

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

Направленность Информационные системы и технологии

БАКАЛАВРСКАЯ РАБОТА

на тему

Разработка web-приложения по продаже и
аренде недвижимости на основе
микросервисной архитектуры

Студент _____ Борискин Кирилл Эдуардович
(подпись, дата) (ФИО полностью)

Руководитель _____ Макарычев П.П.
(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, 2024

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«25» декабря 2023 г

**ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА**

1. Студент Борискин Кирилл Эдуардович гр. 20ВИ1 факультета ВТ _____
<Фамилия, имя, отчество полностью>
2. Тема работы Разработка web-приложения по продаже и аренде недвижимости на основе микросервисной архитектуры

- Тема утверждена приказом ПГУ № _____ от " _____ " _____ 20 _____
2. Руководитель работы Макарычев П.П. _____
3. Задание на работу (назначение разработки, исходные данные и т.п.)

Назначение: обеспечение возможности продавать и сдавать в аренду объекты недвижимости
Основные функции системы:
1. Регистрация и авторизация пользователей в системе
2. Возможность пользователя создавать объявления
3. Возможность пользователя удалять свои объявления
4. Возможность пользователя просматривать объявления
5. Использование пагинации

6. Использование фильтров для поиска
7. Возможность пользователя ставить оценки объявлениям

Технология разработки: объектно-ориентированное программирование.

Язык программирования: C#, Javascript

Среда программирования: Visual Studio 2022

4. Перечень подлежащих разработке вопросов

1. анализ предметной области и постановка задачи;
2. проектирование программных средств;
3. реализация программных средств;
4. тестирование программных средств;
5. оформление ВКР.

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%	08.04.2024 – 13.04.2024	
2	Проектирование программных средств	20%	15.04.2024 – 25.04.2024	
3	Реализация программных средств	30%	26.04.2024 – 12.05.2024	
4	Тестирование программных средств	20%	13.05.2024 – 22.05.2024	
5	Оформление пояснительной записки	20%	23.05.2024 – 03.06.2024	

Дата выдачи задания «21» декабря 2023г.

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 219 листов, 30 рисунков, 32 таблицы, 4 приложения и 11 использованных источников.

НЕДВИЖИМОСТЬ, ПРОДАЖА, АРЕНДА, C#, ASP.NET, EF CORE, SQL, HTML, CSS, JAVASCRIPT, REST API, WEBAPI, МИКРОСЕРВИСНАЯ АРХИТЕКТУРА.

Объект разработки: веб-приложение по продаже и аренде недвижимости.

Предмет разработки: разработка web-приложения для продажи и аренды недвижимости, основанное на микросервисной архитектуре взаимодействия.

Цель работы: разработать программное обеспечение для продажи и аренды недвижимости с использованием микросервисной архитектуры.

Технологии разработки: C#, Visual Studio 2022, JavaScript, HTML, CSS, SQL Server 2022.

Результат работы: в процессе разработки проводилось изучение основных средств разработки и в итоге было разработано и отлажено web-приложение для продажи и аренды недвижимости с использованием микросервисной архитектуры.

					ПГУ 09.03.02 – 8BP191.05 ПЗ			
Изм.	Лист	№ докум.	Подпись	Дата	Разработка web-приложения по продаже и аренде недвижимости на основе микросервисной архитектуры Пояснительная записка	Лит.	Лист	Листов
Разраб.		Борискин К.Э						
Провер.		Макарычев П.П					4	219
Реценз.						Группа 20ВИ1		
Н. Контр.		Такташкин Д.В						
Утверд.		Козлов А.Ю.						

Содержание

Введение	7
1 Постановка задачи и анализ аналогов разрабатываемой системы продажи и аренды недвижимости.....	9
1.1 Анализ бизнес-процессов предприятия	9
1.2 Анализ функциональных систем аналогичного назначения и выбор прототипа	9
1.3 Выбор технологий проектирования и разработки	18
1.4 Постановка и предполагаемое решение задачи проектирования информационной системы предприятия	19
2 Проектирование приложения по продаже и аренде недвижимости	22
2.1 Проектирование логических моделей баз данных	22
2.2 Проектирование программного обеспечения.....	27
2.2.1 Диаграмма вариантов использования.....	27
2.2.2 Диаграмма компонентов	30
2.2.3 Диаграмма классов	32
2.2.4 Диаграмма деятельности	38
2.2.5 Диаграмма последовательности.....	40
3 Планирование и расчет бюджета разрабатываемой системы.....	43
4 Реализация и тестирование системы.....	45
4.1 Реализация структур программного средства	45
4.2 Реализация кода информационной системы	52
4.3 Разработка графического приложения	84
4.4 Тестирование информационной системы.....	94
4.4.1 Классы эквивалентности	94
4.4.2 Функциональная диаграмма и таблица решений.....	97
4.4.3 Функциональное тестирование	102
4.4.4 Метрика Холстеда	107

Заключение	113
Список использованных источников	115
Приложение А (справочное) Глоссарий	117
Приложение Б (рекомендуемое) Код программы	119
Приложение В (рекомендуемое) Исходный код интерфейса	166
Приложение Г (рекомендуемое) Тестирование приложения	193

Введение

Продажа и аренда недвижимости являются важными элементами, определяющими жизненный путь людей, предоставляя им возможности для личностного роста, стабильности и реализации собственных амбиций. Эти услуги играют центральную роль в формировании уровня жизни и благополучия, помогая каждому найти пространство, которое соответствует его уникальным потребностям и желаниям.

В современном мире, где доступность информации и технологии постоянно развиваются, потенциальные покупатели и арендаторы обладают все большими возможностями для самостоятельного поиска и выбора недвижимости. Интернет-платформы, мобильные приложения и виртуальные туры по объектам позволяют людям изучать варианты, не выходя из дома, делая процесс выбора более удобным и персонализированным. Это открывает новые перспективы для принятия обоснованных решений, ведущих к обретению идеального дома или рабочего пространства.

В ходе разработки приложения были проведены:

- анализ предметной области;
- анализ требований к системе;
- проектирование логических моделей баз данных;
- разработка диаграммы вариантов использования;
- разработка диаграммы компонентов;
- разработка диаграммы классов;
- разработка диаграммы деятельности;
- разработка диаграммы последовательности;
- разработка клиентской части приложения;
- разработка серверной части приложения;
- разработка интерфейса приложения;
- тестирование конфигурации.

При создании приложения использовались современные подходы к проектированию и разработке программного обеспечения.

1 Постановка задачи и анализ аналогов разрабатываемой системы продажи и аренды недвижимости

1.1 Анализ бизнес-процессов предприятия

Для системы по продаже и аренде недвижимости были выделены следующие бизнес-процессы:

- поиск недвижимости – позволяет пользователям находить недвижимость по их критериям;
- публикация объявлений – позволяет продавцам размещать объявления о продаже своей недвижимости;
- администрирование объявлений – позволяет продавцам изменять и удалять объявления;
- оценка недвижимости – позволяет пользователям выставять оценки недвижимости, доступной для аренды.

Анализ этих бизнес-процессов позволяет определить ключевые функциональные требования к разрабатываемой системе, а также выявить возможные улучшения и оптимизации процессов для повышения эффективности и удовлетворения потребностей пользователей и бизнеса.

Исходя из вышеописанных бизнес-процессов, необходимо разработать систему, которая будет поддерживать функциональность поиска недвижимости, размещения объявлений, администрирования объявлений и оценки недвижимости. Это требует разработки соответствующих модулей, интерфейсов и функциональности, а также интеграции с базой данных и внешними сервисами, если необходимо.

1.2 Анализ функциональных систем аналогичного назначения и выбор прототипа

Для анализа функциональных систем аналогичного назначения были выделены следующие критерии:

- функциональность – набор возможностей и характеристик системы, обеспечивающих выполнение необходимых операций и процессов;
- интерфейс пользователя – удобство и простота использования приложения, обеспечивающие интуитивное взаимодействие пользователя с системой;
- технологическая платформа – базовые технологии и архитектура, на которых построена система, определяющие ее производительность и совместимость;
- масштабируемость – способность системы эффективно работать и справляться с увеличением объема данных и количества пользователей;
- безопасность – уровень защиты данных и системы от несанкционированного доступа, угроз и уязвимостей;
- интеграция с другими системами – возможность системы взаимодействовать и обмениваться данными с другими программными продуктами и сервисами.

В качестве систем для сравнения выбрано 3 популярных сервиса для покупки и аренды недвижимости: ЦИАН, Юла и Авито.Недвижимость.

ЦИАН имеет широкий спектр инструментов для поиска, покупки, продажи и аренды недвижимости и обладает понятным интерфейсом, который, тем не менее, может быть недостаточно понятен для новых пользователей из-за перегруженности. В нем применяются геоинформационные системы. Также данный сервис хорошо масштабируется в рамках страны, имеет высокие стандарты защиты данных и хорошие возможности для интеграции с другими российскими системами.

Юла – это широкая платформа для объявлений, где можно купить и продать разнообразные товары, включая недвижимость, функции поиска подкреплены фильтрами по типу недвижимости, цене, местоположению и другим параметрам, интерфейс простой и понятный, есть мобильное приложение для удобного доступа к сервису. Основан на простых технологиях

для обеспечения быстрой работы при большом количестве пользователей. Данный сервис хорошо масштабируется и популярен в разных регионах России. В нем предпринимаются базовые меры безопасности, включая проверку объявлений и пользователей. Основной акцент идет на самодостаточность сервиса, поэтому у него ограниченные возможности для интеграции.

Авито.Недвижимость является специализированной частью сервиса Авито, посвящен исключительно недвижимости, его возможности включают в себя покупку, продажу и аренду любой недвижимости, в том числе коммерческой и земельных участков. Интерфейс детализированный и функциональный, представляет подробные фильтры и карту для удобного поиска объектов, а высококачественные фотографии и подробные описания положительно влияют на опыт пользователя с данным приложением. Также в нем используются продвинутое технологии для фильтрации и отображения объявлений, улучшения поиска и предоставления аналитики пользователю. Сервис эффективно масштабируется, имеет продвинутое меры безопасности, а также API для интеграции со сторонними сервисами, что упрощает взаимодействие с другими платформами.

Все вышеописанные системы имеют свои сильные и слабые стороны. Юла выделяется простотой использования и широким охватом, в то время как Авито.Недвижимость предлагает более глубокую специализацию и функционал в области недвижимости, а также лучшую безопасность и возможности интеграции. ЦИАН, в свою очередь, специализируется исключительно на недвижимости и предлагает один из самых продвинутых инструментов для поиска и анализа рынка, делая упор на детальность и точность информации.

Для объективного выбора прототипа был применен Метод Анализа Иерархий (МАИ) Томаса Саати.

МАИ – математический инструмент системного подхода к сложным проблемам принятия решений. МАИ не предписывает лицу, принимающему решение (ЛПР), какого-либо «правильного» решения, а позволяет ему в

интерактивном режиме найти такой вариант (альтернативу), который наилучшим образом согласуется с его пониманием сути проблемы и требованиями к ее решению.

МАИ позволяет понятным и рациональным образом структурировать сложную проблему принятия решений в виде иерархии, сравнить и выполнить количественную оценку альтернатив вариантов решения.

В МАИ Саати предлагает математическую обработку экспертных оценок на основе матричных вычислений и аддитивной (суммирующей) сверки критериев [1].

Метод Саати является одним из наиболее часто используемых методов в задачах многокритериальных систем и заключается в иерархическом представлении элементов, определяющих суть решаемой проблемы.

Построим матрицу попарных сравнений критериев. Результат построения приведен в таблице 1.

Таблица 1 – Попарные сравнения критериев

	Функциональнос ть	Интерфе йс	Платфор ма	Масштабируемос ть	Безопаснос ть	Интеграц ия
Функциональнос ть	1,00	0,50	3,00	3,00	3,00	1,00
Интерфейс	2,00	1,00	5,00	5,00	3,00	2,00
Платформа	0,33	0,20	1,00	0,50	0,33	0,50
Масштабируемос ть	0,33	0,20	2,00	1,00	2,00	0,50
Безопасность	0,33	0,33	3,00	0,50	1,00	0,20
Интеграция	1,00	0,50	2,00	2,00	5,00	1,00
Сумма	5,00	2,73	16,00	12,00	14,33	5,20

Далее нужно построить нормализованную матрицу, что достигается делением каждого элемента на сумму соответствующего столбца.

Результат построения нормализованной матрицы приведен в таблице 2.

Таблица 2 – Нормализованная матрица критериев

	Функциональ ность	Интерф ейс	Платфо рма	Масштабируе мость	Безопасн ость	Интегра ция
Функциональ ность	0,20	0,18	0,19	0,25	0,21	0,19
Интерфейс	0,40	0,37	0,31	0,42	0,21	0,38
Платформа	0,07	0,07	0,06	0,04	0,02	0,10
Масштабируе мость	0,07	0,07	0,13	0,08	0,14	0,10
Безопасность	0,07	0,12	0,19	0,04	0,07	0,04
Интеграция	0,20	0,18	0,13	0,17	0,35	0,19

После чего необходимо построить вектор средних значений каждой строки. Полученный столбец средних значений задает «веса» критериев с точки зрения поставленной цели. Этот столбец называют весовым столбцом критериев по цели. Результат построения приведен в таблице 3.

Таблица 3 – Средние критерии

Критерий	Среднее значение
Функциональность	0,20
Интерфейс	0,35
Платформа	0,06
Масштабируемость	0,10
Безопасность	0,09
Интеграция	0,20

Как видно из результатов столбца, наибольший вес имеет критерий «Интерфейс», за ним следуют функциональность и интеграция, а платформа, масштабируемость и безопасность имеют наименьший вес, что означает что они являются самыми незначительными с точки зрения поставленной цели.

Для применения методики анализа полученных матриц необходимо построить аналогичные матрицы сравнения вариантов по каждому из критериев.

Матрица попарных сравнений и получение векторов весов по критерию «функциональность» приведены в таблицах 4, 5.

Таблица 4 – Матрица попарных сравнений по критерию «функциональность»

	ЦИАН	Юла	Авито.Недвижимость
ЦИАН	1,00	2,00	0,50
Юла	0,50	1,00	0,33
Авито.Недвижимость	2,00	3,00	1,00
Сумма	3,50	6,00	1,83

Таблица 5 – Получение вектора весов объектов по критерию «функциональность»

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение
ЦИАН	0,29	0,33	0,27	0,30
Юла	0,14	0,17	0,18	0,16
Авито.Недвижимость	0,57	0,50	0,55	0,54

Матрица попарных сравнений и получение векторов весов по критерию «интерфейс» приведены в таблицах 6, 7.

Таблица 6 – Матрица попарных сравнений по критерию «интерфейс»

	ЦИАН	Юла	Авито.Недвижимость
ЦИАН	1,00	0,33	0,50
Юла	3,00	1,00	0,20
Авито.Недвижимость	2,00	5,00	1,00
Сумма	6,00	6,33	1,70

Таблица 7 – Получение вектора весов объектов по критерию «интерфейс»

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение

Продолжение таблицы 7

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение
ЦИАН	0,17	0,05	0,29	0,17
Юла	0,50	0,16	0,12	0,26
Авито.Недвижимость	0,33	0,79	0,59	0,57

Матрица попарных сравнений и получение векторов весов по критерию «платформа» приведены в таблицах 8, 9.

Таблица 8 – Матрица попарных сравнений по критерию «платформа»

	ЦИАН	Юла	Авито.Недвижимость
ЦИАН	1,00	3,00	0,50
Юла	0,33	1,00	0,33
Авито.Недвижимость	2,00	3,00	1,00
Сумма	3,33	7,00	1,83

Таблица 9 – Получение вектора весов объектов по критерию «платформа»

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение
ЦИАН	0,29	0,50	0,27	0,35
Юла	0,10	0,17	0,18	0,15
Авито.Недвижимость	0,57	0,50	0,55	0,54

Матрица попарных сравнений и получение векторов весов по критерию «масштабируемость» приведены в таблицах 10, 11.

Таблица 10 – Матрица попарных сравнений по критерию «масштабируемость»

	ЦИАН	Юла	Авито.Недвижимость
ЦИАН	1,00	3,00	0,33
Юла	0,33	1,00	0,20
Авито.Недвижимость	3,00	5,00	1,00

Продолжение таблицы 10

	ЦИАН	Юла	Авито.Недвижимость
Сумма	4,33	9,00	1,53

Таблица 11 – Получение вектора весов объектов по критерию
«масштабируемость»

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение
ЦИАН	0,29	0,50	0,18	0,32
Юла	0,10	0,17	0,11	0,12
Авито.Недвижимость	0,86	0,83	0,55	0,75

Матрица попарных сравнений и получение векторов весов по критерию «безопасность» приведены в таблицах 12, 13.

Таблица 12 – Матрица попарных сравнений по критерию «безопасность»

	ЦИАН	Юла	Авито.Недвижимость
ЦИАН	1,00	4,00	0,50
Юла	0,25	1,00	0,20
Авито.Недвижимость	2,00	5,00	1,00
Сумма	3,25	10,00	1,70

Таблица 13 – Получение вектора весов объектов по критерию «безопасность»

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение
ЦИАН	0,29	0,67	0,27	0,41
Юла	0,07	0,17	0,11	0,12
Авито.Недвижимость	0,57	0,83	0,55	0,65

Матрица попарных сравнений и получение векторов весов по критерию «интеграция» приведены в таблицах 14, 15.

Таблица 14 – Матрица попарных сравнений по критерию «интеграция»

	ЦИАН	Юла	Авито.Недвижимость
ЦИАН	1,00	3,00	0,50
Юла	0,33	1,00	0,11
Авито.Недвижимость	2,00	9,00	1,00
Сумма	3,33	13,00	1,61

Таблица 15 – Получение вектора весов объектов по критерию «интеграция»

	ЦИАН	Юла	Авито.Недвижимость	Среднее значение
ЦИАН	0,29	0,50	0,27	0,35
Юла	0,10	0,17	0,06	0,11
Авито.Недвижимость	0,57	1,50	0,55	0,87

Из получившихся векторов весов объектов по критериям осталось составить матрицу весов альтернатив по каждому критерию, состоящую из полученных весовых столбцов.

Результат построения приведен в таблице 16.

Таблица 16 – Матрица весов альтернатив

	Функциональность	Интерфейс	Платформа	Масштабируемость	Безопасность	Интеграция
ЦИАН	0,30	0,17	0,35	0,32	0,41	0,35
Юла	0,16	0,26	0,15	0,12	0,12	0,11
Авито.Недвижимость	0,54	0,57	0,54	0,75	0,65	0,87

Умножая полученную матрицу на вектор весов критериев (таблица 3) по правилу строка на столбец (матрично), получаем веса альтернатив с точки зрения достижения цели. Весы альтернатив приведены в таблице 17.

Таблица 17 – Веса альтернатив

Прототип	Вес альтернатив
ЦИАН	0,28
Юла	0,18
Авито.Недвижимость	0,65

Таким образом, на основе присвоенных рангов критериев была построена матрица попарных сравнений, нормализована и вычислены веса критериев. Затем эти веса были использованы для получения итоговых оценок каждого прототипа, что позволило выбрать Авито.Недвижимость как наиболее подходящий прототип благодаря его высоким оценкам по всем критериям.

1.3 Выбор технологий проектирования и разработки

Приложение для продажи и аренды недвижимости разрабатывается на основе микросервисной архитектуры. Выбор данной архитектуры обусловлен необходимостью обеспечения высокой масштабируемости, упрощения процесса развертывания и возможности независимого развития различных компонентов системы. Микросервисная архитектура позволяет разрабатывать, тестировать и развертывать каждый сервис независимо, что способствует более быстрому внесению изменений и улучшению функционала приложения.

Для разработки приложения был выбран язык программирования C# в сочетании с фреймворком ASP.NET Core, выбор обусловлен тем, что язык C# обладает высокой производительностью и поддерживается экосистемой .NET, а ASP.NET Core хорошо подходит для создания масштабируемых веб-сервисов, обладает встроенной поддержкой зависимостей и широкими возможностями по настройке

Для хранения данных используется реляционная система управления базами данных SQL Server. Это решение было принято, исходя из следующих соображений:

- SQL обеспечивает строгую согласованность данных и поддержку транзакционности, что критически важно для систем, управляющих сведениями о недвижимости;

- SQL легко масштабируется и поддерживает сложные запросы, что необходимо для обработки больших объемов данных в недвижимости.

Таким образом, выбор C#, ASP.NET Core и SQL для разработки микросервисного приложения по продаже и аренде недвижимости обеспечивает надежность, безопасность и масштабируемость решения. Эти технологии позволяют создать мощное и удобное приложение, которое может эффективно обрабатывать большие объемы данных, обеспечивать высокую производительность и быть готовым к постоянным изменениям рынка и технологий.

1.4 Постановка и предполагаемое решение задачи проектирования информационной системы предприятия

В результате анализа предметной области было принято решение о необходимости разработать приложение, позволяющее создавать свои объявления о продаже/аренде недвижимости и просматривать чужие. Объявления должны включать в себя всю необходимую для покупателя/арендатора информацию. При добавлении объявления необходимо учесть цену недвижимости и ее площадь, тип объявления и всю необходимую информацию о недвижимости, чтобы принять решение о приобретении/аренде. Впоследствии это позволит другим пользователям найти подходящую недвижимость из списка доступных объявлений.

Разрабатываемая система будет базироваться на платформе ASP.NET Core, обладать микросервисной архитектурой со всеми ее преимуществами, а для хранения данных будет использована СУБД SQL Server. В качестве языка разработки будет выбран C# ввиду высокой производительности и простоты разработки.

В качестве прототипа будет использовано приложение Авито.Недвижимость, так как оно обладает широкими возможностями для покупки, продажи и аренды недвижимости, имеет детализированный и функциональный интерфейс с подробными фильтрами и картой, продвинутые технологии для фильтрации и отображения объявлений, так же оно эффективно масштабируется, имеет продвинутые меры безопасности и широкие возможности для интеграции с другими сервисами с помощью API.

Приложение должно иметь понятный пользовательский интерфейс и обновлять его при действиях пользователя, чтобы отображать только актуальные данные. Кроме того, данные должны сохраняться, для того чтобы при повторном входе в приложение пользователь мог увидеть все свои и чужие объявления.

Предполагаемым результатом будет полнофункциональная система, позволяющая пользователям создавать и просматривать объявления о недвижимости. Объявления будут содержать все необходимые параметры, такие как цена, площадь, тип недвижимости и другие важные характеристики, что позволит покупателям и арендаторам принимать обоснованные решения.

Пользовательский интерфейс приложения будет разработан с учетом удобства и интуитивной понятности, предоставляя подробные фильтры для удобного поиска объектов недвижимости. Сохранение данных о пользователях обеспечит возможность просмотра своих и чужих объявлений при повторном входе.

Приложение будет обладать мощными возможностями интеграции, поддерживая взаимодействие с другими сервисами через API, что упростит работу с внешними системами и повысит функциональность.

Важной частью разработки станет обеспечение высокого уровня безопасности данных пользователей и объявлений, включая использование современных методов аутентификации и авторизации. Эффективная масштабируемость приложения будет достигнута за счет оптимизации запросов

к базе данных и управления ресурсами, что позволит поддерживать производительность при увеличении количества пользователей и данных.

Предполагается, что разрабатываемое приложение будет превосходить прототип Авито.Недвижимость по ряду ключевых критериев. Интерфейс будет более современным и интуитивно понятным, учитывающим последние тенденции в UX/UI дизайне. Фильтры и возможности поиска будут улучшены, позволяя пользователям быстрее находить нужные объекты недвижимости. Расширенные возможности интеграции с другими сервисами, включая специализированные API для риэлторов и агентств недвижимости, обеспечат высокую функциональность. Использование новейших технологий и методов проектирования обеспечат более высокий уровень безопасности данных.

Таким образом, разрабатываемое приложение не только будет удобным и функциональным для пользователей, но и готовым к интеграции с другими системами.

2 Проектирование приложения по продаже и аренде недвижимости

2.1 Проектирование логических моделей баз данных

Процесс проектирования базы данных представляет собой последовательность переходов от словесного описания информационной структуры предметной области к формализованному описанию объектов предметной области в терминах некоторой модели.

Логическое проектирование базы данных – это описание базы данных в терминах принятой логической модели данных, в нашем случае – реляционной. Результат – схема реляционной базы данных, в которой информационные объекты и связи, полученные на этапе концептуального проектирования, представляются в виде, допустимом для реализации средствами реляционной модели. Полученные информационные объекты должны отвечать правилам нормализации [2].

В результате анализа предметной области было принято решение выделить 3 базы данных для хранения данных о пользователях, о недвижимости и об оценках на недвижимость

На рисунке 1 изображена логическая диаграмма базы данных для хранения объектов недвижимости.

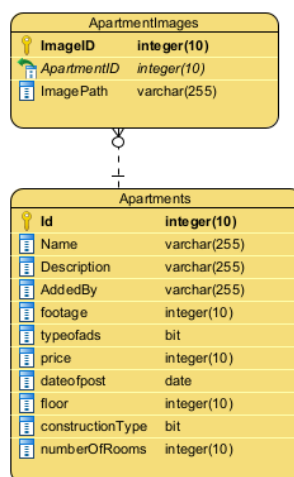


Рисунок 1 – Диаграмма базы данных для объектов недвижимости

База данных объектов недвижимости предназначена для хранения информации о недвижимости, таких как текстовые данные и пути изображений.

Описание атрибутов сущностей ApartmentImages и Apartments приведены в таблицах 18, 19.

Таблица 18 – Описание атрибутов сущности ApartmentImages

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
ImageID	integer		PK
ApartmentID	integer	Not null	FK
ImagePath	varchar	Not null	

Таблица 19 – Описание атрибутов сущности Apartments

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
Id	integer		PK
Name	varchar	Not null	
Description	varchar	Not null	
AddedBy	varchar	Not null	
footage	integer	Not null	
typeofads	bit	Not null	
price	integer	Not null	
dateofpost	date	Not null	
floor	integer	Not null	
constructionType	bit	Not null	
numberOfRooms	Integer	Not null	

На рисунке 2 изображена логическая диаграмма базы данных для хранения информации о пользователях. База данных пользователей предназначена для хранения информации о ролях и пользователях.

пользователя и ролей, обеспечивая связь между ними. Это позволяет пользователям иметь множество ролей и наоборот, ролям быть назначенными множеству пользователей.

Таблица `AspNetUserClaims` предоставляет систему для хранения дополнительных данных о пользователях в формате утверждений (claims). Утверждения могут включать любую информацию о пользователе, например, имя, фамилию, роль или любой другой атрибут, который необходимо сохранить.

Описания атрибутов сущностей базы данных хранения пользователей приведены в таблицах 20 – 23.

Таблица 20 – Описание атрибутов сущности `AspNetUsers`

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
Id	varchar		PK
UserName	varchar		
NormalizedUserName	varchar		
Email	varchar		
NormalizeEmail	varchar		
EmailConfirmed	bit	Not null	
PasswordHash	varchar		
SecurityStamp	varchar		
ConcurrencyStamp	varchar		
PhoneNumber	varchar		
PhoneNumberConfirmed	bit	Not null	
TwoFactorEnabled	bit	Not null	
LockoutEnd	date		
LockoutEnabled	bit	Not null	
AccessFailedCount	integer	Not null	

Таблица 21 – Описание атрибутов сущностиAspNetUserClaims

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
Id	integer		PK
UserId	varchar	Not null	FK
ClaimType	varchar		
ClaimValue	varchar		

Таблица 22 – Описание атрибутов сущностиAspNetUserRoles

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
UserId	integer		PK
RoleId	integer		PK

Таблица 23 – Описание атрибутов сущностиAspNetRoles

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
Id	varchar		PK
Name	varchar		
NormalizedName	varchar		
ConcurrencyStamp	varchar		

На рисунке 3 изображена логическая диаграмма для базы данных оценок, состоящая из одной таблицы.

Таблица Rates предназначена для хранения оценок пользователей и состоит из 4 атрибутов: идентификатор таблицы, номер объекта недвижимости, имя пользователя, добавившего оценку и саму оценку.

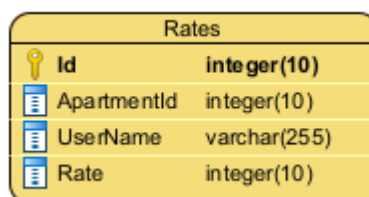


Рисунок 3 – Диаграмма базы данных для хранения оценок

Описание атрибутов сущности базы данных оценок приведено в таблице 24.

Таблица 24 – Описание атрибутов сущности Rates

Атрибуты	Тип данных	Нулевое значение	Ключевое поле
Id	integer		PK
ApartmentId	integer	Not null	
UserName	varchar	Not null	
Rate	integer	Not null	

Таким образом, разработанные сущности и их взаимосвязи обеспечивают грамотное и эффективное хранение информации в приложении. Каждая из трех баз данных структурирована таким образом, чтобы поддерживать целостность данных, удобство доступа и масштабируемость. Использование таких подходов позволяет обеспечить надежную работу системы, а также простоту и удобство в дальнейшем развитии и поддержке приложения.

2.2 Проектирование программного обеспечения

2.2.1 Диаграмма вариантов использования

В приложении должна быть возможность:

- зарегистрироваться;
- осуществить вход по логину и паролю;
- добавить свое объявление;
- посмотреть список своих объявлений;
- посмотреть список чужих объявлений;
- ставить оценки объявлениям.

Регистрация и вход осуществляется в отдельной вкладке, в которой можно будет придумать логин и пароль для входа и записать их в соответствующие текстовые поля или ввести свои логин и пароль для входа. Если логин или

регистрация завершились успехом, происходит переход на вкладку со всеми доступными для просмотра объявлениями и их оценками.

В представлении, отвечающем за личный кабинет пользователя, должна быть возможность добавить свое объявление с помощью отправки формы. В этой форме пользователь может указать адрес объекта, описание со всей необходимой для клиента информацией, цену, площадь, а также тип объявления.

Авторизованному пользователю доступна, помимо входа в личный кабинет, возможность оставлять оценки на объявления других пользователей. Для этого нужно из выпадающего списка выбрать оценку и нажать кнопку «оценить» напротив нужного объекта недвижимости.

На основе сформулированных требований к сервису была разработана диаграмма вариантов использования, представленная на рисунке 4.

Диаграмма вариантов использования (англ. Use-case diagram) – диаграмма, описывающая, какой функционал разрабатываемой программной системы доступен каждой группе пользователей [3].

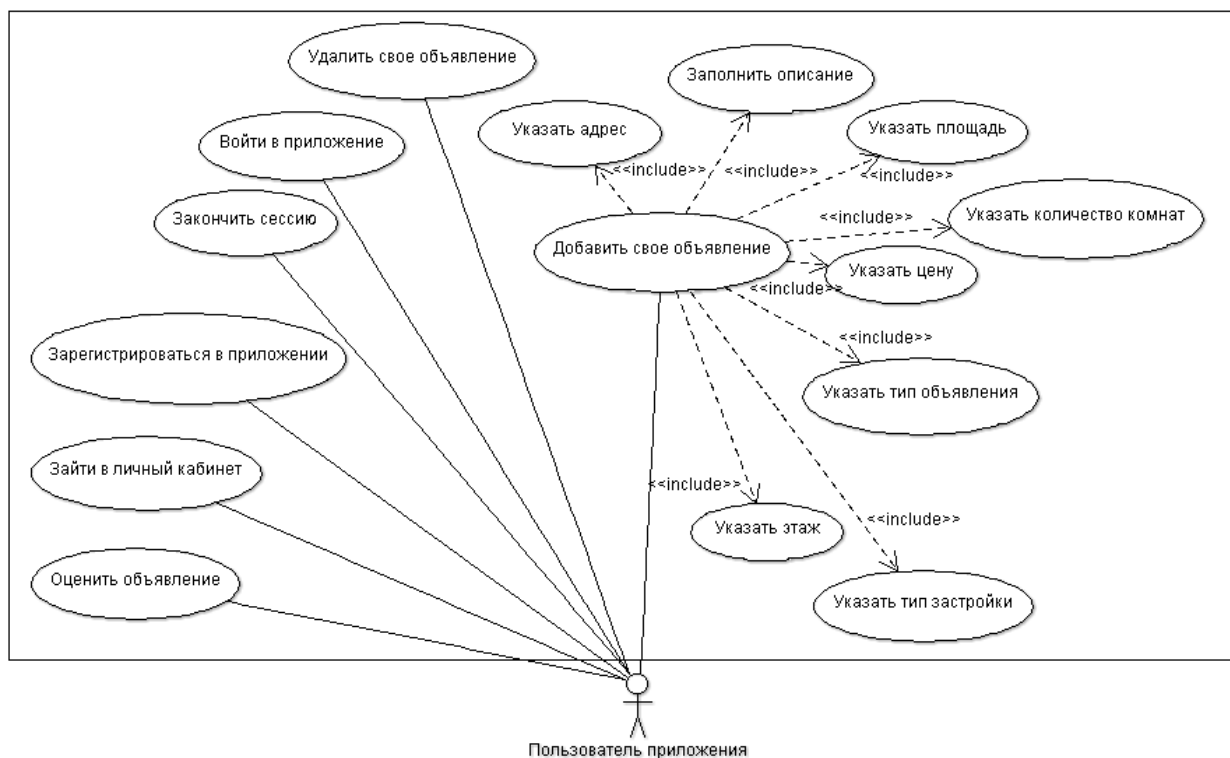


Рисунок 4 – Диаграмма вариантов использования системы

На рисунке 4 изображена диаграмма вариантов использования для разрабатываемой системы.

Актер, представленный на диаграмме вариантов использования – пользователь приложения.

Для актёра «Пользователь приложения» доступны следующие варианты использования:

- зарегистрироваться в приложении – пользователь придумывает логин и пароль, которые будут его идентифицировать, а также указывает адрес электронной почты;
- войти в приложение – осуществляется аутентификация пользователя по логину и паролю;
- закончить сессию – пользователь имеет возможность выйти из аккаунта, если хочет закончить сессию, сменить аккаунт или зарегистрировать новый;
- зайти в личный кабинет – для аутентифицированного пользователя доступны объявления о продаже/сдаче недвижимости, которые он добавлял;
- оценить объявление – для аутентифицированного пользователя доступна возможность оценить объявление по шкале от 1 до 5;
- удалить свое объявление – при необходимости пользователь имеет возможность удалить свое объявление;
- добавить свое объявление – пользователю доступна возможность добавлять свои объявления, которые другие пользователи могут увидеть на вкладке Главная, этот вариант использования включает в себя возможность добавить название объявления, заполнить описание, указать площадь недвижимости, его цену, тип объявления, а также этаж, количество комнат и тип застройки.

Все варианты использования реализованы функциями в приложении.

2.2.2 Диаграмма компонентов

Стиль микросервисов – это подход, при котором единое приложение строится как набор небольших сервисов, каждый из которых работает в собственном процессе и коммуницирует с остальными используя легковесные механизмы, как правило HTTP. Эти сервисы построены вокруг потребностей и развертываются независимо с использованием полностью автоматизированной среды. Существует абсолютный минимум централизованного управления этими сервисами. Сами по себе эти сервисы могут быть написаны на разных языках и использовать разные технологии хранения данных.

Микросервисное приложение уместнее всего сравнить с монолитным приложением, построенном как единое целое. Приложения часто включают три основные части: пользовательский интерфейс, база данных и сервер. Серверная часть обрабатывает HTTP запросы, выполняет доменную логику, запрашивает и обновляет данные в БД. Любое изменение в системе приводит к пересборке и развертыванию новой версии серверной части приложения.

Монолитный сервер – довольно очевидный способ построения подобных систем. Вся логика по обработке запросов выполняется в единственном процессе, при этом можно использовать возможности языка программирования для разделения приложения на классы и функции. Можно запускать и тестировать приложение на машине разработчика и использовать стандартный процесс развертывания для проверки изменений перед опубликованием. Можно масштабировать монолитное приложение горизонтально путем запуска нескольких физических серверов за балансировщиком нагрузки.

Монолитные приложения могут быть успешными, но все больше людей разочаровываются в них, особенно в свете того, что все больше приложений развертываются в облаке. Любые изменения, даже самые небольшие, требуют пересборки и развертывания всего монолита. С течением времени, становится труднее сохранять хорошую модульную структуру, изменения логики одного

модуля имеют тенденцию влиять на код других модулей. Масштабировать приходится все приложение целиком, даже если это требуется только для одного модуля этого приложения [4].

При проектировании серверной части приложения было выделено 3 микросервиса: микросервис для работы с пользователями, который будет выполнять функции регистрации и аутентификации пользователей, микросервис недвижимости, который будет работать с недвижимостью, и микросервис оценок, который будет работать с оценками на недвижимость. Из основных модулей приложения была построена диаграмма компонентов, которая изображена на рисунке 5.

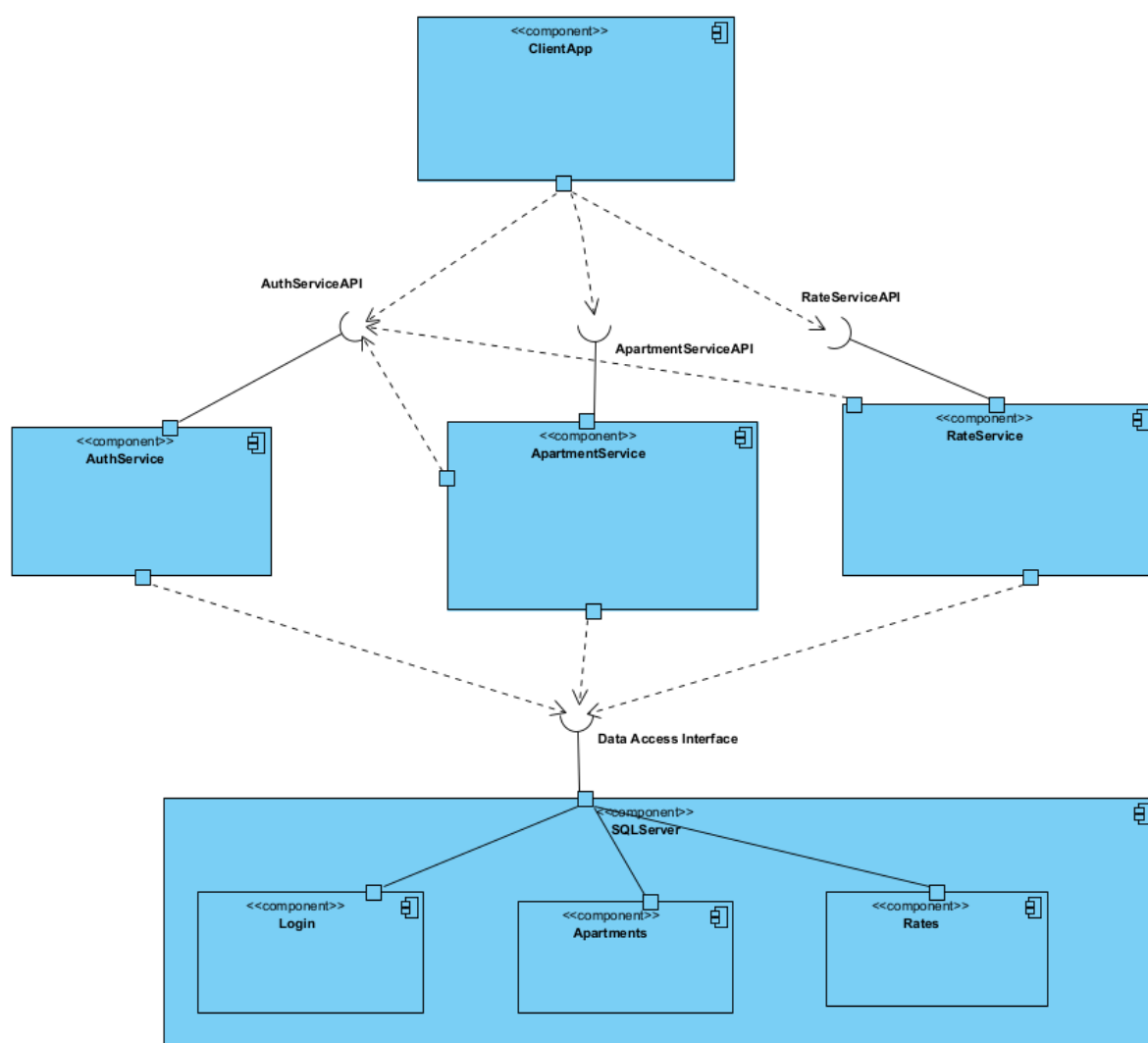


Рисунок 5 – Диаграмма компонентов системы

На диаграмме изображены три микросервиса: AuthService, ApartmentService и RateService, а также клиентское приложение ClientApp. Клиентское приложение обращается к микросервисам через их API с помощью HTTP-запросов для получения данных. Микросервис AuthService предоставляет AuthServiceAPI. ApartmentService, в свою очередь, предоставляет ApartmentServiceAPI и обращается к AuthServiceAPI. RateService также использует AuthServiceAPI и предоставляет свой API – RateServiceAPI. Все три микросервиса обращаются к серверу SQL через Data Access Interface для доступа к данным в базах данных.

2.2.3 Диаграмма классов

Для приложения были построены четыре диаграммы классов: одна для клиентского приложения (рисунок 6) и три для микросервисов.

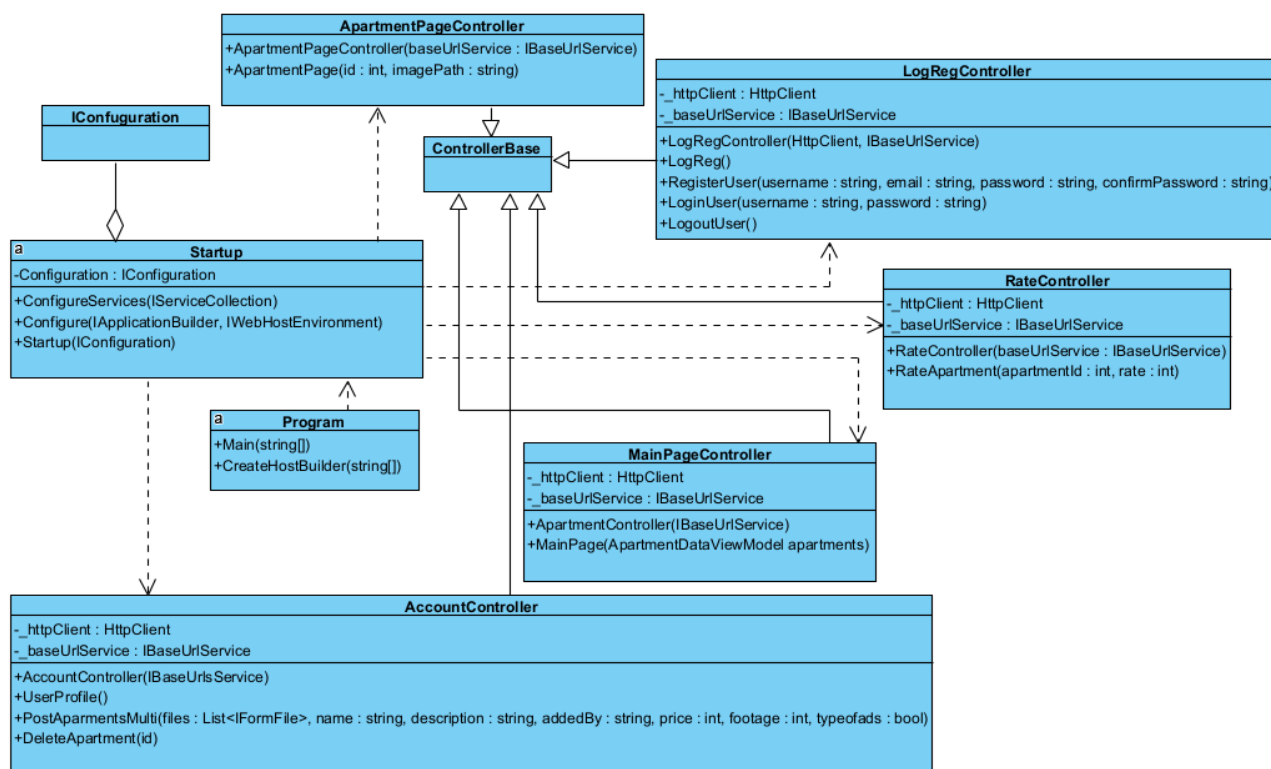


Рисунок 6 – Диаграмма классов для клиентского приложения

Диаграмма классов клиентского приложения по продаже и аренде недвижимости включает несколько ключевых компонентов, взаимодействующих друг с другом для обеспечения функциональности системы.

Центральным элементом системы является класс Program, содержащий методы Main и CreateHostBuilder, которые инициализируют приложение и создают объект хоста.

Класс Startup отвечает за конфигурацию приложения, реализуя методы ConfigureServices и Configure, а также используя интерфейс IConfiguration для доступа к настройкам конфигурации.

В приложении присутствуют несколько контроллеров, каждый из которых наследует базовый функционал от класса ControllerBase.

Класс AccountController управляет функциональностью, связанной с аккаунтами пользователей, и содержит методы для публикации и удаления недвижимости, а также метод для получения профиля пользователя. Этот класс использует экземпляры HttpClient и IBaseUrlsService для выполнения HTTP-запросов.

Контроллер ApartmentPageController управляет отображением страниц апартаментов и взаимодействует с базовым сервисом URL через соответствующие методы.

Класс LogRegController обрабатывает регистрацию и авторизацию пользователей, предоставляя методы для регистрации, входа в систему и выхода из нее.

Контроллер MainPageController отвечает за главную страницу, используя сервисы для получения данных об объектах недвижимости и их изображения.

Класс RateController управляет функциональностью, связанной с оценкой недвижимости, предоставляя метод для оценки конкретного объекта.

Взаимодействие между этими классами осуществляется через внедрение зависимостей (DI), что позволяет гибко управлять компонентами и тестировать их.

Каждый контроллер использует `HttpClient` и `IbaseUrlsService` для взаимодействия с внешними сервисами и API.

Таким образом, диаграмма иллюстрирует архитектуру приложения, построенного на принципах модульности и повторного использования компонентов.

На рисунке 7 изображена диаграмма классов микросервиса недвижимости. Микросервис недвижимости имеет точку входа – класс `Program`, который создает и настраивает веб-хост, используя `CreateHostBuilder`, который вызывает метод `ConfigureWebHostDefaults` для настройки веб-хоста с использованием класса `Startup`.

Класс `Startup` в ASP.NET Core используется для настройки и инициализации веб-приложения. Он включает два метода: `ConfigureServices`, где регистрируются необходимые сервисы в контейнере зависимостей, и `Configure`, где настраивается конвейер обработки запросов с помощью различных `middleware`-компонентов. Эти методы обеспечивают гибкость и контроль над конфигурацией приложения, начиная от настройки баз данных и аутентификации до маршрутизации и обработки статических файлов.

Класс `ApartmentsController` наследует от класса `ControllerBase`, обрабатывает HTTP-запросы, связанные с недвижимостью. Класс содержит методы для получения, добавления, обновления и удаления данных о недвижимости из базы данных.

Также данный класс использует `HttpClient` для отправки запросов к другому микросервису (`AuthService`) для аутентификации.

Класс `ApartmentDbContext` представляет собой контекст базы данных, настраивает соединение с базой данных и предоставляет доступ к таблицам.

Модель `ApartmentNew` представляет собой сущность недвижимости и отображает таблицу в базе данных. Она содержит свойства для идентификации, названия, описания, добавившего пользователя, площади, типа объявления и цены объекта недвижимости.

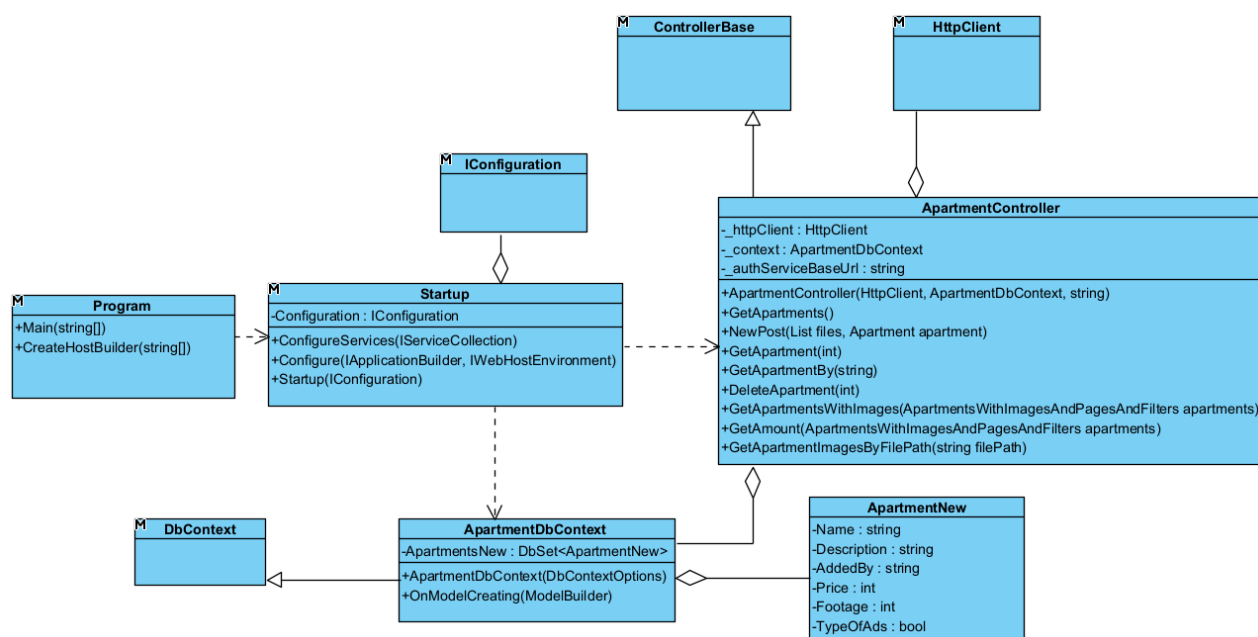


Рисунок 7 – Диаграмма классов для микросервиса недвижимости

На рисунке 8 изображена диаграмма классов микросервиса оценок. Микросервис оценок также имеет точку входа – класс Program, который создает и настраивает веб-хост, используя CreateHostBuilder, который вызывает метод ConfigureWebHostDefaults для настройки веб-хоста с использованием класса Startup. Класс Startup используется для настройки приложения ASP.NET Core. Класс RateController наследует от класса ControllerBase, обрабатывает HTTP-запросы, связанные с оценками. Класс содержит методы для получения, добавления и обновления данных об оценках из базы данных, а также метод для получения всех средних оценок для каждой недвижимости. Также данный класс использует HttpClient для отправки запросов к другому микросервису (AuthService) для аутентификации. Класс RatetDbContext представляет собой контекст базы данных, настраивает соединение с базой данных и предоставляет доступ к таблицам. Модель RateModel представляет собой сущность оценки и отображает таблицу в базе данных. Она содержит свойства для идентификации, ID объекта, имени добавившего оценку в базу и для самой оценки.

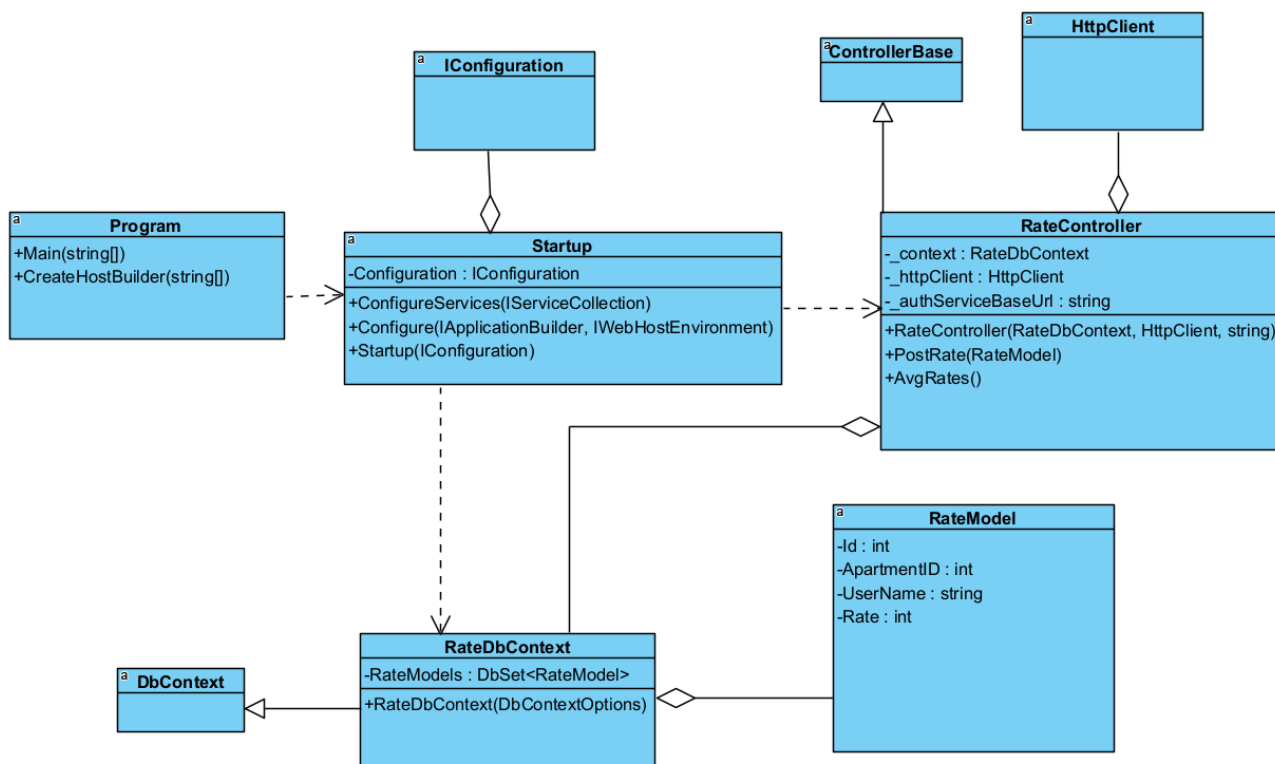


Рисунок 8 – Диаграмма классов для микросервиса оценок

На рисунке 9 изображена диаграмма классов микросервиса аутентификации.

Данный микросервис также имеет точку входа – класс Program, который создает и настраивает веб-хост, используя CreateHostBuilder, который вызывает метод ConfigureWebHostDefaults для настройки веб-хоста с использованием класса Startup.

Класс Startup используется для настройки приложения ASP.NET Core. Класс AuthenticationController обрабатывает HTTP-запросы для проверки аутентификации пользователя по Bearer токenu в заголовке запроса. Данный класс использует UserManager<AppUser> для работы с пользователями и проверки их аутентификации.

Класс `LoginController` отвечает за логику логина пользователя. Класс обрабатывает POST-запросы с данными пользователя для аутентификации и генерации JWT токена, который затем возвращается клиенту.

Класс `RegisterController` отвечает за регистрацию новых пользователей. Данный класс обрабатывает POST-запросы с данными нового пользователя, создает новую учетную запись в системе и генерирует JWT токен для нового пользователя.

Классы `AppUser`, `AppRole` и `AuthDbContext` отвечают за работу с данными пользователей и ролей в базе данных. `AppUser` и `AppRole` представляют собой модели пользователей и ролей, а `AuthDbContext` – контекст базы данных для работы с этими моделями.

Класс `GenerateBearerTokenService` представляет сервис для генерации Bearer токенов аутентификации. Данный класс используется для создания JWT токенов на основе информации о пользователе и секретного ключа приложения. Интерфейс `IGenerateBearerToken` определяет метод для генерации Bearer токенов аутентификации. Этот интерфейс используется для внедрения зависимостей в классы, которые требуют генерации токенов.

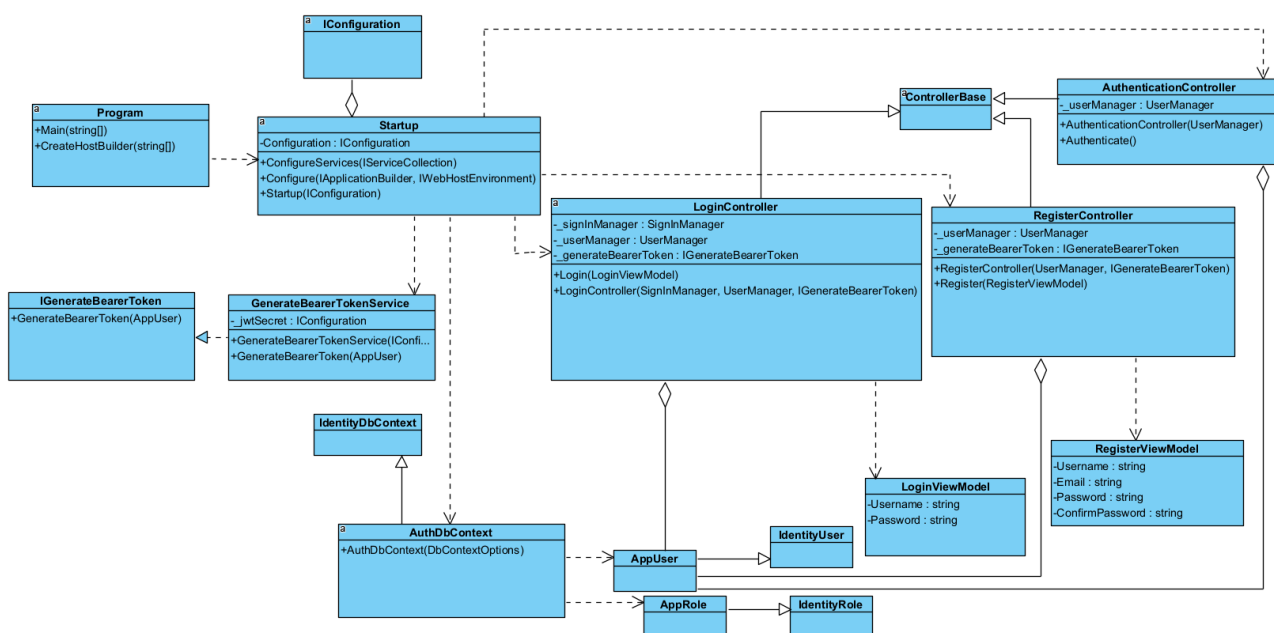


Рисунок 9 – Диаграмма классов для микросервиса аутентификации

Таким образом, представленные диаграммы классов демонстрируют эффективную структуру приложения для покупки и аренды недвижимости. Такая организация классов обеспечивает ясное и логичное представление данных, а также упрощает взаимодействие между различными компонентами системы.

Спроектированные связи между сущностями способствуют устойчивой и масштабируемой архитектуре, что обеспечивает стабильную и производительную работу приложения.

2.2.4 Диаграмма деятельности

Диаграмма деятельности – это вид диаграммы в UML, который используется для моделирования рабочих процессов или потоков действий в системе.

Диаграмма детализирует последовательность выполняемых действий и их зависимости друг от друга, включая разветвления, параллельные процессы, точки принятия решений и условия переходов.

Данная диаграмма помогает визуализировать алгоритмы, бизнес-процессы и логику выполнения задач в системе, что упрощает понимание и анализ структуры и динамики процессов.

Она также позволяет выявить потенциальные узкие места и оптимизировать рабочие процессы, предоставляя визуальное представление взаимодействий между различными элементами системы.

Используя такие диаграммы, можно лучше понять и задокументировать функциональные требования, взаимодействия и последовательности действий в сложных системах, что способствует более эффективному проектированию и внедрению решений.

Для приложения была построена диаграмма деятельности (рисунок 10) для процесса добавления объявления.

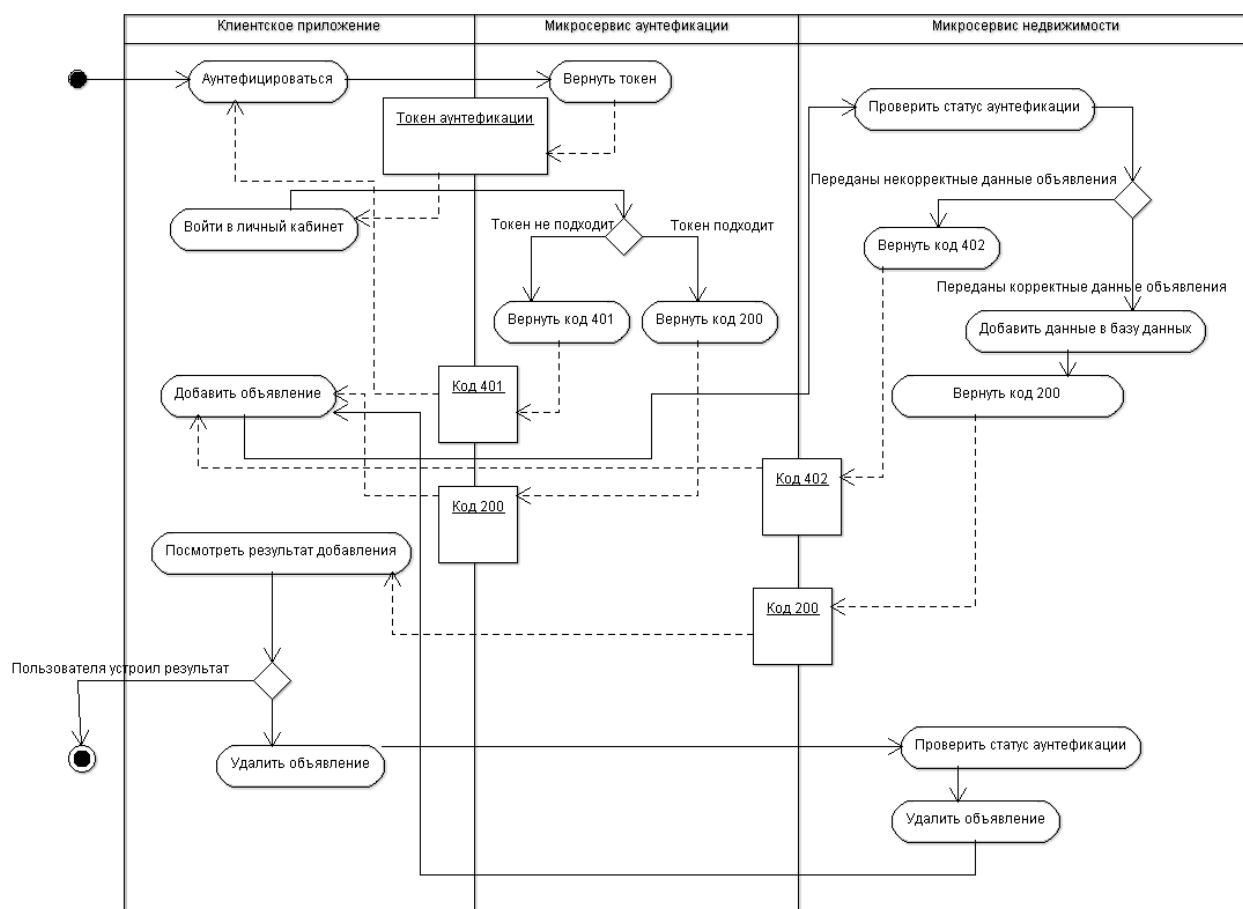


Рисунок 10 – Диаграмма деятельности добавления объявления

Для того чтобы добавить объявление, для начала нужно аутентифицироваться на сайте и получить токен аутентификации, затем зайти в личный кабинет. При попытке войти в личный кабинет клиентское приложение обращается к микросервису аутентификации и проверяет правильность токена, если токен не подходит, например если истекло время сессии, то приложению возвращается код 401 (Unauthorized) и пользователь возвращается на страницу аутентификации, а если токен правильный, то происходит успешный вход в личный кабинет. Затем пользователь должен с помощью формы ввести данные объявления, и эти данные отправляются в микросервис недвижимости. Микросервис вновь проверяет правильность токена, а затем корректность введенных данных, если они некорректны, возвращается код 402 (Bad Request) с описанием всех некорректно введенных полей, если возвращается код 200, это

значит, что объявление успешно добавлено и пользователь может в своем личном кабинете посмотреть корректность введенных им данных, если пользователя устраивает результат, выполнение заканчивается, если пользователю хочется что-то изменить, то он может удалить получившееся объявление и создать новое.

2.2.5 Диаграмма последовательности

Диаграмма последовательности – это графическое представление, которое отображает взаимодействие между объектами в системе.

Диаграмма показывает порядок обмена сообщениями между этими объектами во времени, а также их жизненный цикл, позволяя визуализировать процесс выполнения различных операций и взаимосвязей.

Диаграмма последовательности состоит из следующих элементов:

- объекты – сущности, которые взаимодействуют друг с другом;
- линии жизни – вертикальные линии с блоком активности, отображают активность или исполнение функций объекта;
- сообщения – стрелки, направленные от одной линии жизни к другой, обозначают обмен информацией между объектами.

Для приложения была построена диаграмма последовательности для иллюстрации того, как приложение будет реагировать на фильтрацию объявлений и выставление оценки объявлению.

Для этого пользователю необходимо найти с помощью фильтров нужное объявление, затем нажать на объявление для подробной информации и поставить оценку недвижимости.

Диаграмма последовательности для данной ситуации изображена на рисунке 11.

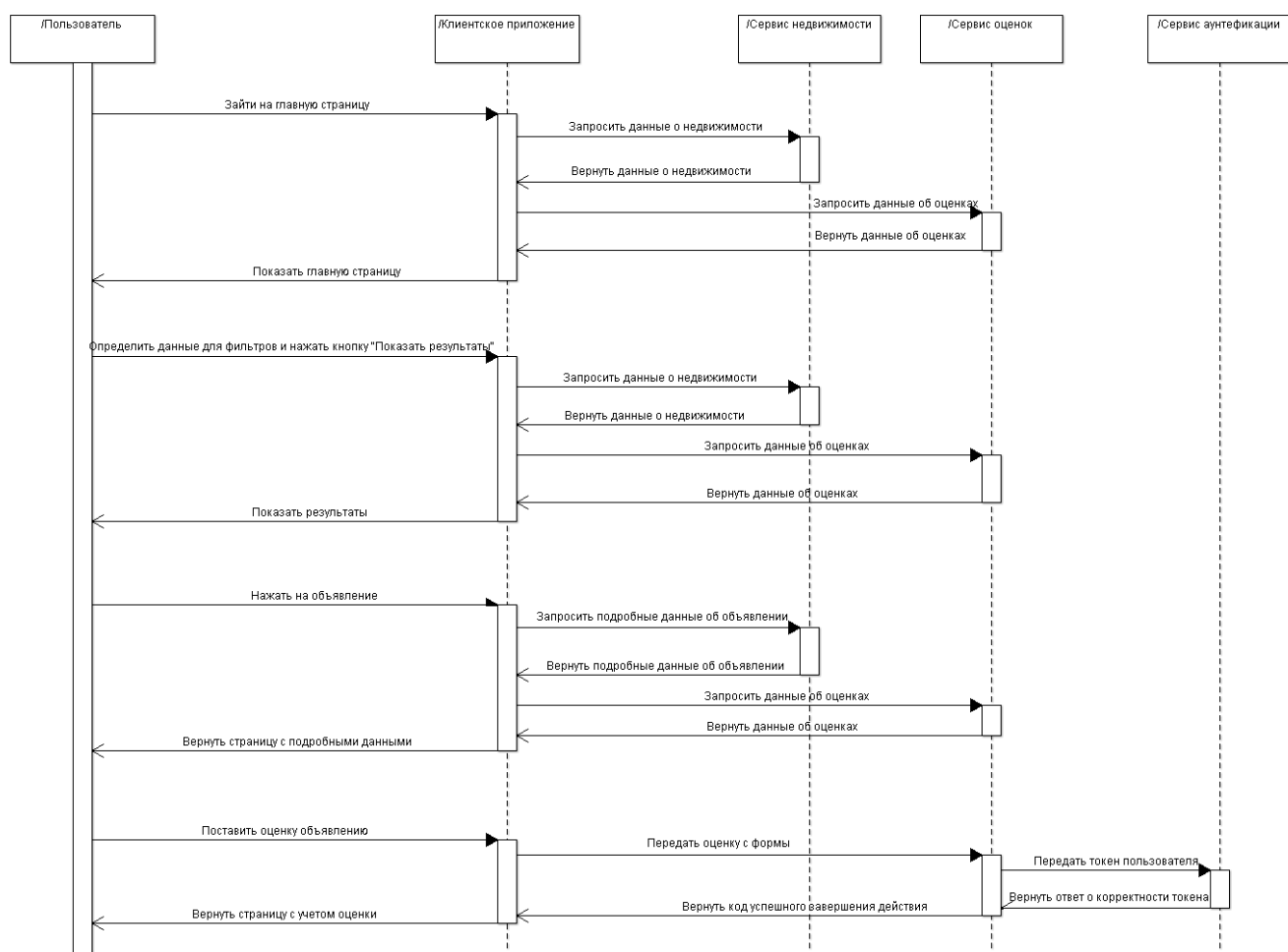


Рисунок 11 – Диаграмма последовательности для фильтрации и оценки объектов недвижимости

Когда пользователь заходит на главную страницу, приложение обращается к сервису недвижимости, получает необходимые данные, которые нужно вывести в представлении, и возвращает их пользователю.

Затем пользователь вводит данные в поля для фильтра и нажимает кнопку «Показать результаты», по аналогии со входом на главную страницу клиентское приложение обращается к сервису недвижимости и получает данные согласно фильтрам и возвращает их на экран.

Если пользователь хочет узнать подробности об объявлении, он нажимает на него и клиентской частью генерируется соответствующее представление.

И наконец, если пользователь хочет поставить оценку недвижимости, приложение обращается к сервису оценок, где проверяется, аутентифицирован ли пользователь и если да, то оценка успешно добавляется в базу данных, а пользователю возвращается представление с учетом выставленной им оценки.

3 Планирование и расчет бюджета разрабатываемой системы

Для разработки системы продажи и аренды недвижимости необходимо провести оценку затрат на различные компоненты проекта. Включенные в расчет аспекты включают в себя:

- затраты на API Яндекс.Карт – Так как разрабатываемое приложение связано с недвижимостью, необходимо иметь доступ к системе адресов, чтобы пользователь мог удобно выбирать адрес из доступных вариантов, а также видеть на карте тот адрес, который он просматривает;
- затраты на серверную инфраструктуру – Приложение должно работать бесперебойно, а для этого необходим сервер, который будет хранить данные и обеспечивать выполнение запросов;
- оплата труда персонала – для сайта должен быть предусмотрен персонал, отвечающий за администрирование и модерирование сайта;
- дополнительные расходы – в этот тип расходов включена регистрация доменного имени, приобретение SSL сертификата, лицензии на программное обеспечение а также расходы, связанные с продвижением сайта.

Для оценки затрат на разработку системы продажи и аренды недвижимости необходимо рассмотреть каждый аспект проекта и провести приблизительный расчет бюджета.

Затраты на API Яндекс.Карт составят около 22500 рублей в месяц. Это необходимо для обеспечения доступа к системе адресов и картам, что позволит пользователям удобно выбирать и просматривать адреса недвижимости. За год это составит 270000 рублей.

Серверная инфраструктура, включающая аренду виртуального сервера с базовыми характеристиками (8 GB RAM, 4 vCPUs, 160 GB SSD) и дополнительные услуги, обойдется в 6750 рублей в месяц. Это необходимо для обеспечения бесперебойной работы приложения, хранения данных и выполнения запросов. За год это составит 81000 рублей.

Для администрирования и модерирования сайта потребуется персонал. Системный администратор будет получать 50400 рублей в месяц, один модератор контента – 33600 рублей в месяц, а два разработчика – по 84000 рублей в месяц каждый. В итоге, расходы на персонал составят 252000 рублей в месяц, что равно 3024000 рублей в год.

Дополнительные расходы включают регистрацию доменного имени, приобретение SSL сертификата, лицензии на программное обеспечение и расходы на продвижение сайта. Регистрация доменного имени обойдется примерно в 900 рублей в год, SSL сертификат – 2250 рублей в год, лицензии на ПО – 9000 рублей в месяц, а продвижение сайта – 18000 рублей в месяц. В пересчете на год это составит 360000 рублей.

Общие годовые затраты составят: 270000 рублей за API Яндекс.Карт, 81000 рублей за серверную инфраструктуру, 3024000 рублей за оплату труда персонала и 360000 рублей за дополнительные расходы. В сумме это составляет 3735000 рублей в год.

4 Реализация и тестирование системы

4.1 Реализация структур программного средства

Для описания структур программного средства и того, как они взаимодействуют с пользователем было принято решение построить модель черного ящика.

Модель есть отображение свойств какого-либо объекта при его изучении. При моделировании свойства одного объекта переносятся на другой таким образом, чтобы взаимосвязь свойств модели и свойств изучаемого объекта была аналогичной. Отображения объектов называются моделями, а процесс их создания – моделированием. Модель «Черный ящик» – это система, в которой внешнему наблюдателю доступны лишь входные и выходные величины, а структура и внутренние процессы не известны.

Любая вещь, любой предмет, любое явление, любой познаваемый объект – всегда первоначально выступает как «черный ящик». Название модели «черный ящик» образно подчеркивает полное отсутствие сведений о внутреннем содержании «ящика»: в этой модели задаются, фиксируются, перечисляются только входные и выходные связи системы со средой. Графическая модель типа «черный ящик» отображает только связи системы со средой, в виде перечня «входов» и «выходов»

Модель типа «черный ящик», несмотря на внешнюю простоту и отсутствие сведений о внутренности системы, часто оказывается полезной. Во многих случаях эксплуатации системы достаточно содержательного словесного описания входов и выходов; тогда модель «черного ящика» является просто их списком [5].

На рисунке 12 представлена модель черного ящика для общего случая взаимодействия с клиентским приложением.



Рисунок 12 – Модель черного ящика для общего случая взаимодействия с клиентским приложением

В данной модели основными входами являются HTTP запросы от пользователей, данные, вводимые через пользовательский интерфейс, и API запросы от других сервисов. Выходами системы служат HTTP ответы пользователям, отображаемые данные на пользовательском интерфейсе и API ответы другим сервисам.

Пользователи взаимодействуют с системой через веб-интерфейс или мобильное приложение, отправляя HTTP запросы для выполнения различных операций, таких как создание, обновление или просмотр объявлений о продаже или аренде недвижимости. Эти запросы могут включать данные о цене, площади, местоположении и описании недвижимости. Другие сервисы также могут отправлять API запросы для получения данных или выполнения операций.

На выходе система генерирует HTTP ответы, возвращая пользователям результаты их запросов, такие как списки объявлений или подтверждения успешных операций. Эти ответы включают данные о недвижимости, результаты поиска и фильтрации, а также статусы запросов (например, 200 OK, 404 Not Found).

Отображаемые данные на пользовательском интерфейсе позволяют пользователям видеть актуальную информацию о доступных объектах

недвижимости. Кроме того, система отправляет API ответы другим сервисам, предоставляя необходимые данные для интеграции и взаимодействия с внешними системами. В нашем случае мы отправляем запросы к API ГеоСаджеста.

Таким образом, модель черного ящика демонстрирует, как микросервисное приложение принимает входные данные и преобразует их в выходные результаты, обеспечивая необходимую функциональность для пользователей и других сервисов.

Разрабатываемая система предназначена для создания, управления и просмотра объявлений о продаже и аренде недвижимости.

Основные функции системы включают создание объявлений пользователями, просмотр доступных объявлений, поиск объявлений по различным параметрам, а также управление объявлениями.

Система состоит из нескольких микросервисов, каждый из которых выполняет отдельную функцию. Микросервисы включают микросервис аутентификации и авторизации, микросервис управления объявлениями и микросервис оценок.

Приложение встречает пользователя страницей с формами для логина или регистрации пользователя. На этой странице пользователи могут ввести свои учетные данные для входа в систему или создать новую учетную запись.

Входными данными для данного процесса являются HTTP запросы, которые пользователи отправляют через браузер или мобильное приложение, содержащие информацию для логина или регистрации. Эти данные включают в себя такие поля, как имя пользователя, адрес электронной почты и пароль.

Микросервис логина и регистрации (рисунки 13, 14) принимает HTTP запросы на конечные точки `api/Login` и `api/Register`, возвращая HTTP ответы с кодами состояния и данными аутентификации.

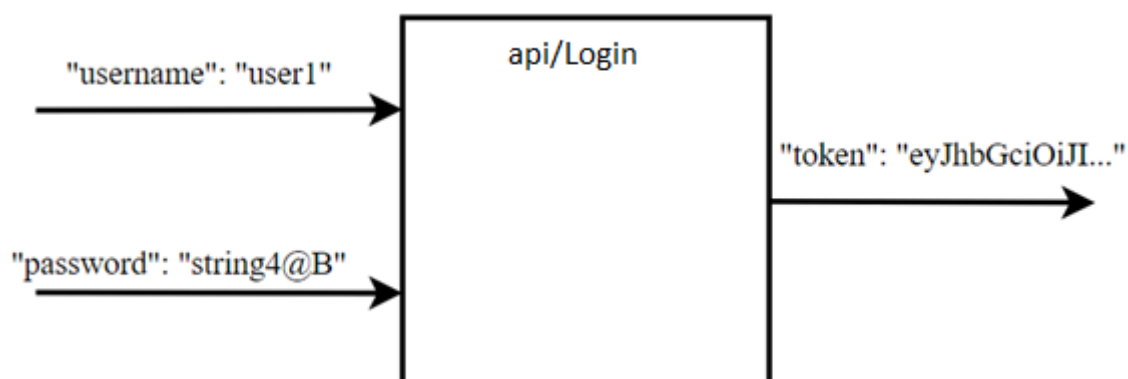


Рисунок 13 – Модель черного ящика для логина пользователя

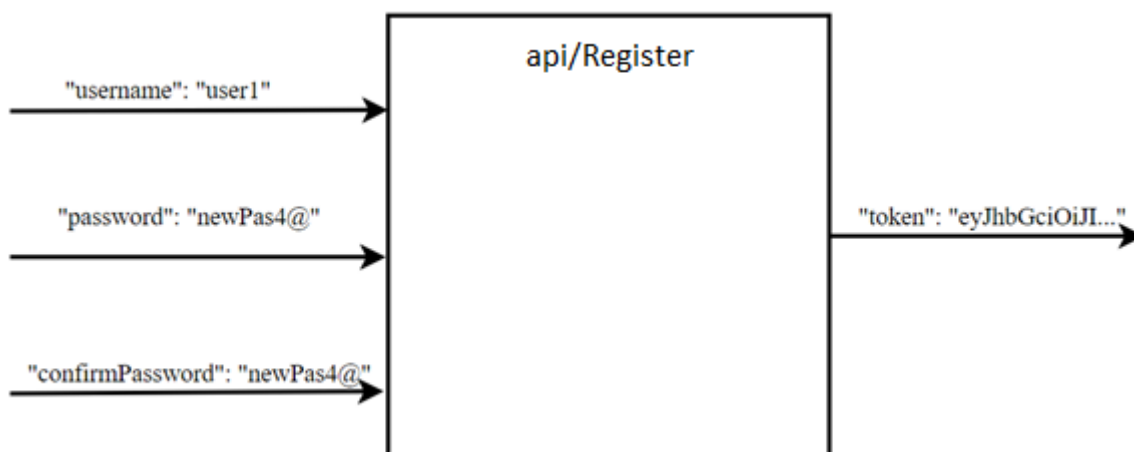


Рисунок 14 – Модель черного ящика для регистрации пользователя

Как видно из рисунка, и при логине и при регистрации пользователей при успешно введенных данных пользователю будет возвращен токен аутентификации, который позволит ему пользоваться возможностями сайта, требующими аутентификации, такими как добавление и удаление собственных объявлений, а также возможность ставить оценки чужим объявлениям.

После успешной регистрации пользователь перенаправляется на страницу с объявлениями, если пользователь хочет искать страницы по определенным критериям, он может воспользоваться формой фильтров, после внесения туда данных и нажатия кнопки «Показать результаты» пользователь получит обновленную страницу с объявлениями. С формы данные отправляются в контроллер клиентского приложения откуда передается запрос микросервису

недвижимости на конечную точку `api/Apartments/ApartmentsWithImage`, далее от микросервиса возвращается список недвижимости, отвечающий заданным критериям в формате JSON. Схема черного ящика для конечной точки `api/Apartments/ApartmentsWithImage` представлена на рисунке 15.

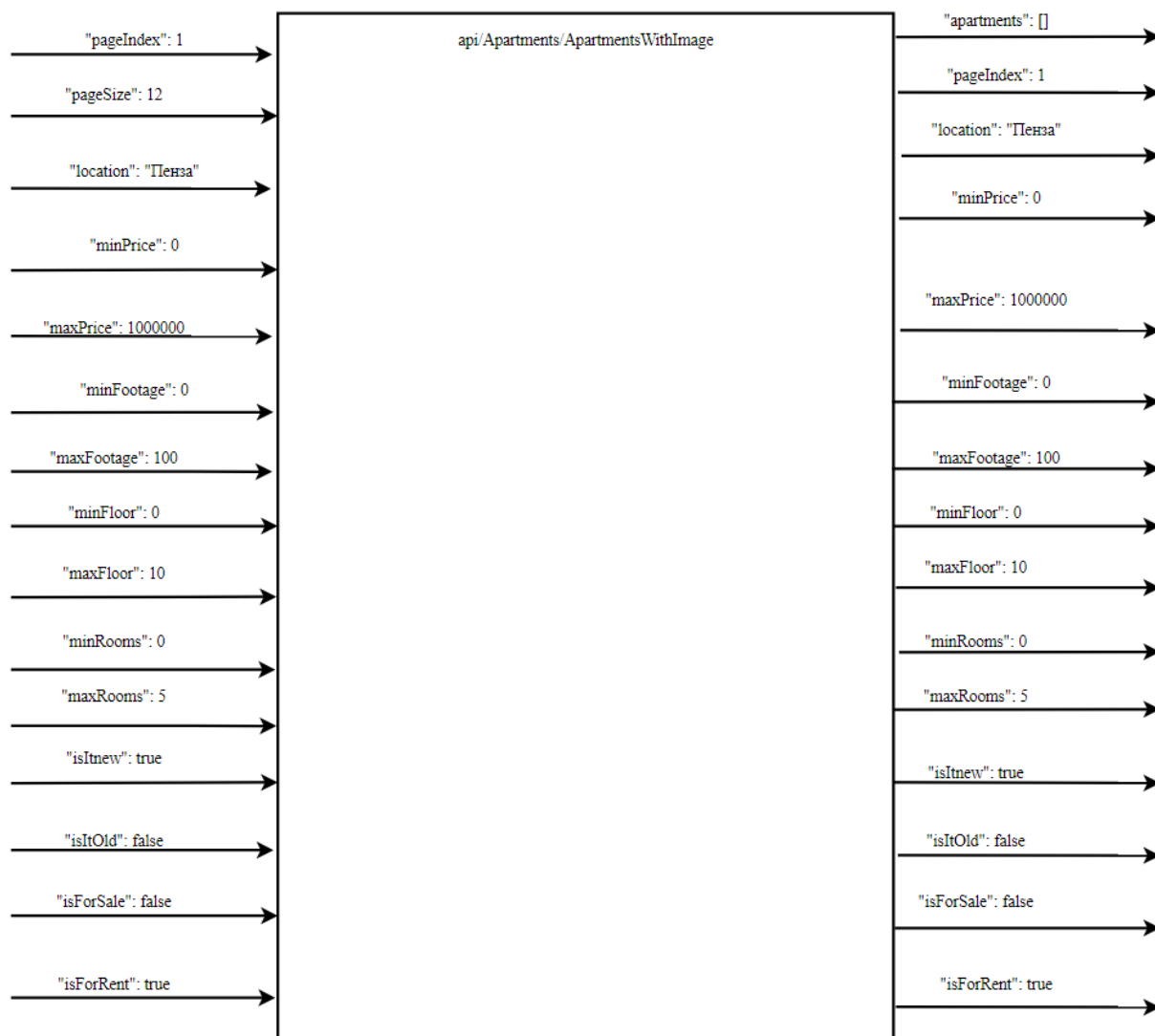


Рисунок 15 – Модель черного ящика для фильтров

Если пользователь аутентифицирован, то он может войти в личный кабинет для возможности управлять своими объявлениями. При попытке войти в личный кабинет токен добавляется в заголовок запроса, затем в запросе проверяется наличие и правильности токена пользователя, за проверку правильности токена отвечает микросервис авторизации, для этого клиентское

приложение посылает запрос на конечную точку `/api/Authentication/authenticate`. Модель черного ящика для проверки токена изображена на рисунке 16.



Рисунок 16 – Модель черного ящика для проверки токена

Если токен оказывается верным, пользователь попадает на страницу личного кабинета. В личном кабинете у пользователя есть возможность удалить свое объявление или добавить новое с помощью соответствующей формы. При нажатии на кнопку «Удалить» номер объявления передается в запросе через контроллер клиентского приложения микросервису недвижимости на конечную точку `api/Apartments/{id}`. При успешном удалении объявления микросервис возвращает ответ 204. Схема черного ящика для удаления объявления показана на рисунке 17.

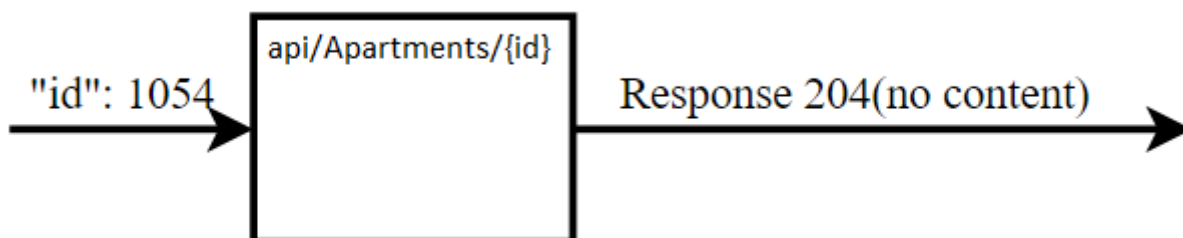


Рисунок 17 – Модель черного ящика для удаления объявления

При добавлении объявления конечная точка `/api/Apartments/postMany` микросервиса недвижимости возвращает сообщение о том, было ли добавлено

объявление без картинок, с картинками или, если расширение отправленных файлов не соответствует расширениям картинок, сообщение о неверном расширении файлов. Схема черного ящика для добавления объявления изображена на рисунке 18.

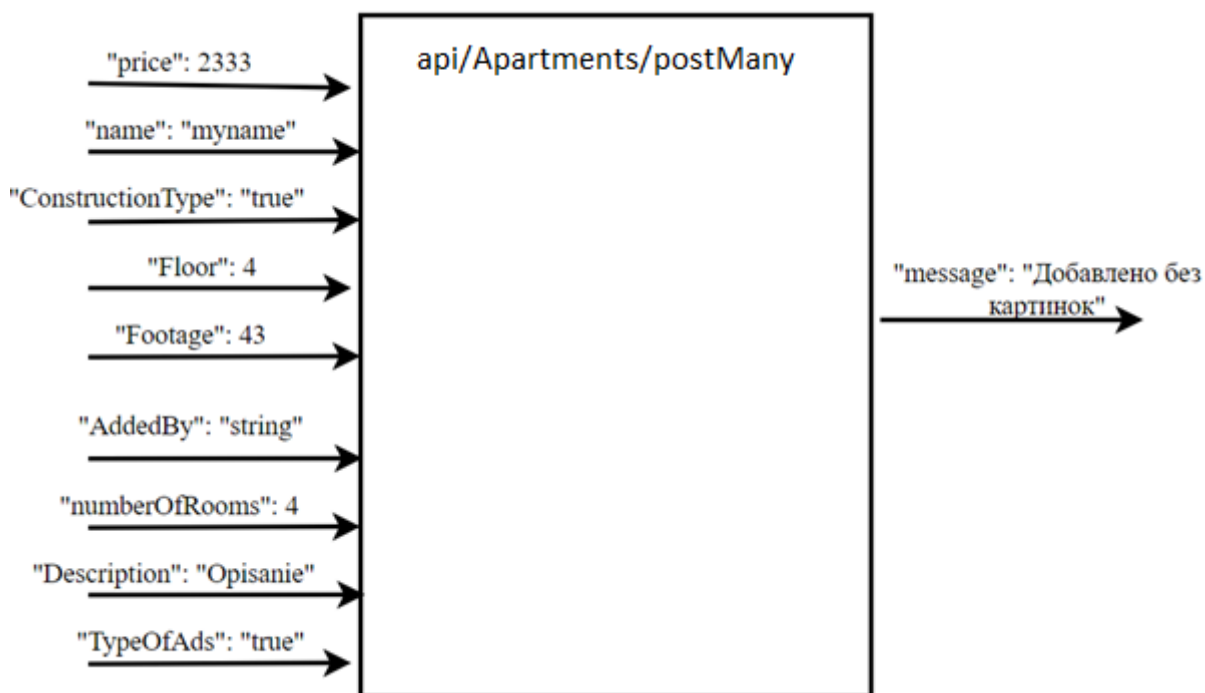


Рисунок 18 – Модель черного ящика для добавления объявления

Если пользователь хочет узнать более подробную информацию об объявлении, например посмотреть описание или увидеть больше картинок, он может нажать кнопку «Подробнее» на карточке объявления или в личном кабинете. При выполнении этого действия пользователь перенаправляется на страницу недвижимости. Если пользователь аутентифицирован и тип объявления обозначен как «Сдача», то пользователь имеет возможность поставить оценку объявлению. Оценка работает следующим образом: если пользователь еще не ставил оценку, то она будет добавлена, а на запрос вернется сообщение «New rate added», если же пользователь уже ставил оценку этому объявлению, то при повторном выставлении предыдущая оценка сменяется новой, и возвращается сообщение «Rate updated». Запрос на оценку передается

через клиентское приложение конечной точке /api/Rate микросервиса оценок. Схема черного ящика для конечной точки /api/Rate приведена на рисунке 19.

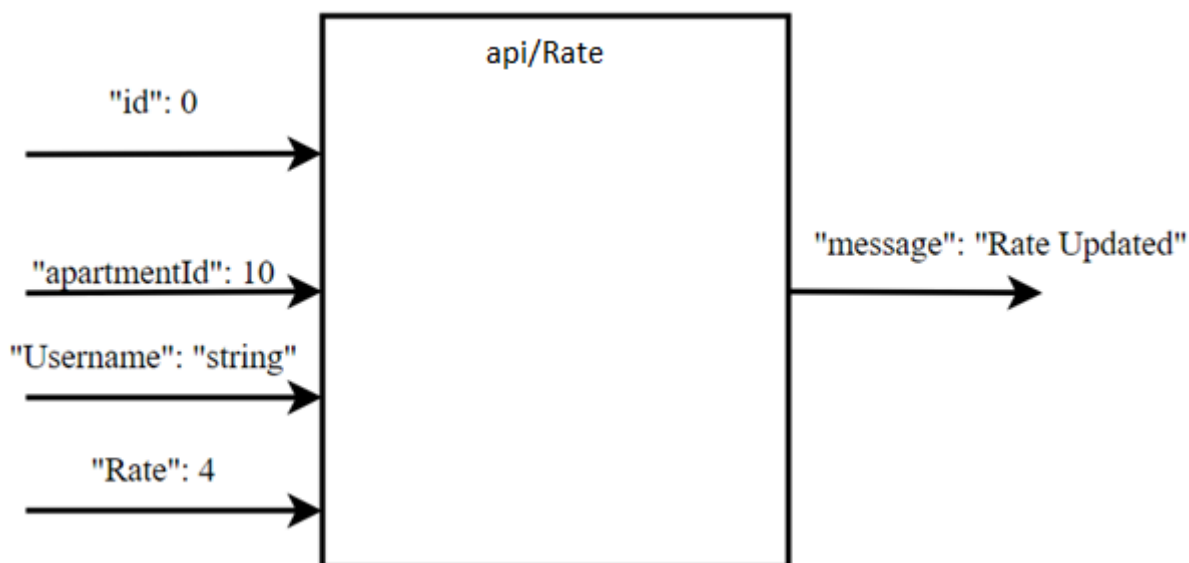


Рисунок 19 – Модель черного ящика для выставления оценки

Таким образом, использование модели черного ящика для описания микросервисного приложения позволяет детально рассмотреть взаимодействие системы с внешними компонентами без углубления во внутренние детали реализации. Это помогает лучше понять структуру и поведение системы, упростить её тестирование и сопровождение, а также обеспечить соответствие функциональных требований. Модель черного ящика показывает, как различные микросервисы взаимодействуют друг с другом и с внешними системами, что позволяет эффективно планировать и реализовывать улучшения в системе.

4.2 Реализация кода информационной системы

Приступая к разработке программного обеспечения, следует иметь в виду, что оно является большой и расширяемой системой, поэтому необходимо принять меры для его упрощения. Для этого ПО разрабатывают по частям, которые называются программными модулями. Программный модуль – это любой фрагмент описания процесса, оформляемый как самостоятельный программный продукт, пригодный для использования в описаниях процесса. Это

означает, что каждый программный модуль программируется, компилируется и отлаживается отдельно от других модулей программы, и, тем самым, физически разделен с другими модулями программы. Более того, каждый разработанный программный модуль может включаться в состав разных программ, если выполнены условия его использования, декларированные в документации по этому модулю. Таким образом, программный модуль может рассматриваться и как средство борьбы со сложностью программ, и как средство борьбы с дублированием в программировании [6].

Архитектура системы базируется на платформе ASP.NET и построена по принципу микросервисов. Основные компоненты системы разделены на несколько слоев, обеспечивающих гибкость, масштабируемость и удобство поддержки.

Клиентский слой отвечает за взаимодействие с пользователем. Включает в себя контроллеры, представления и модели данных, используемые для отображения информации о недвижимости

Слой логики приложения включает микросервисы, которые реализуют бизнес-логику и взаимодействуют с базами данных. Каждый микросервис представляет собой Web-API на платформе ASP.NET.

Слой данных включает в себя базы данных, в которых хранятся данные о пользователях, недвижимости и оценках. Базы данных реализованы на SQL Server.

Контроллер LogRegController выводит представление с формами логина и регистрации, а также реализует основные функции аутентификации: регистрацию, логин и выход пользователя из аккаунта. Он взаимодействует с внешним сервисом аутентификации через HTTP-запросы и использует сессии для хранения JWT-токенов и флагов состояния. Методы контроллера представлены в листингах 1 – 3.

Листинг 1 – Метод RegisterUser контроллера LogRegController

```

public async Task<IActionResult> RegisterUser(string username, string
email, string password, string confirmPassword)
{
    //TempData["Registered"] = false;
    var registrationModel = new { Username = username, Email = email,
    Password = password, ConfirmPassword = confirmPassword };
    var response = await
_httpClient.PostAsJsonAsync($"{_baseUrlService.authServiceBaseUrl}/api/r
egister", registrationModel);

    if (response.IsSuccessStatusCode)
    {
        var token = await response.Content.ReadAsStringAsync();

        HttpContext.Session.SetString("JWTToken", token);

        HttpContext.Session.SetString("registered", "true");

        return RedirectToAction("MainPage", "MainPage");
    }
    else
    {
        return BadRequest("Can't register user");
    }
}

```

Метод RegisterUser отвечает за регистрацию пользователя. Он отправляет POST-запрос на API регистрации с данными пользователя. Если регистрация прошла успешно, полученный JWT-токен сохраняется в сессии, и пользователь перенаправляется на главную страницу. В случае ошибки возвращается статус «BadRequest».

Листинг 2 – Метод LoginUser контроллера LogRegController

```

public async Task<IActionResult> LoginUser(string username, string
password)
{
    var loginModel = new { Username = username, Password = password };
    HttpResponseMessage response = await
_httpClient.PostAsJsonAsync($"{_baseUrlService.authServiceBaseUrl}/api/l
ogin", loginModel);
    if (response.IsSuccessStatusCode)
    {
        var token = await response.Content.ReadAsStringAsync();
        HttpContext.Session.SetString("JWTToken", token);

        HttpContext.Session.SetString("logged", "true");
        // Перенаправление на главную страницу
        return RedirectToAction("MainPage", "MainPage");
    }
}

```

```

else
{
    if (System.Net.HttpStatusCode.BadRequest == response.StatusCode)
    {
        return BadRequest("Invalid login or password");
    }
    else {
        return BadRequest("Probably, apartment service is off");
    }
}
}

```

Метод `LoginUser` отвечает за логин пользователя. Он отправляет POST-запрос на API логина с данными пользователя. Если логин прошел успешно, полученный JWT-токен сохраняется в сессии, и пользователь перенаправляется на главную страницу. В случае ошибки возвращается статус «`BadRequest`».

Листинг 3 – Метод `LogoutUser` контроллера `LogRegController`

```

public async Task<IActionResult> LogoutUser()
{
    HttpContext.Session.Clear();
    HttpContext.Session.SetString("logout", "true");
    return RedirectToAction("LogReg", "LogReg");
}

```

Метод `LogoutUser` отвечает за выход пользователя из аккаунта. Он очищает сессию и устанавливает флаг «`logout`» в сессии. После этого пользователь перенаправляется на страницу логина и регистрации.

`LogRegController` тесно взаимодействует с микросервисом аутентификации, состоящим из трех контроллеров: `LoginController`, `RegisterController` и `AuthenticationController`.

Метод `Login` отвечает за обработку POST-запросов на вход пользователя. Он также использует ASP.NET Core Identity для выполнения проверки учетных данных. Метод последовательно выполняет следующие задачи:

- проверяет валидность модели;
- аутентифицирует пользователя с помощью `SignInManager`’а;
- находит пользователя с помощью `UserManager`’а;
- генерирует токен для пользователя;

– возвращает токен в случае успешного логина или сообщение об ошибке в случае неудачи.

Код метода Login приведен в листинге 4.

Листинг 4 – Метод Login контроллера LoginController

```
[HttpPost]
public async Task<IActionResult> Login(LoginViewModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest("Invalid username or password");
    }

    var result = await
_signInManager.PasswordSignInAsync(model.Username, model.Password, false,
false);

    if (!result.Succeeded)
    {
        return BadRequest("Invalid username or password");
    }
    var user = await
_userManager.FindByNameAsync(model.Username);
    //Генерация токена для пользователя
    var token = _generateBearerToken.GenerateBearerToken(user);

    return Ok(new { Token = token });
}
```

Метод Register контроллера RegisterController работает аналогичным образом, за исключением того, что помимо выдачи токена аутентификации учетные данные нового пользователя добавляются в базу данных. Код метода Register приведен в листинге 5.

Листинг 5 – метод Register контроллера RegisterController

```
[HttpPost]
public async Task<IActionResult> Register(RegisterViewModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var user = new AppUser { UserName = model.Username, Email =
model.Email };

    var result = await _userManager.CreateAsync(user, model.Password);

    if (result.Succeeded)
```

```

    {
        //Генерация токена для пользователя
        var userName = await
        _userManager.FindByNameAsync(model.Username);
        var token = _generateBearerToken.GenerateBearerToken(userName);
        return Ok(new { Token = token });
    }
    else
    {
        return BadRequest(result.Errors);
    }
}

```

Оба метода, Login и Register, используют один и тот же метод генерации JWT-токена `GenerateBearerToken` (AppUser User) класса `GenerateBearerTokenService`, который добавлен в `ConfigureServices` класса `Startup` для удобного использования несколькими контроллерами сразу.

JSON Web Token (JWT) – это полезная нагрузка JSON, содержащая определенную заявку. Ключевое свойство JWT заключается в том, что для подтверждения их действительности нам нужно только взглянуть на сам токен.

JWT состоит из 3 частей: заголовка, полезной нагрузки и подписи.

Наибольшим преимуществом JWT по сравнению с управлением сеансами пользователей с использованием случайного токена в памяти является то, что они позволяют делегировать логику аутентификации стороннему серверу, который может быть как централизованным внутренним сервером аутентификации, так и полностью внешним сторонним поставщиком аутентификации, таким как, например, Auth0.

Внешний сервер аутентификации может быть полностью отделен от нашего сервера приложений и не должен делиться каким-либо секретным ключом с другими элементами сети – а именно с нашим сервером приложений. На нашем сервере не установлен секретный ключ, чтобы он не мог быть случайно потерян или украден. Кроме того, для работы аутентификации не требуется прямой связи между сервером аутентификации и сервером приложений [7].

Код генерации токена представлен в листинге 6.

Листинг 6 – метод GenerateBearerToken класса GenerateBearerTokenService

```
public string GenerateBearerToken(AppUser user)
{
    var tokenHandler = new JwtSecurityTokenHandler();
    var key = Encoding.ASCII.GetBytes(_jwtSecret["JWT:Key"]);

    //Создание описания токена, содержащего параметры и данные для
    создания JWT
    var tokenDescriptor = new SecurityTokenDescriptor
    {

        Audience = "your_audience",
        Issuer = «your_issuer»,
        //Создание идентификации (Subject) токена с утверждениями о
        пользователе
        Subject = new ClaimsIdentity(new Claim[]
        {
            //Добавление утверждения о имени пользователя
            new Claim(ClaimTypes.Name, user.UserName),

            //Добавление утверждения об успешной аутентификации
            new Claim(ClaimTypes.Authentication, «true»),
        }),
        //Установка срока действия токена (7 дней с текущего момента)
        Expires = DateTime.UtcNow.AddDays(7),

        //Установка параметров подписи токена с использованием
        симметричного ключа HMAC-SHA256
        SigningCredentials = new SigningCredentials(new
        SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };
    //Создание JWT-токена на основе описания токена
    var token = tokenHandler.CreateToken(tokenDescriptor);
    //Преобразование JWT-токена в строку
    return tokenHandler.WriteToken(token);
}
```

Данный метод создает JWT (JSON Web Token) для аутентификации пользователя. Он принимает объект AppUser, представляющий пользователя, и использует секретный ключ для подписи токена. Метод создает объект JwtSecurityTokenHandler, который отвечает за создание и управление JWT. Секретный ключ берется из конфигурации приложения. Затем метод создает описание токена, включающее в себя параметры аудитории, издателя, утверждения о пользователе (такие как имя пользователя и факт успешной аутентификации), срок действия токена и параметры подписи, использующие симметричный ключ и алгоритм HMAC-SHA256. После этого метод создает

токен на основе описания с помощью `CreateToken` и преобразует его в строку с помощью `WriteToken`, возвращая эту строку.

Помимо контроллеров для логина и регистрации, микросервис `AuthService` содержит контроллер `AuthenticationController`, который используется в большом количестве методов клиентского приложения, а также другими микросервисами для проверки аутентификации пользователя. Контроллер состоит всего из одного метода `Authenticate()`, который выполняет проверку аутентификации пользователя и возвращает его имя, если он аутентифицирован. Он использует атрибут `HttpGet`, чтобы быть доступным по GET-запросу с маршрутом «authenticate». В методе проверяется, аутентифицирован ли текущий пользователь с помощью свойства `User.Identity.IsAuthenticated`. Если пользователь аутентифицирован, метод извлекает его имя с помощью свойства `User.Identity.Name` и возвращает ответ с кодом 200 (OK), содержащий имя пользователя в виде JSON. Если пользователь не аутентифицирован, метод возвращает ответ с кодом 400 (Bad Request). Код метода приведен в листинге 7.

Листинг 7 – Метод `Authenticate` контроллера `AuthenticationController`

```
[HttpGet («authenticate»)]
public IActionResult Authenticate()
{
    if (User.Identity.IsAuthenticated)
    {
        var userName = User.Identity.Name;

        return Ok(new { user = userName });
    }
    else { return BadRequest(); }
}
```

После успешной аутентификации пользователя его перенаправляет на главную страницу сайта, за представление данной страницы отвечает метод `MainPage` контроллера `MainPageController`, код которого приведен в листинге 8.

Листинг 8 – Метод `MainPage` контроллера `MainPageController`

```
public async Task<IActionResult> MainPage(
    string location,
    int page = 1,
    int pageSize = 12,
```

```

int minFootage = 0,
int maxFootage = int.MaxValue,
int minPrice = 0,
int maxPrice = int.MaxValue,
int minFloor = 0,
int maxFloor = int.MaxValue,
bool isItNew = false,
bool isItOld = false,
bool isForSale = false,
bool isForRent = false,
int minRooms = 0,
int maxRooms = int.MaxValue)
{
    var queryParameters = new List<string>
    {
        $"page={page}",
        $"pageSize={pageSize}",
        $"minPrice={minPrice}",
        $"maxPrice={maxPrice}",
        $"minFootage={minFootage}",
        $"maxFootage={maxFootage}",
        $"minFloor={minFloor}",
        $"maxFloor={maxFloor}",
        $"isItNew={isItNew}",
        $"isItOld={isItOld}",
        $"isForSale={isForSale}",
        $"isForRent={isForRent}",
        $"minRooms={minRooms}",
        $"maxRooms={maxRooms}"
    };
    if (!string.IsNullOrEmpty(location))
    {
        queryParameters.Add($"location={location}");
    }
    var queryString = string.Join("&", queryParameters);
    HttpResponseMessage responseForApartments = await
_httpClient.GetAsync($"({_baseUrlService.apartmentServiceBaseUrl}/api/Apa
rtments/apartmentsWithImage?{queryString}");
    if (!responseForApartments.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить объявления");
    }
    HttpResponseMessage responseForApartmentsAmount = await
_httpClient.GetAsync($"({_baseUrlService.apartmentServiceBaseUrl}/api/Apa
rtments/AmountOfApartments?{queryString}");
    if (!responseForApartmentsAmount.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить данные о количестве
объявлений");
    }
    HttpResponseMessage responseForRates = await
_httpClient.GetAsync($"({_baseUrlService.rateServiceBaseUrl}/api/rate/Avg
Rates");
    if (!responseForRates.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить оценки объявлений");
    }
}

```

```

    }
    var jsonStringForApartments = await
responseForApartments.Content.ReadAsStringAsync();
    var apartments =
JsonConvert.DeserializeObject<ApartmentsWithImagesAndPage>(jsonStringForA
partments);

    var jsonStringForRates = await
responseForRates.Content.ReadAsStringAsync();
    var rate =
JsonConvert.DeserializeObject<List<AvgRate>>(jsonStringForRates);
    var jsonStringForApartmentsAmount = await
responseForApartmentsAmount.Content.ReadAsStringAsync();
    var apartmentsAmount =
JsonConvert.DeserializeObject<int>(jsonStringForApartmentsAmount);
    var tokenString = HttpContext.Session.GetString("JWTToken");
    string strUser = "";
    if (tokenString != null)
    {
        var jwtToken = JObject.Parse(tokenString);
        jwtToken.TryGetValue("token", out var token);
        var tokenValue = token.ToString();
        _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);
        HttpResponseMessage responseForAuth = await
_httpClient.GetAsync($"{_baseUrlService.authServiceBaseUrl}/api/authenti
cation/authenticate");
        if (responseForAuth.IsSuccessStatusCode)
        {
            var userRaw =
responseForAuth.Content.ReadAsStringAsync().Result;
            var JobjUser = JObject.Parse(userRaw);
            strUser = JobjUser["user"].ToString();
        }
    }
    var model = new ApartmentDataViewModel
    {
        User = new User
        {
            UserName = strUser
        },
        Apartments = apartments,
        AvgRate = rate,
        NumberOfApartments = apartmentsAmount,
        Location = location,
        CurrentPage = page,
        MinPrice = minPrice,
        MaxPrice = maxPrice,
        MinFloor = minFloor,
        MaxFloor = maxFloor,
        MinFootage = minFootage,
        MaxFootage = maxFootage,
        MinRooms = minRooms,
        MaxRooms = maxRooms,
        IsForSale = isForSale,
        IsForRent = isForRent,
    }
}

```

```

        IsItNew = isItNew,
        IsItOld = isItOld
    };
    return View(model);
}
}

```

Данный метод получает параметры запроса, такие как размер страницы, номер страницы и критерии фильтрации недвижимости. Затем он отправляет несколько запросов к удаленным сервисам для получения данных о недвижимости, оценках и общем количестве объектов. Полученные данные обрабатываются и преобразуются в объекты модели представления. Метод также проверяет аутентификацию пользователя и формирует модель представления с информацией об объектах недвижимости, оценках, пользователе и параметрах фильтрации. В конце метод возвращает представление с сформированной моделью для отображения на странице.

Данный сервис активно обращается к удаленным сервисам для получения данных, которые необходимо вывести в представление, в частности он обращается к микросервису ApartmentService для получения данных о недвижимости. Вкратце опишем его методы.

Метод GetApartments() обрабатывает HTTP GET-запрос для получения списка недвижимости. Он использует асинхронный подход для выполнения запроса к базе данных. Затем он возвращает список объектов в виде коллекции IEnumerable<Apartment>. _context представляет доступ к базе данных, а метод ToListAsync() используется для выполнения асинхронного запроса и получения списка объектов недвижимости из базы данных. Код метода представлен в листинге 9.

Листинг 9 – метод GetApartments контроллера ApartmentsController

```

[HttpGet]
public async Task<ActionResult<IEnumerable<Apartment>>>
GetApartments()
{
    return await _context.Apartments.ToListAsync();
}

```

Данный метод позволяет получить только текстовые данные о недвижимости и не позволяет фильтровать возвращаемые данные, для получения отфильтрованных данных вместе с изображениями предназначен метод `GetApartmentsWithImages`.

Данный метод обрабатывает HTTP GET-запрос для получения списка объектов с изображениями и дополнительными фильтрами. Он принимает параметры запроса, такие как местоположение, номер страницы, размер страницы, минимальная и максимальная площадь, цена, этаж, количество комнат и другие. Затем метод выполняет запрос к базе данных, используя LINQ-запрос для фильтрации объектов недвижимости в соответствии с переданными параметрами.

Результаты запроса фильтруются и проецируются на новый тип данных `ApartmentWithImage`, который содержит информацию о недвижимости и путь к изображению. Затем изображения считываются из файловой системы, преобразуются в формат `base64` и добавляются к соответствующим объектам недвижимости.

Наконец, результаты запроса упаковываются в объект `ApartmentsWithImagesAndPageAndFilters`, который содержит список объектов с изображениями, информацию о текущей странице и значения фильтров. Возвращается ответ `Ok` с этими данными в формате `JSON`.

Код метода `GetApartmentsWithImages` представлен в листинге 10.

Листинг 10 – Метод `GetApartmentsWithImages` контроллера

`ApartmentsController`

```
[HttpGet("ApartmentsWithImage")]
public async
Task<ActionResult<IEnumerable<ApartmentsWithImagesAndPageAndFilters>>>
GetApartmentsWithImages(
    string location = "",
    int page = 1,
    int pageSize = 12,
    int minFootage = 0,
    int maxFootage = int.MaxValue,
    int minPrice = 0,
```

```

int maxPrice = int.MaxValue,
int minFloor = 0,
int maxFloor = int.MaxValue,
bool isItNew = false,
bool isItOld = false,
bool isForSale = false,
bool isForRent = false,
int minRooms = 0,
int maxRooms = int.MaxValue)
{
    var query = _contextApartments
.Where(apartment =>
    apartment.Name.Contains(location) &&
    apartment.Footage >= minFootage &&
    apartment.Footage <= maxFootage &&
    apartment.Price >= minPrice &&
    apartment.Price <= maxPrice &&
    apartment.Floor >= minFloor &&
    apartment.Floor <= maxFloor &&
    (!isItNew || apartment.ConstructionType) &&
    (!isItOld || !apartment.ConstructionType) &&
    (!isForSale || apartment.TypeOfAds) &&
    (!isForRent || !apartment.TypeOfAds) &&
    apartment.NumberOfRooms >= minRooms &&
    apartment.NumberOfRooms <= maxRooms);
    var result = query
        .Select(apartment => new ApartmentWithImage
        {
            Id = apartment.Id,
            Name = apartment.Name,
            Description = apartment.Description,
            AddedBy = apartment.AddedBy,
            Footage = apartment.Footage,
            TypeOfAds = apartment.TypeOfAds,
            Price = apartment.Price,
            DateOfPost = apartment.DateOfPost,
            Floor = apartment.Floor,
            ConstructionType = apartment.ConstructionType,
            NumberOfRooms = apartment.NumberOfRooms,
            ImagePath = _contextApartmentsImage
                .Where(image => image.ApartmentId == apartment.Id)
                .Select(image => image.ImagePath)
                .FirstOrDefault(),
            ImageData = ""
        })
        .Skip((page - 1) * pageSize)
        .Take(pageSize)
        .ToList();

    foreach (var item in result)
    {
        if (item.ImagePath != null)
        {
            var imageBytes = await
System.IO.File.ReadAllBytesAsync(item.ImagePath);
            var imageBase64 = Convert.ToBase64String(imageBytes);

```

```

        item.ImageData = $"data:image/jpeg;base64,{imageBase64}";
    }
}
var apartmentsAndPage = new ApartmentsWithImagesAndPageAndFilters
{
    Apartments = result,
    PageIndex = page,
    Location = location,
    MinPrice = minPrice,
    MaxPrice = maxPrice,
    MinFloor = minFloor,
    MaxFloor = maxFloor,
    MinFootage = minFootage,
    MaxFootage = maxFootage,
    IsItOld = isItOld,
    IsItNew = isItNew,
    IsForRent = isForRent,
    IsForSale = isForSale,
    MinRooms = minRooms,
    MaxRooms = maxRooms,
};

return Ok(apartmentsAndPage);
}

```

Метод `GetAmount` предназначен для получения количества объектов недвижимости по заданным критериям, что используется в представлении главной страницы сайта для отображения количества найденных с помощью формы фильтрации объектов недвижимости. Код метода представлен в листинге 11.

Листинг 11 – метод `GetAmount` контроллера `ApartmentsController`

```

[HttpGet («AmountOfApartments»)]
public async Task<ActionResult<int>> GetAmount(
    string location = "",
    int minFootage = 0,
    int maxFootage = int.MaxValue,
    int minPrice = 0,
    int maxPrice = int.MaxValue,
    int minFloor = 0,
    int maxFloor = int.MaxValue,
    bool isItNew = false,
    bool isItOld = false,
    bool isForSale = false,
    bool isForRent = false,
    int minRooms = 0,
    int maxRooms = int.MaxValue)
{
    var query = _context.Apartments
        .Where(apartment =>
            apartment.Name.Contains(location) &&

```

```

        apartment.Footage >= minFootage &&
        apartment.Footage <= maxFootage &&
        apartment.Price >= minPrice &&
        apartment.Price <= maxPrice &&
        apartment.Floor >= minFloor &&
        apartment.Floor <= maxFloor &&
        (!isItNew || apartment.ConstructionType) &&
        (!isItOld || !apartment.ConstructionType) &&
        (!isForSale || apartment.TypeOfAds) &&
        (!isForRent || !apartment.TypeOfAds) &&
        apartment.NumberOfRooms >= minRooms &&
        apartment.NumberOfRooms <= maxRooms)
        .Count();
        return Ok(query);
    }

```

Для добавления объявлений и изображений к ним служит метод `NewPost`. Данный метод обрабатывает `POST` запрос, который принимает список файлов и данные апартаментов. Он начинается с извлечения аутентификационного токена из заголовка запроса. Если токен отсутствует, возвращается ошибка «Отсутствует токен».

Далее извлекается токен из строки «Bearer». Затем происходит запрос к службе аутентификации для проверки токена. Если запрос не проходит успешно, возвращается ошибка «Токен не соответствует».

После успешной аутентификации из ответа получается информация о пользователе. Если пользователь, указанный в апартаментах, не совпадает с текущим пользователем, возвращается ошибка «Отказано в доступе».

Затем проверяется наличие загружаемых файлов. Если файлы отсутствуют, данные апартаментов сохраняются без изображений.

Если файлы присутствуют, происходит сохранение информации об апартаментах и загрузка каждого файла на сервер. Проверяется расширение каждого файла, и если оно допустимо, файл сохраняется на сервере. После сохранения всех файлов возвращается сообщение об успешном добавлении апартаментов с файлами.

Код метода `NewPost` приведен в листинге 12.

Листинг 12 – метод `NewPost` контроллера `ApartmentsController`

```
[HttpPost("postMany")]
```

```

public async Task<ActionResult<Apartment>> NewPost(List<IformFile>
files, [FromForm] Apartment apartment)
{
    var strUser = "";
    var token = Request.Headers["Authorization"].FirstOrDefault();
    if (string.IsNullOrEmpty(token))
    {
        return Unauthorized("Token is missing");
    }
    var tokenWithoutBearer = token.Substring(7);
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenWithoutBearer);
    HttpResponseMessage response = await
_httpClient.GetAsync($"{_authServiceBaseUrl}/api/authentication/authentic
ate");
    if (!response.IsSuccessStatusCode)
    {
        return Unauthorized(new {message = "Токен не соответствует"});
    }
    var userRaw = response.Content.ReadAsStringAsync().Result;
    var JobjUser = JObject.Parse(userRaw);
    strUser = JobjUser["user"].ToString();
    if (strUser != apartment.AddedBy)
    {
        return Unauthorized(new {message = "Отказано в доступе"});
    }
    if (files == null || files.Count == 0)
    {
        //Сохранение информации об апартаменте в базе данных
        _context.Apartments.Add(apartment);
        await _context.SaveChangesAsync();
        return Ok(new { message = «Добавлено без картинок» });
    }
    else
    {
        //Сохранение информации об апартаменте в базе данных
        _context.Apartments.Add(apartment);

        await _context.SaveChangesAsync();
        //OrdinalIgnoreCase позволяет проводить проверку без учета
регистра
        var allowedExtensions = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)
        {
            ".jpg",
            ".jpeg",
            ".png",
            ".gif",
            ".bmp"
        };
        foreach (var file in files)
        {
            if (file.Length > 0)
            {
                //Проверка на расширение файла
                string ext = Path.GetExtension(file.FileName);

```

```

        if (!allowedExtensions.Contains(ext))
        {
            return BadRequest(new { message = "Неверное
расширение файла" });
        }
        //Генерация уникального имени файла
        var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName);
        //Путь для сохранения файла
        var filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Images", fileName);
        //Сохранение файла на сервере
        using (var stream = new FileStream(filePath,
FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }
        //Сохранение информации о файле в базу данных
        var apartmentImage = new ApartmentImages
        {
            ApartmentId = apartment.Id,
            ImagePath = filePath
        };
        _context.ApartmentsImage.Add(apartmentImage);
    }
}
await _context.SaveChangesAsync();
return Ok(new { message = "Добавлено с файлами" });
}
}

```

Для получения всех изображений для недвижимости, содержащей файл с заданным путем служит метод `GetApartmentImagesByFilePath`. Данный метод обрабатывает GET запрос для получения изображений апартаментов по их пути на сервере. Он принимает строковый параметр `filePath`, который представляет путь к изображению.

Сначала метод ищет запись о картинке в базе данных с соответствующим путем файла. Если запись не найдена, возвращается ошибка «Изображение с указанным путем не найдено».

Далее метод получает `ApartmentId` из найденной записи и ищет все записи о картинках для этого `ApartmentId`. Если записи не найдены, возвращается ошибка «Изображения для данного `ApartmentId` не найдены».

После получения всех путей к изображениям он загружает каждое изображение из файла, добавляет его в список и возвращает список байтовых массивов, представляющих изображения, в ответе. Если происходит ошибка при чтении файла, возвращается сообщение о ошибке чтения файла изображения.

Код метода `GetApartmentImagesByFilePath` представлен в листинге 13.

Листинг 13 – метод `GetApartmentImagesByFilePath` контроллера

`ApartmentsController`

```
[HttpGet("GetApartmentImagesByFilePath")]
public async Task<ActionResult<IEnumerable<ApartmentImages>>>
GetApartmentImagesByFilePath(string filePath)
{
    // Найти запись ApartmentImage по пути файла
    var apartmentImage = await
_context.ApartmentsImage.FirstOrDefaultAsync(ai => ai.ImagePath ==
filePath);
    if (apartmentImage == null)
    {
        return NotFound(«Изображение с указанным путем не найдено»);
    }
    // Получить ApartmentId из найденной записи ApartmentImage
    var apartmentId = apartmentImage.ApartmentId;
    // Найти все записи ApartmentImage, соответствующие этому ApartmentId
    var apartmentImages = await _context.ApartmentsImage
        .Where(ai => ai.ApartmentId == apartmentId)
        .ToListAsync();
    if (apartmentImages == null || apartmentImages.Count == 0)
    {
        return NotFound(«Изображения для данного ApartmentId не
найжены»);
    }
    // Создать список путей к изображениям
    var imagePaths = apartmentImages.Select(ai => ai.ImagePath);
    // Создать список для хранения изображений
    var images = new List<byte[]>();
    // Загрузить изображения из файлов и добавить их в список
    foreach (var path in imagePaths)
    {
        try
        {
            var imageBytes = await
System.IO.File.ReadAllBytesAsync(path);
            images.Add(imageBytes);
        }
        catch (Exception ex)
        {
            // Обработка ошибок чтения файла
            // Замените этот блок кодом, соответствующим вашим
потребностям
        }
    }
}
```

```

        return BadRequest($"»Ошибка чтения файла изображения:
{ex.Message}»);
    }
}
return Ok(images);
}

```

Для получения объекта недвижимости по его идентификатору предназначен метод `GetApartment`. Сначала данный метод пытается найти недвижимость с указанным идентификатором в базе данных. Если недвижимость не найдена, возвращается ответ с кодом статуса 404 (Not Found). Если найдена – она возвращается в качестве результата запроса с кодом статуса 200 (OK). Код метода представлен в листинге 14.

Листинг 14 – Метод `GetApartment` контроллера `ApartmentsController`

```

[HttpGet («{id}»)]
public async Task<ActionResult<Apartment>> GetApartment(int id)
{
    var apartment = await _context.Apartments.FindAsync(id);
    if (apartment == null)
    {
        return NotFound();
    }
    return apartment;
}

```

Для получения всех объектов недвижимости, добавленных определенным пользователем используется метод `GetApartmentBy`. Данный метод обрабатывает GET запрос для получения списка апартаментов, добавленных указанным пользователем `addedBy`. Он принимает имя пользователя в качестве параметра `user`.

Метод сначала выбирает все апартаменты из базы данных и проецирует их на объекты типа `ApartmentWithImage`, включая основные данные об апартаменте и путь к изображению, если таковое имеется.

Затем метод фильтрует список апартаментов по указанному пользователю `user`, выбирая только те, которые были добавлены этим пользователем.

Наконец, список отфильтрованных апартаментов возвращается в качестве результата запроса с кодом статуса 200 (OK).

Код метода `GetApartmentBy` представлен в листинге 15.

Листинг 15 – Метод GetApartmentBy контроллера ApartmentsController

```
[HttpGet («GetByAddedBy{addedBy}»)]
public async Task<ActionResult<ApartmentWithImage>> GetApartmentBy(string
user)
{
    var apartments = await _context.Apartments.
        Select(apartment => new ApartmentWithImage
        {
            Id = apartment.Id,
            Name = apartment.Name,
            Description = apartment.Description,
            AddedBy = apartment.AddedBy,
            Footage = apartment.Footage,
            TypeOfAds = apartment.TypeOfAds,
            Price = apartment.Price,
            DateOfPost = apartment.DateOfPost,
            Floor = apartment.Floor,
            ConstructionType = apartment.ConstructionType,
            NumberOfRooms = apartment.NumberOfRooms,
            ImagePath = _context.ApartmentsImage
                .Where(image => image.ApartmentId == apartment.Id)
                .Select(image => image.ImagePath)
                .FirstOrDefault(),
            ImageData = ""
        })
        .Where(a => a.AddedBy == user)
        .ToListAsync();
    return Ok(apartments);
}
```

Для удаления объектов недвижимости используется метод DeleteApartment. Этот метод обрабатывает HTTP DELETE запрос для удаления апартаментов с указанным идентификатором.

Сначала он проверяет наличие токена авторизации в заголовке запроса. Если токен отсутствует, метод возвращает код статуса 401 (Unauthorized) с сообщением «Token is missing».

Затем он извлекает имя пользователя из токена авторизации и проверяет его наличие и корректность. Если токен успешно прошел аутентификацию, метод извлекает имя пользователя из ответа.

Далее метод проверяет, добавлен ли апартамент пользователем, который пытается его удалить. Если да, то выполняется попытка удалить апартамент из базы данных. Если апартамент не найден, возвращается код статуса 400 (BadRequest) с сообщением «Apartment not found».

Если операция удаления прошла успешно, метод возвращает код статуса 204 (No Content), указывая, что операция прошла успешно, но не возвращает тело ответа.

Если пользователь, пытающийся удалить апартамент, не соответствует тому, кто его добавил, метод возвращает код статуса 401 (Unauthorized) с сообщением «Отказано в доступе».

Код метода представлен в листинге 16.

Листинг 16 – метод DeleteApartment контроллера ApartmentsController

```
[HttpDelete("{id}")]
public async Task<ActionResult> DeleteApartment(int id)
{
    var strUser = "";
    var token = Request.Headers["Authorization"].FirstOrDefault();
    if (string.IsNullOrEmpty(token))
    {
        return Unauthorized("Token is missing");
    }
    var tokenWithoutBearer = token.Substring(7);
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenWithoutBearer);
    HttpResponseMessage response = await
_httpClient.GetAsync($"{_authServiceBaseUrl}/api/authentication/authentic
ate");
    if (response.IsSuccessStatusCode)
    {
        var userRaw = response.Content.ReadAsStringAsync().Result;
        var JobjUser = JObject.Parse(userRaw);
        strUser = JobjUser[«user»].ToString();
        //Тут необходимо включить проверку токена и удаляемого значения
        var addedBy = _context.Apartments.Where(u=> u.Id ==
id).FirstOrDefault();
        if (addedBy.AddedBy == strUser)
        {
            var apartment = await _context.Apartments.FindAsync(id);
            if (apartment == null)
            {
                return BadRequest("Apartment not found");
            }
            _context.Apartments.Remove(apartment);
            await _context.SaveChangesAsync();
            //Возвращает ответ 204 No Content, указывая, что операция
прошла успешно, но не возвращает тело ответа.
            Return NoContent();
        }
        else
        {
            return Unauthorized(new {message = "Отказано в доступе"});
        }
    }
}
```

```

    }
}
else
{
    return Unauthorized(new {message = "Токен не соответствует"});
}
}

```

Попав на главную страницу сайта, пользователь может воспользоваться фильтрами для поиска, после нажатия на кнопку «Показать результаты» пользователь перенаправляется на ту же страницу, но с обновленными данными, то же самое происходит при нажатии на страницы.

Помимо использования фильтров и переключения страниц пользователь также может нажать на кнопку «Подробнее» карточки товара, в таком случае пользователя перенаправит на страницу объекта недвижимости, за вывод данных в представление страницы объекта отвечает единственный метод `ApartmentPage` контроллера `ApartmentPageController` клиентского приложения. Код метода представлен в листинге 17.

Листинг 17 – Метод `ApartmentPage` контроллера `ApartmentPageController`

```

public async Task<IActionResult> ApartmentPage(int id, string imagePath)
{
    HttpClient httpClient = new HttpClient();
    HttpResponseMessage responseForRates = await
httpClient.GetAsync($"({_baseUrlService.rateServiceBaseUrl})/api/rate/AvgRates");
    if (!responseForRates.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить оценки объявлений");
    }
    var jsonStringForRates = await
responseForRates.Content.ReadAsStringAsync();
    var rate =
JsonConvert.DeserializeObject<List<AvgRate>>(jsonStringForRates);
    HttpResponseMessage responseForApartment = await
httpClient.GetAsync($"({_baseUrlService.apartmentServiceBaseUrl})/api/Apartments/{id}");
    var jsonResponseForApartment = await
responseForApartment.Content.ReadAsStringAsync();
    var apartment =
JsonConvert.DeserializeObject<Apartment>(jsonResponseForApartment);
    var apartment1 = new ApartmentPageViewModel
    {
        Id = apartment.Id,
        Price = apartment.Price,
        Footage = apartment.Footage,
        TypeOfAds = apartment.TypeOfAds,
    }
}

```

```

        Description = apartment.Description,
        Name = apartment.Name,
        date = apartment.date,
        AddedBy = apartment.AddedBy,
        Floor = apartment.Floor,
        ConstructionType = apartment.ConstructionType,
        NumberOfRooms = apartment.NumberOfRooms,
        AvgRate = rate,
        ImagePath = imagePath
        //Вот тут добавить среднюю оценку
    };

    HttpResponseMessage responseForImages = await
httpClient.GetAsync($"{_baseUrlService.apartmentServiceBaseUrl}/api/Apartments/GetApartmentImagesByFilePath?filePath={imagePath}");
    //Если картинки были найдены, добавить к модели картинки
    if (responseForImages.IsSuccessStatusCode)
    {
        var jsonResponseForImages = await
responseForImages.Content.ReadAsStringAsync();
        var _images =
JsonConvert.DeserializeObject<List<byte[]>>(jsonResponseForImages);
        apartment1.images = _images;
    }
    return View(apartment1);
}

```

Данный метод предназначен для отображения страницы с информацией об апартаменте. Он принимает два параметра: `id` – идентификатор апартамента и `imagePath` – путь к изображению апартамента.

Сначала метод отправляет запрос на получение средних оценок объектов недвижимости на сервис оценок. Если запрос успешен, он десериализует ответ в список объектов `AvgRate`, содержащий средние оценки.

Затем метод отправляет запрос на получение информации об апартаменте с указанным идентификатором. Полученные данные десериализуются в объект `Apartment`.

Далее создается объект `ApartmentPageViewModel`, содержащий информацию об апартаменте, средние оценки и путь к изображению.

Затем метод отправляет запрос на получение изображений апартамента по указанному пути `imagePath`. Если изображения найдены, они десериализуются в список байтов и добавляются к модели `ApartmentPageViewModel`.

Наконец, метод возвращает представление `ApartmentPage`, передавая в него созданный объект `ApartmentPageViewModel`, который будет использоваться для отображения информации о конкретном апартаменте на странице.

На странице объявления пользователь может получить дополнительную информацию об объекте недвижимости, такую как дополнительные фотографии, описание объявления и адрес. Помимо этого, если тип объявления «Сдача», то пользователь может поставить оценку объявлению. За выставление оценки отвечает единственный метод `RateApartment` контроллера `RateController`. Код метода приведен в листинге 18.

Листинг 18 – Метод `RateApartment` контроллера `RateController`

```
public async Task<IActionResult> RateApartment(int apartmentId, int rate,
string imagePath)
{
    var tokenString = HttpContext.Session.GetString("JWTToken");
    if (tokenString == null)
    {
        HttpContext.Session.SetString("RateAccess", "false");
        return RedirectToAction("LogReg", "LogReg");
    }
    var jwtToken = JObject.Parse(tokenString);
    jwtToken.TryGetValue("token", out var token);
    var tokenValue = token.ToString();
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);
    HttpResponseMessage response = await
_httpClient.GetAsync($"{_baseUrlService.authServiceBaseUrl}/api/authenti
cation/authenticate");
    if (!response.IsSuccessStatusCode)
    {
        HttpContext.Session.SetString("RateAccess", "false");
        return RedirectToAction("LogReg", "LogReg");
    }
    var userRaw = response.Content.ReadAsStringAsync().Result;
    var JobjUser = JObject.Parse(userRaw);
    var strUser = JobjUser["user"].ToString();
    var rateModel = new { ApartmentId = apartmentId, UserName = strUser,
Rate = rate };
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);
    HttpResponseMessage responseClient = await
_httpClient.PostAsJsonAsync($"{_baseUrlService.rateServiceBaseUrl}/api/r
ate", rateModel);
    if (responseClient.IsSuccessStatusCode)
    {
        HttpContext.Session.SetString("RateStatus", "true");
```

```

        return RedirectToAction("ApartmentPage", "ApartmentPage", new {
            id = apartmentId, imagePath = imagePath });
    }
    else
    {
        var responseBody =
            responseClient.Content.ReadAsStringAsync().Result;
        return BadRequest(responseClientBody);
    }
}

```

Данный метод принимает следующие параметры: `apartmentId` – идентификатор апартаментов, `rate` – оценка апартаментов, `imagePath` – путь к изображению апартаментов.

Сначала метод получает JWT-токен из сессии пользователя. Если токен отсутствует, метод устанавливает в сессии переменную `RateAccess` в значение «false» и перенаправляет пользователя на страницу входа/регистрации.

Затем метод проверяет валидность токена, отправляя запрос на сервис аутентификации. Если запрос успешен, из ответа извлекается имя пользователя.

Далее формируется модель оценки `rateModel`, содержащая идентификатор апартаментов, имя пользователя и оценку.

После этого метод устанавливает заголовок авторизации с JWT-токеном и отправляет запрос на сервис оценок для создания новой оценки. Если запрос успешен, устанавливается переменная `RateStatus` в сессии в значение «true», и пользователь перенаправляется на страницу апартаментов с указанным идентификатором и путем к изображению.

Если запрос на создание оценки не удался, метод возвращает ошибку `BadRequest` с телом ответа от сервиса.

Для выставления оценки метод `RateApartment` обращается к методу `PostRate` единственного контроллера `RateController`, находящегося внутри микросервиса `RateService`, который используется для взаимодействия с оценками. Вкратце опишем все методы контроллера, помимо конструктора.

Для выставления оценки объявлению используется метод `PostRate`. Этот метод обрабатывает HTTP POST запросы и принимает объект оценки (`RateModel`) в качестве параметра.

Сначала проверяется валидность модели данных с помощью ModelState. Если модель невалидна, возвращается ответ с ошибкой BadRequest, содержащий информацию о неверных данных.

Затем метод извлекает токен из заголовка запроса и проверяет его наличие. Если токен отсутствует, возвращается ответ с ошибкой Unauthorized.

Далее метод добавляет токен в заголовок запроса и отправляет запрос к методу аутентификации другого микросервиса для проверки токена.

Если токен прошел проверку успешно, извлекается имя пользователя из токена и сравнивается с именем пользователя, указанным в объекте оценки. Если имена не совпадают, возвращается ответ с ошибкой Unauthorized.

После этого метод проверяет наличие существующей записи оценки для данного пользователя и объекта недвижимости. Если запись уже существует, оценка обновляется. В противном случае создается новая запись.

В конечном итоге метод возвращает ответ с сообщением об успешном обновлении или добавлении оценки, либо с ошибкой Unauthorized, если токен не прошел проверку.

Код метода представлен в листинге 19.

Листинг 19 – метод PostRate контроллера RateController

```
[HttpPost]
public async Task<IActionResult> PostRate(RateModel rate)
{
    var strUser = "";

    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }
    // Получаем токен из заголовка запроса
    var token = Request.Headers["Authorization"].FirstOrDefault();

    // Проверяем, что токен присутствует
    if (string.IsNullOrEmpty(token))
    {
        return Unauthorized(new {message = "Токен отсутствует"});
    }
    var tokenWithoutBearer = token.Substring(7);
```

```

        // Добавляем токен в заголовок запроса к методу контроллера другого
        // микросервиса
        _httpClient.DefaultRequestHeaders.Authorization = new
        AuthenticationHeaderValue("Bearer", tokenWithoutBearer);

        // Отправляем данные к методу контроллера другого микросервиса
        HttpResponseMessage response = await
        _httpClient.GetAsync($"{_authServiceBaseUrl}/api/authentication/authentic
        ate");

        if (response.IsSuccessStatusCode)
        {

            var userRaw = response.Content.ReadAsStringAsync().Result;

            var JobjUser = JObject.Parse(userRaw);

            strUser = JobjUser["user"].ToString();

            if (strUser != rate.UserName)
            {
                return Unauthorized(new {message = "Отказано в доступе"});
            }

            var existingRate = _context.RateModels.FirstOrDefault(r =>
            r.UserName == rate.UserName && r.ApartmentId == rate.ApartmentId);

            // Если запись существует, обновляем ее
            if (existingRate != null)
            {
                existingRate.Rate = rate.Rate; // Обновляем оценку
                _context.SaveChanges();
                return Ok("Rate updated");
            }
            // Если записи нет, добавляем новую
            else
            {
                _context.RateModels.Add(rate);
                _context.SaveChanges();
                return Ok("New rate added");
            }
        }
        else
        {
            return Unauthorized();
        }
    }
}

```

За получение средних оценок объявлений отвечает метод `AvgRates`. Этот метод обрабатывает HTTP GET запросы и возвращает средние оценки для каждого объекта недвижимости.

Сначала метод выполняет запрос к базе данных для получения всех записей оценок. Затем записи группируются по `ApartmentId` с использованием

метода `GroupBy`. Для каждой группы вычисляется среднее значение оценки с помощью метода `Average`. Полученные результаты сохраняются в виде списка анонимных объектов, содержащих `ApartmentId` и среднюю оценку.

Наконец, метод возвращает ответ с данными о средних оценках в формате JSON.

Помимо просмотра объявлений и выставления им оценок пользователь также может зайти в свой личный кабинет (ЛК), нажав на кнопку «Личный кабинет» в навигационной панели сайта. После этого происходит перенаправление на страницу личного кабинета, за вывод представления ЛК отвечает метод `UserProfile` контроллера `AccountController`. Код метода приведен в листинге 20.

Листинг 20 – метод `UserProfile` контроллера `AccountController`

```
public async Task<IActionResult> UserProfile()
{
    var tokenString = HttpContext.Session.GetString("JWTToken");
    if (tokenString == null)
    {
        HttpContext.Session.SetString("EnterProfile", "false");
        return RedirectToAction("LogReg", "LogReg");
    }
    var jwtToken = JObject.Parse(tokenString);
    jwtToken.TryGetValue("token", out var token);
    var tokenValue = token.ToString();
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);
    HttpResponseMessage responseForAuth = await
_httpClient.GetAsync($"{_baseUrlService.authServiceBaseUrl}/api/authenti
cation/authenticate");
    if (!responseForAuth.IsSuccessStatusCode)
    {
        return RedirectToAction("LogReg", "LogReg");
    }
    var userRaw = responseForAuth.Content.ReadAsStringAsync().Result;
    var JobjUser = JObject.Parse(userRaw);
    var strUser = JobjUser["user"].ToString();
    HttpResponseMessage responseForApartments = await
_httpClient.GetAsync($"{_baseUrlService.apartmentServiceBaseUrl}/api/Apa
rtments/GetByAddedBystring?user={strUser}");
    if (!responseForApartments.IsSuccessStatusCode)
    {
        return BadRequest("failed to load apartments");
    }
}
```

```

        HttpResponseMessage responseForRate = await
_httpClient.GetAsync($"{_baseUrlService.rateServiceBaseUrl}/api/rate/Avg
Rates");
        if (!responseForRate.IsSuccessStatusCode)
        {
            return BadRequest("failed to load rates");
        }
        var jsonStringForApartments = await
responseForApartments.Content.ReadAsStringAsync();
        var apartments =
JsonConvert.DeserializeObject<List<ApartmentsWithImages>>(jsonStringForAp
artments);
        var jsonStringForRate = await
responseForRate.Content.ReadAsStringAsync();
        var rate =
JsonConvert.DeserializeObject<List<AvgRate>>(jsonStringForRate);
        var apartmentsForModel = new ApartmentsWithImagesAndPage
        {
            Apartments = apartments,
            PageIndex = 0,
        };
        var model = new ApartmentDataViewModel
        {
            User = new User()
            {
                UserName = strUser
            },

            Apartments = apartmentsForModel,
            AvgRate = rate
        };
        return View(model);
    }

```

Сначала метод проверяет наличие JWT-токена в сеансе пользователя. Если токен отсутствует, пользователь перенаправляется на страницу входа или регистрации.

После получения токена из сеанса, метод использует его для аутентификации запроса к другому микросервису для проверки подлинности пользователя. Если запрос успешен, метод извлекает информацию о пользователе из ответа.

Затем метод отправляет запрос к микросервису, который предоставляет список объявлений, добавленных пользователем, и к микросервису, который предоставляет средние оценки объявлений.

После успешного получения данных метод создает модель представления, содержащую информацию о пользователе, список объявлений пользователя и средние оценки.

Наконец, метод возвращает представление профиля пользователя с созданной моделью.

В ЛК пользователь может совершать следующие действия:

- посмотреть подробные данные о своих объявлениях, нажав на кнопку «Подробнее», в таком случае будет вызван метод `ApartmentPage` контроллера `ApartmentPageController`;
- удалить свое объявление, в таком случае при нажатии на кнопку «Удалить» номер объявления с формы отправляется в метод `DeleteApartment` контроллера `AccountController`;
- добавить новое объявление, в таком случае данные с формы добавления после нажатия кнопки «Добавить» отправляются в метод `PostApartmentsMulti` контроллера `AccountController`.

Мы уже рассмотрели метод `ApartmentPage`, так что ограничимся описанием методов `PostApartmentsMulti` и `DeleteApartment`.

Метод `PostApartmentsMulti` используется для добавления объявления в базу данных. Данный метод предназначен для обработки POST-запросов на добавление новых объявлений с возможностью загрузки нескольких изображений и другой информации об объекте недвижимости. Он начинается с проверки наличия JWT-токена в сессии пользователя и, если токен отсутствует, возвращает ошибку с сообщением «Invalid token». Затем метод извлекает токен из сессии и устанавливает его в HTTP-заголовок авторизации для последующих запросов.

Создается объект `MultipartFormDataContent` для отправки файлов и других данных. Для каждого файла из списка `files` создается объект `StreamContent`, который затем добавляется в `formData`. В этот же объект добавляются строки с информацией об объекте недвижимости: адрес, описание, кто добавил, цена,

площадь, тип объявления, этаж, тип строительства, количество комнат, а также текущая дата и время (добавляется поле `dateOfPost`).

После этого метод отправляет POST-запрос с использованием `formData` на указанный URL микросервиса для добавления нового объявления. Если запрос успешен, в сессию добавляется строка «`ApartmentAdded`» со значением «`true`», и пользователь перенаправляется на страницу профиля. Если запрос не успешен, метод возвращает ошибку с текстом ответа от сервера.

Таким образом, метод обрабатывает отправку данных о новом объекте, включая загрузку изображений, проверку авторизации и перенаправление пользователя после успешного выполнения операции.

Код метода представлен в листинге 21.

Листинг 21 – метод `PostApartmentsMulti` контроллера `AccountController`

```
public async Task<IActionResult> PostApartmentsMulti(List<IformFile>
files, string name, string description, string addedBy, int price, int
footage, bool typeOfAds, int floor, bool constructionType, int
numberOfRooms)
{
    // Проверка токена JWT
    var tokenString = HttpContext.Session.GetString("JWTToken");
    if (tokenString == null)
    {
        return BadRequest("Invalid token.");
    }
    var token = JObject.Parse(tokenString)["token"].ToString();
    // Установка токена авторизации в HTTP заголовок
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);
    // Создание объекта FormData для отправки файлов и других данных
    var formData = new MultipartFormDataContent();
    // Добавление файлов в FormData
    foreach (var file in files)
    {
        var fileContent = new StreamContent(file.OpenReadStream());
        formData.Add(fileContent, "files", file.FileName);
    }
    // Добавление других данных в FormData
    formData.Add(new StringContent(name), "name");
    formData.Add(new StringContent(description), "description");
    formData.Add(new StringContent(addedBy), "addedBy");
    formData.Add(new StringContent(price.ToString()), "price");
    formData.Add(new StringContent(footage.ToString()), "footage");
    formData.Add(new StringContent(typeOfAds.ToString()), "typeOfAds");
    formData.Add(new StringContent(floor.ToString()), "floor");
}
```

```

        formData.Add(new StringContent(constructionType.ToString()),
"constructionType");
        formData.Add(new StringContent(numberOfRooms.ToString()),
"numberOfRooms");
        //Добавить дату тут
        var date = DateTime.UtcNow;
        formData.Add(new StringContent(date.ToString()), "dateOfPost");
        // Отправка POST запроса с использованием FormData
        var response = await
_httpClient.PostAsync($"{_baseUrlService.apartmentServiceBaseUrl}/api/Ap
artments/postMany", formData);
        // Обработка ответа
        if (response.IsSuccessStatusCode)
        {
            HttpContext.Session.SetString("ApartmentAdded", "true");
            return RedirectToAction("UserProfile", "Account");
        }
        else
        {
            var responseBody = await response.Content.ReadAsStringAsync();
            return BadRequest(responseBody);
        }
    }
}

```

За удаление объявлений из интерфейса клиентского приложения отвечает метод `DeleteApartment`. Данный метод предназначен для удаления объявления по его идентификатору (id). Сначала он проверяет наличие JWT-токена в сессии пользователя. Если токен присутствует, он извлекается из строки и устанавливается в HTTP-заголовок авторизации для последующих запросов. Если токен отсутствует, метод возвращает ошибку с сообщением «wrong token when delete».

Затем формируется URL-адрес API микросервиса, отвечающего за удаление объявления, включающий идентификатор объявления. Метод отправляет DELETE-запрос на этот URL. Если запрос выполнен успешно, в сессию добавляется строка «deleted» со значением «true», и пользователь перенаправляется на страницу профиля.

Таким образом, метод обрабатывает удаление объявления, проверяя авторизацию пользователя, отправляя запрос на удаление и обрабатывая результат.

И наконец, если пользователь хочет закончить текущую сессию пользования приложением, он может выйти из своего аккаунта, нажав кнопку

«Выйти» на навигационной панели сайта. После этого вызывается метод LogoutUser контроллера LogRegController. В данном методе очищаются все данные сессии, затем устанавливается строка «logout» в сессию с значением «true» для обозначения того, что пользователь успешно вышел из системы, и наконец, пользователя перенаправляет на метод LogReg контроллера LogReg, где он может войти заново для новой сессии.

В данном разделе были подробно описаны все контроллеры и методы приложения. Каждый из методов был тщательно проанализирован с точки зрения его функциональности, взаимодействия с другими микросервисами. Все методы контроллеров выполняют свои задачи корректно и эффективно, обеспечивая основные функциональные возможности приложения. Исходный код клиентской части и микросервисов приведен в приложении Б.

Таким образом, все методы контроллеров приложения работают корректно и взаимодействуют друг с другом для обеспечения удобного и эффективного пользовательского опыта. Приложение продемонстрировало высокую степень надежности и безопасности, что делает его отличным инструментом для управления данными об апартаментах и их оценками.

4.3 Разработка графического приложения

Проектирование графического интерфейса пользователя – создании системы взаимодействия пользователя с элементами системы.

Общие принципы проектирования пользовательских интерфейсов:

- интерфейс должен быть интуитивно понятным, таким, чтобы пользователю не требовалось объяснять, как им пользоваться;
- чаще всего, пользователи в интерфейсе сначала ищут сущность (существительное), а затем действие (глагол) к ней;
- чем быстрее человек увидит результат – тем лучше;
- следует с осторожностью предоставлять пользователю возможность, по установке личных настроек;

– чем больше пользователь работает с какой-то конкретной задачей, тем больше он на ней концентрируется и тем меньше перестает замечать подсказки и сообщения, выводимые программой [8].

При разработке графического приложения использовались Razor Pages, они позволяют создавать страницы с кодом Razor, которые могут обрабатывать запросы. Razor Pages представляют альтернативу стандартной модели MVC для построения приложения.

Для отображения данных о недвижимости, а также регистрации и аутентификации, было разработано 4 представления. Помимо этого был изменен стандартный файл шаблона представлений `_Layout.cshtml`, отвечающий за отображение навигационной панели сайта. На рисунке 20 изображено представление для входа и регистрации.

Продажа и аренда недвижимости Главная Личный кабинет Логин/Регистрация

Добро пожаловать на наш сайт

Вход

Имя пользователя: 1

Пароль: 2

Войти 3

Регистрация

Имя пользователя: 4

Пароль: 5

Повторите пароль: 6

Зарегистрироваться 7

Рисунок 20 – Представление для входа и регистрации

Цифрами на рисунке отмечены основные элементы пользовательского интерфейса:

- 1 – поле ввода логина;
- 2 – поле ввода пароля;
- 3 – кнопка входа;
- 4 – поле ввода имени пользователя;
- 5 – поле ввода пароля;
- 6 – поле для подтверждения пароля;
- 7 – кнопка регистрации.

В этом представлении пользователь имеет возможность создать новый аккаунт или авторизоваться в существующем.

На рисунке 21 изображено представление для главной страницы сайта.

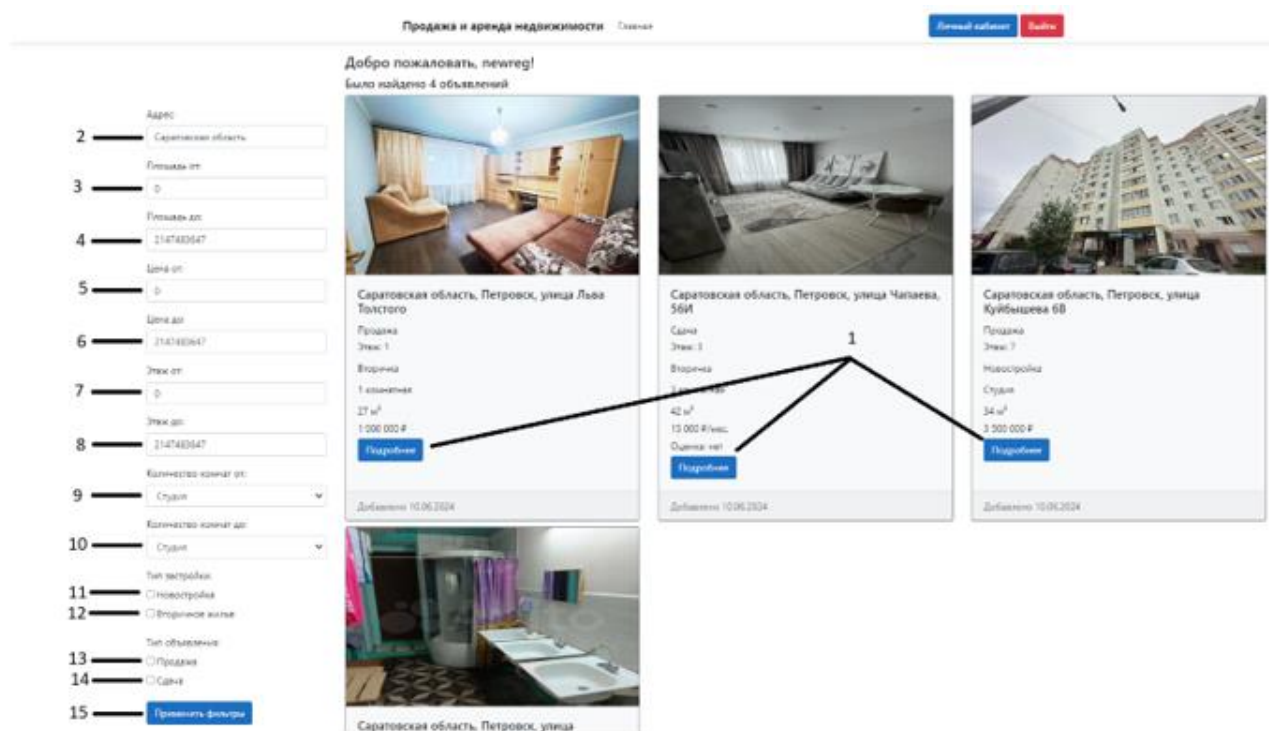


Рисунок 21 – Представление для главной страницы сайта

Рассмотрим основные интерактивные элементы представления главной страницы сайта:

- 1 – кнопка «Подробнее» для перехода на страницу объявления;
- 2 – поле для указания адреса поиска;
- 3 – поле для указания минимальной площади;
- 4 – поле для указания максимальной площади;

- 5 – поле для указания минимальной цены;
- 6 – поле для указания максимальной цены;
- 7 – поле для указания минимального этажа;
- 8 – поле для указания максимального этажа;
- 9 – выпадающий список для указания минимального количества комнат (от «Студия» до «5 и более комнат»);
- 10 – выпадающий список для указания максимального количества комнат (от «Студия» до «5 и более комнат»);
- 11 – флажок для указания, является ли тип застройки «Новостройка»;
- 12 – флажок для указания, является ли тип застройки «Вторичное жилье»;
- 13 – флажок для указания, является ли тип объявления «Продажа»;
- 14 – флажок для указания, является ли тип объявления «Сдача»;
- 15 – кнопка «Применить фильтры» для вывода объявлений, соответствующих фильтрам.

В этом представлении можно просмотреть все добавленные объявления, зайти на страницу конкретного объявления а также применить фильтры при необходимости.

При нажатии на кнопку «Применить фильтры» часть выбранных фильтров сохраняется, чтобы пользователю не пришлось каждый раз заново вводить те же фильтры с незначительными изменениями.

Также на главной странице предусмотрены элементы пагинации, чтобы пользователи могли удобно просматривать объявления. При переходе на другую страницу выбранные фильтры остаются примененными, обеспечивая непрерывность поиска.

На рисунке 22 изображены элементы пагинации представления главной страницы сайта.

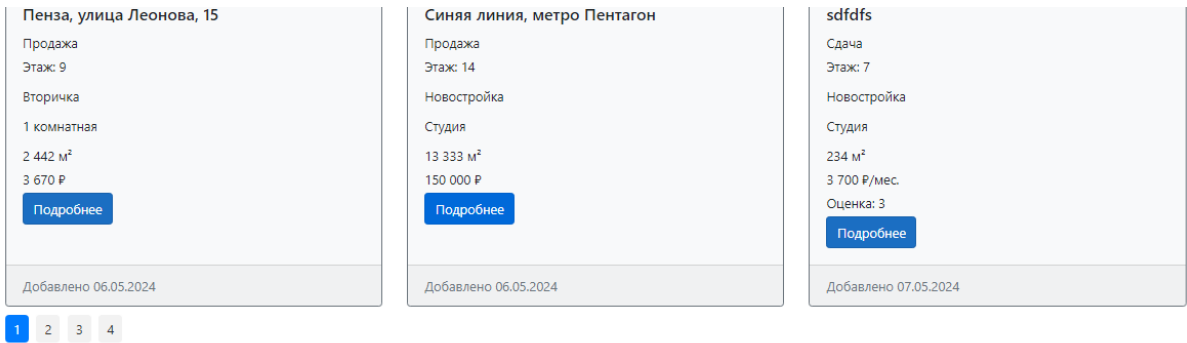


Рисунок 22 – Элементы пагинации в представлении главной страницы

Элементы пагинации используются для разбиения большого объема данных на более мелкие страницы, чтобы улучшить удобство просмотра и навигации для пользователя. Они помогают уменьшить время загрузки страницы, облегчить поиск нужной информации и избежать перегрузки интерфейса.

На рисунках 23, 24 изображено представление личного кабинета пользователя.

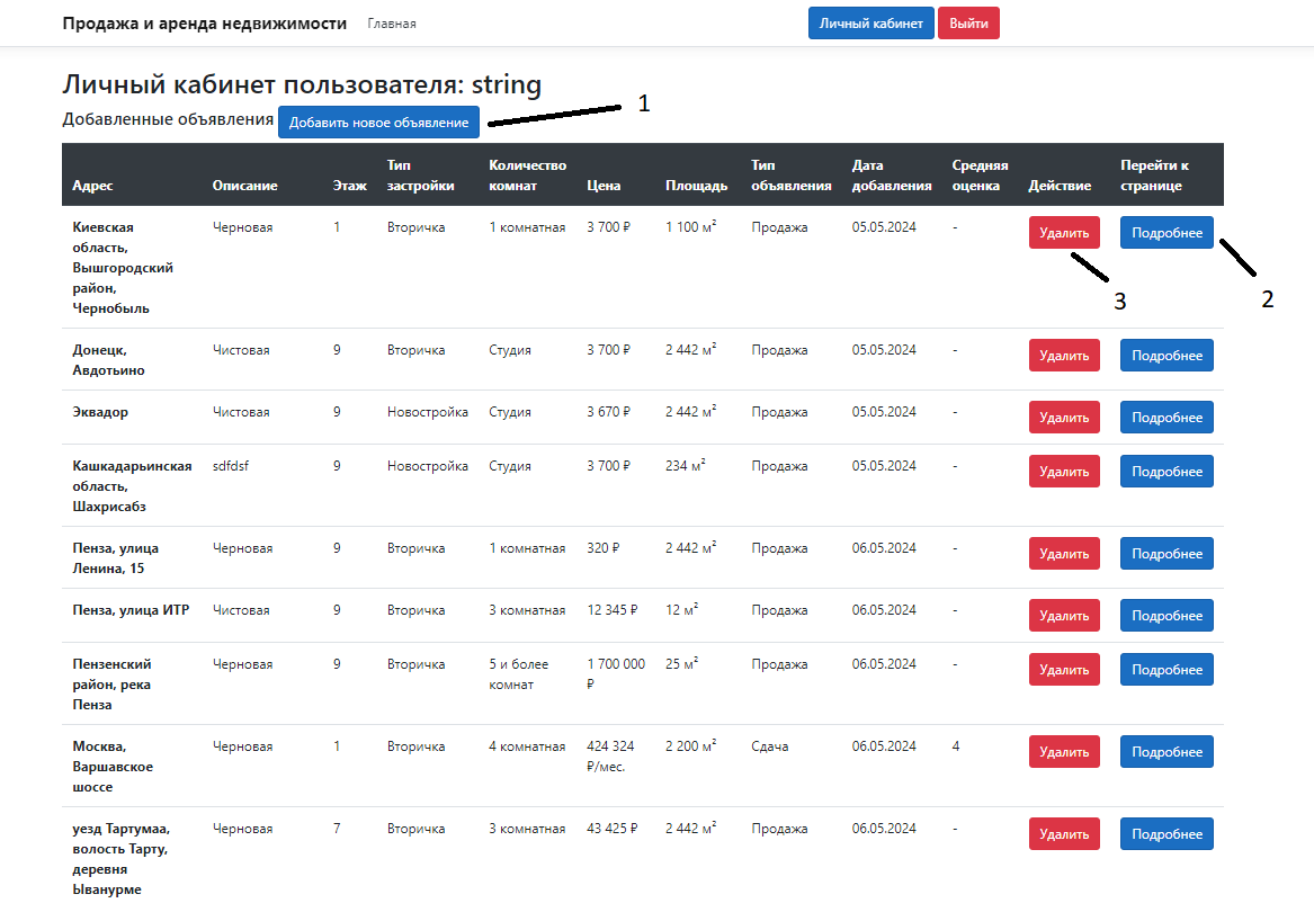


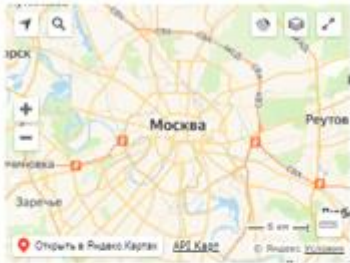
Рисунок 23 – Представление личного кабинета, объявления пользователя

Продажа и аренда недвижимости Главная

Личный кабинет Выйти

Добавление

Адрес:



Описание:

Этаж:

Тип Застройки:

Количество комнат:

Цена:

Площадь:

Тип Объявления:

Выберите файлы: No file chosen

1 2 3 4 5 6 7 8 9 10 11

Рисунок 24 – Представление личного кабинета, форма добавления

На рисунке 23 изображена верхняя часть представления для личного кабинета пользователя. Цифрами на нём отмечены основные элементы пользовательского интерфейса:

- 1 – кнопка «Добавить новое объявление» для прокрутки к форме добавления;
- 2 – кнопка «Подробнее» для перехода на страницу объявления;
- 3 – кнопка «Удалить» для удаления объявления.

На рисунке 24 изображена форма для добавления объявления в представлении личного кабинета. Цифрами на нём отмечены основные элементы пользовательского интерфейса:

- 1 – поле для указания адреса объекта недвижимости;

- 2 – интерактивная карта для просмотра правильности введенного адреса объекта недвижимости;
- 3 – поле для описания объявления;
- 4 – поле для указания этажа;
- 5 – выпадающий список для указания типа застройки («Новостройка» или «Вторичка»);
- 6 – выпадающий список для указания количества комнат (от «Студия» до «5 и более комнат»);
- 7 – поле для указания цены объекта недвижимости;
- 8 – поле для указания площади объекта;
- 9 – выпадающий список для указания типа объявления («Продажа» или «Сдача»).

В данном представлении пользователи могут управлять своими объявлениями различными способами. Во-первых, они имеют возможность просматривать все объявления, которые были добавлены с их аккаунта, что позволяет легко отслеживать и управлять активными предложениями. Кроме того, пользователи могут перейти на страницу любого из своих объявлений, чтобы увидеть его детальную информацию и внести необходимые изменения. При необходимости, они могут удалить любое из своих объявлений, если оно больше не актуально или нужно сделать какие-то правки. Также доступна форма для добавления новых объявлений, позволяющая пользователям быстро и удобно разместить новые предложения, заполнив все необходимые поля и загрузив соответствующие изображения. Таким образом, данное представление обеспечивает контроль над процессом управления объявлениями, делая его интуитивно понятным и функционально насыщенным.

На рисунках 25, 26 изображено представление для страницы объявления, которое состоит из двух частей: Основная информации об объявлении и интерактивная карта, расположенная ниже.

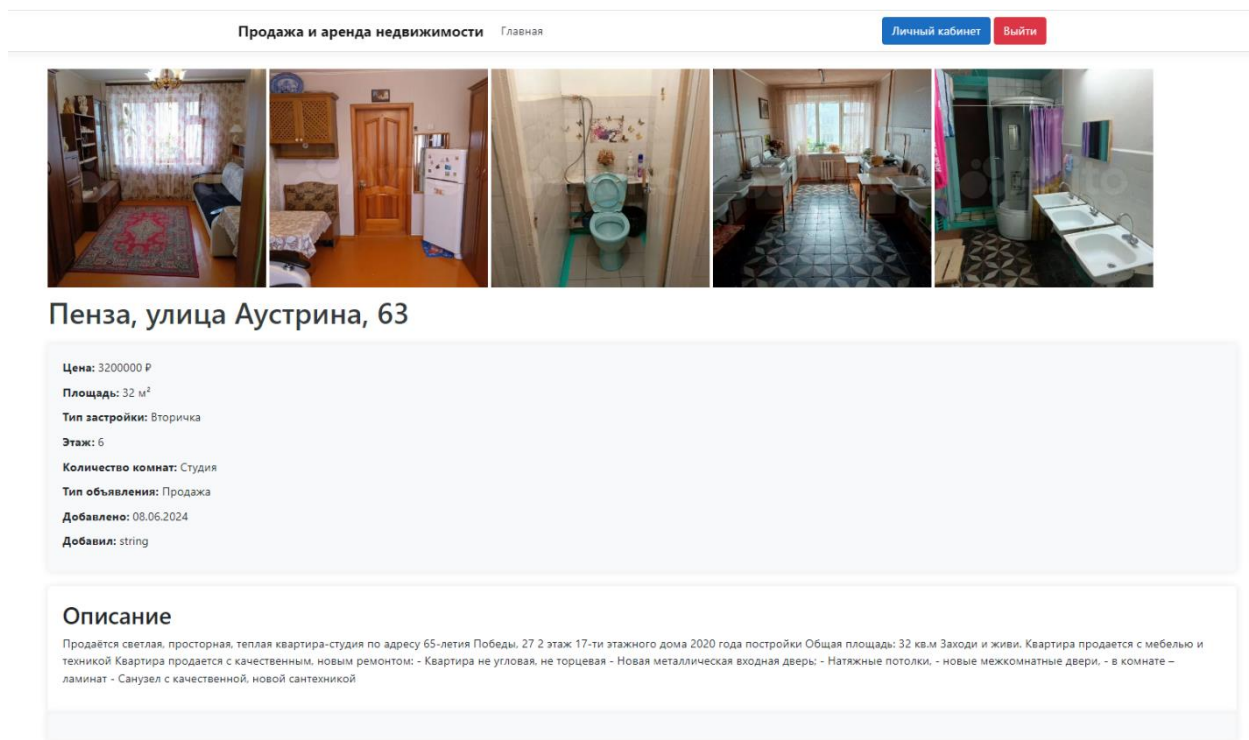


Рисунок 25 – Представление для страницы объявления, верхняя часть

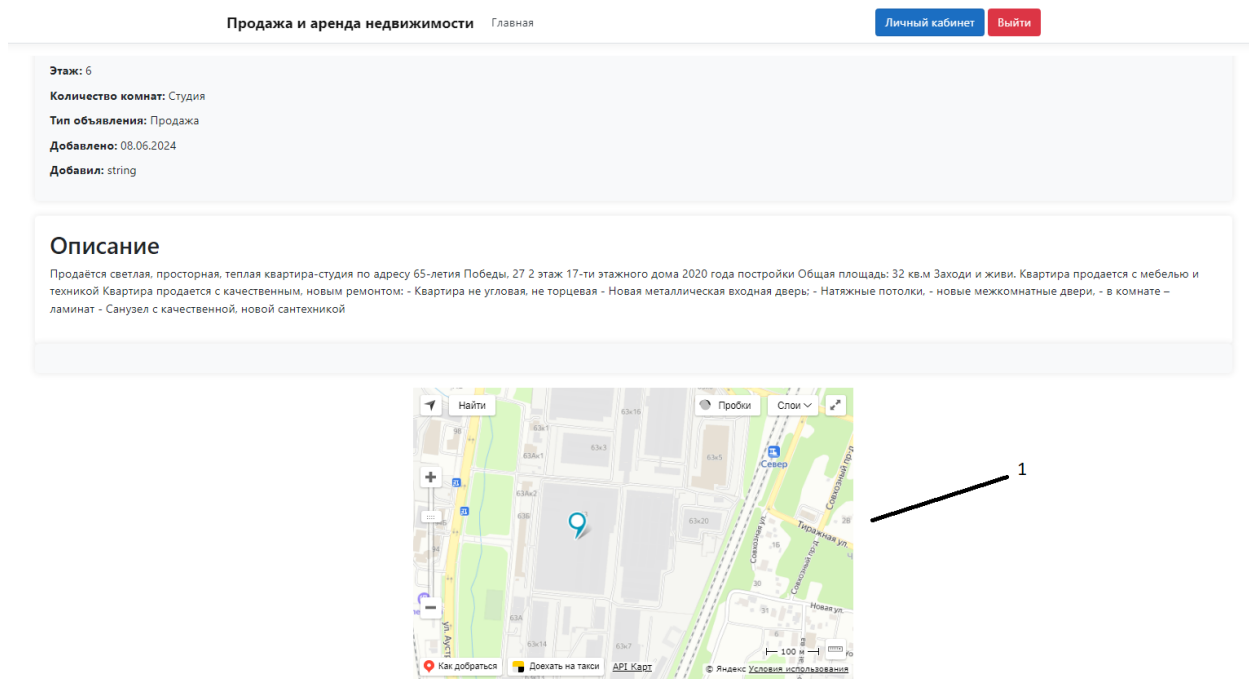


Рисунок 26 – Представление для страницы объявления, нижняя часть

На рисунке 25 изображено представление для страницы объявления, на нем отображены данные объявления в удобном для пользователя виде. На рисунке 26 показана нижняя часть страницы с единственным интерактивным

элементом – картой, на которой изображено расположения объекта недвижимости. Карта отмечена цифрой 1.

Также если при добавлении объявления был выбран тип объявления «Сдача», то пользователь получает возможность видеть оценку объявления на главной странице сайта, а при открытии страницы объявления появляется форма для добавления оценки.

Если при создании объявления был выбран тип объявления «Сдача», то в представлении также появится форма для выставления оценки (рисунок 27).

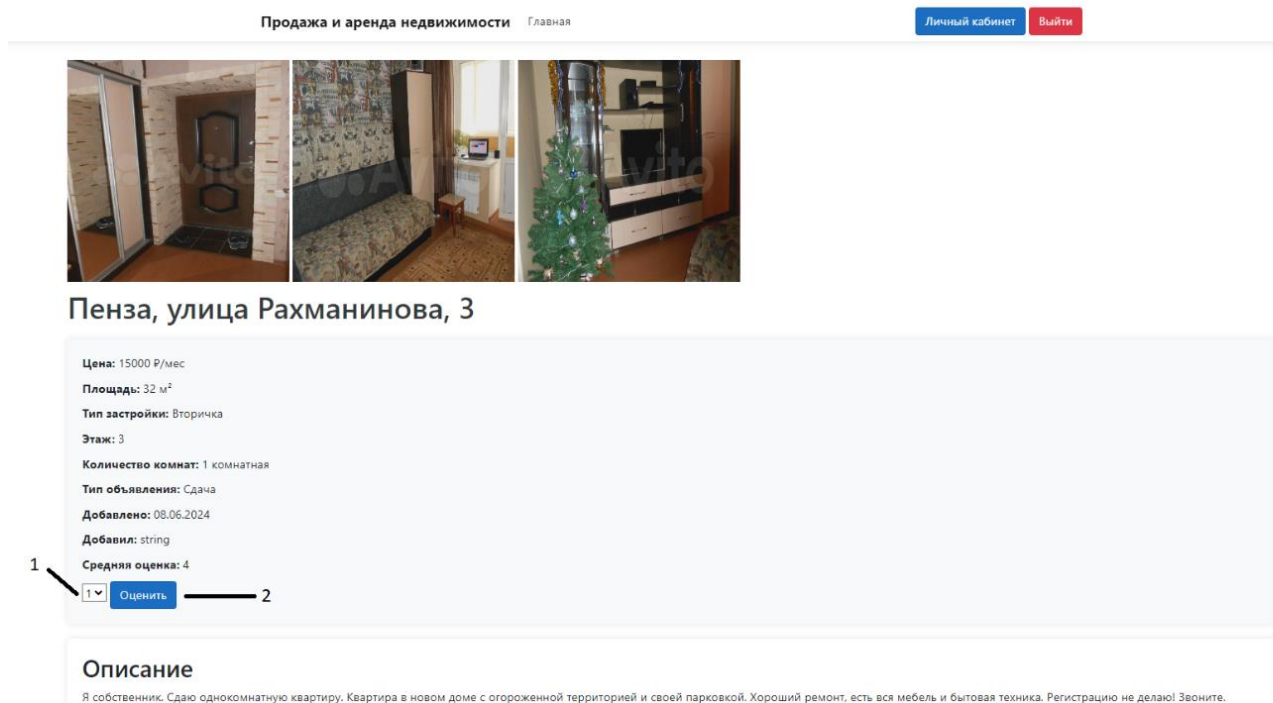


Рисунок 27 – Представление для страницы объявления, форма для оценки

Рассмотрим элементы, из которых состоит форма добавления оценки:

- 1 – выпадающий список для выбора выставляемой оценки (от 1 до 5);
- 2 – кнопка «Оценить» для выставления оценки объявлению.

В данном представлении можно посмотреть дополнительные данные объявления, такие как дополнительные изображения, имя пользователя, добавившего объявление, описание объявления а также карту, на которой изображен объект недвижимости.

На рисунках 28, 29 изображены разные версии навигационной панели сайта в зависимости от того, аутентифицирован ли пользователь в системе.



Рисунок 28 – Представление навигационной панели сайта, пользователь не вошел в аккаунт



Рисунок 29 – Представление навигационной панели сайта, пользователь вошел в аккаунт

На рисунке 28 изображено представление навигационной панели сайта, разметка которого содержится в файле шаблона представлений. Когда пользователь не зарегистрировался в системе, справа от личного кабинета отображается кнопка «Логин/Регистрация». Цифрами на нем отмечены основные элементы пользовательского интерфейса:

- 1 – кнопка «Продажа и аренда недвижимости» для перехода на главную страницу сайта;
- 2 – кнопка «Главная» для перехода на главную страницу сайта;
- 3 – кнопка «Личный кабинет» для перехода в личный кабинет пользователя;
- 4 – кнопка «Логин/Регистрация» для перехода на страницу для логина и регистрации.

На рисунке 29 изображено представление навигационной панели сайта в случае, если пользователь вошел в систему. Оно отличается тем, что вместо кнопки «Логин/Регистрация» отображается кнопка «Выйти», при нажатии на которую данные сессии пользователя стираются, а пользователь перенаправляется на страницу регистрации.

Данный шаблон представления позволяет пользователям легко перемещаться между основными разделами сайта, улучшая удобство и навигацию.

Также в приложении реализован механизм вывода сообщений, которые отображаются в представлениях при определенных действиях пользователя, таких как добавление объявлений, оценок, вход в аккаунт, регистрация и т. д.

Исходный код интерфейса приведен в приложении В.

Таким образом, данные представления обеспечивают удобную и интуитивно понятную навигацию по сайту, что значительно улучшает пользовательский опыт. Благодаря продуманной структуре и доступности основных функций, пользователи могут легко и быстро находить необходимую информацию, что повышает общую удовлетворенность от использования приложения.

4.4 Тестирование информационной системы

4.4.1 Классы эквивалентности

Тестирование методом чёрного ящика определяется как методика тестирования, при которой функциональность тестируемого приложения (AUT) тестируется без учета внутренней структуры кода, деталей реализации и знания внутренних путей программного обеспечения.

Этот тип тестирования полностью основан на требованиях и спецификациях программного обеспечения. В тестирование методом чёрного ящика фокусируются на входах и выходах программной системы, не заботясь о внутренних знаниях программ.

Вышеупомянутый чёрный ящик может быть любой программной системой, которую нужно протестировать. Например, операционная система, такая как Linux, веб-сайт, такой как Google, база данных, такая как Oracle, или даже просто разработанное приложение.

В тестировании методом чёрного ящика можно протестировать эти приложения, просто сосредоточившись на входах и выходах, не зная их внутренней реализации кода [9].

Для реализации тестирования методом черного ящика были использованы классы эквивалентности, метод граничных значений и функциональное тестирование.

Классом эквивалентности называется набор данных, который запускает одни и те же модули и должен приводить к одним и тем же результатам [10].

В приведенном примере тестируется форма ввода пароля, которая предусматривает, что введенное значение должно быть длиной от 8 до 50 символов.

Для того, чтобы в дальнейшем была возможность составить тест-кейс, требуется обозначить все классы эквивалентности, которые делятся на правильные и неправильные.

Представление классов эквивалентности для тестирования поля ввода пароля представлено в таблице 25.

Таблица 25 – Классы эквивалентности для поля ввода пароля

	Правильные КЭ	Неправильные КЭ
Длина	8 – 50 (1)	0 – 7 (2) 51 – ∞ (3)
Структура	Имеет заглавную букву (4) Имеет строчную букву (5) Имеет спец. Символ (6)	Не имеет заглавной буквы (7) Не имеет строчной буквы (8) Не имеет спец.символов (9)
Содержание	Английские строчные буквы (10) Английские заглавные буквы (11) Спец. Символы (12)	Все остальное (13)

На основе классов эквивалентности были составлены тест-кейсы, которые состоят из списка входящих классов эквивалентности, а также из входных и выходных данных.

Тест-кейс для поля ввода пароля представлен в таблице 26.

Таблица 26 – Классы эквивалентности для поля ввода пароля: тест-кейсы

№	КЭ	Входные данные	Выходные данные
1	Все правильные 1, 4, 5, 6, 10, 11, 12	string4@B	Успешно
2	2, 4, 5, 6, 10, 11, 12	str4@B	Ошибка: пароль должен иметь длину не менее 8 символов
3	1, 5, 6, 10, 12	string4@b	Ошибка: пароль должен содержать хотя бы одну заглавную букву
4	1, 5, 6, 10, 12, 13	string4@bЯЯЯ	Ошибка: пароль должен содержать хотя бы одну заглавную букву
5	1, 4, 5, 10, 11	string4B	Ошибка: пароль должен содержать хотя бы один неалфавитно-цифровой символ
6	1, 4, 6, 11, 12	STRING4@B	Ошибка: Пароль должен содержать хотя бы одну строчную букву
7	1, 4, 6, 11, 12, 13	STRING4@Вяяя	Ошибка: Пароль должен содержать хотя бы одну строчную букву
8	3, 4, 5, 6, 10, 11, 12	string4@BBBBBBBBBBBBBB BBBBBBBBBBBBBBBBBB BBBBBBBBBBBBBB	Ошибка: Пароль не должен превышать 50 символов

Результаты тест-кейсов 2 – 8 приведены на рисунках 1 – 7 приложения Г.

4.4.2 Функциональная диаграмма и таблица решений

Метод функциональных диаграмм, также известных как диаграммы причинно-следственных связей, служит для систематического выбора наиболее эффективных тестов. Этот метод также имеет дополнительное преимущество, поскольку позволяет выявлять неполноты и неоднозначности в исходных спецификациях.

Функциональная диаграмма представляет собой формальный язык, на который переводится спецификация, изначально написанная на естественном языке.

Используя функциональные диаграммы, спецификации разбиваются на более управляемые части, после чего определяются и нумеруются причины и следствия. Эти элементы преобразуются в булевский граф, дополненный примечаниями для учета ограничений. Далее граф преобразуется в таблицу решений, что позволяет систематически генерировать тесты, исходя из различных комбинаций условий.

Таблица причин и следствий, построенная для тестирования формы добавления нового объявления, представлена в таблице 27.

Таблица 27 – Причины и следствия для формы добавления объявления

Причины	
П1	Адрес введен
П2	Адрес корректен
П3	Описание введено
П4	Описание корректно
П5	Этаж введен
П6	Этаж корректен
П7	Цена введена
П8	Цена корректна

Продолжение таблицы 27

Причины	
П9	Площадь введена
П10	Площадь корректна
Следствия	
C1	Сообщение: «Поле должно содержать хотя бы одну букву»
C2	Сообщение: «Поле должно содержать хотя бы одну букву. Поле должно начинаться не с пробела»
C3	Сообщение: «Поле должно содержать хотя бы одну букву»
C4	Сообщение: «Поле должно содержать хотя бы одну букву. Поле должно начинаться не с пробела»
C5	Сообщение: «Please fill out this field»
C6	Сообщение «Value must be greater than or equal to 1» или «Value must be less than or equal to 2147483647»
C7	Сообщение: «Please fill out this field»
C8	Сообщение «Value must be greater than or equal to 1» или «Value must be less than or equal to 2147483647»
C9	Сообщение: «Please fill out this field»
C10	Сообщение «Value must be greater than or equal to 1» или «Value must be less than or equal to 2147483647»
C11	Сообщение «Объявление успешно добавлено»

Для графического представления причин и следствий была построена функциональная диаграмма, отражающая связь между входными и выходными данными.

Функциональная диаграмма причин и следствий приведена на рисунке 30.

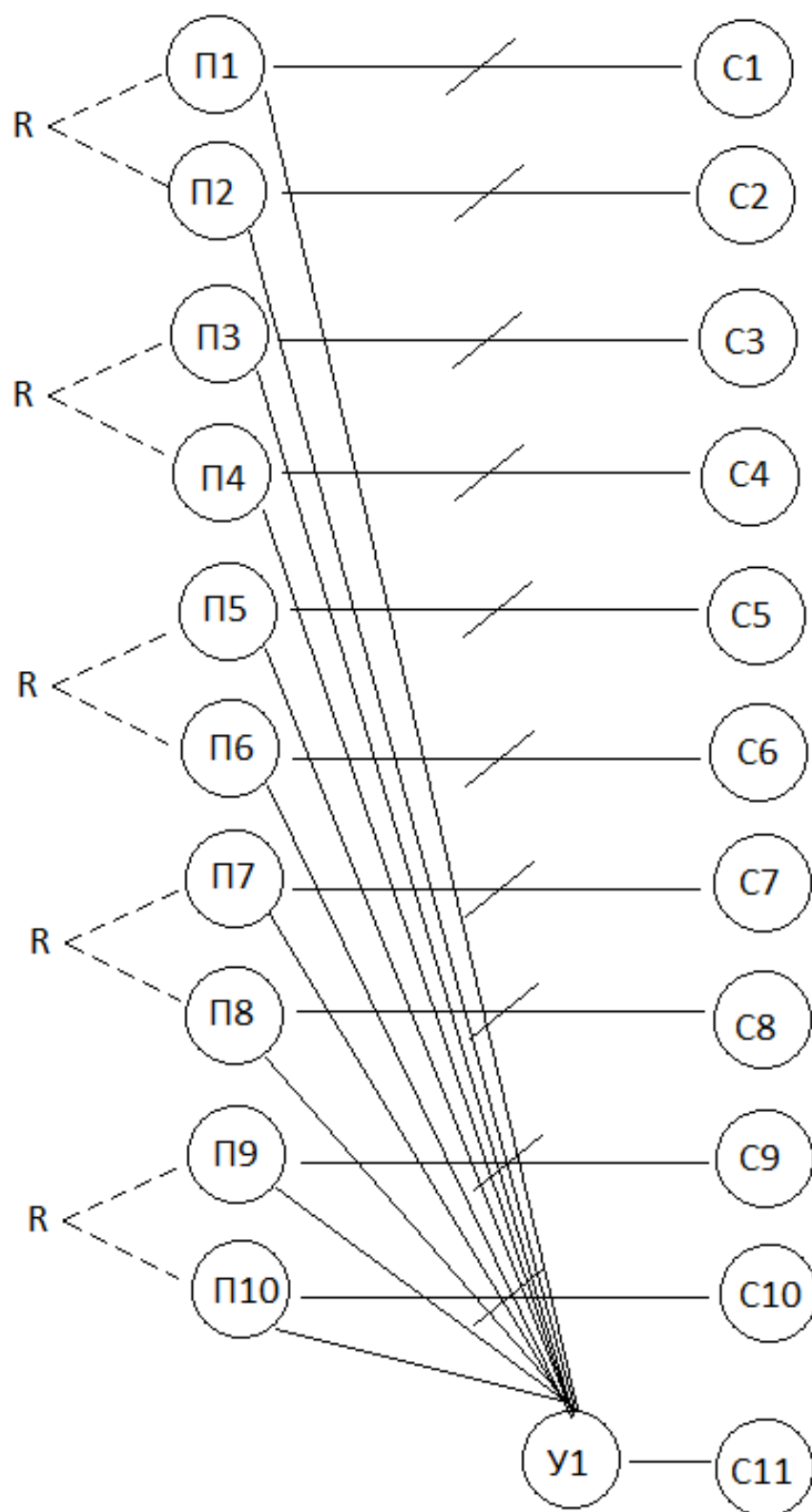


Рисунок 30 – Функциональная диаграмма причин и следствий

На диаграмме показано, что если первая причина не выполняется, то выполняется первое следствие. Таким же образом, если вторая причина не

выполняется, то выполняется второе следствие, и так далее. Также вторая причина зависит от первой: если первая причина не будет выполнена, то вторая тоже не может быть выполненной, так как адрес объявления не может быть корректным, если он не введен вовсе. Таким же образом зависят друг от друга причины 3 и 4, 5 и 6, 7 и 8, а также 9 и 10.

В случае, если все причины успешно выполнены, то выполняется одиннадцатое следствие, которое сообщает о том, что объявление добавлено.

На основе таблицы причин и следствия была построена таблица решений, отражающая, как зависят друг от друга следствия и причины на разных комбинациях.

Таблица решений представлена в таблице 28.

Таблица 28 – Таблица решений

	1	2	3	4	5	6	7	8	9
П1	1	0	1	1	1	1	1	1	1
П2	0	0	1	1	1	1	1	1	1
П3	1	0	1	1	1	0	1	1	1
П4	0	0	1	1	1	0	1	1	1
П5	1	0	1	1	1	1	1	0	1
П6	0	0	1	1	1	1	0	0	1
П7	1	0	1	1	0	1	1	1	1
П8	0	0	0	1	0	1	1	1	1
П9	1	0	1	1	1	1	1	1	1
П10	0	0	0	0	1	1	1	1	1
С1	0	1	0	0	0	0	0	0	0
С2	1	0	0	0	0	0	0	0	0
С3	0	1	0	0	0	1	0	0	0
С4	1	0	0	0	0	1	0	0	0
С5	0	1	0	0	0	0	1	0	0
С6	1	0	0	0	0	0	0	1	0
С7	0	1	0	0	1	0	0	0	0
С8	1	0	1	0	1	0	0	1	0
С9	0	1	0	0	0	0	0	1	0
С10	1	0	1	1	0	0	0	1	0
С11	0	0	0	0	0	0	0	0	1

Далее были составлены тест-кейсы, представленные в таблице 29.

Таблица 29 – Таблица решений: тест-кейсы

№	Входные данные	Выходные данные
1	Адрес: « @\$» Описание: «@\$» Этаж: «-1» Цена: «-1» Площадь: «-1»	Сообщение: «Поле не должно начинаться с пробела. Поле должно содержать хотя бы одну букву» Сообщение: «Поле должно содержать хотя бы одну букву» Сообщение: «Value must be greater than or equal to 0». Результат на рисунке 8 приложения Г.
2	Адрес: (нет) Описание: (нет) Этаж: (нет) Цена: (нет) Площадь: (нет)	Сообщение: «Please fill out this field» Сообщение: «Поле должно содержать хотя бы одну букву» Сообщение: «Поле должно содержать хотя бы одну букву». Результат на рисунке 9 приложения Г.
3	Адрес: «Пенза, улица Аустрина, 162» Описание: «Описание» Этаж: 8 Цена: 9999999999999999 Площадь: 9999999999999999	Сообщение: «Value must be less than or equal to 2147483647» Результат на рисунке 10 приложения Г.
4	Адрес: «Пенза, улица Аустрина, 162» Описание: «Описание» Этаж: 8 Цена: 18000 Площадь: «-3»	Сообщение: «Value must be greater than or equal to 1» Результат на рисунке 11 приложения Г.
5	Адрес: «Пенза, улица Аустрина, 162» Описание: «Описание» Этаж: 8 Цена: (нет) Площадь: 45	Сообщение: «Please fill out this field» Результат на рисунке 12 приложения Г.
6	Адрес: «Пенза, улица Аустрина, 162» Описание: (нет) Этаж: 8 Цена: 14000 Площадь: 45	Сообщение: «Please fill out this field» Сообщение: «Поле должно содержать хотя бы одну букву» Результат на рисунке 13 приложения Г.

Продолжение таблицы 29

7	Адрес: «Пенза, улица Аустрина, 162» Описание: «Описание» Этаж: 999999999999999 Цена: 14000 Площадь: 45	Сообщение: «Value must be less than or equal to 2147483647» Результат на рисунке 14 приложения Г.
8	Адрес: «Пенза, улица Аустрина, 162» Описание: «Описание» Этаж: (нет) Цена: 14000 Площадь: 45	Сообщение: «Please fill out this field» Результат на рисунке 15 приложения Г.
9	Адрес: «Пенза, улица Чкалова, 15» Описание: «<Информация>» Этаж: «4» Цена: «15000» Площадь: «43»	Сообщение: «Объявление успешно добавлено» Результат на рисунке 16 приложения Г.

В таблице перечислены входные и соответствующие выходные данные.

В ходе тестирования было выяснено, что форма добавления нового объявления работает правильно, а при некорректных данных оповещает об этом пользователя.

4.4.3 Функциональное тестирование

В результате тестирования приложения были проверены функции авторизации, регистрации, создания объявления, удаления объявления и выставления оценки объявлению.

Результаты тестов приведены в таблице 30.

Таблица 30 – Результаты тестирования

№ п/п	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
----------	----------------	-----------------------------	-------------------------	---------------------------

Продолжение таблицы 30

№ п/п	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
1	Проверка регистрации	После ввода логина, пароля и подтверждения пароля и нажатия на кнопку «Зарегистрироваться» пользователь будет направлен на главную страницу, на которой будет показано сообщение «Регистрация выполнена успешно»	Пользователь был перенаправлен на главную страницу, на которой было показано сообщение «Регистрация выполнена успешно»	Тест успешно пройден. Результат на рисунках 17, 18 приложения Г.
2	Проверка авторизации	После ввода логина и пароля и нажатия на кнопку «Войти» пользователь будет направлен на главную страницу, на которой будет показано сообщение «Вход выполнен успешно»	Пользователь был перенаправлен на главную страницу, на которой было показано сообщение «Вход выполнен успешно»	Тест успешно пройден. Результат на рисунках 19, 20 приложения Г.
3	Проверка создания объявления	После ввода данных объявления и нажатия кнопки «Добавить» пользователь будет перенаправлен на страницу личного кабинета, на которой будет выведено сообщение	Пользователь был перенаправлен на страницу личного кабинета, на которой было выведено сообщение «Объявление успешно	Тест успешно пройден. Результат на рисунках 21, 22 приложения Г.

Продолжение таблицы 30

№ п/п	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
		«Объявление успешно добавлено», а объявление появится на главной странице и в личном кабинете пользователя	добавлено», а объявление появилось на главной странице и в личном кабинете пользователя	
4	Проверка удаления объявления	После нажатия на кнопку «Удалить» пользователь будет перенаправлен на страницу личного кабинета, на которой будет выведено сообщение «Объявление успешно удалено», а выбранное объявление будет более недоступно для просмотра на главной странице и в личном кабинете	Пользователь был перенаправлен на страницу личного кабинета, на которой было выведено сообщение «Объявление успешно удалено», удаленное объявление более недоступно для просмотра на главной странице и в личном кабинете	Тест успешно пройден. Результат на рисунках 23, 24 приложения Г.
5	Проверка выставления оценки недвижимости	После нажатия на кнопку «Оценить» и выбора оценки из выпадающего списка пользователь перенаправится на страницу объявления, на которой выводится сообщение «Оценка	Произошло перенаправление на страницу объявления, на которой было показано сообщение «Объявление успешно добавлено», а	Тест успешно пройден. Результат на рисунках 25, 26 приложения Г.

Продолжение таблицы 30

№ п/п	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
		успешно добавлена», а значение средней оценки изменяется соответственно добавленной оценке	значение средней оценки было изменено в соответствии с добавленной оценкой	
6	Проверка фильтров	После выставления на главной странице значений в полях фильтров и нажатия на кнопку «Показать результаты» будут найдены соответствующие объявления	После выставления фильтров были найдены соответствующие объявления	Тест успешно пройден. Результат на рисунках 27, 28 приложения Г.
7	Проверка переключения страницы объявления	После нажатия на кнопку номера страницы в представлении главной страницы будут показаны объявления, соответствующие выбранной странице	Были показаны объявления, соответствующие выбранной странице	Тест успешно пройден. Результат на рисунках 29, 30 приложения Г.
8	Проверка выхода из аккаунта	После нажатия на кнопку «Выйти» пользователь будет перенаправлен на страницу для логина и регистрации, на	Был осуществлен выход из аккаунта, при этом пользователь не может более поставить оценку	Тест успешно пройден. Результат на рисунках 31, 32

Продолжение таблицы 30

№ п/п	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
		которой будет выведено сообщение «Выход прошел успешно», после этого пользователь не сможет зайти в свой личный кабинет или поставить оценку	или зайти в личный кабинет	приложения Г.
9	Проверка неавторизованного входа в личный кабинет	При нажатия на кнопку «Личный кабинет» пользователь будет перенаправлен на страницу логина и регистрации, на которой будет выведено сообщение «Войдите или зарегистрируйтесь, чтобы получить доступ к личному кабинету»	Пользователь был перенаправлен на страницу логина и регистрации, на которой было выведно соответствующее сообщение о том, что необходимо войти или зарегистрироваться, чтобы получить доступ к личному кабинету	Тест успешно пройден. Результат на рисунках 33, 34 приложения Г.
10	Попытка поставить оценку, не будучи авторизованным в приложение	При нажатия на кнопку «Оценить» на странице объявления пользователь будет перенаправлен на страницу логина и регистрации, на которой будет	Пользователь был перенаправлен на страницу логина и регистрации, на которой было выведно соответствующее сообщение о том,	Тест успешно пройден. Результат на рисунках 35, 36 приложения Г.

Окончание таблицы 30

№ п/п	Название теста	Предполагаемый результат	Полученный результат	Результат тестирования
		выведено сообщение «Войдите или зарегистрируйтесь, чтобы ставить оценки»	что необходимо войти или зарегистрироваться, чтобы ставить оценки	

Результаты тестирования свидетельствуют о правильной работе приложения.

4.4.4 Метрика Холстеда

Для оценки сложности программного кода генерации главной страницы сайта, представленной ранее (листинг 8) была использована метрика Холстеда.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- число уникальных операндов программы (словарь операндов);
- общее число операторов в программе;
- общее число операндов в программе [11].

В качестве входных данных функция принимает данные о параметрах поиска, после чего эти параметры отправляются микросервису недвижимости и получившийся ответ после десериализации возвращается в качестве модели.

Далее загружаются данные о количестве объявлений, данные об оценках и данные о имени текущего пользователя. Если пользователь не аутентифицирован – его имя в представлении сайта обозначается как «Аноним». Если не удалось загрузить оценки, возвращается сообщение «Не удалось

загрузить оценки объявлений». Если не загрузились данные о недвижимости, будь то количество объявлений или сами объявления, возвращается «Не удалось загрузить данные о количестве объявлений» или «Не удалось загрузить объявления» соответственно.

Для использования метрики Холстеда требуется подсчитать количество операторов и операндом. Искомые данные представлены в таблицах 31, 32.

Таблица 31 – Операторы

Операторы	Число вхождений
public	1
async	1
Task<IactionResult>	1
Int	7
bool	4
=	10
await	5
_httpClient.GetAsync	4
+	22
if	3
!	2
responseForApartments.IsSuccessStatusCode	1
BadRequest	2
JsonConvert.DeserializeObject	3
HttpContext.Session.GetString	1
string	1
var	8
JsonObject.Parse	2
TryGetValue	1

Продолжение таблицы 31

Операторы	Число вхождений
ToString	1
_httpClient.DefaultRequestHeaders.Authorization	1
AuthenticationHeaderValue	1
responseForAuth.Content.ReadAsStringAsync().Result	1
View	1
{	19
}	19
(	30
)	30
,	16
;	27
n1 = 30	N1 = 253

Таблица 32 – Операнды

Операнды	Число вхождений
MainPage	1
page	6
pageSize	4
minFootage	4
maxFootage	4
minPrice	4
maxPrice	4
minFloor	4
maxFloor	4
isItNew	4

Продолжение таблицы 32

Операнды	Число вхождений
isItOld	4
isForSale	4
isForRent	4
minRooms	4
maxRooms	4
HttpResponseMessage	4
responseForApartments	3
_httpClient	5
_baseUrlsService.apartmentServiceBaseUrl	2
apartmentsWithImage	1
int.MaxValue	3
responseForApartmentAmount	2
_baseUrlsService.rateServiceBaseUrl	1
responseForRates	3
jsonStringForApartments	2
apartments	3
jsonStringForRates	2
rate	3
jsonStringForApartmentsAmount	2
apartmentsAmount	2
HttpContext.Session.GetString	1
JWTToken	1
tokenString	3
jwtToken	1
token	2

Окончание таблицы 32

Операнды	Число вхождений
tokenValue	1
AuthenticationHeaderValue	1
_httpClient.DefaultRequestHeaders.Authorization	1
responseForAuth	2
_baseUrlService.authServiceBaseUrl	1
userRaw	1
JobUser	1
strUser	2
User	2
UserName	1
ApartmentDataViewModel	1
View	1
model	2
n2 = 48	N2 = 123

Исходя из этих данных, можно рассчитать промежуточные значения, используя которые можно получить искомую сложность функции.

Длина функции рассчитывается по формуле:

$$N = N1 + N2 = 253 + 123 = 376,$$

где N1 – число вхождений операторов;

N2 – число вхождений операндов.

Словарь функции рассчитывается по формуле:

$$n = n1 + n2 = 30 + 48 = 78,$$

где n1 – количество операторов;

n2 – количество операндов.

Объем функции рассчитывается по формуле:

$$HV = N * \log_2(n) = 376 * 6.2854 = 2362.77,$$

где N – длина функции;

n – словарь функции.

Потенциальный объём функции:

$$V^* = (2 + n') * \log_2(2 + n') = 50 * \log_2 50 = 282.19,$$

где n' – количество уникальных операндов.

Теоретический уровень программы рассчитывается по формуле:

$$L = V^* / HV = 282.19 / 2362.77 = 0.119,$$

где V^* – потенциальный объём функции;

HV – объём функции.

Сложность функции рассчитывается по формуле:

$$HD = n1/2 * N2/n2 = 30/2 * 123/48 = 38.44,$$

где $n1$ – количество операторов;

$N2$ – число вхождений операндов;

$n2$ – количество операндов.

Исходя из расчетов, было определено, что сложность функции MainPage равна 38.44, что является показателем высокой сложности функции.

Заключение

В результате выполнения выпускной квалификационной работы были спроектированы, разработаны и протестированы микросервисы и графический интерфейс приложения для покупки и аренды недвижимости. Проведенный анализ предметной области позволил сформулировать задачи на проектирование приложения, что стало основой для дальнейшей разработки.

Разработка приложения включала создание трех баз данных, в которых грамотно структурированы сущности, обеспечивающие надежное хранение информации и эффективное взаимодействие между различными компонентами системы. Этот подход позволил достичь высокой производительности и масштабируемости приложения.

Предполагаемый результат заключался в создании полнофункциональной системы, позволяющей пользователям создавать и просматривать объявления о недвижимости, содержащие все необходимые параметры, такие как цена, площадь, тип недвижимости и другие важные характеристики. Пользовательский интерфейс был разработан с учетом удобства и интуитивной понятности, предоставляя подробные фильтры для удобного поиска объектов недвижимости.

Важной частью разработки стало обеспечение высокого уровня безопасности данных пользователей и объявлений, включая использование современных методов аутентификации и авторизации.

Разрабатываемое приложение превосходит прототип Авито.Недвижимость по ряду ключевых критериев:

- интерфейс приложения современный и интуитивно понятный, учитывающий последние тенденции в UX/UI дизайне;
- улучшенные фильтры и возможности поиска позволяют пользователям быстрее находить нужные объекты недвижимости;

- расширенные возможности интеграции с другими сервисами, такими как API Яндекс.Карт обеспечивают высокую функциональность;
- использование новейших технологий и методов обеспечивает безопасность данных.

Результаты тестирования подтвердили правильность работы приложения, показав, что оно функционирует в соответствии с поставленными задачами. Все основные тесты были успешно пройдены, что свидетельствует о надежности и готовности системы к использованию.

Таким образом, предполагаемый результат частично совпал с фактическим. Разработанное приложение не только удобно и функционально для пользователей, но и обеспечивает высокую производительность и безопасность.

Список использованных источников

1. Метод анализа иерархий Т.Саати // Studfile [Электронный ресурс]. URL: <https://studfile.net/preview/9979518/> (дата обращения: 06.06.2024)
2. Базы данных. Методические указания по выполнению курсовой работы // Zenodo [Электронный ресурс]. URL: <https://zenodo.org/records/53822> (дата обращения: 30.05.2024);
3. Диаграмма вариантов использования // Flexberry [Электронный ресурс]. URL: https://flexberry.github.io/ru/fd_usecasediagram.html/ (дата обращения: 30.05.2024);
4. Микросервисы // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/post/249183/> (дата обращения: 06.06.2024);
5. Общая теория систем Лабораторный практикум // Белорусский государственный университет информатики и радиоэлектроники [Электронный ресурс]. URL: https://www.bsuir.by/m/12_101945_1_87685.pdf (дата обращения: 30.05.2024);
6. Разработка структуры программы и модульное программирование // Факультет вычислительной математики и кибернетики Московского государственного университета имени М. В. Ломоносова [Электронный ресурс]. URL: <https://cmcmsu.info/download/programming.technologies.07.pdf> (дата обращения 06.06.2024);
7. Механизм авторизации с использованием технологии JWT // Cyberleninka [Электронный ресурс]. URL: <https://cyberleninka.ru/article/n/mehanizm-avtorizatsii-s-ispolzovaniem-tehnologii-jwt/viewer> (дата обращения: 06.06.2024);
8. Проектирование графического интерфейса пользователя // Хабр [Электронный ресурс]. URL: <https://habr.com/ru/post/208966/> (дата обращения 06.06.2024);

9. Тестирование черного ящика // CoderLessons [Электронный ресурс]. URL: <https://coderlessons.com/tutorials/kachestvo-programmnogo-obespecheniia/ruchnoe-testirovanie/testirovanie-chernogo-iaschchika-2>(дата обращения: 06.06.2024);
10. Тестирование методом черного ящика // Timofeev.ru [Электронный ресурс]. URL: <https://temofeev.ru/info/articles/testirovanie-metodom-chernogo-yashchika/> (дата обращения: 06.06.2024);
11. Общие сведения о программном обеспечении // НОУ ИНТУИТ [Электронный ресурс]. URL: https://intuit.ru/studies/professional_skill_improvements/17031/courses/874/lecture/14289?page=2 (дата обращения: 06.06.2024).

Приложение А
(справочное)
Глоссарий

UML (Unified Modeling Language) – стандартный язык визуального моделирования для описания, спецификации, проектирования и документирования систем программного обеспечения.

HTTP (Hypertext Transfer Protocol) – протокол передачи гипертекста, используемый для обмена данными между веб-браузером и веб-сервером в интернете.

HTML (HyperText Markup Language) – язык разметки гипертекста, используемый для создания и структурирования веб-страниц и их содержимого.

CSS (Cascading Style Sheets) – язык таблиц стилей, используемый для описания внешнего вида и форматирования HTML-документов.

ASP.NET Core – кроссплатформенный фреймворк для создания веб-приложений и веб-сервисов, разработанный компанией Microsoft.

Middleware-компонент – это программный компонент в ASP.NET Core, который обрабатывает HTTP-запросы и ответы. Он может выполнять действия до и после передачи запроса следующему компоненту в конвейере обработки. Middleware часто используется для таких задач, как аутентификация, логирование и обработка ошибок.

Identity Framework – система управления пользователями для ASP.NET, предоставляющая функционал регистрации, аутентификации и управления ролями.

Entity Framework Core – объектно-реляционный отображающий (ORM) фреймворк для .NET, позволяющий разработчикам взаимодействовать с базами данных с помощью .NET объектов.

Web-API – интерфейс программирования приложений для создания HTTP-сервисов, позволяющих взаимодействовать с данными и функциональностью веб-приложений через запросы.

Приложение Б
(рекомендуемое)
Код программы

Листинг Б.1 – Код модуля Program.cs клиентского приложения

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace webapp
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
        }
    }
}

```

Листинг Б.2 – Код модуля Startup.cs клиентского приложения

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using webapp.BaseUrls;

namespace webapp
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
        // services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddHttpContextAccessor();

            services.AddControllersWithViews();

            services.AddHttpClient();
        }
    }
}

```

```

services.AddSingleton<IConfiguration>(Configuration);

services.AddLogging();

services.AddSession(options =>
{
    // Настройки сеанса, например, время жизни
    options.Cookie.Name = ".MyApp.Session";
    options.IdleTimeout = TimeSpan.FromMinutes(30);
    options.Cookie.HttpOnly = false;
    options.Cookie.IsEssential = true;
});

services.AddSingleton<IBaseUrlsService, BaseUrlsService>();
}

public void Configure(IApplicationBuilder app,
IWebHostEnvironment env, ILogger<Startup> logger)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }

    app.UseSession();
    app.UseAuthentication();

    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern:
"{controller=LogReg}/{action=LogReg}/{id?}");
        endpoints.MapControllers();
    });
}
}
}

```

Листинг Б.3 – Код модуля AccountController.cs клиентского приложения

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

```

```

using System.Net.Http.Headers;
using System.Net.Http;
using System.Threading.Tasks;
using Newtonsoft.Json.Linq;
using webapp.Models;
using Newtonsoft.Json;
using System.Collections.Generic;
using webapp.BaseUrls;
using System;
using System.Text;

namespace webapp.Controllers
{
    /// <summary>
    /// Контроллер для действий, связанных с личным кабинетом
    пользователя
    /// Включает в себя методы для вывода информации в представление
    личного кабинета,
    /// а также для добавления и удаления квартир в нем
    /// </summary>
    public class AccountController : Controller
    {
        private readonly HttpClient _httpClient = new HttpClient();

        private readonly IBaseUrlsService _baseUrlsService;

        public AccountController(IBaseUrlsService baseUrlsService)
        {
            _baseUrlsService = baseUrlsService;
        }

        /// <summary>
        /// Метод для вывода данных в представление
        /// </summary>
        /// <returns></returns>
        public async Task<IActionResult> UserProfile()
        {
            var tokenString = HttpContext.Session.GetString("JWTToken");

            if (tokenString == null)
            {
                HttpContext.Session.SetString("EnterProfile", "false");
                return RedirectToAction("LogReg", "LogReg");
            }
            var jwtToken = JObject.Parse(tokenString);

            jwtToken.TryGetValue("token", out var token);

            var tokenValue = token.ToString();

            _httpClient.DefaultRequestHeaders.Authorization = new
            AuthenticationHeaderValue("Bearer", tokenValue);

            HttpResponseMessage responseForAuth = await
            _httpClient.GetAsync($"{_baseUrlsService.authServiceBaseUrl}/api/authenti
            cation/authenticate");

```

```

        if (!responseForAuth.IsSuccessStatusCode)
        {
            return RedirectToAction("LogReg", "LogReg");
        }
        var userRaw =
responseForAuth.Content.ReadAsStringAsync().Result;

        var JObjectUser = JObject.Parse(userRaw);

        var strUser = JObjectUser["user"].ToString();

        HttpResponseMessage responseForApartments = await
_httpClient.GetAsync($"{_baseUrlService.apartmentServiceBaseUrl}/api/Apa
rtments/GetByAddedByString?user={strUser}");

        if (!responseForApartments.IsSuccessStatusCode)
        {
            return BadRequest("failed to load apartments");
        }

        HttpResponseMessage responseForRate = await
_httpClient.GetAsync($"{_baseUrlService.rateServiceBaseUrl}/api/rate/Avg
Rates");

        if (!responseForRate.IsSuccessStatusCode)
        {
            return BadRequest("failed to load rates");
        }
        var jsonStringForApartments = await
responseForApartments.Content.ReadAsStringAsync();

        var apartments =
JsonConvert.DeserializeObject<List<ApartmentsWithImages>>(jsonStringForAp
artments);

        var jsonStringForRate = await
responseForRate.Content.ReadAsStringAsync();

        var rate =
JsonConvert.DeserializeObject<List<AvgRate>>(jsonStringForRate);

        var apartmentsForModel = new ApartmentsWithImagesAndPage
        {
            Apartments = apartments,
            PageIndex = 0,
        };

        var model = new ApartmentDataViewModel
        {
            User = new User()
            {
                UserName = strUser
            },

```

```

        Apartments = apartmentsForModel,
        AvgRate = rate
    };

    return View(model);
}

public async Task<IActionResult>
PostApartmentsMulti(List<IFormFile> files, string name, string
description, string addedBy, int price, int footage, bool typeOfAds, int
floor, bool constructionType, int numberOfRooms)
{
    // Проверка токена JWT
    var tokenString = HttpContext.Session.GetString("JWTToken");
    if (tokenString == null)
    {
        return BadRequest("Invalid token.");
    }

    var token = JObject.Parse(tokenString)["token"].ToString();

    // Установка токена авторизации в HTTP заголовок
    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", token);

    // Создание объекта FormData для отправки файлов и других
данных
    var formData = new MultipartFormDataContent();

    // Добавление файлов в FormData
    foreach (var file in files)
    {
        var fileContent = new
StreamContent(file.OpenReadStream());
        formData.Add(fileContent, "files", file.FileName);
    }

    // Добавление других данных в FormData
    formData.Add(new StringContent(name), "name");
    formData.Add(new StringContent(description), "description");
    formData.Add(new StringContent(addedBy), "addedBy");
    formData.Add(new StringContent(price.ToString()), "price");
    formData.Add(new StringContent(footage.ToString()),
"footage");
    formData.Add(new StringContent(typeOfAds.ToString()),
"typeOfAds");
    formData.Add(new StringContent(floor.ToString()), "floor");
    formData.Add(new StringContent(constructionType.ToString()),
"constructionType");
    formData.Add(new StringContent(numberOfRooms.ToString()),
"numberOfRooms");
    //Добавить дату тут
    var date = DateTime.UtcNow;

```

```

        formData.Add(new StringContent(date.ToString()),
"dateOfPost");
        // Отправка POST запроса с использованием FormData
        var response = await
_httpClient.PostAsync($"{_baseUrlService.apartmentServiceBaseUrl}/api/Ap
artments/postMany", formData);

        // Обработка ответа
        if (response.IsSuccessStatusCode)
        {
            HttpContext.Session.SetString("ApartmentAdded", "true");
            return RedirectToAction("UserProfile", "Account");
        }
        else
        {
            var responseBody = await
response.Content.ReadAsStringAsync();
            return BadRequest(responseBody);
        }
    }

    /// <summary>
    /// Метод для удаления квартир пользователя
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    public async Task<IActionResult> DeleteApartment(int id)
    {
        var tokenString = HttpContext.Session.GetString("JWTToken");

        if (tokenString != null)
        {
            var jwtToken = JObject.Parse(tokenString);

            jwtToken.TryGetValue("token", out var token);

            var tokenValue = token.ToString();

            _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);
        }
        else
        {
            return BadRequest("wrong token when delete");
        }

        // URL сервера, где находится API для удаления квартиры
        var url =
$"{_baseUrlService.apartmentServiceBaseUrl}/api/apartments/{id}";

        var response = await _httpClient.DeleteAsync(url);

        if (response.IsSuccessStatusCode)
        {
            HttpContext.Session.SetString("deleted", "true");
            //Возврат представления личного кабинета

```

```

        return RedirectToAction("UserProfile", "Account");
    }
    else
    {
        return BadRequest("server return code unseccessful,
apartment is not deleted");
    }
}
}
}

```

Листинг Б.4 – Код модуля ApartmentPageController.cs клиентского приложения

```

using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using System.Collections.Generic;
using System.Net.Http;
using System.Threading.Tasks;
using webapp.BaseUrls;
using webapp.Models;

namespace webapp.Controllers
{
    public class ApartmentPageController : Controller
    {
        private readonly IBaseUrlsService _baseUrlsService;

        public ApartmentPageController(IBaseUrlsService baseUrlsService)
        {
            _baseUrlsService = baseUrlsService;
        }

        public async Task<IActionResult> ApartmentPage(int id, string
imagePath)
        {
            HttpClient httpClient = new HttpClient();

            HttpResponseMessage responseForRates = await
httpClient.GetAsync($"{_baseUrlsService.rateServiceBaseUrl}/api/rate/AvgR
ates");

            if (!responseForRates.IsSuccessStatusCode)
            {
                return BadRequest("Не удалось загрузить оценки
объявлений");
            }
            var jsonStringForRates = await
responseForRates.Content.ReadAsStringAsync();

            var rate =
JsonConvert.DeserializeObject<List<AvgRate>>(jsonStringForRates);

            HttpResponseMessage responseForApartment = await
httpClient.GetAsync($"{_baseUrlsService.apartmentServiceBaseUrl}/api/Apar
tments/{id}");

```

```

        var jsonResponseForApartment = await
responseForApartment.Content.ReadAsStringAsync();

        var apartment =
JsonConvert.DeserializeObject<Apartment>(jsonResponseForApartment);

        var apartment1 = new ApartmentPageViewModel
        {
            Id = apartment.Id,
            Price = apartment.Price,
            Footage = apartment.Footage,
            TypeOfAds = apartment.TypeOfAds,
            Description = apartment.Description,
            Name = apartment.Name,
            date = apartment.date,
            AddedBy = apartment.AddedBy,
            Floor = apartment.Floor,
            ConstructionType = apartment.ConstructionType,
            NumberOfRooms = apartment.NumberOfRooms,
            AvgRate = rate,
            ImagePath = imagePath
            //Вот тут добавить среднюю оценку
        };

        HttpResponseMessage responseForImages = await
httpClient.GetAsync($"{_baseUrlService.apartmentServiceBaseUrl}/api/Apar
tments/GetApartmentImagesByFilePath?filePath={imagePath}");
        //Если картинки были найдены, добавить к модели картинки
        if (responseForImages.IsSuccessStatusCode)
        {
            var jsonResponseForImages = await
responseForImages.Content.ReadAsStringAsync();
            var _images =
JsonConvert.DeserializeObject<List<byte[]>>(jsonResponseForImages);
            apartment1.images = _images;
        }
        return View(apartment1);
    }
}
}

```

Листинг Б.5 – Код модуля LogRegController.cs клиентского приложения

```

using Microsoft.AspNetCore.Mvc;
using System.Net.Http;
using System.Threading.Tasks;
using Microsoft.Extensions.Configuration;
using System.Net.Http.Json;
using Microsoft.AspNetCore.Http;
using webapp.BaseUrls;

namespace ClientApp.Controllers
{
    /// <summary>
    /// Контроллер для логина и регистрации пользователя
    /// </summary>

```

```

public class LogRegController : Controller
{
    /// <summary>
    /// Переменная Http-клиента
    /// </summary>
    private readonly HttpClient _httpClient;

    private readonly IBaseUrlsService _baseUrlsService;

    /// <summary>
    /// Конструктор контроллера
    /// </summary>
    /// <param name="httpClient"></param>
    /// <param name="configuration"></param>
    public LogRegController(HttpClient httpClient, IConfiguration
configuration, IBaseUrlsService baseUrlsService)
    {
        _httpClient = httpClient;
        _baseUrlsService = baseUrlsService;
    }

    /// <summary>
    /// Метод для возврата представления страницы логина и
регистрации
    /// </summary>
    /// <returns></returns>
    public IActionResult LogReg()
    {
        return View();
    }

    /// <summary>
    /// Метод для регистрации пользователя в системе
    /// </summary>
    /// <param name="username"></param>
    /// <param name="email"></param>
    /// <param name="password"></param>
    /// <param name="confirmPassword"></param>
    /// <returns></returns>
    public async Task<IActionResult> RegisterUser(string username,
string email, string password, string confirmPassword)
    {
        //TempData["Registered"] = false;
        var registrationModel = new { Username = username, Email =
email, Password = password, ConfirmPassword = confirmPassword };
        var response = await
_httpClient.PostAsJsonAsync($"{_baseUrlsService.authServiceBaseUrl}/api/r
egister", registrationModel);

        if (response.IsSuccessStatusCode)
        {
            var token = await response.Content.ReadAsStringAsync();

            HttpContext.Session.SetString("JWTToken", token);

            HttpContext.Session.SetString("registered", "true");

```

```

        return RedirectToAction("MainPage", "MainPage");
    }
    else
    {
        return BadRequest("Can't register user");
    }
}

/// <summary>
/// Метод для логина пользователя в системе
/// </summary>
/// <param name="username"></param>
/// <param name="password"></param>
/// <returns></returns>
public async Task<IActionResult> LoginUser(string username,
string password)
{
    var loginModel = new { Username = username, Password =
password };

    HttpResponseMessage response = await
_httpClient.PostAsJsonAsync($"{_baseUrlService.authServiceBaseUrl}/api/login", loginModel);

    if (response.IsSuccessStatusCode)
    {
        var token = await response.Content.ReadAsStringAsync();

        HttpContext.Session.SetString("JWTToken", token);

        HttpContext.Session.SetString("logged", "true");
        // Перенаправление на главную страницу
        return RedirectToAction("MainPage", "MainPage");
    }
    else
    {
        if (System.Net.HttpStatusCode.BadRequest ==
response.StatusCode)
        {
            return BadRequest("Invalid login or password");
        }
        else {
            return BadRequest("Probably, apartment service is
off");
        }
    }
}

public async Task<IActionResult> LogoutUser()
{
    HttpContext.Session.Clear();
    HttpContext.Session.SetString("logout", "true");
    return RedirectToAction("LogReg", "LogReg");
}
}

```

```
}
```

Листинг Б.6 – Код модуля MainPageController.cs клиентского приложения

```
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;
using System.Collections.Generic;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;
using webapp.BaseUrls;
using webapp.Models;

namespace webapp.Controllers
{
    /// <summary>
    /// Контроллер для главной страницы сайта
    /// Включает в себя метод для отображения информации в представлении
    /// </summary>
    public class MainPageController : Controller
    {

        private readonly HttpClient _httpClient = new HttpClient();

        private readonly IBaseUrlsService _baseUrlsService;

        public MainPageController(IBaseUrlsService baseUrlsService)
        {
            _baseUrlsService = baseUrlsService;
        }
        /// <summary>
        /// Метод для отображения представления
        /// </summary>
        /// <returns></returns>
        public async Task<IActionResult> MainPage(
            string location,
            int page = 1,
            int pageSize = 12,
            int minFootage = 0,
            int maxFootage = int.MaxValue,
            int minPrice = 0,
            int maxPrice = int.MaxValue,
            int minFloor = 0,
            int maxFloor = int.MaxValue,
            bool isItNew = false,
            bool isItOld = false,
            bool isForSale = false,
            bool isForRent = false,
            int minRooms = 0,
            int maxRooms = int.MaxValue)
        {
            var queryParameters = new List<string>
            {
                $"page={page}",
            }
        }
    }
}
```

```

        $"pageSize={pageSize}",
        $"minPrice={minPrice}",
        $"maxPrice={maxPrice}",
        $"minFootage={minFootage}",
        $"maxFootage={maxFootage}",
        $"minFloor={minFloor}",
        $"maxFloor={maxFloor}",
        $"isItNew={isItNew}",
        $"isItOld={isItOld}",
        $"isForSale={isForSale}",
        $"isForRent={isForRent}",
        $"minRooms={minRooms}",
        $"maxRooms={maxRooms}"
    };

    if (!string.IsNullOrEmpty(location))
    {
        queryParameters.Add($"location={location}");
    }

    var queryString = string.Join("&", queryParameters);

    HttpResponseMessage responseForApartments = await
        _httpClient.GetAsync($"{{_baseUrlService.apartmentServiceBaseUrl}}/api/Apartments/apartmentsWithImage?{queryString}");

    if (!responseForApartments.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить объявления");
    }

    HttpResponseMessage responseForApartmentsAmount = await
        _httpClient.GetAsync($"{{_baseUrlService.apartmentServiceBaseUrl}}/api/Apartments/AmountOfApartments?{queryString}");

    if (!responseForApartmentsAmount.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить данные о количестве объявлений");
    }

    HttpResponseMessage responseForRates = await
        _httpClient.GetAsync($"{{_baseUrlService.rateServiceBaseUrl}}/api/rate/AvgRates");

    if (!responseForRates.IsSuccessStatusCode)
    {
        return BadRequest("Не удалось загрузить оценки объявлений");
    }

    var jsonStringForApartments = await
        responseForApartments.Content.ReadAsStringAsync();

```

```

        var apartments =
JsonConvert.DeserializeObject<ApartmentsWithImagesAndPage>(jsonStringForA
partments);

        var jsonStringForRates = await
responseForRates.Content.ReadAsStringAsync();

        var rate =
JsonConvert.DeserializeObject<List<AvgRate>>(jsonStringForRates);

        var jsonStringForApartmentsAmount = await
responseForApartmentsAmount.Content.ReadAsStringAsync();

        var apartmentsAmount =
JsonConvert.DeserializeObject<int>(jsonStringForApartmentsAmount);

        var tokenString = HttpContext.Session.GetString("JWTToken");

        string strUser = "";

        if (tokenString != null)
        {
            var jwtToken = JObject.Parse(tokenString);

            jwtToken.TryGetValue("token", out var token);

            var tokenValue = token.ToString();

            _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);

            HttpResponseMessage responseForAuth = await
_httpClient.GetAsync($"{_baseUrlService.authServiceBaseUrl}/api/authenti
cation/authenticate");

            if (responseForAuth.IsSuccessStatusCode)
            {
                var userRaw =
responseForAuth.Content.ReadAsStringAsync().Result;

                var JObjectUser = JObject.Parse(userRaw);

                strUser = JObjectUser["user"].ToString();
            }
        }

        var model = new ApartmentDataViewModel
        {
            User = new User
            {
                UserName = strUser
            },
            Apartments = apartments,
            AvgRate = rate,
            NumberOfApartments = apartmentsAmount,

```

```

        Location = location,
        CurrentPage = page,
        MinPrice = minPrice,
        MaxPrice = maxPrice,
        MinFloor = minFloor,
        MaxFloor = maxFloor,
        MinFootage = minFootage,
        MaxFootage = maxFootage,
        MinRooms = minRooms,
        MaxRooms = maxRooms,
        IsForSale = isForSale,
        IsForRent = isForRent,
        IsItNew = isItNew,
        IsItOld = isItOld
    };

    return View(model);
}
}
}

```

Листинг Б.7 – Код модуля RateController.cs клиентского приложения

```

using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json.Linq;
using System.Net.Http.Headers;
using System.Net.Http;
using System.Threading.Tasks;
using System.Net.Http.Json;
using webapp.BaseUrls;

namespace webapp.Controllers
{
    /// <summary>
    /// Контроллер для оценки квартиры на главной странице сайта
    /// </summary>
    public class RateController : Controller
    {
        private readonly HttpClient _httpClient = new HttpClient();

        private readonly IBaseUrlsService _baseUrlsService;

        public RateController(IBaseUrlsService baseUrlsService)
        {
            _baseUrlsService = baseUrlsService;
        }
        /// <summary>
        /// Метод для отправки оценки для квартиры
        /// </summary>
        /// <returns></returns>
        public async Task<IActionResult> RateApartment(int apartmentId,
int rate, string imagePath)
        {
            var tokenString = HttpContext.Session.GetString("JWTToken");

```

```

        if (tokenString == null)
        {
            HttpContext.Session.SetString("RateAccess", "false");
            return RedirectToAction("LogReg", "LogReg");
        }
        var jwtToken = JObject.Parse(tokenString);

        jwtToken.TryGetValue("token", out var token);

        var tokenValue = token.ToString();

        _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);

        HttpResponseMessage response = await
_httpClient.GetAsync($"{_baseUrlService.authServiceBaseUrl}/api/authenti
cation/authenticate");

        if (!response.IsSuccessStatusCode)
        {
            HttpContext.Session.SetString("RateAccess", "false");
            return RedirectToAction("LogReg", "LogReg");
        }
        var userRaw = response.Content.ReadAsStringAsync().Result;

        var JObjectUser = JObject.Parse(userRaw);

        var strUser = JObjectUser["user"].ToString();

        var rateModel = new { ApartmentId = apartmentId, UserName =
strUser, Rate = rate };

        _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenValue);

        HttpResponseMessage responseClient = await
_httpClient.PostAsJsonAsync($"{_baseUrlService.rateServiceBaseUrl}/api/r
ate", rateModel);

        if (responseClient.IsSuccessStatusCode)
        {
            HttpContext.Session.SetString("RateStatus", "true");
            return RedirectToAction("ApartmentPage", "ApartmentPage",
new { id = apartmentId, imagePath = imagePath });
        }
        else
        {
            var responseClientBody =
responseClient.Content.ReadAsStringAsync().Result;
            return BadRequest(responseClientBody);
        }
    }
}

```

Листинг Б.8 – Код модуля Apartment.cs клиентского приложения, содержащий модель Apartment

```

using System;
using System.ComponentModel.DataAnnotations;

namespace webapp.Models
{
    /// <summary>
    /// Модель данных, представляющая квартиру в системе
    /// Включает в себя id квартиры, Название(адрес), Описание и имя
    /// пользователя, добавившего ее
    /// </summary>
    public class Apartment
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Пожалуйста, укажите название
квартиры.")]
        [StringLength(100, MinimumLength = 3, ErrorMessage = "Название
квартиры должно быть от 3 до 100 символов.")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Пожалуйста, добавьте описание
квартиры.")]
        public string Description { get; set; }

        [Required(ErrorMessage = "Пожалуйста, укажите имя пользователя,
который добавил квартиру.")]
        [StringLength(50, MinimumLength = 2, ErrorMessage = "Имя
пользователя должно быть от 3 до 50 символов.")]
        public string AddedBy { get; set; }

        //Новые свойства квартиры
        [Required(ErrorMessage = "Укажите площадь квартиры")]
        [Range(0, int.MaxValue, ErrorMessage = "Значение должно быть
неотрицательным")]
        public int Footage { get; set; }

        [Required(ErrorMessage = "Выберите тип объявления(0 - Сдача, 1 -
Продажа)")]
        public bool TypeOfAds { get; set; }

        [Required(ErrorMessage = "Укажите цену квартиры")]
        [Range(0, int.MaxValue, ErrorMessage = "Значение должно быть
неотрицательным")]
        public int Price { get; set; }

        public DateTime date { get; set; }
        public int Floor { get; set; }
        public bool ConstructionType { get; set; }
        public int NumberOfRooms { get; set; }
    }
}

```

Листинг Б.9 – Код модуля ApartmentDataViewModel.cs клиентского приложения, содержащий модель ApartmentDataViewModel

```
using System.Collections.Generic;

namespace webapp.Models
{
    /// <summary>
    /// Модель данных для представления главной страницы сайта
    /// Включает в себя список квартир и список средних оценок на
    /// квартиры
    /// </summary>
    public class ApartmentDataViewModel
    {
        public User User { get; set; }
        public ApartmentsWithImagesAndPage Apartments { get; set; }

        public List<AvgRate> AvgRate { get; set; }

        public int NumberOfApartments { get; set; }

        public int CurrentPage { get; set; }
        public string Location { get; set; }
        public int MinPrice { get; set; }
        public int MaxPrice { get; set; }
        public int MinFootage { get; set; }
        public int MaxFootage { get; set; }
        public int MinFloor { get; set; }
        public int MaxFloor { get; set; }
        public bool IsItNew { get; set; }
        public bool IsItOld { get; set; }
        public bool IsForSale { get; set; }
        public bool IsForRent { get; set; }
        public int MinRooms { get; set; }
        public int MaxRooms { get; set; }

    }
}
```

Листинг Б.10 – Код модуля ApartmentPageViewModel.cs клиентского приложения, содержащий модель ApartmentPageViewModel

```
using System.Collections.Generic;

namespace webapp.Models
{
    public class ApartmentPageViewModel:Apartment
    {
        public List<byte[]> images { get; set; }

        public List<AvgRate> AvgRate { get; set; }

        public string ImagePath { get; set; }
    }
}
```

```
}
```

Листинг Б.11 – Код модуля ApartmentsWithImages.cs клиентского приложения, содержащий модель ApartmentsWithImages

```
using System;

namespace webapp.Models
{
    public class ApartmentsWithImages
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string AddedBy { get; set; }
        public int Footage { get; set; }
        public bool TypeOfAds { get; set; }
        public int Price { get; set; }
        public DateTime date { get; set; }
        public string ImagePath { get; set; }
        public string ImageData { get; set; }
        public int Floor { get; set; }
        public bool ConstructionType { get; set; }
        public int NumberOfRooms { get; set; }
    }
}
```

Листинг Б.12 – Код модуля ApartmentsWithImagesAndPage.cs клиентского приложения, содержащий модель ApartmentWithImagesAndPage

```
using System.Collections.Generic;

namespace webapp.Models
{
    public class ApartmentsWithImagesAndPage
    {
        public List<ApartmentsWithImages> Apartments { get; set; }
        public int PageIndex { get; set; }
    }
}
```

Листинг Б.13 – Код модуля AvgRate.cs клиентского приложения, содержащий модель AvgRate

```
namespace webapp.Models
{
    /// <summary>
    /// Модель данных, представляющая средний рейтинг апартаментов.
    /// Содержит идентификатор апартаментов и их средний рейтинг.
    /// </summary>
    public class AvgRate
    {
        public int ApartmentID { get; set; }
    }
}
```

```

        public float AverageRate { get; set; }
    }
}

```

Листинг Б.14 – Код модуля ErrorViewModel.cs клиентского приложения, содержащий модель ErrorViewModel

```

namespace webapp.Models
{
    /// <summary>
    /// Стандартная модель данных, используемая для передачи информации
    об ошибках в представления.
    /// Включает идентификатор запроса, который может быть использован
    для отслеживания или логирования ошибок.
    /// </summary>
    public class ErrorViewModel
    {
        public string RequestId { get; set; }

        public bool ShowRequestId => !string.IsNullOrEmpty(RequestId);
    }
}

```

Листинг Б.15 – Код модуля User.cs клиентского приложения, содержащий модель User

```

namespace webapp.Models
{
    /// <summary>
    /// Модель данных, представляющая пользователя
    /// Включает в себя имя пользователя
    /// </summary>
    public class User
    {
        public string UserName { get; set; }
    }
}

```

Листинг Б.16 – Код модуля BaseUrls.cs клиентского приложения

```

namespace webapp.BaseUrls
{
    public class BaseUrls
    {
        public const string apartmentServiceBaseUrl =
"https://localhost:44361";
        public const string rateServiceBaseUrl =
"https://localhost:44347";
        public const string authServiceBaseUrl =
"https://localhost:44326";
    }
}

```

Листинг Б.17 – Код модуля BaseUrlsService.cs клиентского приложения

```

namespace webapp.BaseUrls
{
    public class BaseUrlsService : IBaseUrlsService
    {
        public string apartmentServiceBaseUrl =>
BaseUrls.apartmentServiceBaseUrl;
        public string rateServiceBaseUrl => BaseUrls.rateServiceBaseUrl;
        public string authServiceBaseUrl => BaseUrls.authServiceBaseUrl;
    }
}

```

Листинг Б.18 – Код модуля IBaseUrlsService.cs клиентского приложения

```

namespace webapp.BaseUrls
{
    public interface IBaseUrlsService
    {
        string apartmentServiceBaseUrl { get; }
        string rateServiceBaseUrl { get; }
        string authServiceBaseUrl { get; }
    }
}

```

Листинг Б.19 – Код модуля Program.cs микросервиса недвижимости

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace ApartmentService
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
        }
    }
}

```

Листинг Б.20 – Код модуля Startup.cs микросервиса недвижимости

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;

```

```

namespace ApartmentService
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
        services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title =
                "ApartmentService", Version = "v1" });
            });

            services.AddDbContext<ApartmentDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection
")));

            services.AddControllers();
        }
        public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
                app.UseSwagger();
                app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "ApartmentService v1"));
            }

            app.UseHttpsRedirection();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }
    }
}

```

Листинг Б.21 – Код модуля ApartmentController.cs микросервиса недвижимости

```

using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http.Headers;
using System.Net.Http;
using System.Threading.Tasks;
using Newtonsoft.Json.Linq;
using ApartmentService.Models;
using Microsoft.AspNetCore.Http;
using System.IO;
using System;

namespace ApartmentService.Controllers
{
    /// <summary>
    /// Контроллер для операций, связанных с апартаментами
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class ApartmentsController : ControllerBase
    {
        /// <summary>
        /// Экземпляр Http-клиента для создания Http-запроса
        /// </summary>
        private readonly HttpClient _httpClient = new HttpClient();

        /// <summary>
        /// Поле для представления контекста базы данных для управления
        /// данными, связанных с апартаментами в контроллере
        /// </summary>
        private readonly ApartmentDbContext _context;

        private readonly string _authServiceBaseUrl =
            "https://localhost:44326";

        /// <summary>
        /// Конструктор контроллера
        /// </summary>
        /// <param name="context"></param>
        public ApartmentsController(ApartmentDbContext context)
        {
            _context = context;
        }

        /// <summary>
        /// Get-метод для получения квартир из базы данных
        /// </summary>
        /// <returns></returns>
        [HttpGet]
        public async Task<ActionResult<IEnumerable<Apartment>>>
            GetApartments()
        {
            return await _context.Apartments.ToListAsync();
        }
    }
}

```

```

[HttpGet("ApartmentsWithImage")]
public async
Task<ActionResult<IEnumerable<ApartmentsWithImagesAndPageAndFilters>>>
GetApartmentsWithImages(
    string location = "",
    int page = 1,
    int pageSize = 12,
    int minFootage = 0,
    int maxFootage = int.MaxValue,
    int minPrice = 0,
    int maxPrice = int.MaxValue,
    int minFloor = 0,
    int maxFloor = int.MaxValue,
    bool isItNew = false,
    bool isItOld = false,
    bool isForSale = false,
    bool isForRent = false,
    int minRooms = 0,
    int maxRooms = int.MaxValue)
{
    var query = _context.Apartments
        .Where(apartment =>
            apartment.Name.Contains(location) &&
            apartment.Footage >= minFootage &&
            apartment.Footage <= maxFootage &&
            apartment.Price >= minPrice &&
            apartment.Price <= maxPrice &&
            apartment.Floor >= minFloor &&
            apartment.Floor <= maxFloor &&
            (!isItNew || apartment.ConstructionType) &&
            (!isItOld || !apartment.ConstructionType) &&
            (!isForSale || apartment.TypeOfAds) &&
            (!isForRent || !apartment.TypeOfAds) &&
            apartment.NumberOfRooms >= minRooms &&
            apartment.NumberOfRooms <= maxRooms);

    var result = query
        .Select(apartment => new ApartmentWithImage
        {
            Id = apartment.Id,
            Name = apartment.Name,
            Description = apartment.Description,
            AddedBy = apartment.AddedBy,
            Footage = apartment.Footage,
            TypeOfAds = apartment.TypeOfAds,
            Price = apartment.Price,
            DateOfPost = apartment.DateOfPost,
            Floor = apartment.Floor,
            ConstructionType = apartment.ConstructionType,
            NumberOfRooms = apartment.NumberOfRooms,
            ImagePath = _context.ApartmentsImage
                .Where(image => image.ApartmentId ==
                    apartment.Id)
                .Select(image => image.ImagePath)
        })

```

```

        .FirstOrDefault(),
        ImageData = ""
    })
    .Skip((page - 1) * pageSize)
    .Take(pageSize)
    .ToList();

    foreach (var item in result)
    {
        if (item.ImagePath != null)
        {
            var imageBytes = await
System.IO.File.ReadAllBytesAsync(item.ImagePath);
            var imageBase64 = Convert.ToBase64String(imageBytes);
            item.ImageData =
$"data:image/jpeg;base64,{imageBase64}";
        }
    }

    var apartmentsAndPage = new
ApartmentsWithImagesAndPageAndFilters
    {
        Apartments = result,
        PageIndex = page,
        Location = location,
        MinPrice = minPrice,
        MaxPrice = maxPrice,
        MinFloor = minFloor,
        MaxFloor = maxFloor,
        MinFootage = minFootage,
        MaxFootage = maxFootage,
        IsItOld = isItOld,
        IsItNew = isItNew,
        IsForRent = isForRent,
        IsForSale = isForSale,
        MinRooms = minRooms,
        MaxRooms = maxRooms,
    };

    return Ok(apartmentsAndPage);
}

[HttpGet("AmountOfApartments")]
public async Task<ActionResult<int>> GetAmount(
    string location = "",
    int minFootage = 0,
    int maxFootage = int.MaxValue,
    int minPrice = 0,
    int maxPrice = int.MaxValue,
    int minFloor = 0,
    int maxFloor = int.MaxValue,
    bool isItNew = false,
    bool isItOld = false,
    bool isForSale = false,
    bool isForRent = false,
    int minRooms = 0,

```

```

        int maxRooms = int.MaxValue)
    {
        var query = _contextApartments
            .Where(apartment =>
                apartment.Name.Contains(location) &&
                apartment.Footage >= minFootage &&
                apartment.Footage <= maxFootage &&
                apartment.Price >= minPrice &&
                apartment.Price <= maxPrice &&
                apartment.Floor >= minFloor &&
                apartment.Floor <= maxFloor &&
                (!isItNew || apartment.ConstructionType) &&
                (!isItOld || !apartment.ConstructionType) &&
                (!isForSale || apartment.TypeOfAds) &&
                (!isForRent || !apartment.TypeOfAds) &&
                apartment.NumberOfRooms >= minRooms &&
                apartment.NumberOfRooms <= maxRooms)
            .Count();
        return Ok(query);
    }

    [HttpPost("postMany")]
    public async Task<ActionResult<Apartment>>
        NewPost(List<IFormFile> files, [FromForm] Apartment apartment)
    {
        var strUser = "";

        var token =
            Request.Headers["Authorization"].FirstOrDefault();

        if (string.IsNullOrEmpty(token))
        {
            return Unauthorized("Token is missing");
        }

        var tokenWithoutBearer = token.Substring(7);

        _httpClient.DefaultRequestHeaders.Authorization = new
            AuthenticationHeaderValue("Bearer", tokenWithoutBearer);

        HttpResponseMessage response = await
            _httpClient.GetAsync($"{_authServiceBaseUrl}/api/authentication/authentic
            ate");

        if (!response.IsSuccessStatusCode)
        {
            return Unauthorized(new {message = "Токен не
            соответствует"});
        }

        var userRaw = response.Content.ReadAsStringAsync().Result;

        var JObjectUser = JObject.Parse(userRaw);

        strUser = JObjectUser["user"].ToString();
    }

```

```

        if (strUser != apartment.AddedBy)
        {
            return Unauthorized(new { message = "Отказано в
доступе" });
        }

        if (files == null || files.Count == 0)
        {
            //Сохранение информации об апартаменте в базе данных
            _contextApartments.Add(apartment);

            await _context.SaveChangesAsync();

            return Ok(new { message = "Добавлено без картинок" });
        }
        else
        {
            //Сохранение информации об апартаменте в базе данных
            _contextApartments.Add(apartment);

            await _context.SaveChangesAsync();

            //OrdinalIgnoreCase позволяет проводить проверку без
учета регистра
            var allowedExtensions = new
HashSet<string>(StringComparer.OrdinalIgnoreCase)
            {
                ".jpg",
                ".jpeg",
                ".png",
                ".gif",
                ".bmp"
            };

            foreach (var file in files)
            {
                if (file.Length > 0)
                {
                    //Проверка на расширение файла
                    string ext = Path.GetExtension(file.FileName);

                    if (!allowedExtensions.Contains(ext))
                    {
                        return BadRequest(new { message = "Неверное
расширение файла" });
                    }

                    //Генерация уникального имени файла
                    var fileName = Guid.NewGuid().ToString() +
Path.GetExtension(file.FileName);

                    //Путь для сохранения файла
                    var filePath =
Path.Combine(Directory.GetCurrentDirectory(), "Images", fileName);

                    //Сохранение файла на сервере

```

```

        using (var stream = new FileStream(filePath,
        FileMode.Create))
        {
            await file.CopyToAsync(stream);
        }

        //Сохранение информации о файле в базу данных
        var apartmentImage = new ApartmentImages
        {
            ApartmentId = apartment.Id,
            ImagePath = filePath
        };
        _context.ApartmentsImage.Add(apartmentImage);
    }
    await _context.SaveChangesAsync();

    return Ok(new { message = "Добавлено с файлами" });
}

[HttpGet("GetApartmentImagesByFilePath")]
public async Task<ActionResult<IEnumerable<ApartmentImages>>>
GetApartmentImagesByFilePath(string filePath)
{
    //Найти запись ApartmentImage по пути файла
    var apartmentImage = await
    _context.ApartmentsImage.FirstOrDefaultAsync(ai => ai.ImagePath ==
    filePath);

    if (apartmentImage == null)
    {
        return NotFound("Изображение с указанным путем не
        найдено");
    }

    //Получить ApartmentId из найденной записи ApartmentImage
    var apartmentId = apartmentImage.ApartmentId;

    //Найти все записи ApartmentImage, соответствующие этому
    ApartmentId
    var apartmentImages = await _context.ApartmentsImage
        .Where(ai => ai.ApartmentId == apartmentId)
        .ToListAsync();

    if (apartmentImages == null || apartmentImages.Count == 0)
    {
        return NotFound("Изображения для данного ApartmentId не
        найдены");
    }

    //Создать список путей к изображениям
    var imagePaths = apartmentImages.Select(ai => ai.ImagePath);

    //Создать список для хранения изображений
    var images = new List<byte[]>();

```

```

        //Загрузить изображения из файлов и добавить их в список
        foreach (var path in imagePaths)
        {
            try
            {
                var imageBytes = await
System.IO.File.ReadAllBytesAsync(path);
                images.Add(imageBytes);
            }
            catch (Exception ex)
            {
                //Обработка ошибок чтения файла
                //Замените этот блок кодом, соответствующим вашим
потребностям
                return BadRequest($"Ошибка чтения файла изображения:
{ex.Message}");
            }
        }

        return Ok(images);
    }

    /// <summary>
    /// Get-Метод для получения квартиры по ее номеру в БД
    /// </summary>
    /// <param name="id"></param>
    /// <returns></returns>
    [HttpGet("{id}")]
    public async Task<ActionResult<Apartment>> GetApartment(int id)
    {
        var apartment = await _context.Apartments.FindAsync(id);

        if (apartment == null)
        {
            return NotFound();
        }

        return apartment;
    }

    /// <summary>
    /// Get-Метод для получения квартир, добавленных определенным
пользователем
    /// </summary>
    /// <param name="user"></param>
    /// <returns></returns>
    [HttpGet("GetByAddedBy{addedBy}")]
    public async Task<ActionResult<ApartmentWithImage>>
GetApartmentBy(string user)
    {
        var apartments = await _context.Apartments.
            Select(apartment => new ApartmentWithImage
            {
                Id = apartment.Id,
                Name = apartment.Name,

```

```

        Description = apartment.Description,
        AddedBy = apartment.AddedBy,
        Footage = apartment.Footage,
        TypeOfAds = apartment.TypeOfAds,
        Price = apartment.Price,
        DateOfPost = apartment.DateOfPost,
        Floor = apartment.Floor,
        ConstructionType = apartment.ConstructionType,
        NumberOfRooms = apartment.NumberOfRooms,
        ImagePath = _context.ApartmentsImage
            .Where(image => image.ApartmentId ==
apartment.Id)
            .Select(image => image.ImagePath)
            .FirstOrDefault(),
        ImageData = ""
    })).
    Where(a => a.AddedBy == user).
    ToListAsync();

    return Ok(apartments);
}

/// <summary>
/// Delete-метод для удаления квартиры по id
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
[HttpDelete("{id}")]
public async Task<IActionResult> DeleteApartment(int id)
{
    var strUser = "";

    var token =
Request.Headers["Authorization"].FirstOrDefault();

    if (string.IsNullOrEmpty(token))
    {
        return Unauthorized("Token is missing");
    }

    var tokenWithoutBearer = token.Substring(7);

    _httpClient.DefaultRequestHeaders.Authorization = new
AuthenticationHeaderValue("Bearer", tokenWithoutBearer);

    HttpResponseMessage response = await
_httpClient.GetAsync($"{_authServiceBaseUrl}/api/authentication/authentic
ate");

    if (response.IsSuccessStatusCode)
    {
        var userRaw =
response.Content.ReadAsStringAsync().Result;

        var JObjectUser = JObject.Parse(userRaw);

```

```

        strUser = JObjectUser["user"].ToString();

        //Тут необходимо включить проверку токена и удаляемого
значения
        var addedBy = _context.Apartments.Where(u=> u.Id ==
id).FirstOrDefault();

        if (addedBy.AddedBy == strUser)
        {
            var apartment = await
_context.Apartments.FindAsync(id);
            if (apartment == null)
            {
                return BadRequest("Apartment not found");
            }

            _context.Apartments.Remove(apartment);

            await _context.SaveChangesAsync();

            //Возвращает ответ 204 No Content, указывая, что
операция прошла успешно, но не возвращает тело ответа.
            return NoContent();
        }
        else
        {
            return Unauthorized(new {message = "Отказано в
доступе"});
        }
    }
    else
    {
        return Unauthorized(new {message = "Токен не
соответствует"});
    }
}
}

```

Листинг Б.22 – Код модуля DbContext.cs микросервиса недвижимости

```

using ApartmentService.Models;
using Microsoft.EntityFrameworkCore;

namespace ApartmentService
{
    /// <summary>
    /// Контекст базы данных для работы с квартирами. Включает набор
DbSet для доступа к данным о квартирах.
    /// </summary>
    public class ApartmentDbContext : DbContext
    {
        public DbSet<Apartment> Apartments { get; set; }
        public DbSet<ApartmentImages> ApartmentsImage { get; set; }
    }
}

```

```

        public ApartmentDbContext(DbContextOptions<ApartmentDbContext>
options)
        : base(options)
        {
        }
        protected override void OnModelCreating(ModelBuilder
modelBuilder)
        {
        }
    }
}

```

Листинг Б.23 – Код модуля Apartment.cs микросервиса недвижимости, содержащий модель Apartment

```

using Newtonsoft.Json;
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Text.Json.Serialization;

namespace ApartmentService.Models
{
    /// <summary>
    /// Модель данных, представляющая квартиру в системе.
    /// Включает основные атрибуты квартиры, такие как название, описание
и информацию о том, кем была добавлена.
    /// </summary>
    [Table("Apartments")]
    public class Apartment
    {
        public int Id { get; set; }

        [Required(ErrorMessage = "Пожалуйста, укажите название
квартиры.")]
        [StringLength(100, MinimumLength = 3, ErrorMessage = "Название
квартиры должно быть от 3 до 100 символов.")]
        public string Name { get; set; }

        [Required(ErrorMessage = "Пожалуйста, добавьте описание
квартиры.")]
        public string Description { get; set; }

        [Required(ErrorMessage = "Пожалуйста, укажите имя пользователя,
который добавил квартиру.")]
        [StringLength(50, MinimumLength = 2, ErrorMessage = "Имя
пользователя должно быть от 3 до 50 символов.")]
        public string AddedBy { get; set; }

        //Новые свойства квартиры
        [Required(ErrorMessage = "Укажите площадь квартиры")]
        [Range(0, int.MaxValue, ErrorMessage = "Значение должно быть
неотрицательным")]
        public int Footage { get; set; }
    }
}

```

```

        [Required(ErrorMessage = "Выберите тип объявления(0 - Сдача, 1 -
Продажа)")]
        public bool TypeOfAds { get; set; }

        [Required(ErrorMessage = "Укажите цену квартиры")]
        [Range(0, int.MaxValue, ErrorMessage = "Значение должно быть
неотрицательным")]
        public int Price { get; set; }

        [JsonPropertyName("date")]
        public DateTime DateOfPost { get; set; }
        public int Floor { get; set; }
        public bool ConstructionType { get; set; }
        public int NumberOfRooms { get; set; }
    }
}

```

Листинг Б.24 – Код модуля ApartmentsImages.cs микросервиса недвижимости, содержащий модель ApartmentsWithImages

```

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

namespace ApartmentService.Models
{
    [Table("ApartmentImages")]
    public class ApartmentImages
    {
        [Key]
        public int ImageId { get; set; }
        public int ApartmentId { get; set; }
        public string ImagePath { get; set; }
    }
}

```

Листинг Б.25 – Код модуля ApartmentsWithImagesAndPageAndFilters.cs микросервиса недвижимости, содержащий модель

ApartmentsWithImagesAndPageAndFilters

```

using System.Collections.Generic;

namespace ApartmentService.Models
{
    public class ApartmentsWithImagesAndPageAndFilters
    {
        public List<ApartmentWithImage> Apartments { get; set; }
        public int PageIndex { get; set; }
        public string Location { get; set; }
        public int MinPrice { get; set; }
        public int MaxPrice { get; set; }
        public int MinFootage { get; set; }
        public int MaxFootage { get; set; }
        public int MinFloor { get; set; }
    }
}

```

```

        public int MaxFloor { get; set; }
        public bool IsItNew { get; set; }
        public bool IsItOld { get; set; }
        public bool IsForSale { get; set; }
        public bool IsForRent { get; set; }
        public int MinRooms { get; set; }
        public int MaxRooms { get; set; }
    }
}

```

**Листинг Б.26 – Код модуля ApartmentWithImage.cs микросервиса
недвижимости, содержащий модель ApartmentWithImage**

```

using System;
using System.Text.Json.Serialization;

namespace ApartmentService.Models
{
    public class ApartmentWithImage
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public string Description { get; set; }
        public string AddedBy { get; set; }
        public int Footage { get; set; }
        public bool TypeOfAds { get; set; }
        public int Price { get; set; }

        [JsonPropertyName("date")]
        public DateTime DateOfPost { get; set; }
        public string ImagePath { get; set; }
        public string ImageData { get; set; }
        public int Floor { get; set; }
        public bool ConstructionType { get; set; }
        public int NumberOfRooms { get; set; }
    }
}

```

Листинг Б.27 – Код модуля Program.cs микросервиса аутентификации

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace aunthefication
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>

```

```

        {
            webBuilder.UseStartup<Startup>();
        });
    }
}

```

Листинг Б.28 – Код модуля Startup.cs микросервиса аутентификации

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using AuthService.Models;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.IdentityModel.Tokens;
using System.Text;
using AuthService;

namespace aunthefication
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            services.AddSwaggerGen(c =>
            {
                c.SwaggerDoc("v1", new OpenApiInfo { Title =
"aunthefication", Version = "v1" });
            });
            services.AddDbContext<AuthDbContext>(options =>
options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection
")));
            services.AddIdentity<AppUser, AppRole>(options =>
{
            options.Password.RequireDigit = true;
            options.Password.RequireLowercase = true;
            options.Password.RequireUppercase = true;
            options.Password.RequireNonAlphanumeric = true;
            options.Password.RequiredLength = 8;
        })
            .AddEntityFrameworkStores<AuthDbContext>()
            .AddDefaultTokenProviders();
            services.AddScoped<IGenerateBearerToken,
GenerateBearerTokenService>();

```

```

        services.AddAuthentication(options =>
        {
            options.DefaultAuthenticateScheme =
JwtBearerDefaults.AuthenticationScheme;
            options.DefaultChallengeScheme =
JwtBearerDefaults.AuthenticationScheme;
        })
        .AddJwtBearer(options =>
        {
            options.TokenValidationParameters = new
TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidateAudience = true,
                ValidateLifetime = true,
                ValidateIssuerSigningKey = true,
                ValidIssuer = Configuration["Jwt:Issuer"],
                ValidAudience = Configuration["Jwt:Audience"],
                IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(Configuration["Jwt:Key"]))
            };
        });
    }

    public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "aunthefication v1"));
        }

        app.UseHttpsRedirection();

        app.UseAuthentication();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}

```

Листинг Б.29 – Код модуля AuthenticationController.cs микросервиса аутентификации

```

using AuthService.Models;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

namespace AuthService.Controllers
{
    /// <summary>
    /// Контроллер для проверки аутентификации пользователя по Bearer
    /// токену в заголовке запроса
    /// </summary>
    [Authorize]
    [ApiController]
    [Route("api/[controller]")]
    public class AuthenticationController : ControllerBase
    {
        private readonly UserManager<AppUser> _userManager;

        public AuthenticationController(UserManager<AppUser> userManager)
        {
            _userManager = userManager;
        }

        /// <summary>
        /// Метод для проверки аутентификации пользователя
        /// </summary>
        /// <returns>Возвращает имя пользователя</returns>
        [HttpGet("authenticate")]
        public IActionResult Authenticate()
        {
            if (User.Identity.IsAuthenticated)
            {
                var userName = User.Identity.Name;

                return Ok(new { user = userName });
            }
            else { return BadRequest(); }
        }
    }
}

```

Листинг Б.30 – Код модуля LoginController.cs микросервиса аутентификации

```

using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;
using AuthService.Models;
using System.Threading.Tasks;

namespace AuthService.Controllers
{
    /// <summary>
    /// Контроллер для логина пользователя
    /// </summary>
    [Route("api/[controller]")]
    [ApiController]
    public class LoginController : ControllerBase
    {

```

```

{
    private readonly SignInManager<AppUser> _signInManager;
    private readonly UserManager<AppUser> _userManager;
    private readonly IGenerateBearerToken _generateBearerToken;

    public LoginController(SignInManager<AppUser> signInManager,
        UserManager<AppUser> userManager, IGenerateBearerToken
        generateBearerToken)
    {
        _signInManager = signInManager;
        _userManager = userManager;
        _generateBearerToken = generateBearerToken;
    }

    /// <summary>
    /// Метод для логина пользователя
    /// После успешного логина добавляет пользователя в базу данных
микросервиса
    /// При успешном логине генерирует токен аутентификации и
возвращает его клиенту
    /// </summary>
    /// <param name="model"></param>
    /// <returns></returns>
    [HttpPost]
    public async Task<IActionResult> Login(LoginViewModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest("Invalid username or
password");
        }

        var result = await
        _signInManager.PasswordSignInAsync(model.Username, model.Password, false,
        false);

        if (!result.Succeeded)
        {
            return BadRequest("Invalid username or
password");
        }
        var user = await
        _userManager.FindByNameAsync(model.Username);
        //Генерация токена для пользователя
        var token =
        _generateBearerToken.GenerateBearerToken(user);

        return Ok(new { Token = token });
    }
}

```

Листинг Б.31 – Код модуля RegisterController.cs микросервиса аутентификации

```

using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Mvc;

```

```

using AuthService.Models;
using System.Threading.Tasks;

namespace AuthService.Controllers
{
    /// <summary>
    /// Контроллер для регистрации пользователя
    /// </summary>
    [Route("api/[controller]")]
    [ApiController]
    public class RegisterController : ControllerBase
    {
        private readonly UserManager<AppUser> _userManager;
        private readonly IGenerateBearerToken _generateBearerToken;

        public RegisterController(UserManager<AppUser> userManager,
            IGenerateBearerToken generateBearerToken)
        {
            _userManager = userManager;
            _generateBearerToken = generateBearerToken;
        }

        /// <summary>
        /// Метод для регистрации пользователя
        /// После успешной регистрации добавляет пользователя в базу
        данных микросервиса
        /// </summary>
        /// <param name="model"></param>
        /// <returns></returns>
        [HttpPost]
        public async Task<IActionResult> Register(RegisterViewModel
model)
        {
            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }

            var user = new AppUser { Username = model.Username, Email =
model.Email };

            var result = await _userManager.CreateAsync(user,
model.Password);

            if (result.Succeeded)
            {
                //Генерация токена для пользователя
                var userName = await
_userManager.FindByNameAsync(model.Username);
                var token =
_generateBearerToken.GenerateBearerToken(userName);
                return Ok(new { Token = token });
            }
            else
            {
                return BadRequest(result.Errors);
            }
        }
    }
}

```

```

    }
}
}
}

```

Листинг Б.32 – Код модуля GenerateBearerTokenService.cs микросервиса аутентификации

```

using AuthService.Models;
using Microsoft.IdentityModel.Tokens;
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Text;
using System;
using Microsoft.Extensions.Configuration;

namespace AuthService
{
    /// <summary>
    /// Сервис для генерации Bearer токена аутентификации
    /// </summary>
    public class GenerateBearerTokenService:IGenerateBearerToken
    {
        private readonly IConfiguration _jwtSecret;

        public GenerateBearerTokenService(IConfiguration jwtSecret)
        {
            _jwtSecret = jwtSecret;
        }

        /// <summary>
        /// Генерирует Bearer токен для указанного пользователя
        /// </summary>
        /// <param name="user"></param>
        /// <returns></returns>
        public string GenerateBearerToken(AppUser user)
        {
            var tokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes(_jwtSecret["JWT:Key"]);

            //Создание описания токена, содержащего параметры и данные
            для создания JWT
            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Audience = "your_audience",
                Issuer = "your_issuer",
                //Создание идентификации (Subject) токена с утверждениями
                о пользователе
                Subject = new ClaimsIdentity(new Claim[]
                {
                    //Добавление утверждения о имени пользователя
                    new Claim(ClaimTypes.Name, user.UserName),

                    //Добавление утверждения об успешной аутентификации

```

```

        new Claim(ClaimTypes.Authentication, "true"),
    )),
    //Установка срока действия токена (7 дней с текущего
момента)
    Expires = DateTime.UtcNow.AddDays(7),

    //Установка параметров подписи токена с использованием
симметричного ключа HMAC-SHA256
    SigningCredentials = new SigningCredentials(new
SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)
    };
    //Создание JWT-токена на основе описания токена
    var token = tokenHandler.CreateToken(tokenDescriptor);
    //Преобразование JWT-токена в строку
    return tokenHandler.WriteToken(token);
}

}

}

```

Листинг Б.33 – Код модуля IGenerateBearerToken.cs микросервиса аутентификации

```

using AuthService.Models;

namespace AuthService
{
    /// <summary>
    /// Интерфейс для генерации Bearer токена аутентификации
    /// </summary>
    public interface IGenerateBearerToken
    {
        string GenerateBearerToken(AppUser user);
    }
}

```

Листинг Б.34 – Код модуля AuthModel.cs микросервиса аутентификации, содержащий модель AuthModel

```

using Microsoft.AspNetCore.Identity;
using Microsoft.AspNetCore.Identity.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;

namespace AuthService.Models
{
    public class AppUser : IdentityUser
    {
        // Дополнительные свойства пользователя, если необходимо
    }
    public class AppRole : IdentityRole
    {
        // Дополнительные свойства роли, если необходимо
    }
}

```

```

        public class AuthDbContext : IdentityDbContext<AppUser, AppRole,
string>
        {
            public AuthDbContext(DbContextOptions<AuthDbContext> options)
                : base(options)
            {
            }
        }
    }
}

```

Листинг Б.35 – Код модуля LoginViewModel.cs микросервиса аутентификации, содержащий модель LoginViewModel

```

using System.ComponentModel.DataAnnotations;

namespace AuthService.Models
{
    /// <summary>
    /// Модель данных, представляющая данные для входа в систему, включая
имя пользователя и пароль.
    /// </summary>
    public class LoginViewModel
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [DataType(DataType.Password)]
        public string Password { get; set; }
    }
}

```

Листинг Б.36 – Код модуля RegisterViewModel.cs микросервиса аутентификации, содержащий модель RegisterViewModel

```

using System.ComponentModel.DataAnnotations;

namespace AuthService.Models
{
    /// <summary>
    /// Модель данных, представляющая данные для регистрации в системе
    /// Включает в себя имя пользователя, его email, пароль и поле для
подтверждения пароля
    /// </summary>
    public class RegisterViewModel
    {
        [Required]
        public string Username { get; set; }

        [Required]
        [EmailAddress]
        public string Email { get; set; }

        [Required]

```

```

        [DataType(DataType.Password)]
        public string Password { get; set; }

        [Compare("Password", ErrorMessage = "Passwords do not match")]
        [DataType(DataType.Password)]
        public string ConfirmPassword { get; set; }
    }
}

```

Листинг Б.37 – Код модуля Program.cs микросервиса оценок

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;

namespace RateService
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                })
        }
    }
}

```

Листинг Б.38 – Код модуля Startup.cs микросервиса оценок

```

using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.OpenApi.Models;
using RateService.Controllers;

namespace RateService
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        public void ConfigureServices(IServiceCollection services)
        {

```

```

        services.AddControllers();
        services.AddSwaggerGen(c =>
        {
            c.SwaggerDoc("v1", new OpenApiInfo { Title =
"RateService", Version = "v1" });
        });

        services.AddDbContext<RateDbContext>(options =>
        {

options.UseSqlServer(Configuration.GetConnectionString("DefaultConnection
"));

        });

    }
    public void Configure(IApplicationBuilder app,
IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
            app.UseSwagger();
            app.UseSwaggerUI(c =>
c.SwaggerEndpoint("/swagger/v1/swagger.json", "RateService v1"));
        }

        app.UseHttpsRedirection();

        app.UseRouting();

        app.UseAuthorization();

        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
}

```

Листинг Б.39 – Код модуля DbContext.cs микросервиса оценок

```

using Microsoft.EntityFrameworkCore;
using RateService.Models;

namespace RateService.Controllers
{
    /// <summary>
    /// Контекст базы данных для работы с оценками квартир. Включает
набор DbSet для доступа к данным о квартирах.
    /// </summary>
    public class RateDbContext : DbContext
    {
        public RateDbContext(DbContextOptions<RateDbContext> options) :
base(options)
        {

```

```

    }
    public DbSet<RateModel> RateModels { get; set; }
}
}

```

Листинг Б.40 – Код модуля RateController.cs микросервиса оценок

```

using Microsoft.AspNetCore.Mvc;
using Newtonsoft.Json.Linq;
using RateService.Models;
using System.Linq;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Threading.Tasks;

namespace RateService.Controllers
{
    /// <summary>
    /// Контроллер для взаимодействия с базой данных оценок и для
    агрегации этих данных
    /// </summary>
    [ApiController]
    [Route("api/[controller]")]
    public class RateController : ControllerBase
    {
        private readonly RateDbContext _context;

        private readonly HttpClient _httpClient = new HttpClient();

        private readonly string _authServiceBaseUrl =
"https://localhost:44326";

        public RateController(RateDbContext context)
        {
            _context = context;
        }

        /// <summary>
        /// Post-Метод для добавления оценки в БД
        /// Если пользователь пытается добавить вторую оценку на той же
квартире, метод перенаправляет запрос на изменение
        /// </summary>
        /// <param name="rate"></param>
        /// <returns></returns>
        [HttpPost]
        public async Task<IActionResult> PostRate(RateModel rate)
        {
            var strUser = "";

            if (!ModelState.IsValid)
            {
                return BadRequest(ModelState);
            }
            // Получаем токен из заголовка запроса
            var token =
Request.Headers["Authorization"].FirstOrDefault();

```

```

        // Проверяем, что токен присутствует
        if (string.IsNullOrEmpty(token))
        {
            return Unauthorized(new {message = "Токен отсутствует"});
        }
        var tokenWithoutBearer = token.Substring(7);

        // Добавляем токен в заголовок запроса к методу контроллера
        другого микросервиса
        _httpClient.DefaultRequestHeaders.Authorization = new
        AuthenticationHeaderValue("Bearer", tokenWithoutBearer);

        // Отправляем данные к методу контроллера другого
        микросервиса
        HttpResponseMessage response = await
        _httpClient.GetAsync($"{_authServiceBaseUrl}/api/authentication/authentic
        ate");

        if (response.IsSuccessStatusCode)
        {
            var userRaw =
            response.Content.ReadAsStringAsync().Result;

            var JObjectUser = JObject.Parse(userRaw);

            strUser = JObjectUser["user"].ToString();

            if (strUser != rate.UserName)
            {
                return Unauthorized(new {message = "Отказано в
                доступе"});
            }

            var existingRate = _context.RateModels.FirstOrDefault(r
            => r.UserName == rate.UserName && r.ApartmentId == rate.ApartmentId);

            // Если запись существует, обновляем ее
            if (existingRate != null)
            {
                existingRate.Rate = rate.Rate; // Обновляем оценку
                _context.SaveChanges();
                return Ok("Rate updated");
            }
            // Если записи нет, добавляем новую
            else
            {
                _context.RateModels.Add(rate);
                _context.SaveChanges();
                return Ok("New rate added");
            }
        }
        else
        {
            return Unauthorized();
        }
    }
}

```

```

    }
}

/// <summary>
/// Get-Метод для возврата средних оценок для квартир
/// </summary>
/// <returns></returns>
[HttpGet("AvgRates")]
public IActionResult AvgRates()
{
    //Вычисление средних оценок для каждой квартиры, группируя
записи по ApartmentId
    var avgRates = _context.RateModels
        .GroupBy(r => r.ApartmentId)
        .Select(g => new { ApartmentId = g.Key, AverageRate =
g.Average(r => r.Rate) })
        .ToList();

    return Ok(avgRates);
}
}
}

```

Листинг Б.41 – Код модуля RateModel.cs микросервиса оценок, содержащий модель RateModel

```

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

namespace RateService.Models
{
    /// <summary>
    /// Модель данных, представляющая оценки квартир в системе
    /// Включает в себя id оценки, id квартиры, Имя пользователя,
сделавшего оценку и саму оценку
    /// </summary>
    [Table("Rates")]
    public class RateModel
    {
        [Key]
        public int Id { get; set; }
        public int ApartmentId { get; set; }
        public string UserName { get; set; }
        public int Rate { get; set; }
    }
}

```

Приложение В
(рекомендуемое)
Исходный код интерфейса

Листинг В.1 – Разметка представления LogReg.cshtml

```

@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor HttpContextAccessor
@ {
    ViewData["Title"] = "Логин/Регистрация";
}
@ {
    HttpContext httpContext = HttpContextAccessor.HttpContext;
} < !DOCTYPE html >
< html >
< head >
< script src = "~/js/logReg.js" > < /script> < title > Вход /
Регистрация < /title> < /head> < body style = "margin-top:80px" >
< div style = "text-align: center; padding-top: 50px;" >
< h2 > Добро пожаловать на наш сайт < /h2>
<!--Определение сообщений при попытке выполнить действия, не будучи
авторизованным в систему-->
@if(httpContext.Session.GetString("EnterProfile") == "false") { < div class
= "alert alert-danger"
    role = "alert" >
        Войдите или зарегистрируйтесь, чтобы получить доступ к личному
кабинету < /div>
        httpContext.Session.Remove("EnterProfile");
    }
@if(httpContext.Session.GetString("RateAccess") == "false") { < div class
= "alert alert-danger"
    role = "alert" >
        Войдите или зарегистрируйтесь, чтобы ставить оценки < /div>
        httpContext.Session.Remove("RateAccess");
    }
@if(httpContext.Session.GetString("logout") == "true") { < div class =
"alert alert-info"
    role = "alert" >
        Выход из аккаунта прошел успешно < /div>
        httpContext.Session.Remove("logout");
    } < div style = "display: inline-block; margin-top: 30px;" >
<!-- Форма логина -->
< div style = "margin-bottom: 20px;" >
< h3 > Вход < /h3> < form action = "/LogReg/LoginUser"
method = "post" >
    < div style = "margin-bottom: 10px;" >
        < label
for = "username" > Имя пользователя: < /label><br / >
        < input type = "text"
id = "username"
name = "username"
required / >
    < /div> < div style = "margin-bottom: 10px;" >
        < label
for = "password" > Пароль: < /label><br / >
        < input type = "password"
id = "password"
name = "password"
required / >

```

```

    < /div> < button type = "submit"
style = "background-color: #007bff; color: #fff; padding: 10px 20px;
border: none; border-radius: 3px; cursor: pointer;" > Войти < /button> <
/form> < /div>
<!-- Форма регистрации -->
< div >
< h3 > Регистрация < /h3> < form id = "regForm"
action = "/LogReg/RegisterUser"
method = "post" >
< div style = "margin-bottom: 10px;" >
< label
for = "regUsername" > Имя пользователя: < /label><br / >
< input type = "text"
id = "regUsername"
name = "username"
required / >
< /div> < div style = "margin-bottom: 10px;" >
< span id = "usernameError"
style = "color: red;" > < /span> < /div> < input type = "hidden"
id = "email"
name = "email"
value = "eddybor@gmail.com" / >
< div style = "margin-bottom: 10px;" >
< label
for = "regPassword" > Пароль: < /label><br / >
< input type = "password"
id = "regPassword"
name = "password"
required / >
< /div> < div style = "margin-bottom: 10px;" >
< label
for = "confirmPassword" > Повторите пароль: < /label><br / >
< input type = "password"
id = "confirmPassword"
name = "confirmPassword"
required / >
< /div> < div style = "margin-bottom: 10px;" >
< span id = "passwordError"
style = "color: red;" > < /span> < /div> < button type = "submit"
id = "regSubmit"
style = "background-color: #28a745; color: #fff; padding: 10px 20px;
border: none; border-radius: 3px; cursor: pointer;" > Зарегистрироваться
< /button> < /form> < /div> < /div> < /div> < /body> < /html>

```

Листинг В.2 – Разметка представления MainPage.cshtml

```

@using Microsoft.AspNetCore.Http
@Inject IHttpContextAccessor HttpContextAccessor
@ {
    ViewData["Title"] = "Главная";
}
@ {
    HttpContext httpContext = HttpContextAccessor.HttpContext;
}
@model ApartmentDataViewModel
< !DOCTYPE html >

```

```

< html >
< head >
< script src = "https://code.jquery.com/jquery-3.6.0.min.js" > <
/script> < link rel = "stylesheet"
href = "~/css/MainPage.css" >
<!--
< script src = "~/js/scrollScript.js" > < /script>
-->
< script src = "https://api-
maps.yandex.ru/2.1/?lang=ru_RU&apikey=00ba64fd-3590-4b6b-8660-
be8f0ff799a0&suggest_apikey=21a30eeb-bac5-44db-9170-498665d71a51"
type = "text/javascript" >
< /script> < script src = "~/js/MainPage.js" > < /script> < /head> <
body style = "margin-top:80px" >
< div class = "container-fluid" >
< div class = "row" >
<!-- Столбец с фильтрами -->
< div class = "col-md-2" >
@for(int i = 1; i <= 6; i++) { < br / >
} < form method = "get"
action = "@Url.Action("
MainPage ", "
MainPage ")" >
< div class = "form-group" >
< label
for = "location" > Адрес: < /label> < input type = "text"
class = "form-control"
id = "suggest"
name = "location"
value = "@Model.Location" >
< /div> < div style = "margin-bottom: 10px;" >
< span id = "nameError"
style = "color: red;" > < /span> < /div> < div class = "form-group" >
< label
for = "minArea" > Площадь от: < /label> < input type = "number"
class = "form-control"
id = "minFootage"
name = "minFootage"
min = "0"
max = "@int.MaxValue"
value = "@Model.MinFootage" >
< /div> < div class = "form-group" >
< label
for = "maxArea" > Площадь до: < /label> < input type = "number"
class = "form-control"
id = "maxFootage"
name = "maxFootage"
min = "0"
max = "@int.MaxValue"
value = "@Model.MaxFootage" >
< /div> < div class = "form-group" >
< label
for = "minPrice" > Цена от: < /label> < input type = "number"
class = "form-control"
id = "minPrice"
name = "minPrice"

```

```

min = "0"
max = "@int.MaxValue"
value = "@Model.MinPrice" >
  < /div> < div class = "form-group" >
    < label
for = "maxPrice" > Цена до: < /label> < input type = "number"
class = "form-control"
id = "maxPrice"
name = "maxPrice"
min = "0"
max = "@int.MaxValue"
value = "@Model.MaxPrice" >
  < /div> < div class = "form-group" >
    < label
for = "minFloor" > Этаж от: < /label> < input type = "number"
class = "form-control"
id = "minFloor"
name = "minFloor"
min = "0"
max = "@int.MaxValue"
value = "@Model.MinFloor" >
  < /div> < div class = "form-group" >
    < label
for = "maxFloor" > Этаж до: < /label> < input type = "number"
class = "form-control"
id = "maxFloor"
name = "maxFloor"
min = "0"
max = "@int.MaxValue"
value = "@Model.MaxFloor" >
  < /div> < div class = "form-group" >
    < label
for = "minRooms" > Количество комнат от: < /label> < select class =
"form-control"
id = "minRooms"
name = "minRooms"
required >
  < option value = "0" > Студия < /option> < option value = "1" > 1
комнатная < /option> < option value = "2" > 2 комнатная < /option> <
option value = "3" > 3 комнатная < /option> < option value = "4" > 4
комнатная < /option> < option value = "5" > 5 и более комнат < /option> <
/ select> < /div> < div class = "form-group" >
  < label
for = "maxRooms" > Количество комнат до: < /label> < select class =
"form-control"
id = "maxRooms"
name = "maxRooms"
required >
  < option value = "0" > Студия < /option> < option value = "1" > 1
комнатная < /option> < option value = "2" > 2 комнатная < /option> <
option value = "3" > 3 комнатная < /option> < option value = "4" > 4
комнатная < /option> < option value = "5" > 5 и более комнат < /option> <
/ select> < /div> < div class = "form-group" >
  < label > Тип застройки: < /label> < div >
    < label >
    < input type = "checkbox"

```

```

id = "isItNewCheckbox"
name = "isItNew"
value = "true" > Новостройка < /label> < /div> < div >
    < label >
    < input type = "checkbox"
name = "isItOld"
value = "true" > Вторичное жилье < /label> < /div> < /div> < div class =
"form-group" >
    < label > Тип объявления: < /label> < div >
    < label >
    < input type = "checkbox"
name = "isForSale"
value = "true" > Продажа < /label> < /div> < div >
    < label >
    < input type = "checkbox"
name = "isForRent"
value = "true" > Сдача < /label> < /div> < /div> < button type = "submit"
class = "btn btn-primary" > Применить фильтры < /button> < /form> < /div>
<!-- Основной контент -->
< div class = "col-md-10" >
    <!--
    < button onclick = "location.href='/Account/UserProfile'"
class = "btn btn-primary"
style = "float:right" > Личный кабинет < /button>
    -->
    < h1 > Главная < /h1>
@if(httpContext.Session.GetString("logged") == "true") { < div class =
"alert alert-success"
    role = "alert" >
        Вход выполнен успешно < /div>
        httpContext.Session.Remove("logged");
}
@if(httpContext.Session.GetString("registered") == "true") { < div class
= "alert alert-success"
    role = "alert" >
        Регистрация выполнена успешно < /div>
        httpContext.Session.Remove("registered");
} < h4 >
Добро пожаловать,
@if(Model.User.UserName == "") { < text > Аноним! < /text>
} else { < text > @Model.User.UserName! < /text>
} < /h4>
@if(Model.Apartments.Apartments.Count == 0) { < h5 > Не найдено ни одного
объявления < /h5>
} else { < h5 > Было найдено @Model.NumberOfApartments объявлений < /h5>
< div class = "row" >
    @foreach(var apartment in Model.Apartments.Apartments) { < div class
= "col-md-4 card-deck" >
        < div class = "card bg-light border-secondary" >
        < div class = "card-img-top"
style = "height: 300px; overflow:hidden" >
            @if(apartment.ImageData == "") { < img class = "card-img-top"
src = "https://via.placeholder.com/300x300?text=Нет+Фото"
alt = "Card image cap" >
            } else { < img class = "card-img-top"
src = "@apartment.ImageData"

```

```

        alt = "Card image cap" >
    }
    <!--card-img-top-->
    < /div> < div class = "card-body" >
        < h5 class = "card-title" > @apartment.Name < /h5> < p class =
"card-text" >
            @if(apartment.TypeOfAds) { < text > Продажа < /text>
            } else { < text > Сдача < /text>
        } < /p> < p class = "card-title" > Этаж: @apartment.Floor < /p> < p
class = "card-title" >
            @if(apartment.ConstructionType) { < text > Новостройка < /text>
            } else { < text > Вторичка < /text>
        } < /p> < p class = "card-title" >
            @switch(apartment.NumberOfRooms) {
                case 0:
                    < text > Студия < /text>
                    break;
                case 1:
                    < text > 1 комнатная < /text>
                    break;
                case 2:
                    < text > 2 комнатная < /text>;
                    break;
                case 3:
                    < text > 3 комнатная < /text>
                    break;
                case 4:
                    < text > 4 комнатная < /text>
                    break;
                default:
                    < text > 5 и более комнат < /text>
                    break;
            } < /p> < p class = "card-text" > @String.Format("{0:#,0}",
apartment.Footage) м² < /p> < p class = "card-text" >
            @if(apartment.TypeOfAds) { < text > @String.Format("{0:#,0}",
apartment.Price)₽ < /text>
            } else { < text > @String.Format("{0:#,0}", apartment.Price)₽ /
мес. < /text>
        } < /p>
    <!--
    < p class = "card-text" > @apartment.Description < /p>
    -->
    < p class = "card-text" >
        @if(apartment.TypeOfAds) {
        } else { < text > Оценка: < /text>
        @
        Model.AvgRate.FirstOrDefault(avg => avg.ApartmentID ==
apartment.Id) ? .AverageRate != null ?
        Math.Round(Model.AvgRate.FirstOrDefault(avg => avg.ApartmentID
== apartment.Id) ? .AverageRate ?? 0, 1).ToString() :
        "нет"
        )
    }
    < /p>
    <!--форма для главной страницы-->
    < form id = "MoreInfo"

```

```

        enctype = "multipart/form-data"
        action = "/ApartmentPage/ApartmentPage"
        method = "get" >
            < input type = "hidden"
            name = "id"
            value = "@apartment.Id" / >
            <!--Alotofcrutches-->
            < input type = "hidden"
            name = "imagePath"
            value = "@apartment.ImagePath" / >
            < button type = "submit"
            class = "btn btn-primary" > Подробнее < /button> < /form>
            <!--card-body-->
            < /div> < div class = "card-footer text-muted" >
                Добавлено @apartment.date.ToString().Substring(0, 10) < /div>
            <!--card-->
            < /div>
        <!--col-md-4-->
    < /div>
}
<!--row-->
< /div>
}
@ {
    int totalItems = Model.NumberOfApartments; // Общее количество
объявлений
    int itemsPerPage = 12; // Количество элементов на странице
    int totalPages = (int) Math.Ceiling((double) totalItems /
itemsPerPage); // Вычисление общего количества страниц
} < div style = "text-align: center;" >
    < div class = "pagination" >
        < ul style = "list-style: none; padding: 0; margin: 0;" >
            @for(var i = 1; i <= totalPages; i++) { < li style = "display: inline-
block; margin-right: 5px;" >
                < a href = "@Url.Action("
                MainPage ", new { page = i,
                location = Model.Location,
                pageSize = 12,
                minPrice = Model.MinPrice,
                maxPrice = Model.MaxPrice,
                minFootage = Model.MinFootage,
                maxFootage = Model.MaxFootage, minFloor = Model.MinFloor,
                maxFloor = Model.MaxFloor,
                isItNew = Model.IsItNew,
                isItOld = Model.IsItOld,
                isForSale = Model.IsForSale,
                isForRent = Model.IsForRent,
                minRooms = Model.MinRooms,
                maxRooms = Model.MaxRooms
                })
                " style="
                display: block;
                padding: 5 px 10 px;
                background - color: #f2f2f2;

```

```

color: #333; text-decoration: none; border-radius: 5px; @(i ==
Model.CurrentPage ? "background-color: #007bff; color: #fff;" :
"color:#333")
">@i</a> < /li>
} < /ul> < /div> < /div> < /div> < /div>
<!--Container fluid-->
< /div>
<!--На всякий случай добавил ссылку на ionicons, если приглянется что-то
оттуда-->
< script type = "module"
src = "https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm.js" >
< /script> < script nomodule src =
"https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js" > < /script>
< /body> < /html>

```

Листинг В.3 – Разметка представления ApartmentPage.cshtml

```

@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor HttpContextAccessor
@ {
    HttpContext httpContext = HttpContextAccessor.HttpContext;
}
@model ApartmentPageViewModel
< html >
< head >
< title > Информация о недвижимости < /title> < link rel = "stylesheet"
href = "~/css/ApartmentPage.css" / >
< meta charset = "utf-8" / >
< script src = "https://api-
maps.yandex.ru/2.1/?lang=ru_RU&apikey=00ba64fd-3590-4b6b-8660-
be8f0ff799a0"
type = "text/javascript" > < /script> < style > #map {
    width: 600 px;
    height: 400 px;
    margin: 20 px auto;
} < /style> < /head> < body style = "margin-top:80px" >
< div class = "container-fluid" >
< div class = "gallery" >
    @if(Model.images == null) { < img src =
"https://via.placeholder.com/150x150?text=Нет+Фото"
    alt = "Нет картинки" / >
    } else {
    @foreach(var images in Model.images) {
        string base64String = Convert.ToBase64String(images); < img src =
"data:image/jpeg;base64,@base64String"
        style = "width:300px;height:300px" / >
    }
} < /div> < h1 > @Model.Name < /h1> < div class = "details" >
< p > < strong > Цена: < /strong> @(Model.TypeOfAds ? Model.Price+" ₺"
: Model.Price+" ₰/мес
")< /p> < p > < strong > Площадь: < /strong> @Model.Footage м² < /p >
< p >
< strong > Тип застройки: < /strong> @(Model.ConstructionType ?
"Новостройка" : "Вторичка") < /p> < p > < strong > Этаж: < /strong>
@Model.Floor < /p >
< p > < strong > Количество комнат: < /strong>

```

```

@switch(Model.NumberOfRooms) {
    case 0:
        < text > Студия < /text>
        break;
    case 1:
        < text > 1 комнатная < /text>
        break;
    case 2:
        < text > 2 комнатная < /text>
        break;
    case 3:
        < text > 3 комнатная < /text>
        break;
    case 4:
        < text > 4 комнатная < /text>
        break;
    default:
        < text > 5 и более комнат < /text>
        break;
} < /p> < p > < strong > Тип объявления: < /strong> @(Model.TypeOfAds ?
"Продажа" : "Сдача")< /p >
< p > < strong > Добавлено: < /strong> @Model.date.ToString("dd/MM /
yyyy ")< /p> < p > < strong > Добавил: < /strong> @Model.AddedBy< /p >
@if(httpContext.Session.GetString("RateStatus") == "true") { < div
class = "alert alert-success"
role = "alert" >
    Оценка успешно добавлена < /div>
    httpContext.Session.Remove("RateStatus");
}
@if(!Model.TypeOfAds) { < p >
    < strong > Средняя оценка: < /strong> @(
    Model.AvgRate.FirstOrDefault(avg => avg.ApartmentID == Model.Id) ?
    .AverageRate != null ?
        Math.Round(Model.AvgRate.FirstOrDefault(avg => avg.ApartmentID ==
Model.Id) ? .AverageRate ?? 0, 1).ToString() :
        "Нет оценки"
    ) < /p>
< form id = "Rate"
action = "/Rate/RateApartment"
method = "post" >
    < input type = "hidden"
name = "apartmentId"
value = "@Model.Id" / >
    <!--A ton of crutches below-->
    < input type = "hidden"
name = "imagePath"
value = "@Model.ImagePath" / >
    < select name = "rate" >
        < 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 type = "submit"
class = "btn btn-primary" > Оценить < /button> < /form>
} < /div>
< div class = "description" >
    < h2 > Описание < /h2> < p > @Model.Description < /p> < /div> < /div>
< div id = "map" > < /div>

```

```

@section Scripts { < script type = "text/javascript" >
    ymaps.ready(init);

    function init() {
        // Получаем адрес из модели
        var address = '@Html.Raw(Model.Name)'; // Обратите внимание на
использование
        console.log('Адрес из модели:', address); // Отладочный вывод
        var country = 'Россия'; // Указываем страну для повышения точности
геокодирования
        // Создание карты
        var myMap = new ymaps.Мap("map", {
            center: [55.76, 37.64], // Москва, центральное значение для
инициализации
            zoom: 10
        });
        // Геокодирование адреса
        ymaps.geocode(address + ', ' + country).then(function(res) {
            if (res.geoObjects.getLength() > 0) {
                var geoObject = res.geoObjects.get(0);
                var coordinates = geoObject.geometry.getCoordinates();
                var foundAddress = geoObject.getAddressLine();
                var countryCode = geoObject.getCountryCode();
                // Логирование для отладки
                console.log('Найденный адрес:', foundAddress);
                console.log('Координаты:', coordinates);
                console.log('Код страны:', countryCode);
                // Проверяем, что найденный адрес находится в России
                if (countryCode === 'RU') {
                    // Установка центра карты на найденный адрес
                    myMap.setCenter(coordinates, 16);
                    // Добавление метки на карту
                    var myPlacemark = new ymaps.Placemark(coordinates, {
                        balloonContent: foundAddress
                    }, {
                        preset: 'islands#icon',
                        iconColor: '#0095b6'
                    });
                    myMap.geoObjects.add(myPlacemark);
                } else {
                    alert('Адрес найден, но он находится в другой стране: ' +
foundAddress);
                }
            } else {
                alert('Адрес не найден. Пожалуйста, проверьте правильность
ввода.');
```

Листинг В.4 – Разметка представления MainPage.cshtml

```

@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor HttpContextAccessor
@ {
    ViewData["Title"] = "Главная";
}
@ {
    HttpContext httpContext = HttpContextAccessor.HttpContext;
}
@model ApartmentDataViewModel
< !DOCTYPE html >
< html >
< head >
< script src = "https://code.jquery.com/jquery-3.6.0.min.js" > <
/s>script> < link rel = "stylesheet"
href = "~/css/MainPage.css" >
<!--
< script src = "~/js/scrollScript.js" > < /script>
-->
< script src = "https://api-
maps.yandex.ru/2.1/?lang=ru_RU&apikey=00ba64fd-3590-4b6b-8660-
be8f0ff799a0&suggest_apikey=21a30eeb-bac5-44db-9170-498665d71a51"
type = "text/javascript" >
< /script> < script src = "~/js/MainPage.js" > < /script> < /head> <
body style = "margin-top:80px" >
< div class = "container-fluid" >
< div class = "row" >
<!-- Столбец с фильтрами -->
< div class = "col-md-2" >
@for(int i = 1; i <= 6; i++) { < br / >
} < form method = "get"
action = "@Url.Action("
MainPage ", "
MainPage ")" >
< div class = "form-group" >
< label
for = "location" > Адрес: < /label> < input type = "text"
class = "form-control"
id = "suggest"
name = "location"
value = "@Model.Location" >
< /div> < div style = "margin-bottom: 10px;" >
< span id = "nameError"
style = "color: red;" > < /span> < /div> < div class = "form-group" >
< label
for = "minArea" > Площадь от: < /label> < input type = "number"
class = "form-control"
id = "minFootage"
name = "minFootage"
min = "0"
max = "@int.MaxValue"
value = "@Model.MinFootage" >
< /div> < div class = "form-group" >
< label
for = "maxArea" > Площадь до: < /label> < input type = "number"

```

```

class = "form-control"
id = "maxFootage"
name = "maxFootage"
min = "0"
max = "@int.MaxValue"
value = "@Model.MaxFootage" >
  < /div> < div class = "form-group" >
    < label
for = "minPrice" > Цена от: < /label> < input type = "number"
class = "form-control"
id = "minPrice"
name = "minPrice"
min = "0"
max = "@int.MaxValue"
value = "@Model.MinPrice" >
  < /div> < div class = "form-group" >
    < label
for = "maxPrice" > Цена до: < /label> < input type = "number"
class = "form-control"
id = "maxPrice"
name = "maxPrice"
min = "0"
max = "@int.MaxValue"
value = "@Model.MaxPrice" >
  < /div> < div class = "form-group" >
    < label
for = "minFloor" > Этаж от: < /label> < input type = "number"
class = "form-control"
id = "minFloor"
name = "minFloor"
min = "0"
max = "@int.MaxValue"
value = "@Model.MinFloor" >
  < /div> < div class = "form-group" >
    < label
for = "maxFloor" > Этаж до: < /label> < input type = "number"
class = "form-control"
id = "maxFloor"
name = "maxFloor"
min = "0"
max = "@int.MaxValue"
value = "@Model.MaxFloor" >
  < /div> < div class = "form-group" >
    < label
for = "minRooms" > Количество комнат от: < /label> < select class =
"form-control"
id = "minRooms"
name = "minRooms"
required >
  < option value = "0" > Студия < /option> < option value = "1" > 1
комнатная < /option> < option value = "2" > 2 комнатная < /option> <
option value = "3" > 3 комнатная < /option> < option value = "4" > 4
комнатная < /option> < option value = "5" > 5 и более комнат < /option> <
/select> < /div> < div class = "form-group" >
  < label

```

```

for = "maxRooms" > Количество комнат до: < /label> < select class =
"form-control"
id = "maxRooms"
name = "maxRooms"
required >
    < option value = "0" > Студия < /option> < option value = "1" > 1
комнатная < /option> < option value = "2" > 2 комнатная < /option> <
option value = "3" > 3 комнатная < /option> < option value = "4" > 4
комнатная < /option> < option value = "5" > 5 и более комнат < /option> <
/option> < /select> < /div> < div class = "form-group" >
    < label > Тип застройки: < /label> < div >
    < label >
    < input type = "checkbox"
id = "isItNewCheckbox"
name = "isItNew"
value = "true" > Новостройка < /label> < /div> < div >
    < label >
    < input type = "checkbox"
name = "isItOld"
value = "true" > Вторичное жилье < /label> < /div> < /div> < div class =
"form-group" >
    < label > Тип объявления: < /label> < div >
    < label >
    < input type = "checkbox"
name = "isForSale"
value = "true" > Продажа < /label> < /div> < div >
    < label >
    < input type = "checkbox"
name = "isForRent"
value = "true" > Сдача < /label> < /div> < /div> < button type = "submit"
class = "btn btn-primary" > Применить фильтры < /button> < /form> < /div>
<!-- Основной контент -->
< div class = "col-md-10" >
    <!--
    < button onclick = "location.href='/Account/UserProfile'"
class = "btn btn-primary"
style = "float:right" > Личный кабинет < /button>
-->
    < h1 > Главная < /h1>
@if(httpContext.Session.GetString("logged") == "true") { < div class =
"alert alert-success"
    role = "alert" >
        Вход выполнен успешно < /div>
    httpContext.Session.Remove("logged");
}
@if(httpContext.Session.GetString("registered") == "true") { < div class
= "alert alert-success"
    role = "alert" >
        Регистрация выполнена успешно < /div>
    httpContext.Session.Remove("registered");
} < h4 >
Добро пожаловать,
@if(Model.User.UserName == "") { < text > Аноним! < /text>
} else { < text > @Model.User.UserName! < /text>
} < /h4>

```

```

@if(Model.Apartments.Apartments.Count == 0) { < h5 > Не найдено ни одного
объявления < /h5>
} else { < h5 > Было найдено @Model.NumberOfApartments объявлений < /h5>
< div class = "row" >
    @foreach(var apartment in Model.Apartments.Apartments) { < div class
= "col-md-4 card-deck" >
        < div class = "card bg-light border-secondary" >
            < div class = "card-img-top"
style = "height: 300px; overflow:hidden" >
                @if(apartment.ImageData == "") { < img class = "card-img-top"
src = "https://via.placeholder.com/300x300?text=Нет+Фото"
alt = "Card image cap" >
                } else { < img class = "card-img-top"
src = "@apartment.ImageData"
alt = "Card image cap" >
            }
        <!--card-img-top-->
        < /div> < div class = "card-body" >
            < h5 class = "card-title" > @apartment.Name < /h5> < p class =
"card-text" >
                @if(apartment.TypeOfAds) { < text > Продажа < /text>
                } else { < text > Сдача < /text>
            } < /p> < p class = "card-title" > Этаж: @apartment.Floor < /p> < p
class = "card-title" >
                @if(apartment.ConstructionType) { < text > Новостройка < /text>
                } else { < text > Вторичка < /text>
            } < /p> < p class = "card-title" >
                @switch(apartment.NumberOfRooms) {
                    case 0:
                        < text > Студия < /text>
                        break;
                    case 1:
                        < text > 1 комнатная < /text>
                        break;
                    case 2:
                        < text > 2 комнатная < /text>;
                        break;
                    case 3:
                        < text > 3 комнатная < /text>
                        break;
                    case 4:
                        < text > 4 комнатная < /text>
                        break;
                    default:
                        < text > 5 и более комнат < /text>
                        break;
                } < /p> < p class = "card-text" > @String.Format("{0:#,0}",
apartment.Footage) м² < /p> < p class = "card-text" >
                @if(apartment.TypeOfAds) { < text > @String.Format("{0:#,0}",
apartment.Price)₽ < /text>
                } else { < text > @String.Format("{0:#,0}", apartment.Price) /
мес. < /text>
            } < /p>
        <!--
        < p class = "card-text" > @apartment.Description < /p>
        -->
    }
}

```

```

        < p class = "card-text" >
        @if(apartment.TypeOfAds) {
        } else { < text > Оценка: < /text>
        @(
            Model.AvgRate.FirstOrDefault(avg => avg.ApartmentID ==
apartment.Id) ? .AverageRate != null ?
            Math.Round(Model.AvgRate.FirstOrDefault(avg => avg.ApartmentID
== apartment.Id) ? .AverageRate ?? 0, 1).ToString() :
            "нет"
        )
        }
    < /p>
    <!--форма для главной страницы-->
    < form id = "MoreInfo"
    enctype = "multipart/form-data"
    action = "/ApartmentPage/ApartmentPage"
    method = "get" >
        < input type = "hidden"
        name = "id"
        value = "@apartment.Id" / >
        <!--Alotofcrutches-->
        < input type = "hidden"
        name = "imagePath"
        value = "@apartment.ImagePath" / >
        < button type = "submit"
        class = "btn btn-primary" > Подробнее < /button> < /form>
    <!--card-body-->
    < /div> < div class = "card-footer text-muted" >
        Добавлено @apartment.date.ToString().Substring(0, 10) < /div>
    <!--card-->
    < /div>
    <!--col-md-4-->
    < /div>
}
<!--row-->
< /div>
}
@ {
    int totalItems = Model.NumberOfApartments; // Общее количество
объявлений
    int itemsPerPage = 12; // Количество элементов на странице
    int totalPages = (int) Math.Ceiling((double) totalItems /
itemsPerPage); // Вычисление общего количества страниц
} < div style = "text-align: center;" >
    < div class = "pagination" >
        < ul style = "list-style: none; padding: 0; margin: 0;" >
            @for(var i = 1; i <= totalPages; i++) { < li style = "display: inline-
block; margin-right: 5px;" >
                < a href = "@Url.Action("
MainPage ", new { page = i,
location = Model.Location,
pageSize = 12,
minPrice = Model.MinPrice,
maxPrice = Model.MaxPrice,
minFootage = Model.MinFootage,
maxFootage = Model.MaxFootage, minFloor = Model.MinFloor,

```

```

        maxFloor = Model.MaxFloor,
        isItNew = Model.IsItNew,
        isItOld = Model.IsItOld,
        isForSale = Model.IsForSale,
        isForRent = Model.IsForRent,
        minRooms = Model.MinRooms,
        maxRooms = Model.MaxRooms
    })
" style="
display: block;
padding: 5 px 10 px;
background - color: #f2f2f2;
color: #333; text-decoration: none; border-radius: 5px; @(i ==
Model.CurrentPage ? "background-color: #007bff; color: #fff;" :
"color:#333")
">@i</a> < /li>
} < /ul> < /div> < /div> < /div> < /div>
<!--Container fluid-->
< /div>
<!--На всякий случай добавил ссылку на ionicons, если приглянется что-то
оттуда-->
< script type = "module"
src = "https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.esm.js" >
< /script> < script nomodule src =
"https://unpkg.com/ionicons@7.1.0/dist/ionicons/ionicons.js" > < /script>
< /body> < /html>

```

Листинг В.5 – Код представления _Layout.cshtml

```

@using Microsoft.AspNetCore.Http
@inject IHttpContextAccessor HttpContextAccessor
@ {
    HttpContext httpContext = HttpContextAccessor.HttpContext;
}
< !DOCTYPE html >
< html lang = "en" >
< head >
< meta charset = "utf-8" / >
< meta name = "viewport"
content = "width=device-width, initial-scale=1.0" / >
< title > @ViewData["Title"] < /title> < link rel = "stylesheet"
href = "~/lib/bootstrap/dist/css/bootstrap.min.css" / >
< link rel = "stylesheet"
href = "~/css/site.css" / >
< /head> < body >
< header style = "position:fixed; top: 0;
left: 0;
width: 100 % ;
background - color: #ffffff;
z - index: 1000;
max - height: fit - content "> < nav class = "navbar navbar-expand-sm
navbar-togglerable-sm navbar-light bg-white border-bottom box-shadow mb-3"
>
< div class = "container" >
< a class = "navbar-brand font-weight-bold"

```

```

asp - area = ""
asp - controller = "MainPage"
asp - action = "MainPage" > Продажа и аренда недвижимости < /a> < button
class = "navbar-toggler"
type = "button"
data - toggle = "collapse"
data - target = ".navbar-collapse"
aria - controls = "navbarSupportedContent"
aria - expanded = "false"
aria - label = "Toggle navigation" >
    < span class = "navbar-toggler-icon" > < /span> < /button> < div class
= "navbar-collapse collapse d-sm-inline-flex justify-content-between" >
    < ul class = "navbar-nav flex-grow-1" >
    < li class = "nav-item" >
    < a class = "nav-link text-dark"
asp - area = ""
asp - controller = "MainPage"
asp - action = "MainPage" > Главная < /a> < /li>
    <!--
    < li class = "nav-item" >
    < a class = "nav-link text-dark"
asp - area = ""
asp - controller = "LogReg"
asp - action = "LogReg" > Логин и регистрация < /a> < /li> < li class =
"nav-item" >
    < a class = "nav-link text-dark"
asp - area = ""
asp - controller = "Account"
asp - action = "UserProfile" > Личный кабинет < /a> < /li>
    -->
< /ul> < button onclick = "location.href='/Account/UserProfile'"
class = "btn btn-primary"
style = "margin-right:5px" > Личный кабинет < /button>
@ {
    var isAuthenticated = HttpContext.Session.GetString("JWTToken") !=
null;
}
@if(!isAuthenticated) { < button onclick =
"location.href='/LogReg/LogReg'"
    class = "btn btn-success" > Логин / Регистрация < /button>
} else { < button onclick = "return confirmLogout()"
    class = "btn btn-danger" > Выйти < /button>
}
< script type = "text/javascript" >
    function confirmLogout() {
        if (confirm('Вы точно хотите выйти?')) {
            window.location.href = '/LogReg/LogoutUser';
        }
        return false;
    } < /script>
< /div> < /div> < /nav> < /header> < div class = "container"
style = "max-width:80%; margin:0 auto" >
    < main role = "main"
class = "pb-3" >
    @RenderBody() < /main> < /div>
    <!-- Подвал убран на время разработки

```

```

    < footer class = "border-top footer text-muted" >
    < div class = "container" >
    & copy;
2024 - webapp - < a asp - area = ""
asp - controller = "Home"
asp - action = "Privacy" > Privacy < /a> < /div> < /footer>-->
< script src = "~/lib/jquery/dist/jquery.min.js" > < /script> < script
src = "~/lib/bootstrap/dist/js/bootstrap.bundle.min.js" > < /script> <
script src = "~/js/site.js"
asp - append - version = "true" > < /script>
@await RenderSectionAsync("Scripts", required: false) < /body> < /html>

```

Листинг В.6 – Код файла стилей site.css

```

a.navbar-brand {
    white-space: normal;
    text-align: center;
    word-break: break-all;
}

/* Provide sufficient contrast against white background */
a {
    color: #0366d6;
}

.btn-primary {
    color: #fff;
    background-color: #1b6ec2;
    border-color: #1861ac;
}

.nav-pills .nav-link.active, .nav-pills .show > .nav-link {
    color: #fff;
    background-color: #1b6ec2;
    border-color: #1861ac;
}

/* Sticky footer styles
----- */
html {
    font-size: 14px;
}
@media (min-width: 768px) {
    html {
        font-size: 16px;
    }
}

.border-top {
    border-top: 1px solid #e5e5e5;
}
.border-bottom {
    border-bottom: 1px solid #e5e5e5;
}

```

```

.box-shadow {
  box-shadow: 0 .25rem .75rem rgba(0, 0, 0, .05);
}

button.accept-policy {
  font-size: 1rem;
  line-height: inherit;
}

/* Sticky footer styles
----- */
html {
  position: relative;
  min-height: 100%;
}

body {
  margin-bottom: 60px;
}

.footer {
  position: absolute;
  bottom: 0;
  width: 100%;
  white-space: nowrap;
  line-height: 60px; /* Vertically center the text there */
}

```

Листинг В.7 – Код файла стилей ApartmentPage.css

```

h1 {
  color: #343a40;
  font-size: 2.5rem;
  margin-bottom: 20px;
}

.details {
  background-color: #f8f9fa;
  padding: 20px;
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

.details p {
  font-size: 1rem;
  margin-bottom: 10px;
}

.gallery img {
  width: 300px;
  height: auto;
  margin-bottom: 10px;
}

.description {

```

```

background-color: #fff;
padding: 20px;
border-radius: 5px;
box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
margin-top: 20px;
}

```

Листинг В.8 – Код файла стилей MainPage.css

```

.image{
    margin-bottom:10px;
}

.card-img-top{
    height: 300px;
    overflow: hidden;
}

.info h3{
    font-size:21px;
    margin-bottom:0;
}

.card-body .card-text {
    margin-bottom: 5px;
}

.col-md-4{
    margin-bottom:10px;
}

```

Листинг В.9 – Код js-файла logReg.js

```

document.addEventListener('DOMContentLoaded', function () {
    var regForm = document.getElementById('regForm');
    var passwordInput = document.getElementById('regPassword');
    var confirmPasswordInput =
document.getElementById('confirmPassword');
    var passwordError = document.getElementById('passwordError');
    var regUserNameInput = document.getElementById('regUsername');

    //Функция, ограничивающая ввод пробелов и символов, отличных от
английских букв и спец символов
    function restrictInput(event) {
        // Получаем введенный символ
        var char = event.key;

        //Разрешаем ввод только для заданных символов, исключая пробелы
        var allowedCharacters = /[a-zA-Z0-9 !"#$$%&'()*+,-
.\/:;<=>?@[\\]\^_`{|}~]/;
        if (!allowedCharacters.test(char) || char === ' ') {

```

```

        // Если введенный символ не разрешен или это пробел,
        предотвращаем его добавление
        event.preventDefault();
    }
}
//Функция для проверки ограничений на пароль
function validatePassword() {
    var password = passwordInput.value;
    //var confirmPassword = confirmPasswordInput.value;

    //Проверка на соответствие ограничениям
    var passwordRequiresNonAlphanumeric = /[^\W_]/.test(password);
    var passwordRequiresDigit = /\d/.test(password);
    var passwordRequiresUpper = /[A-Z]/.test(password);
    var passwordRequiresLower = /[a-z]/.test(password);
    var passwordMaxLengthValid = password.length <= 50;
    var passwordMinLengthValid = password.length >= 8;

    // Установка сообщения об ошибке
    var errorMessage = '';
    if (!passwordRequiresNonAlphanumeric) {
        errorMessage += 'Пароль должен содержать хотя бы один
неалфавитно-цифровой символ. ';
    }
    if (!passwordRequiresDigit) {
        errorMessage += 'Пароль должен содержать хотя бы одну цифру.
';
    }
    if (!passwordRequiresUpper) {
        errorMessage += 'Пароль должен содержать хотя бы одну
заглавную букву. ';
    }
    if (!passwordRequiresLower) {
        errorMessage += 'Пароль должен содержать хотя бы одну
строчную букву. ';
    }
    if (!passwordMaxLengthValid) {
        errorMessage += 'Пароль не должен превышать 50 символов. ';
    }
    if (!passwordMinLengthValid) {
        errorMessage += 'Пароль должен иметь длину не менее 8
символов ';
    }

    //Отображение сообщения об ошибке
    passwordError.textContent = errorMessage;

    return passwordRequiresNonAlphanumeric &&
        passwordRequiresDigit &&
        passwordRequiresUpper &&
        passwordRequiresLower &&
        passwordMaxLengthValid &&
        passwordMinLengthValid;
}
//Подписка на события ввода пароля и подтверждения пароля
passwordInput.addEventListener('input', validatePassword);

```

```

regUserNameInput.addEventListener('keypress', restrictInput);
passwordInput.addEventListener('keypress', restrictInput);
regUserNameInput.addEventListener('keypress', restrictInput);

regForm.addEventListener('submit', function (e) {
    //Проверка пароля перед отправкой формы
    if (!validatePassword()) {
        //Отмена отправки формы
        e.preventDefault();
        //Вывод сообщения об ошибке
        passwordError.textContent = 'Выполните все условия для
пароля.';
    }

    //Проверка совпадения паролей
    if (passwordInput.value !== confirmPasswordInput.value) {
        // Отмена отправки формы
        e.preventDefault();
        //Вывод сообщения об ошибке
        passwordError.textContent = 'Пароли не совпадают.';
    }
});
});

```

Листинг В.10 – Код js-файла MainPage.js

```

document.addEventListener('DOMContentLoaded', function () {
    //Функция, которая позволяет вводить только цифры
    function restrictInput(event) {
        var charCode = event.charCode;
        //Проверка, является ли символ цифрой (charCode 48-57)
        if (charCode !== 0 && (charCode < 48 || charCode > 57)) {
            event.preventDefault();
        }
    }

    //Получение всех полей ввода типа number
    var numberInputs = document.querySelectorAll('input[type="number"]');
    numberInputs.forEach(function (input) {
        //Добавление обработчика события keypress для каждого поля
        input.addEventListener('keypress', restrictInput);
    });

    //Функция, которая позволяет вводить только цифры
    function restrictInput(event) {
        var charCode = event.charCode;
        // Проверка, является ли символ цифрой (charCode 48-57)
        if (charCode !== 0 && (charCode < 48 || charCode > 57)) {
            event.preventDefault();
        }
    }

    //Функция для проверки и замены значений min и max
    function checkAndSwapValues(minInput, maxInput) {
        var minValue = parseInt(minInput.value);

```

```

        var maxValue = parseInt(maxInput.value);
        //Если minValue больше maxValue, меняем их местами
        if (minValue > maxValue) {
            minInput.value = maxValue;
            maxInput.value = minValue;
        }
    }
    //Получение всех полей ввода типа number
    var numberInputs = document.querySelectorAll('input[type="number"]');
    numberInputs.forEach(function (input) {
        //Добавление обработчика события keypress для каждого поля
        input.addEventListener('keypress', restrictInput);
    });

    //Получение полей minPrice и maxPrice
    var minPriceInput = document.getElementById('minPrice');
    var maxPriceInput = document.getElementById('maxPrice');

    //Получение полей minFootage и maxFootage
    var minFootageInput = document.getElementById('minFootage');
    var maxFootageInput = document.getElementById('maxFootage');

    //Получение полей minFloor и maxFloor
    var minFloorInput = document.getElementById('minFloor');
    var maxFloorInput = document.getElementById('maxFloor');

    //Добавление обработчика события blur для полей minPrice и maxPrice
    minPriceInput.addEventListener('blur', function () {
        checkAndSwapValues(minPriceInput, maxPriceInput);
    });
    maxPriceInput.addEventListener('blur', function () {
        checkAndSwapValues(minPriceInput, maxPriceInput);
    });

    //Добавления обработчика события blur для полей minFootage и
    maxFootage
    minFootageInput.addEventListener('blur', function () {
        checkAndSwapValues(minFootageInput, maxFootageInput);
    });
    maxFootageInput.addEventListener('blur', function () {
        checkAndSwapValues(minFootageInput, maxFootageInput);
    });

    //Добавления обработчика события blur для полей minFloor и maxFloor
    minFloorInput.addEventListener('blur', function () {
        checkAndSwapValues(minFloorInput, maxFloorInput);
    });
    maxFloorInput.addEventListener('blur', function () {
        checkAndSwapValues(minFloorInput, maxFloorInput);
    });
});
ymaps.ready(init);
function init() {
    var suggestView = new ymaps.SuggestView('suggest', {
        //Примерный прямоугольник, охватывающий территорию России
        boundedBy: [[41.1851, 19.6389], [82.1710, 180.0]],
    });
}

```

```

        //Ограничить результаты указанной областью
        strictBounds: true,
        //Максимальное количество результатов
        results: 5
    });

    var inputField = document.getElementById('suggest');
    var nameError = document.getElementById('nameError');

    suggestView.events.add('select', function (e) {
        var selectedItem = e.get('item').value;
        inputField.value = selectedItem;

        //Очищаем сообщение об ошибке
        nameError.textContent = '';
    });

    inputField.addEventListener('blur', function () {
        var inputValue = inputField.value;
        if (inputValue) {
            ymaps.suggest(inputValue, {
                //Примерный прямоугольник, охватывающий территорию России
                boundedBy: [[41.1851, 19.6389], [82.1710, 180.0]],
                //Максимальное количество результатов
                results: 5,
                //Ограничение только населенными пунктами
                kind: 'locality'
            }).then(function (items) {
                if (items.length === 0 || !items.some(item => item.value
=== inputValue)) {
                    nameError.textContent = 'Адрес не найден';
                    inputField.value = '';
                } else {
                    nameError.textContent = '';
                }
            });
        } else {
            nameError.textContent = 'Адрес не найден';
        }
    });
}

```

Листинг В.11 – Код js-файла pagination.js

```

document.addEventListener("DOMContentLoaded", function () {
    var currentPage = 1;
    //Количество апартаментов на одной странице
    var apartmentsPerPage = 10;

    //Обработчик для кнопки "Предыдущая страница"
    document.getElementById("prevPage").addEventListener("click",
function () {
    if (currentPage > 1) {
        currentPage--;
        updateApartments();
    }
}

```

```

    }
  });

  //Обработчик для кнопки "Следующая страница"
  document.getElementById("nextPage").addEventListener("click",
function () {
    //Здесь нужно проверить, есть ли еще апартаменты на следующей
    //странице
    //Если да, то увеличить currentPage и обновить отображение
    //апартаментов
    currentPage++;
    updateApartments();
  });

  //Функция для обновления отображения апартаментов на текущей странице
  function updateApartments() {
    //Здесь вы должны получить апартаменты для текущей страницы с
    //помощью AJAX-запроса
    //После получения данных обновите содержимое вашего контейнера с
    //апартаментами
    //Например:
    var startIndex = (currentPage - 1) * apartmentsPerPage;
    var endIndex = startIndex + apartmentsPerPage;
    var currentApartments = Model.Apartments.slice(startIndex,
    endIndex);

    //Обновление содержимого контейнера с апартаментами
    var apartmentsContainer =
document.getElementById("apartmentsContainer");
    apartmentsContainer.innerHTML = ""; // Очистка содержимого
    currentApartments.forEach(function (apartment) {
      //Добавление апартамента в контейнер

apartmentsContainer.appendChild(createApartmentElement(apartment));
    });

    //Обновление номера текущей страницы
    document.getElementById("currentPage").textContent = currentPage;
  }

  //Функция для создания HTML-элемента для апартамента
  function createApartmentElement(apartment) {
    //Создание элемента и добавление необходимых данных
    var apartmentElement = document.createElement("div");
    apartmentElement.classList.add("apartment");
    //Здесь добавьте остальную разметку для отображения информации об
    //апартаментах
    return apartmentElement;
  }
  //Инициализация страницы
  updateApartments();
});

```

Листинг В.12 – Код js-файла userProfile.js

```

document.addEventListener('DOMContentLoaded', function () {
    var adForm = document.getElementById('postForm');
    var nameInput = document.getElementById('suggest');
    var descriptionInput = document.getElementById('description');
    var nameError = document.getElementById('nameError');
    var descriptionError = document.getElementById('descriptionError');

    function validateInput(input, errorElement) {
        var value = input.value;

        //Проверка на соответствие ограничениям
        var startsWithSpace = /^\s/.test(value);
        var containsLetter = /[a-zA-Za-яА-Я]/.test(value);

        //Установка сообщения об ошибке
        var errorMessage = '';
        if (startsWithSpace) {
            errorMessage += 'Поле не должно начинаться с пробела. ';
        }
        if (!containsLetter) {
            errorMessage += 'Поле должно содержать хотя бы одну букву. ';
        }

        //Отображение сообщения об ошибке
        errorElement.textContent = errorMessage;

        //Возвращает true, если нет ошибок, иначе false
        return !startsWithSpace && containsLetter;
    }

    nameInput.addEventListener('input', function () {
        validateInput(nameInput, nameError);
    });
    descriptionInput.addEventListener('input', function () {
        validateInput(descriptionInput, descriptionError);
    });

    //Подписка на событие отправки формы
    adForm.addEventListener('submit', function (e) {
        //Проверка полей перед отправкой формы
        var isNameValid = validateInput(nameInput, nameError);
        var isDescriptionValid = validateInput(descriptionInput,
descriptionError);

        //Если одно из полей не прошло проверку, отменяем отправку формы
        if (!isNameValid || !isDescriptionValid) {
            e.preventDefault();
        }
    });
});

```

Приложение Г
(рекомендуемое)
Тестирование приложения

Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Пароль должен иметь длину не менее 8 символов

Зарегистрироваться

Рисунок Г.1 – Результат тест-кейса №2 классов эквивалентности, пароль слишком короткий

Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Пароль должен содержать хотя бы одну заглавную букву.

Зарегистрироваться

Рисунок Г.2 – Результат тест-кейса №3 классов эквивалентности, пароль не содержит заглавных букв

Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Пароль должен содержать хотя бы одну заглавную букву.

Зарегистрироваться

Рисунок Г.3 – Результат тест-кейса №4 классов эквивалентности, пароль содержит только русские заглавные буквы

Регистрация

Имя пользователя:

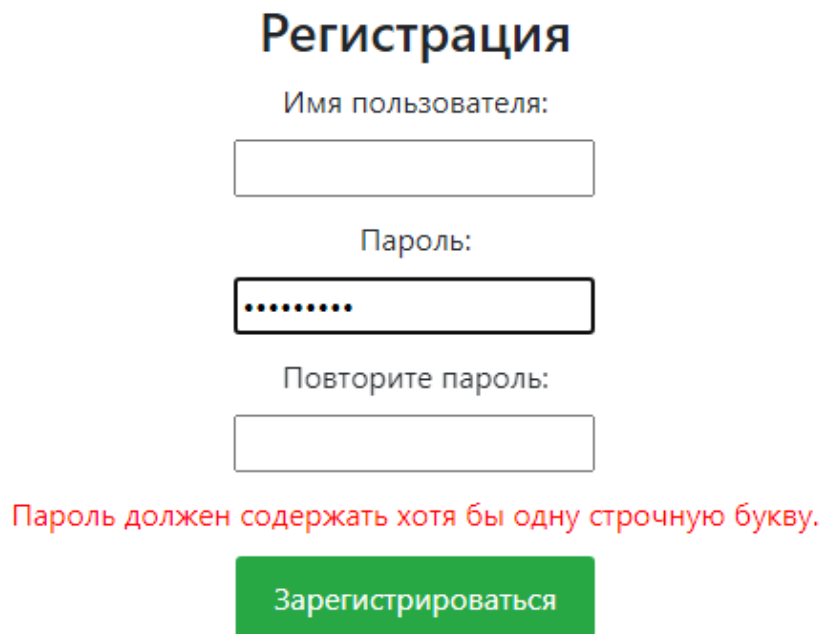
Пароль:

Повторите пароль:

Пароль должен содержать хотя бы один неалфавитно-цифровой символ.

Зарегистрироваться

Рисунок Г.4 – Результат тест-кейса №5 классов эквивалентности, пароль не содержит неалфавитно-цифровых символов



Регистрация

Имя пользователя:

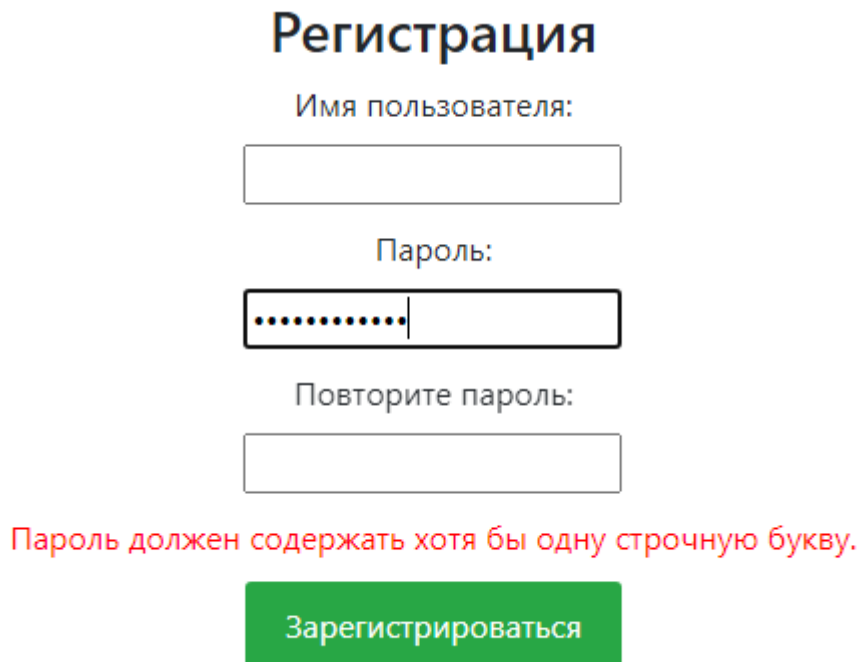
Пароль:

Повторите пароль:

Пароль должен содержать хотя бы одну строчную букву.

Зарегистрироваться

Рисунок Г.5 – Результат тест-кейса №6 классов эквивалентности, пароль не содержит строчных букв



Регистрация

Имя пользователя:

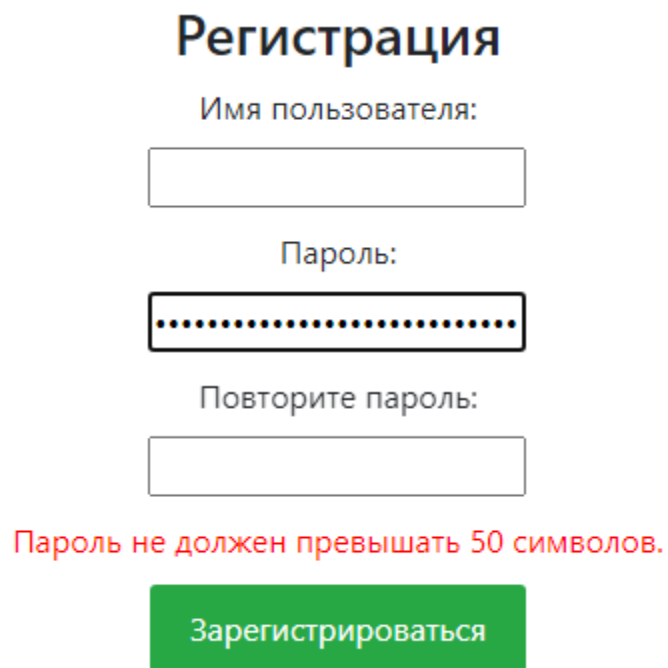
Пароль:

Повторите пароль:

Пароль должен содержать хотя бы одну строчную букву.

Зарегистрироваться

Рисунок Г.6 – Результат тест-кейса №7, пароль содержит только русские строчные буквы



Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Пароль не должен превышать 50 символов.

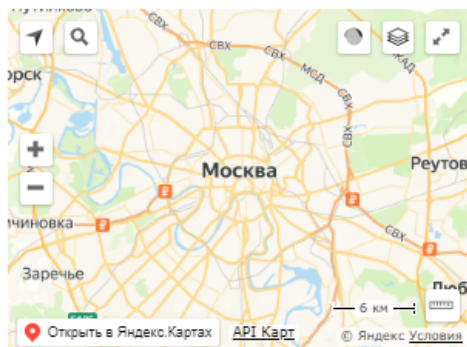
Зарегистрироваться

Рисунок Г.7 – Результат тест-кейса №8 классов эквивалентности, пароль превышает предел в 50 символов

Адрес:

@ \$

Поле должно содержать хотя бы одну букву.



Описание:

@ \$

Поле должно содержать хотя бы одну букву.

Этаж:

-1

Тип Застройки:

Новостройка

Value must be greater than or equal to 0.

Количество комнат:

Студия

Цена:

-1

Площадь:

-1

Тип Объявления:

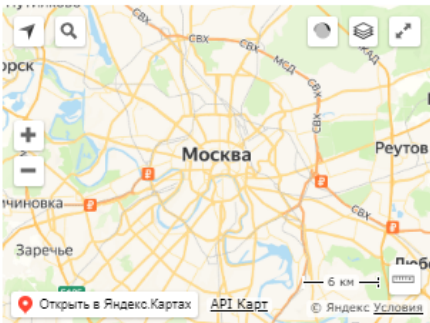
Продажа

Рисунок Г.8 – Результат тест-кейса №1 таблицы решений, все значения недопустимые

Адрес:

Поле должно содержать хотя бы одну букву.

! Please fill out this field.



Описание:

Поле должно содержать хотя бы одну букву.

Этаж:

Тип Застройки:

Новостройка

Количество комнат:

Студия

Цена:

Площадь:

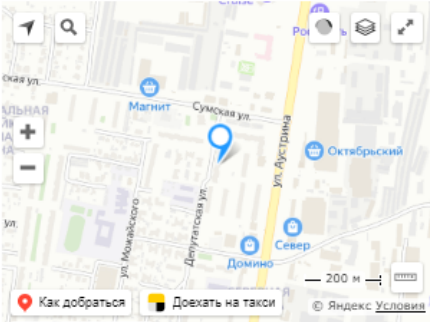
Тип Объявления:

Продажа

Рисунок Г.9 – Результат тест-кейса №2 таблицы решений, все значения отсутствуют

Адрес:

Пенза, улица Аустрина, 162



Описание:

Описание

Этаж:

8

Тип Застройки:

Новостройка

Количество комнат:

Студия

Цена:

9999999999999999

Площадь:

9999999999999999

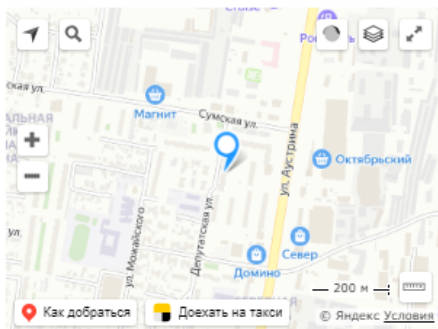
Тип Объявления:

Продажа

Выберите файлы: [Choose Files](#) No file chosen

Рисунок Г.10 – Результат тест-кейса №3 таблицы решений, недопустимые значения цены и площади

Адрес:



Описание:

Этаж:

Тип Застройки:

Количество комнат:

Цена:

Площадь:

Тип Объявления:

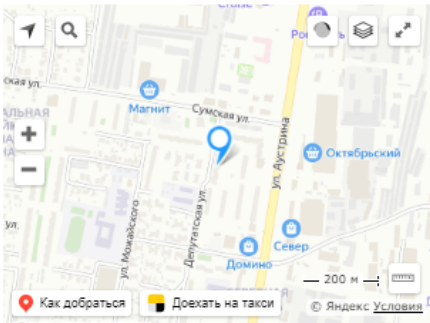
! Value must be greater than or equal to 1.

Выберите файлы: No file chosen

Рисунок Г.11 – Результат тест-кейса №4 таблицы решений, недопустимое значение площади

Адрес:

Пенза, улица Аустрина, 162



Описание:

Описание

Этаж:

8

Тип Застройки:

Новостройка

Количество комнат:

Студия

Цена:

Площадь:

45

Тип Объявления:

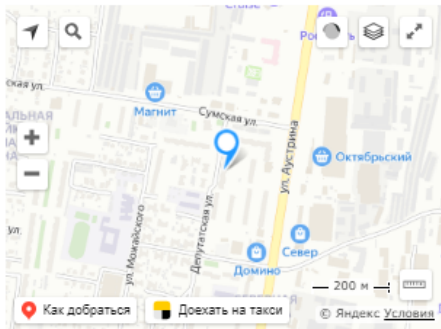
Продажа

Please fill out this field.

Рисунок Г.12 – Результат тест-кейса №5 таблицы решений, значение цены отсутствует

Адрес:

Пенза, улица Аустроина, 162



Описание:

Поле должно содержать хотя бы одну букву.

Please fill out this field.

Этаж:

8

Тип Застройки:

Новостройка

Количество комнат:

Студия

Цена:

14000

Площадь:

45

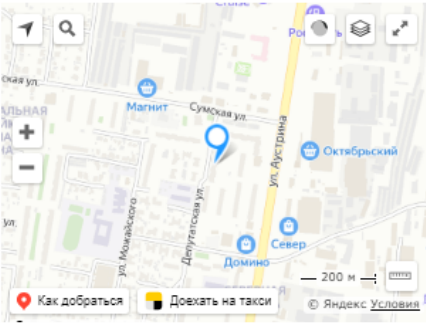
Тип Объявления:

Продажа

Рисунок Г.13 – Результат тест-кейса №6 таблицы решений, описание отсутствует, остальные значения верные

Адрес:

Пенза, улица Аустрина, 162



Описание:

Описание

Этаж:

999999999999999

Тип Застройки:

Новостройка

Количество комнат:

Студия

Цена:

14000

Площадь:

45

Тип Объявления:

Продажа

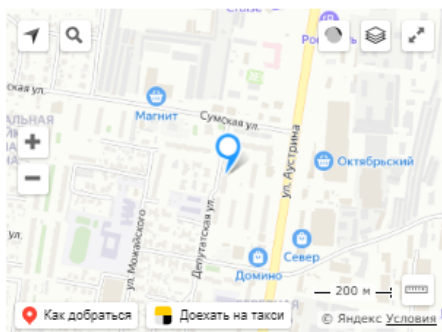
Выберите файлы: No file chosen

Value must be less than or equal to 2147483647.

Рисунок Г.14 – Результат тест-кейса №7 таблицы решений, недопустимое значение этажа, остальные значения верные

Адрес:

Пенза, улица Аустрина, 162



Описание:

Описание

Этаж:

|

Тип Застройки:

Новостройка

! Please fill out this field.

Количество комнат:

Студия

Цена:

14000

Площады:

45

Тип Объявления:

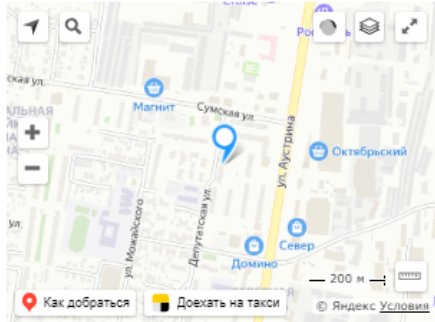
Продажа

Рисунок Г.15 – Результат тест-кейса №8 таблицы решений, значение этажа отсутствует, остальные значения верные

Объявление успешно добавлено.

Адрес:

Пенза, улица Аустрина, 162



Описание:

<Информация>

Этаж:

6

Тип Застройки:

Новостройка

Количество комнат:

Студия

Цена:

14000

Площадь:

45

Тип Объявления:

Продажа

Рисунок Г.16 – Результат тест-кейса №9 таблицы решений, все значения верные, объявление добавлено успешно

Регистрация

Имя пользователя:

newbieNate1234

Пароль:

.....

Повторите пароль:

.....

Зарегистрироваться

Рисунок Г.17 – Результат теста №1, ввод верных данных для регистрации

Продажа и аренда недвижимости

Главная

Личный кабинет

Выйти

Главная

Регистрация выполнена успешно

Добро пожаловать, newbieNate1234!

Было найдено 37 объявлений

Адрес:

Площадь от:

0

Площадь до:

2147483647

Цена от:

0

Цена до:

2147483647

Этаж от:

0

Этаж до:

2147483647

Количество комнат от:

Студия

Нет Фото

Киевская область, Вышгородский район, Чернобыль

Продажа

Этаж: 1

Вторичка

1 комнатная

1 100 м²

3 700 ₴

Подробнее

Донецк, Авдотьино

Продажа

Этаж: 9

Вторичка

Студия

2 442 м²

3 700 ₴

Подробнее

48 мм

32 мм

Экватор

Продажа

Этаж: 9

Новостройка

Студия

2 442 м²

3 670 ₴

Подробнее

Рисунок Г.18 – Результат теста №1, сообщение об успешной регистрации

Добро пожаловать на наш сайт

Вход

Имя пользователя:

Пароль:

Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Рисунок Г.19 – Результат теста №2, ввод верных данных для входа

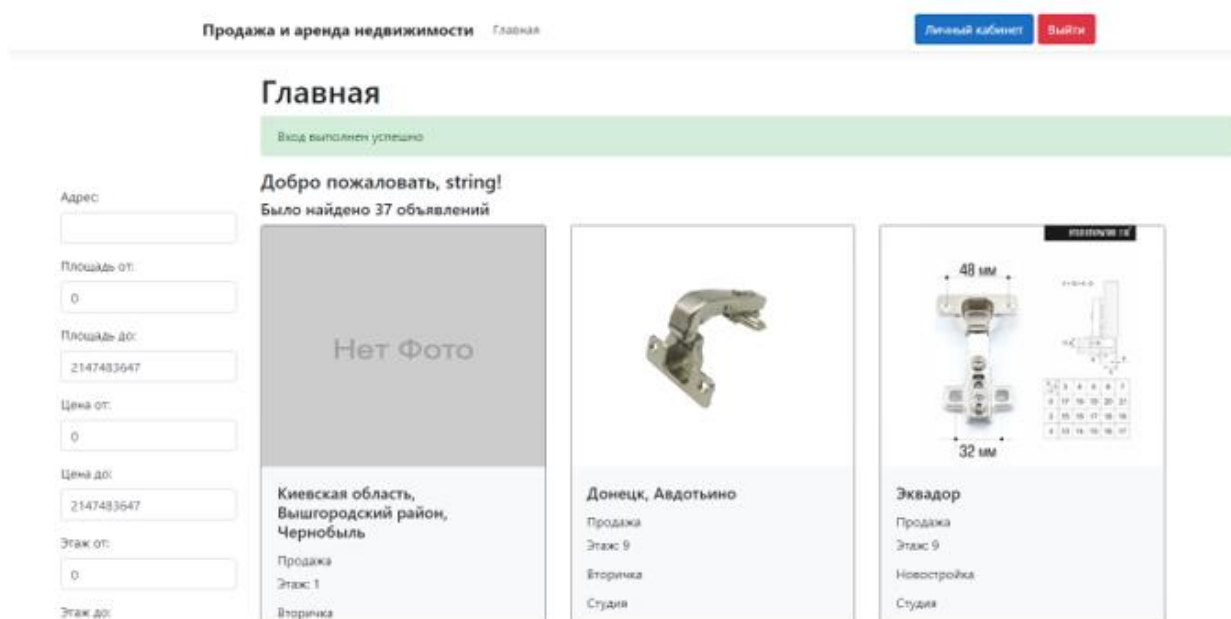
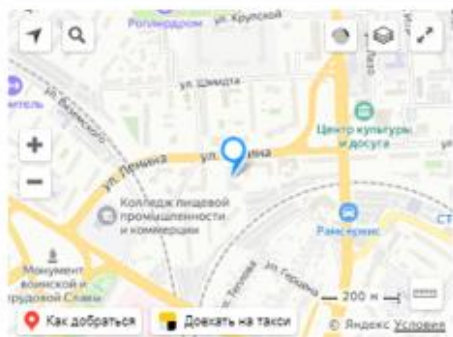


Рисунок Г.20 – Результат теста №2, сообщение об успешном входе

Добавление

Адрес:

Пенза, улица Ленина, 18



Описание:

+79061578166, звонить сюда

Этаж:

3

Тип Застройки:

Вторичка

Количество комнат:

2 комнатная

Цена:

3500000

Площады:

45

Тип Объявления:

Продажа

Выберите файлы: Choose Files недвижка1_2.jpg

Добавить

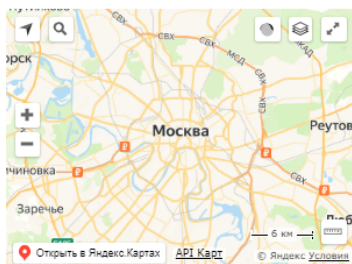
Рисунок Г.21 – Результат теста №3, ввод верных данных для добавления объявления

Пенза, улица
Ленина, 18 +79061578166, звонить сюда 3 Вторичка 2 комнатная 3 500 000 Р 45 м² Продажа 12.06.2024 - Удалить Подробнее

Добавление

Объявление успешно добавлено.

Адрес:



Описание:

Рисунок Г.22 – Результат теста №3, сообщение об успешном добавлении

localhost:44388 says
Это действие нельзя отменить. Вы уверены?

OK Cancel

Продажа и аренда недвиж...

Личный кабинет

Добавленные объявления

Адрес	Описание	Этаж	Тип застройки	Количество комнат	Цена	Площадь	Тип объявления	Дата добавления	Средняя оценка	Действие	Перейти к странице
Киевская область, Вышгородский район, Чернобыль	Черновая	1	Вторичка	1 комнатная	3 700 Р	1 100 м ²	Продажа	05.05.2024	-	Удалить	Подробнее
Донецк, Авдотьино	Чистовая	9	Вторичка	Студия	3 700 Р	2 442 м ²	Продажа	05.05.2024	-	Удалить	Подробнее
Эквадор	Чистовая	9	Новостройка	Студия	3 670 Р	2 442 м ²	Продажа	05.05.2024	-	Удалить	Подробнее

Рисунок Г.23 – Результат теста №4, нажатие на кнопку «Удалить»

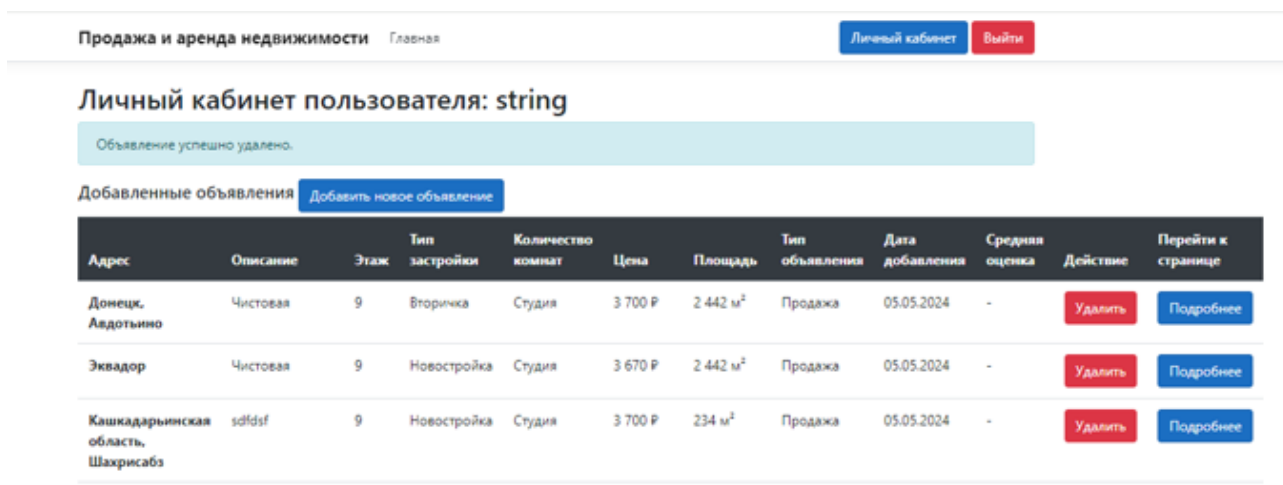


Рисунок Г.24 – Результат теста №4, сообщение об успешном добавлении

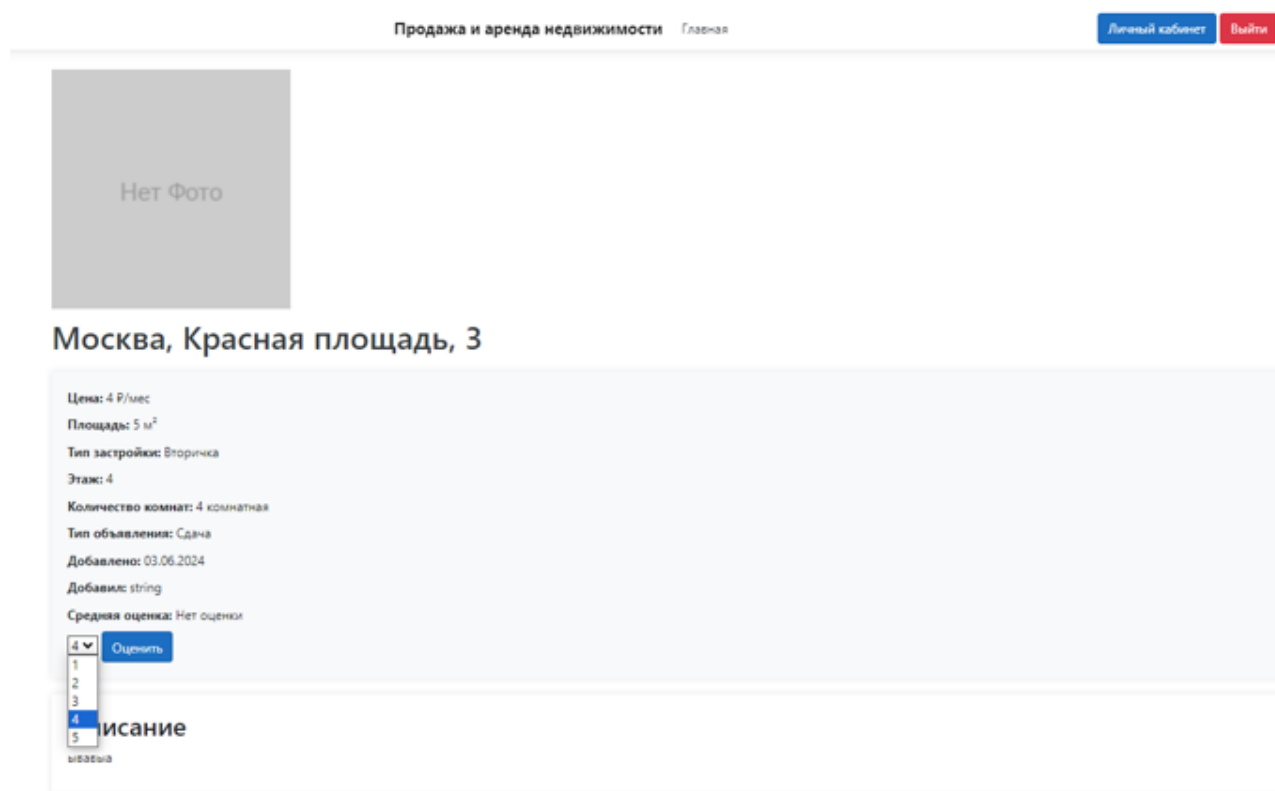


Рисунок Г.25 – Результат теста №5, выставление оценки «4»

Продажа и аренда недвижимости Главная

Нет Фото

Москва, Красная площадь, 3

Цена: 4 Р/мес
Площадь: 5 м²
Тип застройки: Вторичка
Этаж: 4
Количество комнат: 4 комнатная
Тип объявления: Сдача
Добавлено: 03.06.2024
Добавил: string

Оценка успешно добавлена

Средняя оценка: 4
1 Оценить

Описание

ыааааа

Рисунок Г.26 – Результат теста №5, сообщение об успешном добавлении объявления

Адрес:

Площадь от:

Площадь до:

Цена от:

Цена до:

Этаж от:

Этаж до:

Количество комнат от:

Количество комнат до:

Тип застройки:
☐ Новостройка
☐ Вторичное жилье

Тип объявления:
☐ Продажа
☒ Сдача

Рисунок Г.27 – Результат теста №6, ввод данных в поля для фильтров

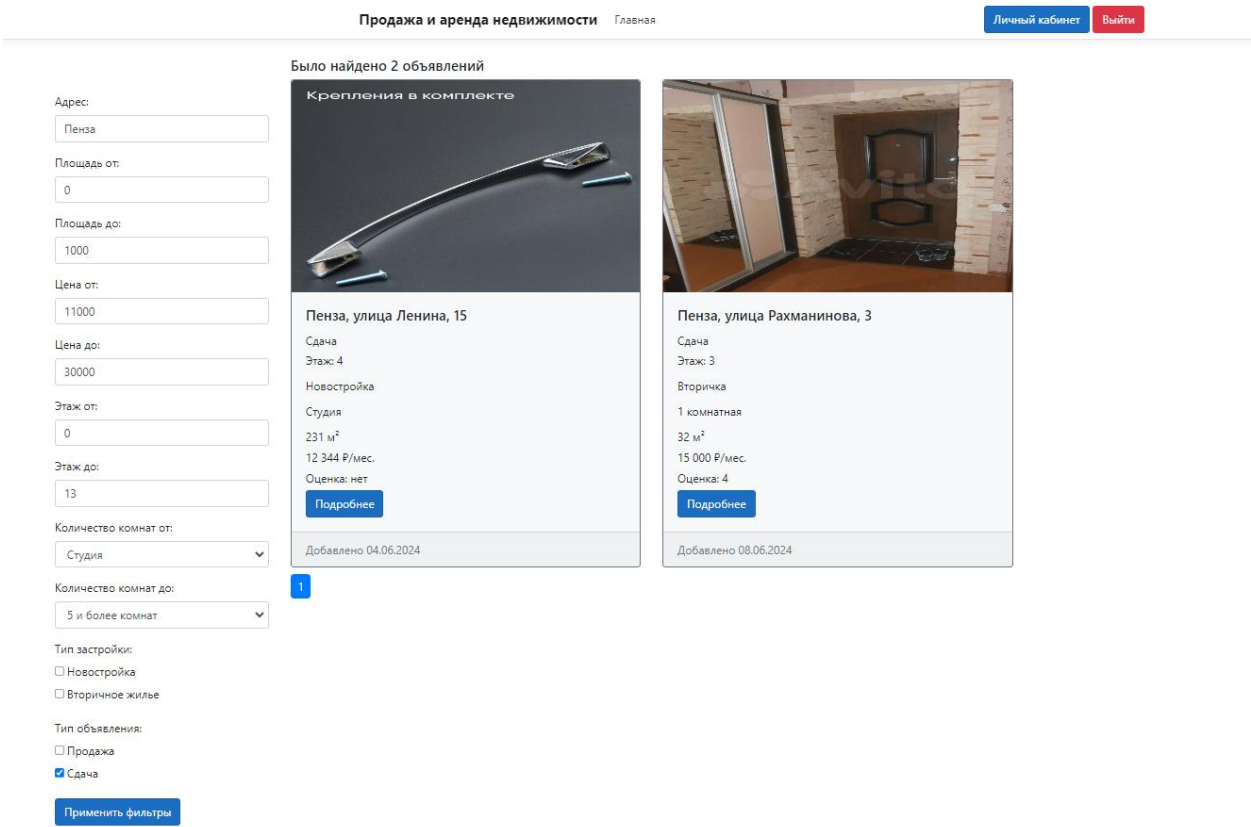


Рисунок Г.28 – Результат теста №6, вывод данных согласно введенным фильтрам

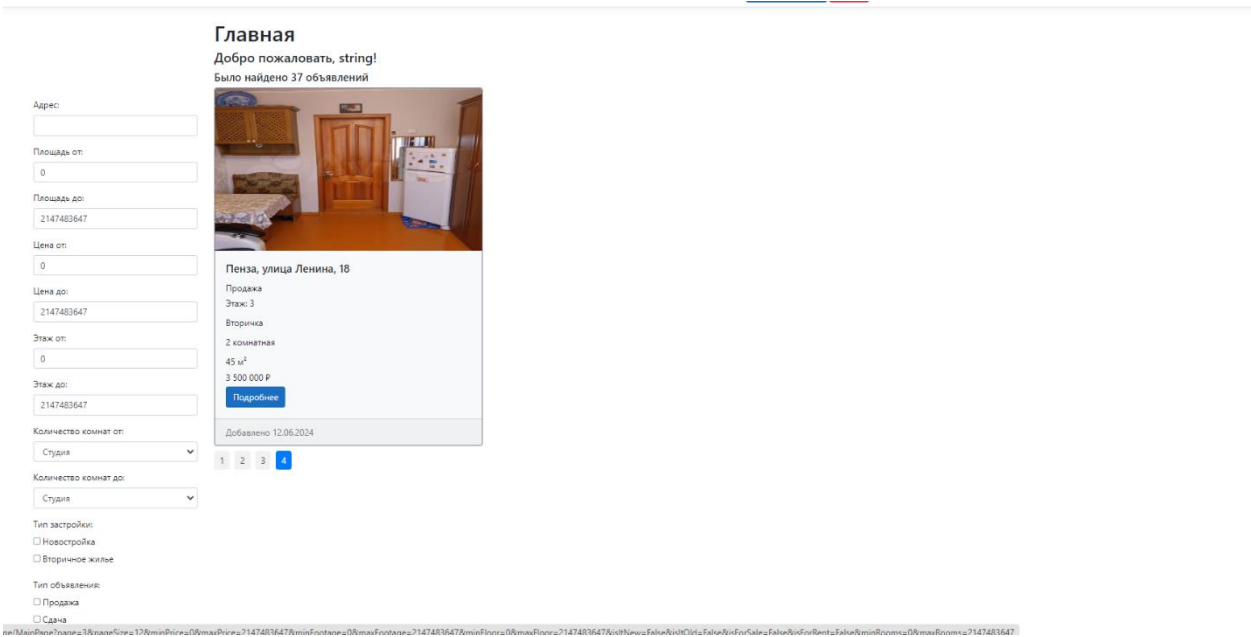


Рисунок Г.29 – Результат теста №7, содержимое страницы №4

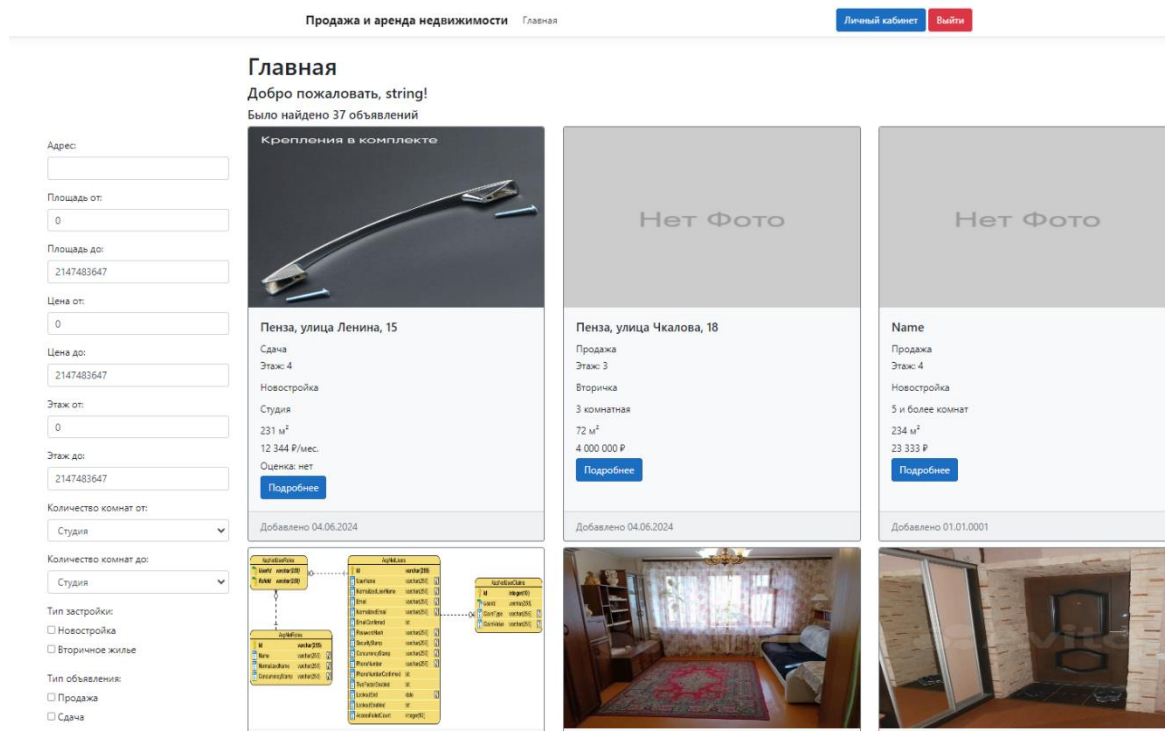


Рисунок Г.30 – Результат теста №7, содержимое страницы №3

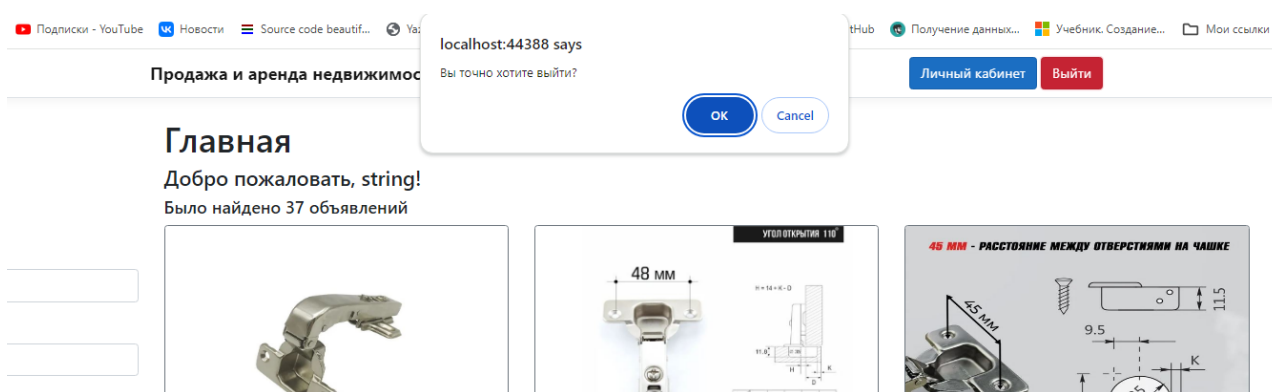


Рисунок Г.31 – Результат теста №8, нажатие кнопки «Выйти»

Добро пожаловать на наш сайт

Выход из аккаунта прошел успешно

Вход

Имя пользователя:

Пароль:

Войти

Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Зарегистрироваться

Рисунок Г.32 – Результат теста №8, сообщение об успешном выходе

Продажа и аренда недвижимости Главная Личный кабинет Логин/Регистрация

Главная
Добро пожаловать, Аноним!
Было найдено 37 объявлений

Адрес:

Площадь от:

Площадь до:

Цена от:

Цена до:

Этаж от:

Этаж до:

Количество комнат от:

Количество комнат до:

Тип застройки:

Донецк, Авдотьино
Продажа
Этаж: 9
Вторичка
2 442 м²
3 700 Р
[Подробнее](#)
Добавлено 05.05.2024

Эквадор
Продажа
Этаж: 9
Новостройка
2 442 м²
3 670 Р
[Подробнее](#)
Добавлено 05.05.2024

Кашкадарьинская область, Шахрисабз
Продажа
Этаж: 9
Новостройка
234 м²
3 700 Р
[Подробнее](#)
Добавлено 05.05.2024

Рисунок Г.33 – Результат теста №9, попытка неавторизованного пользователя зайти в личный кабинет

Добро пожаловать на наш сайт

Войдите или зарегистрируйтесь, чтобы получить доступ к личному кабинету

Вход

Имя пользователя:

Пароль:

Войти

Регистрация

Имя пользователя:

Пароль:

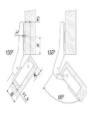
Повторите пароль:

Зарегистрироваться

Рисунок Г.34 – Результат теста №9, сообщение о том, что необходимо войти или зарегистрироваться для использования ЛК

Параметры для установки /
Parameters for installation

D	6	4	5	6	7
H	8	13	4	47	14
	2	13	1	12	14



sdfdfs

Цена: 3700 Р/мес

Площадь: 234 м²

Тип застройки: Новостройка

Этаж: 7

Количество комнат: Студия

Тип объявления: Сдача

Добавлено: 07.05.2024

Добавил: string

Средняя оценка: 3

1

Оценить

Рисунок Г.35 – Результат теста №10, попытка неавторизованного пользователя поставить оценку объявлению

Добро пожаловать на наш сайт

Войдите или зарегистрируйтесь, чтобы ставить оценки

Вход

Имя пользователя:

Пароль:

Войти

Регистрация

Имя пользователя:

Пароль:

Повторите пароль:

Зарегистрироваться

Рисунок Г.36 – Результат теста №10, сообщение о том, что необходимо войти или зарегистрироваться для выставления оценки