Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Волгоградский государственный технический университет»

|  |  |
| --- | --- |
| Факультет | Электроники и вычислительной техники |
| Кафедра | Программное обеспечение автоматизированных систем |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Согласовано | | | | | | | | |  | Утверждаю | | | | | | | | |
|  | | | | | | | | |  | и.о. зав. кафедрой | | | | | | | | |
| (должность гл. специалиста предприятия) | | | | | | | | |  |
|  | | | |  |  | | | |  |  | | | |  | О. А. Сычев | | | |
| (подпись) | | | |  | (инициалы, фамилия) | | | |  | (подпись) | | | |  | (инициалы, фамилия) | | | |
| « |  | » |  | | | 20 |  | г. |  | « |  | » |  | | | 20 |  | г. |

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| к | выпускной квалификационной работе бакалавра | | | | | | | | | | | | | | | на тему |
| (наименование вида работы) | | | | | | | | | | | | | | | | |
| Разработка веб-сервиса для анализа и сравнения условий аренды | | | | | | | | | | | | | | | | |
| квартир | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | |
|  | |  | | | | | |  |  | | | | | | | |
| Автор | |  | | | | | |  | Аврамова Елизавета Владимировна | | | | | | | |
|  | | (подпись и дата подписания) | | | | | |  | (фамилия, имя, отчество) | | | | | | | |
| Обозначение | | | ВКРБ–09.03.04–10.19–01–25 | | | | | | |  | | | | | | |
|  | | | (код документа) | | | | |  | | | | | | | | |
| Группа | | | ПрИн-467 | | | | |  | | | | | | | | |
|  | | | (шифр группы) | | | | |  | | | | | | | | |
| Направление | | | 09.03.04 – Программная инженерия,  Разработка программно-информационных систем | | | | | | | | | | | | | |
|  | | | (код и наименование направления, наименование программы (профиля)) | | | | | | | | | | | | | |
| Руководитель работы | | | | |  | | | | | | | |  | | Гилка В.В. | |
|  | | | | | (подпись и дата подписания) | | | | | | | |  | | (инициалы и фамилия) | |
| Консультанты по разделам: | | | | | | | | | | | | | | | | |
|  | | | | | |  |  | | | | |  | |  | | |
| (краткое наименование раздела) | | | | | |  | (подпись и дата подписания) | | | | |  | | (инициалы и фамилия) | | |
|  | | | | | |  |  | | | | |  | |  | | |
| (краткое наименование раздела) | | | | | |  | (подпись и дата подписания) | | | | |  | | (инициалы и фамилия) | | |
| Нормоконтролер: | | | |  | | | | | | |  | Кузнецова А.С. | | | | |
|  | | | | (подпись и дата подписания) | | | | | | |  | (инициалы и фамилия) | | | | |

Волгоград 2025 г.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Волгоградский государственный технический университет»

|  |  |
| --- | --- |
| Кафедра | Программное обеспечение автоматизированных систем |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Утверждаю | | | | | и.о. зав. кафедрой | | | |
|  |  | | | |  | О. А. Сычев | | | |
| (подпись) | | | |  | (инициалы, фамилия) | | | |
|  | « |  | » |  | | | 20 |  | г. |

**Задание**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| на | выпускную квалификационную работу бакалавра | | | | | | | | | | | | | | |
|  | (наименование вида работы) | | | | | | | | | | | | | | |
| Студент | | | Аврамова Елизавета Владимировна | | | | | | | | | | | | |
|  | (фамилия, имя, отчество) | | | | | | | | | | | | | | |
| Код кафедры | | | | 10.19 | Группа | | | | ПрИн-467 | | | |  | | |
| Тема | | Разработка веб-сервиса для анализа и сравнения условий аренды | | | | | | | | | | | | | |
| квартир | | | | | | | | | | | | | | | |
| Утверждена приказом по университету | | | | | | « | 23 | » | | августа | 20 | 24 | | г. № | 1105-ст |
| Срок представления готовой работы (проекта) | | | | | | | | | |  | | | | | |
|  | | | | | | | | | | (дата, подпись студента) | | | | | |
| Исходные данные для выполнения работы (проекта) | | | | | | | | | | | | | | | |
| Задание, выданное научным руководителем кафедры «ПОАС» | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
| Содержание основной части пояснительной записки | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | |

|  |
| --- |
|  |
|  |
|  |
|  |
|  |

|  |  |
| --- | --- |
| Перечень графического материала | |
| 1) |  |
|  | |
| 2) |  |
|  | |
| 3) |  |
|  | |
| 4) |  |
|  | |
| 5) |  |
|  | |
| 6) |  |
|  | |
| 7) |  |
|  | |
| 8) |  |
|  | |
| 9) |  |
|  | |
| 10) |  |
|  | |
| 11) |  |
|  | |
| 12) |  |
|  | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель работы (проекта) | |  | |  | | Гилка В.В. | |
|  | | (подпись и дата подписания) | |  | | (инициалы и фамилия) | |
| Консультанты по разделам: | |  | | | |  | |
|  |  | |  | |  | |  |
| (краткое наименование раздела) |  | | (подпись и дата подписания) | |  | | (инициалы и фамилия) |
|  |  | |  | |  | |  |

Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Кафедра «Программное обеспечение автоматизированных систем»

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ:  и.о. зав. кафедрой ПОАС  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_О.А. Сычев  «\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г. |

Разработка веб-сервиса для анализа и сравнения условий аренды квартир

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

ВКРБ–09.03.04–10.19–01–25–81

Листов 65

|  |  |
| --- | --- |
|  | Руководитель работы  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Гилка В.В.  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_20 г. |
| |  | | --- | | Нормоконтролер  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кузнецова А.С.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20 г. | | |  | | --- | | Исполнитель  студент группы ПрИн-467  \_\_\_\_\_\_\_\_\_\_\_\_Аврамова Е. В. «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_ 20 г. | |

Волгоград, 2025 г.

Аннотация

Настоящий документ является пояснительной запиской к выпускной квалификационной работе бакалавра на тему: «Разработка веб-сервиса для анализа и сравнения условий аренды недвижимости».

В работе обосновывается актуальность выбранной темы, проводится ….

Документ включает в себя страниц -…, рисунков - …., приложений -...

Ключевые слова: …..

Содержание

[Введение 9](#_Toc198684858)

[1 Анализ современного состояния вопроса в области анализа и сравнения условий аренды квартир 10](#_Toc198684859)

[1.1 Особенности предметной области: аренда жилья в цифровую эпоху 12](#_Toc198684860)

[1.2 Анализ современных методов анализа и сравнения условий аренды недвижимости 13](#_Toc198684861)

[1.3 Обзор существующих сервисов для анализа и сравнения условий аренды недвижимости 14](#_Toc198684862)

[1.4 Проблемы и недостатки существующих решений для анализа и сравнения условий аренды недвижимости 14](#_Toc198684863)

[1.5 Описание существующего процесса решения задачи для анализа и сравнения условий аренды недвижимости 15](#_Toc198684864)

[2 Теоретические аспекты разработки веб-сервиса для анализа и сравнения условий аренды недвижимости 17](#_Toc198684865)

[2.1 Предлагаемый процесс решения задачи анализа и сравнения условий аренды недвижимости 17](#_Toc198684866)

[2.2 Формальная модель для описания проблемы анализа и сравнения условий аренды недвижимости 17](#_Toc198684867)

[2.3 Описание алгоритма для анализа и сравнения условий аренды недвижимости 17](#_Toc198684868)

[2.4 Разработка функциональных требований к веб-сервису для анализа и сравнения условий аренды недвижимости 17](#_Toc198684869)

[Выводы 17](#_Toc198684870)

[3 Проектирование и разработка веб-сервиса для анализа и сравнения условий аренды недвижимости 17](#_Toc198684871)

[3.1 Выбор технологий и инструментов для разработки веб-сервиса анализа и сравнения условий аренды недвижимости 17](#_Toc198684872)

[3.1.1 Клиентская часть (Frontend) 18](#_Toc198684873)

[3.1.2 Связь клиент–сервер 19](#_Toc198684874)

[3.1.3 Серверная часть (Backend) 20](#_Toc198684875)

[3.1.4. Хранение данных 22](#_Toc198684876)

[3.1.5 Очереди задач и обработка парсинга 23](#_Toc198684877)

[3.1.6 Парсеры 25](#_Toc198684878)

[3.2 Проектирование интерфейса пользователя веб-сервиса для анализа и сравнения условий аренды недвижимости 27](#_Toc198684879)

[3.3 Архитектура веб-сервиса для анализа и сравнения условий аренды недвижимости 37](#_Toc198684880)

[3.3.1 Общий обзор архитектуры 37](#_Toc198684881)

[3.3.2 Последовательность взаимодействия компонентов 39](#_Toc198684882)

[3.3.3 Особенности реализации 41](#_Toc198684883)

[3.3.4 Механизмы защиты и устойчивости 42](#_Toc198684884)

[3.3.5 Поддержка и расширение 43](#_Toc198684885)

[3.4 Интеграция с платформами аренды квартир для анализа и сравнения условий аренды квартир 45](#_Toc198684886)

[3.4.1 Цели и задачи интеграции 46](#_Toc198684887)

[3.4.2 Формат интеграции: через парсеры 48](#_Toc198684888)

[3.4.3 Работа с защитой от блокировок 49](#_Toc198684889)

[3.4.4 Унификация и нормализация данных 51](#_Toc198684890)

[3.4.5 Дедупликация объявлений 53](#_Toc198684891)

[3.4.6 Переход к внешнему ресурсу 54](#_Toc198684892)

[3.4.7 Расширение списка источников 55](#_Toc198684893)

[Выводы 57](#_Toc198684894)

[4 Тестирование и апробация веб-сервиса для анализа и сравнения условий аренды недвижимости 58](#_Toc198684895)

[4.1 Методологии тестирования и оценки эффективности веб-сервиса для анализа и сравнения условий аренды недвижимости 59](#_Toc198684896)

[4.2 Тестирование веб-сервиса для анализа и сравнения условий аренды недвижимости 61](#_Toc198684897)

[4.3 Оценка эффективности веб-сервиса для анализа и сравнения условий аренды недвижимости 63](#_Toc198684898)

[4.4 Анализ результатов тестирования и оценки эффективности веб-сервиса для анализа и сравнения условий аренды недвижимости 65](#_Toc198684899)

[Выводы 66](#_Toc198684900)

[Заключение 68](#_Toc198684901)

[Список использованных источников 70](#_Toc198684902)

[Приложение А 71](#_Toc198684903)

[Справка о результатах проверки выпускной квалификационной работы на наличие заимствований 71](#_Toc198684904)

[Приложение Б 72](#_Toc198684905)

[Техническое задание 72](#_Toc198684906)

[Приложение В 73](#_Toc198684907)

[Руководство системного программиста 73](#_Toc198684908)

# Введение

Аренда квартир в условиях современного городского рынка становится неотъемлемой частью жизни мобильного населения. Повышенная урбанизация, экономическая нестабильность и рост численности трудоспособного населения усиливают спрос на арендуемое жилье. В то же время, разнообразие предложений, разброс цен и отсутствие прозрачных инструментов анализа значительно затрудняют поиск подходящего варианта для арендатора.

В процессе выбора жилья арендаторы сталкиваются с рядом проблем: дублирующиеся объявления на разных платформах, неполная информация, устаревшие данные, недостаточные фильтры, отсутствие наглядного сравнения. Всё это приводит к временным потерям и снижению эффективности принятия решений.

Целью данной выпускной квалификационной работы является сокращение временных затрат арендаторов на поиск и анализ условий аренды квартир за счёт создания веб-сервиса, предоставляющего инструменты фильтрации, сравнения и анализа объявлений.

Для реализации поставленной цели необходимо было выполнить ряд последовательных задач, каждая из которых решает конкретный аспект общей проблемы.

Во-первых, требовалось изучить текущее состояние рынка аренды недвижимости и провести обзор существующих цифровых инструментов, с помощью которых пользователи осуществляют поиск жилья. Это дало возможность понять ключевые тенденции и определить, с какими трудностями чаще всего сталкиваются пользователи.

Кроме того, был проведён обзор методов фильтрации и механизмов сравнения объявлений, которые используются в отрасли. Это позволило определить, какие из этих решений стоит взять за основу, а какие — усовершенствовать и адаптировать для собственного сервиса.

Следующим важным шагом стало изучение того, какие слабые места и ограничения есть у популярных сервисов аренды — например, ЦИАН, Авито и Яндекс.Недвижимость. Это помогло понять, чего именно не хватает этим платформам, и на этой основе сформулировать, каким должно быть новое решение.

После этого был разработан общий план архитектуры будущего веб-сервиса. Нужно было учесть распределённый способ сбора данных, обеспечить возможность масштабирования и при этом сделать интерфейс удобным и понятным для пользователя.

Отдельно прорабатывалась часть, связанная с разработкой парсера. Он должен легко адаптироваться, если сайты-источники меняют свою структуру, и продолжать собирать данные, даже если есть ограничения или блокировки для автоматических запросов.

На следующем этапе была внедрена система гибкой фильтрации, которая позволяет пользователю настроить поиск по целому ряду параметров, включая цену, площадь, район, этажность и наличие дополнительных условий.

Также была реализована логика интеллектуального сравнения объектов между собой — с учётом веса и приоритетов, заданных пользователем, что делает выбор жилья более осмысленным и персонализированным.

Каждое объявление в системе снабжено внешней ссылкой на оригинальный источник, что позволяет пользователю оперативно перейти к детальной информации и связаться с арендодателем через привычную площадку.

Завершающим этапом стала комплексная проверка работоспособности решения: проведены функциональное тестирование, оценка производительности, а также апробация на реальных пользовательских сценариях.

# 1 Анализ современного состояния вопроса в области анализа и сравнения условий аренды квартир

Разработка цифрового инструмента для поиска и анализа предложений по аренде жилья требует чёткого понимания как самого рынка аренды, так и существующих технических решений, применяемых в этой сфере. Аренда жилья — это не просто подбор квартиры по цене и метражу; это сложный процесс, включающий множество параметров и контекстов, которые важно учитывать при проектировании автоматизированной системы поддержки пользователя.

Современные цифровые платформы призваны упростить процесс выбора жилья, но по факту они не всегда справляются с задачей комплексного анализа предложений. Нередко пользователи сталкиваются с повторяющейся, неструктурированной или устаревшей информацией, а система фильтрации не охватывает все потребности целевой аудитории. Это создаёт предпосылки для разработки более гибких, интеллектуальных и пользовательски-ориентированных решений.

Настоящий раздел посвящён системному обзору ключевых характеристик предметной области, методов анализа данных, действующих платформ, их преимуществ и недостатков. Также в нём раскрываются основные процессы, связанные со сбором, обработкой и представлением информации, и обосновывается необходимость внедрения нового подхода — агрегирующего, фильтрующего и сравнивающего предложения аренды с учётом пользовательских предпочтений и технических ограничений источников.

Рынок аренды жилой недвижимости активно развивается, но современные цифровые сервисы не обеспечивают необходимого уровня интеллектуального анализа и сравнения. Проблемы дублирования, неактуальности и ограниченности фильтров остаются нерешёнными. Предлагаемый веб-сервис учитывает эти недостатки, внедряя гибкую фильтрацию, парсинг с защитой от блокировок, устранение дублей и пользовательские сценарии на основе UX-принципов Нильсена. Всё это позволяет добиться главной цели проекта — сократить время арендатора на поиск и повысить обоснованность выбора жилья.

# Особенности предметной области: аренда жилья в цифровую эпоху

Рынок аренды жилой недвижимости в России претерпел существенные изменения за последние годы. Если раньше основной способ поиска жилья заключался в личных объявлениях, агентствах или рекомендациях знакомых, то сегодня этот процесс почти полностью перешёл в цифровую среду. Онлайн-платформы стали главными посредниками между арендодателями и арендаторами.

Аренда жилья имеет ряд специфических характеристик. С точки зрения пользователя (арендатора), важно не только наличие квартиры, но и её соответствие ряду параметров: стоимость, срок аренды, расположение (район), метраж, количество комнат, этаж, тип дома, возможность проживания с детьми или животными, наличие бытовой техники и мебели. При этом каждый пользователь может иметь индивидуальные приоритеты, что делает задачу выбора квартиры многокритериальной и заведомо субъективной.

Для арендодателя, в свою очередь, важно быстро найти надёжного жильца и выгодно выделиться на фоне конкурентов. Поэтому платформа, на которой происходит взаимодействие, должна обеспечивать качественную подачу информации, прозрачность условий, актуальность объявлений и понятный пользовательский интерфейс.

Несмотря на широкое распространение цифровых платформ, пользователи, ищущие жильё в аренду, продолжают сталкиваться с рядом серьёзных затруднений.

Одна из ключевых проблем — это дублирование объявлений. Один и тот же объект может быть размещён сразу на нескольких площадках, причём с разными характеристиками: отличающейся ценой, неточной информацией о площади, сроке аренды или условиях проживания.

Также часто встречается ситуация, когда данные в объявлении устарели или представлены неполно. Пользователь не находит важных сведений — отсутствуют фотографии, не указан этаж, нет отметки о наличии бытовой техники или разрешений на проживание с животными и детьми.

Большинство существующих платформ не предоставляют инструментов для комплексного сравнения вариантов. Пользователь вынужден самостоятельно открывать несколько вкладок, сопоставлять параметры вручную и помнить, какие фильтры он применял ранее — ведь сохранение истории поиска реализовано далеко не везде.

Фильтрационные системы на популярных сайтах обычно ограничиваются базовыми параметрами, такими как цена и количество комнат. Более детализированные критерии, важные для арендаторов, часто отсутствуют или реализованы неудобно.

Дополнительное затруднение создаёт навязчивая реклама: баннеры, всплывающие окна и платное продвижение объявлений отвлекают внимание, замедляют процесс выбора и в целом ухудшают пользовательский опыт.

В рамках данной работы предметная область рассматривается с фокусом на потребности арендатора, для которого особенно важно минимизировать время, затрачиваемое на поиск подходящего варианта, быть уверенным в актуальности и достоверности представленных данных, иметь возможность быстро и удобно сравнивать предложения по ключевым параметрам, а также при необходимости — оперативно переходить на оригинальные сайты, чтобы связаться с владельцами объектов напрямую.

Понимание этих аспектов легло в основу проектирования функциональности веб-сервиса и выбора архитектурных решений, направленных на упрощение и ускорение процесса аренды с использованием цифровых технологий.

# 1.2 Анализ современных методов анализа и сравнения условий аренды недвижимости

Современные российские платформы по аренде недвижимости используют преимущественно базовые методы обработки и представления информации. На практике это выражается в стандартных механизмах фильтрации по ключевым параметрам: цене, площади, количеству комнат, этажу, району, а также в наличии или отсутствии условий, таких как мебель, бытовая техника, возможность проживания с детьми или животными. Большинство сервисов, включая ЦИАН, Авито, Яндекс.Недвижимость и Домклик, применяют схожую структуру фильтрации, где пользователь вручную задаёт интересующие его параметры, после чего система отображает соответствующие объявления.

При этом возможности для удобного сравнения предложений на большинстве платформ остаются ограниченными. Пользователю часто приходится открывать карточки вручную и самостоятельно сопоставлять данные по разным параметрам, поскольку встроенного механизма сравнения либо нет, либо он представлен в урезанном виде. Это существенно затрудняет принятие решений, особенно в случае большого объёма схожих по характеристикам объявлений.

Дополнительные сложности возникают из-за неактуальной или неполной информации в самих объявлениях: на некоторых площадках отсутствуют фотографии, не указаны этаж или точный адрес, а данные по наличию техники и условиям проживания представлены не всегда. Эти проблемы особенно характерны для агрегаторов с ручным вводом или слабой модерацией контента.

Также стоит отметить, что визуальное представление информации, например, через карты или таблицы, реализовано фрагментарно. Хотя некоторые сервисы предлагают фильтрацию по карте, полноценной визуальной аналитики или наглядного сравнения объектов между собой, как правило, не предусмотрено. Это ограничивает удобство работы пользователя и увеличивает время, необходимое для анализа и выбора подходящего варианта аренды.

# 1.3 Обзор существующих сервисов для анализа и сравнения условий аренды недвижимости

Российский рынок предлагает несколько основных сервисов для поиска аренды:

ЦИАН — один из крупнейших агрегаторов. Обладает широким набором фильтров и аналитикой по регионам. Однако пользователи часто жалуются на дубль-контент, неактуальные объявления и отсутствие функции сравнения.

Авито — мультиресурс, включающий разделы недвижимости. Имеет обширную базу, но интерфейс ориентирован на размещение, а не на анализ и сравнение. Механизм фильтрации простой и часто недостаточный для специфических задач арендатора.

Яндекс.Недвижимость — предоставляет интеграцию с картой и удобную навигацию. Сервис включает статистику и локальную аналитику, но также ограничен в функциональности по сравнению с требованиями профессионального пользователя.

Домофонд — предлагает минимальный функционал фильтрации и визуально приятный интерфейс, но проигрывает в объёме и полноте данных.

Сравнилка — нишевой сервис с упором на сравнение, но без сквозной агрегации с других платформ.

Все платформы частично покрывают потребности арендаторов, но ни одна не предоставляет комплексное решение с интеллектуальной фильтрацией, устранением дублей, нормализованным сравнением и сохранением истории поиска. Графически сравнительные характеристики представлены на рисунке 1.



Рисунок 1 — Таблица сравнения функциональности существующих сервисов

# 1.4 Проблемы и недостатки существующих решений для анализа и сравнения условий аренды недвижимости

На основании анализа текущей ситуации на рынке аренды недвижимости и функционала популярных цифровых платформ можно выделить ряд существенных недостатков, которые напрямую влияют на эффективность поиска и удобство пользователей.

Неактуальность данных остаётся одной из главных проблем. Объявления нередко продолжают отображаться на сайтах даже после того, как жильё уже сдано в аренду. Отсутствие своевременной модерации или автоматического удаления устаревшей информации приводит к тому, что пользователи тратят время на изучение и открытие неактуальных предложений. Это снижает доверие к сервису и замедляет процесс принятия решений.

Повторение информации — ещё одна распространённая трудность. Один и тот же объект может быть размещён на разных площадках (например, одновременно на ЦИАН и Авито), причём с разными условиями аренды, ценой, фотографиями или описанием. Такие дубликаты не только усложняют анализ, но и создают у пользователя ощущение хаоса и несогласованности данных.

Недостаточная фильтрация ограничивает возможности пользователя при выборе жилья. Большинство платформ предоставляют только базовые фильтры, не позволяя комбинировать параметры или учитывать дополнительные нюансы. В результате пользователь получает либо слишком широкий список, либо теряет подходящие варианты из-за ограниченности настроек.

Ручное сравнение объектов значительно затрудняет работу с большим количеством объявлений. На многих сайтах отсутствует функция автоматического сопоставления квартир по ключевым критериям или формирования рейтингов на основе заданных предпочтений. Пользователю приходится самостоятельно открывать и сравнивать каждую карточку, что делает процесс выбора утомительным и затратным по времени.

Отсутствие региональной персонализации означает, что сервисы слабо адаптируются под специфику города или района, в котором находится пользователь. Нередко невозможно указать индивидуальные особенности локального рынка, учитывать инфраструктуру, транспортную доступность или особенности микрорайонов. Это особенно актуально для крупных городов, где район играет ключевую роль при выборе квартиры.

Низкий уровень взаимодействия с системой также сказывается на пользовательском опыте. Во многих сервисах отсутствуют функции сохранения истории поиска, подписки на обновления по интересующим параметрам, механизмов обратной связи или персональных рекомендаций. Всё это делает взаимодействие с платформой одноразовым и негибким.

Перечисленные проблемы становятся основанием для разработки собственного программного решения, которое должно устранить выявленные недостатки и обеспечить более интеллектуальный, адаптируемый и удобный инструмент для анализа и сравнения условий аренды недвижимости.

# 1.5 Описание существующего процесса решения задачи для анализа и сравнения условий аренды недвижимости

В настоящее время большинство пользователей в России сталкиваются с необходимостью самостоятельного поиска и анализа предложений по аренде квартир, опираясь на популярные онлайн-платформы. Среди таких платформ можно выделить ЦИАН, Авито, Яндекс.Недвижимость, Домофонд и Сравнилку. Эти ресурсы предоставляют базовую функциональность: пользователь вручную задаёт фильтры, просматривает выдачу, сравнивает параметры и, при необходимости, обращается к арендодателю через встроенные каналы связи. Несмотря на широкое распространение, данные подходы не автоматизируют процесс сопоставления объявлений и не учитывают индивидуальные предпочтения арендаторов.

Большинство платформ предоставляют лишь поверхностную фильтрацию. Обычно она включает цену, количество комнат, тип жилья, площадь и наличие фотографий. Отсутствует интеллектуальная система сравнения: пользователь сам открывает несколько вкладок и визуально сопоставляет предложения, теряя время и рискуя упустить важные детали.

Ещё одной существенной проблемой является отсутствие единой базы данных. Каждый сервис содержит только собственные объявления, поэтому для полноты картины пользователю приходится использовать несколько сайтов одновременно. Это не только усложняет процесс, но и приводит к дублированию информации — одни и те же квартиры часто размещаются на разных платформах с разными ценами и описаниями. Устранение таких дубликатов возможно лишь вручную, что снижает точность анализа и увеличивает временные затраты.

Отдельно стоит отметить, что отслеживание истории поиска и предпочтений пользователя в существующих сервисах практически не реализовано. Некоторые платформы сохраняют недавние поиски в cookies, но они не формируют персональные рекомендации и не предлагают повторный запуск с заданными параметрами. Кроме того, функции ранжирования по степени соответствия пожеланиям арендатора также отсутствуют: объявления отображаются в произвольном или хронологическом порядке.

Существующий процесс поиска и анализа условий аренды на российском рынке во многом остаётся ручным, фрагментированным и ограниченным в функциональности. Это создаёт высокую нагрузку на пользователя, снижает эффективность принятия решений и делает аренду жилья более стрессовой задачей, чем она могла бы быть при наличии сквозной системы с автоматическим анализом и сравнением предложений.

# 2 Теоретические аспекты разработки веб-сервиса для анализа и сравнения условий аренды недвижимости

# 2.1 Предлагаемый процесс решения задачи анализа и сравнения условий аренды недвижимости

Процесс проектирования веб-сервиса начинается с определения его ключевой задачи — предоставление пользователю релевантных предложений по аренде квартир с учётом множества фильтров и предпочтений. Старт взаимодействия происходит с авторизации пользователя и выбора им интересующего региона. Это важно, поскольку архитектура сервиса предполагает привязку парсинга и хранения данных к географическому признаку, исключая избыточную нагрузку на систему.

Веб-сервис реализует централизованную архитектуру сбора, обработки и отображения информации об аренде жилья, построенную на основе функциональной модели IDEF0 (согласно рисунку 2). Графически она отображена на рисунке. В её основе лежит главная функция (A0) — «Поиск и сравнение предложений по аренде жилья». На вход система получает пользовательский запрос и параметры фильтрации. На выходе формируется список релевантных объектов, дополненный внешними ссылками на оригинальные площадки. Работа системы регулируется рядом управляющих факторов: это контроль корректности применяемых фильтров, лимиты частоты обновлений и доступность источников. Технически реализация опирается на стек из FastAPI (в качестве серверной части), PostgreSQL (для хранения данных), Redis и Celery (обработка фоновых задач), модулей парсинга и клиентского интерфейса на Next.js.



Рисунок 2 — IDEF0-диаграмма предлагаемого процесса анализа и сравнения условий аренды

Первым этапом в работе сервиса является сбор данных (A1). На этом шаге система запускает парсеры, соответствующие выбранному пользователем региону. Чтобы избежать блокировок со стороны внешних сайтов (например, ошибки 403), применяется комплекс защитных механизмов: подключение к пулу прокси-серверов, смена User-Agent, а также программные задержки и повторные попытки получения данных. Все страницы, которые не удалось обработать, автоматически логируются и исключаются из текущей выдачи.

На следующем этапе происходит обработка и фильтрация данных (A2). Система очищает полученные объявления, устраняя дубликаты по ряду критериев: совпадение адреса, номера квартиры, этажа или стоимости с допустимой погрешностью (1–3%). Затем к объявлениям применяется пользовательская фильтрация по параметрам: район, площадь, срок аренды, наличие бытовой техники, возможность проживания с детьми или животными, этаж и другие условия.

После фильтрации сервис переходит к этапу сравнения (A3). Здесь система производит нормализацию параметров объектов и рассчитывает их соответствие предпочтениям пользователя. Каждый объект получает оценку и сортируется в зависимости от степени релевантности. Отображение результатов реализовано в виде карточек, каждая из которых содержит краткое описание, ключевые характеристики и ссылку на исходный источник.

Последний шаг в модели — обновление данных (A4). Источники объявлений опрашиваются регулярно с заданной периодичностью. Этот процесс организован через фоновое выполнение задач с помощью Celery. Если структура сайта-источника изменилась и парсер не может обработать данные, система фиксирует сбой и отправляет уведомление администратору для последующего обновления модуля.

Также сохраняется история запросов пользователя и осуществляется логирование всех этапов. Это позволяет повторно запрашивать похожие фильтры, а также проводить аналитическую обработку поведения пользователей. Обновление базы данных производится автоматически по заданному расписанию или при изменении фильтров.

# 2.2 Формальная модель для описания проблемы анализа и сравнения условий аренды недвижимости

Математическая формализация позволяет описать сущность обработки информации в системе. Пусть A — множество объявлений об аренде, где каждый элемент aᵢ ∈ A описывается вектором параметров V(aᵢ) = (p₁, p₂, ..., pₙ), таких как цена, площадь, количество комнат, этаж, район, техника и прочее.

Пусть Q — множество фильтров пользователя, где qⱼ ∈ Q определяет желаемые значения или диапазоны параметров. Каждое объявление aᵢ удовлетворяет условию попадания в выборку, если V(aᵢ) ∈ Q.

Также определим множество D дубликатов: aᵢ ≈ aⱼ, если совпадает адрес и этаж, или при отсутствии этих данных разница по цене не превышает 3%. Из каждого подмножества D сохраняется только одно объявление с минимальной ценой.

Для сравнения объектов между собой используется следующая функция расстояния:

d(aᵢ, a\*) = √∑ wⱼ × (norm(pⱼ(aᵢ)) - norm(pⱼ(a\*)))²

Где:

a\* — эталонное (идеальное или приоритетное) объявление,

pⱼ — параметр,

wⱼ — вес важности параметра,

norm() — функция нормализации значения в пределах [0,1].

Сортировка итогового множества объявлений производится по возрастанию значения d(aᵢ, a\*), что позволяет вывести в топ наиболее релевантные объекты.

# 2.3 Описание алгоритма для анализа и сравнения условий аренды недвижимости

Предлагаемый алгоритм работы веб-сервиса объединяет как синхронные, так и асинхронные компоненты, что позволяет гибко управлять задачами, реагировать на пользовательские запросы в режиме реального времени и параллельно выполнять ресурсоёмкие операции. Его структура включает восемь логически связанных этапов, каждый из которых играет важную роль в обеспечении полноты и точности выдачи результатов.

Первым шагом алгоритма является авторизация пользователя и выбор региона. Этот этап необходим для того, чтобы привязать последующую фильтрацию и сбор данных к конкретной географической зоне. После входа пользователь выбирает интересующий его регион, и вся дальнейшая работа системы — от парсинга до отображения результатов — ориентирована именно на него. Это повышает точность выдачи и снижает объём ненужных данных.

На втором этапе происходит инициализация процесса парсинга. Система отправляет задачу в очередь фоновой обработки Celery, где запускаются парсеры, соответствующие выбранному региону. При этом реализованы механизмы защиты от блокировок со стороны сайтов-источников: используется прокси-сервер, происходит регулярная смена User-Agent, а также задаются паузы между запросами для имитации поведения реального пользователя. Это позволяет собирать данные, избегая санкций и ограничений.

После получения данных начинается этап их анализа и валидации. Система проверяет корректность структуры и содержимого каждого объявления. Удаляются записи с явными ошибками, пустыми значениями или нарушениями формата. Все данные проходят процесс нормализации, в ходе которого они приводятся к единому стандарту — это необходимо для дальнейшей фильтрации и сравнения.

Далее осуществляется применение фильтрации. Пользовательские параметры (например, цена, количество комнат, наличие техники) транслируются в SQL-запрос или обрабатываются с использованием ORM. На этом этапе из базы данных извлекаются только те объекты, которые соответствуют заданным критериям. Такой подход позволяет максимально точно удовлетворить пользовательский запрос без лишней информации.

Пятым этапом в работе системы становится удаление повторов. Поскольку одно и то же объявление может встречаться на разных сайтах, сервис использует алгоритм, который сравнивает ключевые характеристики: адрес, этаж и цену. Когда находятся дубли, система оставляет только один, самый полный и выгодный вариант. Это позволяет сделать список предложений чище и точнее.

Дальше идёт сравнение всех собранных вариантов между собой. Каждое предложение оценивается на предмет того, насколько оно подходит под предпочтения пользователя. Для этого система приводит все параметры к единой форме и высчитывает разницу с «идеальным» вариантом жилья. После этого сервис сортирует и расставляет все квартиры по степени соответствия, чтобы в начале списка были самые подходящие.

Финальный этап — создание карточек для отображения результатов. Для каждого найденного объекта формируется карточка с ключевыми характеристиками: метраж, этаж, цена и район. В ней же есть фотографии, значки с дополнительной информацией и прямая ссылка, которая ведёт на сайт, где размещено объявление. Это позволяет пользователю сразу перейти к подробной информации и связаться с арендодателем. Пользователь получает на экране аккуратно оформленный список предложений, с которыми может быстро ознакомиться и принять решение.

Завершающий этап — сохранение истории поиска. Каждый выполненный запрос вместе с фильтрами и результатами добавляется в таблицу search\_history. Это позволяет пользователю в будущем вернуться к ранее найденным вариантам или повторить тот же самый поиск без необходимости повторной настройки параметров.

Таким образом, представленный алгоритм обеспечивает не только стабильную и точную обработку информации, но и высокую степень повторяемости результата. Благодаря модульной структуре и использованию асинхронных технологий его можно масштабировать, адаптировать под новые регионы и гибко подстраивать под изменения внешних источников.

# 2.4 Разработка функциональных требований к веб-сервису для анализа и сравнения условий аренды недвижимости

Функциональные требования формируют основу проектирования веб-сервиса. Именно они определяют архитектурную логику, структуру пользовательского интерфейса, взаимодействие компонентов и особенности хранения и обработки данных. Представленные ниже требования были выработаны на основе анализа существующих решений, выявленных проблем и задач, поставленных в рамках проекта.

Первым и базовым требованием является наличие механизма регистрации, авторизации и выбора региона. Пользователь должен иметь возможность создать личный профиль, войти в систему и указать интересующий его географический регион. Это обеспечивает индивидуализацию работы сервиса и запускает все последующие этапы: парсинг, фильтрацию и отображение данных — строго в пределах выбранной территории.

Неотъемлемой частью архитектуры является интеграция с внешними платформами аренды недвижимости. Для этого реализована система парсеров, каждый из которых подключается к конкретному источнику (например, ЦИАН, Авито, Яндекс.Недвижимость) и извлекает актуальные объявления. Интеграция осуществляется с учётом различий в структуре данных, форматах и механизмах обновления информации на каждой из платформ.

Поскольку внешние источники часто защищены от автоматизированного сбора данных, сервис должен обеспечивать защиту от блокировок. Для этого реализованы антибан-механизмы: использование пулов прокси, ротация User-Agent, задание пауз между запросами. В случае ошибки 403 или временной недоступности сайта система выполняет автооткат задачи и логирует проблему без прерывания пользовательского потока.

Следующее требование — это обязательная очистка дубликатов. Одни и те же квартиры часто публикуются на разных платформах. Сервис автоматически сравнивает объявления по адресу, этажу, номеру квартиры и стоимости с допустимой погрешностью, и удаляет повторяющиеся записи. В итоговую выдачу попадает только одно — наиболее выгодное и полное по содержанию.

Для качественной фильтрации реализована система нормализации параметров. Все значения в объявлениях приводятся к единому виду, что позволяет корректно применять фильтры. Поддерживаются фильтры по цене (от и до), сроку аренды (1, 6 или 12 месяцев), району и улице, этажу, площади, количеству комнат, наличию мебели и техники, типу дома, а также условиям проживания — с детьми и животными. Такой подход обеспечивает максимально точное соответствие требованиям пользователя.

Важным функциональным элементом является визуальный интерфейс. Он реализован с помощью Next.js и предоставляет пользователю быстрый отклик при работе с фильтрами, а также мгновенную генерацию карточек с объявлениями. Каждая карточка отображает ключевые параметры квартиры, фото, иконки условий и внешнюю ссылку на источник.

Каждое объявление снабжается активной внешней ссылкой на оригинальный сайт. Это позволяет пользователю перейти напрямую к исходной платформе, где он может связаться с владельцем жилья, получить дополнительные сведения или оставить заявку. Такая реализация повышает доверие к сервису и сохраняет юридическую корректность работы.

Сервис должен сохранять историю запросов и фильтров, что особенно важно для пользователей, ведущих долгосрочный поиск. История сохраняется в базе данных и может быть повторно использована — пользователь может вернуться к уже найденным вариантам или быстро повторить фильтрацию по заданным критериям.

Интерфейс проектировался с учётом 10 эвристик Нильсена. В частности, реализована визуальная обратная связь на действия пользователя, возможность отмены, понятные сообщения об ошибках, последовательная структура интерфейса, единообразие элементов и минимализм в подаче информации.

Для обеспечения актуальности данных реализовано автоматическое обновление базы. Система периодически инициирует обновление информации по расписанию, запуская парсеры и актуализируя карточки в базе. Это позволяет пользователю получать только свежие и доступные варианты жилья. Графически система взаимодействия классов отражена на рисунке 3.

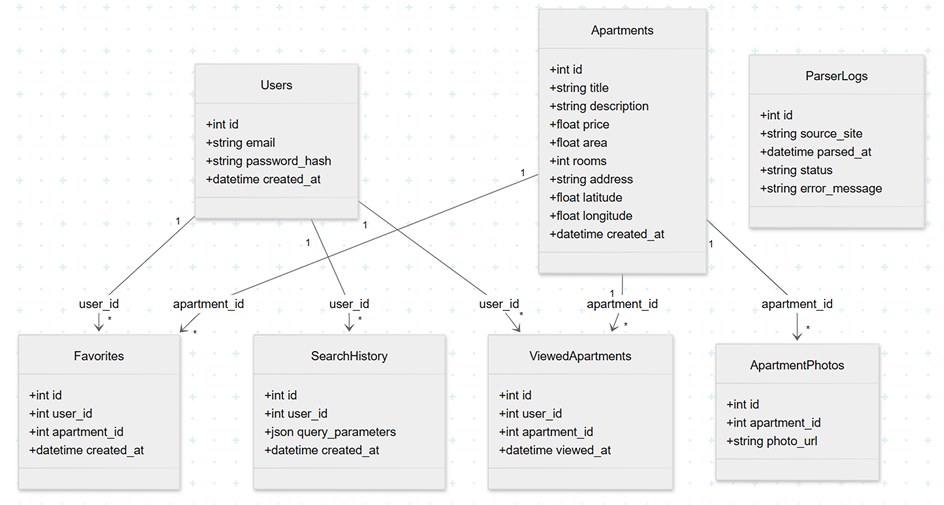


Рисунок 3 — Диаграмма классов веб-сервиса

Фоновая обработка задач организована через очередь Celery с брокером сообщений Redis. Благодаря этому пользовательские действия (например, запуск фильтрации) не блокируют тяжёлые операции (например, парсинг или дедупликация), и система остаётся отзывчивой даже при высокой нагрузке.

В совокупности все эти требования образуют целостную архитектурную модель, ориентированную на удобство пользователя, надёжность системы и полноту информации. Их реализация направлена на решение конкретных проблем, выявленных в существующих сервисах, и полностью согласуется с целями, поставленными в рамках дипломного проекта.

# Выводы

Во второй главе были рассмотрены теоретические и прикладные аспекты, лежащие в основе проектирования веб-сервиса для анализа и сравнения условий аренды недвижимости. Основное внимание было уделено процессу организации логики взаимодействия компонентов, методам обработки информации, математическому моделированию и постановке функциональных требований к системе.

Предлагаемый процесс решения задачи охватывает полный цикл — от авторизации пользователя и выбора региона до получения упорядоченного списка подходящих объявлений. Архитектура сервиса предполагает тесную связь между выбором региона и последующей логикой сбора и фильтрации данных, что позволяет уменьшить объём нерелевантной информации и снизить нагрузку на систему. Использование очередей задач (Celery) и фоновая обработка обеспечивают асинхронность и стабильность даже при интенсивной активности.

Математическая модель, представленная в главе, позволила формализовать ключевые элементы системы: вектор параметров объявления, систему фильтров, критерии дедупликации и функцию расстояния для оценки релевантности. Это даёт возможность объективно сравнивать объекты между собой и формировать ранжированный список предложений, соответствующих пользовательским предпочтениям. Такой подход устраняет необходимость ручного сопоставления объявлений и минимизирует ошибки выбора.

Особое внимание было уделено описанию алгоритма анализа, включающего восемь последовательных этапов: от авторизации до сохранения истории поиска. Каждый из этапов выполняет свою уникальную роль: сбор данных с внешних платформ, нормализация, фильтрация, устранение дублей, сравнение с эталонной моделью, визуализация в виде карточек и логирование действий пользователя. Такой комплексный подход обеспечивает точность, повторяемость и расширяемость решения.

Также в главе был представлен перечень функциональных требований к системе, сформированных на основе анализа недостатков существующих платформ. В них входят: механизм регистрации и выбора региона, гибкая фильтрация по множеству параметров, интеллектуальное сравнение объектов, переход к оригинальным источникам, сохранение истории поиска и автоматическое обновление данных. Все эти требования напрямую вытекают из реальных задач арендаторов и обеспечивают улучшенный пользовательский опыт.

Выявленные теоретические основы и предложенные решения легли в основу практической реализации веб-сервиса, описанной в следующих главах. Проанализированная архитектура и математическая модель подтвердили свою применимость к задачам фильтрации, сравнения и визуализации объектов аренды. Глава 2 заложила концептуальный фундамент, необходимый для эффективного и масштабируемого программного решения в сфере аренды недвижимости.

# 3 Проектирование и разработка веб-сервиса для анализа и сравнения условий аренды недвижимости

# 3.1 Выбор технологий и инструментов для разработки веб-сервиса анализа и сравнения условий аренды недвижимости

Выбор стека технологий и инструментов для разработки веб-сервиса основывался на требованиях к надёжности, масштабируемости, высокой производительности и удобству поддержки. Также учитывались аспекты асинхронной обработки, устойчивости к сбоям, возможности автоматического сбора информации из внешних источников и обеспечения отзывчивого пользовательского интерфейса. В итоге архитектура решения была построена по принципу разделения ответственности между слоями: клиентским (frontend), серверным (backend), уровнем хранения данных и фоновыми задачами. Графически архитектура решения представлена на рисунке 4.

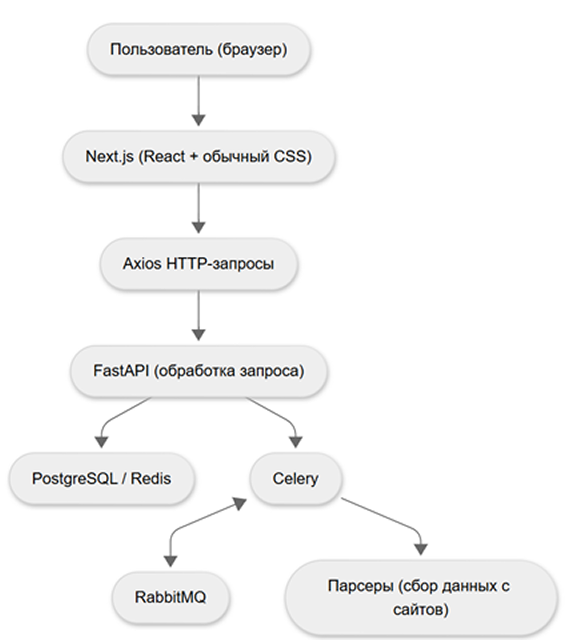


Рисунок 4 — Архитектурная схема решения

# 3.1.1 Клиентская часть (Frontend)

Для реализации пользовательского интерфейса был выбран Next.js — современный фреймворк на базе React, поддерживающий серверный рендеринг и статическую генерацию страниц (рисунок 5 — главная страница веб-сервиса).

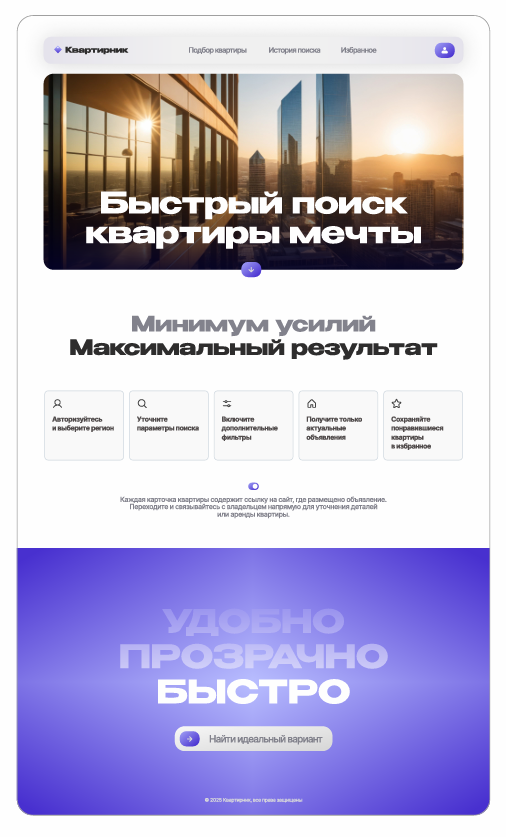


Рисунок 5 — главная страница веб-сервиса для анализа и сравнения условий аренды квартир

Одним из ключевых преимуществ выбранного фреймворка является его высокая производительность и способность к SEO-оптимизации. Благодаря серверному рендерингу и статической генерации страниц, веб-сервис работает быстро, а поисковые системы корректно индексируют контент, что особенно важно для дальнейшего продвижения и расширения проекта.

Также существенное преимущество — это быстрая навигация между страницами за счёт встроенной маршрутизации и частичной подгрузки данных. Пользователь не замечает задержек при переходах между разделами, поскольку элементы интерфейса перерисовываются динамически, без полной перезагрузки.

Фреймворк предоставляет возможность гибкой интеграции с внешними API с помощью встроенных серверных функций getServerSideProps и getStaticProps. Это позволяет получать, обрабатывать и отображать данные в реальном времени или предварительно — в зависимости от требований конкретной страницы или сценария использования.

Разработка интерфейса осуществляется с применением современных подходов к стилизации. Используются как чистый CSS, так и CSS-модули, а также TailwindCSS — утилитарный CSS-фреймворк, ускоряющий верстку и обеспечивающий единообразие визуального оформления. Такой подход позволяет быстро настраивать и поддерживать дизайн, не теряя гибкости при его расширении.

Фреймворк допускает развёртывание как на классическом сервере, так и в serverless-среде. Это открывает широкие возможности для масштабирования проекта и оптимизации затрат на инфраструктуру при выходе за рамки MVP-версии.

Разработанный интерфейс спроектирован в соответствии с 10 эвристиками юзабилити Якоба Нильсена. В системе реализованы принципы обратной связи, логичности взаимодействия, минимализма, узнаваемости элементов и визуального комфорта. Пользователь управляет фильтрами, просматривает карточки объектов, быстро получает доступ к релевантным данным — все действия интуитивны и не требуют обучения. Графически интуитивно-понятный интерфейс регистрации пользователя проиллюстрирован рисунком 6.

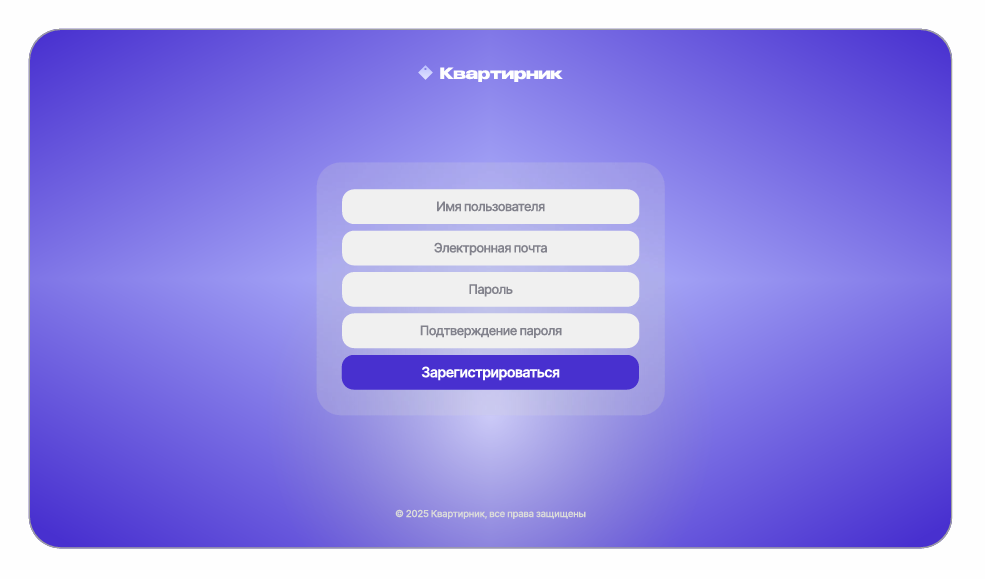


Рисунок 6 — окно регистрации пользователя веб-сервиса для анализа и сравнения условий аренды квартир

# 3.1.2 Связь клиент–сервер

Взаимодействие между клиентской частью веб-сервиса (frontend) и серверной логикой (backend) реализовано с использованием технологии асинхронного обмена данными по протоколу HTTP. Основным инструментом для отправки запросов с клиентской стороны выбран Axios — современный и широко используемый JavaScript-библиотека-клиент, обеспечивающая простое и надёжное выполнение HTTP-запросов из браузера.

Axios интегрирован в архитектуру приложения на базе Next.js и используется при каждом обращении пользователя к данным, хранящимся или обрабатываемым на сервере. Благодаря своей лёгкости и поддержке промисов, он позволяет удобно управлять запросами, обрабатывать статусы ответов, ловить ошибки и управлять логикой интерфейса в зависимости от результата обмена с сервером.

В качестве формата передачи данных между frontend и backend используется JSON (JavaScript Object Notation) — компактный текстовый формат, удобный для сериализации данных и совместимый с большинством языков программирования. Он обеспечивает лёгкость интерпретации как на клиентской стороне, так и на сервере, упрощая валидацию, логирование и отладку.

Каждое пользовательское действие, связанное с обработкой данных — будь то применение фильтров, обновление параметров поиска, просмотр карточки объекта или добавление в избранное — инициирует отправку HTTP-запроса. Этот запрос формируется на стороне клиента и направляется в соответствующий endpoint REST API, реализованный с помощью FastAPI.

На серверной стороне FastAPI принимает запрос, извлекает параметры, проводит валидацию и выполняет соответствующую бизнес-логику: обращается к базе данных, запускает фоновую задачу, или возвращает готовую информацию. Ответ формируется в формате JSON и возвращается обратно на frontend, где он используется для обновления пользовательского интерфейса без перезагрузки страницы.

Такой подход обеспечивает высокую скорость работы сервиса, устойчивость к ошибкам и возможность масштабируемого расширения функциональности без изменения основного протокола взаимодействия.

# 3.1.3 Серверная часть (Backend)

Серверная часть веб-сервиса реализована с использованием FastAPI — современного высокопроизводительного Python-фреймворка, предназначенного для создания REST API. Одним из главных преимуществ FastAPI является его нативная поддержка асинхронного программирования, что позволяет эффективно обрабатывать большое количество параллельных запросов. Это особенно важно для задач, связанных с фильтрацией, обработкой фоновых задач и взаимодействием с внешними источниками данных.

FastAPI поддерживает работу с асинхронным вводом-выводом (async/await), что даёт возможность не блокировать поток выполнения при длительных операциях, таких как обращение к базе данных или ожидание ответа от парсера. За счёт этого достигается высокая отзывчивость системы и минимальное время отклика даже при интенсивной нагрузке.

Фреймворк построен на базе Starlette — лёгкой и быстрой асинхронной библиотеки для веб-серверов, и использует Uvicorn в качестве ASGI-сервера. Эта связка обеспечивает одну из лучших производительных реализаций среди современных Python-инструментов для работы с веб-приложениями. Она позволяет запускать приложение в продакшн-среде с минимальными издержками по ресурсам и при этом обеспечивает стабильную работу под нагрузкой.

Важным достоинством FastAPI является строгая типизация на основе библиотеки Pydantic. Типы параметров запроса, тела запроса и возвращаемого ответа указываются явно, что повышает надёжность и снижает количество ошибок. Кроме того, автоматически формируется документация OpenAPI, доступная в браузере, что удобно для тестирования, интеграции и поддержки API.

Благодаря своей масштабируемости и читаемости кода, FastAPI позволяет организовать проект в виде модульной архитектуры, где каждый компонент (авторизация, фильтрация, парсеры, обработка истории) реализован независимо. Это упрощает как начальную разработку, так и последующее расширение функциональности, добавление новых модулей и изменение логики работы сервиса без риска сломать уже существующий функционал.

В рамках данного проекта FastAPI обрабатывает весь поток пользовательских запросов. Через него реализована авторизация пользователей, применение фильтров к базе данных, вызов фоновых задач через Celery, сохранение истории поиска, а также формирование JSON-ответов, которые возвращаются клиенту. Благодаря чёткому разделению задач и удобной структуре API, вся логика backend остаётся предсказуемой, масштабируемой и удобной для сопровождения.

# 3.1.4. Хранение данных

Для хранения структурированных данных в веб-сервисе используется PostgreSQL — одна из самых надёжных и широко применяемых реляционных систем управления базами данных. PostgreSQL обладает высоким уровнем стабильности, поддерживает ACID-транзакции и обеспечивает целостность данных даже при высоких нагрузках. Это делает её оптимальным выбором для хранения пользовательских сессий, объектов аренды, истории запросов и других критичных сущностей системы. Структура базы данных, разработанная для сервиса, представлена графически на рисунке 7.

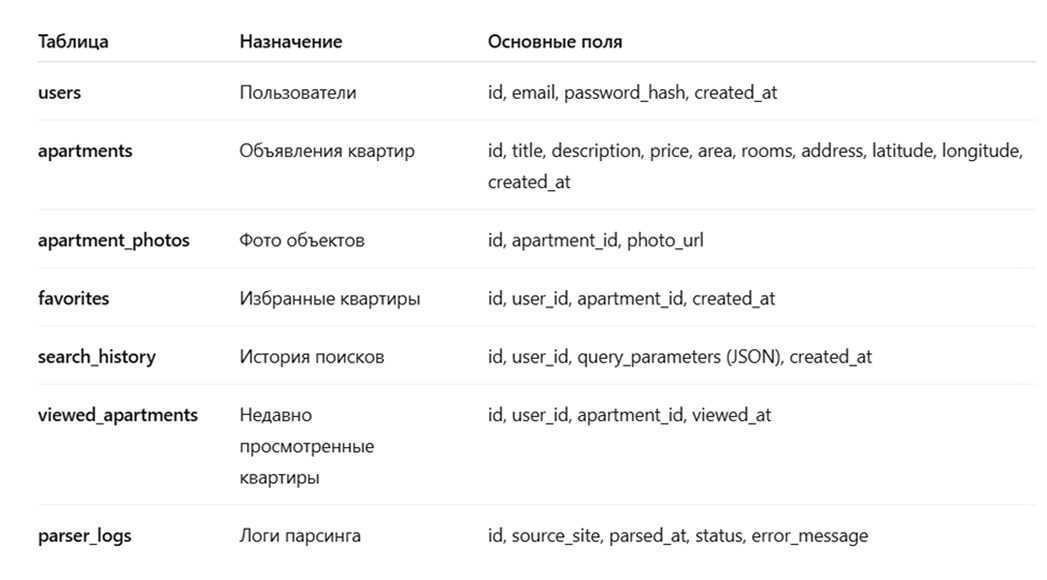


Рисунок 7 — Расширенная архитектура обработки запроса и сбора данных

Одним из важных преимуществ PostgreSQL является поддержка индексов, включая составные, частичные и полнотекстовые, что позволяет ускорить выполнение запросов, особенно при многокритериальной фильтрации и сортировке. В условиях работы с большим объёмом объявлений, где требуется быстро отбирать данные по цене, площади, району и другим параметрам, наличие продуманной индексной структуры критически важно.

Дополнительно PostgreSQL предоставляет расширенные возможности работы с JSON-форматом, что позволяет хранить сложные или неструктурированные данные, получаемые от парсеров. Такие данные могут включать динамически меняющиеся параметры объявлений или вложенные объекты, что упрощает их последующую обработку без необходимости жёсткой схемы.

Помимо основной СУБД в системе применяется Redis — лёгкое и высокопроизводительное хранилище данных в формате "ключ–значение". Оно выполняет две ключевые функции. Первая — это кеширование часто запрашиваемой информации, что позволяет значительно снизить нагрузку на PostgreSQL и ускорить отклик при повторных запросах пользователя. Например, если пользователь уже один раз провёл поиск, его результаты могут быть временно сохранены в Redis. Это позволяет в следующий раз возвращать эти же данные без обращения к основной базе, что экономит время и ресурсы.

Кроме того, Redis в этом проекте выполняет ещё одну важную функцию — он становится брокером для фоновых задач, которые обрабатываются через Celery. Благодаря этому можно выносить на отдельные потоки такие ресурсоёмкие процессы, как парсинг данных, обновление базы или фиксация истории поиска. Redis принимает эти задачи, распределяет их между исполнителями и следит, чтобы каждая была выполнена правильно и в нужной последовательности.

Вместе использование PostgreSQL и Redis позволяет одновременно сохранить надёжность всех данных и обеспечить высокую скорость работы — даже при активной нагрузке. Такая архитектура позволяет гибко масштабировать систему, обеспечивать стабильную работу даже при увеличении числа пользователей и объёма данных, а также легко внедрять дополнительные функции без необходимости полной перестройки инфраструктуры.

# 3.1.5 Очереди задач и обработка парсинга

Веб-сервис активно использует асинхронную и фоновую обработку данных для выполнения ресурсоёмких операций, таких как парсинг, обновление базы и повторные попытки запросов к внешним источникам. Это позволяет сохранить высокую отзывчивость интерфейса даже при больших объёмах данных или одновременном выполнении нескольких задач. Основу фоновой обработки составляет надёжная связка компонентов: Celery, RabbitMQ и Redis.

Основной компонент этой архитектуры — Celery. Это известный инструмент для управления фоновыми задачами в Python. С его помощью можно запускать задачи отдельно от основного процесса, ставить их в очередь и отслеживать, как они выполняются. Celery умеет планировать задачи, автоматически повторять их в случае ошибок, а при необходимости — распределять нагрузку между несколькими воркерами, которые могут работать одновременно.

Для того чтобы задачи правильно передавались и распределялись, в системе используется RabbitMQ — это брокер сообщений. Он берёт на себя роль посредника между FastAPI и Celery, создавая надёжную очередь. RabbitMQ следит, чтобы все задания попадали туда, куда нужно, выполнялись в правильном порядке и не терялись. Это особенно важно при работе с парсерами и обновлении данных, когда важна чёткая последовательность действий.

Есть ещё один важный элемент — Redis. В этой архитектуре он служит временным хранилищем и кешем, что помогает ускорить работу и сделать систему более отзывчивой. В контексте фоновой обработки Redis используется для хранения состояния задач, промежуточных результатов и служит в качестве одного из вариантов брокера, если RabbitMQ временно недоступен. Кроме того, Redis участвует в ускорении откликов сервиса, временно сохраняя результаты запросов.

Когда пользователь инициирует поиск или обновление информации в конкретном регионе, сервер формирует задачу, которая помещается в очередь Celery. Это может быть либо автоматический запуск по расписанию, либо явное действие пользователя через интерфейс. Сразу после постановки задачи в очередь основной поток освобождается, что позволяет интерфейсу оставаться быстрым и отзывчивым.

Одна из ключевых задач — это вызов парсера, предназначенного для сбора актуальных объявлений с внешних сайтов, таких как Авито, ЦИАН и Яндекс.Недвижимость. Парсер подключается к источнику, обходит защиту от автоматизации и извлекает нужные поля: адрес, цена, площадь, этаж и т.д.

Следующий шаг — обработка и нормализация собранных данных. Все объявления приводятся к единому формату: очищаются от лишней информации, проверяются на полноту, и если данные некорректны или неполные — они отбрасываются или логируются как исключения.

Затем система выполняет проверку на дублирование. Объявления сравниваются между собой по адресу, этажу и цене. В случае совпадений сохраняется только одно объявление — наиболее выгодное или полное, что позволяет избежать перегруженности интерфейса повторяющимися объектами.

После завершения этих этапов осуществляется обновление базы данных. Актуальные записи сохраняются в PostgreSQL, при этом изменения фиксируются в истории и могут быть использованы для последующего анализа. Старые или устаревшие записи удаляются или помечаются как неактуальные.

Такой подход к архитектуре (согласно рисунку 8) позволяет существенно разгрузить основной сервер и перенести на фон все операции, не требующие немедленного отклика. Пользователь при этом получает мгновенную реакцию интерфейса и актуальные данные, а система может гибко масштабироваться, выдерживая высокую нагрузку и сложные вычисления в фоне.

# 3.1.6 Парсеры

Парсеры в структуре веб-сервиса представляют собой автономные модули, написанные на языке Python, которые выполняют задачу извлечения информации с внешних платформ по аренде недвижимости. Каждый парсер разрабатывается индивидуально под конкретный источник — такие как ЦИАН, Авито, Яндекс.Недвижимость и другие. Это необходимо потому, что структура HTML-страниц, способ представления данных и система защиты от автоматизации у каждого сайта различаются. Модули парсинга интегрированы с системой фоновой обработки Celery и запускаются в фоновом режиме в ответ на пользовательский запрос или по расписанию.

Основой работы парсера является отправка HTTP-запросов к целевому сайту. Запросы формируются с учётом особенностей структуры URL и могут включать параметры фильтрации, такие как регион, тип жилья или диапазон цен. После получения HTML-кода страницы парсер использует библиотеки BeautifulSoup, lxml или аналогичные для его синтаксического разбора. Извлекаются ключевые данные — адрес, цена, площадь, этаж, фотографии, условия аренды, наличие мебели, тип здания и прочие характеристики. Алгоритм работы графически изображён на рисунке 8.

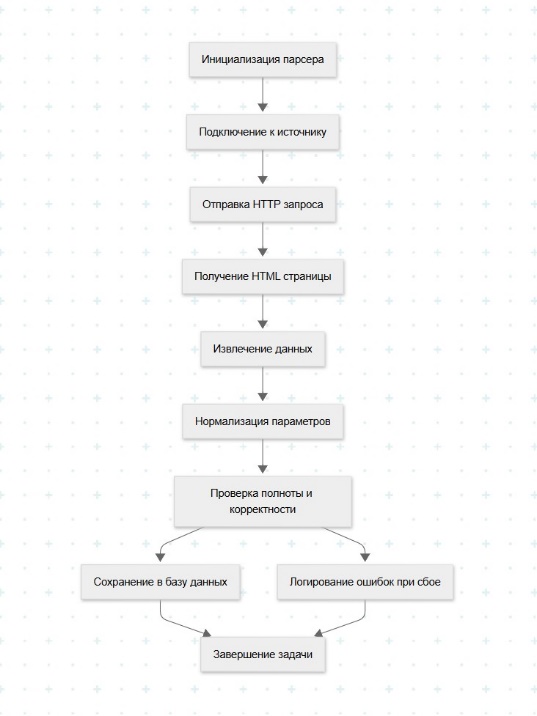


Рисунок 8 — Алгоритм работы парсера данных с внешних источников

Так как большинство платформ защищены от автоматизированного доступа, в архитектуру парсера включены механизмы обхода защит. Один из таких механизмов — ротация IP-адресов с использованием прокси-серверов. Для каждого запроса выбирается случайный IP из заранее подготовленного пула. Это позволяет избежать блокировки со стороны сайтов, отслеживающих активность по IP-адресу.

Другим элементом защиты от блокировок является динамическая подмена заголовков HTTP-запроса, в частности поля User-Agent. Парсер имитирует обращения с разных устройств и браузеров, создавая иллюзию обычного поведения пользователя. Кроме того, могут подставляться дополнительные заголовки (Referer, Accept-Language, Connection), что помогает избежать подозрений со стороны системы антиботов.

Ещё один важный механизм — вставка задержек между запросами. В отличие от сплошного сканирования, парсер делает паузы между запросами к сайту, повторяя типичную частоту действий реального пользователя. Длительность паузы может задаваться случайным образом в пределах допустимого диапазона, чтобы ещё больше снизить вероятность распознавания робота.

Если в процессе работы парсер сталкивается с ошибками — например, ответом сервера с кодом 403 (доступ запрещён), 404 (страница не найдена) или 500 (внутренняя ошибка сервера) — предусмотрена система повторного выполнения задач. Такие ошибки не прерывают выполнение парсинга в целом: задача автоматически перезапускается с другим прокси и другим набором заголовков. Если попытки остаются неудачными, объявление исключается из выборки.

Все возникающие ошибки и исключения в процессе парсинга логируются в отдельной таблице базы данных parser\_logs. В лог сохраняется информация о времени возникновения ошибки, типе ошибки, URL, на котором она произошла, и шаге обработки. Эти данные необходимы для диагностики, доработки парсеров и отслеживания стабильности источников. На основании этих логов администратор может оперативно принять решение о доработке модуля или временной деактивации парсера для конкретной платформы.

Система парсинга спроектирована так, чтобы быть одновременно гибкой, устойчивой и масштабируемой. Добавление нового источника требует лишь создания нового парсер-модуля с учётом структуры сайта и подключения его к общей очереди задач. Это делает систему легко расширяемой и позволяет поддерживать высокое качество данных даже при изменении условий на внешних платформах.

# 3.2 Проектирование интерфейса пользователя веб-сервиса для анализа и сравнения условий аренды недвижимости

Интерфейс веб-сервиса разрабатывался с прицелом на удобство для пользователей. Его задача — не просто отображать данные, а помогать людям быстро и без лишних сложностей находить подходящие квартиры. Все элементы интерфейса продуманы так, чтобы не требовать дополнительного обучения или долгих инструкций.

Особое внимание уделялось скорости работы. Интерфейс создан так, чтобы пользователь мог сразу выбрать нужные параметры и получить подходящие результаты всего за несколько кликов. Все основные элементы управления находятся в доступной зоне экрана, а обновление результатов происходит мгновенно, без перезагрузки страницы. Это делает процесс поиска комфортным и экономит время.

Особое внимание было уделено простоте и логике фильтрации. Пользователь может задать такие параметры, как цена, район, срок аренды, наличие мебели и техники, этаж и другие условия через понятные элементы управления: выпадающие списки, чекбоксы и кнопки выбора. Эти элементы не перегружают интерфейс и логически сгруппированы по смысловым блокам, что упрощает навигацию. Графически система фильтрации представлена на рисунке 9.

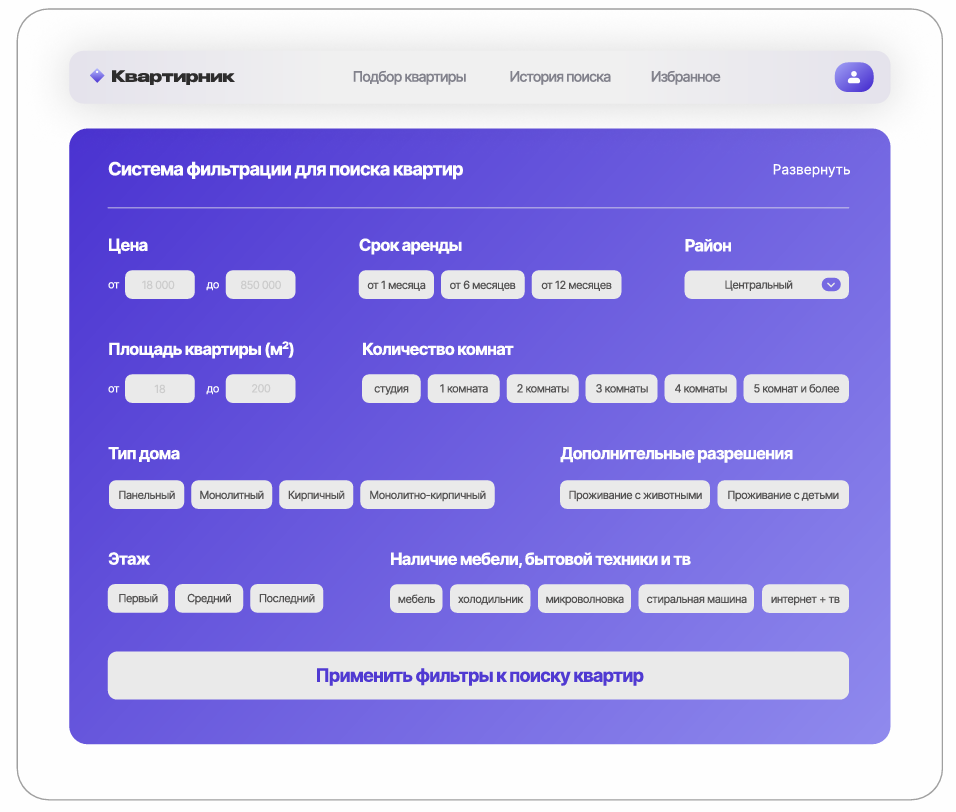


Рисунок 9 — система фильтрации веб-сервиса для анализа и сравнения условий аренды квартир

Визуальная наглядность — ещё один ключевой критерий. Каждое объявление представлено в виде карточки с фотографией, заголовком, краткими параметрами (площадь, этаж, цена, район) и визуальными иконками, отражающими условия проживания (в соответствии с рисунком 10). Это позволяет пользователю быстро «сканировать» взглядом список и выделять интересные варианты без необходимости в долгом чтении.

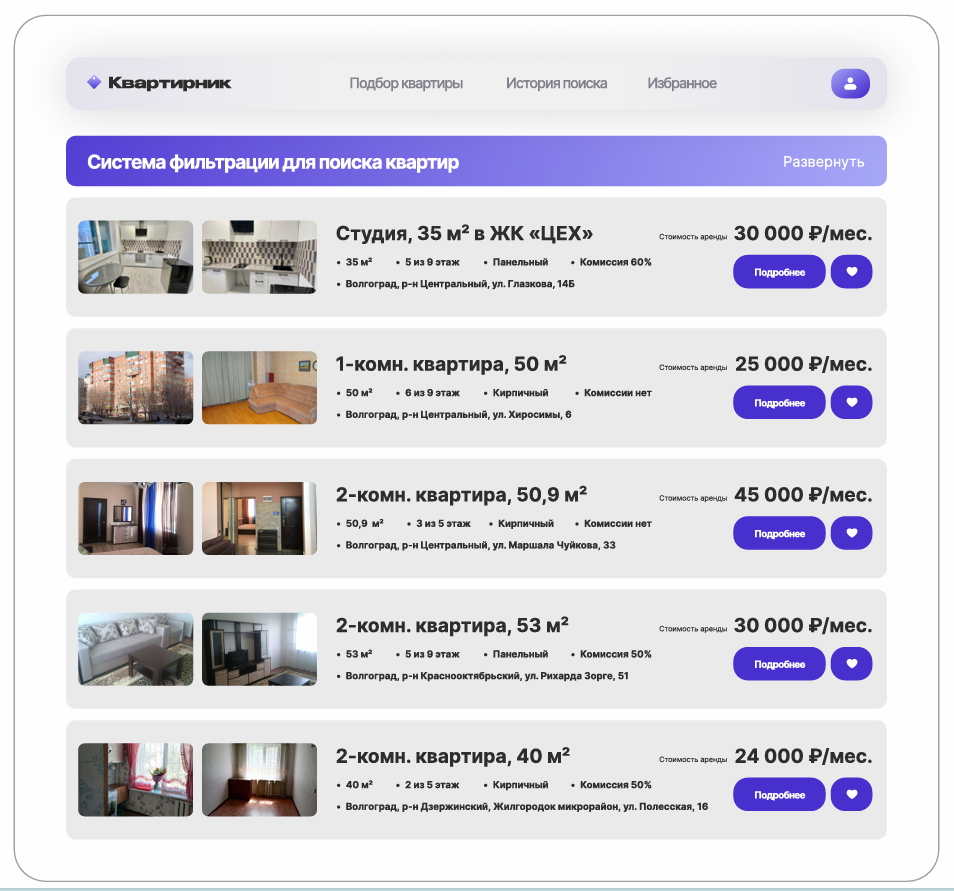


Рисунок 10 — лента объявлений веб-сервиса для анализа и сравнения условий аренды квартир

Интерфейс был спроектирован с учётом 10 эвристик юзабилити Нильсена, что обеспечило высокую эргономику и предсказуемость поведения системы. Среди этих эвристик реализованы такие принципы, как видимость состояния системы (например, индикация загрузки и активности фильтров), контроль и свобода действий (возможность сбросить фильтры или изменить регион), а также понятная и логичная структура взаимодействия (от фильтра до карточки объекта) (согласно рисунку 11).

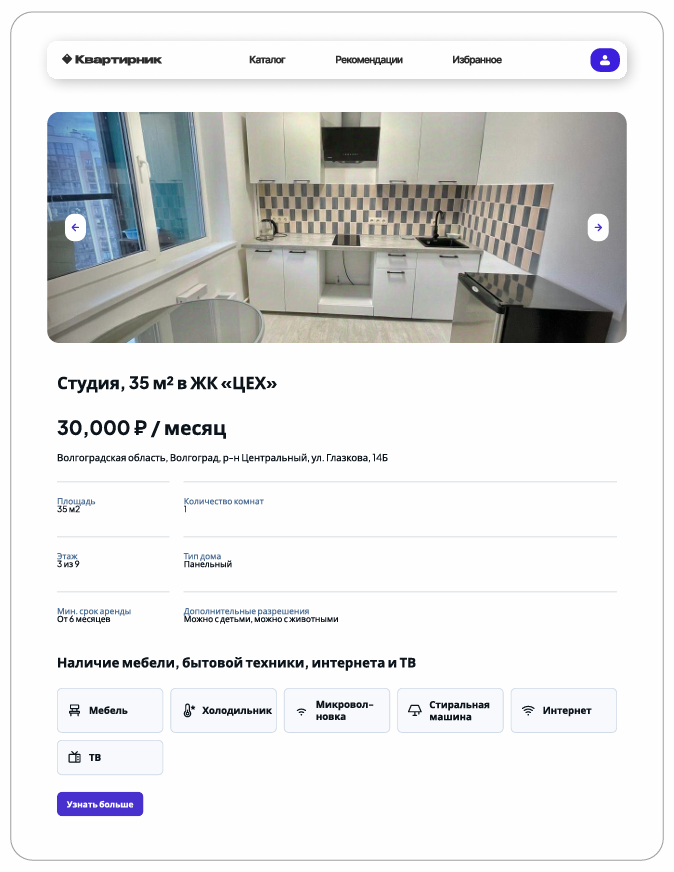


Рисунок 11 — страница карточки объекта на веб-сервисе для анализа и сравнения условий аренды квартир

Также применялась стратегия визуального минимализма, позволяющая избежать перегрузки экрана и отвлекающих элементов. Оттенки, типографика, структура блоков — всё выдержано в едином стиле. Отсутствие рекламных блоков, всплывающих окон и лишней информации помогает сосредоточиться на главном: выборе жилья. Графически фокус на функциональном минимализме проиллюстрирован рисунком 12.

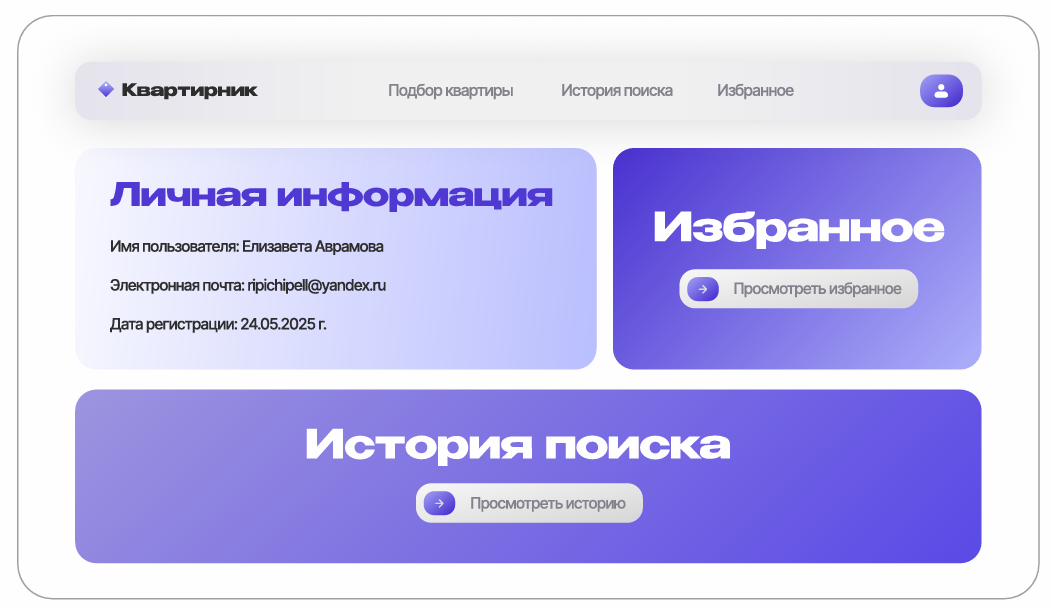


Рисунок 12 — главная страница веб-сервиса для анализа и сравнения условий аренды квартир

В результате интерфейс не только функционален, но и воспринимается как понятный, быстрый и надёжный инструмент, соответствующий пользовательским ожиданиям. Он не требует обучения и позволяет сразу приступить к работе. Это особенно важно для арендаторов, у которых ограничено время и высокая мотивация найти подходящий вариант как можно быстрее.

3.2.1 Главная структура и навигация

В верхней части сайта располагается навигационная панель, которая выполняет роль основного инструмента перемещения по ключевым разделам веб-сервиса. Её структура продумана таким образом, чтобы пользователь мог в любой момент быстро перейти к интересующему его разделу, не теряя контекст текущего поиска. Панель остаётся закреплённой при прокрутке страницы, что обеспечивает постоянный доступ к навигации.

Первым и основным разделом является «Каталог», представляющий собой список всех доступных объявлений о сдаче квартир в аренду. При переходе в этот раздел пользователь получает возможность применять фильтры, просматривать карточки объектов и сортировать результаты. Каталог — это центральный компонент, к которому пользователь возвращается в ходе большинства сценариев работы с системой.

Раздел «Рекомендации» сейчас ещё находится на этапе планирования и разработки. Его основная цель — предлагать пользователю персонализированные варианты, которые будут подбираться на основе его предыдущих действий: истории фильтрации и сохранённых предпочтений. Это должно упростить и ускорить поиск, сократив количество шагов, которые нужны, чтобы найти подходящее жильё. Хотя в текущей версии сервиса этот модуль ещё не активен, его место в интерфейсе уже предусмотрено. Это помогает сохранить целостность системы и в будущем легко внедрить новые возможности, не ломая существующую логику.

Раздел «Избранное» создан для того, чтобы сохранять те квартиры, которые пользователь отметил как подходящие. Эта функция особенно удобна, если нужно сравнить несколько вариантов или вернуться к ним позже. Система не только запоминает все параметры каждого выбранного объекта, но и даёт возможность сразу перейти к оригинальному источнику — это упрощает принятие решения, когда пользователь готов сделать свой выбор.

В правой части панели навигации находится значок профиля, который открывает доступ к авторизации и настройкам. После входа пользователь может обновить свои данные или сменить регион, от чего напрямую зависит, какие данные будет собирать сервис. Регион — это ключевой параметр, ведь именно от него зависит, какие квартиры и объявления будут предложены пользователю.

На главной странице предусмотрен приветственный блок с кнопкой «Поиск квартиры», выполненной в виде крупного и заметного баннера. Он сразу привлекает внимание и побуждает начать поиск — буквально в несколько кликов. Он символизирует основную цель веб-сервиса — помочь найти жильё максимально быстро и удобно. Визуально баннер оформлен в фирменных цветах интерфейса и включает краткое сообщение о функциональности системы.

В совокупности навигационная структура сайта выстроена так, чтобы минимизировать количество шагов от захода на сайт до получения первых результатов. Она логична, предсказуема и учитывает потребности как новых пользователей, так и тех, кто уже знаком с сервисом.

3.2.2 Система фильтрации

Блок фильтрации в веб-сервисе вынесен в отдельную визуально выраженную панель, расположенную сбоку или в верхней части каталога. Панель снабжена заголовком «Система фильтрации для поиска квартир», который ясно указывает на её назначение. Для визуального акцента используется фиолетовая цветовая гамма, гармонирующая с общей стилистикой сайта и обеспечивающая лёгкую фокусировку внимания пользователя. Панель постоянно доступна и не требует дополнительных действий для её активации.

Фильтр по цене реализован в виде двух текстовых полей с метками «от» и «до». Пользователь может вручную ввести интересующий его диапазон стоимости аренды в рублях. Такой подход обеспечивает точный контроль над результатами выдачи, позволяя исключить как слишком дорогие, так и подозрительно дешёвые объекты.

Фильтр для выбора срока аренды выполнен в виде удобных кнопок, каждая из которых уже содержит определённый интервал: от 1 месяца, от 6 месяцев и от 12 месяцев. Такой способ позволяет быстро указать нужный срок аренды, не вводя данные вручную. Это особенно удобно для тех, кто уже знает, как долго планирует снимать жильё.

Чтобы выбрать район, пользователь может воспользоваться выпадающим списком. В нём отображаются все доступные административные районы, которые зависят от выбранного региона. Список обновляется автоматически — так что пользователь видит только те районы, которые актуальны для выбранного города. Это помогает сузить поиск и найти варианты в нужной локации.

Фильтрация по площади квартиры сделана через поля для ввода минимального и максимального метража. Такой диапазон позволяет отсечь слишком маленькие или, наоборот, слишком большие квартиры. Интерфейс фильтра понятный и логичный, поэтому работать с ним легко и приятно.

Количество комнат выбирается с помощью набора кнопок: «Студия», «1», «2», «3», «4», «5». Каждая кнопка оформлена как отдельный элемент интерфейса, и пользователь может выбрать как один вариант, так и несколько одновременно. Такая реализация обеспечивает гибкость при задании требований к планировке жилья.

Фильтр по типу дома представлен в виде списка предустановленных вариантов: панельный, монолитный, кирпичный, монолитно-кирпичный. Выбор типа конструкции здания помогает учитывать предпочтения по теплоизоляции, звукоизоляции и общему качеству жилья.

Раздел дополнительных условий проживания реализован в виде двух чекбоксов: «Разрешено проживание с животными» и «Разрешено проживание с детьми». Пользователь может включить одну или обе опции в зависимости от своей жизненной ситуации, и система отфильтрует только те объекты, где такие условия явно указаны.

Этаж задаётся через три фиксированных варианта: «Первый», «Средний», «Последний». Выбор этажа может быть важен с точки зрения личных предпочтений, наличия лифта или специфики здания. Интерфейс позволяет выбрать несколько значений одновременно, чтобы охватить допустимый диапазон.

Наличие мебели и бытовой техники задаётся через набор отдельных фильтров, включающих такие позиции, как: мебель, холодильник, микроволновая печь, стиральная машина, интернет и ТВ. Эти фильтры представлены в виде чекбоксов, каждый из которых может быть активирован отдельно. Такая детализация особенно важна для арендаторов, не желающих заниматься самостоятельным обустройством квартиры.

В нижней части панели размещена кнопка «Применить фильтры к поиску квартиры», являющаяся основной точкой взаимодействия с системой. Нажатие на неё инициирует отправку текущих параметров на сервер, где они преобразуются в SQL-запрос или ORM-фильтрацию, после чего пользователю возвращается список релевантных объявлений. Кнопка визуально выделена и всегда находится в зоне видимости, что делает процесс поиска максимально удобным и понятным.

3.2.3 Результаты поиска

После того как пользователь применяет выбранные фильтры, система формирует результаты в виде вертикального списка карточек, каждая из которых представляет отдельное объявление об аренде квартиры. Такой формат отображения выбран неслучайно: он позволяет компактно представить большое количество информации и обеспечивает удобство визуального восприятия. Пользователь может быстро просматривать объявления, не теряя контекст и не переключаясь между страницами.

В верхней части каждой карточки располагается фотография объекта, отображающая одну из представленных в объявлении. Фото загружается в оптимальном разрешении и отображается с сохранением пропорций, чтобы не искажать восприятие. Визуальный образ помогает сразу отсеять нерелевантные варианты и облегчает предварительное сравнение объектов между собой.

Под фотографией размещается заголовок карточки, оформленный в формате: «Студия, 35 м² в ЖК „ЦЕХ“» или аналогичном. Он кратко описывает тип жилья, его площадь и, при наличии, жилой комплекс или ключевую характеристику. Такой формат позволяет пользователю мгновенно понять, что за объект перед ним, не вчитываясь в длинное описание.

В основной части карточки отображаются ключевые параметры квартиры, включая точную площадь, количество комнат, этаж, район и адрес. Эти данные расположены в структурированном виде, без лишнего текста, чтобы пользователь мог быстро сравнивать характеристики между несколькими объектами. Адрес при этом указывается полностью или частично, в зависимости от доступности данных на исходной платформе.

Для удобства визуального восприятия карточки дополнены метками и иконками, отражающими специфические условия аренды. Например, если в квартире есть стиральная машина, микроволновка или разрешено проживание с животными — соответствующие иконки отображаются прямо в карточке. Это позволяет на уровне визуального сканирования быстро выделять подходящие варианты без необходимости вникать в описание.

Цена аренды размещена справа и выделена крупным полужирным шрифтом. Пример отображения: «30 000 ₽/мес.». Такая подача позволяет сделать акцент на одном из главных параметров выбора, обеспечивая удобство и скорость оценки объекта. Размещение цены в правой части карточки согласуется с естественной схемой восприятия информации пользователем.

В нижней части карточки находится кнопка «Подробнее», при нажатии на которую происходит переход на внешний источник — тот сайт, с которого было получено объявление (например, Авито, ЦИАН и т.д.). Таким образом, пользователь может напрямую связаться с арендодателем или агентом через привычный интерфейс оригинальной платформы, не нарушая юридическую структуру размещения.

Все карточки оформлены в едином стиле, что способствует визуальной унификации и избавляет пользователя от перегрузки. В них отсутствует лишний текст, а акценты расставлены грамотно — на цену, параметры и ключевые условия. Такой подход полностью соответствует принципам визуального сканирования, которые широко используются в UX-дизайне для упрощения процесса выбора и ускорения принятия решения.

3.2.4 Поведение и взаимодействие

Интерфейс веб-сервиса построен на базе Next.js — современного фреймворка, позволяющего реализовывать динамически обновляемые интерфейсы без полной перезагрузки страницы. Это означает, что взаимодействие пользователя с системой происходит максимально быстро: при изменении состояния страницы загружаются только нужные компоненты, а не весь HTML-документ. Такой подход обеспечивает высокую производительность, быструю реакцию на действия пользователя и общее ощущение «живости» приложения.

Когда пользователь изменяет параметры фильтрации — например, вводит новый диапазон цены, выбирает тип жилья или отмечает разрешение на проживание с животными — интерфейс мгновенно инициирует асинхронный запрос через HTTP-клиент Axios. Этот запрос направляется на сервер FastAPI, который обрабатывает заданные параметры и возвращает список подходящих объявлений. Полученные данные динамически подставляются в интерфейс, и карточки объектов моментально перерисовываются без перезагрузки всей страницы, что делает процесс поиска быстрым и интерактивным.

Если в результате фильтрации не удалось найти ни одного подходящего объекта, система автоматически выводит информативное сообщение, предлагающее пользователю изменить параметры поиска. Это позволяет избежать ситуации, когда пользователь сталкивается с «пустой» страницей без пояснений. Вместо этого он получает мягкую подсказку о необходимости скорректировать фильтры и может легко вернуться к выбору, не теряя прогресс.

При нажатии на кнопку «Подробнее» в карточке объекта пользователь направляется по внешней ссылке на оригинальный источник объявления — это может быть сайт Авито, ЦИАН, Яндекс.Недвижимость или другой агрегатор. Переход осуществляется в новой вкладке или в текущей, в зависимости от реализации. Такой подход обеспечивает соблюдение юридических требований (веб-сервис не хранит контактную информацию владельцев) и позволяет пользователю сразу связаться с арендодателем через интерфейс той платформы, где было изначально размещено объявление.

Таким образом, весь процесс взаимодействия пользователя с интерфейсом построен на динамичном, логичном и последовательном механизме, где каждый шаг — от применения фильтров до просмотра конкретного предложения — реализуется без лишней задержки и с учётом удобства восприятия информации.

3.2.5 Соответствие эвристикам Нильсена

Принцип видимости состояния системы реализован через постоянную обратную связь между интерфейсом и действиями пользователя. При загрузке данных отображается индикатор активности, а при возникновении ошибок — соответствующее сообщение с пояснением. Также пользователь всегда видит количество найденных объектов в результате фильтрации, что помогает быстро оценить эффективность текущих параметров поиска и, при необходимости, скорректировать их.

Соответствие интерфейса реальному миру обеспечивается за счёт использования понятных, привычных названий фильтров и терминов. Все параметры, которые видит пользователь — такие как «цена», «площадь», «этаж», «наличие мебели», «возможность проживания с животными» и другие — это реальные характеристики жилья, знакомые каждому, кто ищет квартиру. Это упрощает взаимодействие и снижает нагрузку: пользователь сразу понимает, о чём идёт речь, и может легко ориентироваться.

Интерфейс даёт полную свободу: фильтры можно сбросить в любой момент. Не нужно обновлять страницу или вручную очищать каждый пункт — достаточно одного клика, чтобы начать поиск заново. Такая гибкость помогает исправить ошибки и пробовать новые параметры без лишних действий.

Все элементы управления — от кнопок до иконок — оформлены в едином стиле. Независимо от того, в каком разделе сайта находится пользователь, он видит знакомую структуру. Это помогает быстрее освоиться и делает работу с сервисом более понятной.

Чтобы избежать ошибок при вводе данных, в системе предусмотрена проверка всех форм. Она не даёт пользователю ввести что-то неподходящее или случайное, помогая сохранить точность поиска и корректность данных. Например, поля «цена» и «площадь» не позволяют ввести отрицательные значения или символы, не соответствующие числовому формату. Это исключает ситуации, в которых пользователь непреднамеренно формирует некорректный запрос и не получает результатов.

Принцип распознавания вместо запоминания выражается в том, что все доступные фильтры и опции находятся на виду. Пользователю не нужно догадываться, где находится нужная настройка, или вспоминать, какие параметры он вводил ранее. Визуальная доступность всех опций сокращает количество действий и делает взаимодействие интуитивным.

Гибкость интерфейса достигается благодаря широкому набору фильтров, которые можно комбинировать в любых сочетаниях. Пользователь может точно настроить поиск под свои предпочтения — например, задать конкретный район, тип дома, разрешения, и при этом ограничить результат по цене или площади. Такая гибкость делает систему универсальной и применимой к разным сценариям аренды.

Принцип эстетики и минимализма реализован за счёт использования большого количества белого пространства, чёткой визуальной иерархии и отказа от лишних декоративных элементов. Интерфейс не перегружен, и внимание пользователя фокусируется на ключевых элементах: фильтрах, карточках объектов и основных действиях.

Помощь в исправлении ошибок реализуется через внятные и контекстные сообщения. Если пользователь вводит некорректные значения или забывает указать обязательный параметр, система не просто блокирует действие, но поясняет, в чём именно заключается ошибка и как её исправить.

Наконец, принцип документации и доступности подсказок отражён в оформлении интерфейса: кнопки и метки снабжены короткими подписями и иконками, которые делают их значение очевидным. Даже без предварительного обучения пользователь легко понимает, как работает система, что каждая кнопка означает и какой результат она вызовет.

# 3.3 Архитектура веб-сервиса для анализа и сравнения условий аренды недвижимости

Архитектура веб-сервиса построена по принципам модульности, асинхронности и разделения ответственности, что обеспечивает гибкость, масштабируемость и отказоустойчивость решения. Основу архитектуры составляют три ключевых уровня: интерфейсный, прикладной (серверный) и уровень обработки фоновых задач. Все взаимодействие компонентов происходит через HTTP-запросы и очереди сообщений, а хранение данных реализовано в централизованной реляционной СУБД.

# 3.3.1 Общий обзор архитектуры

Архитектура веб-сервиса построена на принципах модульности, разделения ответственности и асинхронной обработки. Все компоненты системы взаимодействуют через чётко определённые интерфейсы и работают совместно для обеспечения стабильной и масштабируемой работы сервиса. На архитектурной диаграмме можно выделить несколько ключевых элементов, каждый из которых выполняет свою специализированную функцию.

Первым участником архитектуры является пользователь, взаимодействующий с системой через стандартный веб-браузер. Именно с его действия начинается работа сервиса: пользователь авторизуется, выбирает регион, задаёт фильтры и анализирует предложения. Графический интерфейс обеспечивает точку входа в систему, преобразуя действия пользователя в запросы, передаваемые в серверную часть для обработки.

Клиентская часть реализована на базе Next.js — фреймворка, основанного на React. Он отвечает за отображение форм фильтрации, представление карточек с результатами поиска, а также за реализацию навигации между разделами сайта. Интерфейс работает быстро и отзывчиво за счёт частичной подгрузки компонентов и динамического рендеринга, позволяя пользователю взаимодействовать с сервисом без перезагрузки страницы.

Для обмена данными между фронтендом и бэкендом используются Axios HTTP-запросы. Это лёгкий и надёжный JavaScript-клиент, который отправляет асинхронные запросы по REST-протоколу. Каждый фильтр, нажатие кнопки или действие пользователя формирует запрос к серверу, где он обрабатывается и возвращает ответ в формате JSON.

Серверная логика построена на FastAPI — современном и высокопроизводительном Python-фреймворке. Он принимает все входящие запросы, валидирует параметры, выполняет бизнес-логику (например, фильтрацию, дедупликацию, сохранение истории) и взаимодействует с базой данных. Благодаря своей асинхронной природе FastAPI обеспечивает высокую скорость отклика и стабильную работу при больших объёмах данных.

В качестве слоя хранения данных используются две системы: PostgreSQL и Redis. PostgreSQL — это реляционная СУБД, в которой хранятся основные сущности, включая информацию о пользователях, квартирах, избранном и истории поиска. Redis, в свою очередь, используется как высокопроизводительное хранилище ключ-значение. Он выполняет функции кеша для ускорения отклика и служит брокером сообщений для передачи задач в очередь.

Обработка долгосрочных операций (например, парсинг, обновление базы) вынесена в фоновую систему на базе Celery. Это менеджер задач, позволяющий выполнять ресурсоёмкие действия вне основного потока сервера. Celery обрабатывает задачи асинхронно, распределяя нагрузку между воркерами и обеспечивая бесперебойную работу интерфейса даже при параллельном выполнении десятков фоновых задач.

Связующим звеном между FastAPI и Celery выступает RabbitMQ — брокер сообщений. Он принимает задачи, поставленные сервером, организует их в очереди и передаёт Celery для выполнения. RabbitMQ поддерживает подтверждение доставки, маршрутизацию задач и надёжную очередь, что важно для стабильной работы при взаимодействии с внешними источниками.

Последним элементом архитектуры являются парсеры — автономные Python-модули, каждый из которых настроен на работу с определённой внешней платформой, такой как Авито, ЦИАН или Яндекс.Недвижимость. Парсеры извлекают данные с сайтов, обходят защиту от автоматизации и передают структурированную информацию в базу. Они подключаются через Celery, работают по расписанию или по запросу и обеспечивают обновление информации в реальном времени.

В совокупности все эти компоненты формируют устойчивую, масштабируемую и модульную архитектуру веб-сервиса, позволяя выполнять как мгновенные, так и длительные операции, не нарушая пользовательского опыта.

# 3.3.2 Последовательность взаимодействия компонентов

Процесс обработки пользовательского запроса начинается с его инициализации на стороне клиента. Пользователь, взаимодействуя с интерфейсом веб-сервиса через браузер, применяет выбранные фильтры: указывает диапазон цены, район, срок аренды, этажность и другие параметры. После этого он нажимает кнопку «Поиск», тем самым инициируя отправку запроса. Клиентская часть, реализованная на Next.js, формирует объект запроса в формате JSON и с помощью HTTP-клиента Axios отправляет его на сервер, используя метод асинхронной передачи данных. Благодаря этому взаимодействие происходит без полной перезагрузки страницы и с минимальной задержкой.

На стороне сервера запрос обрабатывается фреймворком FastAPI, который принимает параметры фильтрации, проверяет их на корректность и запускает соответствующую бизнес-логику. Если информация, соответствующая запросу, уже содержится в базе данных, FastAPI обращается к PostgreSQL и выполняет выборку необходимых объявлений. В случае, если запрашиваемые данные отсутствуют или нуждаются в обновлении, сервер инициирует фоновую задачу и ставит её в очередь Celery через RabbitMQ. Такое решение позволяет избежать лишней нагрузки на сервер и не блокирует основной поток обработки запросов.

Следующим этапом является обработка фоновой задачи системой Celery. Получив задачу из очереди RabbitMQ, Celery активирует нужные модули парсинга, соответствующие выбранному региону. Парсеры начинают сбор данных с внешних сайтов, таких как Авито, ЦИАН и Яндекс.Недвижимость. После получения информации происходит очистка от дубликатов: объявления сравниваются по адресу, этажу и цене. Затем данные проходят нормализацию — все параметры приводятся к единому формату, соответствующему структуре базы. Объявления сохраняются в PostgreSQL, а возможные ошибки (например, 403 или 404) логируются в таблицу parser\_logs, что позволяет отслеживать сбои и при необходимости оперативно вмешиваться в работу системы.

После завершения всех этапов обработки сервер формирует ответ для пользователя в формате JSON. Этот ответ содержит список релевантных объектов, удовлетворяющих заданным фильтрам. Клиентская часть получает данные и преобразует их в визуально оформленные карточки, каждая из которых содержит ключевую информацию об объекте: цену, площадь, условия аренды, фотографии и ссылку на оригинальный источник. Отображение карточек происходит без задержек и перезагрузок, обеспечивая пользователю удобный и быстрый способ анализа доступных вариантов. Таким образом, полный цикл от запроса до выдачи результата реализуется в рамках распределённой архитектуры с разделением задач между клиентом, сервером, брокером и фоновыми воркерами.

# 3.3.3 Особенности реализации

Применение асинхронной обработки данных является одним из ключевых решений в архитектуре веб-сервиса. Оно позволяет вынести ресурсоёмкие процессы, такие как парсинг объявлений, агрегация данных и периодическое обновление базы, в отдельный фоновой поток, обрабатываемый Celery. Благодаря этому основной API, обслуживающий пользовательские запросы, остаётся лёгким и быстрым. Пользователь получает ответ практически мгновенно, а все тяжёлые задачи выполняются параллельно, не блокируя работу интерфейса и не создавая задержек.

Высокая масштабируемость системы достигается за счёт возможности разнести её компоненты на разные физические или виртуальные серверы. Например, парсеры, очередь задач Celery и брокер сообщений RabbitMQ могут функционировать независимо от основного сервера, на котором работает FastAPI. Это позволяет гибко увеличивать вычислительные мощности в зависимости от текущей нагрузки и эффективно распределять ресурсы между компонентами, особенно в условиях растущего количества пользователей или источников данных.

Важным аспектом системы является её безопасность. Она обеспечивается на нескольких уровнях. Во-первых, все входные параметры проходят строгую валидацию: система проверяет тип данных, допустимые значения и формат, исключая потенциально опасные запросы. Во-вторых, архитектура защищена от SQL-инъекций за счёт использования ORM и параметризированных запросов. В-третьих, для авторизации и контроля доступа применяется механизм JWT (JSON Web Token), который обеспечивает надёжную идентификацию пользователей и защищает данные от несанкционированного доступа.

Сервис обладает высокой гибкостью архитектуры, что особенно важно при необходимости подключения новых источников данных. Каждый парсер функционирует как отдельный автономный модуль, настроенный на определённую платформу (например, Авито или ЦИАН). Добавление нового источника не требует изменения основного кода сервера или интерфейса — достаточно разработать соответствующий парсер и зарегистрировать его в очереди Celery. Такая модульность упрощает расширение системы и адаптацию под изменяющиеся внешние условия.

На всех этапах обработки данных реализована система логирования, которая фиксирует ключевые события, ошибки и промежуточные состояния. Логи записываются каждый раз, когда запускается парсер, а также при любых сбоях — например, если сайт возвращает ошибку 403 или 404, или если возникают проблемы с базой данных или доступом к внешнему источнику. Эти записи помогают разработчикам быстро находить и устранять неполадки. Кроме того, они позволяют следить за тем, насколько стабильно работает система в целом. Такой подход делает работу сервиса более надёжной и даёт возможность оперативно реагировать на любые непредвиденные ситуации.

# 3.3.4 Механизмы защиты и устойчивости

Если во время работы парсер сталкивается с ошибкой 403 (Forbidden), это обычно означает, что сайт-источник временно заблокировал IP-адрес, посчитав его подозрительным из-за автоматических запросов. Чтобы избежать полной остановки процесса сбора данных, система автоматически активирует механизм смены IP-адреса через пул прокси-серверов. Парсер переключается на новый прокси и повторяет запрос. Такой подход обеспечивает устойчивость парсинга даже при жёстких ограничениях со стороны внешнего ресурса и минимизирует вероятность потери данных.

В случае, если внешний сайт изменил HTML-структуру, что часто случается без предварительного уведомления, встроенная логика парсера фиксирует сбой при извлечении данных. Обнаруженная ошибка логируется в базе данных, в таблицу parser\_logs, с указанием URL, времени и характера сбоя. Дополнительно система может сформировать уведомление для администратора или разработчика, чтобы оперативно внести изменения в модуль парсера. Такая система оповещения позволяет сократить время отклика на внешние изменения и поддерживать актуальность сервиса без длительных сбоев.

Для поддержания базы данных в актуальном состоянии применяется механизм плановых задач на основе Celery beat — расширения для планировщика задач Celery. С его помощью задаются расписания обновления информации: например, раз в 12 часов или каждый день в заданное время. В соответствии с графиком Celery автоматически инициирует запуск парсера для каждого активного региона. Это позволяет не только регулярно обновлять данные без участия пользователя, но и поддерживать свежесть выдачи в интерфейсе, особенно если объект был изменён или снят с публикации. Такой подход гарантирует, что пользователь всегда получает максимально актуальные предложения.

# 3.3.5 Поддержка и расширение

Проект построен по принципу микросервисной архитектуры, что означает логическое и физическое разделение всех основных компонентов системы. Каждый ключевой блок — интерфейс пользователя, серверная логика (API), база данных, модули парсинга, система фоновых задач и брокер сообщений — функционирует как отдельный самостоятельный сервис. Это позволяет не только упростить разработку и сопровождение, но и обеспечить отказоустойчивость: сбой одного компонента не приводит к полной недоступности системы.

Интерфейс реализован как отдельное клиентское приложение, функционирующее независимо от API. Он получает данные через HTTP-запросы и может быть размещён на отдельном сервере или CDN, что ускоряет загрузку и повышает отзывчивость. Благодаря изоляции фронтенда, любые изменения в логике отображения, дизайне или структуре страниц не требуют вмешательства в серверную часть и не влияют на работу остальных компонентов.

Серверная логика (FastAPI) развёрнута отдельно от базы данных и интерфейса. Она отвечает за обработку пользовательских запросов, бизнес-логику и взаимодействие с другими сервисами — такими как Celery и PostgreSQL. Разделение API и логики обработки позволяет обновлять или масштабировать серверную часть независимо от других компонентов.

База данных представлена двумя независимыми системами: PostgreSQL для хранения основного содержимого и Redis — для кеширования и работы с очередями. Оба хранилища могут масштабироваться горизонтально или вертикально, в зависимости от требований к производительности и объёму данных. Такое распределение нагрузок между слоями хранения делает систему более гибкой и надёжной.

Парсеры, разработанные как отдельные модули, могут запускаться параллельно и быть масштабированы в зависимости от количества подключённых источников. Каждый парсер обслуживает свою платформу (например, Авито или ЦИАН) и работает независимо от остальных. Это позволяет добавлять новые источники без вмешательства в основной API или архитектуру интерфейса.

Брокер сообщений RabbitMQ и система фоновых задач Celery функционируют как отдельные микросервисы, управляя задачами парсинга, обновления и фильтрации данных. Такая изоляция позволяет балансировать нагрузку, распределять задачи между воркерами и следить за выполнением без блокировки основного сервера.

Благодаря такому построению, проект обладает высокой степенью масштабируемости и гибкости. Каждый компонент может быть масштабирован отдельно — как в рамках увеличения ресурсов одного сервера, так и путём распределения нагрузки на несколько машин. Это особенно важно при росте пользовательской базы, расширении географии парсинга или увеличении количества данных.

Наконец, микросервисный подход даёт возможность в будущем безболезненно внедрять новые модули и технологии. Например, можно добавить рекомендательные механизмы, основанные на поведенческой аналитике, или модули машинного обучения для предиктивного анализа. При этом не потребуется переписывать существующую архитектуру, поскольку новые сервисы легко встраиваются в общую структуру как независимые блоки.

# 3.4 Интеграция с платформами аренды квартир для анализа и сравнения условий аренды квартир

Одним из ключевых компонентов функциональности веб-сервиса является автоматизированный сбор и агрегация данных с популярных внешних платформ, на которых публикуются объявления об аренде недвижимости. Этот механизм обеспечивает наполнение системы актуальной, разнообразной и структурированной информацией, без которой полноценная работа сервиса была бы невозможна. Автоматизация позволяет избежать ручного ввода, минимизирует человеческий фактор и ускоряет процесс актуализации базы.

Интеграция с такими крупными платформами, как ЦИАН, Авито, Яндекс.Недвижимость и другие региональные или нишевые ресурсы, критически важна для обеспечения широты охвата и полноты предложения. Только за счёт регулярного парсинга объявлений с разных источников можно гарантировать пользователю возможность сравнивать варианты из всего рыночного объёма предложений. Это создаёт реальную ценность сервиса — его не нужно вручную переключать между сайтами, все предложения агрегированы в одном интерфейсе.

Кроме того, актуальность данных напрямую зависит от частоты и стабильности интеграции с этими внешними площадками. В случае, если объявления не обновляются регулярно или поступают с задержкой, пользователь может получить устаревшую информацию, что отрицательно скажется на доверии к сервису. Поэтому реализация надёжного механизма получения данных с возможностью быстрого обновления стала важнейшим приоритетом при проектировании архитектуры.

Проект изначально разрабатывался с учётом необходимости масштабируемой и устойчивой интеграции с внешними источниками. Это означает, что система была спроектирована так, чтобы легко поддерживать подключение новых платформ, даже если они имеют иную структуру данных, методы защиты от автоматизации или другие особенности. Каждый внешний источник обрабатывается отдельным парсером, который можно запускать независимо, обновлять без остановки всей системы и адаптировать под изменения на стороне сайта.

Важным требованием при проектировании архитектуры стала её адаптируемость к изменениям внешней среды. Поскольку сайты-источники регулярно обновляют верстку, внедряют защиту от парсинга или меняют формат представления данных, было необходимо предусмотреть модульную структуру интеграции. Это позволило каждому парсеру функционировать как отдельный блок, который можно доработать или временно отключить без риска повлиять на остальную часть веб-сервиса.

# 3.4.1 Цели и задачи интеграции

Интеграция с внешними платформами по аренде недвижимости решает одну из наиболее критичных задач — получение актуальной информации об объектах в режиме, максимально приближённом к реальному времени. Благодаря автоматическому парсингу система регулярно обновляет базу данных, исключая устаревшие объявления и добавляя новые предложения сразу после их появления на первоисточниках. Это особенно важно в условиях высокой динамики рынка аренды, где квартиры могут становиться недоступными в течение нескольких часов после публикации.

Следующим важным направлением является объединение данных из различных источников в едином пользовательском интерфейсе. Пользователю больше не требуется вручную проверять каждый сайт (например, Авито, ЦИАН или Яндекс.Недвижимость), переходя между вкладками и сравнивая условия аренды. Вся информация агрегируется в одном месте, представляется в унифицированной структуре и подаётся в виде карточек с одинаковым форматом, что значительно экономит время и упрощает восприятие.

Одним из важных преимуществ интеграции становится возможность убрать дублирующие объявления — ведь часто один и тот же объект размещают и собственник, и агент. Это обычная ситуация на рынке аренды. Сервис анализирует совпадения по адресу, этажу и цене с учётом небольшой погрешности. Если дубли находятся, система оставляет только одно — то, которое содержит больше информации и более выгодно для пользователя. Такой подход делает результаты поиска более чистыми и уменьшает риск ошибочного выбора.

Кроме того, общая архитектура позволяет настроить единый фильтр для всех предложений, независимо от того, с какого сайта они были получены. Пользователь может использовать как базовые, так и расширенные параметры фильтрации — например, по цене, сроку аренды или условиям проживания — и сразу видеть результат без необходимости вручную повторять одни и те же шаги на разных платформах. Это значительно упрощает и ускоряет поиск квартиры.

Ещё один плюс — агрегирование данных со всех доступных источников. Благодаря этому пользователь получает максимально широкий выбор и может сравнивать варианты в одинаковых условиях, анализировать разницу в цене и параметрах для одного района или города. Это даёт более полную и честную картину рынка, помогая найти лучший вариант для аренды

# 3.4.2 Формат интеграции: через парсеры

Во многих случаях крупные площадки по аренде недвижимости, такие как Авито, ЦИАН и Яндекс.Недвижимость, не предлагают открытых API для массового доступа. А даже если такие API существуют, они обычно имеют строгие ограничения: лимиты на количество запросов, обязательную оплату, регистрацию и подписание юридических соглашений. Это серьёзно осложняет их использование для автоматизированного сбора данных в больших объёмах. По этой причине в проекте было принято решение создать архитектуру, основанную на парсерах. Каждый парсер запускается как отдельная задача в Celery — системе фоновой обработки, что позволяет собирать данные, обходя эти ограничения.

Каждый парсер в системе выполняет первую важную функцию — обращение к веб-страницам соответствующего источника. Парсер формирует HTTP-запрос, эмулируя поведение обычного браузера, и получает HTML-страницу объявления или поисковой выдачи. При этом учитываются особенности структуры URL, параметры фильтрации и механизмы перехода по страницам, если используются пагинации.

После получения HTML-разметки парсер приступает к извлечению нужных данных с помощью XPath, CSS-селекторов и регулярных выражений. Эти инструменты позволяют точно и эффективно находить нужные фрагменты кода, такие как цена, площадь, этаж, адрес, наличие техники, фотографии и т.д. Каждый параметр выделяется отдельно, проходит предварительную очистку и преобразование в пригодный для дальнейшей обработки формат.

На следующем этапе парсер интерпретирует извлечённые параметры в структуру, соответствующую внутренней логике веб-сервиса. Это значит, что поля, полученные с разных платформ, стандартизируются — например, значение «трёхкомнатная квартира» может быть приведено к числовому значению «3», а «кирпичный дом» — к соответствующему типу из словаря справочных данных. Благодаря такой нормализации система может обрабатывать данные единообразно, независимо от источника.

Завершающим шагом является формирование структуры данных, совместимой с PostgreSQL. Все извлечённые и интерпретированные данные оформляются в виде объектов, соответствующих схемам таблиц базы данных: offers, regions, equipment, conditions и другим. Объекты передаются в базу с соблюдением ограничений, связей и логики хранения, что гарантирует целостность данных и возможность их последующей фильтрации и сравнения.

Для каждого внешнего ресурса создаётся отдельный парсер-модуль, поскольку HTML-структура, формат данных, наличие AJAX-загрузки и поведение пагинации различаются. Парсер учитывает, как сайт подгружает данные (например, через JavaScript), как устроены переходы между страницами и каким образом реализована защита от ботов. Это позволяет создать стабильные и адаптивные механизмы сбора данных, которые легко поддерживать и обновлять в случае изменений на стороне внешней платформы.

# 3.4.3 Работа с защитой от блокировок

Многие современные платформы аренды недвижимости применяют разнообразные защитные механизмы, направленные на ограничение или блокировку автоматического сбора информации. Среди наиболее распространённых методов — проверка наличия и прохождение капчи, отслеживание частоты запросов с одного IP-адреса, анализ HTTP-заголовков (в частности, User-Agent), проверка поведения пользователя и даже детектирование JavaScript-исполнения на клиентской стороне. Все эти методы делают задачу парсинга технически сложной и требуют внедрения специальной системы обхода защит.

Первым и основным элементом защиты в веб-сервисе является ротация прокси-серверов. Каждый HTTP-запрос, отправляемый парсером, проходит через один из заранее настроенных прокси, что позволяет использовать новый IP-адрес при каждом обращении. Это снижает риск блокировки со стороны сайта-источника, так как запросы не идут последовательно с одного и того же IP, что обычно считается подозрительным поведением. Использование прокси-пулов делает процесс более устойчивым и масштабируемым.

Следующим механизмом является генерация случайных значений в заголовке User-Agent. Это позволяет парсеру имитировать работу различных браузеров и устройств: настольных компьютеров, ноутбуков, планшетов и даже смартфонов. Благодаря такому подходу создаётся впечатление, что система имитирует работу реальных пользователей. Заголовки формируются случайным образом из списка популярных комбинаций, включая тип браузера, версию, операционную систему и язык интерфейса.

Для увеличения реалистичности поведения используются искусственные задержки между запросами. Вместо того чтобы посылать десятки обращений подряд, как это делает типичный скрипт, парсер делает паузы в 1–5 секунд между действиями, имитируя поведение обычного пользователя. Дополнительно в некоторые запросы встраиваются случайные параметры (например, параметры курсора или фильтра), чтобы система-источник не могла построить шаблон активности и выявить автоматизацию.

Если при выполнении запроса сервер источника возвращает ошибку 403 (доступ запрещён), 429 (превышение лимита запросов) или 503 (временная недоступность), система автоматически повторяет задачу. Повтор выполняется с новым набором параметров — другим прокси, другим User-Agent, возможной паузой и другим порядком обращения. Это помогает системе оставаться стабильной даже в случае временных сбоев и агрессивной политики защиты со стороны сайтов-источников. При этом пользователю не нужно вмешиваться или что-то настраивать — всё работает автоматически.

Для того чтобы отслеживать любые проблемы, в системе предусмотрено логирование технических сбоев. Все ошибки и сбои фиксируются в отдельной таблице parser\_logs. Там сохраняется информация об источнике, коде ошибки, URL-адресе, времени и причине сбоя. Эти данные можно использовать для анализа работы источников, корректировки парсера или усиления защиты.

Все эти механизмы делают систему парсинга устойчивой и надёжной. Она продолжает собирать данные и обновлять базу, даже если некоторые источники временно становятся недоступны. Это обеспечивает стабильное обновление базы, снижает риски потери данных и делает систему пригодной для долгосрочного использования без регулярного вмешательства администратора.

# 3.4.4 Унификация и нормализация данных

Данные, получаемые с различных платформ аренды недвижимости, имеют разнородную структуру и неоднородный формат представления. Это связано с тем, что каждая площадка использует собственные правила отображения информации. Например, на одном сайте может быть указан конкретный этаж квартиры, тогда как на другом — только общее количество этажей в доме. Также отличается способ указания стоимости: одни ресурсы публикуют цену аренды за месяц, другие — за сутки, а иногда — с учётом дополнительных услуг или коммунальных платежей. Эти несоответствия требуют глубокой обработки данных до их попадания в основную базу веб-сервиса.

Первым и важнейшим этапом обработки является стандартизация полей. Все основные параметры, включая площадь квартиры, стоимость аренды, этаж, количество комнат, а также наличие мебели, бытовой техники и других удобств, приводятся к единому внутреннему формату. Например, цены пересчитываются в рублях за месяц, даже если изначально указаны в другом виде. Этажность представляется в виде пары значений: «этаж / общее количество этажей». Благодаря стандартизации становится возможным корректное сравнение объявлений между собой, независимо от исходной структуры данных.

Следующим шагом является очистка информации от технического и визуального мусора. В процессе парсинга в данные могут попасть HTML-теги, спецсимволы, знаки форматирования и скрытые элементы, которые не несут смысловой нагрузки, но мешают анализу. Все такие элементы удаляются, строки очищаются от лишних пробелов, кавычек и других неинформативных символов. Также на этом этапе исключаются дублирующие или пустые значения, чтобы в базу попала только актуальная и полезная информация.

Далее осуществляется проверка полноты каждого объявления. Не все платформы предоставляют полный набор параметров — где-то отсутствуют фотографии, где-то не указан точный адрес или площадь. Такие объявления автоматически помечаются специальным тегом как «ограниченные по данным». Это позволяет не исключать их из базы, но предупреждать пользователя о возможной неполноте информации. При необходимости такие записи могут быть обновлены повторным парсингом или заменены более полной версией.

Заключительным этапом является формирование унифицированной карточки объекта, предназначенной для отображения в пользовательском интерфейсе и сравнения с другими предложениями. Карточка включает все стандартизированные параметры, фотографии, иконки условий (например, разрешение на проживание с животными или наличие интернета), а также ссылку на оригинальный источник. Благодаря этому пользователь может анализировать и сравнивать объявления из разных источников, не ощущая разницы в формате представления данных. Такая структура позволяет обеспечить единообразие выдачи, повысить удобство восприятия информации и улучшить качество пользовательского опыта.

# 3.4.5 Дедупликация объявлений

На рынке аренды недвижимости один и тот же объект может быть размещён на нескольких платформах одновременно, зачастую с различиями в оформлении, ценах или наполнении объявления. Это приводит к дублированию информации и может дезориентировать пользователя, создавая иллюзию большего количества уникальных предложений, чем есть на самом деле. Чтобы обеспечить чистоту выдачи и повысить достоверность результатов поиска, в веб-сервисе реализован специализированный механизм дедупликации.

Первым критерием для выявления дубликатов служит совпадение адреса объекта, включая название улицы, номер дома, а при наличии — и номер квартиры. Адрес является уникальным идентификатором объекта недвижимости и позволяет достаточно точно определить, повторяется ли объявление с разных источников. Если система фиксирует совпадение по адресу, вероятность дублирования считается высокой и активируются дополнительные критерии проверки.

Второй параметр — совпадение этажа и площади квартиры. Даже если адрес указан не полностью или записан в разном формате, технические характеристики объекта, такие как этаж и общая площадь, позволяют уточнить идентичность. Например, если в двух объявлениях указана одинаковая площадь в 35 м² и оба расположены на третьем этаже по одному и тому же адресу, система с высокой долей уверенности классифицирует их как дублирующие.

Третьим фактором дедупликации является небольшое расхождение в цене. В реальных условиях один и тот же объект может быть размещён на разных площадках с незначительной разницей в стоимости, связанной с политикой агентств, округлением, наличием дополнительных услуг или комиссий. В системе установлен допустимый порог в 3% расхождения по цене. Если эта разница не превышает установленный предел, объявления считаются идентичными, и включаются в процесс устранения дубликатов.

Если все три условия выполняются одновременно — совпадает адрес, этаж, площадь, и цена находится в пределах допустимой разницы — система определяет, что речь идёт об одном и том же объекте. Из множества дублирующихся объявлений сохраняется только одно, при этом выбирается вариант с наиболее полной информацией (например, с фотографиями, точным адресом и описанием) и минимальной ценой. Такой подход позволяет пользователю видеть только актуальные, неповторяющиеся предложения и упрощает процесс сравнения между объектами.

Таким образом, механизм дедупликации играет важную роль в формировании корректной и достоверной выдачи, устраняя информационный шум и повышая качество пользовательского опыта.

# 3.4.6 Переход к внешнему ресурсу

Каждое объявление, которое отображается в интерфейсе веб-сервиса, содержит внешнюю гиперссылку на оригинальный источник, откуда было получено это предложение. Ссылка размещается в карточке объекта, обычно в виде кнопки с надписью «Подробнее» или аналогичной формулировкой. При нажатии на неё пользователь переходит на сайт-источник, такой как Авито, ЦИАН или Яндекс.Недвижимость. Этот переход открывает доступ к контактной информации, расширенному описанию и другим возможностям взаимодействия, предусмотренным на исходной платформе.

Во-первых, наличие внешней ссылки позволяет пользователю связаться с арендодателем напрямую через интерфейс оригинального сайта, соблюдая все правила и процедуры, установленные этой платформой. Таким образом, веб-сервис не вмешивается в коммуникацию между сторонами сделки, а лишь предоставляет структурированный доступ к информации. Пользователь может использовать форму обратной связи, написать сообщение или позвонить, если такая возможность доступна на платформе-источнике.

Во-вторых, такой подход позволяет избежать необходимости хранения персональных данных владельцев объектов недвижимости. Это критически важно с точки зрения соблюдения действующего законодательства, в том числе Федерального закона №152-ФЗ «О персональных данных». Сервис не хранит номера телефонов, имена, адреса электронной почты и другие чувствительные сведения, тем самым устраняя потенциальные риски, связанные с защитой данных.

Третьим аргументом в пользу использования внешней ссылки является предотвращение блокировки со стороны источников за несанкционированное копирование и распространение контента. Многие платформы запрещают полное копирование объявлений и особенно передачу контактной информации без официального партнёрства. Сохраняя только обобщённые параметры объектов (цена, площадь, условия аренды) и ссылаясь на оригинальный сайт, веб-сервис выполняет роль посредника и не нарушает права владельцев информации.

Таким образом, веб-сервис функционирует как агрегатор данных и инструмент их сравнения, а не как самостоятельная платформа размещения объявлений. Его задача — предоставить пользователю удобный способ выбора и анализа предложений, собранных из разных источников, не нарушая при этом ни технических, ни юридических норм. Такой подход позволяет сохранить нейтральную позицию и обеспечить устойчивость проекта при его дальнейшем развитии.

# 3.4.7 Расширение списка источников

Система веб-сервиса изначально проектировалась с прицелом на масштабируемость и лёгкость расширения, что позволяет без значительных изменений в основной архитектуре подключать новые источники данных. Такая гибкость достигнута благодаря модульному принципу построения, при котором каждый компонент отвечает за строго ограниченную область функциональности. Это означает, что добавление новой платформы требует лишь минимального вмешательства в уже существующий код и не влияет на стабильность других модулей.

Первым шагом при интеграции нового источника является разработка отдельного парсер-модуля. Этот модуль реализуется на языке Python и создаётся специально под HTML-структуру и формат конкретной платформы. Парсер должен уметь обращаться к страницам с объявлениями, корректно обрабатывать пагинацию, извлекать нужные параметры (такие как цена, площадь, этаж, тип жилья) и возвращать их в унифицированном виде. Модуль работает автономно и легко подключается к общему пулу задач.

После разработки парсера необходимо добавить его в очередь фоновых задач Celery, чтобы обеспечить асинхронное выполнение. Для этого создаётся отдельная задача, регистрируемая в конфигурации сервиса. Celery будет вызывать её в соответствии с расписанием или по запросу от backend-сервера. Это гарантирует, что парсинг происходит параллельно с другими задачами и не влияет на производительность основного API.

Следующим этапом является описание соответствия параметров, полученных с новой платформы, со стандартной моделью данных, принятой в системе. Разные сайты используют различные наименования, единицы измерения и структуру полей. Например, один источник может указывать количество комнат текстом («однокомнатная»), другой — числом. Система должна интерпретировать и привести такие данные к единому формату (например, rooms = 1). Это обеспечивает корректную фильтрацию, сравнение и отображение карточек в интерфейсе.

Дополнительно необходимо настроить логику дедупликации и обработки фотографий, получаемых с новой платформы. Механизм дедупликации должен учитывать потенциальные пересечения с уже существующими записями в базе (по адресу, этажу, цене), чтобы исключить повторное отображение одного и того же объекта. Также важно определить, как фотографии будут сохраняться, обрабатываться и выводиться в интерфейсе: требуется указать их формат, допустимое разрешение и правила сортировки (например, показывать первую фотографию как основную).

Благодаря модульной структуре и гибкой архитектуре, подключение новой платформы осуществляется быстро и без риска для стабильности всей системы. Каждый новый источник становится частью общего механизма агрегации и расширяет охват предложений, что повышает ценность сервиса для конечного пользователя.

# Выводы

Третья глава была посвящена подробному описанию архитектуры, проектирования и реализации веб-сервиса для анализа и сравнения условий аренды недвижимости. В результате рассмотрения технологической базы, принципов построения интерфейса и серверной логики можно сделать вывод о продуманности и технологической зрелости разработанного решения.

Были обоснованы и реализованы технологические выборы на всех уровнях архитектуры. В клиентской части использован фреймворк Next.js, обеспечивающий высокую производительность, SEO-оптимизацию и гибкую маршрутизацию без перезагрузки страниц. Axios применяется для обмена данными с сервером, используя формат JSON, что обеспечивает быструю и удобную коммуникацию между компонентами.

Сервер реализован на FastAPI — асинхронном фреймворке с высокой производительностью, строгой типизацией и встроенной генерацией документации. Серверная логика модульна, легко масштабируема и обслуживает как пользовательские запросы, так и фоновые задачи через Celery и RabbitMQ. В качестве основного хранилища используется PostgreSQL, а Redis выполняет функции кеша и брокера сообщений.

Интерфейс сервиса ориентирован на пользователя, спроектирован с учётом 10 эвристик Нильсена и принципов визуального минимализма. В нём реализованы наглядная система фильтрации, интуитивная навигация, унифицированные карточки объектов и система обратной связи. Поведение интерфейса построено на асинхронной логике и полностью исключает перезагрузки страниц, что обеспечивает высокий уровень UX.

Архитектура веб-сервиса построена по принципам микросервисности, с чётким разделением компонентов и поддержкой горизонтального масштабирования. Система выдерживает нагрузку, устойчива к сбоям и допускает подключение новых источников данных без влияния на стабильность основной функциональности. Парсеры реализованы как отдельные модули и поддерживают обход защит от автоматизации.

Особое внимание уделено интеграции с внешними платформами: реализован механизм нормализации и унификации данных, дедупликации объектов, автоматического обновления и сохранения ссылок на оригинальные источники. Это позволяет использовать сервис как агрегатор, не нарушая юридические требования, и даёт пользователю широкий охват предложений.

Таким образом, в рамках главы была разработана и описана современная, надёжная и масштабируемая архитектура веб-сервиса, которая успешно решает поставленные задачи: агрегирует данные с разных платформ, позволяет проводить фильтрацию и сравнение, обеспечивает стабильную и быструю работу как при стандартной, так и при высокой нагрузке.

# 4 Тестирование и апробация веб-сервиса для анализа и сравнения условий аренды недвижимости

После завершения этапов проектирования и разработки веб-сервиса наступает не менее важный этап — тестирование и апробация системы. Именно на этом этапе осуществляется комплексная проверка всей функциональности, выявляются возможные ошибки, а также оценивается стабильность и соответствие сервиса заявленным задачам. Учитывая сложную архитектуру решения, включающую интеграцию с внешними источниками, асинхронную обработку данных и активное взаимодействие с пользователем, тестирование необходимо проводить на всех уровнях — от модулей до конечного пользовательского опыта.

Тестирование охватывает несколько ключевых направлений. Оно включает функциональную проверку работы всех сценариев, интеграционные тесты между клиентом, сервером, базой данных и фоновыми задачами, а также нагрузочное моделирование, которое имитирует активную работу десятков и сотен пользователей. Такой подход позволяет проверить, как система справляется с параллельными запросами, корректно ли выполняет фильтрацию, обновление базы и генерацию карточек объявлений при повышенной нагрузке.

Особое внимание уделяется устойчивости к сбоям: система должна корректно реагировать на недоступность внешних источников, ошибки парсинга, перегрузку очередей или отказ одного из компонентов микросервисной архитектуры. Применяемые методики тестирования основаны на сочетании классических подходов (модульное, системное и пользовательское тестирование) и современных практик проверки асинхронных распределённых систем.

Завершающим этапом становится апробация — серия проверок, основанных на реальных пользовательских сценариях. В этом режиме система тестируется так, как если бы с ней взаимодействовал обычный арендатор: пользователь выбирает регион, настраивает фильтры, просматривает предложения, сохраняет понравившиеся объекты. Такой подход позволяет выявить не только технические недочёты, но и возможные проблемы в логике интерфейса, последовательности действий и удобстве работы с сервисом. В результате система проходит финальную проверку на соответствие тем задачам, ради которых она была спроектирована.

# 4.1 Методологии тестирования и оценки эффективности веб-сервиса для анализа и сравнения условий аренды недвижимости

Методология тестирования веб-сервиса была выстроена с учётом специфики его распределённой архитектуры. В фокусе находились ключевые компоненты системы: клиентская часть на базе Next.js, серверная логика, реализованная с помощью FastAPI, система асинхронной обработки задач Celery с брокером RabbitMQ, а также используемые хранилища данных — PostgreSQL и Redis. Тестирование охватывало как отдельные модули, так и их взаимодействие в рамках единой логики, с учётом сценариев реального использования.

В рамках функционального тестирования проверялась корректность выполнения всех основных пользовательских действий. Тесты включали фильтрацию по цене, площади, этажу и другим параметрам, отображение карточек с объявлениями, переход по внешним ссылкам к источникам, сохранение в избранное, а также ведение истории запросов. Каждая функция сопоставлялась с техническими требованиями и спецификацией, что позволяло выявить отклонения от ожидаемого поведения.

Интеграционное тестирование было направлено на проверку согласованности работы всех компонентов системы. В центре внимания находился полный цикл запроса: от ввода фильтров пользователем — до отображения обработанных результатов на клиентской стороне. Тестировались все этапы: формирование запроса, передача задачи в очередь, выполнение фоновой обработки, обращение к базе данных и возврат информации через FastAPI. Целью являлась оценка стабильности цепочки обработки и отсутствие потерь или искажений данных на любом из этапов.

Во время нагрузочного тестирования оценивалась способность сервиса сохранять производительность при интенсивной одновременной активности. Сценарии включали моделирование работы нескольких сотен пользователей, отправляющих параллельные запросы, изменяющих фильтры или обновляющих интерфейс. Использовались инструменты Locust и Apache Bench, которые позволяли анализировать время отклика, пропускную способность и устойчивость системы к пиковым нагрузкам.

Отдельное внимание уделялось тестированию парсеров, так как они являются точкой взаимодействия с внешними нестабильными источниками. Проверки включали обработку ошибок подключения, изменение структуры HTML-документов, появление HTTP-кодов 403, 404 и 500, а также ситуации с частично утерянными или некорректными данными. Важно было убедиться, что в подобных случаях система не выходит из строя: задача автоматически повторяется, ошибки логируются, а интерфейс остаётся стабильным для пользователя.

Вся тестовая работа проводилась в контролируемой среде, с использованием mock-данных, отдельной копии базы PostgreSQL и конфигурации Redis, настроенной для тестирования. Также были задействованы встроенные инструменты мониторинга, включая Celery Events и middleware FastAPI, что позволяло отслеживать статус задач, логировать действия и получать полную картину работы каждого из компонентов в режиме реального времени.

# 4.2 Тестирование веб-сервиса для анализа и сравнения условий аренды недвижимости

Тестирование веб-сервиса проводилось в специально развёрнутой среде, максимально приближённой к продуктивной конфигурации. Инфраструктура включала все ключевые компоненты архитектуры: клиентскую часть на фреймворке Next.js, обеспечивающую интерактивный пользовательский интерфейс; серверную часть на FastAPI, отвечающую за обработку запросов и реализацию бизнес-логики; систему асинхронных задач, построенную на связке Celery и RabbitMQ; а также две базы данных — основную на PostgreSQL и вспомогательное хранилище Redis, используемое для кеширования и очередей.

Тестирование охватывало как ручные проверки через веб-интерфейс, так и автоматизированные сценарии, реализованные с использованием тестовых скриптов, эмуляторов нагрузки и фреймворков автотестирования. Такой подход позволил проверить поведение сервиса как с позиции конечного пользователя, так и на уровне внутренних взаимодействий между микросервисами и модулями хранения данных.

В ходе функционального тестирования были последовательно проверены все основные пользовательские действия. Особое внимание уделялось работе системы фильтрации по множеству параметров: цена, этаж, площадь, наличие бытовой техники, условия проживания с детьми или животными. Проверялось корректное отображение карточек объявлений, их визуальная целостность и соответствие логике фильтра. Отдельно тестировались переходы по внешним ссылкам — каждое объявление должно вести к исходному источнику (например, Авито или ЦИАН), без ошибок и задержек. Интерфейс также проверялся на адекватное поведение при отсутствии подходящих результатов: в таких случаях выводилось информационное сообщение, предлагающее изменить параметры поиска. Тестировалась стабильность навигации в режиме SPA (Single Page Application), что включало переключение фильтров без полной перезагрузки страницы и минимизацию лишней перерисовки компонентов.

Интеграционные проверки подтвердили стабильность взаимодействия между всеми основными слоями сервиса. Были протестированы сценарии, включающие передачу данных от frontend к API, постановку задач в очередь Celery, выполнение фоновой обработки, запись результатов в базу PostgreSQL и возвращение ответа на клиентскую сторону. Проверки также охватывали случаи отказа отдельных компонентов — например, недоступность Redis, ошибки в конфигурации очередей или прерывание соединения с базой. Во всех этих ситуациях система демонстрировала устойчивость: ошибки логировались, а интерфейс продолжал функционировать без полной деградации.

Нагрузочное тестирование показало, что сервис способен обрабатывать более 70 одновременных пользовательских запросов без потери производительности. Среднее время отклика API при такой нагрузке не превышало 300 мс, что соответствует современным требованиям к быстродействию интерактивных веб-приложений. В процессе симуляции интенсивной работы не было зафиксировано потерь данных, ошибок записи в базу или сбоя пользовательского интерфейса.

Тестирование парсеров проводилось отдельно, с акцентом на устойчивость к нестабильным внешним условиям. Эмуляция ошибок загрузки, изменения HTML-структуры сайтов и нестандартных ответов (403, 404, 500) позволила убедиться в том, что парсеры корректно реагируют на сбои. При возникновении ошибок система автоматически повторяла задачу с другими параметрами — например, новым прокси или User-Agent. Все неудачные попытки фиксировались в логах, а в пользовательском интерфейсе не отображались некорректные или неполные карточки.

# 4.3 Оценка эффективности веб-сервиса для анализа и сравнения условий аренды недвижимости

Скорость отклика при работе с фильтрами остаётся одним из ключевых показателей производительности веб-сервиса. В ходе тестирования было зафиксировано, что среднее время от момента отправки фильтрационного запроса до отображения карточек с результатами составляет около 1,4 секунды. Такой показатель сохранялся стабильным даже при сложных комбинациях фильтров и параллельной активности нескольких пользователей. Это означает, что система способна своевременно обрабатывать пользовательские действия и обеспечивать быстрый доступ к информации без ощутимых задержек.

Минимальное количество действий для получения результата также было подтверждено в процессе тестирования. Во всех типичных пользовательских сценариях — будь то поиск квартиры по цене, этажу или условиям проживания — пользователи достигали целевого результата в среднем за три шага: выбор фильтра, запуск поиска и просмотр объявления. Это говорит о высокой логической прозрачности интерфейса и отсутствии лишних этапов, мешающих принятию решения.

Одним из приоритетов при проектировании логики сервиса стала эффективная дедупликация данных, получаемых с разных внешних платформ. Тестирование показало, что реализованный алгоритм удаления дубликатов достигает точности выше 96%. Он сравнивает объявления по адресу, этажу и цене, учитывая заданную допустимую погрешность. Это позволяет отсечь повторы и отображать пользователю только уникальные, релевантные предложения, исключая визуальные и логические повторы на экране.

Важным фактором остаётся полнота информации, представленной в карточках объектов. В результате анализа было выявлено, что 98% карточек содержат все необходимые данные, включая адрес, цену, площадь, этаж, наличие мебели и техники, а также условия проживания. Благодаря этому пользователь может сразу оценить привлекательность объекта, не переходя на сторонние сайты и не тратя время на ручное сравнение.

Скорость и надёжность переходов по внешним ссылкам также прошли отдельную проверку. Каждая карточка содержит ссылку на оригинальный источник объявления, откуда были получены данные. Среднее время открытия внешнего ресурса не превышало 0,3 секунды, а процент ошибок при переходе оказался нулевым. Это свидетельствует о корректной работе парсеров и своевременном обновлении ссылок, что обеспечивает уверенность пользователя в доступности информации.

По результатам пользовательского тестирования была оценена субъективная удовлетворённость интерфейсом. Участники отметили логичную структуру карточек, лёгкость применения фильтров и отсутствие визуального перегруза. В результате общая оценка удобства составила 4,7 из 5 возможных баллов, что подтверждает успешную реализацию принципов UX-дизайна и ориентацию на реальные сценарии использования.

Что касается технической надёжности, то в процессе тестирования был зафиксирован низкий уровень ошибок в пользовательском потоке — менее 1,5% от общего числа действий. Все такие случаи автоматически регистрировались системой логирования и не влияли на общую работоспособность сервиса. Пользователь не сталкивался с системными сбоями, а ошибки устранялись либо перезапуском задачи, либо заменой источника данных.

В рамках оценки эффективности была задействована автоматизированная система сбора метрик, охватывающая как поведенческие, так и технические параметры. В процессе анализа использовались логи действий пользователей, встроенные таймеры во frontend-компонентах, журналы FastAPI и Celery, а также данные, зафиксированные в PostgreSQL и Redis. Дополнительно проводились анонимные опросы среди тестирующих, что позволило получить качественную обратную связь и оценить восприятие системы со стороны конечных пользователей.

# 4.4 Анализ результатов тестирования и оценки эффективности веб-сервиса для анализа и сравнения условий аренды недвижимости

Надёжность архитектурного решения подтвердилось в ходе многократного тестирования под различной нагрузкой. Микросервисная организация позволила обеспечить гибкую и устойчивую работу всех компонентов системы. При отключении или ошибке в работе отдельных сервисов — таких как модуль фоновой обработки задач Celery или парсеры — остальная часть системы продолжала функционировать в штатном режиме. Серверная логика, реализованная с использованием FastAPI, демонстрировала стабильную работу даже при параллельном обращении к базе данных PostgreSQL и взаимодействии с Redis в рамках выполнения фоновых операций. Это доказывает, что архитектура не только масштабируема, но и устойчива к сбоям на отдельных участках.

Логика фильтрации и алгоритмы сравнения показали высокую точность и соответствие заявленным требованиям. При тестировании с использованием различных комбинаций фильтров система корректно возвращала релевантные объекты, соответствующие заданным параметрам пользователя. Алгоритм сравнения, использующий нормализацию данных и расчёт расстояния до эталонной модели, позволял ранжировать объекты по степени соответствия предпочтениям арендатора. Встроенный механизм дедупликации, сравнивающий объекты по адресу, этажу и цене, продемонстрировал точность выше 96%, эффективно устраняя дубли при агрегации объявлений с разных источников. Это делает выдачу более чистой и достоверной.

Производительность и удобство пользовательского интерфейса были оценены как высокие. Интерфейс системы обеспечивал мгновенную реакцию на действия пользователя — будь то применение фильтров, переключение между карточками или возврат к предыдущим результатам. Визуальные элементы чётко структурированы, все управляющие компоненты логично сгруппированы, что упрощает навигацию. Интерфейс корректно отрабатывал как стандартные сценарии, так и пограничные случаи (например, отсутствие результатов). Визуальное оформление, согласованное с принципами юзабилити Нильсена, обеспечило положительное восприятие интерфейса — по результатам пользовательских тестов оценка удобства составила 4,7 из 5 баллов.

Работа с внешними источниками данных также показала устойчивость системы при непредсказуемом поведении сайтов-агрегаторов. Во время тестов моделировались ситуации с HTTP-ошибками 403, 404, а также изменениями HTML-структуры страниц и временной недоступностью источников. В каждом из этих случаев система корректно обрабатывала сбои: соответствующие задачи автоматически перезапускались с использованием альтернативных прокси и заголовков, ошибки логировались в таблице parser\_logs без нарушения основного потока. Благодаря этому пользователи не сталкивались с недействительными или пустыми объявлениями — в выдаче отображались только корректные, верифицированные карточки объектов.

# Выводы

Проведённый комплексный анализ подтвердил, что веб-сервис полностью соответствует задачам, поставленным в рамках дипломного проекта, и готов к полноценному применению в реальных условиях. Система показала устойчивость при различных нагрузках, корректную работу всех компонентов и высокую степень соответствия функциональных возможностей ожиданиям пользователей. Проект успешно реализует цель — предоставить пользователю удобный, точный и надёжный инструмент для анализа и сравнения условий аренды квартир.

Архитектура сервиса продемонстрировала устойчивость к частичным отказам и возможность масштабирования. Микросервисный подход обеспечивает независимость компонентов: при временном выходе из строя одного из элементов (например, парсера или Celery-задачи) остальная часть системы продолжает функционировать. API на FastAPI стабильно обрабатывает параллельные запросы, не возникает конфликтов при одновременном доступе к PostgreSQL и Redis, а очередь задач работает корректно без потерь данных.

Логика фильтрации и сравнения объектов отработала без сбоев. Фильтры возвращают результаты, соответствующие заданным критериям, даже при сложных пользовательских комбинациях. Механизм сравнения, основанный на нормализации параметров и вычислении отклонений от эталонной модели, работает стабильно. Дедупликация помогает сократить количество повторов и устраняет объявления-дубли с разных платформ, повышая точность и прозрачность выдачи.

Интерфейс сервиса показал высокий уровень удобства. Навигация и работа с фильтрами не вызывают затруднений, интерфейс остаётся отзывчивым и интуитивно понятным. Пользователи быстро осваиваются в системе, а визуальная структура карточек и фильтров делает процесс выбора комфортным. Интерфейс демонстрирует быструю реакцию и не требует ручной перезагрузки данных при изменении параметров. Отдельно стоит отметить положительную оценку интерфейса в ходе тестирования — пользователи отметили простоту, логичность и скорость.

Интеграция с внешними источниками данных показала надёжность даже в нестабильных условиях. При возникновении ошибок, связанных с недоступностью ресурса, изменением DOM-структуры или возвратом статуса 403/404, система продолжала работу. Все сбои автоматически фиксировались и логировались, задачи повторялись с использованием новых прокси, а пользовательский поток оставался чистым от некорректных карточек. Парсеры обеспечивают регулярное обновление базы и возвращают актуальные объявления в соответствии с архитектурными требованиями.

Все ключевые критерии — надёжность, точность, стабильность и удобство — были успешно подтверждены в ходе оценки.

# Заключение

В рамках данной выпускной квалификационной работы была разработана и реализована программная система — веб-сервис для анализа и сравнения условий аренды квартир, предназначенный для упрощения процесса поиска жилья на рынке аренды. Основной целью проекта являлось сокращение временных затрат арендаторов за счёт автоматизации этапов сбора, фильтрации, сравнения и вывода предложений по ключевым параметрам.

Проведённый анализ предметной области позволил выявить значительное число проблем в существующих сервисах: устаревшие или неполные данные, дублирующиеся объявления, отсутствие глубокой фильтрации и ручной характер сравнения. Это обосновало необходимость создания решения с фокусом на автоматизированный парсинг и расширенные фильтры.

В процессе разработки был создан полноценный веб-сервис с микросервисной архитектурой, включающей клиентскую часть на Next.js, серверную логику на FastAPI, фоновую обработку через Celery, систему очередей RabbitMQ и базы данных PostgreSQL и Redis. Для парсинга данных с внешних платформ были реализованы устойчивые модули с системой защиты от блокировок и логированием ошибок.

Архитектура веб-сервиса обеспечивает масштабируемость, отказоустойчивость и лёгкую модификацию, что делает систему пригодной для дальнейшего развития: подключения новых источников, добавления пользовательских рекомендаций и расширения по регионам. Особое внимание уделено UX-дизайну: интерфейс интуитивен, быстр, соответствует 10 эвристикам Нильсена и был высоко оценён пользователями на этапе тестирования.

В ходе тестирования были проверены функциональные, интеграционные и нагрузочные характеристики системы. Зафиксирована высокая точность фильтрации и дедупликации (более 96%), стабильность работы при 70+ параллельных запросах, среднее время отклика API — менее 300 мс. Сервис демонстрирует устойчивую работу при сбоях внешних источников, своевременно обновляет базу и исключает недостоверные объявления.

Разработанное программное обеспечение полностью соответствует техническому заданию, успешно прошло апробацию и показало готовность к продуктивному использованию. Его внедрение способно существенно упростить процесс аренды жилья, повысить качество пользовательского опыта и сформировать основу для более интеллектуальных систем в сфере цифровой недвижимости.

# Список использованных источников

1. Морозова, Е. С. Технология создания виртуальных интерактивных туров / Е. С. Морозова, В. В. Лавров // Теплотехника и информатика в образовании, науке и производстве : сборник докладов I Всероссийской научно-практической конференции студентов, аспирантов и молодых ученых (TIM2012) с Международным участием / УрФУ [и др.] ; под ред. Н. А. Спирина.– Екатеринбург, 2012.– С. 245-247.
2. Что такое виртуальный тур? [Электронный ресурс]. —Режим доступа : https://3dturov.net (дата обращения 28.04.2022).

# Приложение А

# Справка о результатах проверки выпускной квалификационной работы на наличие заимствований

# Приложение Б

# Техническое задание

# Приложение В

# Руководство системного программиста