубже понять его архитектурные и логические особенности, что позволяет создать качественный и устойчивый программный продукт.

5.2 Оценка качества программного кода

5.2.1 Метрики Холстеда

Для количественной оценки сложности и логической структуры метода используется модель программных метрик Холстеда. Необходимо посчитать 4 характеристики программного кода: длина программы (N), словарь программы (n), объем программы (V), сложность программы (D). Словарь программы состоит из суммы числа вхождений простых операторов (n1) и числа вхождений

простых операндов (n2). Длина программы состоит из суммы общего количества простых операторов (N1) и общего количества простых операндов (N2) [24].

Были рассчитаны сумма вхождений и общее количество операторов и операндов для метода создания новой записи об объекте недвижимости CreateProperty(), представленного на листинге Ж.1 в приложении Ж. результаты представлены в таблице 13.

Таблица 13 – Расчеты значений

№	Операторы	Число операторов	Операнды	Число операндов
1	public	1	Property	3
2	async	1	property	4
3	Task<>	1	_houseFactory	1
4	if	1	apartmentFactory	2
5	else	1	property.OwnerId	1
6	=	5	property.Title	1
7	.	9	property.Address	1
8	()	6	property.Price	1
9	{	3	property.Rooms	1
10	}	3	property.Description	1
11	await	2	property.PhotoUrl	1
12	Add()	1	_context.Properties	1
13	SaveChangesAsync()	1	_changeNotifier.NotifyPropertyCreated	1
14	return	1	newProperty	3
	Итого:	56		23

На основе полученных данных произведен расчет характеристик, представленные в таблице 14, что позволило оценить количественные параметры сложности рассматриваемого метода.

Таблица 14 – Расчет характеристик

Характеристика	Значение
Словарь (n)	37
Длина (N)	107
Объем (V)	558,8
Вычислительная сложность (D)	15,52
Усилия (H)	8673,4

Все полученные значения метрики Холстеда находятся в допустимых пределах. Длина реализации не превышает длину программы, что указывает на отсутствие избыточности и логических недочетов. Потенциальная сложность кода невысока, а объем – сбалансирован относительно количества используемых конструкций. Метод реализован корректно, логически завершен и отражает рациональное использование средств языка программирования.

5.2.2 Метрика Чепина

Для дополнительной количественной оценки сложности кода была применена метрика Чепина, которая анализирует переменные, задействованные в работе программного модуля, и классифицирует их по функциональному назначению.

Метрика применяется к методу CreateProperty(), выполняющему добавление нового объекта недвижимости. Исходный код данного модуля представлен в листинге Ж.1 приложения Ж.

При расчете метрики Чепина используется весь набор переменных, формирующих список ввода-вывода, который затем делится на 4 группы:

– переменные ввода, используемые для вычислений и вывода (P);
– переменные, изменяемые внутри программы (M);
– переменные, контролирующие работу модуля (C);
– переменные, не используемые в программе (T).

Поскольку переменная может выполнять несколько функций одновременно, необходимо учитывать ее в каждой соответствующей группе. Формула Метрика Чепина:

$$Q = a1*P + a2*M + a3*C + a4*T,$$

где a1, a2, a3, a4 – весовые коэффициенты.

Весовые коэффициенты отражают различную степень влияния каждой группы на сложность программы. Наибольший вес (3) присвоен группе C, как определяющий поток управления. Остальные группы имеют коэффициенты:

a1=1, a2=2, a4=0,5. Коэффициент группы T не равен 0, поскольку «паразитные» переменные, не усложняя поток данных, могут затруднять понимание кода.

В методе CreateProperty() выделены: 8 входных переменных (P), 4 изменяемых переменных (M), 1 управляющая переменная (C) и 0 неиспользуемых переменных (T). Таким образом, формула подсчета метрики Чепина для данного модуля:

$$Q = 1*8+2*4+3*1+0,5*0=8+8+3+0=19$$

Полученное значение метрики Чепина свидетельствует о том, что данный компонент характеризуется относительно небольшим размером и невысокой сложностью кода.

6 Расчет бюджета разработки

6.1 Планирование разработки и оценка бюджета

Для эффективной организации процесса разработки приложения, а также оценки временных и финансовых затрат, была создана диаграмма Ганта с помощью инструмента Project Libre [23]. Весь процесс разработки приложения состоит из пяти этапов, включающих несколько подзадач, выполнение которых необходимо для перехода к следующему этапу.

Для разработки данного приложения выделены следующие этапы:

– постановка задания и анализ требований – включает в себя формулировку задачи, анализ аналогичных решений и формулирование требований к приложению;

– проектирование приложения – разработка структуры данных, логической и концептуальной модели базы данных, а также проектирование пользовательского интерфейса;

– реализация программной части – создание базы данных, серверной и клиентской части приложения;

– контроль качества программного обеспечения – проведение модульного и функционального тестирования, выбор метрик и анализ качества кода;

– оформление отчета – подготовка пояснительной записки.

Краткое описание задач по этапам приведено на рисунке 31 и в приложении А.

Такое поэтапное разбиение разработки позволило структурировать процесс и обеспечить последовательное выполнение всех задач. Каждый из этапов играет важную роль в создании надежного и функционального продукта. Благодаря четкому планированию удалось избежать перегрузок, соблюсти сроки и своевременно переходить от одного этапа к другому.

Название	Продолжи...	Начало	Окончание
Начало проекта	0 дней	08.04.2025, 08:00	08.04.2025, 08:00
Постановка задания и анализ требований	**4 дней**	**08.04.2025, 08:00**	**11.04.2025, 17:00**
Постановка задачи	1 день	08.04.2025, 08:00	08.04.2025, 17:00
Анализ систем аналогичного назначения	1 день	09.04.2025, 08:00	09.04.2025, 17:00
Формулировка требований	2 дней	10.04.2025, 08:00	11.04.2025, 17:00
Проектирование приложения	**6,5 дней**	**14.04.2025, 08:00**	**22.04.2025, 13:00**
Проектирование структур данных	1,5 дней	14.04.2025, 08:00	15.04.2025, 13:00
Концептуальная модель данных	1 день	15.04.2025, 13:00	16.04.2025, 13:00
Логическая модель базы данных	2 дней	16.04.2025, 13:00	18.04.2025, 13:00
Проектирование пользовательского интерфейса	2 дней	18.04.2025, 13:00	22.04.2025, 13:00
Реализация программной части	**16 дней**	**23.04.2025, 08:00**	**14.05.2025, 17:00**
Реализация базы данных	4 дней	23.04.2025, 08:00	28.04.2025, 17:00
Реализация серверной части	7 дней	29.04.2025, 08:00	07.05.2025, 17:00
Реализация клиентской части	5 дней	08.05.2025, 08:00	14.05.2025, 17:00
Контроль качества программного обеспечения	**11,5 дней**	**15.05.2025, 08:00**	**30.05.2025, 13:00**
Обоснование выбора режима тестирования и метрик кода	0,5 дней	15.05.2025, 08:00	15.05.2025, 13:00
Тестирование программных средств	5 дней	15.05.2025, 13:00	22.05.2025, 13:00
Модульное тестирование	2 дней	22.05.2025, 13:00	26.05.2025, 13:00
Тестирование черным ящиком	1 день	26.05.2025, 13:00	27.05.2025, 13:00
Метрики качества программного кода	2 дней	27.05.2025, 13:00	29.05.2025, 13:00
Метрика Чепина	1 день	29.05.2025, 13:00	30.05.2025, 13:00
Оформление отчета	**5 дней**	**30.05.2025, 13:00**	**06.06.2025, 13:00**
Оформление пояснительной записки	5 дней	30.05.2025, 13:00	06.06.2025, 13:00
Конец проекта	0 дней	06.06.2025, 13:00	06.06.2025, 13:00

Рисунок 31 – Сведения об этапах разработки приложения

Для визуального представления временных параметров разработки приложения была составлена диаграмма Ганта, представленная на рисунке 32 и в приложении А.

Диаграмма Ганта отразила последовательность выполнения задач, их продолжительность, а также взаимосвязи между ними. Такой подход способствовал более эффективному распределению ресурсов.

С помощью диаграммы стало легче контролировать ход выполнения проекта и следить за соблюдением сроков. Она может наглядно показать какие задачи уже завершены, какие находятся в процессе, а какие только предстоят. Это позволяет вовремя замечать отклонение от плана и при необходимости пересматривать график работы, чтобы не выйти за рамки установленного времени.

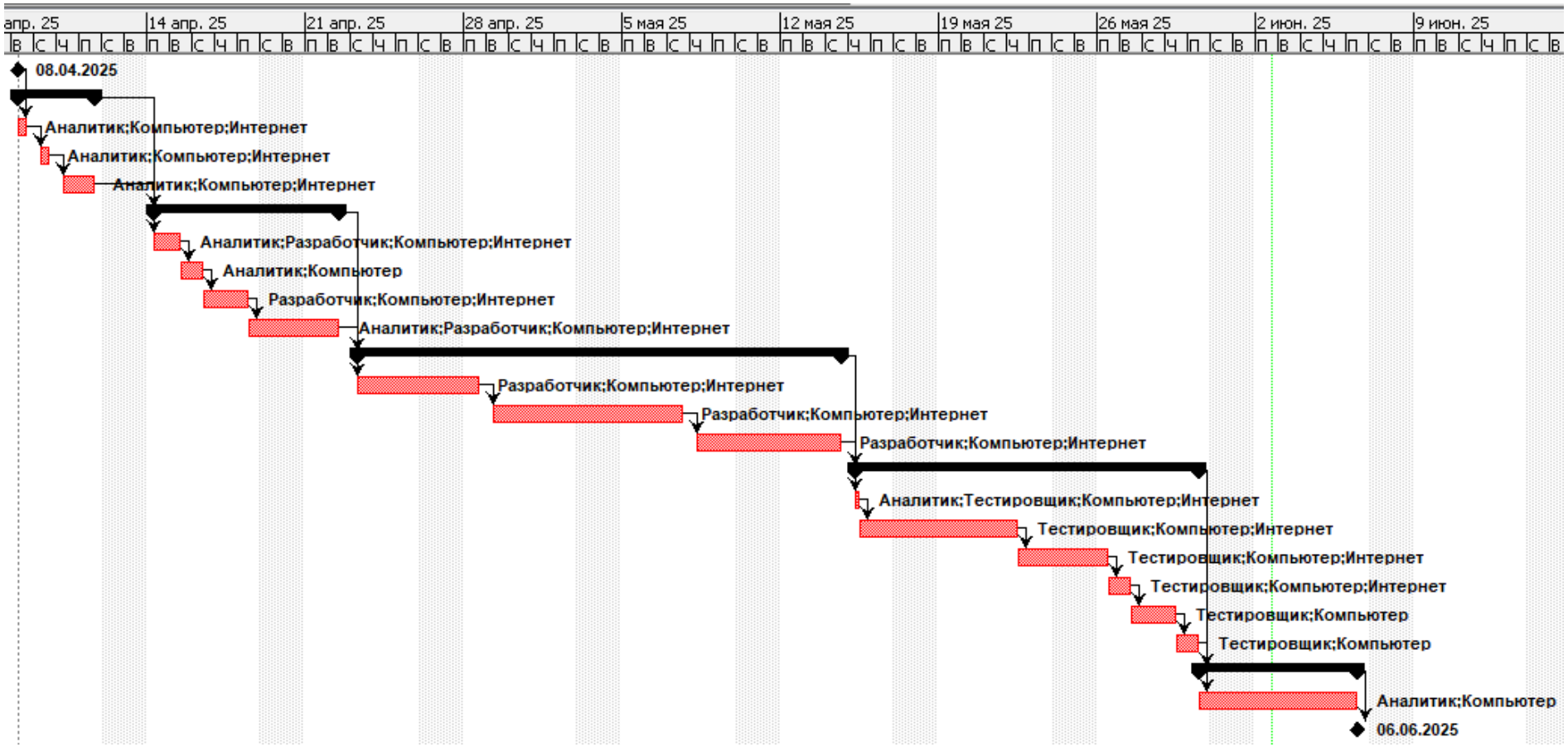

Рисунок 32 – Этапы разработки приложения на диаграмме Ганта

Первый этап включает задачи, связанные с постановкой цели разработки, анализом систем аналогичного назначения и формированием перечня требований к создаваемому программному продукту. Второй этап включает в себя разработку структур данных, создание концептуальной и логической модели данных, также интерфейса будущего приложения. Третий этап посвящен реализации всех модулей системы – базы данных, серверной и клиентской частей. Четвертый этап завершает цикл разработки и охватывает мероприятия по проверке качества: обоснование методик тестирования, проведение модульных и функциональных тестов, анализ метрик программного кода и итоговая подготовка проектной документации.

Для расчета бюджета проекта была произведена оценка стоимости всех задействованных ресурсов. Соответствующие данные представлены на рисунке 33.

	Название	Стандартная ставка	Ставка сверхурочных
	Аналитик	180,00 ₽/час	220,00 ₽/час
	Разработчик	230,00 ₽/час	270,00 ₽/час
	Тестировщик	150,00 ₽/час	190,00 ₽/час
	Компьютер	50000,00 ₽/час	
	Интернет	0,00 ₽	

Рисунок 33 – Стоимость задействованных ресурсов

На рисунке 34 и в приложении А, приведена итоговая оценка общего бюджета проекта с учетом продолжительности всех этапов и стоимости применяемых ресурсов.

Расчет бюджета позволил заранее оценить финансовую сторону разработки и избежать непредвиденных затрат. Учет стоимости рабочего персонала и используемого оборудования помог сформировать реалистичное представление о необходимых ресурсах.

Название	Стоимость
Начало проекта	0,00 ₽
⊟Постановка задания и анализ требований	**5760,00 ₽**
Постановка задачи	1440,00 ₽
Анализ систем аналогичного назначения	1440,00 ₽
Формулировка требований	2880,00 ₽
⊟Проектирование приложения	**16600,00 ₽**
Проектирование структур данных	4920,00 ₽
Концептуальная модель данных	1440,00 ₽
Логическая модель базы данных	3680,00 ₽
Проектирование пользовательского интерфейса	6560,00 ₽
⊟Реализация программной части	**29440,00 ₽**
Реализация базы данных	7360,00 ₽
Реализация серверной части	12880,00 ₽
Реализация клиентской части	9200,00 ₽
⊟Контроль качества программного обеспечения	**14520,00 ₽**
Обоснование выбора режима тестирования и метрик кода	1320,00 ₽
Тестирование программных средств	6000,00 ₽
Модульное тестирование	2400,00 ₽
Тестирование черным ящиком	1200,00 ₽
Метрики качества программного кода	2400,00 ₽
Метрика Чепина	1200,00 ₽
⊟Оформление отчета	**7200,00 ₽**
Оформление пояснительной записки	7200,00 ₽
Конец проекта	0,00 ₽
	Итого: 73520 ₽

Рисунок 34 – Общая оценка финансовых затрат

Таким образом, итоговый анализ показывает, что совокупные затраты на разработку составляют 73 520 рублей, а общая продолжительность выполнения проекта – 43 дня.

Заключение

В результате выполнения выпускной квалификационной работы было разработано веб-приложение, предназначенное для размещения, поиска и бронирования объектов жилых помещений. В системе реализованы ключевые функции, необходимые как для пользователей, ищущих жилье, так и для администраторов, управляющих контентом и пользовательскими данными.

В процессе работы был проведен анализ существующих решений на рынке аренды жилья. Это позволило выделить наиболее важные требования к разрабатываемому приложению и учесть недостатки аналогов. Также было проведено планирование разработки, включая оценку сроков и затрат.

На этапе проектирования созданы модели базы данных и архитектура системы. Вся логика приложения реализована на платформе ASP.NET Core с использованием Entity Framework и технологии клиентской разработки (HTML, CSS, JavaScript). Также использовались паттерны проектирования, что повысило надежность и расширяемость системы.

Работа программы проверялась при помощи как модульного, так и функционального тестирования. Все запланированные проверки завершились успешно, что подтверждает корректную работу приложения.

Таким образом, задача, поставленная по реализации приложения аренды жилых помещений, выполнена в полном объеме.

Список литературы

1. PropTechBuzz. Обзор мирового рынка PropTech [Электронный ресурс]. – URL: https://www.proptechbuzz.com (Дата обращения: 26.05.2024).

2. Федеральная служба государственной статистики. Цифровизация в жилищной сфере: тренды 2023 года [Электронный ресурс]. – URL: https://rosstat.gov.ru (Дата обращения: 26.05.2024).

3. Ключевые аспекты администрирования информационных систем [Электронный ресурс] // Gendalf.ru. – URL: https://gendalf.ru/news/marketing/klyuchevye-aspekty-administrirovaniya/ (Дата обращения: 26.05.2024).

4. REST API – архитектурный стиль взаимодействия компонентов [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Glossary/REST (Дата обращения: 26.05.2024).

5. Сайт ЦИАН [Электронный ресурс]. – URL: https://www.cian.ru (Дата обращения: 28.05.2024).

6. Сайт Авито Недвижимость [Электронный ресурс]. – Режим доступа: https://www.avito.ru (Дата обращения: 28.05.2024).

7. Сайт Яндекс.Недвижимость [Электронный ресурс]. – Режим доступа: https://realty.yandex.ru (Дата обращения: 28.05.2024).

8. DomClick – сервис по работе с недвижимостью от Сбербанка [Электронный ресурс]. – Режим доступа: https://www.domclick.ru (Дата обращения: 28.05.2024).

9. СберАренда – платформа для аренды жилья [Электронный ресурс]. – Режим доступа: https://sberarenda.ru (Дата обращения: 28.05.2024).

10. Date C. J. Введение в системы баз данных. – М.: Вильямс, 2013. – 944 с.

11. Sky.pro. Логическая модель базы данных: что это и как её создать [Электронный ресурс]. – Режим доступа: https://sky.pro/wiki/sql/logicheskaya-model-bazy-dannyh-chto-eto-i-kak-eyo-sozdat/ (Дата обращения: 30.05.2024).

12. Refactoring.Guru. Паттерн Strategy [Электронный ресурс]. – URL: https://refactoring.guru/ru/design-patterns/strategy (Дата обращения: 02.06.2024).

13. Refactoring.Guru. Паттерн Observer [Электронный ресурс]. – URL: https://refactoring.guru/ru/design-patterns/observer (Дата обращения: 02.06.2024).

14. Корольков А. И. Базы данных. Проектирование, реализация и сопровождение. – СПб.: Питер, 2021. – 480 с.

15. Фаулер М. Архитектура корпоративных приложений. – М.: Вильямс, 2022. – 560 с.

16. Рудаков А. Многослойная архитектура приложений: как это устроено [Электронный ресурс] // Habr.com. – URL: https://habr.com/ru/articles/667922/ (Дата обращения: 30.05.2024).

17. Refactoring.Guru. Паттерн Factory Method [Электронный ресурс]. – URL: https://refactoring.guru/ru/design-patterns/factory-method (Дата обращения: 02.06.2024).

18. MDN Web Docs. REST API [Электронный ресурс]. – URL: https://developer.mozilla.org/ru/docs/Glossary/REST (Дата обращения: 02.06.2024).

19. Sky.pro. Что такое юнит-тесты и модульное тестирование [Электронный ресурс]. – URL: https://sky.pro/wiki/profession/chto-takoe-unit-testy-i-modulnoe-testirovanie/ (Дата обращения: 02.06.2024).

20. GeeksforGeeks. Integration Testing [Электронный ресурс]. – URL: https://www.geeksforgeeks.org/integration-testing/ (Дата обращения: 03.06.2024).

21. Васильев В. В. Метрики программного кода: теория и практика // Системный анализ и прикладная информатика. – 2020. – №2(36). – С. 54–59. – URL: https://elibrary.ru/item.asp?id=44579924 (Дата обращения: 03.06.2024).

22. Никитин П. А. Методы оценки сложности программных модулей на основе метрик Чепина // Информационные технологии. – 2021. – №5. – С. 67–72. – URL: https://cyberleninka.ru/article/n/metodika-otsenki-slozhnosti-na-osnove-metodov-chepina (Дата обращения: 03.06.2024).

23. ProjectLibre – программа управления проектами [Электронный ресурс]. – Режим доступа: https://www.projectlibre.com (Дата обращения: 30.05.2024).

Приложение А
(рекомендуемое)
Графическая часть

Рисунок А.1 – Диаграмма вариантов использования

Название	Продолжи...	Начало	Окончание
Начало проекта	0 дней	08.04.2025, 08:00	08.04.2025, 08:00
⊟Постановка задания и анализ требований	**4 дней**	**08.04.2025, 08:00**	**11.04.2025, 17:00**
Постановка задачи	1 день	08.04.2025, 08:00	08.04.2025, 17:00
Анализ систем аналогичного назначения	1 день	09.04.2025, 08:00	09.04.2025, 17:00
Формулировка требований	2 дней	10.04.2025, 08:00	11.04.2025, 17:00
⊟Проектирование приложения	**6,5 дней**	**14.04.2025, 08:00**	**22.04.2025, 13:00**
Проектирование структур данных	1,5 дней	14.04.2025, 08:00	15.04.2025, 13:00
Концептуальная модель данных	1 день	15.04.2025, 13:00	16.04.2025, 13:00
Логическая модель базы данных	2 дней	16.04.2025, 13:00	18.04.2025, 13:00
Проектирование пользовательского интерфейса	2 дней	18.04.2025, 13:00	22.04.2025, 13:00
⊟Реализация программной части	**16 дней**	**23.04.2025, 08:00**	**14.05.2025, 17:00**
Реализация базы данных	4 дней	23.04.2025, 08:00	28.04.2025, 17:00
Реализация серверной части	7 дней	29.04.2025, 08:00	07.05.2025, 17:00
Реализация клиентской части	5 дней	08.05.2025, 08:00	14.05.2025, 17:00
⊟Контроль качества программного обеспечения	**11,5 дней**	**15.05.2025, 08:00**	**30.05.2025, 13:00**
Обоснование выбора режима тестирования и метрик кода	0,5 дней	15.05.2025, 08:00	15.05.2025, 13:00
Тестирование программных средств	5 дней	15.05.2025, 13:00	22.05.2025, 13:00
Модульное тестирование	2 дней	22.05.2025, 13:00	26.05.2025, 13:00
Тестирование черным ящиком	1 день	26.05.2025, 13:00	27.05.2025, 13:00
Метрики качества программного кода	2 дней	27.05.2025, 13:00	29.05.2025, 13:00
Метрика Чепина	1 день	29.05.2025, 13:00	30.05.2025, 13:00
⊟Оформление отчета	**5 дней**	**30.05.2025, 13:00**	**06.06.2025, 13:00**
Оформление пояснительной записки	5 дней	30.05.2025, 13:00	06.06.2025, 13:00
Конец проекта	0 дней	06.06.2025, 13:00	06.06.2025, 13:00

Рисунок А.2 – Сведения об этапах разработки приложения

Название	Стоимость
Начало проекта	0,00 ₽
⊟Постановка задания и анализ требований	**5760,00 ₽**
Постановка задачи	1440,00 ₽
Анализ систем аналогичного назначения	1440,00 ₽
Формулировка требований	2880,00 ₽
⊟Проектирование приложения	**16600,00 ₽**
Проектирование структур данных	4920,00 ₽
Концептуальная модель данных	1440,00 ₽
Логическая модель базы данных	3680,00 ₽
Проектирование пользовательского интерфейса	6560,00 ₽
⊟Реализация программной части	**29440,00 ₽**
Реализация базы данных	7360,00 ₽
Реализация серверной части	12880,00 ₽
Реализация клиентской части	9200,00 ₽
⊟Контроль качества программного обеспечения	**14520,00 ₽**
Обоснование выбора режима тестирования и метрик кода	1320,00 ₽
Тестирование программных средств	6000,00 ₽
Модульное тестирование	2400,00 ₽
Тестирование черным ящиком	1200,00 ₽
Метрики качества программного кода	2400,00 ₽
Метрика Чепина	1200,00 ₽
⊟Оформление отчета	**7200,00 ₽**
Оформление пояснительной записки	7200,00 ₽
Конец проекта	0,00 ₽
	Итого: 73520 ₽

Рисунок А.3 – Общая оценка финансовых затрат

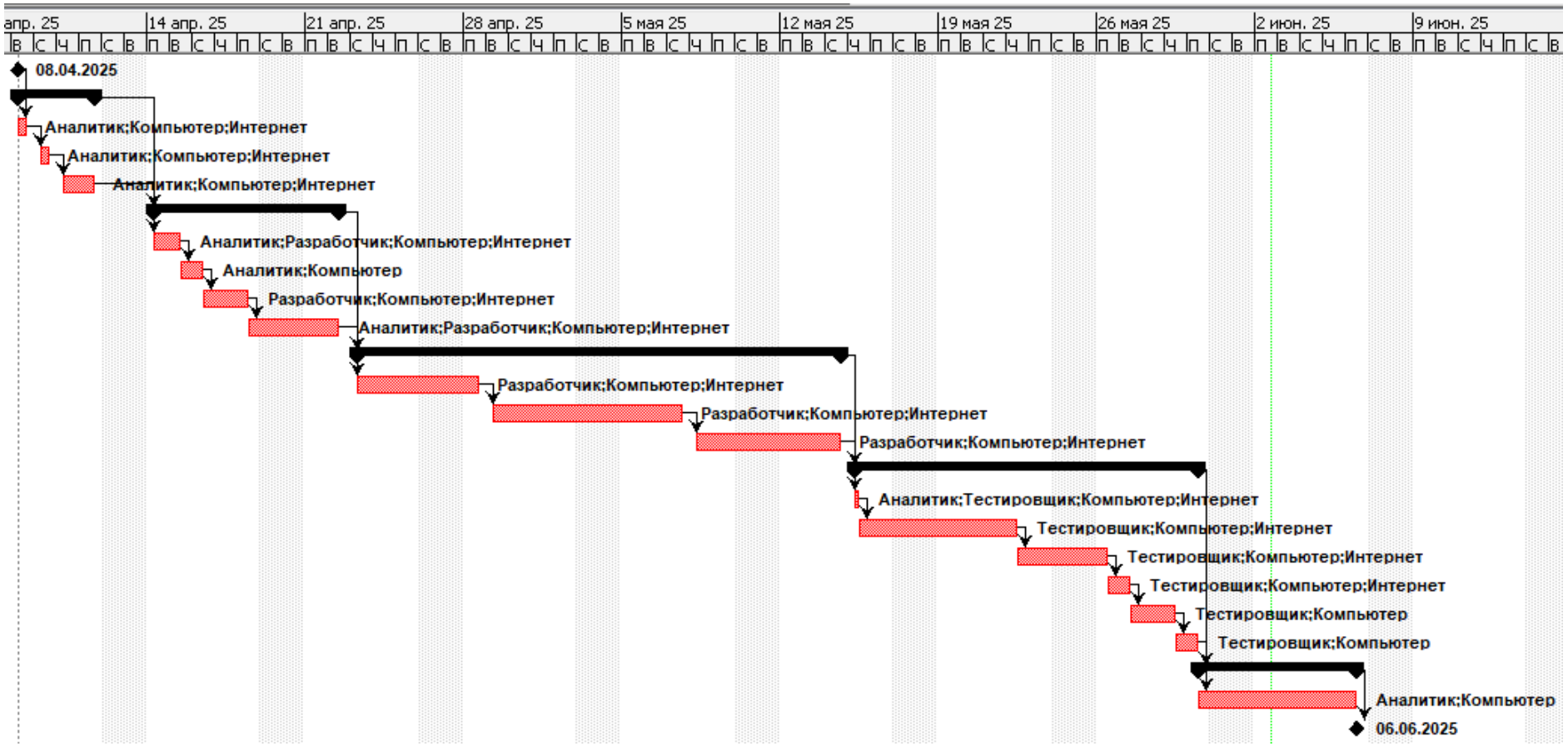

Рисунок А.4 – Этапы разработки приложения на диаграмме Ганта

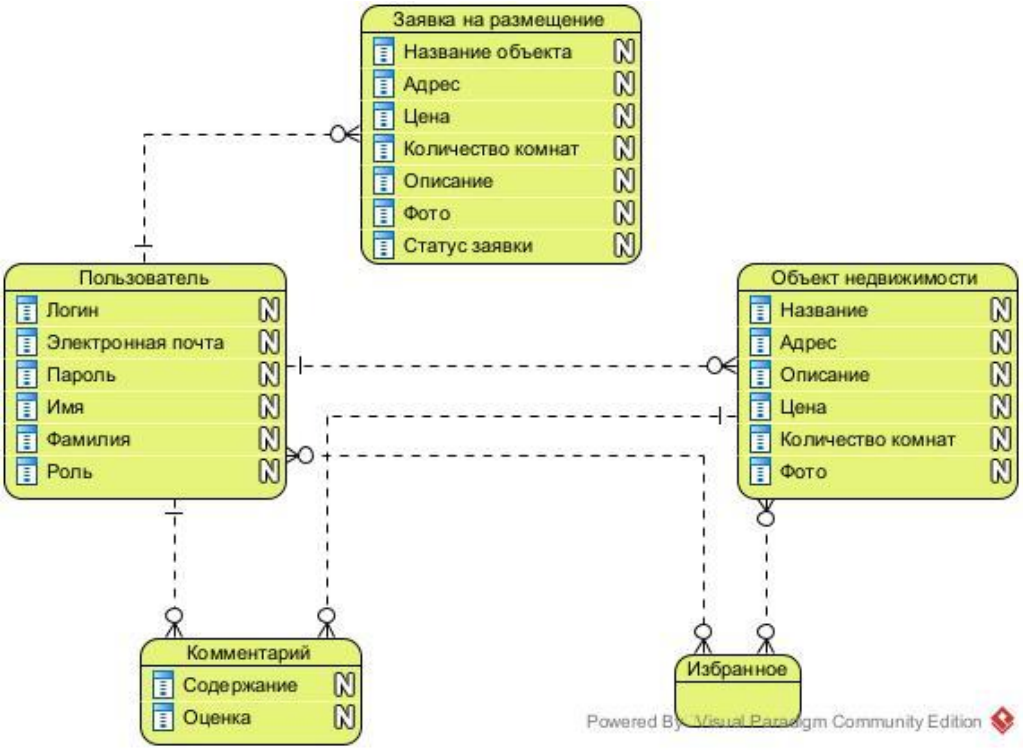

Рисунок А.5 – Диаграмма концептуальной модели

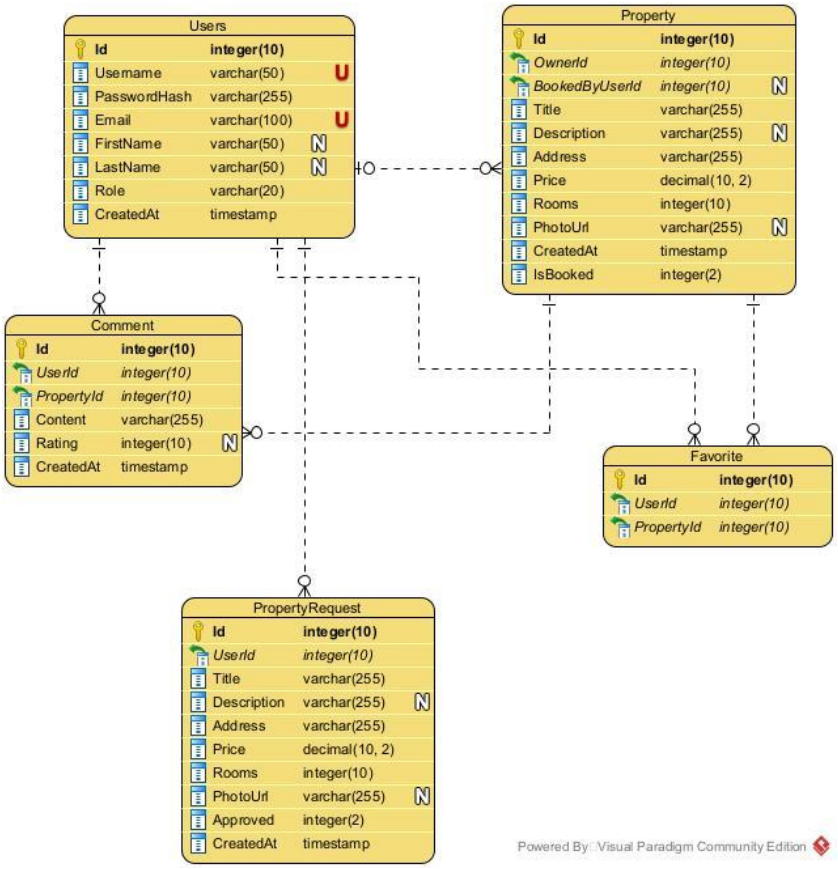

Рисунок А.6 – Диаграмма логической модели

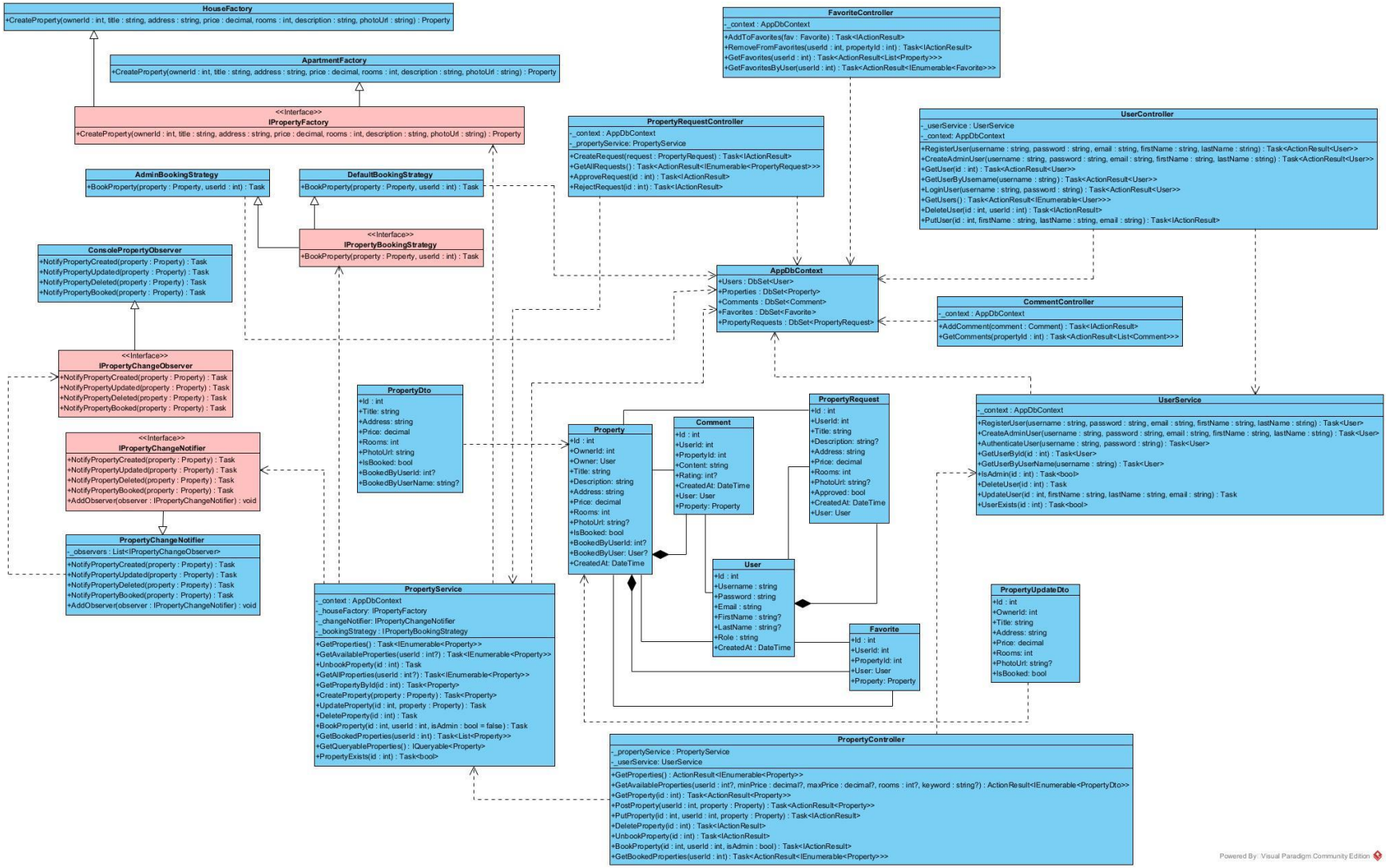

Рисунок А.7 – Диаграмма классов

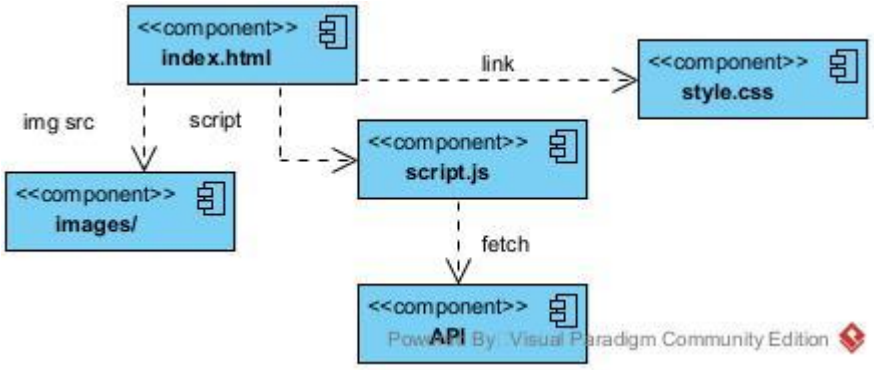

Рисунок А.8 – Диаграмма компонентов клиентской части

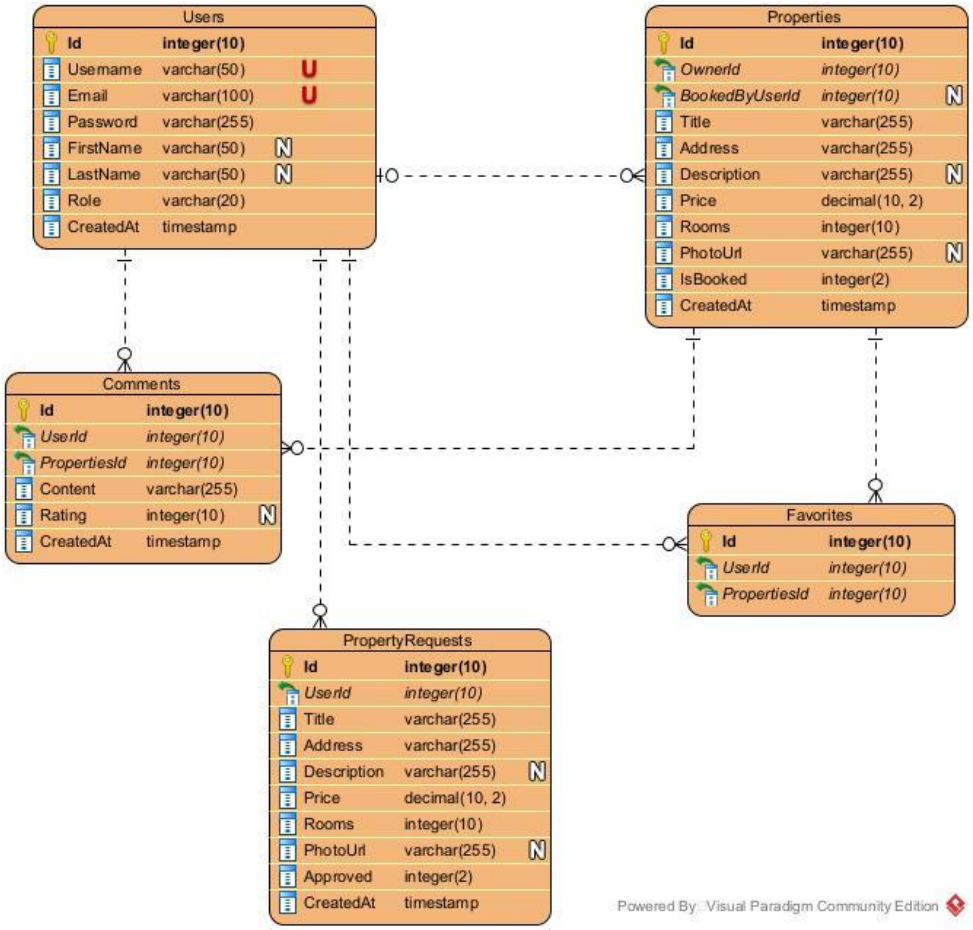

Рисунок А.9 – Диаграмма физической модели

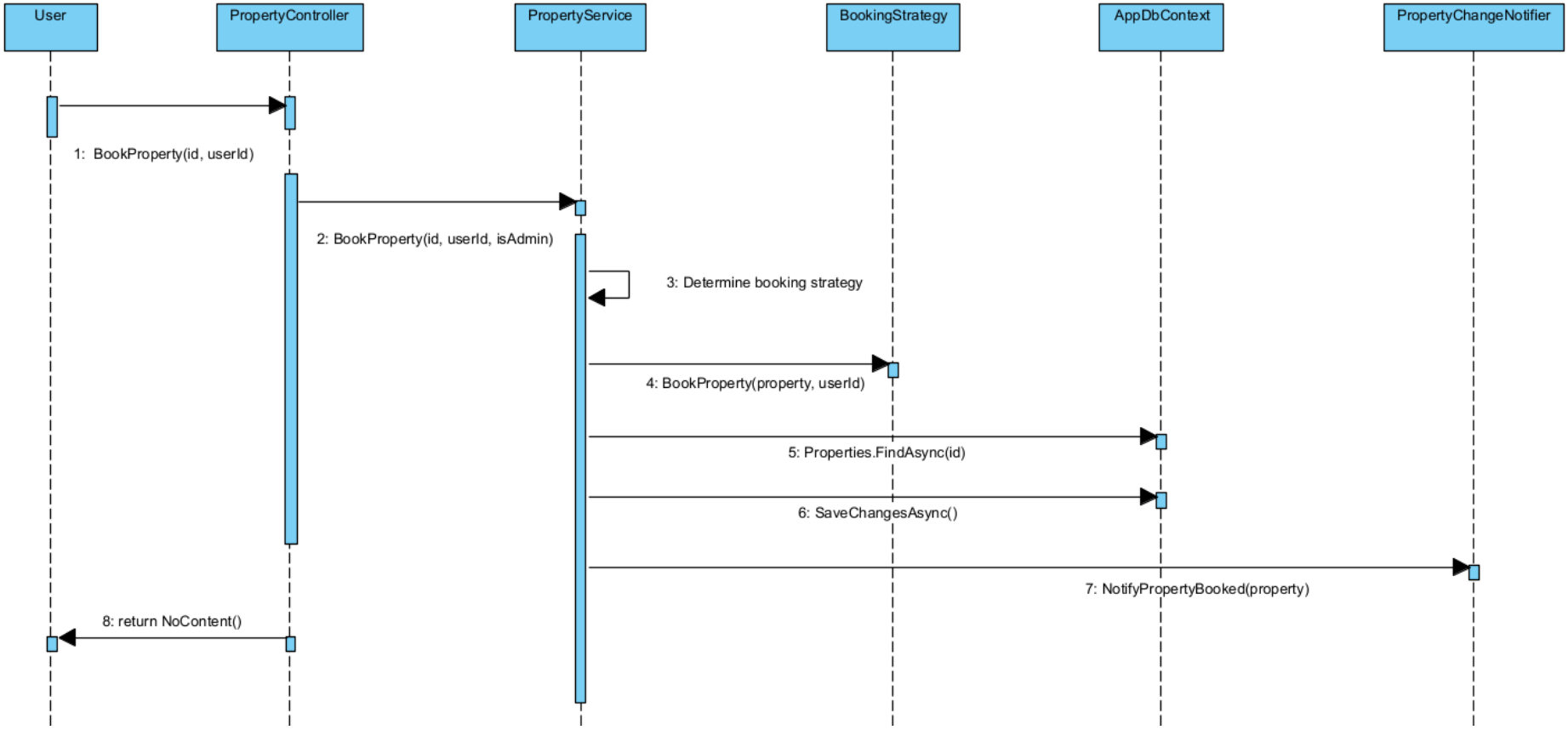

Рисунок А.10 – Диаграмма коммуникаций

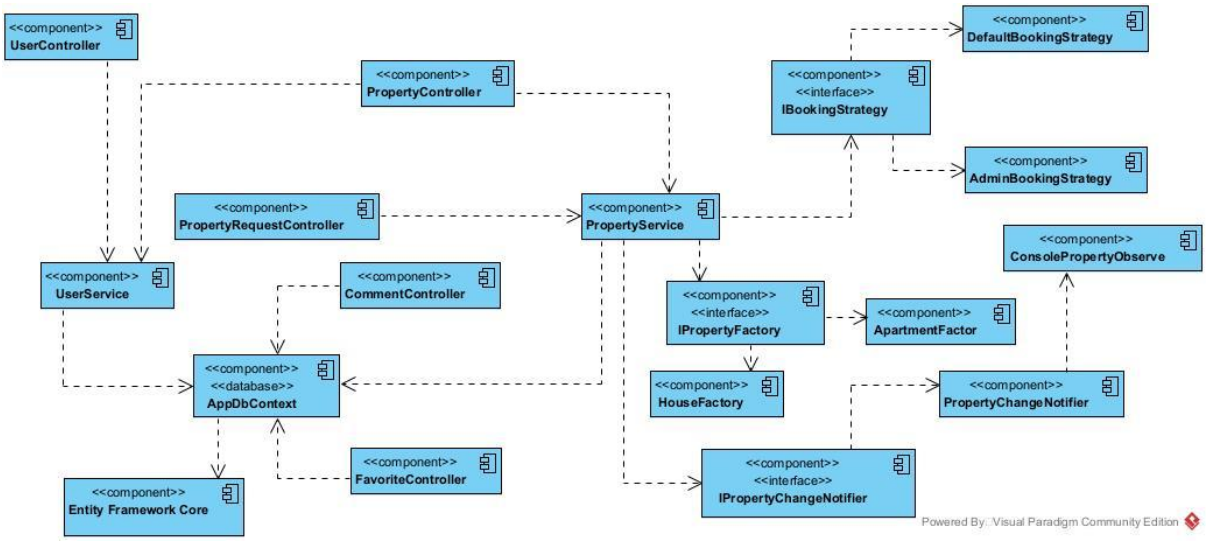

Рисунок А.11 – Диаграмма компонентов серверной части

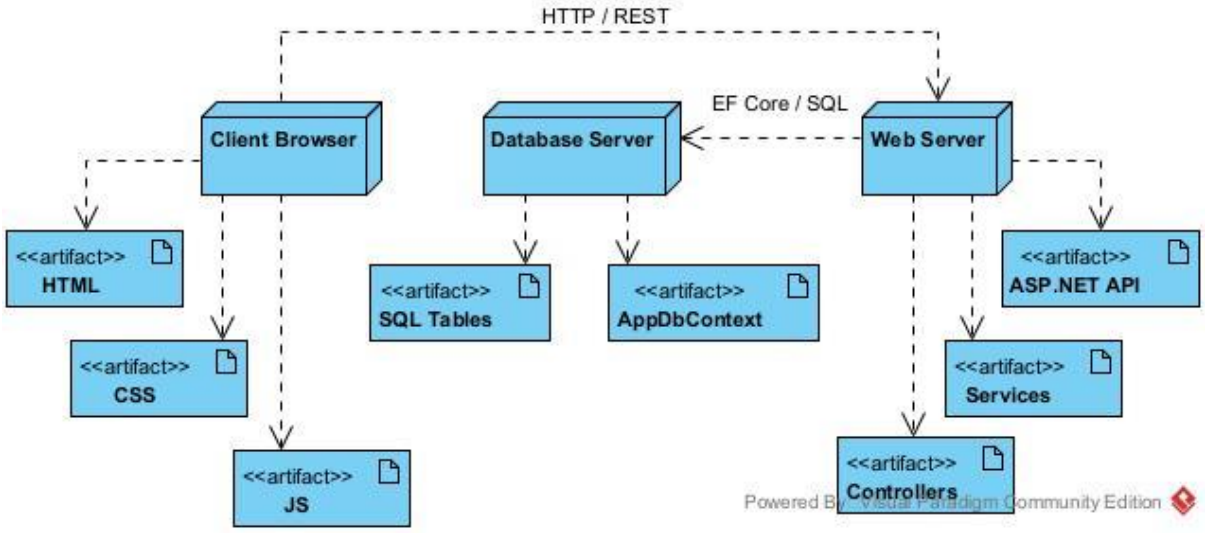

Рисунок А.12 – Диаграмма развертывания

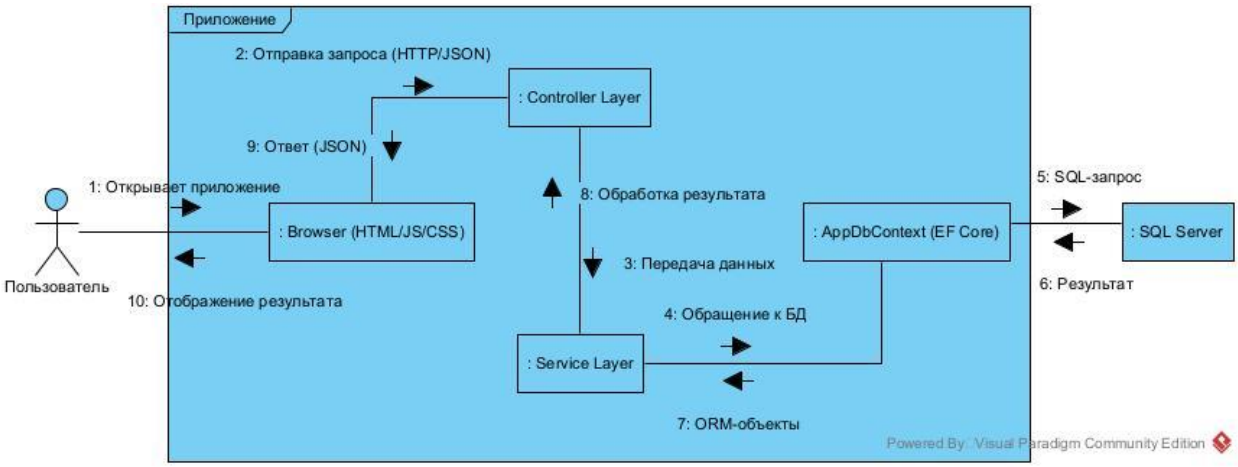

Рисунок А.13 – Диаграмма коммуникаций

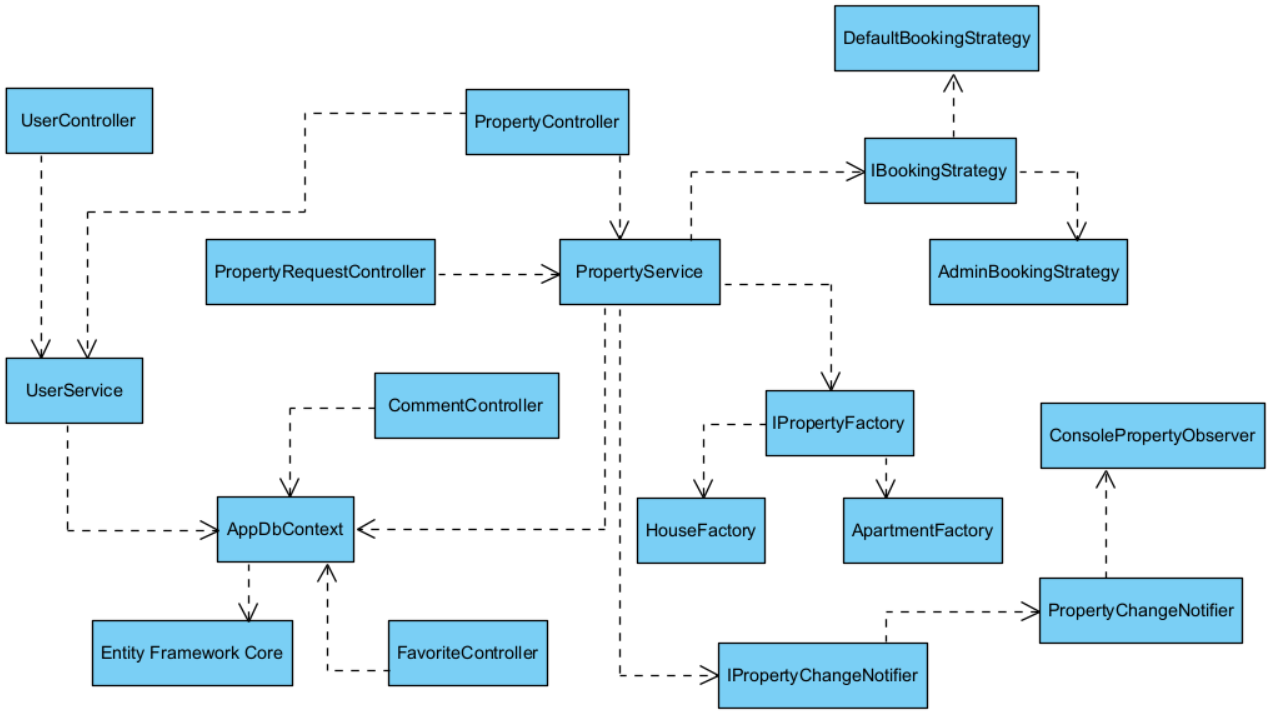

Рисунок А.14 – Структурно-функциональная диаграмма архитектуры серверной части

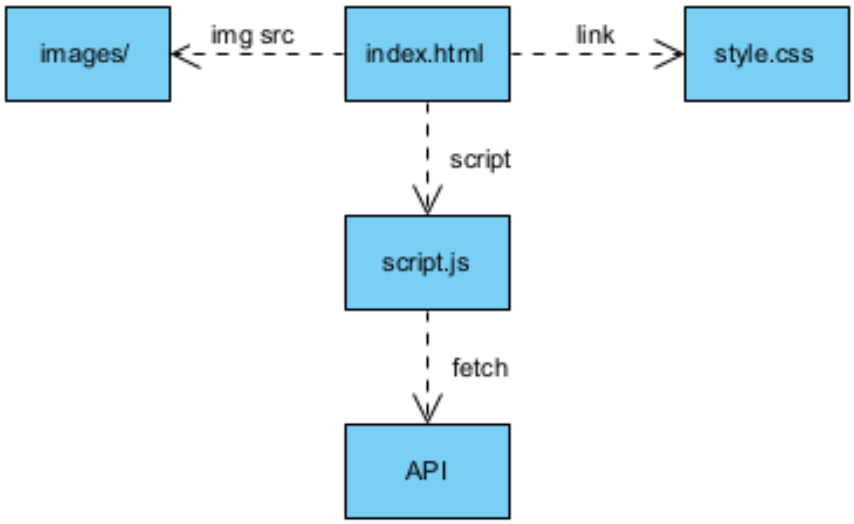

Рисунок А.15 – Структурно-функциональная диаграмма клиентской части

Приложение Б

(рекомендуемое)

Спецификация атрибутов сущности и связи между ними

Таблица Б.1 – Спецификация атрибутов сущности Users

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID пользователя	Yes	No
Username	VARCHAR(50)	Yes	Имя пользователя	No	No
Email	VARCHAR(100)	Yes	Адрес электронной почты	No	No
Password	VARCHAR(255)	Yes	Пароль пользователя	No	No
FirstName	VARCHAR(50)	No	Имя пользователя	No	No
LastName	VARCHAR(50)	No	Фамилия пользователя	No	No
Role	VARCHAR(20)	Yes	Роль	No	No
CreatedAt	DATETIME2	Yes	Дата регистрации	No	No

Таблица Б.2 – Спецификация атрибутов сущности Properties

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID объекта недвижимости	Yes	No
Title	VARCHAR(255)	Yes	Заголовок объявления	No	No
Address	VARCHAR(255)	Yes	Адрес объекта	No	No
Description	VARCHAR(255)	No	Описание недвижимости	No	No
Price	DECIMAL(10,2)	Yes	Стоимость аренды	No	No
Rooms	INTEGER	Yes	Количество комнат	No	No
PhotoUrl	VARCHAR(255)	No	Ссылка на изображение объекта	No	No
IsBooked	BOOLEAN	Yes	Флаг, указывающий, забронирован ли объект	No	No
OwnerId	INTEGER	Yes	Внешний ключ администратора, разместившего объект	No	Yes
BookedByUserId	INTEGER	No	Внешний ключ на пользователя, забронировавшего объект	No	Yes
CreatedAt	DATETIME2	Yes	Дата размещения объявления	No	No

Таблица Б.3 – Спецификация атрибутов сущности Comments

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID комментария	Yes	No
UserId	INTEGER	Yes	Внешний ключ на пользователя	No	Yes
PropertyId	INTEGER	Yes	Внешний ключ на объект недвижимости	No	Yes
Content	VARCHAR(255)	Yes	Текст комментария	No	No
Rating	INTEGER	No	Оценка от 1 до 5	No	No

Продолжение таблицы Б.3

Name	Datatype	Required	Definition	Is PK	Is FK
CreatedAt	DATETIME2	Yes	Дата размещения объявления	No	No

Таблица Б.4 – Спецификация атрибутов сущности Favorites

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID записи избранного	Yes	No
UserId	INTEGER	Yes	Внешний ключ на пользователя	No	Yes
PropertyId	INTEGER	Yes	Внешний ключ на объект недвижимости	No	Yes

Таблица Б.5 – Спецификация атрибутов сущности PropertyRequests

Name	Datatype	Required	Definition	Is PK	Is FK
Id	INTEGER	Yes	ID записи избранного	Yes	No
UserId	INTEGER	Yes	Внешний ключ на пользователя	No	Yes
Title	VARCHAR(255)	Yes	Заголовок объявления	No	No
Address	VARCHAR(255)	Yes	Адрес объекта	No	No
Description	VARCHAR(255)	No	Описание недвижимости	No	No
Price	DECIMAL(10,2)	Yes	Стоимость аренды	No	No
Rooms	INTEGER	Yes	Количество комнат	No	No
PhotoUrl	VARCHAR(255)	No	Ссылка на изображение объекта	No	No
Approved	BOOLEAN	Yes	Флаг утверждения заявки	No	No
CreatedAt	DATETIME2	Yes	Дата размещения объявления	No	No

Таблица Б.6 – Связи между сущностями базы данных

From Table	To Table	FK Field	Cardinality	On Delete Behavior	Nullable
Properties	Users	OwnerId	Many to One	CASCADE	No
Properties	Users	BookedByUserId	Many to One	SET NULL / Custom	Yes
Comments	Users	UserId	Many to One	RESTRICT	No
Comments	Properties	PropertyId	Many to One	CASCADE	No
Favorites	Users	UserId	Many to One	RESTRICT	No
Favorites	Properties	PropertyId	Many to One	RESTRICT	No

Приложение В

(обязательное)

Листинг серверной части

Листинг В.1 – Файл CommentController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RentalPlatform.Data;
using RentalPlatform.Models;

namespace RentalPlatform.Controllers
{
    [ApiController]
    [Route("api/comments")]
    public class CommentHandlerController : ControllerBase
    {
        private readonly MainDbContext _database;

        public CommentHandlerController(MainDbContext database)
        {
            _database = database;
        }

        [HttpPost("add")]
        public async Task<IActionResult> SubmitComment([FromBody] Comment
newComment)
        {
            _database.Comments.Add(newComment);
            await _database.SaveChangesAsync();
            return Ok(newComment);
        }

        [HttpGet("by-property/{propertyId}")]
        public async Task<ActionResult<IEnumerable<Comment>>> GetComments(int
propertyId)
        {
            _context.Comments.Add(comment);
            await _context.SaveChangesAsync();
            return Ok(comment);

            if (comment.Rating is < 1 or > 5)
                return BadRequest("Оценка должна быть от 1 до 5");

        }

        [HttpGet("{propertyId}")]
        public async Task<ActionResult<List<Comment>>> GetComments(int
propertyId)
        {
            var comments = await _context.Comments
                .Where(c => c.PropertyId == propertyId)
                .ToListAsync();
            return Ok(comments);
        }
    }

}
```

Листинг В.2 – Файл FavoriteController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RentalPlatform.Data;
using RentalPlatform.Models;

namespace RentalPlatform.Controllers
```

```
{
    [ApiController]
    [Route("api/favorites")]
    public class FavoritesController : ControllerBase
    {
        private readonly MainDbContext _database;

        public FavoritesController(MainDbContext database)
        {
            _database = database;
        }

        [HttpPost("add")]
        public async Task<IActionResult> MarkAsFavorite([FromBody] Favorite
item)
        {
            _database.Favorites.Add(item);
            await _database.SaveChangesAsync();
            return Ok(item);
        }

        [HttpGet("{userId}")]
        public async Task<ActionResult<IEnumerable<Favorite>>>
GetUserFavorites(int userId)
        {        {
            var favs = await _database.Favorites
                .Where(f => f.UserId == userId)
                .Include(f => f.Property)
                .ToListAsync();

            return Ok(favs);
        }
            if (!exists)
            {
                _context.Favorites.Add(fav);
                await _context.SaveChangesAsync();
            }

            return Ok();
        }

        [HttpDelete("{userId}/{propertyId}")]
        public async Task<IActionResult> RemoveFromFavorites(int userId, int
propertyId)
        {
            var favorite = await _context.Favorites
                .FirstOrDefaultAsync(f => f.UserId == userId && f.PropertyId
== propertyId);

            if (favorite == null)
            {
                return NotFound("Избранное не найдено.");
            }

            _context.Favorites.Remove(favorite);
            await _context.SaveChangesAsync();

            return NoContent();
        }

        [HttpGet("{userId}")]
        public async Task<ActionResult<List<Property>>> GetFavorites(int
userId)
```

```
        {
            var properties = await _context.Favorites
                .Where(f => f.UserId == userId)
                .Select(f => f.Property)
                .ToListAsync();
            return Ok(properties);
        }

        [HttpGet("user/{userId}")]
        public async Task<ActionResult<IEnumerable<Favorite>>>
GetFavoritesByUser(int userId)
        {
            var favorites = await _context.Favorites
                .Include(f => f.Property)
                .Where(f => f.UserId == userId)
                .ToListAsync();

            return Ok(favorites);
        }

    }
}
```

Листинг В.3 – Файл PropertyController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RentalPlatform.Models;
using RentalPlatform.Services;

namespace RentalPlatform.Controllers
{
    [ApiController]
    [Route("api/properties")]
    public class HousingController : ControllerBase
    {
        private readonly PropertyService _propertyService;
        private readonly UserService _userService;

        public HousingController(PropertyService propService, UserService
userService)
        {
            _propertyService = propService;
            _userService = userService;
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<Property>>> FetchAll()
        {
            var result = await _propertyService.GetAllProperties(null);
            return Ok(result);
        }

        [HttpGet("available")]
        public async Task<ActionResult<IEnumerable<PropertyDto>>>
GetAvailable()
        {
            var properties = await
_propertyService.GetAllAvailableProperties();
            return Ok(properties);
            [FromQuery] int? userId,
            [FromQuery] decimal? minPrice,
            [FromQuery] decimal? maxPrice,
```

```
            [FromQuery] int? rooms,
            [FromQuery] string? keyword)
        {
            var isAdmin = userId != null && await
_userService.IsAdmin(userId.Value);

            IQueryable<Property> query =
_propertyService.GetQueryableProperties()
                .Include(p => p.BookedByUser);

            if (!isAdmin)
                query = query.Where(p => !p.IsBooked);

            if (minPrice != null)
                query = query.Where(p => p.Price >= minPrice);

            if (maxPrice != null)
                query = query.Where(p => p.Price <= maxPrice);

            if (rooms != null)
                query = query.Where(p => p.Rooms == rooms);

            if (!string.IsNullOrWhiteSpace(keyword))
            {
                var lowered = keyword.ToLower();
                query = query.Where(p => p.Title.ToLower().Contains(lowered)
|| p.Address.ToLower().Contains(lowered));
            }

            var list = await query.ToListAsync();

            var dtoList = list.Select(p => new PropertyDto
            {
                Id = p.Id,
                Title = p.Title,
                Address = p.Address,
                Price = p.Price,
                Rooms = p.Rooms,
                PhotoUrl = p.PhotoUrl,
                IsBooked = p.IsBooked,
                BookedByUserId = p.BookedByUser?.Id,
                BookedByUserName = p.BookedByUser != null ?
$"{p.BookedByUser.FirstName} {p.BookedByUser.LastName}" : null
            }).ToList();

            return Ok(dtoList);
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<Property>> GetProperty(int id)
        {
            var property = await _propertyService.GetPropertyById(id);
            if (property == null)
            {
                return NotFound();
            }
            return Ok(property);
        }

        [HttpPost]
        public async Task<ActionResult<Property>> PostProperty([FromQuery]
int userId, Property property)
        {
```

```
            if (!await _userService.IsAdmin(userId))
            {
                return Unauthorized("Only admin can add new property.");
            }
            var createdProperty = await
_propertyService.CreateProperty(property);
            return CreatedAtAction(nameof(GetProperty), new { id =
createdProperty.Id }, createdProperty);
        }

        [HttpPut("unbook/{id}")]
        public async Task<IActionResult> UnbookProperty(int id)
        {
            try
            {
                await _propertyService.UnbookProperty(id);
                return NoContent();
            }
            catch (KeyNotFoundException ex)
            {
                return NotFound(ex.Message);
            }
        }

        [HttpPut("{id}")]
        public async Task<IActionResult> PutProperty(int id, [FromQuery] int
userId, Property property)
        {
            try
            {
                if (id != property.Id)
                {
                    return BadRequest("Property id does not match URL id");
                }
                if (!await _userService.IsAdmin(userId))
                {
                    return Unauthorized("Only admin can update property.");
                }
                await _propertyService.UpdateProperty(id, property);
                return NoContent();
            }
            catch (ArgumentException ex)
            {
                return BadRequest(ex.Message);
            }
            catch (KeyNotFoundException ex)
            {
                return NotFound(ex.Message);
            }
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> DeleteProperty(int id)
        {
            try
            {
                await _propertyService.DeleteProperty(id);
                return NoContent();
            }
            catch (KeyNotFoundException ex)
            {
                return NotFound(ex.Message);
            }
```

```
        }

        [HttpPut("book/{id}")]
        public async Task<IActionResult> BookProperty(int id, [FromQuery] int
userId, [FromQuery] bool isAdmin = false)
        {
            try
            {
                await _propertyService.BookProperty(id, userId, isAdmin);
                return NoContent();
            }
            catch (ArgumentException ex)
            {
                return BadRequest(ex.Message);
            }
            catch (KeyNotFoundException ex)
            {
                return NotFound(ex.Message);
            }
        }
        [HttpGet("booked")]
        public async Task<ActionResult<IEnumerable<Property>>>
GetBookedProperties([FromQuery] int userId)
        {
            var properties = await
_propertyService.GetBookedProperties(userId);
            return Ok(properties);
        }

    }

}
```

Листинг В.4 – Файл PropertyRequestController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RentalPlatform.Data;
using RentalPlatform.Models;
using RentalPlatform.Services;

namespace RentalPlatform.Controllers
{
    [ApiController]
    [Route("api/requests")]
    public class PropertyRequestsController : ControllerBase
    {
        private readonly MainDbContext _database;
        private readonly PropertyService _propertyService;

        public PropertyRequestsController(MainDbContext database,
PropertyService propertyService)
        {
            _database = database;
            _propertyService = propertyService;
        }

        [HttpPost("submit")]
        public async Task<IActionResult> SubmitRequest([FromBody]
PropertyRequest request)
        {
            _database.PropertyRequests.Add(request);
            await _database.SaveChangesAsync();
```

```
            return Ok(request);
        }

        [HttpGet("pending")]
        public async Task<ActionResult<IEnumerable<PropertyRequest>>>
ViewAllRequests()
        {
            var result = await _database.PropertyRequests
                .Include(r => r.User)
                .Where(r => !r.Approved)
                .ToListAsync();
            return Ok(result);
        }

        [HttpPost("approve/{requestId}")]
        public async Task<IActionResult> ConfirmRequest(int requestId)
        {
            var req = await _database.PropertyRequests.FindAsync(requestId);
            if (req == null) return NotFound();

            var newProperty = new Property
            {
                OwnerId = req.UserId,
                Title = req.Title,
                Description = req.Description,
                Address = req.Address,
                Price = req.Price,
                Rooms = req.Rooms,
                PhotoUrl = req.PhotoUrl
            };

            await _propertyService.CreateProperty(newProperty);

            req.Approved = true;
            await _database.SaveChangesAsync();

            return Ok(newProperty);
        }

        [HttpDelete("reject/{requestId}")]
        public async Task<IActionResult> Reject(int requestId)
        {
            var item = await _database.PropertyRequests.FindAsync(requestId);
            if (item == null) return NotFound("Не удалось найти заявку");

            _database.PropertyRequests.Remove(item);
            await _database.SaveChangesAsync();
            return NoContent();
        }
    }
}
```

Листинг В.5 – Файл UserController.cs

```
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using RentalPlatform.Data;
using RentalPlatform.Models;
using RentalPlatform.Services;

namespace RentalPlatform.Controllers
{
    [ApiController]
```

```
    [Route("api/users")]
    public class AccountController : ControllerBase
    {
        private readonly UserService _userService;
        private readonly MainDbContext _database;

        public AccountController(UserService userService, MainDbContext db)
        {
            _userService = userService;
            _database = db;
        }

        [HttpPost("register")]
        public async Task<ActionResult<User>> Register(string login, string
pass, string email, string fname, string lname)
        {
            try
            {
                var result = await _userService.RegisterUser(login, pass,
email, fname, lname);
                return CreatedAtAction(nameof(GetById), new { id = result.Id
}, result);
            }
            catch (ArgumentException ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpPost("admin")]
        public async Task<ActionResult<User>> CreateAdmin(string login,
string pass, string email, string fname, string lname)
        {
            try
            {
                var admin = await _userService.CreateAdminUser(login, pass,
email, fname, lname);
                return CreatedAtAction(nameof(GetById), new { id = admin.Id
}, admin);
            }
            catch (ArgumentException ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet("{id}")]
        public async Task<ActionResult<User>> GetById(int id)
        {
            var user = await _database.Users.FindAsync(id);
            if (user == null) return NotFound();
            return Ok(user);
        }

        [HttpGet("byname")]
        public async Task<ActionResult<User>> GetByUsername(string username)
        {
            var result = await _userService.GetUserByUserName(username);
            if (result == null) return NotFound();
            return Ok(result);
        }

        [HttpPost("login")]
```

```
        public async Task<ActionResult<User>> Authenticate(string login,
string pass)
        {
            try
            {
                var result = await _userService.AuthenticateUser(login,
pass);
                if (result == null) return Unauthorized("Неверные данные");
                return Ok(result);
            }
            catch (ArgumentException ex)
            {
                return BadRequest(ex.Message);
            }
        }

        [HttpGet]
        public async Task<ActionResult<IEnumerable<User>>> AllUsers()
        {
            var list = await _database.Users.ToListAsync();
            return Ok(list);
        }

        [HttpDelete("{id}")]
        public async Task<IActionResult> Remove(int id, [FromQuery] int
requesterId)
        {
            if (!await _userService.IsAdmin(requesterId))
                return Unauthorized("Доступ только для администратора");

            try
            {
                await _userService.DeleteUser(id);
                return NoContent();
            }
            catch (KeyNotFoundException ex)
            {
                return NotFound(ex.Message);
            }
        }

        [HttpPut("{id}")]
        public async Task<IActionResult> Update(int id, [FromQuery] string
fname, [FromQuery] string lname, [FromQuery] string mail)
        {
            try
            {
                await _userService.UpdateUser(id, fname, lname, mail);
                return NoContent();
            }
            catch (KeyNotFoundException ex)
            {
                return NotFound(ex.Message);
            }
            catch (ArgumentException ex)
            {
                return BadRequest(ex.Message);
            }
        }
    }
}
```

Листинг В.6 – Файл AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using RentalPlatform.Models;

namespace RentalPlatform.Data
{
    public class MainDbContext : DbContext
    {
        public MainDbContext(DbContextOptions<MainDbContext> options) :
base(options) {}

        public DbSet<User> Users { get; set; }
        public DbSet<Property> Properties { get; set; }
        public DbSet<Comment> Comments { get; set; }
        public DbSet<Favorite> Favorites { get; set; }
        public DbSet<PropertyRequest> PropertyRequests { get; set; }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            base.OnModelCreating(builder);

            builder.Entity<Property>()
                .HasOne(p => p.Owner)
                .WithMany()
                .HasForeignKey(p => p.OwnerId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.Entity<PropertyRequest>()
                .HasOne(r => r.User)
                .WithMany()
                .HasForeignKey(r => r.UserId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.Entity<Property>()
                .HasOne(p => p.BookedByUser)
                .WithMany()
                .HasForeignKey(p => p.BookedByUserId)
                .OnDelete(DeleteBehavior.NoAction);

            builder.Entity<Comment>()
                .Property(c => c.Rating)
                .HasDefaultValue(5);

            builder.Entity<Comment>()
                .HasOne(c => c.User)
                .WithMany()
                .HasForeignKey(c => c.UserId)
                .OnDelete(DeleteBehavior.Restrict);

            builder.Entity<Comment>()
                .HasOne(c => c.Property)
                .WithMany()
                .HasForeignKey(c => c.PropertyId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.Entity<Favorite>()
                .HasOne(f => f.User)
                .WithMany()
                .HasForeignKey(f => f.UserId)
                .OnDelete(DeleteBehavior.Restrict);

            builder.Entity<Favorite>()
```

```
                .HasOne(f => f.Property)
                .WithMany()
                .HasForeignKey(f => f.PropertyId)
                .OnDelete(DeleteBehavior.Cascade);
        }
    }
}
```

Листинг В.7 – Файл PropertyDto.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Rental.DTO
{
    public class PropertyDto
    {
        [Required]
        public int OwnerId { get; set; }

        [Required, MaxLength(255)]
        public string? Title { get; set; }

        public string? Description { get; set; }

        [Required, MaxLength(255)]
        public string? Address { get; set; }

        [Required]
        [Column(TypeName = "decimal(10,2)")]
        public decimal Price { get; set; }

        [Required]
        public int Rooms { get; set; }

        [MaxLength(255)]
        public string? PhotoUrl { get; set; }

        public bool IsBooked { get; set; } = false;
    }

    public class PropertyUpdateDto
    {
        [Required]
        public int Id { get; set; }

        [Required]
        public int OwnerId { get; set; }

        [Required, MaxLength(255)]
        public string? Title { get; set; }

        public string? Description { get; set; }

        [Required, MaxLength(255)]
        public string? Address { get; set; }

        [Required]
        [Column(TypeName = "decimal(10,2)")]
        public decimal Price { get; set; }

        [Required]
        public int Rooms { get; set; }
```

```
        [MaxLength(255)]
        public string? PhotoUrl { get; set; }

        public bool IsBooked { get; set; } = false;
    }
}
```

Листинг В.8 – Файл Comment.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Rental.Models
{
    [Table("comments")]
    public class Comment
    {
        [Key]
        [Column("id")]
        public int Id { get; set; }

        [Required, Column("user_id")]
        public int UserId { get; set; }

        [Required, Column("property_id")]
        public int PropertyId { get; set; }

        [Required, Column("content")]
        public string Content { get; set; }

        [Column("created_at")]
        public DateTime CreatedAt { get; set; } = DateTime.Now;

        [Column("rating")]
        public int? Rating { get; set; }

        public User? User { get; set; }
        public Property? Property { get; set; }
    }
}
```

Листинг В.9 – Файл Favorite.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Rental.Models
{
    [Table("favorites")]
    public class Favorite
    {
        [Key]
        [Column("id")]
        public int Id { get; set; }

        [Required, Column("user_id")]
        public int UserId { get; set; }

        [Required, Column("property_id")]
        public int PropertyId { get; set; }

        public User? User { get; set; }
```

```
        public Property? Property { get; set; }
    }
}
```

Листинг В.10 – Файл Property.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Rental.Models
{
    public class Property
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }

        [Required, Column("owner_id")]
        public int OwnerId { get; set; }
        public User? Owner { get; set; }

        [Required, MaxLength(255), Column("title")]
        public string? Title { get; set; }

        [Column("description")]
        public string? Description { get; set; }

        [Required, MaxLength(255), Column("address")]
        public string? Address { get; set; }

        [Required]
        [Column(TypeName = "decimal(10,2)")]
        public decimal Price { get; set; }

        [Required, Column("rooms")]
        public int Rooms { get; set; }

        [MaxLength(255), Column("photo_url")]
        public string? PhotoUrl { get; set; }

        [Column("is_booked")]
        public bool IsBooked { get; set; } = false;

        [ForeignKey("booked_by_user_id")]
        [Column("booked_by_user_id")]
        public int? BookedByUserId { get; set; }

        public User? BookedByUser { get; set; }

        [Column("created_at")]
        public DateTime CreatedAt { get; set; } = DateTime.Now;
    }
}
```

Листинг В.11 – Файл PropertyDto.cs

```
namespace Rental.Models
{
    public class PropertyDto
    {
        public int Id { get; set; }
        public string? Title { get; set; }
        public string? Address { get; set; }
```

```
        public decimal Price { get; set; }
        public int Rooms { get; set; }
        public string? PhotoUrl { get; set; }
        public bool IsBooked { get; set; }

        public int? BookedByUserId { get; set; }
        public string? BookedByUserName { get; set; }
    }
}
```

Листинг В.12 – Файл PropertyRequest.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Rental.Models
{
    [Table("property_requests")]
    public class PropertyRequest
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public int UserId { get; set; }

        [Required]
        public string Title { get; set; }

        public string? Description { get; set; }

        [Required]
        public string Address { get; set; }

        [Required]
        public decimal Price { get; set; }

        [Required]
        public int Rooms { get; set; }

        public string? PhotoUrl { get; set; }

        public DateTime CreatedAt { get; set; } = DateTime.Now;

        public bool Approved { get; set; } = false;

        public User? User { get; set; }
    }
}
```

Листинг В.13 – Файл User.cs

```
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace Rental.Models
{
    public class User
    {
        [Key]
        [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
        public int Id { get; set; }
```

```
        [Required, MaxLength(50)]
        [Column("username")]
        public string Username { get; set; }

        [Required, MaxLength(255)]
        [Column("password_hash")]
        public string PasswordHash { get; set; }

        [Required, MaxLength(100)]
        [Column("email")]
        public string Email { get; set; }

        [MaxLength(50), Column("first_name")]
        public string? FirstName { get; set; }

        [MaxLength(50), Column("last_name")]
        public string? LastName { get; set; }

        [Required, MaxLength(20)]
        public string Role { get; set; } = "user";

        [Column("created_at")]
        public DateTime CreatedAt { get; set; } = DateTime.Now;
    }
}
```

Листинг В.14 – Файл AdminBookingStrategy.cs

```
using Microsoft.EntityFrameworkCore;
using Rental.Data;
using Rental.Models;
using System;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
{
    public class AdminBookingStrategy : IPropertyBookingStrategy
    {
        private readonly AppDbContext _context;

        public AdminBookingStrategy(AppDbContext context)
        {
            _context = context;
        }
        public async Task BookProperty(Property property, int userId)
        {
            if (property == null)
            {
                throw new KeyNotFoundException($"Property with id
{property.Id} not found.");
            }
            if (userId == 0)
            {
                throw new ArgumentException("Please provide valid user id.");
            }

            property.IsBooked = true;
            property.BookedByUserId = userId;
            property.OwnerId = userId;

            _context.Entry(property).State = EntityState.Modified;

            try
```

```
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!await PropertyExists(property.Id))
                {
                    throw new KeyNotFoundException($"Property with id
{property.Id} not found.");
                }
                else
                {
                    throw;
                }
            }
        }
        private async Task<bool> PropertyExists(int id)
        {
            return await _context.Properties.AnyAsync(e => e.Id == id);
        }
    }
}
```

Листинг В.15 – Файл ApartmentFactory.cs

```
using Rental.Models;

namespace Rental.Services.Patterns
{
    public class ApartmentFactory : IPropertyFactory
    {
        public Property CreateProperty(int ownerId, string title, string
address, decimal price, int rooms, string description, string photoUrl)
        {

            return new Property
            {
                OwnerId = ownerId,
                Title = title,
                Address = address,
                Price = price,
                Rooms = rooms,
                Description = description,
                PhotoUrl = photoUrl
            };
        }
    }
}
```

Листинг В.16 – Файл ConsolePropertyObserver.cs

```
using Rental.Models;
using System;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
{
    public class ConsolePropertyObserver : IPropertyChangeObserver
    {
        public async Task OnPropertyCreated(Property property)
        {
```

```
            Console.WriteLine($"Property created: {property.Title} (ID:
{property.Id})");
        }

        public async Task OnPropertyUpdated(Property property)
        {
            Console.WriteLine($"Property updated: {property.Title} (ID:
{property.Id})");
        }
        public async Task OnPropertyDeleted(Property property)
        {
            Console.WriteLine($"Property deleted: {property.Title} (ID:
{property.Id})");
        }
        public async Task OnPropertyBooked(Property property)
        {
            Console.WriteLine($"Property booked: {property.Title} (ID:
{property.Id})");
        }
    }
}
```

Листинг В.17 – Файл DefaultBookingStrategy.cs

```
using Microsoft.EntityFrameworkCore;
using Rental.Data;
using Rental.Models;
using System;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
{
    public class DefaultBookingStrategy : IPropertyBookingStrategy
    {
        private readonly AppDbContext _context;

        public DefaultBookingStrategy(AppDbContext context)
        {
            _context = context;
        }
        public async Task BookProperty(Property property, int userId)
        {
            if (property == null)
            {
                throw new KeyNotFoundException($"Property with id
{property.Id} not found.");
            }
            if (userId == 0)
            {
                throw new ArgumentException("Please provide valid user id.");
            }

            property.IsBooked = true;
            property.BookedByUserId = userId;

            _context.Entry(property).State = EntityState.Modified;
            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!await PropertyExists(property.Id))
```

```
                {
                    throw new KeyNotFoundException($"Property with id
{property.Id} not found.");
                }
                else
                {
                    throw;
                }
            }
        }
        private async Task<bool> PropertyExists(int id)
        {
            return await _context.Properties.AnyAsync(e => e.Id == id);
        }
    }
}
```

Листинг В.18 – Файл HouseFactory.cs

```
using Rental.Models;

namespace Rental.Services.Patterns
{
    public class HouseFactory : IPropertyFactory
    {
        public Property CreateProperty(int ownerId, string title, string
address, decimal price, int rooms, string description, string photoUrl)
        {
            return new Property
            {
                OwnerId = ownerId,
                Title = title,
                Address = address,
                Price = price,
                Rooms = rooms,
                Description = description,
                PhotoUrl = photoUrl
            };
        }
    }

}
```

Листинг В.19 – Файл IPropertyBookingStrategy.cs

```
using Rental.Models;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
{
    public interface IPropertyBookingStrategy
    {
        Task BookProperty(Property property, int userId);
    }
}
```

Листинг В.20 – Файл IPropertyChangeNotifier.cs

```
using Rental.Models;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
```

```
{
    public interface IPropertyChangeNotifier
    {
        void AddObserver(IPropertyChangeObserver observer);
        void RemoveObserver(IPropertyChangeObserver observer);
        Task NotifyPropertyCreated(Property property);
        Task NotifyPropertyUpdated(Property property);
        Task NotifyPropertyDeleted(Property property);
        Task NotifyPropertyBooked(Property property);
    }
}
```

Листинг В.21 – Файл IPropertyChangeObserver.cs

```
using Rental.Models;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
{
    public interface IPropertyChangeObserver
    {
        Task OnPropertyCreated(Property property);
        Task OnPropertyUpdated(Property property);
        Task OnPropertyDeleted(Property property);
        Task OnPropertyBooked(Property property);
    }
}
```

Листинг В.22 – Файл IPropertyFactory.cs

```
using Rental.Models;

namespace Rental.Services.Patterns
{
    public interface IPropertyFactory
    {
        Property CreateProperty(int ownerId, string title, string address,
decimal price, int rooms, string description, string photoUrl);
    }
}
```

Листинг В.23 – Файл PropertyChangeNotifier.cs

```
using Rental.Models;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace Rental.Services.Patterns
{
    public class PropertyChangeNotifier : IPropertyChangeNotifier
    {
        private readonly List<IPropertyChangeObserver> _observers = new
List<IPropertyChangeObserver>();

        public void AddObserver(IPropertyChangeObserver observer)
        {
            _observers.Add(observer);
        }
        public void RemoveObserver(IPropertyChangeObserver observer)
        {
            _observers.Remove(observer);
        }
        public async Task NotifyPropertyCreated(Property property)
```

```
        {
            foreach (var observer in _observers)
            {
                await observer.OnPropertyCreated(property);
            }
        }
        public async Task NotifyPropertyUpdated(Property property)
        {
            foreach (var observer in _observers)
            {
                await observer.OnPropertyUpdated(property);
            }
        }
        public async Task NotifyPropertyDeleted(Property property)
        {
            foreach (var observer in _observers)
            {
                await observer.OnPropertyDeleted(property);
            }
        }
        public async Task NotifyPropertyBooked(Property property)
        {
            foreach (var observer in _observers)
            {
                await observer.OnPropertyBooked(property);
            }
        }
    }
}
```

Листинг В.24 – Файл PropertyService.cs

```
using Microsoft.EntityFrameworkCore;
using Rental.Data;
using Rental.Models;
using Rental.Services.Patterns;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Rental.Services
{
    public class PropertyService
    {
        private readonly AppDbContext _context;
        private readonly IPropertyFactory _houseCreator;
        private readonly IPropertyChangeNotifier _notifier;

        public PropertyService(AppDbContext context, IPropertyChangeNotifier
notifier)
        {
            _context = context;
            _houseCreator = new HouseFactory();
            _notifier = notifier;
            _notifier.AddObserver(new ConsolePropertyObserver());
        }

        public async Task<IEnumerable<Property>> RetrieveAllAsync()
        {
            return await _context.Properties.ToListAsync();
        }
```

```
        public async Task<IEnumerable<Property>> GetFreeProperties(int?
userId = null)
        {
            return await _context.Properties
                .Where(p => !p.IsBooked)
                .ToListAsync();
        }

        public async Task CancelBooking(int propertyId)
        {
            var item = await _context.Properties.FindAsync(propertyId);
            if (item == null)
                throw new KeyNotFoundException($"Объект с ID {propertyId} не
найден.");

            item.IsBooked = false;
            item.BookedByUserId = null;

            await _context.SaveChangesAsync();
            await _notifier.NotifyPropertyUpdated(item);
        }

        public async Task<IEnumerable<Property>> FetchAll(int? userId)
        {
            return await _context.Properties.ToListAsync();
        }

        public async Task<Property?> FindById(int id)
        {
            return await _context.Properties.FindAsync(id);
        }

        public async Task<Property> AddNew(Property property)
        {
            IPropertyFactory apartmentFactory = new ApartmentFactory();
            Property created;

            if (property.Rooms >= 3)
            {
                created = _houseCreator.CreateProperty(
                    property.OwnerId, property.Title, property.Address,
                    property.Price, property.Rooms, property.Description,
property.PhotoUrl);
            }
            else
            {
                created = apartmentFactory.CreateProperty(
                    property.OwnerId, property.Title, property.Address,
                    property.Price, property.Rooms, property.Description,
property.PhotoUrl);
            }

            _context.Properties.Add(created);
            await _context.SaveChangesAsync();
            await _notifier.NotifyPropertyCreated(created);

            return created;
        }

        public async Task Modify(int id, Property updated)
        {
            if (id != updated.Id)
```

```
                throw new ArgumentException("ID объекта не соответствует
запрашиваемому.");

            _context.Entry(updated).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
                await _notifier.NotifyPropertyUpdated(updated);
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!await Exists(id))
                    throw new KeyNotFoundException($"Объект с ID {id}
отсутствует.");
                else
                    throw;
            }
        }

        public async Task Remove(int id)
        {
            var property = await _context.Properties.FindAsync(id);
            if (property == null)
                throw new KeyNotFoundException($"Объект с ID {id} не
найден.");

            _context.Properties.Remove(property);
            await _context.SaveChangesAsync();
            await _notifier.NotifyPropertyDeleted(property);
        }

        public IQueryable<Property> QueryProperties()
        {
            return _context.Properties.AsQueryable();
        }

        public async Task Reserve(int propertyId, int userId, bool admin =
false)
        {
            var prop = await _context.Properties.FindAsync(propertyId);
            IPropertyBookingStrategy strategy = admin
                ? new AdminBookingStrategy(_context)
                : new DefaultBookingStrategy(_context);

            await strategy.BookProperty(prop, userId);
            await _notifier.NotifyPropertyBooked(prop);
        }

        public async Task<List<Property>> GetUserBookings(int userId)
        {
            return await _context.Properties
                .Where(p => p.IsBooked && p.BookedByUserId == userId)
                .ToListAsync();
        }

        private async Task<bool> Exists(int id)
        {
            return await _context.Properties.AnyAsync(p => p.Id == id);
        }
    }
}
```

Листинг В.25 – Файл UserService.cs

```
using Microsoft.EntityFrameworkCore;
using Rental.Data;
using Rental.Models;
using System;
using System.Threading.Tasks;

namespace Rental.Services
{
    public class UserService
    {
        private readonly AppDbContext _context;

        public UserService(AppDbContext context)
        {
            _context = context;
        }

        public async Task<User?> RegisterAsync(string username, string
password, string email, string firstName, string lastName)
        {
            if (string.IsNullOrWhiteSpace(username) ||
string.IsNullOrWhiteSpace(password) || string.IsNullOrWhiteSpace(email))
                throw new ArgumentException("Обязательные поля не могут быть
пустыми.");

            var existing = await _context.Users.FirstOrDefaultAsync(u =>
u.Username == username || u.Email == email);
            if (existing != null)
                throw new ArgumentException("Пользователь с таким логином или
email уже зарегистрирован.");

            var user = new User
            {
                Username = username,
                PasswordHash = password,
                Email = email,
                FirstName = firstName,
                LastName = lastName,
                CreatedAt = DateTime.Now,
                Role = "user"
            };

            _context.Users.Add(user);
            await _context.SaveChangesAsync();

            return user;
        }

        public async Task<User?> CreateAdminAsync(string username, string
password, string email, string firstName, string lastName)
        {
            if (string.IsNullOrWhiteSpace(username) ||
string.IsNullOrWhiteSpace(password) || string.IsNullOrWhiteSpace(email))
                throw new ArgumentException("Недопустимые значения для
регистрации администратора.");

            var duplicate = await _context.Users.FirstOrDefaultAsync(u =>
u.Username == username || u.Email == email);
            if (duplicate != null)
                throw new ArgumentException("Администратор с такими данными
уже существует.");
```

```
            var admin = new User
            {
                Username = username,
                PasswordHash = password,
                Email = email,
                FirstName = firstName,
                LastName = lastName,
                CreatedAt = DateTime.Now,
                Role = "admin"
            };

            _context.Users.Add(admin);
            await _context.SaveChangesAsync();

            return admin;
        }

        public async Task<User?> Authenticate(string username, string
password)
        {
            if (string.IsNullOrEmpty(username) ||
string.IsNullOrEmpty(password))
                throw new ArgumentException("Необходимо ввести имя
пользователя и пароль.");

            var user = await _context.Users.FirstOrDefaultAsync(u =>
u.Username == username);
            if (user == null || user.PasswordHash != password)
                return null;

            return user;
        }

        public async Task<User?> GetById(int id)
        {
            return await _context.Users.FindAsync(id);
        }

        public async Task<User?> GetByUsername(string username)
        {
            return await _context.Users.FirstOrDefaultAsync(u => u.Username
== username);
        }

        public async Task<bool> IsAdministrator(int id)
        {
            var user = await GetById(id);
            return user?.Role == "admin";
        }

        public async Task RemoveUser(int id)
        {
            var user = await _context.Users.FindAsync(id);
            if (user == null)
                throw new KeyNotFoundException($"Пользователь с ID {id} не
найден.");

            _context.Users.Remove(user);
            await _context.SaveChangesAsync();
        }
```

```
        public async Task ModifyUser(int id, string firstName, string
lastName, string email)
        {
            var user = await _context.Users.FindAsync(id);
            if (user == null)
                throw new KeyNotFoundException($"Пользователь с ID {id}
отсутствует.");

            user.FirstName = firstName;
            user.LastName = lastName;
            user.Email = email;

            _context.Entry(user).State = EntityState.Modified;

            try
            {
                await _context.SaveChangesAsync();
            }
            catch (DbUpdateConcurrencyException)
            {
                if (!await Exists(id))
                    throw new KeyNotFoundException($"Пользователь с ID {id}
не существует.");
                else
                    throw;
            }
        }

        private async Task<bool> Exists(int id)
        {
            return await _context.Users.AnyAsync(u => u.Id == id);
        }
    }
}
```

Листинг В.26 – Файл Program.cs

```
using Microsoft.EntityFrameworkCore;
using Rental.Data;
using Rental.Services;
using Rental.Services.Patterns;

var builder = WebApplication.CreateBuilder(args);
// Настройка контекста базы данных с подключением
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlServer(
        builder.Configuration.GetConnectionString("DefaultConnection") +
";TrustServerCertificate=True"));

// Регистрация сервисов
builder.Services.AddScoped<PropertyService>();
builder.Services.AddScoped<UserService>();
builder.Services.AddSingleton<IPropertyChangeNotifier,
PropertyChangeNotifier>();
// Подключение контроллеров и Swagger
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen();
// Конфигурация CORS-политики
builder.Services.AddCors(options =>
{
    options.AddPolicy("AllowAll", policy =>
    {
```

```
        policy.AllowAnyOrigin()
              .AllowAnyMethod()
              .AllowAnyHeader();
    });
});
var app = builder.Build();
app.UseCors("AllowAll");
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.MapGet("/", () => "Сервис аренды недвижимости запущен.");
app.MapGet("/users", async (AppDbContext context) =>
{
    var allUsers = await context.Users.ToListAsync();
    return Results.Ok(allUsers);
});
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();

app.Run();
```

Приложение Г

(обязательное)

Листинг клиентской части

Листинг Г.1 – Файл index.html

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>Сервис аренды жилья</title>
  <link
href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;500;600;700&d
isplay=swap" rel="stylesheet" />
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-
awesome/6.0.0-beta3/css/all.min.css" />
  <link rel="stylesheet" href="style.css" />
</head>
<body>

  <header class="site-header fade-in">
    <img src="images/icon.png" alt="Лого" class="site-logo" />
    <h1 class="brand-title">RentDomik</h1>
    <button id="switchTheme" class="theme-toggle">
      <i class="fas fa-adjust"></i>
    </button>
  </header>

  <div class="popup" id="logoutConfirm">
    <div class="popup-body">
      <h3>Выход</h3>
      <p>Подтвердите завершение сессии</p>
      <button class="btn-cancel" onclick="closeModal()">Нет</button>
    </div>
  </div>

  <section id="authSection">
    <h1><i class="fas fa-sign-in-alt"></i> Авторизация</h1>
    <form id="authForm">
      <label for="userLogin">Логин:</label>
      <input type="text" id="userLogin" required />
      <label for="userPassword">Пароль:</label>
      <input type="password" id="userPassword" required />
      <button type="submit" class="btn-main"><i class="fas fa-sign-in-
alt"></i> Войти</button>
    </form>
    <button id="toggleRegister" class="btn-alt">
      <i class="fas fa-user-plus"></i> Создать аккаунт
    </button>
  </section>

  <section id="registrationSection" style="display: none;">
    <h1><i class="fas fa-user-plus"></i> Регистрация</h1>
    <form id="regForm">
      <label for="newLogin">Имя пользователя:</label>
      <input type="text" id="newLogin" required />
      <label for="newPass">Пароль:</label>
      <input type="password" id="newPass" required />
      <label for="email">Эл. почта:</label>
      <input type="email" id="email" required />
      <label for="firstName">Имя:</label>
      <input type="text" id="firstName" required />
      <label for="lastName">Фамилия:</label>
      <input type="text" id="lastName" required />
```

```
      <button type="submit" class="btn-main"><i class="fas fa-user-plus"></i>
Зарегистрироваться</button>
    </form>
    <button id="returnLogin" class="btn-alt">
      <i class="fas fa-arrow-left"></i> Назад
    </button>
  </section>

  <main id="mainApp" style="display: none;">
    <h1><i class="fas fa-home"></i> Каталог недвижимости</h1>

    <section id="adminTools" style="display: none;">
      <div class="admin-controls">
        <button id="btnAdd" class="btn-main"><i class="fas fa-plus-
circle"></i> Новое объявление</button>
        <button id="btnEdit" class="btn-alt"><i class="fas fa-edit"></i>
Изменить</button>
        <button id="btnShowAll" class="btn-main"><i class="fas fa-list"></i>
Все объекты</button>
        <button id="btnCancelAdmin" class="btn-cancel" style="display: none;"
onclick="clearProperties()">Скрыть</button>
        <button id="btnUsers" class="btn-alt"><i class="fas fa-users"></i>
Пользователи</button>
        <button id="btnProfileAdmin" class="btn-alt"><i class="fas fa-
user"></i> Профиль</button>
        <button onclick="createAdminUser()" class="btn-main"><i class="fas
fa-cogs"></i> Админ</button>
        <button onclick="loadPropertyRequests()" class="btn-main"><i
class="fas fa-envelope-open-text"></i> Заявки</button>
        <button onclick="logoutUser()" class="btn-cancel"><i class="fas fa-
sign-out-alt"></i> Выход</button>
      </div>
    </section>

    <form id="formAddProperty" style="display: none;">
      <!-- Поля добавления -->
      <label for="ownerId">ID владельца:</label>
      <input type="number" id="ownerId" required />
      <label for="title">Название:</label>
      <input type="text" id="title" required />
      <label for="location">Адрес:</label>
      <input type="text" id="location" required />
      <label for="cost">Стоимость:</label>
      <input type="number" id="cost" required />
      <label for="rooms">Комнаты:</label>
      <input type="number" id="rooms" required />
      <label for="desc">Описание:</label>
      <input type="text" id="desc" />
      <label for="photo">Ссылка на фото:</label>
      <input type="text" id="photo" />
      <button type="submit" class="btn-main"><i class="fas fa-plus-
circle"></i> Сохранить</button>
      <button type="button" class="btn-cancel"
onclick="hideAddPropertyForm()"><i class="fas fa-times"></i> Отмена</button>
    </form>

    <form id="formEditProperty" style="display: none;">
      <!-- Поля редактирования -->
      <label for="editId">ID:</label>
      <input type="number" id="editId" required />
      <label for="editOwnerId">ID владельца:</label>
      <input type="number" id="editOwnerId" required />
      <label for="editTitle">Название:</label>
```

```
      <input type="text" id="editTitle" required />
      <label for="editLocation">Адрес:</label>
      <input type="text" id="editLocation" required />
      <label for="editCost">Стоимость:</label>
      <input type="number" id="editCost" required />
      <label for="editRooms">Комнаты:</label>
      <input type="number" id="editRooms" required />
      <label for="editDesc">Описание:</label>
      <input type="text" id="editDesc" />
      <label for="editPhoto">Ссылка на фото:</label>
      <input type="text" id="editPhoto" />
      <button type="submit" class="btn-main"><i class="fas fa-edit"></i>
Обновить</button>
      <button type="button" class="btn-cancel"
onclick="hideUpdatePropertyForm()"><i class="fas fa-times"></i>
Отмена</button>
    </form>

    <section id="userOptions" class="user-tools">
      <button id="btnViewListings" class="btn-main"><i class="fas fa-
list"></i> Просмотр объектов</button>
      <button id="btnCancelUser" class="btn-cancel" style="display: none;"><i
class="fas fa-times"></i> Скрыть</button>
      <button id="btnFavs" class="btn-main"><i class="fas fa-heart"></i>
Избранное</button>
      <button id="btnBooked" class="btn-main"><i class="fas fa-book"></i>
Бронирования</button>
      <button id="btnProfileUser" class="btn-alt"><i class="fas fa-user"></i>
Профиль</button>
      <button onclick="showPropertyRequestForm()" class="btn-main"><i
class="fas fa-paper-plane"></i> Заявка</button>
      <button onclick="logoutUser()" class="btn-cancel"><i class="fas fa-
sign-out-alt"></i> Выйти</button>
    </section>

    <form id="formPropertyRequest" style="display: none;">
      <h3>Новая заявка</h3>
      <label for="reqTitle">Название:</label>
      <input type="text" id="reqTitle" required />
      <label for="reqAddress">Адрес:</label>
      <input type="text" id="reqAddress" required />
      <label for="reqCost">Стоимость:</label>
      <input type="number" id="reqCost" required />
      <label for="reqRooms">Комнаты:</label>
      <input type="number" id="reqRooms" required />
      <label for="reqDesc">Описание:</label>
      <input type="text" id="reqDesc" />
      <label for="reqPhoto">Фото (URL):</label>
      <input type="text" id="reqPhoto" />
      <button type="submit" class="btn-main">Отправить</button>
      <button type="button" class="btn-cancel"
onclick="hidePropertyRequestForm()">Отмена</button>
    </form>

    <button id="btnToggleFilters">Фильтры</button>

    <div id="filtersPanel" style="display: none;">
      <label>Цена от <input type="number" id="filterMin" /></label>
      <label>до <input type="number" id="filterMax" /></label>
      <label>Комнат: <input type="number" id="filterRooms" /></label>
      <input type="text" id="filterSearch" placeholder="Поиск..." />
      <button id="btnResetFilters" onclick="resetFilters()">Сбросить</button>
```

```
      <div id="loadingBox" style="display: none;"><i class="fas fa-spinner
fa-spin"></i> Поиск...</div>
      <p id="totalCount"></p>
    </div>

    <div id="listingsContainer"></div>

    <section id="userDirectory" style="display: none;">
      <ul id="userList"></ul>
      <button class="btn-cancel" onclick="hideUsers()"><i class="fas fa-
times"></i> Скрыть</button>
    </section>

    <section id="requestsBoard" style="display: none;">
      <h2>Заявки пользователей</h2>
      <ul id="requestsList"></ul>
      <button class="btn-cancel" onclick="hideRequests()">Скрыть</button>
    </section>

    <section id="profileSection" style="display: none;">
      <h2><i class="fas fa-user"></i> Профиль</h2>
      <form id="profileForm">
        <label for="profName">Имя:</label>
        <input type="text" id="profName" required />
        <label for="profSurname">Фамилия:</label>
        <input type="text" id="profSurname" required />
        <label for="profEmail">Email:</label>
        <input type="email" id="profEmail" required />
        <button type="submit" class="btn-main"><i class="fas fa-user-
edit"></i> Сохранить</button>
        <button type="button" class="btn-cancel"
onclick="hideUserProfile()"><i class="fas fa-times"></i> Отмена</button>
      </form>
    </section>
  </main>

  <script src="script.js"></script>
</body>
</html>
```

Листинг Г.2 – Файл style.css

```
:root {
  --primary: #b9976b;
  --primary-dark: #8b5e3c;
  --accent-orange: #d9a066;
  --accent-orange-light: #f3d5b5;
  --danger: #c56c60;
  --danger-dark: #8b3a2c;
  --bg-light: #fdf6f0;
  --text-main: #4b3832;
  --text-accent: #8b5e3c;
  --shadow: rgba(0, 0, 0, 0.1);
}

body {
  font-family: 'Poppins', sans-serif;
  margin: 0;
  padding: 80px 0 0;
  background: linear-gradient(135deg, #fdf6f0, #f3e5dc);
  color: var(--text-main);
  overflow-x: hidden;
  line-height: 1.6;
```

```
  transition: background 0.4s ease, color 0.4s ease;
}

h1, h2 {
  text-align: center;
  color: var(--text-accent);
  margin: 2rem 0 1rem;
  font-weight: 700;
}

#loginContainer, #registerContainer, #appContainer {
  display: flex;
  flex-direction: column;
  align-items: center;
  min-height: 100vh;
  padding: 20px;
}

form {
  background: linear-gradient(145deg, #ffffff, var(--bg-light));
  backdrop-filter: blur(10px);
  padding: 30px;
  border-radius: 25px;
  box-shadow: 0 10px 30px var(--shadow);
  max-width: 500px;
  width: 100%;
  display: flex;
  flex-direction: column;
  gap: 20px;
}

label {
  font-weight: 600;
  color: var(--text-accent);
}

input {
  padding: 12px 16px;
  border: 1px solid #cceff4;
  border-radius: 12px;
  font-size: 16px;
  transition: all 0.3s ease;
}

input:focus {
  border-color: var(--primary);
  box-shadow: 0 0 10px rgba(67, 206, 162, 0.5);
  outline: none;
}

button {
  background: linear-gradient(135deg, var(--primary), var(--primary-dark));
  color: white;
  font-weight: 600;
  border: none;
  padding: 12px 20px;
  border-radius: 12px;
  cursor: pointer;
  transition: all 0.3s ease;
  box-shadow: 0 4px 10px var(--shadow);
}

button:hover {
```

```
  transform: translateY(-2px);
  box-shadow: 0 8px 20px var(--shadow);
  filter: brightness(1.05);
}

button:disabled {
  background: #ccc;
  color: #777;
  cursor: not-allowed;
  box-shadow: none;
}

.primary-btn {
  background: linear-gradient(135deg, var(--primary), var(--primary-dark));
}

.secondary-btn {
  background: linear-gradient(135deg, var(--accent-orange-light), var(--
accent-orange));
}

.cancel-button {
  background: linear-gradient(135deg, var(--danger), var(--danger-dark));
}

.primary-btn:hover, .secondary-btn:hover {
  transform: scale(1.05);
  filter: brightness(1.1);
}

#propertyList {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 25px;
  max-width: 1200px;
  margin: 0 auto;
  padding: 0 20px;
}

.property-card .card-body {
  flex-grow: 1;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
}

.property-card h3 {
  color: var(--text-accent);
}

.property-card button {
  margin-top: 10px;
  width: 100%;
  padding: 12px;
  font-size: 14px;
  border-radius: 10px;
}

.admin-buttons {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
  gap: 20px;
  justify-content: center;
```

```
  margin: 30px 0;
  padding: 0 20px;
  width: 100%;
  max-width: 1000px;
}

#mainHeader {
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  height: 70px;
  background: linear-gradient(90deg, var(--primary), var(--primary-dark));
  display: flex;
  align-items: center;
  padding: 0 30px;
  z-index: 1000;
  box-shadow: 0 4px 10px var(--shadow);
}

.logo {
  height: 50px;
  margin-right: 20px;
}

.app-title {
  font-size: 26px;
  font-weight: 700;
  color: white;
  letter-spacing: 1px;
}

body.dark-theme {
  background: linear-gradient(135deg, #3e2c23, #2d1b12);
  color: #e6d3c5;
}

body.dark-theme input {
  background-color: #5a463a;
  color: #f1e6dd;
  border: 1px solid #7d5f4d;
}

body.dark-theme .property-card {
  background: #4a372c;
  color: #f1e6dd;
}

body.dark-theme h1, body.dark-theme h2 {
  color: #d1b6a1;
}

.modal {
  position: fixed;
  top: 0; left: 0; right: 0; bottom: 0;
  background: rgba(0,0,0,0.6);
  display: flex;
  justify-content: center;
  align-items: center;
  opacity: 0;
  pointer-events: none;
  transition: opacity 0.3s ease;
  z-index: 999;
```

```
}

.modal.show {
  opacity: 1;
  pointer-events: auto;
}

.modal-content {
  background: #fff;
  border-radius: 16px;
  padding: 30px;
  max-width: 400px;
animation: scaleIn 0.3s ease forwards;
}

@keyframes scaleIn {
  from {
    transform: scale(0.9);
    opacity: 0;
  }
  to {
    transform: scale(1);
    opacity: 1;
  }
}

.stagger-fade {
  opacity: 0;
  transform: translateY(10px);
  animation: staggerFade 0.6s ease forwards;
}

@keyframes staggerFade {
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@media (max-width: 768px) {
  h1 {
    font-size: 24px;
  }

  form {
    padding: 20px;
  }

  .property-card button {
    font-size: 14px;
    padding: 12px;
  }
}
.property-img {
  width: 100%;
  height: 200px;
  object-fit: cover;
  border-radius: 16px;
  margin-bottom: 15px;
}
.comments {
  background: var(--bg-light);
  padding: 15px;
```

```
  border-radius: 14px;
  margin-top: 20px;
  box-shadow: 0 4px 10px var(--shadow);
}

.comments h4 {
  margin-bottom: 10px;
  color: var(--text-accent);
}

.comments p {
  margin: 5px 0;
  padding: 8px 12px;
  background: #ffffff;
  border-radius: 10px;
  font-size: 14px;
  color: var(--text-main);
  box-shadow: 0 2px 5px rgba(0,0,0,0.05);
}

.comments textarea {
  width: 100%;
  border-radius: 10px;
  padding: 10px;
  border: 1px solid #cceff4;
  resize: vertical;
  margin-top: 10px;
}

.comments button {
  margin-top: 10px;
}

.theme-toggle-btn {
  position: fixed;
  top: 10px;
  right: 30px;
  z-index: 2000;
  background: linear-gradient(135deg, var(--primary), var(--primary-dark));
  color: #fff;
  padding: 12px;
  border-radius: 50%;
  font-size: 24px;
  cursor: pointer;
  box-shadow: 0 4px 10px rgba(0,0,0,0.2);
  transition: transform 0.3s ease, background 0.3s ease;
}

.theme-toggle-btn:hover {
  transform: scale(1.1);
}

#userPanel {
  display: flex;
  justify-content: center;
  flex-wrap: wrap;
  gap: 15px;
  padding: 20px;
  background: linear-gradient(90deg, #f3e5dc, #e0cfc2);
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.1);
  border-radius: 20px;
  margin: 20px auto;
  width: fit-content;
```

```
}

.user-buttons {
  display: flex;
  flex-wrap: wrap;
  justify-content: center;
  gap: 15px;
  background: linear-gradient(90deg, #e7d8c2, #d6c0aa);
  padding: 20px;
  border-radius: 20px;
  box-shadow: 0 4px 10px var(--shadow);
  width: fit-content;
  max-width: 90%;
}


#userPanel.show .user-buttons {
  opacity: 1;
  transform: translateY(0);
}

.admin-buttons-panel > button {
  flex: 0 1 auto;
}

.admin-buttons-panel {
  background: var(--bg-light);
}

#userList #users-container {
  background: var(--bg-light);
  padding: 20px;
  border-radius: 16px;
  box-shadow: 0 4px 10px rgba(0,0,0,0.1);
  max-width: 600px;
  margin: 30px auto;
  display: flex;
  flex-direction: column;
  gap: 12px;
}

#users {
  list-style: disc inside;
  padding: 0;
  margin: 0;
}

#users-container > .cancel-button {
  display: inline-block;
  margin: 0 auto;
  padding: 10px 20px;
}

#userProfileContainer {
  display: flex;
  flex-direction: column;
  align-items: center;
  margin: 40px auto;
  width: 100%;
  max-width: 500px;
}

#userProfileContainer h2 {
```

```
  margin: 0 0 20px;
  text-align: center;
}

#userProfileContainer form {
  width: 100%;
  max-width: 360px;
  background: var(--bg-light);
  padding: 20px;
  border-radius: 16px;
  box-shadow: 0 4px 10px rgba(0,0,0,0.1);
}

#resetFiltersBtn {
  all: unset;
  background-color: var(--accent-orange-light);
  color: var(--text-accent);
  border: 2px solid var(--accent-orange);
  border-radius: 10px;
  padding: 10px 20px;
  font-size: 15px;
  font-weight: 600;
  cursor: pointer;
  display: inline-block;
  box-shadow: 0 2px 5px var(--shadow);
  transition: background-color 0.3s ease, box-shadow 0.3s ease;
}

#resetFiltersBtn:hover {
  background-color: var(--accent-orange);
  color: white;
}

#resetFiltersBtn.highlight-reset {
  animation: pulseReset 1s infinite;
  box-shadow: 0 0 10px 3px rgba(217, 160, 102, 0.6); /* из accent-orange */
}

@keyframes pulseReset {
  0%   { box-shadow: 0 0 0 0 rgba(217, 160, 102, 0.6); }
  70%  { box-shadow: 0 0 0 10px rgba(217, 160, 102, 0); }
  100% { box-shadow: 0 0 0 0 rgba(217, 160, 102, 0); }
}

.property-card {
  display: flex;
  flex-direction: column;
  background: linear-gradient(145deg, #ffffff, var(--bg-light));
  border-radius: 22px;
  padding: 16px 18px;
  box-shadow: 0 8px 20px var(--shadow);
  transition: all 0.3s ease;
  height: 100%;
}

.property-card .card-inner {
  flex: 1;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
  align-self: stretch;
}
```

```
.property-img {
  width: 100%;
  height: 200px;
  object-fit: cover;
  border-radius: 16px;
}
#propertyRequestForm {
  max-width: 700px;
  width: 100%;
  margin: 30px auto;
  background: var(--bg-light);
  padding: 30px;
  border-radius: 24px;
  box-shadow: 0 10px 30px var(--shadow);
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 20px;
}

#propertyRequestForm h3 {
  grid-column: span 2;
  text-align: center;
  font-size: 22px;
  margin-bottom: 10px;
}

#propertyRequestForm label {
  font-weight: 600;
  color: var(--text-accent);
  margin-top: 10px;
}

#propertyRequestForm input {
  width: 100%;
}

#propertyRequestForm button {
  grid-column: span 1;
}

#propertyRequestForm button.primary-btn {
  justify-self: end;
}

#propertyRequestForm button.cancel-button {
  justify-self: start;
}
```

Листинг Г.3 – script.js

```
const apiUrl = 'https://localhost:7167/api/property';
const usersApiUrl = 'https://localhost:7167/api/user';
const favoritesApiUrl = 'https://localhost:7167/api/favorite';
const commentsApiUrl = 'https://localhost:7167/api/comment';

let currentUserId = null;
let currentUserName = null;
let isAdmin = false;
let isLoadingProperties = false;


document.getElementById('userPanel').classList.add('show');
```

```
async function loadProperties() {
    if (isLoadingProperties) return;
    isLoadingProperties = true;

    const propertyListDiv = document.getElementById('propertyList');
    document.getElementById('loadingIndicator').style.display = 'block';
    const propertyCountEl = document.getElementById('propertyCount');
    propertyListDiv.innerHTML = '';
    propertyCountEl.textContent = '';

    try {
        let url = isAdmin
            ? `${apiUrl}/available?userId=${currentUserId}`
            : `${apiUrl}/available?userId=${currentUserId}`;

            const min = document.getElementById("minPriceFilter")?.value;
            const max = document.getElementById("maxPriceFilter")?.value;
            const rooms = document.getElementById("roomsFilter")?.value;
            const keyword = document.getElementById("keywordFilter")?.value;

            if (min) url += `&minPrice=${min}`;
            if (max) url += `&maxPrice=${max}`;
            if (rooms) url += `&rooms=${rooms}`;
            if (keyword) url += `&keyword=${encodeURIComponent(keyword)}`;

        const resp = await fetch(url);
        if (!resp.ok) throw new Error(await resp.text());
        const properties = await resp.json();
        propertyCountEl.textContent = `Найдено: ${properties.length}`;
        if (properties.length === 0) {
            propertyListDiv.innerHTML = `<p style="font-weight: 500; color:
#777;">Объявлений не найдено по заданным критериям.</p>`;

document.getElementById('resetFiltersBtn')?.classList.add('highlight-reset');
            return;
        }
        for (let idx = 0; idx < properties.length; idx++) {
            const p = properties[idx];
            const comments = await loadComments(p.id);

            const ratings = comments.map(c => c.rating).filter(r => r !==
null);
            const averageRating = ratings.length > 0
              ? (ratings.reduce((sum, r) => sum + r, 0) /
ratings.length).toFixed(1)
              : null;

            const ratingHtml = averageRating
              ? `<p style="color: #d48b46; font-weight: 600;">Средняя оценка:
${averageRating} </p>`
              : '';


            const imgSrc = p.photoUrl && p.photoUrl.trim() !== ''
                ? p.photoUrl
                : 'images/no.png';

            const bookBtn = isAdmin ? '' : `<button
onclick="bookProperty(${p.id})">Забронировать</button>`;
            const delPropBtn = isAdmin ? `<button
onclick="deleteProperty(${p.id})">Удалить объявление</button>` : '';
```

```
            const favBtn = !isAdmin ? `<button
onclick="addToFavorites(${p.id})"> В избранное</button>` : '';

            const commentsHtml = comments.map(c => {
                const delC = isAdmin
                    ? `<button class="delete-comment-btn"
onclick="deleteComment(${c.id})">Удалить</button>`
                    : '';
                const stars = c.rating !== null ? ` — <span>Оценка:
${'★'.repeat(c.rating)}</span>` : '';
                return `<p><b>Пользователь ${c.userId}:</b>
${c.content}${stars} ${delC}</p>`;
            }).join('');

            const commentForm = !isAdmin
                ? `<textarea id="commentInput${p.id}" placeholder="Ваш
комментарий..."></textarea>
                   <label for="ratingInput${p.id}">Оценка:</label>
                   <select id="ratingInput${p.id}">
                       <option value="" selected disabled>Оцените...</option>
                       <option value="1">1 </option>
                       <option value="2">2 </option>
                       <option value="3">3 </option>
                       <option value="4">4 </option>
                       <option value="5">5 </option>
                   </select>
                   <button onclick="submitComment(${p.id})">Оставить
комментарий</button>`
                : '';

            const card = document.createElement('div');
            card.classList.add('property-card');
            card.style.animationDelay = `${idx * 0.1}s`;
            const bookedInfo = isAdmin && p.isBooked
                ? `<p style="font-weight: 500; color: #b35c5c;">Забронировано
пользователем: <br>ID ${p.bookedByUserId}, ${p.bookedByUserName}</p>`
                : '';

            card.innerHTML = `
              <div class="card-inner">
                <img src="${imgSrc}" class="property-img" alt="Фото
недвижимости">
                <h3>${p.title}</h3>
                <p><i class="fas fa-map-marker-alt"></i> ${p.address}</p>
                <p><i class="fas fa-dollar-sign"></i> $${p.price}</p>
                <p><i class="fas fa-door-open"></i> Комнат: ${p.rooms}</p>
                ${bookedInfo}
                ${ratingHtml}
                <div class="card-buttons">
                    ${bookBtn} ${delPropBtn} ${favBtn}
                </div>
                <div class="comments">
                    <h4>Комментарии:</h4>
                    ${commentsHtml}
                    ${commentForm}
                </div>
              </div>
            `;

            propertyListDiv.appendChild(card);
        }

        if (isAdmin) {
```

```
            document.getElementById('cancelBtn').style.display = 'inline-
block';
        } else {
            document.getElementById('cancelButton').style.display = 'inline-
block';
        }

    } catch (err) {
        console.error(err);
        propertyListDiv.innerHTML = `<p class="error">Ошибка:
${err.message}</p>`;
    } finally {
        isLoadingProperties = false;
        document.getElementById('loadingIndicator').style.display = 'none';

    }
}

async function deleteComment(commentId) {
    if (!confirm('Точно удалить этот комментарий?')) return;
    try {
        const res = await fetch(`${commentsApiUrl}/${commentId}`, { method:
'DELETE' });
        if (!res.ok) throw new Error('Не удалось удалить комментарий');
        await loadProperties();
    } catch (err) {
        alert(err.message);
    }
}

document.getElementById('addPropertyForm').addEventListener('submit', async
(event) => {
    event.preventDefault();

    const ownerId = document.getElementById('ownerId').value;
    const title = document.getElementById('title').value;
    const address = document.getElementById('address').value;
    const price = document.getElementById('price').value;
    const rooms = document.getElementById('rooms').value;
    const description = document.getElementById('description').value;
    const photoUrl = document.getElementById('photoUrl').value;

    const newProperty = {
        ownerId: parseInt(ownerId),
        title: title,
        address: address,
        price: parseFloat(price),
        rooms: parseInt(rooms),
        description: description,
        photoUrl: photoUrl
    };

    try {
        const response = await fetch(`${apiUrl}?userId=${currentUserId}`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(newProperty)
        });

        if (!response.ok) {
            const error = await response.text();
```

```
            throw new Error(`Не удалось добавить объявление! Status:
${response.status}, Error: ${error}`);
        }
        console.log('Объявление успешно добавлено!');
          hideAddPropertyForm()
        loadProperties();
    } catch (error) {
        console.error('Не удалось добавить объявление:', error);
    }
});

document.getElementById('updatePropertyForm').addEventListener('submit',
async (event) => {
    event.preventDefault();

    const updateId = document.getElementById('updateId').value;
    const updateOwnerId = document.getElementById('updateOwnerId').value;
    const updateTitle = document.getElementById('updateTitle').value;
    const updateAddress = document.getElementById('updateAddress').value;
    const updatePrice = document.getElementById('updatePrice').value;
    const updateRooms = document.getElementById('updateRooms').value;
    const updateDescription =
document.getElementById('updateDescription').value;
    const updatePhotoUrl = document.getElementById('updatePhotoUrl').value;

    const updatedProperty = {
        ownerId: parseInt(updateOwnerId),
        title: updateTitle,
        address: updateAddress,
        price: parseFloat(updatePrice),
        rooms: parseInt(updateRooms),
        description: updateDescription,
        photoUrl: updatePhotoUrl
    };

    try {
        const response = await
fetch(`${apiUrl}/${updateId}?userId=${currentUserId}`, {
            method: 'PUT',
            headers: {
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(updatedProperty)
        });

        if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось обновить объявление! Status:
${response.status}, Error: ${error}`);
        }

        console.log('Объявление успешно обновлено!');
       hideUpdatePropertyForm()
        loadProperties();
    } catch (error) {
        console.error('Не удалось обновить объявление:', error);
    }
});


document.getElementById('registerForm').addEventListener('submit', (event) =>
{
    event.preventDefault();
```

```
    registerUser();
});


document.getElementById('loginForm').addEventListener('submit', (event) => {
    event.preventDefault();
    loginUser();
});

async function bookProperty(id) {
    try {
        const response = await
fetch(`${apiUrl}/book/${id}?userId=${currentUserId}`, {
            method: 'PUT'
        });

        if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось забронировать объявление! Status:
${response.status}, Error: ${error}`);
        }

        alert('Объявление успешно забронировано!');
        await loadProperties();
    } catch (error) {
        console.error('Не удалось забронировать объявление:', error);
    }
}

async function deleteProperty(id) {

  try {
      const response = await fetch(`${apiUrl}/${id}`, {
          method: 'DELETE'
      });
    if (!response.ok) {
          const error = await response.text();
          throw new Error(`Не удалось удалить объявление! Status:
${response.status}, Error: ${error}`);
      }
       alert('Объявление успешно удалено!');
      loadProperties();

   } catch (error) {
       console.error('Не удалось удалить объявление:', error);
    }
}

async function createAdminUser() {
    const username = prompt("Введите имя пользователя для админа:");
    if (username === null || username.trim() === "") {
        alert("Имя пользователя обязательно для создания админа");
        return;
    }

    const password = prompt("Введите пароль для админа:");
    if (password === null || password.trim() === "") {
        alert("Пароль обязателен для создания админа");
        return;
    }

    const email = prompt("Введите email для админа:");
    if (email === null || email.trim() === "") {
```

```
        alert("Email обязателен для создания админа");
        return;
    }

    const firstName = prompt("Введите имя для админа:");
    if (firstName === null || firstName.trim() === "") {
        alert("Имя обязательно для создания админа");
        return;
    }

    const lastName = prompt("Введите фамилию для админа:");
    if (lastName === null || lastName.trim() === "") {
        alert("Фамилия обязательна для создания админа");
        return;
    }

    try {
        const response = await
fetch(`${usersApiUrl}/createAdmin?username=${username}&password=${password}&e
mail=${email}&firstName=${firstName}&lastName=${lastName}`, {
            method: 'POST',
            headers: {
                'Content-Type': 'application/json'
            }
        });

        if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось создать админа! Status:
${response.status}, Error: ${error}`);
        }
        alert('Админ успешно создан!');
    } catch (error) {
        console.error("Не удалось создать админа: ", error);
    }
}


function hideAddPropertyForm() {
      document.getElementById('addPropertyForm').style.display = 'none';
}
function hideUpdatePropertyForm() {
  document.getElementById('updatePropertyForm').style.display = 'none';
}
function hideUsers(){
    document.getElementById('userList').style.display = 'none'
}
function hideUserProfile() {
    document.getElementById('userProfileContainer').style.display = 'none';
}
async function loadUsers() {
    const userList = document.getElementById('users');
    userList.innerHTML = '';

      try {
        const response = await fetch(`${usersApiUrl}`);

        if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось загрузить пользователей! Status:
${response.status}, Error: ${error}`);
          }
```

```
       const users = await response.json();
        users.forEach(user => {
             const userItem = document.createElement('li');
               userItem.innerHTML = `${user.username} - ${user.email} <button
onclick="deleteUser(${user.id})">Удалить</button>`
               userList.appendChild(userItem)
        })
    }
     catch (error) {
         console.error('Не удалось загрузить пользователей:', error);
        userList.innerHTML = 'Не удалось загрузить пользователей.';
     }

}
async function loadCurrentUser() {
  try {
         const response = await
fetch(`${usersApiUrl}/username?username=${currentUserName}`);

          if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось загрузить пользователя! Status:
${response.status}, Error: ${error}`);
          }
         const user = await response.json();
        document.getElementById('updateFirstName').value = user.firstName;
        document.getElementById('updateLastName').value = user.lastName;
        document.getElementById('updateEmail').value = user.email;


    }
     catch (error) {
        console.error('Не удалось загрузить пользователя:', error);
     }

}

async function deleteUser(id){
  try {
       const response = await
fetch(`${usersApiUrl}/${id}?userId=${currentUserId}`, {
            method: 'DELETE'
        });

     if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось удалить пользователя! Status:
${response.status}, Error: ${error}`);
        }
       alert('Пользователь успешно удален!');
        loadUsers();
    } catch (error) {
        console.error("Не удалось удалить пользователя: ", error);
    }
}
document.getElementById('showRegisterButton').addEventListener('click',
showRegister);
document.getElementById('backToLoginButton').addEventListener('click',
hideRegister);
});

function switchForm(fromId, toId) {
    const fromEl = document.getElementById(fromId);
```

```
    const toEl = document.getElementById(toId);

    if (fromEl && toEl) {
        fromEl.classList.add('fade-out');
        fromEl.classList.remove('fade-in');

        setTimeout(() => {
            fromEl.style.display = 'none';
            toEl.style.display = 'flex';
            toEl.classList.add('fade-in');
            toEl.classList.remove('fade-out');
        }, 400);
    }
}

document.addEventListener('DOMContentLoaded', () => {
    const loginBtn = document.getElementById('toLogin');
    const registerBtn = document.getElementById('toRegister');

    if (loginBtn) loginBtn.addEventListener('click', () =>
switchForm('registerContainer', 'loginContainer'));
    if (registerBtn) registerBtn.addEventListener('click', () =>
switchForm('loginContainer', 'registerContainer'));
});

    document.addEventListener("DOMContentLoaded", () => {
        const elements = document.querySelectorAll('.stagger-fade');
        elements.forEach((el, i) => {
            el.style.animationDelay = `${i * 0.15}s`;
        });
    });

  function openModal() {
    document.getElementById("exampleModal").classList.add("show");
  }

  function closeModal() {
    document.getElementById("exampleModal").classList.remove("show");
  }

  document.querySelector(".cancel-button").addEventListener("click",
closeModal);

    function toggleTheme() {
    document.body.classList.toggle("dark");
  }

const themeToggleButton = document.getElementById('themeToggle');
const body = document.body;

themeToggleButton.addEventListener('click', () => {
    body.classList.toggle('dark-theme');

    if (body.classList.contains('dark-theme')) {
        localStorage.setItem('theme', 'dark');
    } else {
        localStorage.setItem('theme', 'light');
    }
});

window.addEventListener('load', () => {
    const savedTheme = localStorage.getItem('theme');
    if (savedTheme === 'dark') {
```

```
        body.classList.add('dark-theme');
    }
});

document.getElementById('updateUserForm').addEventListener('submit', async
(event) => {
     event.preventDefault();

    const updateFirstName = document.getElementById('updateFirstName').value;
    const updateLastName = document.getElementById('updateLastName').value;
     const updateEmail = document.getElementById('updateEmail').value;
    try {
        const response = await
fetch(`${usersApiUrl}/${currentUserId}?firstName=${updateFirstName}&lastName=
${updateLastName}&email=${updateEmail}`, {
            method: 'PUT'
        });

     if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось обновить пользователя! Status:
${response.status}, Error: ${error}`);
        }
        alert('Пользователь успешно обновлен!');
        hideUserProfile();
    }
     catch (error) {
        console.error("Не удалось обновить пользователя: ", error);
     }
});

document.getElementById('showPropertiesButtonUser').addEventListener('click',
loadProperties);
document.getElementById('showFavoritesButtonUser').addEventListener('click',
loadFavorites);
document.getElementById('showBookedButtonUser').addEventListener('click',
loadBookedProperties);

document.getElementById('showUserProfileButtonUser').addEventListener('click'
, function(){
    loadCurrentUser();
    document.getElementById('userProfileContainer').style.display = 'flex';
});

fetch(`https://localhost:7167/api/comment/${propertyId}`);

    if (!response.ok) return [];
    return await response.json();
}

function renderComments(comments) {
    return comments.map(c => `<p><b>Пользователь ${c.userId}:</b>
${c.content}</p>`).join('');
}

async function submitComment(propertyId) {
    const contentInput =
document.getElementById(`commentInput${propertyId}`);
    const ratingSelect = document.getElementById(`ratingInput${propertyId}`);

    const content = contentInput?.value.trim();
    const ratingRaw = ratingSelect?.value;
    const rating = ratingRaw ? parseInt(ratingRaw) : null;
```

```
    if (!content) {
        alert("Пожалуйста, введите комментарий");
        return;
    }


        contentInput.value = '';
        if (ratingSelect) ratingSelect.selectedIndex = 0;

        await loadBookedProperties();
    } catch (err) {
        console.error(err);
        alert("Не удалось оставить комментарий: " + err.message);
    }
}

async function addToFavorites(propertyId) {
        alert("Добавлено в избранное!");
    } else {
        alert("Ошибка добавления в избранное.");
    }
}
async function loadFavorites() {
    const propertyListDiv = document.getElementById('propertyList');
    propertyListDiv.innerHTML = '';

            const error = await response.text();
            throw new Error(`Ошибка загрузки избранного: ${error}`);
        }

        const favorites = await response.json();

        for (const favorite of favorites) {
            const property = favorite.property;

            const propertyCard = document.createElement('div');
            propertyCard.classList.add('property-card');

            const imgSrc = property.photoUrl && property.photoUrl.trim() !==
''
                ? property.photoUrl
                : 'images/no.png';

            const comments = await loadComments(property.id);
            const commentsHtml = comments.map(c => `
                <p><b>Пользователь ${c.userId}:</b> ${c.content}</p>
            `).join('');

            propertyCard.innerHTML = `
                <img src="${imgSrc}" class="property-img" alt="Фото
недвижимости">
                <h3>${property.title}</h3>
                <p><strong>Адрес:</strong> ${property.address}</p>
                <p><strong>Цена:</strong> $${property.price}</p>
                <p><strong>Комнат:</strong> ${property.rooms}</p>

                <button onclick="removeFromFavorites(${property.id})"
class="unfavorite-btn">
                    Убрать из избранного
                </button>

                <div class="comments">
```

```
                    <h4>Комментарии:</h4>
                    ${commentsHtml}
                    <textarea id="commentInput${property.id}"
placeholder="Ваш комментарий..."></textarea>
                    <button onclick="submitComment(${property.id})">Оставить
комментарий</button>
                </div>
            `;

            propertyListDiv.appendChild(propertyCard);
        }

        document.getElementById('cancelButton').style.display = 'inline-
block';

    } catch (error) {
        console.error('Не удалось загрузить избранное:', error);
        propertyListDiv.innerHTML = 'Ошибка загрузки избранного.';
    }
}


async function removeFromFavorites(propertyId) {
    if (!confirm('Вы уверены, что хотите убрать из избранного?')) return;
        alert('Удалено из избранного!');
        await loadFavorites();
    } catch (error) {
        console.error('Ошибка удаления из избранного:', error);
    }
}

async function loadBookedProperties() {
    const propertyListDiv = document.getElementById('propertyList');
    propertyListDiv.innerHTML = '';

    try {
        const response = await
fetch(`${apiUrl}/booked?userId=${currentUserId}`);
        if (!response.ok) {
            const error = await response.text();
            throw new Error(`Ошибка загрузки бронированных объявлений:
${error}`);
        }

        const properties = await response.json();

        for (const property of properties) {
            const propertyCard = document.createElement('div');
            propertyCard.classList.add('property-card');

            const imgSrc = property.photoUrl && property.photoUrl.trim() !==
''
                ? property.photoUrl
                : 'images/no.png';

            const comments = await loadComments(property.id);
            const commentsHtml = comments.map(c => {
                const stars = c.rating !== null ? ` — <span>Оценка:
${'★'.repeat(c.rating)}</span>` : '';
                return `<p><b>Пользователь ${c.userId}:</b>
${c.content}${stars}</p>`;
            }).join('');
```

```
            propertyCard.innerHTML = `
                <img src="${imgSrc}" class="property-img" alt="Фото
недвижимости">
                <h3>${property.title}</h3>
                <p><strong>Адрес:</strong> ${property.address}</p>
                <p><strong>Цена:</strong> $${property.price}</p>
                <p><strong>Комнат:</strong> ${property.rooms}</p>
                <button onclick="unbookProperty(${property.id})"
class="unbook-btn">Убрать бронь</button>

                <div class="comments">
                    <h4>Комментарии:</h4>
                    ${commentsHtml}
                    <textarea id="commentInput${property.id}"
placeholder="Ваш комментарий..."></textarea>
                    <label for="ratingInput${property.id}">Оценка:</label>
                    <select id="ratingInput${property.id}">
                        <option value="" selected
disabled>Оцените...</option>
                        <option value="1">1 ★</option>
                        <option value="2">2 ★★</option>
                        <option value="3">3 ★★★</option>
                        <option value="4">4 ★★★★</option>
                        <option value="5">5 ★★★★★</option>
                    </select>
                    <button onclick="submitComment(${property.id})">Оставить
комментарий</button>
                </div>
            `;

            propertyListDiv.appendChild(propertyCard);
        }

        document.getElementById('cancelButton').style.display = 'inline-
block';

    } catch (error) {
        console.error('Не удалось загрузить бронированные объявления:',
error);
        propertyListDiv.innerHTML = 'Ошибка загрузки бронированных
объявлений.';
    }
}

async function unbookProperty(id) {
    if (!confirm('Вы уверены, что хотите убрать бронь?')) return;

    try {
        const response = await fetch(`${apiUrl}/unbook/${id}`, {
            method: 'PUT'
        });

        if (!response.ok) {
            const error = await response.text();
            throw new Error(`Не удалось убрать бронь! Status:
${response.status}, Error: ${error}`);
        }

        alert('Бронь успешно снята!');
        await loadBookedProperties();
    } catch (error) {
        console.error('Ошибка при снятии брони:', error);
    }
```

```
}

document.getElementById('cancelButton').addEventListener('click',
clearProperties);

function clearProperties() {
    const propertyListDiv = document.getElementById('propertyList');
    propertyListDiv.innerHTML = '';
    if (isAdmin) {
        document.getElementById('cancelBtn').style.display = 'none';
    } else {
        document.getElementById('cancelButton').style.display = 'none';
    }
}

function showCancelButton() {
    const cancelButton = document.getElementById('cancelButton');
    cancelButton.style.display = 'inline-block';
    setTimeout(() => {
        cancelButton.classList.add('show');
    }, 10);
}

function hideCancelButton() {
    const cancelButton = document.getElementById('cancelButton');
    cancelButton.classList.remove('show');
    setTimeout(() => {
        cancelButton.style.display = 'none';
    }, 300);
}

    document.addEventListener("DOMContentLoaded", () => {
        const elements = document.querySelectorAll('.stagger-fade');
        elements.forEach((el, i) => {
            el.style.animationDelay = `${i * 0.15}s`;
        });
    });

    function openModal() {
        document.getElementById("exampleModal").classList.add("show");
    }

    function closeModal() {
        document.getElementById("exampleModal").classList.remove("show");
    }

    function toggleTheme() {
        document.body.classList.toggle("dark");
    }

document.addEventListener('DOMContentLoaded', () => {
    const filters = ['minPriceFilter', 'maxPriceFilter', 'roomsFilter',
'keywordFilter'];
    filters.forEach(id => {
        const el = document.getElementById(id);
        if (el) {
            el.addEventListener('input', () => {
                loadProperties();
            });
        }
    });
});
```

```
function resetFilters() {
    document.getElementById("minPriceFilter").value = '';
    document.getElementById("maxPriceFilter").value = '';
    document.getElementById("roomsFilter").value = '';
    document.getElementById("keywordFilter").value = '';

    const btn = document.getElementById('resetFiltersBtn');
    if (btn) {
        btn.classList.remove('highlight-reset');
    }

    loadProperties();
}


document.addEventListener('DOMContentLoaded', () => {
    const toggleBtn = document.getElementById("toggleFiltersBtn");
    const filterDiv = document.getElementById("filters");

    toggleBtn.addEventListener("click", () => {
        if (filterDiv.style.display === "none") {
            filterDiv.style.display = "block";
            toggleBtn.textContent = "Скрыть фильтры";
        } else {
            filterDiv.style.display = "none";
            toggleBtn.textContent = "Показать фильтры";
        }
    });
    ["minPriceFilter", "maxPriceFilter", "roomsFilter",
"keywordFilter"].forEach(id => {
        const el = document.getElementById(id);
        if (el) {
            el.addEventListener("input", () => loadProperties());
        }
    });
});

async function loadPropertyRequests() {
    const response = await
fetch('https://localhost:7167/api/propertyrequest/admin');
    if (!response.ok) {
        alert("Ошибка при загрузке заявок");
        return;
    }

    const requests = await response.json();
    showRequestsUI(requests);
}
function showRequestsUI(requests) {
    const container = document.getElementById("requestList");
    const list = document.getElementById("requests");
    list.innerHTML = "";

    if (requests.length === 0) {
        list.innerHTML = "<p>Нет новых заявок</p>";
    } else {
        requests.forEach(r => {
            const item = document.createElement("li");
            item.innerHTML = `
                <strong>${r.title}</strong> — ${r.address}<br>
                Цена: ${r.price} | Комнат: ${r.rooms}<br>
                Пользователь: ${r.user?.username || "Неизвестно"}<br>
                <button onclick="approveRequest(${r.id})">Одобрить</button>
```

```
                <button onclick="rejectRequest(${r.id})" class="cancel-
button">❌ Отклонить</button>
                <hr>
            `;
            list.appendChild(item);
        });
    }

    container.style.display = "block";
}

async function rejectRequest(id) {
    if (!confirm("Вы уверены, что хотите отклонить заявку?")) return;

    const resp = await
fetch(`https://localhost:7167/api/propertyrequest/reject/${id}`, {
        method: "DELETE"
    });

    if (resp.ok) {
        alert("Заявка отклонена");
        await loadPropertyRequests();
    } else {
        alert("Ошибка при отклонении заявки");
    }
}


async function approveRequest(id) {
    const resp = await
fetch(`https://localhost:7167/api/propertyrequest/approve/${id}`, {
        method: "POST"
    });

    if (resp.ok) {
        alert("Заявка одобрена, объявление создано");
        await loadPropertyRequests(); // обновим список
    } else {
        alert("Ошибка при утверждении заявки");
    }
}
function hideRequests() {
    document.getElementById("requestList").style.display = "none";
}
function showPropertyRequestForm() {
    document.getElementById("propertyRequestForm").style.display = "block";
}

function hidePropertyRequestForm() {
    document.getElementById("propertyRequestForm").style.display = "none";
}
document.getElementById("propertyRequestForm").addEventListener("submit",
async function (e) {
    e.preventDefault();

    const request = {
        userId: currentUserId,
        title: document.getElementById("requestTitle").value,
        address: document.getElementById("requestAddress").value,
        price: parseFloat(document.getElementById("requestPrice").value),
        rooms: parseInt(document.getElementById("requestRooms").value),
        description: document.getElementById("requestDescription").value,
        photoUrl: document.getElementById("requestPhotoUrl").value
```

```
    };

    if (resp.ok) {
        alert("Заявка отправлена администратору");
        hidePropertyRequestForm();
        this.reset();
    } else {
        const errorText = await resp.text();
        alert("Ошибка: " + errorText);
    }
});
```

Приложение Д
(рекомендуемое)
Листинг модульных тестов

Листинг Д.1 – Файл CommentController.cs

```
using Xunit;
using Rental.Controllers;
using Rental.Models;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace Rental.Tests
{
    public class CommentController
    {
        [Fact]
        public async Task AddComment_ReturnsOk()
        {
            var ctx = TestUtils.GetInMemoryContext();
            var controller = new Controllers.CommentController(ctx);

            var comment = new Comment
            {
                UserId = 1,
                PropertyId = 1,
                Content = "все хорошо",
                Rating = 4
            };

            var result = await controller.AddComment(comment);
            var ok = Assert.IsType<OkObjectResult>(result);
            var c = Assert.IsType<Comment>(ok.Value);
            Assert.Equal("все хорошо", c.Content);
        }
    }
}
```

Листинг Д.2 – Файл PropertyServiceTests.cs

```
using Xunit;
using Rental.Services;
using Rental.Models;
using Rental.Data;
using System.Threading.Tasks;
using Rental.Services.Patterns;
using System.Threading.Tasks;

namespace Rental.Tests
{
    public class PropertyServiceTests
    {
        [Fact]
        public async Task BookProperty_ShouldSetIsBookedTrue()
        {
            var context = TestUtils.GetInMemoryContext();

            var property = new Property
            {
                OwnerId = 1,
                Title = "Квартира",
                Address = "Улица",
                Price = 100,
                Rooms = 2,
                IsBooked = false
            };
```

```
            context.Properties.Add(property);
            context.Users.Add(new User { Id = 5, Username = "u", Email = "e",
PasswordHash = "p" });
            await context.SaveChangesAsync();

            var service = new PropertyService(context, new DummyNotifier());

            await service.BookProperty(property.Id, 5, false);

            var updated = await context.Properties.FindAsync(property.Id);
            Assert.True(updated.IsBooked);
            Assert.Equal(5, updated.BookedByUserId);
        }
    }

    public class DummyNotifier : IPropertyChangeNotifier
    {
        public void AddObserver(IPropertyChangeObserver observer) { }

        public void RemoveObserver(IPropertyChangeObserver observer) { }

        public Task NotifyPropertyBooked(Property p) => Task.CompletedTask;

        public Task NotifyPropertyCreated(Property p) => Task.CompletedTask;

        public Task NotifyPropertyDeleted(Property p) => Task.CompletedTask;

        public Task NotifyPropertyUpdated(Property p) => Task.CompletedTask;
    }

}
```

Листинг Д.3 – Файл CommentControllerTests.cs

```
using Xunit;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Rental.Controllers;
using Rental.Models;
using Rental.Data;
using System.Threading.Tasks;
using System;

namespace Rental.Tests
{
    public class CommentControllerTests
    {
        private AppDbContext GetDbContext()
        {
            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase("TestDb")
            .Options;

            return new AppDbContext(options);
        }

        [Fact]
        public async Task AddComment_ReturnsOk_WithCorrectComment()
        {
            var context = GetDbContext();
            var controller = new Controllers.CommentController(context);
            var comment = new Comment
```

```
            {
                UserId = 1,
                PropertyId = 1,
                Content = "Хорошее место!",
                Rating = 5
            };
            var result = await controller.AddComment(comment);
            var okResult = Assert.IsType<OkObjectResult>(result);
            var returnedComment = Assert.IsType<Comment>(okResult.Value);
            Assert.Equal(comment.Content, returnedComment.Content);
        }
    }
}
```

Листинг Д.4 – Файл PropertyRequestControllerTests.cs

```
using Xunit;
using Rental.Controllers;
using Rental.Models;
using Rental.Services;
using Microsoft.AspNetCore.Mvc;
using System.Threading.Tasks;

namespace Rental.Tests
{
    public class PropertyRequestControllerTests
    {
        [Fact]
        public async Task CreateRequest_ShouldReturnOk()
        {
            var ctx = TestUtils.GetInMemoryContext();
            var service = new PropertyService(ctx, new DummyNotifier());
            var controller = new PropertyRequestController(ctx, service);
            var request = new PropertyRequest
            {
                UserId = 1,
                Title = "Тест квартира",
                Address = "ул. Пример",
                Price = 123.45m,
                Rooms = 2
            };
            var result = await controller.CreateRequest(request);
            var ok = Assert.IsType<OkObjectResult>(result);
            var returned = Assert.IsType<PropertyRequest>(ok.Value);
            Assert.Equal("Тест квартира", returned.Title);
        }
    }
}
```

Листинг Д.5 – Файл UserServiceTests.cs

```
using Xunit;
using Rental.Services;
using Rental.Models;
using System.Threading.Tasks;

namespace Rental.Tests
{
    public class UserServiceTests
    {
        [Fact]
        public async Task RegisterUser_ShouldCreateNewUser()
```

```
        {
            var context = TestUtils.GetInMemoryContext();
            var service = new UserService(context);
            var user = await service.RegisterUser("testuser", "123456",
"user@test.com", "Иван", "Иванов");
            Assert.NotNull(user);
            Assert.Equal("testuser", user.Username);
            Assert.Equal("user@test.com", user.Email);
        }
    }
}
```

Листинг Д.6 – Файл TestUtils.cs

```
using Microsoft.EntityFrameworkCore;
using Rental.Data;

namespace Rental.Tests
{
    public static class TestUtils
    {
        public static AppDbContext GetInMemoryContext()
        {
            var options = new DbContextOptionsBuilder<AppDbContext>()
                .UseInMemoryDatabase(databaseName: Guid.NewGuid().ToString())
                .Options;
            return new AppDbContext(options);
        }
    }
}
```

Листинг Д.7 – Файл UserServiceLoginTests.cs

```
using System.Threading.Tasks;
using Xunit;
using Rental.Services;
using Rental.Models;

namespace Rental.Tests
{
    public class LoginUserTests
    {
        [Fact]
        public async Task ValidCredentials_ReturnsMatchingUser()
        {
            // Инициализация контекста в памяти
            var db = TestUtils.GetInMemoryContext();
            var userService = new UserService(db);

            var testUser = new User
            {
                Username = "demoUser",
                PasswordHash = "demoPass",
                Email = "demo@example.com",
                FirstName = "Иван",
                LastName = "Иванов"
            };

            db.Users.Add(testUser);
            await db.SaveChangesAsync();

            // Действие: попытка авторизации
```

```
            var loginResult = await userService.AuthenticateUser("demoUser",
"demoPass");

            // Проверка
            Assert.NotNull(loginResult);
            Assert.Equal("demoUser", loginResult.Username);
        }
    }
}
```

Приложение Е
(рекомендуемое)
Функциональное тестирование

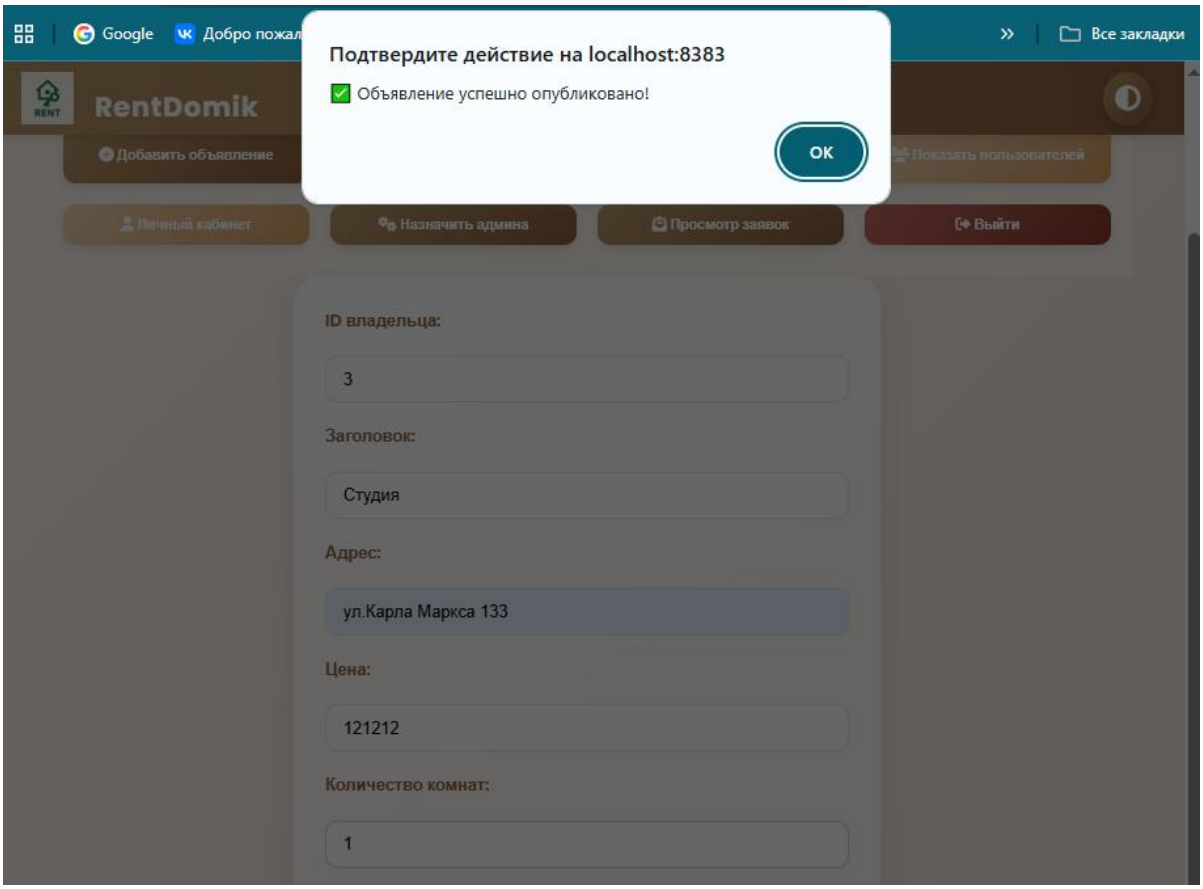

Рисунок Е.1 – Успешная публикация объявления

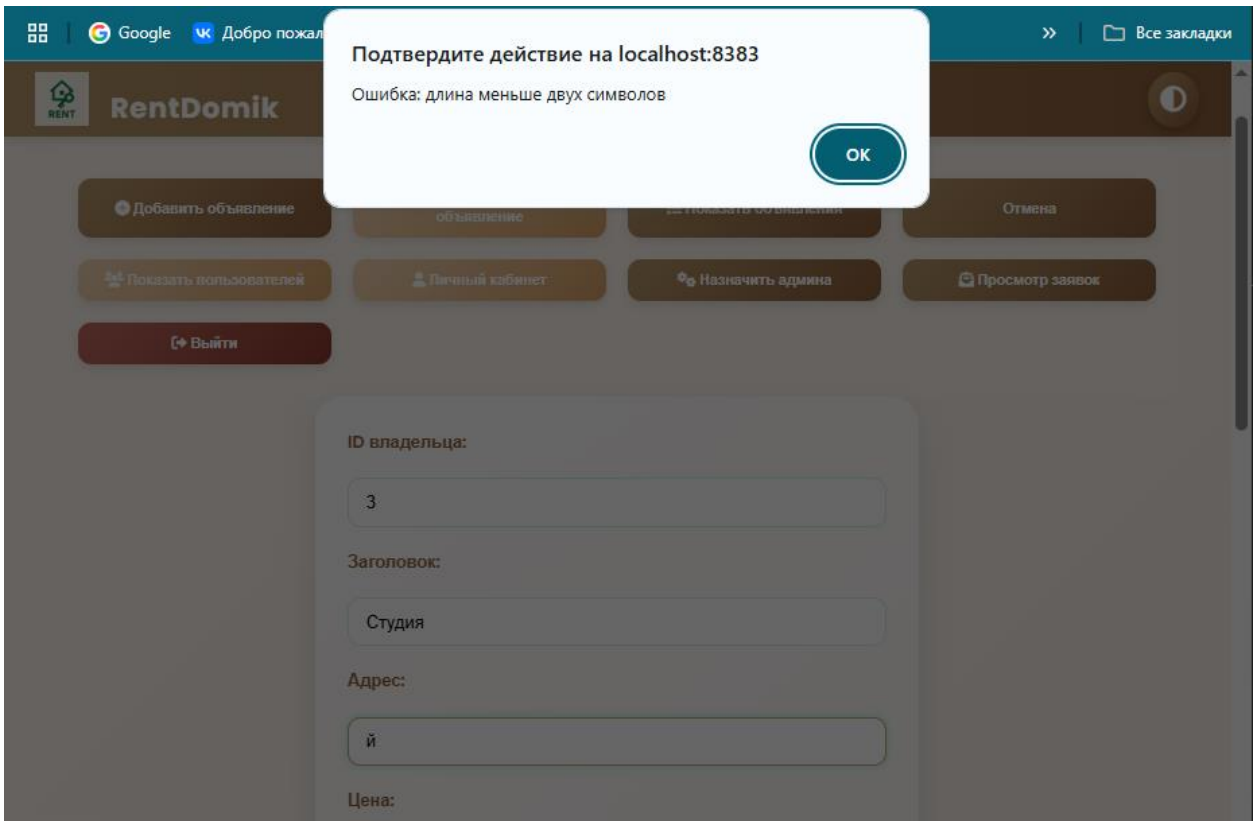

Рисунок Е.2 – Ошибка длина меньше двух символов

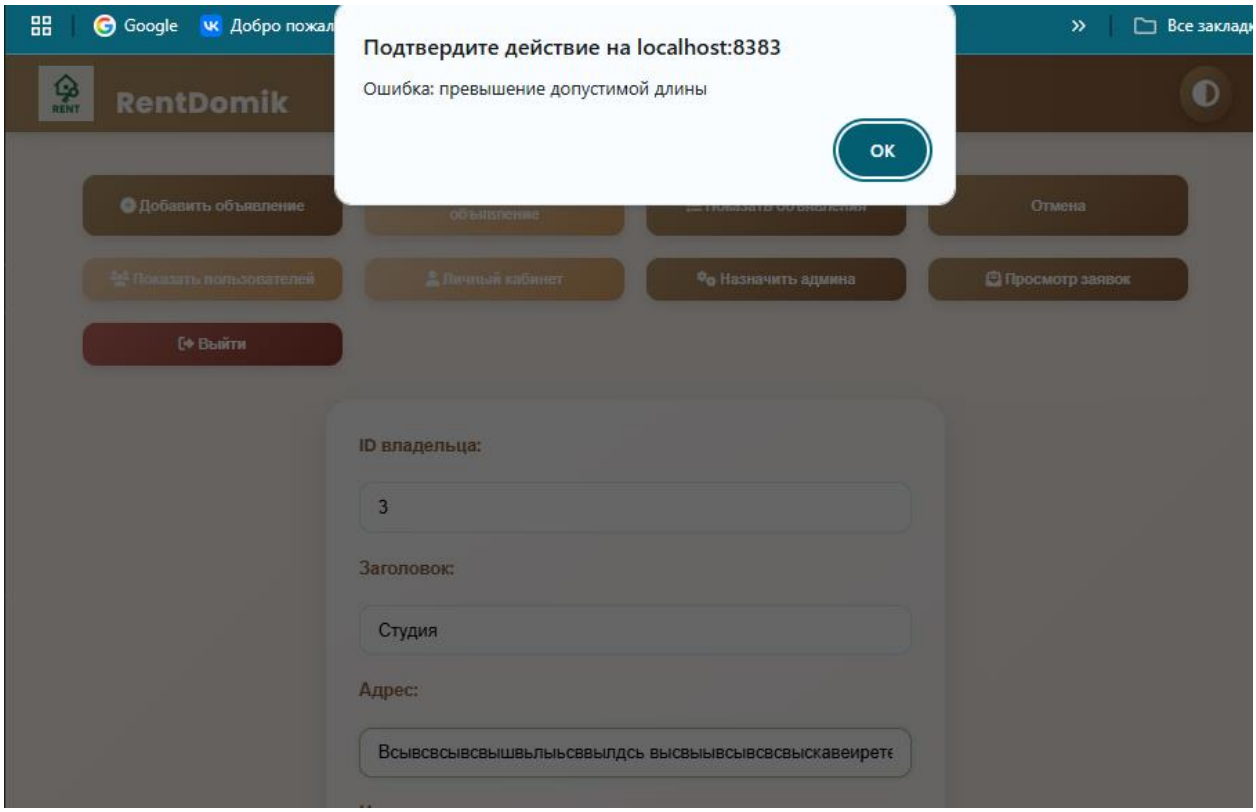

Рисунок Е.3 – Ошибка превышения допустимой длины

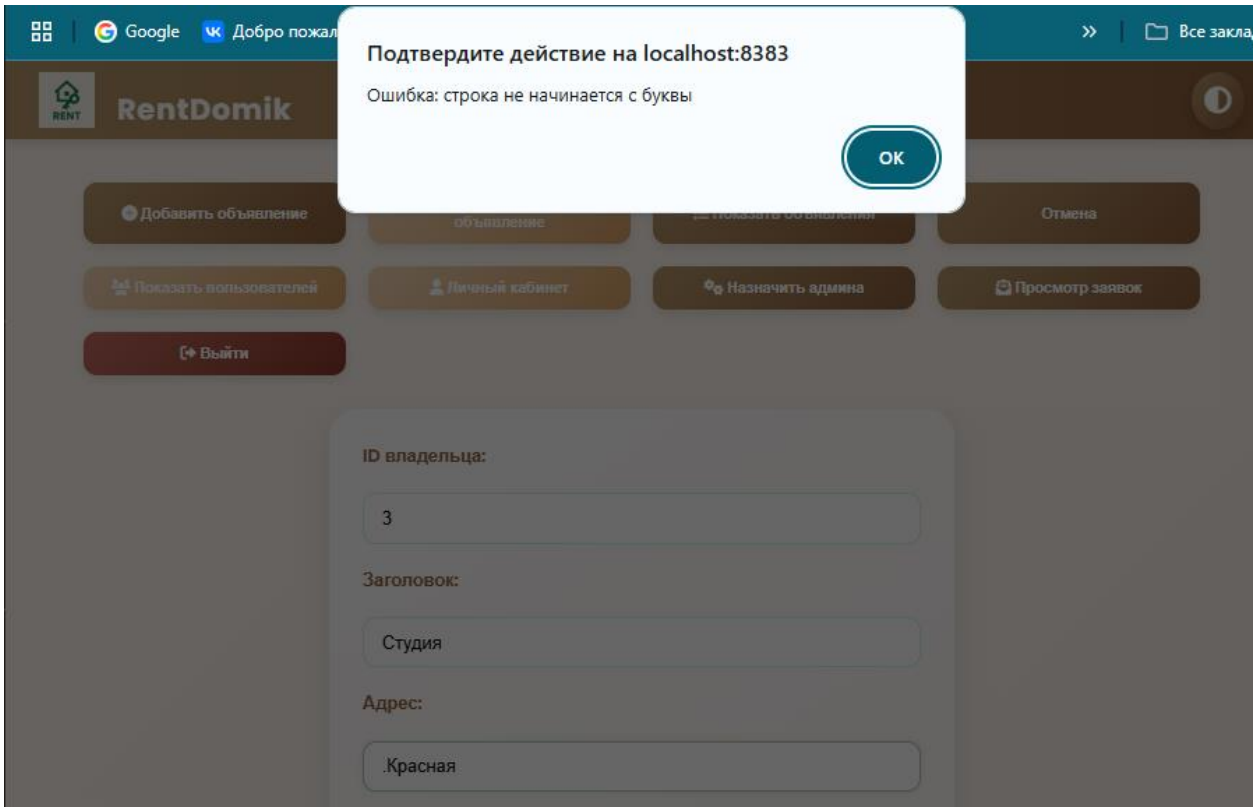

Рисунок Е.4 – Ошибка строка начинается не с буквы

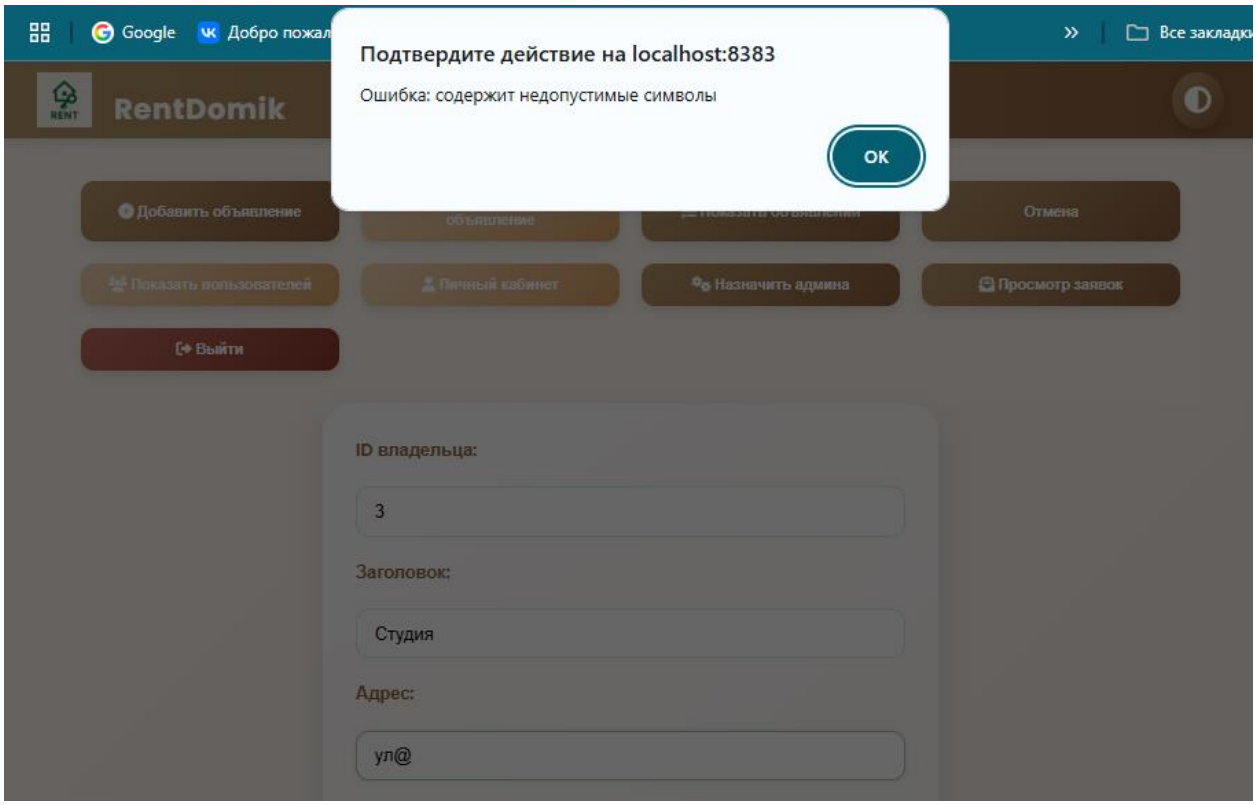

Рисунок Е.5 – Ошибка содержит недопустимые символы

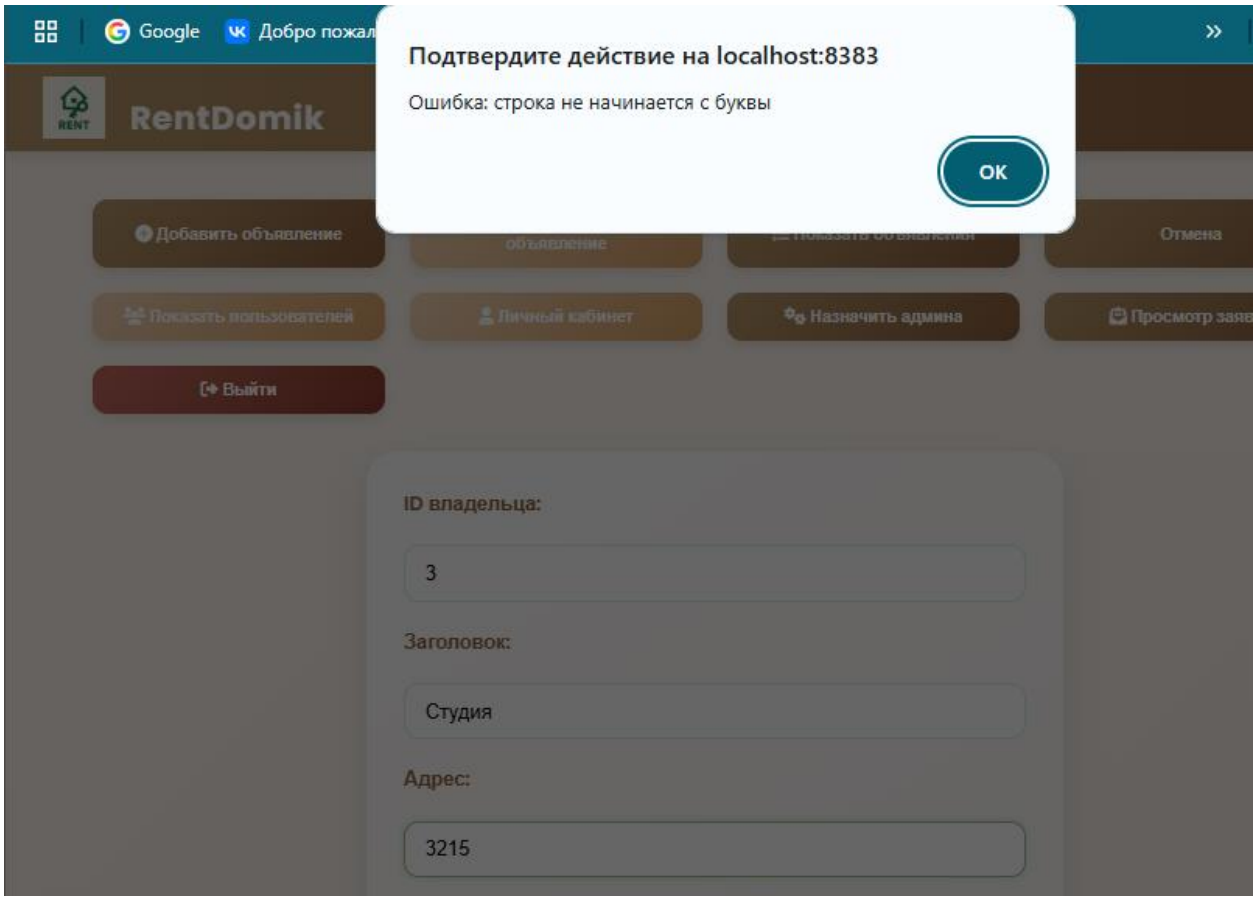

Рисунок Е.6 – Неверный ввод: строка должна начинаться с буквы

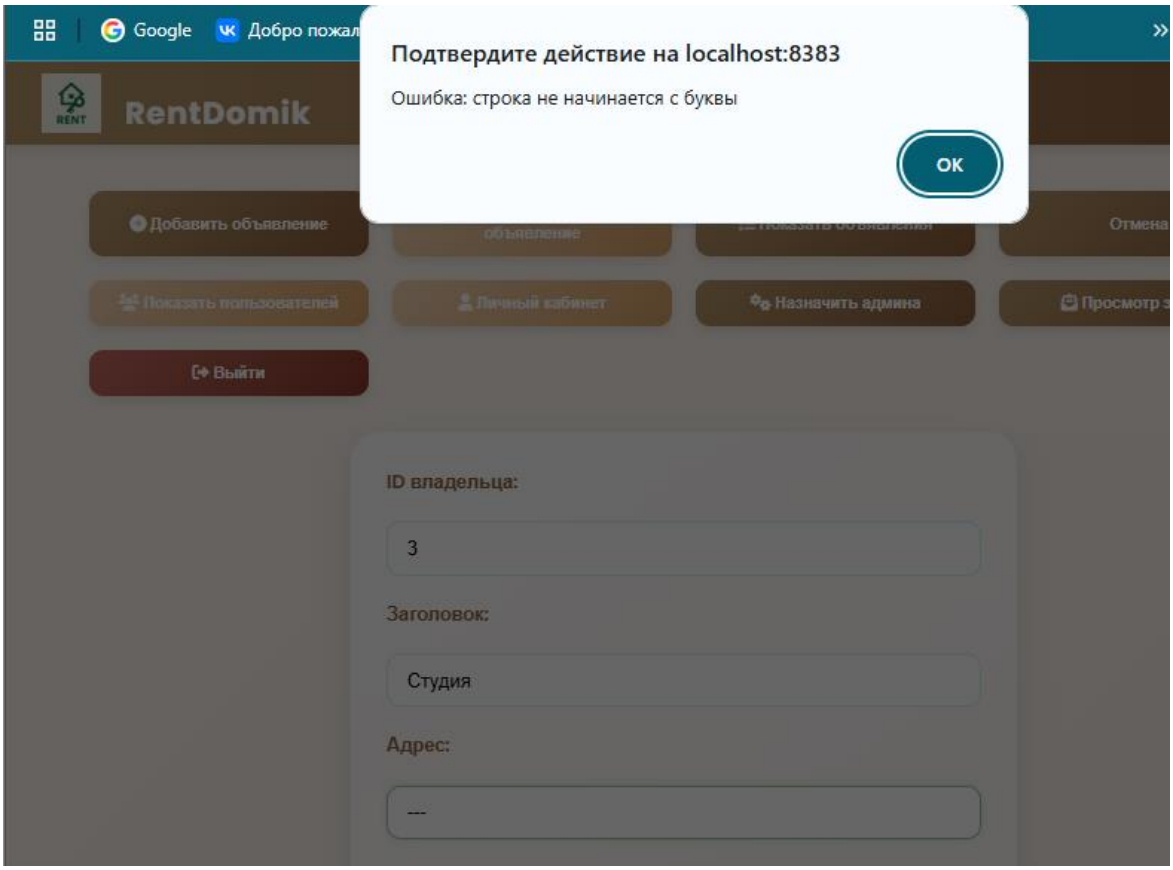

Рисунок Е.7 – Ошибка первый символ не буква

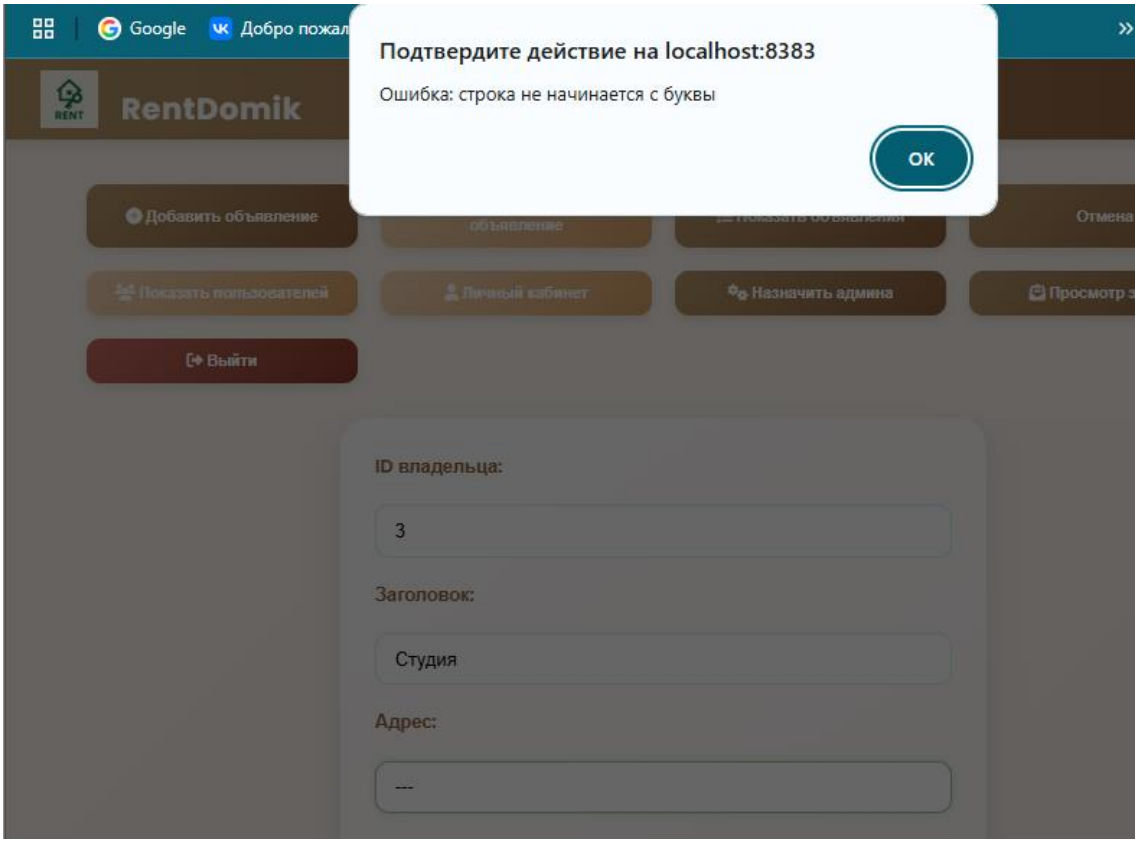

Рисунок Е.8 – Недопустимый формат: начало строки должно быть с буквы

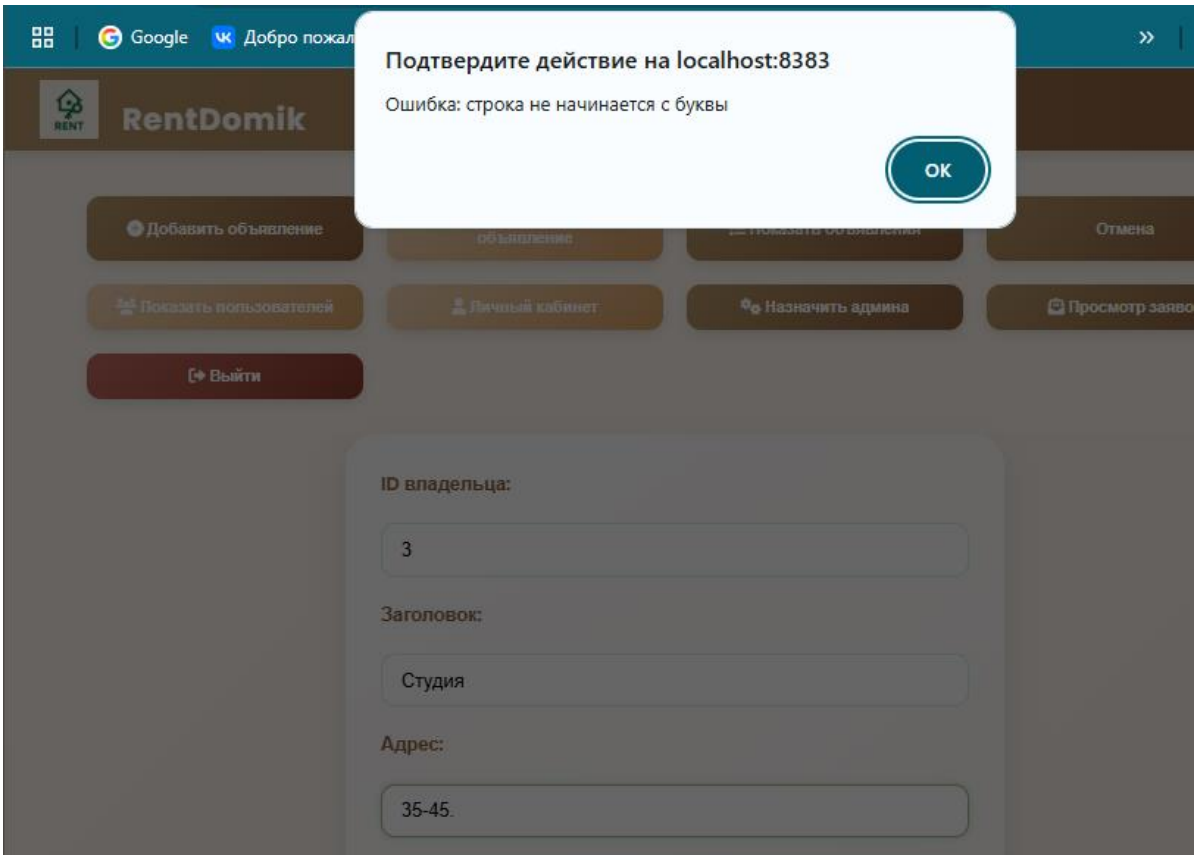

Рисунок Е.9 – Первый символ строки не буква

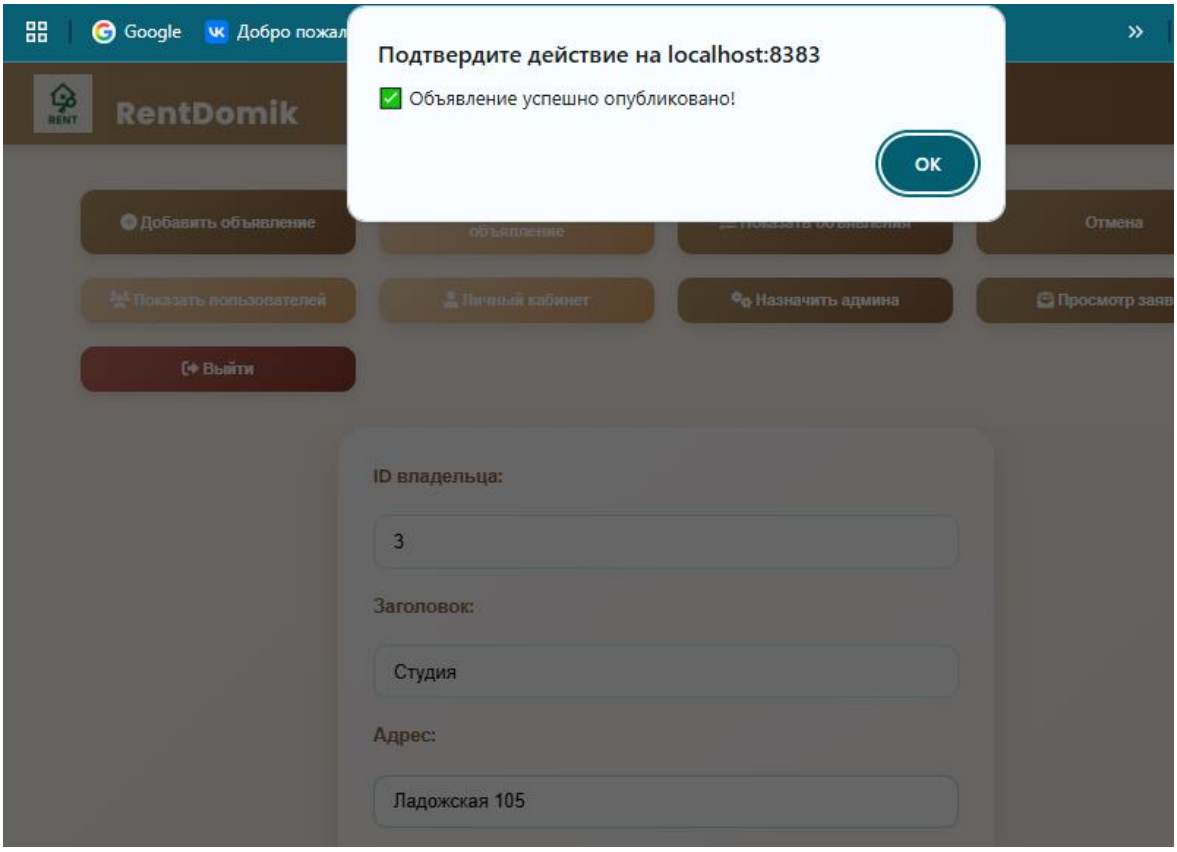

Рисунок Е.10 – Объявление опубликовано

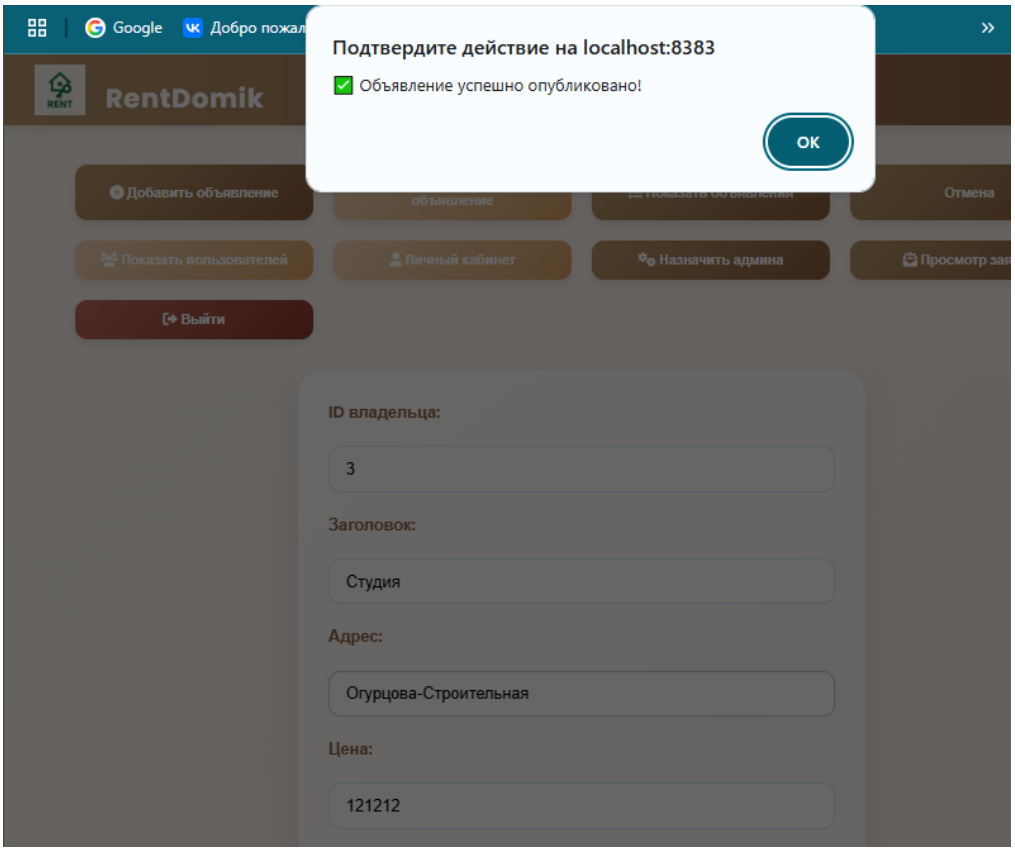

Рисунок Е.11 – Публикация объявления успешна

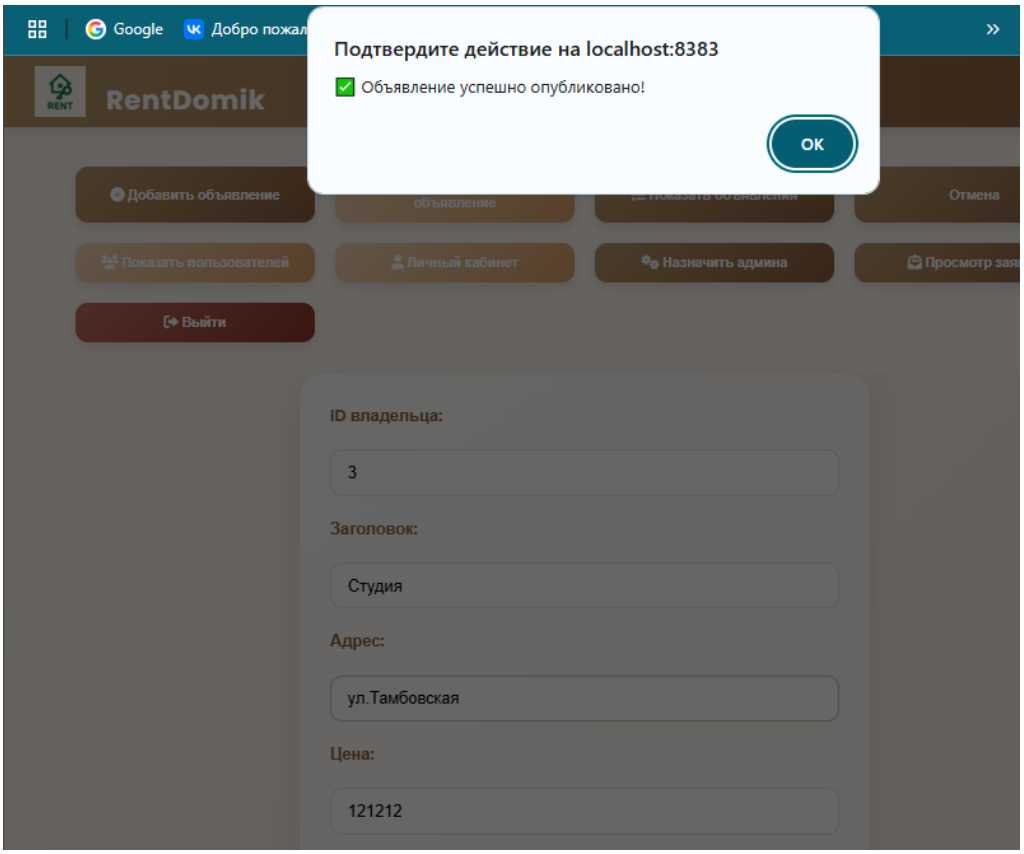

Рисунок Е.12 – Объявление успешно выложено

Приложение Ж

(рекомендуемое)

Листинг функции CreateProperty

```
public async Task<Property> CreateProperty(Property property)
        {
            Property newProperty;
            IPropertyFactory apartmentFactory = new ApartmentFactory();
            if (property.Rooms >= 3)
            {
                newProperty = _houseFactory.CreateProperty(property.OwnerId,
property.Title, property.Address, property.Price, property.Rooms,
property.Description, property.PhotoUrl);
            }
            else
            {
                newProperty = apartmentFactory.CreateProperty(property.OwnerId,
property.Title, property.Address, property.Price, property.Rooms,
property.Description, property.PhotoUrl);
            }
            _context.Properties.Add(newProperty);
            await _context.SaveChangesAsync();
            await _changeNotifier.NotifyPropertyCreated(newProperty);
            return newProperty;
        }
```