

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет
вычислительной техники

Кафедра
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

Направленность Программная инженерия

БАКАЛАВРСКАЯ РАБОТА

на тему

**Мобильное программное приложение для
поддержки процессов функционирования системы
мониторинга объявлений по продаже автомобилей**

Студент _____ Живов Сергей Витальевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Князев В.Н.
(подпись, дата) (фамилия, инициалы)

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

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

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

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

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

Пенза, 2026

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

Факультет
Вычислительной техники

Кафедра
МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____ А.Ю. Козлов
«26» декабря 2025 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Живов Сергей Витальевич гр. 22ВП1 факультета ВТ

<Фамилия, имя, отчество полностью>

2. Тема работы Мобильное программное приложение для поддержки процессов функционирования системы мониторинга объявлений по продаже автомобилей.

Тема утверждена приказом ПГУ № _____ от “___” _____ 20__

3. Руководитель работы к.т.н., доцент кафедры «МОиПЭВМ» Князев В.Н.

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

1) Разработать программное обеспечение для поддержки процессов функционирования системы мониторинга объявлений по продаже автомобилей

2) Среда выполнения приложения: операционная система Android 9 и выше

Среда выполнения сервера: Docker-контейнер

Языковые средства разработки: C#, UML, SQL

Инструментальные средства разработки: Rider, WebStorm, Visual Paradigm, Microsoft Project 2007

Технологии разработки: ASP.NET, React Native, PostgreSQL, MVC, MVVM, Docker

3) Основные функции мобильного приложения:

а) просмотр списка объявлений о продаже автомобилей;

б) добавление и удаление объявлений в список избранного;

в) добавление, удаление и редактирование информации о объявлениях в списке;

г) фильтрация каталога по параметрам транспортного средства;

д) сортировка каталога;

5. Перечень подлежащих разработке вопросов

1) Анализ предметной области и постановка задачи

2) Проектирование

3) Реализация

4) Контроль качества программного обеспечения

5) Оформление пояснительной записки

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1	Анализ предметной области и требований к системе	10%	07.04.2026 – 13.04.2026	
2	Проектирование программных средств	20%	14.04.2026 – 25.04.2026	
3	Реализация программных средств	30%	27.04.2026 – 11.05.2026	
4	Тестирование программных средств	20%	12.05.2026 – 21.05.2026	
5	Оформление пояснительной записки	20%	22.05.2026 – 02.06.2026	

Дата выдачи задания «26» декабря 2025г.

Руководитель бакалаврской работы _____
(подпись, дата)

Задание к исполнению принял студент _____
(подпись, дата)

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

Реферат

Пояснительная записка содержит 118 листов, 51 рисунок, 25 таблиц, 32 использованных источника, 5 приложений.

СИСТЕМА МОНИТОРИНГА ОБЪЯВЛЕНИЙ, ANDROID, IOS, POSTGRESQL, C#, .NET, UML, MVC, MVVM

Объектом разработки является мобильное приложение – система мониторинга объявлений для поиска и анализа объявлений по продаже автомобилей с разных сайтов, фильтрацию и отслеживание актуальных предложений на рынке.

Цель работы – разработка мобильного программного приложения для упрощения процессов поиска и мониторинга объявлений по продаже автомобилей, добавлять объявления в избранное, а также просматривать характеристики автомобиля в объявлении, не переходя на изначальный сайт.

Разработка серверной части происходила на языке программирования C#. В качестве среды разработки использовался Rider. В качестве СУБД для хранения данных была использована СУБД PostgreSQL.

Разработка клиентской части системы мониторинга объявлений производилась на фреймворке React Native на языках программирования JavaScript и TypeScript. В качестве среды разработки использовалась среда программирования WebStorm, операционная система Windows 10. Приложение предназначено для использования под управлением ОС Android версии 9.0 и выше и iOS, а его серверная часть работает под управлением контейнеров Docker.

					ПГУ 09.03.04 – 08БР221.09 ПЗ			
Изм.	Лист	№ докум.	Подп.	Дата	Мобильное программное приложение для поддержки процессов функционирования системы мониторинга объявлений по продаже автомобилей. Пояснительная записка	Лит.	Лист	Листов
Разраб.		Живов С.В.					4	118
Пров.		Князев В.Н.						
Н. контр.		Такташкин Д.В.				Группа 22ВП1		
Утверд.		Козлов А.Ю.						

Содержание

Введение.....	6
1 Анализ предметной области и требований.....	8
1.1 Анализ предметной области и постановка задачи	8
1.2 Анализ программных продуктов	8
1.2.1 Сервис «Autospot»	9
1.2.2 Сервис «Fresh»	9
1.2.3 Сервис «Автокод Поиск».....	10
1.2.4 Сравнение аналогов.....	11
1.3 Анализ целевой аудитории.....	12
1.4 Анализ функциональных требований	14
1.5 Анализ нефункциональных требований.....	22
1.6 Планирование разработки и оценка бюджета	23
2 Проектирование мобильного приложения	43
2.1 Проектирование архитектуры программного обеспечения	43
2.2 Проектирование базы данных	44
2.2.1 Концептуальное проектирование базы данных.....	44
2.2.2 Логическое проектирование базы данных	46
2.3 Проектирование серверной части приложения	47
2.4 Проектирование интерфейса пользователя	49
3 Реализация программного обеспечения.....	56
3.1 Реализация базы данных	56
3.2 Разработка серверной части приложения	59
3.3 Реализация клиентской части приложения.....	62
4 Тестирование программного продукта	65
4.1 Функциональное тестирование черным ящиком	65
4.2 Модульное тестирование	66
4.3 Метрики кода	67
4.3.1 Метрики размера программы	68
4.3.2 Метрики Холстеда.....	68
Заключение.....	72
Список использованных источников.....	74
Приложение А Глоссарий (справочное).....	78
Приложение Б Исходный код серверной части (обязательное)	82
Приложение Г Результаты тестирования серверной части (рекомендуемое).....	106
Приложение Д Исходный код функций тестирования (рекомендуемое)	112

Введение

Сегодня информация стала главным ресурсом развития общества. Она влияет на все сферы жизни человека и определяет то, как люди мыслят и действуют. XXI век считается временем активного развития цифровых технологий. По мнению ученых, в будущем этот процесс затронет экономику, социальную сферу и жизнь каждого человека. Уже сейчас цифровая среда плотно вошла как в работу людей, так и в их повседневные дела [1].

В России этот процесс поддерживается на государственном уровне. В мае 2017 года правительство утвердило программу «Цифровая экономика Российской Федерации». Главная идея программы – сделать цифровые данные основным фактором производства. Это должно повысить качество жизни граждан и сделать страну более конкурентоспособной. Тогда же был подписан Указ Президента «О стратегии развития информационного общества», который определил основные цели развития информационных технологий в стране до 2030 года [1, 2].

Важной особенностью нашего времени стало повсеместное использование мобильных телефонов с выходом в интернет. Мобильный интернет постепенно вытесняет доступ со стационарных компьютеров, потому что это удобнее. Цифры это подтверждают: за 2023 год в России зарегистрировано 227 миллионов мобильных подключений – это больше, чем население страны. На каждые 100 человек приходится более 164 мобильных телефонов, и эта тенденция только усиливается [3].

Рынок продажи автомобилей сегодня практически полностью переместился в интернет. Люди ищут машины на специальных сайтах. Но здесь возникает проблема: информации слишком много, цены постоянно меняются, а объявления появляются и исчезают. Если отслеживать всё вручную, переходя с сайта на сайт, уходит много времени.

Поскольку почти у всех есть смартфон, логично сделать инструмент для мониторинга именно в формате приложения. Оно сможет само собирать данные

с разных сайтов, объединять их в одном месте и показывать пользователю удобный каталог с фильтрами.

Главное преимущество мобильного телефона перед компьютером – это push-уведомления. Приложение может само сообщить пользователю, что появилось подходящее предложение или упала цена на нужную модель. Не нужно сидеть и обновлять страницы вручную. Для пользователей это экономит время и позволяет искать машины с телефона, не обходя при этом десятки сайтов или ездя по автосалонам – всё можно посмотреть в приложении за пару минут.

Написанное выше позволяет заключить, что в условиях роста мобильного интернет-трафика и высокой динамики автомобильного рынка особенно востребованным становится инструмент, сочетающий автоматизированный сбор данных с нескольких площадок и информирование пользователя. В связи с этим актуальной задачей является разработка мобильного приложения для мониторинга цен и предложений на автомобили, чему и посвящена данная работа.

Перспективность разработки мобильного приложения для мониторинга автомобильных объявлений обусловлена несколькими факторами. Во-первых, рынок подержанных автомобилей в России продолжает расти: по данным аналитического агентства «Автостат», в 2025 году объём вторичного рынка превысил 6 миллионов сделок, что формирует устойчивый спрос на инструменты автоматизированного поиска. Во-вторых, развитие технологий веб-скрейпинга и обработки данных позволяет создавать всё более точные и быстрые системы агрегации информации из разнородных источников. В-третьих, тенденция к приложениям, объединяющим множество сервисов в едином интерфейсе, свидетельствует о востребованности решений, избавляющих пользователя от необходимости переключаться между несколькими платформами. Таким образом, разрабатываемое приложение имеет значительный потенциал для масштабирования и коммерциализации.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Разрабатываемое мобильное приложение должно обеспечивать пользователю удобный и интуитивно понятный интерфейс для сравнения объявлений с разных сайтов. С помощью разрабатываемого приложения пользователи могут в одном приложении просматривать объявления с разных сайтов. Также, пользователи имеют возможность отсортировать и отфильтровать каталог объявлений для поиска нужных объявлений. Пользователи могут добавить объявления в избранное, чтобы потом вернуться к объявлениям, без повторного поиска в каталоге.

Таким образом, в приложении должны быть представлены следующие основные функции:

- регистрация/вход пользователя;
- просмотр каталога объявлений;
- сортировка каталога объявлений;
- фильтрация каталога объявлений;
- поиск объявления в каталоге;
- добавление объявления в список понравившегося;
- перейти на сайт объявлений.

Помимо обычных пользователей в системе будет присутствовать администратор. Он имеет следующие функции: добавить объявление в каталог, удалить объявление из каталога, редактировать информацию, получаемую с объявления, в каталоге.

1.2 Анализ программных продуктов

В настоящее время существует различные сервисы по подбору автомобилей. Рассмотрим наиболее известные из них и проведем их сравнительный анализ.

1.2.1 Сервис «Autospot»

«Autospot» – это агрегатор предложений, специализирующийся на автомобилях от официальных дилеров. Платформа позволяет сравнивать цены и условия покупки новых машин и автомобилей с пробегом, а также знакомиться с текущими скидками и акциями дилерских центров [4].

Главная страница «Autospot» представлена на рисунке 1.

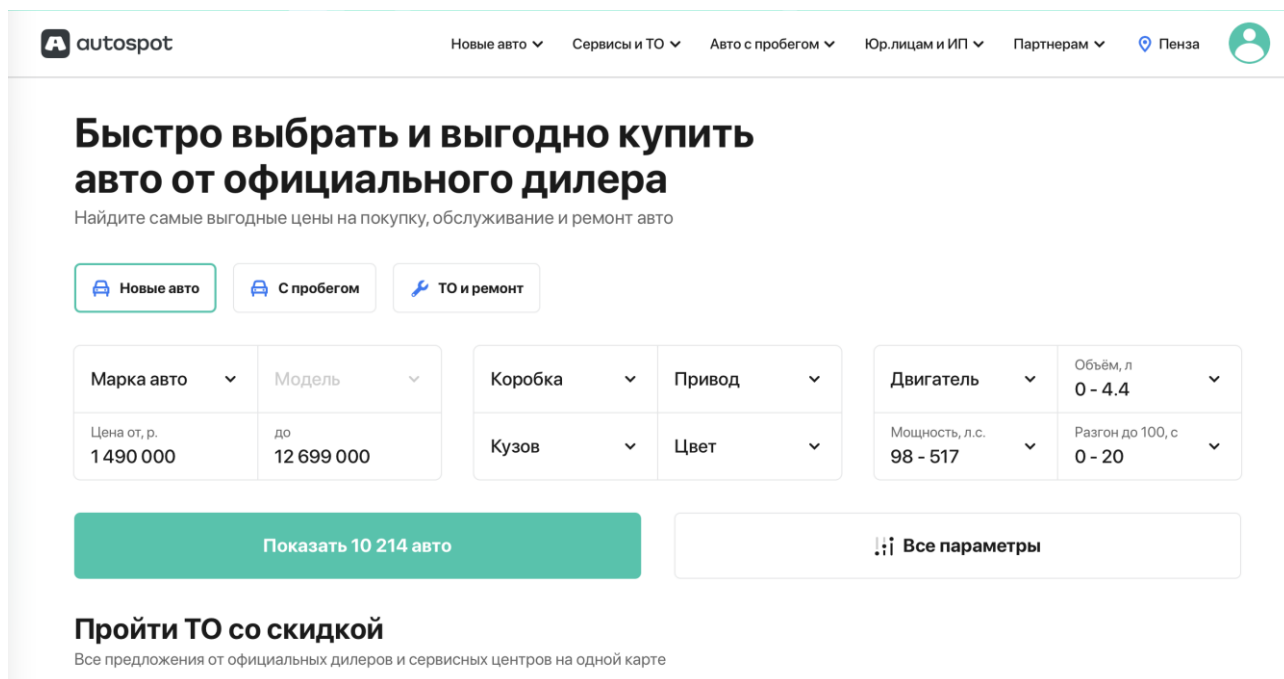


Рисунок 1 – Страница мобильного приложения «Autospot»

К достоинствам относится удобство подбора машины у проверенных дилеров с возможностью онлайн-сравнения их предложений.

К недостаткам данного приложения относится: ориентация на дилерские центры и полное отсутствие частных продавцов, неполная база автомобилей.

1.2.2 Сервис «Fresh»

«Fresh» – это автомобильный маркетплейс, на котором представлены объявления о продаже новых и подержанных автомобилей как от дилеров, так и от частных лиц. Платформа предлагает расширенную фильтрацию, инструменты онлайн-оценки автомобиля, а также встроенные сервисы кредитования и страхования [5].

Главная страница «Fresh» представлена на рисунке 2.

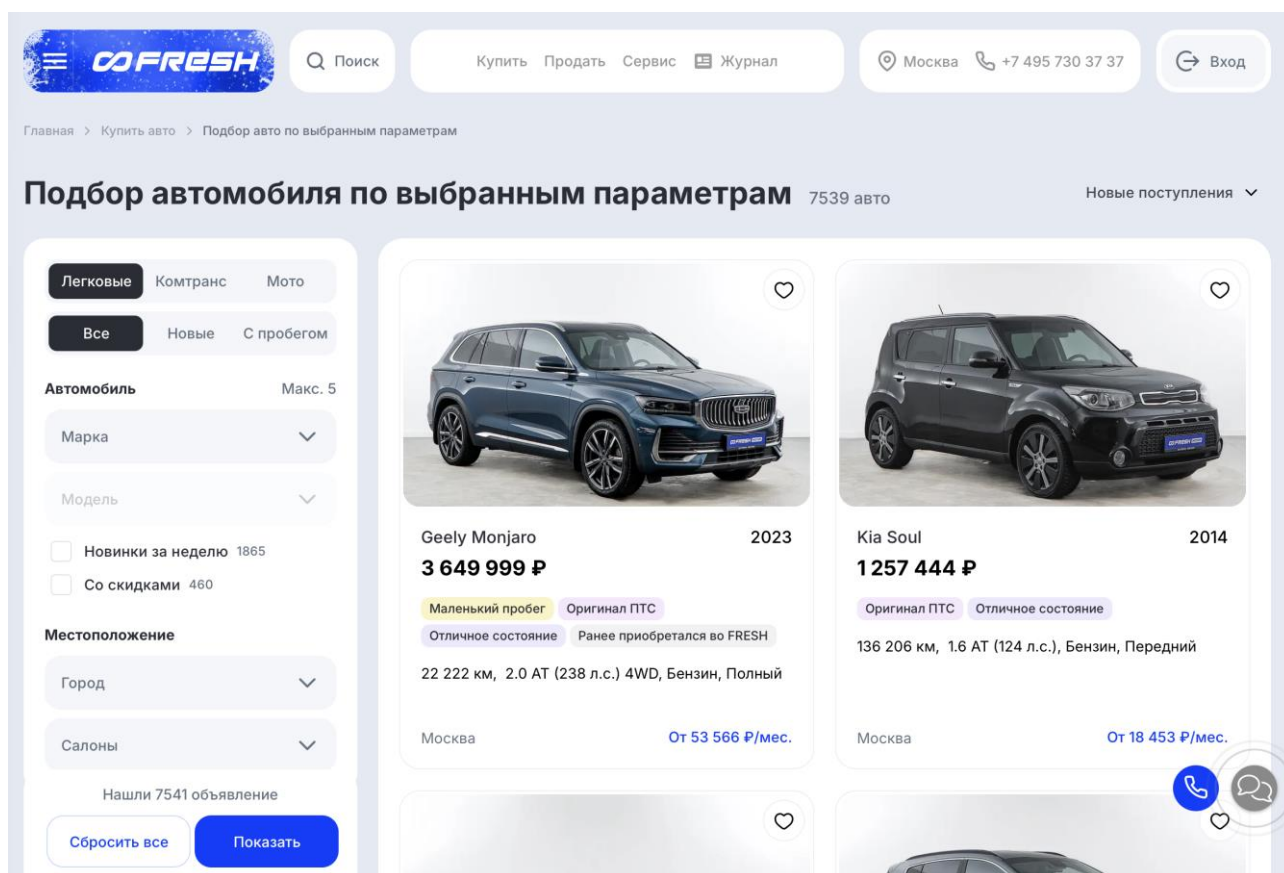


Рисунок 2 – Главная страница сервиса «Fresh»

К достоинствам можно отнести наличие дополнительных финансовых сервисов в одном окне и возможность записи на диагностику через платформу.

К недостаткам данного приложения относится: малое количество актуальных предложений, неудобная навигация, отзывчивость интерфейса.

1.2.3 Сервис «Автокод Поиск»

«Автокод Поиск» – это агрегатор, который собирает объявления о продаже автомобилей с российских онлайн-площадок. Сервис позволяет отслеживать рыночные цены и получать уведомления о новых предложениях [6].

Главная страница приложения «Автокод Поиск» представлена на рисунке 3.

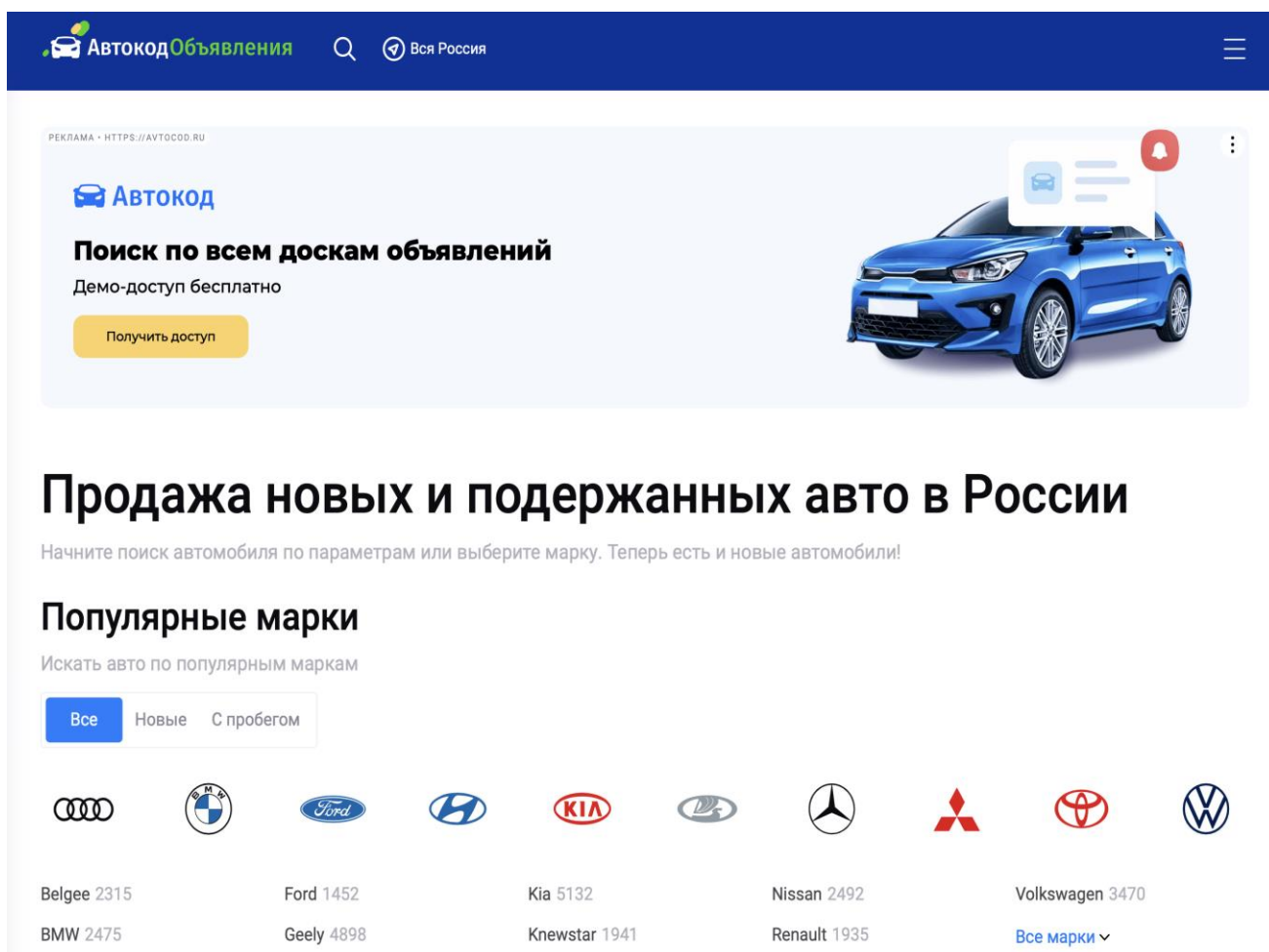


Рисунок 3 – Главная страница «Автокод Поиск»

К достоинствам приложения относится возможность поиска автомобилей сразу по крупным площадкам и уведомления о свежих объявлениях и изменении цены.

К недостаткам данного приложения относится то, что бесплатный доступ ограничен 7 днями, наличие навязчивой рекламы других услуг сервиса.

1.2.4 Сравнение аналогов

На данном этапе был произведен сравнительный анализ рассмотренных ранее веб-сайтов, а также мобильного приложения. В результате анализа полученных данных, с учетом выявленных сходств и различий, была сформирована итоговая таблица 1, которая наглядно демонстрирует ключевые параметры сравнения.

Таблица 1 – Сравнение аналогов

Название сервиса	Источники объявлений	Наличие частных продавцов	Доступ к функционалу	Удобство интерфейса	Наличие мобильного приложения
Autospot	Дилерская сеть	Нет	Бесплатный	Интуитивно-понятный	Есть
Fresh	Собственная база	Есть	Бесплатный	Неудобная навигация	Нет
Автокод поиск	Агрегатор различных площадок	Есть	Платный доступ	Неудобная всплывающая навязчивая реклама	Нет
Разрабатываемое приложение	Агрегатор всех основных площадок	Есть	Массовый пользователь	Интуитивно-понятный	Есть

Таким образом, в результате анализа систем аналогичного назначения было принято решение о разработке собственного приложения, которое будет лишено недостатков, выявленных у рассмотренных выше аналогов.

1.3 Анализ целевой аудитории

Основной целевой аудиторией являются покупатели автомобилей. Согласно данным аналитического агентства «Автостат» и профильных изданий, российский автомобильный рынок в 2025–2026 годах демонстрирует признаки уверенной стабилизации и перехода к осторожному росту, несмотря на сложные макроэкономические условия. В начале 2026 года зафиксирована позитивная динамика: на первой неделе февраля продажи легковых автомобилей выросли на 2,2% по сравнению с предыдущей неделей, почти достигнув уровня прошлого года, а сегмент легкого коммерческого транспорта прибавил сразу 9%. Этот импульс стал продолжением наметившегося во второй половине 2025 года тренда. Таким образом, интерес целевой аудитории к приобретению автомобилей сохраняется [7, 8].

Для определения платформ разработки мобильного приложения проведён анализ распространённости мобильных операционных систем. На рисунке 4 представлены доли мобильных ОС на мировом рынке по данным аналитического сервиса Statcounter [9].

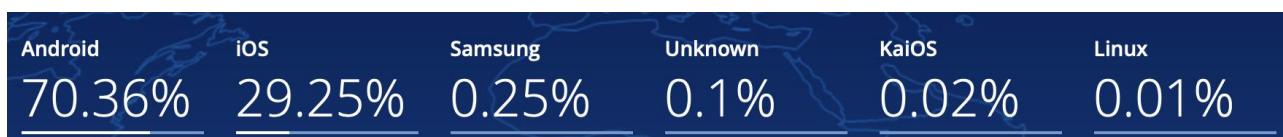


Рисунок 4 – Доли мобильных операционных систем в мировом рынке

Проанализировав рынок, можно сделать вывод, что большинство людей используют операционную систему Android, вторым по численности является операционная система iOS [9].

На рисунке 5 представлены доли мобильных операционных систем на российском рынке [10].

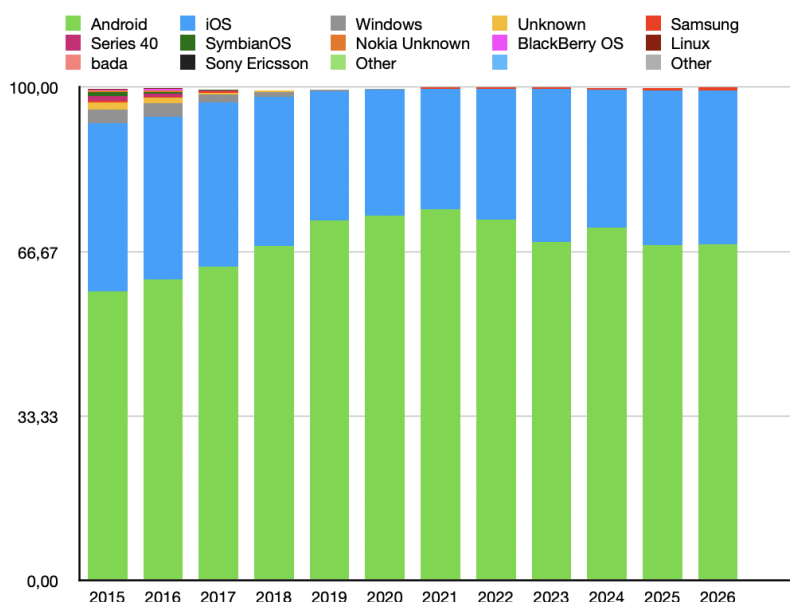


Рисунок 5 – Доли мобильных операционных систем на российском рынке

Проанализировав российский рынок мобильных операционных систем, можно сделать вывод, что несмотря на то, что с 2021 года доля операционной системы Android постепенно падает, данная операционная система занимает лидирующие места [9].

Также, в России существуют свои мобильные операционные системы. По состоянию на 2025 год рынок отечественного мобильного ПО окончательно сформировался, и самыми популярными из них являются:

- «Astra Linux Mobile» – бесспорный лидер среди российских аналогов Android, главной особенностью которого остается высокая безопасность и

полная изоляция от западных сервисов, благодаря чему ОС активно внедряется в силовых структурах и на объектах КИИ;

- «Аврора» – российская ОС, известная своей высокой безопасностью и конфиденциальностью, является основным инструментом для корпоративного сектора и госкомпаний;

- «Ред ОС М» – молодая, но уверенно набирающая обороты российская мобильная операционная система на базе Linux, позиционируясь как гибкое и безопасное решение для импортозамещения.

На рисунке 6 продемонстрирована статистика распространенных версий Android.

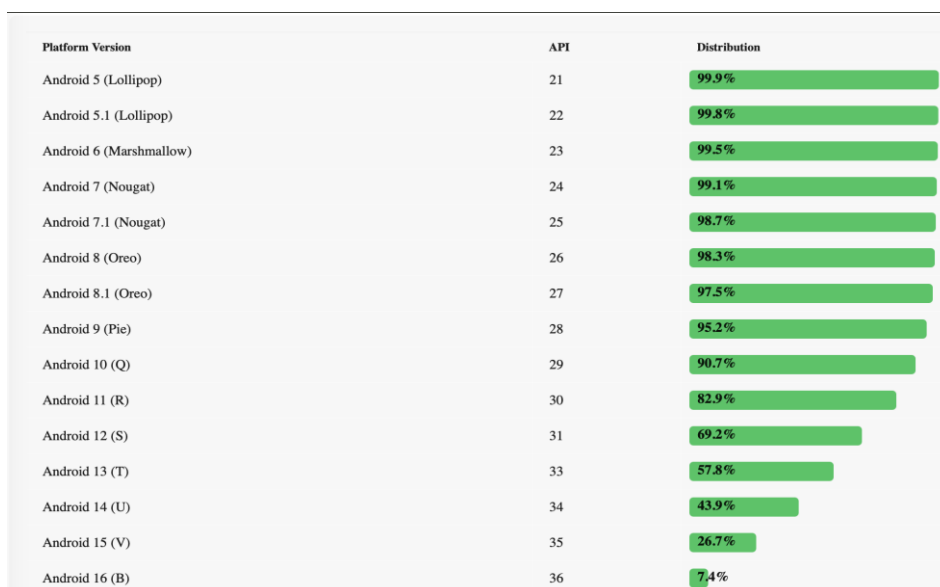


Рисунок 6 – Статистика распространённых версий Android

На основе данных статистики можно сделать вывод, что версия Android 9 довольно распространена среди пользователей. Поэтому мобильное приложение будет разрабатываться на операционной системе Android, таким образом, чтобы оно функционировало на версии Android 9 и выше.

1.4 Анализ функциональных требований

Основными пользователями нашего приложения выступают: пользователь и администратор. Пользователь, работая с мобильным приложением, должен

иметь возможность просмотра, поиска, сортировки и фильтрации каталога. Добавлять объявления в избранное для последующего сравнения или отслеживания.

Администратор приложения, работая с мобильным приложением, должен иметь возможности пользователя, а также он может осуществлять добавление, удаление и редактирование объявлений из каталога.

Также все пользователи имеют возможность регистрации и аутентификации в системе, для доступа к функционалу приложения.

На основе всего вышесказанного была создана диаграмма вариантов использования мобильного приложения

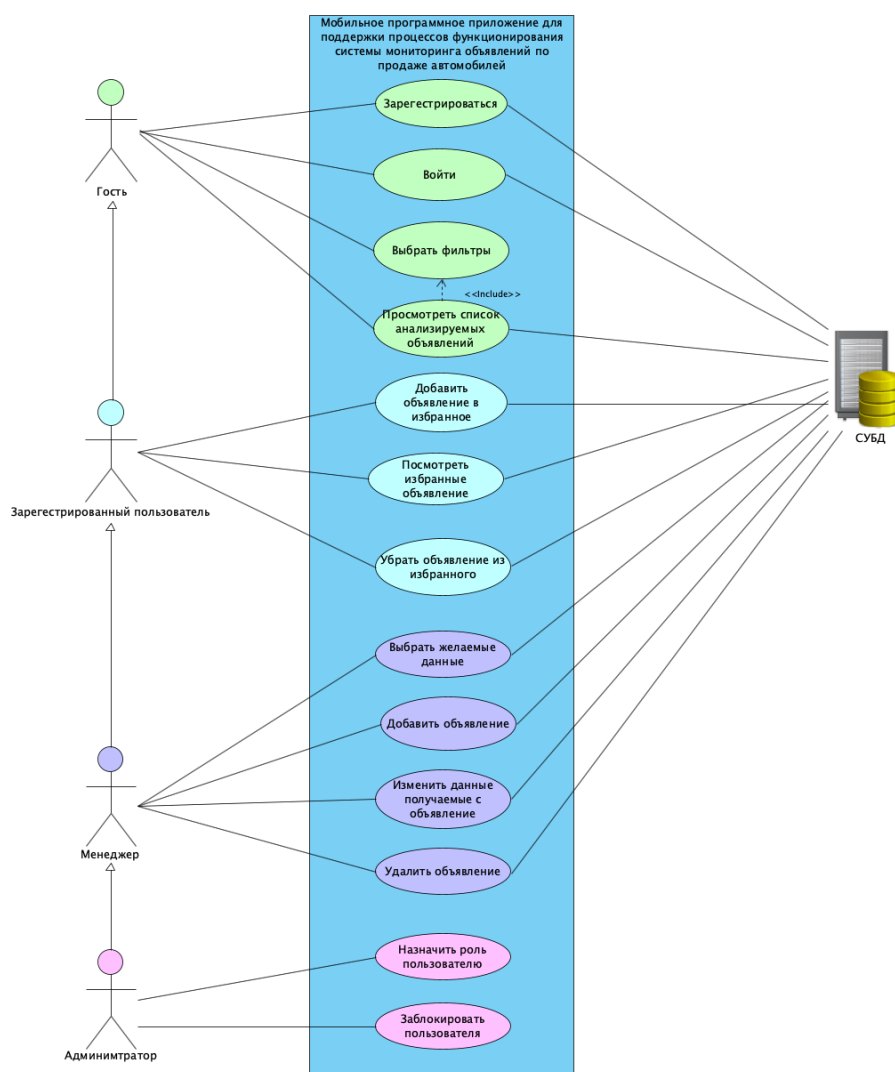


Рисунок 7 – Диаграмма вариантов использования мобильного приложения

На основе диаграммы вариантов использования составим спецификации для основных вариантов использования: «Зарегистрироваться» (рисунок 8), «Авторизоваться» (рисунок 9), «Добавить объявление в избранное» (рисунок 10).

Параметр	Значение
Наименование	Зарегистрироваться
ID	1
Краткое описание	Гость вводит данные для регистрации, система создаёт новую учётную запись
Действующие лица	Гость
Предусловия	Пользователь не авторизован
Основной поток	1. Открытие формы регистрации 2. Ввод логина и пароля 3. Точка расширения «Проверить существования логина» 4. Точка расширения «Добавить пользователя» 5. Подтверждение успешной регистрации
Постусловия	Создана новая учётная запись пользователя

Рисунок 8 – Спецификация варианта использования «Зарегистрироваться»

Наименование	Авторизоваться
ID	2
Краткое описание	Зарегистрированный пользователь входит в систему под своей учётной записью
Действующие лица	Гость
Предусловия	1. Пользователь не авторизован 2. Учётная запись существует
Основной поток	1. Открытие формы авторизации 2. Ввод логина и пароля 3. Точка расширения «Получить пользователя по логину» 4. Точка расширения «Защитить пароль пользователя» 5. Проверить корректность данных 6. Если данные верны 6.1. Открытие сеанса работы пользователя 7. Иначе вывести сообщение о некорректности данных
Постусловия	Пользователь авторизован и имеет доступ к функционалу зарегистрированных пользователей

Рисунок 9 – Спецификация варианта использования «Авторизоваться»

Наименование	Добавить объявление в избранное
ID	3
Краткое описание	Авторизованный пользователь добавляет интересующий его объявление в список избранных
Действующие лица	Зарегистрированный пользователь
Предусловия	Пользователь авторизован
Основной поток	1. Точка расширения «Просмотреть список анализируемых объявлений» 2. Пользователь выбирает объявление из списка 3. Пользователь инициирует добавление в избранное 4. Точка расширения «Добавить избранное объявление» 5. Вывод сообщения о добавлении
Постусловия	Выбранное объявление появляется в личном списке избранного пользователя

Рисунок 10 – Спецификация варианта использования «Добавить объявление в избранное»

Также, на основе диаграммы вариантов использования, составим спецификации для основных вариантов использования. На рисунках были представлены спецификации «Посмотреть избранные объявления» (рисунок 11) и «Добавить объявление» (рисунок 12).

Наименование	Посмотреть избранные объявления
ID	4
Краткое описание	Авторизованный пользователь просматривает список объявлений, которые он ранее добавил в избранное
Действующие лица	Зарегистрированный пользователь
Предусловия	Пользователь авторизован
Основной поток	1. Пользователь переходит в раздел «Избранное» 2. Точка расширения «Получить список избранных объявлений для конкретного пользователя» 3. Точка расширения «Настроить фильтры и/или сортировки» 4. Отображение перечня объявлений с краткой информацией
Постусловия	Пользователь видит список своих избранных объявлений

Рисунок 11 – Спецификация варианта использования «Посмотреть избранные объявления»

Наименование	Добавить объявление
ID	5
Краткое описание	Администратор добавляет новое объявление в систему для последующего анализа объявлений
Действующие лица	Администратор
Предусловия	Администратор авторизован
Основной поток	1. Администратор вводит URL сайта 2. Точка расширения «Получить страницу выбранного объявления» 3. Точка расширения «Выбрать желаемые данные» 4. Точка расширения «Добавить анализируемое объявление» 5. Подтверждение успешного добавления
Постусловия	Новое объявление добавлено в список анализируемых, настроены параметры сбора данных

Рисунок 12 – Спецификация варианта использования «Добавить объявление»

Рассмотрим некоторые варианты использования подробнее. Для вариантов использования «Добавить объявление» и «Добавить объявление в избранное» составлены диаграммы деятельности. Вариант использования добавления объявления администратором в каталог представлен на рисунке 13, вариант использования добавления объявления в избранное авторизованным пользователем представлен на рисунке 14 [11, 13].

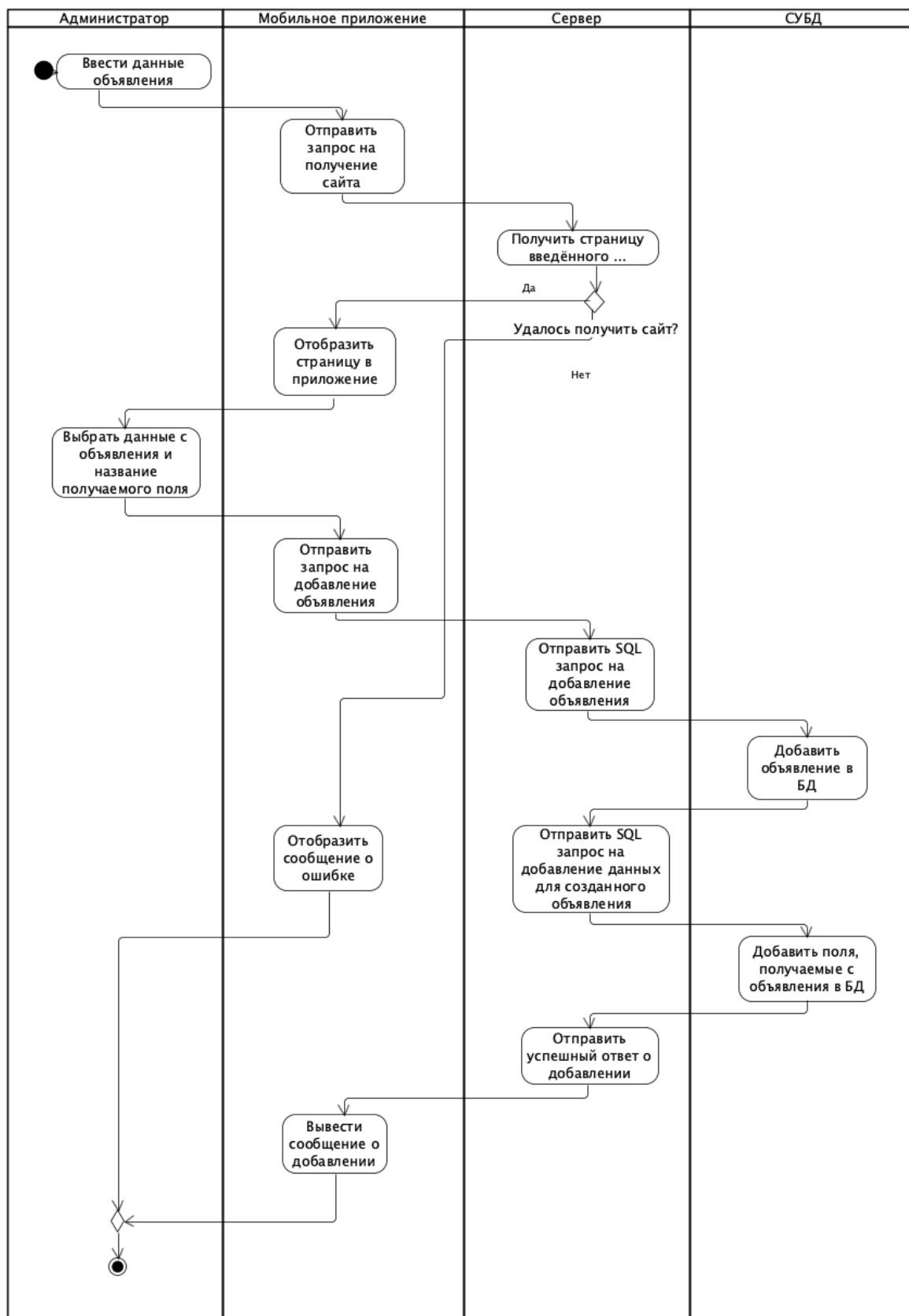


Рисунок 13 – Диаграмма деятельности варианта использования «Добавить объявление»

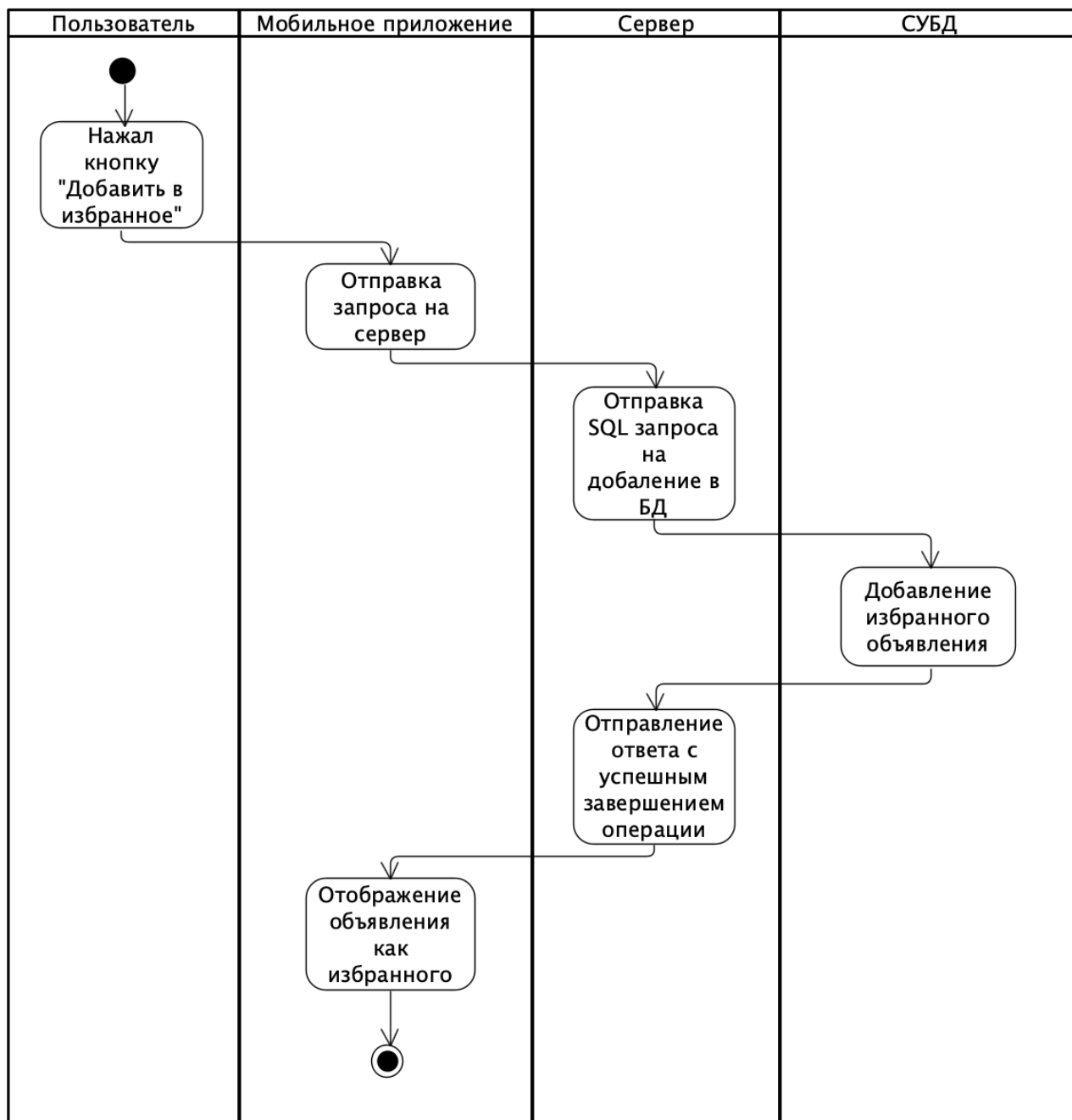


Рисунок 14 – Диаграмма деятельности варианта использования «Добавить объявление в избранное»

Также для варианта использования «Посмотреть список анализируемых объявлений» и для варианта использования «Авторизоваться» были составлены диаграммы последовательности. Диаграмма последовательности просмотра списка анализируемых объявлений была представлена на рисунке 15, диаграмма последовательности авторизации представлена на рисунке 16 [11, 14].

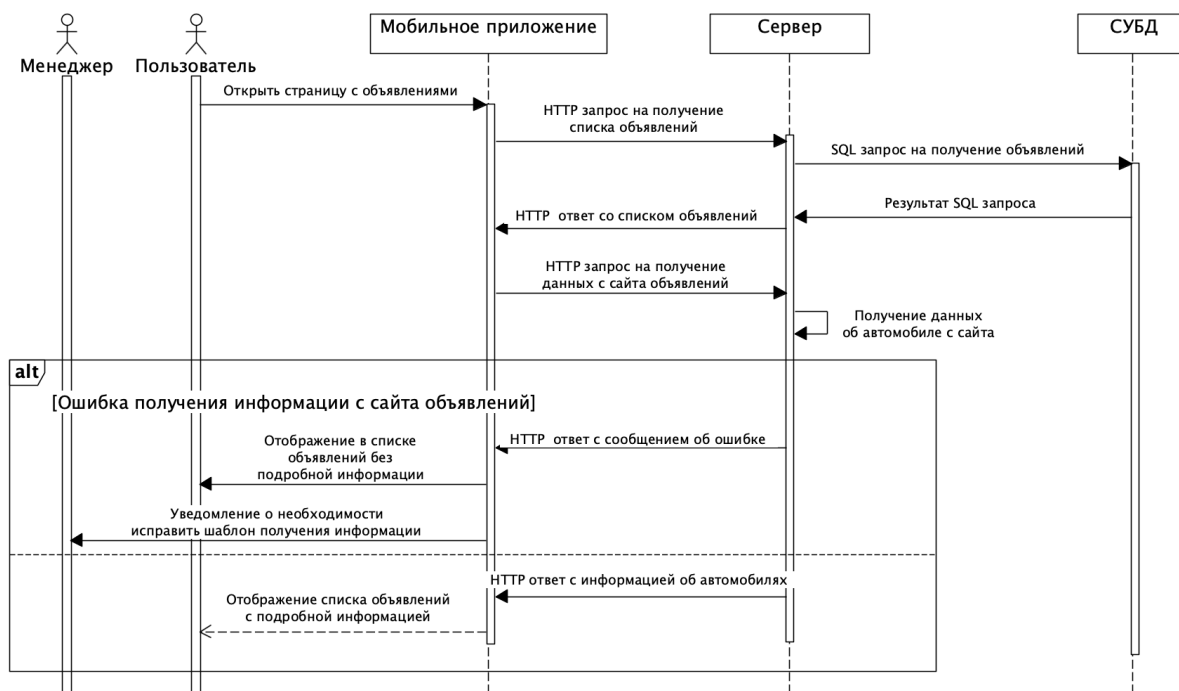


Рисунок 15 – Диаграмма последовательности варианта использования
«Посмотреть список анализируемых объявлений»

Рассмотрим подробнее процесс получения объявлений пользователем, отражённый на диаграмме последовательности (смотри рисунок 15).

Пользователь инициирует процесс, открывая страницу с объявлениями в мобильном приложении. Приложение отправляет на сервер HTTP-запрос на получение списка объявлений. Сервер, в свою очередь, формирует SQL-запрос к СУБД для выборки актуальных объявлений. Получив результат SQL-запроса, сервер возвращает приложению HTTP-ответ со списком объявлений.

После получения списка приложение отправляет повторный HTTP-запрос на сервер – на получение подробных данных о каждом автомобиле непосредственно с сайтов объявлений. Сервер обращается к соответствующим сайтам и извлекает актуальную информацию об автомобилях. На данном этапе возможны два сценария, которые отражены фрагментом alt на диаграмме.

В случае ошибки получения информации с сайта объявлений сервер возвращает приложению HTTP-ответ с сообщением об ошибке. Приложение отображает список объявлений без подробной информации, а менеджеру

направляется уведомление о необходимости исправить шаблон получения информации. В штатном сценарии сервер успешно извлекает данные и возвращает приложению HTTP-ответ с информацией об автомобилях, после чего пользователь видит полный список объявлений с подробной информацией.

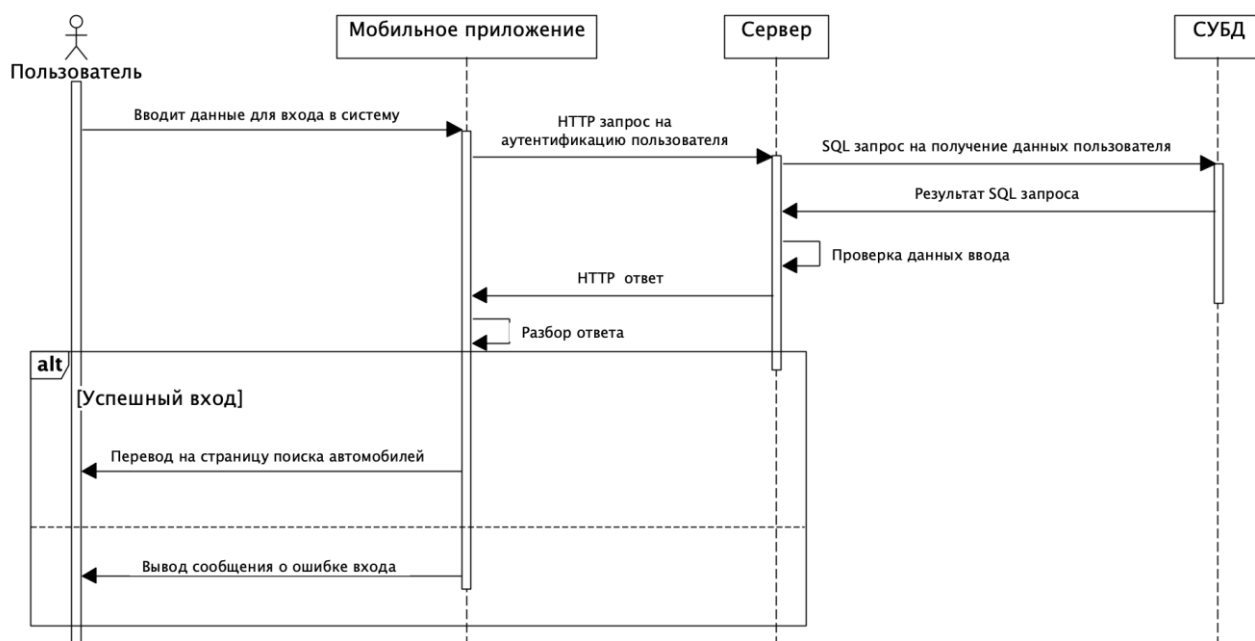


Рисунок 16 – Диаграмма последовательности варианта использования «Войти»

На представленных диаграммах последовательности для отображения альтернативных сценариев применяется фрагмент alt. Согласно учебно-методическому пособию В.Я. Ананьевой, alt является составным фрагментом взаимодействия в нотации UML, который позволяет разделить поток выполнения на несколько ветвей в зависимости от выполнения определённых условий [12]. В материалах Worldskills Russia по проектированию диаграмм последовательности также указывается, что с помощью фрагментов alt моделируются альтернативные варианты развития событий, что делает схему более наглядной и удобной для последующей реализации [14].

На текущем этапе были определены ключевые функции разрабатываемого мобильного приложения. В дополнение к этому были построены диаграммы вариантов использования, деятельности и последовательности, а также подготовлены спецификации для отдельных вариантов использования.

1.5 Анализ нефункциональных требований

На данном этапе анализа требований были сформулированы нефункциональные требования к разрабатываемому программному обеспечению. В качестве методологии разработки была выбрана технология Rational Unified Process (RUP). Этот выбор обусловлен необходимостью итеративной разработки и четкого планирования на каждом этапе: учитывая специфику системы мониторинга объявлений, требования могут уточняться по мере анализа предметной области, а RUP позволяет гибко управлять изменениями и рисками уже на ранних стадиях. В качестве основного архитектурного шаблона предполагается использование Model-View-Controller (MVC), так как он обеспечивает четкое разделение данных, логики обработки объявлений и пользовательского интерфейса, что критически важно для поддержки и масштабирования клиент-серверного приложения [15, 16].

Для моделирования архитектуры и построения диаграмм использовался инструмент Visual Paradigm Community Edition [17], который позволяет создавать диаграммы в нотации UML 2 и обеспечивает интеграцию с документированием проекта.

Серверная часть приложения будет разработана на языке C# с использованием платформы ASP.NET. Выбор данной экосистемы обусловлен высокой производительностью и строгой типизацией, что обеспечивает надежность при обработке и анализе больших потоков данных с досок объявлений. В качестве среды разработки используется Rider, так как эта среда от JetBrains оптимизирована для работы с .NET-проектами, предлагает мощные инструменты рефакторинга и анализа кода, а также встроенную поддержку Docker и баз данных, что ускорит разработку на последующих этапах. Корректность данных будет обеспечиваться СУБД PostgreSQL, взаимодействие с которой реализуется через Entity Framework Core. Использование фреймворка позволит абстрагироваться от низкоуровневых SQL-запросов и ускорить разработку модели данных. Для обеспечения гибкости развертывания и

изоляции серверной части принято решение использовать контейнеризацию Docker.

Клиентская часть представляет собой мобильное приложение, разрабатываемое на языке JavaScript с использованием строгой типизации через TypeScript. Применение TypeScript обосновано необходимостью минимизировать количество ошибок на этапе разработки и сделать код предсказуемым при взаимодействии с серверными моделями данных. Выбор фреймворка React Native продиктован возможностью создания кроссплатформенного решения с нативной производительностью при единой кодовой базе. Разработка клиентской части будет вестись в среде WebStorm, которая, как и Rider, принадлежит экосистеме JetBrains и предоставляет бесшовную интеграцию с инструментами отладки и работы с Git, что удобно в рамках единого подхода к разработке. Итоговое приложение должно обеспечивать интуитивно понятный интерфейс и корректно функционировать на целевых платформах: устройства под управлением ОС Android версии 9.0 и выше и iOS версии 18.5 и выше.

1.6 Планирование разработки и оценка бюджета

Эффективная реализация любого проекта, включая проекты в сфере информационных технологий, невозможна без тщательного и продуманного планирования [18]. Именно оно служит основой для организации всех этапов работы таким образом, чтобы достичь поставленных целей в оптимальные сроки, с минимальными затратами и на должном уровне качества. Кроме того, грамотное планирование способствует снижению рисков, повышению управляемости проекта и позволяет более гибко реагировать на возникающие изменения или непредвиденные обстоятельства в ходе его реализации.

Задачи, представляющие собой этапы работы над проектом, представлены в таблице 2.

Таблица 2 – Перечень задач проекта

№	Название задачи	Длительность	Предшественники
1	Начало проекта	–	–
2	Анализ требований приложения подбора автомобилей	–	–
3	Анализ объявлений о продаже автомобилей и постановка задачи работы	2 дня	1
4	Анализ программных продуктов по поиску и подбору автомобилей	2 дня	1
5	Анализ функциональных требований	3 дня	3;4
6	Анализ требований к мобильному интерфейсу	3 дня	5
7	Анализ целевой аудитории	1 день	6
8	Анализ требований завершен	–	7
9	Выбор технологий разработки приложения подбора автомобилей	–	–
10	Анализ существующих технологий разработки ПО	3 дня	8
11	Выбор технологии разработки ПО	2 дня	10
12	Выбор решения завершен	–	11
13	Проектирование программных средств приложения подбора автомобилей	–	–
14	Проектирование базы данных приложения	2 дня	8;12
15	Проектирование архитектуры серверной части	5 дней	14
16	Проектирование архитектуры мобильного приложения	7 дней	12
17	Проектирование приложения завершенно	–	16
18	Реализация программных средств приложения подбора автомобилей	–	–
19	Реализация базы данных приложения	3 дня	14
20	Реализация серверной части приложения	10 дней	15;19
21	Реализация мобильного приложения	22 дней	17
22	Реализация завершенна	–	20;21
23	Тестирование приложения подбора автомобилей	–	–
24	Функциональное тестирование	2 дня	22
25	Модульное тестирование	2 дня	22
26	Тестирование завершенно	–	24;25
27	Составление и оформление документации	7 дней	26
28	Конец проекта	–	27

Таким образом, для реализации проекта нужно выполнить 28 задач.

В таблице 3 представлено описание затрачиваемых ресурсов.

Таблица 3 – Список ресурсов проекта

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Программист 1	Трудовой	600 Р/ч	800 Р/ч
Программист 2	Трудовой	600 Р/ч	800 Р/ч

Продолжение таблицы 3

Название ресурса	Тип	Стандартная ставка	Ставка сверхурочных
Аналитик	Трудовой	500 Р/ч	600 Р/ч
Дизайнер	Трудовой	500 Р/ч	600 Р/ч
Тестировщик	Трудовой	300 Р/ч	350 Р/ч
Интернет	Затраты	2500 Р/месяц	—
Компьютер	Материальный	500 Р	—
Мобильное устройство	Материальный	300 Р	—
Бумага	Материальный	500 Р	—

Таким образом, в общей сложности, потребуется 9 видов ресурсов.

Распределение ресурсов по задачам представлено в таблице 4.

Таблица 4 – Назначение ресурсов

№	Название задачи	Ресурс	Единицы
1	Начало проекта	—	—
2	Анализ требований приложения подбора автомобилей	—	—
3	Анализ предметной области и постановка задачи	Аналитик Компьютер Интернет Бумага	50% 1 шт. 1 шт.
4	Анализ программных продуктов аналогичного назначения	Аналитик Компьютер Интернет Мобильное устройство Бумага	50% 1 шт. 1 шт. 1 шт.
5	Анализ функциональных требований	Аналитик Компьютер Интернет Бумага	70% 1 шт. 1 шт.
6	Анализ требований к интерфейсу	Дизайнер Аналитик Компьютер Интернет Бумага	50% 50% 1 шт. 1 шт.
7	Анализ целевой аудитории	Аналитик Компьютер Интернет Бумага	50% 1 шт. 1 шт.

Продолжение таблицы 4

№	Название задачи	Ресурс	Единицы
8	Анализ требований завершён	—	—
№	Название задачи	Ресурс	Единицы
9	Выбор технологий разработки приложения подбора автомобилей	—	—
10	Анализ существующих технологий разработки ПО	Программист 1 Программист 2 Аналитик Компьютер Интернет	50% 50% 25% 3 шт.
11	Выбор технологий разработки ПО	Программист 1 Программист 2 Аналитик Компьютер Интернет	50% 50% 30% 3 шт.
12	Выбор средств разработки ПО завершён	—	—
13	Проектирование программных средств приложения подбора автомобилей	—	—
14	Проектирование базы данных	Программист 1 Программист 2 Компьютер Интернет	70% 20% 2 шт.
15	Проектирование архитектуры серверной части приложения	Программист 1 Программист 2 Компьютер Интернет	50% 50% 2 шт.
16	Проектирование архитектуры клиентской части приложения	Программист 1 Программист 2 Дизайнер Компьютер Интернет	50% 50% 100% 3 шт.
17	Проектирование приложения завершено	—	—
18	Реализация программных средств приложения подбора автомобилей	—	—

Окончание таблицы 4

№	Название задачи	Ресурс	Единицы
19	Реализация базы данных	Программист 1 Программист 2 Компьютер Интернет	100% 50% 1 шт.
20	Реализация серверной части приложения	Программист 1 Компьютер Интернет	100% 2 шт.
21	Реализация мобильного приложения	Программист 2 Компьютер Интернет	100% 2 шт.
22	Реализация завершена	—	—
23	Тестирование приложения подбора автомобилей	—	—
24	Функциональное тестирование	Программист 1 Программист 2 Тестирующий Компьютер Интернет Мобильное устройство	20% 20% 100% 3 шт. 4 шт.
25	Модульное тестирование	Программист 1 Программист 2 Тестирующий Компьютер Интернет	20% 20% 100% 3 шт.
26	Тестирование завершено	—	—
27	Составление и оформление документации	Программист 1 Программист 2 Тестирующий Аналитик Компьютер Интернет	100% 100% 100% 100% 4 шт.
28	Конец проекта	—	—

Таким образом, реализацией проекта будут заниматься два программиста, тестировщик, аналитик и один дизайнер, которым для работы нужны четыре компьютера, два мобильных устройства, доступ к интернету и бумага.

Для вычисления длительности реализации проекта в днях и его критического пути построен сетевой график, представленный ниже на рисунке 17.

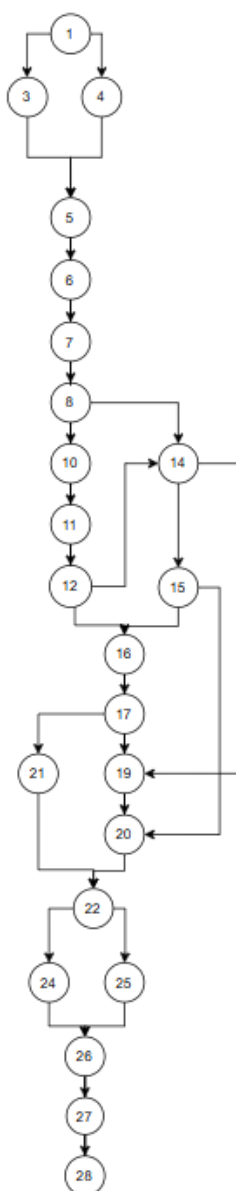


Рисунок 17 – Сетевой график проекта

В таблице 5 представлены расчеты, сделанные вручную, раннего времени начала работ для задач проекта.

Таблица 5 – Расчеты раннего времени начала работ

Работа	Предшественники			Вычисление	Раннее время начала
	№	Раннее время начала	Длительность		
1	—	—	—	—	0
3	1	0	0	0+0	0
4	1	0	0	0+0	0
5	3	0	2	Max(0+2;0+2)	2
	4	0	2		
6	5	2	3	2+3	5
7	6	5	3	5+3	8
8	7	8	1	8+1	9
10	8	9	0	9+0	9
11	10	9	3	9+3	12
12	11	12	2	12+2	14
14	8	9	0	Max(9+0;14+0)	14
	12	14	0		
15	14	14	2	14+2	16
16	12	11	0	Max(11+0;16+5)	21
	15	16	5		
17	16	21	7	21+7	28
19	14	13	2	Max(13+2;28+0)	28
	17	28	0		
20	15	15	5	Max(15+5;28+3)	31
	19	28	3		
21	17	28	0	28+0	28
22	20	31	10	Max(31+10;28+22)	50
	21	28	22		
24	22	50	0	50+0	50
25	22	50	0	50+0	54
26	24	50	2	Max(50+2;50+2)	52
	25	50	2		
27	26	52	0	52+0	52
28	27	52	7	52+7	59

Также было рассчитано позднее время начала работ для задач проекта. Расчеты представлены в таблице 6.

Таблица 6 – Расчеты позднего времени начала работ

Работа	Последователи			Вычисление	Позднее время начала
	№	Позднее время начала	Длительность		
28	-	-	-		59
27	28	59	7	59-7	52
26	27	52	0	52-0	52
25	26	52	2	52-2	50
24	26	52	2	52-2	50
22	24 25	50 50	0	Min(50-0;50-0)	50
21	22	50	22	50-22	28
20	22	50	10	50-10	40
19	20	40	3	40-3	37
17	19 21	37 28	0	Min(37-0;28-0)	28
16	17	28	7	28-7	21
15	16 20	21 40	5	Min(21-5;40-5)	16
14	15 19	16 37	2	Min(16-2;37-2)	14
12	14 16	14 21	0	Min(14-0;21-0)	14
11	12	14	2	14-2	12
10	11	12	3	12-3	9
8	10 14	9 14	0	Min(9-0;14-0)	9
7	8	9	1	9-1	8
6	7	8	3	8-3	5
5	6	5	3	5-3	2
4	5	2	2	2-2	0
3	5	2	2	2-2	0
1	3 4	0 0	0	Min(0-0;0-0)	0

По результатам нахождения раннего и позднего времени начала работ можно найти критический путь проекта. Критический путь проекта – это последовательность задач, которая определяет расчётную дату окончания проекта. Завершение последней задачи критического пути ведёт к завершению всего проекта. Вычисление критического пути представлено в таблице 7

Таблица 7 – Вычисление критического пути проекта

Номер работы	Раннее время начала	Позднее время начала	Резерв времени
1	0	0	0
3	0	0	0
4	0	0	0
5	2	2	0
6	5	5	0
7	8	8	0
8	9	9	0
10	9	9	0
11	12	12	0
12	14	14	0
14	14	14	0
15	16	16	0
16	21	21	0
17	28	28	0
19	28	37	9
20	31	40	9
21	28	28	0
22	50	50	0
24	50	50	0
25	50	54	4
26	52	52	0
27	52	52	0
28	59	59	0

На основе полученных данных из расчета, можно сделать важный и небезосновательный вывод о том, что критический путь проекта проходит через узлы графа, с номерами работ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 21, 22, 24, 26, 27, 28. Общее время, затрачиваемое на выполнение всего проекта, составляет 59 рабочих дня. Граф проекта, где пунктирными линиями указан путь, показывающий за какое минимальное время, можно завершить всю работу над проектом, приведен на рисунке 18.

Задачи, не вошедшие в критический путь, обладают временным резервом – то есть допускают определённую задержку без влияния на итоговые сроки проекта. Наличие резервов позволяет руководителю проекта гибко перераспределять ресурсы: при возникновении трудностей на критических

задачах исполнителей с некритических участков можно временно привлечь для ускорения ключевых работ.

Построенный сетевой граф и выявленный критический путь служат инструментом оперативного контроля за ходом разработки и позволяют заблаговременно выявлять риски срыва сроков проекта.

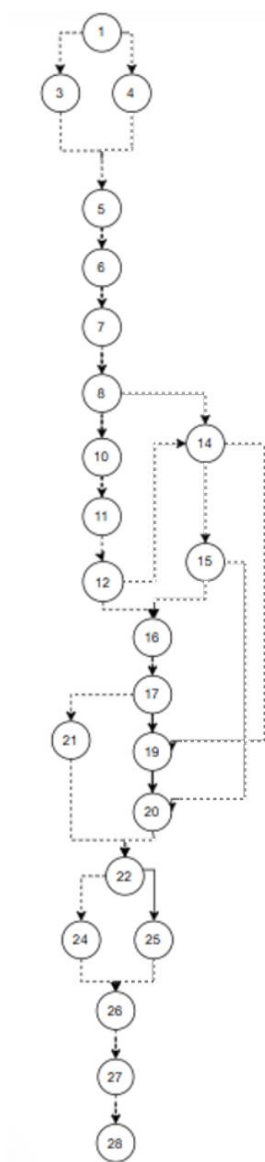


Рисунок 18 – Критический путь проекта

Для проверки корректности расчетов продолжительности проекта, а также для вычисления бюджета, необходимого для создания приложения, воспользуемся системой управления проектами Microsoft Project [19]. Данная

программа позволяет автоматически производить все вычисления, основываясь на данных пользователя.

Список задач из таблицы 4, внесённый в систему Microsoft Project, представлен на рисунке 19. В качестве времени начала проекта была принята дата 24.02.2025.

		Название задачи	Длительнс	Предшественн	Начало	Окончани
1		Начало проекта	0 дней		Пн 23.02.26	Пн 23.02.26
2		▲ Анализ требований приложения подбора автомобилей	9 дней		Пн 23.02.26	Чт 05.03.26
3		Анализ предметной области и постановка задачи	2 дней	1	Пн 23.02.26	Вт 24.02.26
4		Анализ программных продуктов аналогичного назначения	2 дней	1	Пн 23.02.26	Вт 24.02.26
5		Анализ функциональных требований	3 дней	4;3	Ср 25.02.26	Пт 27.02.26
6		Анализ требований к интерфейсу	3 дней	5	Пн 02.03.26	Ср 04.03.26
7		Анализ целевой аудитории	1 день	6	Чт 05.03.26	Чт 05.03.26
8		Анализ требований завершен	0 дней	7	Чт 05.03.26	Чт 05.03.26
9		▲ Выбор технологий разработки приложения подбора автомобилей	5 дней		Пт 06.03.26	Чт 12.03.26
10		Анализ существующих технологий разработки ПО	3 дней	8	Пт 06.03.26	Вт 10.03.26
11		Выбор технологии разработки ПО	2 дней	10	Ср 11.03.26	Чт 12.03.26
12		Выбор решения завершен	0 дней	11	Чт 12.03.26	Чт 12.03.26
13		▲ Проектирование программных средств приложения подбора автомобилей	14 дней		Чт 12.03.26	Ср 01.04.26
14		Проектирование базы данных	2 дней	8;12	Чт 12.03.26	Пн 16.03.26
15		Проектирование архитектуры серверной части приложения	5 дней	14	Вт 17.03.26	Пн 23.03.26
16		Проектирование архитектуры мобильного приложения	7 дней	12;15	Вт 24.03.26	Ср 01.04.26
17		Проектирование приложения завершенно	0 дней	16;14;15	Ср 01.04.26	Ср 01.04.26
18		▲ Реализация программных средств приложения подбора автомобилей	22 дней		Чт 02.04.26	Пт 01.05.26
19		Реализация базы данных	3 дней	14;17	Чт 02.04.26	Пн 06.04.26
20		Реализация серверной части приложения	10 дней	15;19	Вт 07.04.26	Пн 20.04.26
21		Реализация мобильного приложения	22 дней	17	Чт 02.04.26	Пт 01.05.26
22		Реализация завершенна	0 дней	20;21	Пт 01.05.26	Пт 01.05.26
23		▲ Тестирование приложения подбора автомобилей	9 дней		Пн 04.05.26	Чт 14.05.26
24		Функциональное тестирование	2 дней	22	Пн 04.05.26	Вт 05.05.26
25		Модульное тестирование	2 дней	22	Пн 04.05.26	Вт 05.05.26
26		Тестирование завершенно	0 дней	24;25	Вт 05.05.26	Вт 05.05.26
27		Составление и оформление документации	7 дней	26	Ср 06.05.26	Чт 14.05.26
28		Окончание проекта	0 дней	27	Чт 14.05.26	Чт 14.05.26

Рисунок 19 – Задачи проекта в среде Microsoft Project

Построенная системой Microsoft Project диаграмма Ганта приведена на рисунке 20.

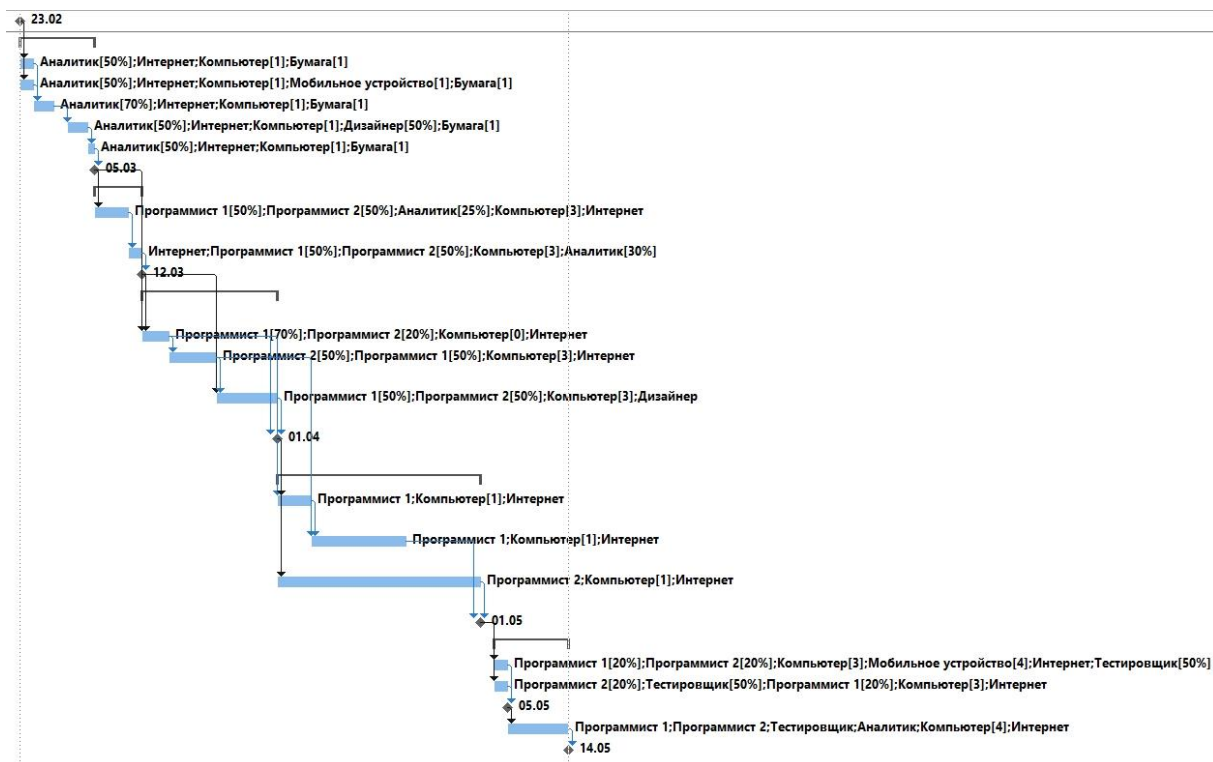


Рисунок 20 – Диаграмма Ганта

На рисунке 21 представлен график трудового ресурса «Аналитик».

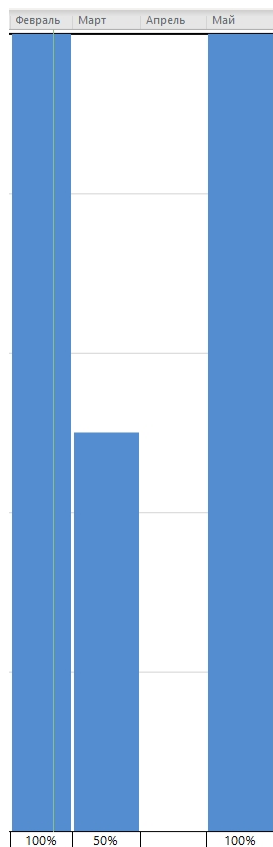


Рисунок 21 – График ресурсов для ресурса «Аналитик»

На рисунке 22 представлен график трудового ресурса «Тестировщик».

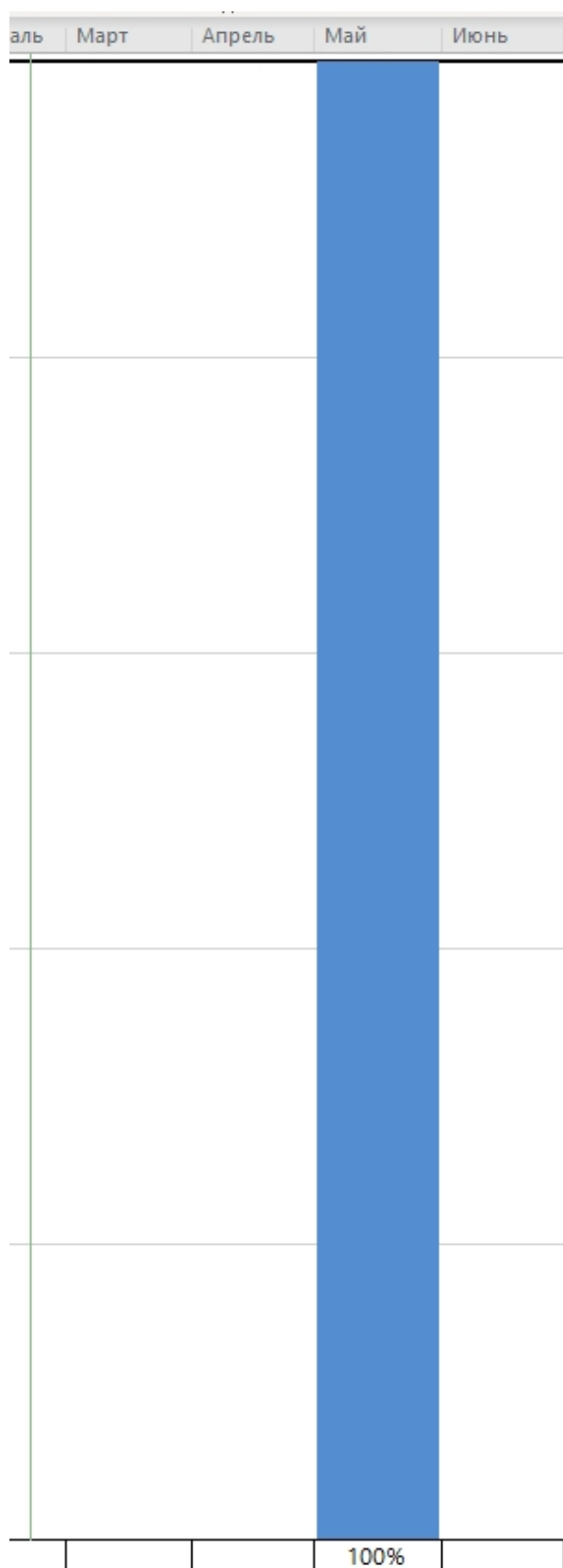


Рисунок 22 – График ресурсов для ресурса «Тестировщик»

На рисунке 23 представлен график трудового ресурса «Дизайнер».

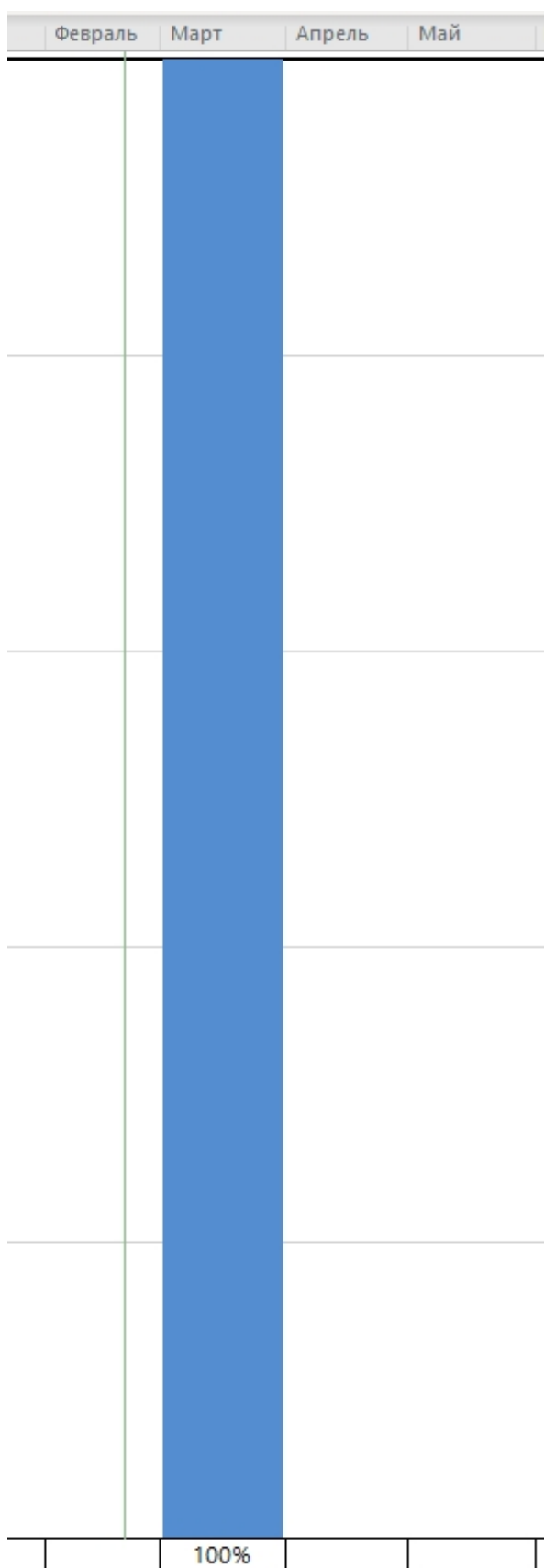


Рисунок 23 – График ресурсов для ресурса «Дизайнер»

На рисунке 24 представлен график трудового ресурса «Компьютер».

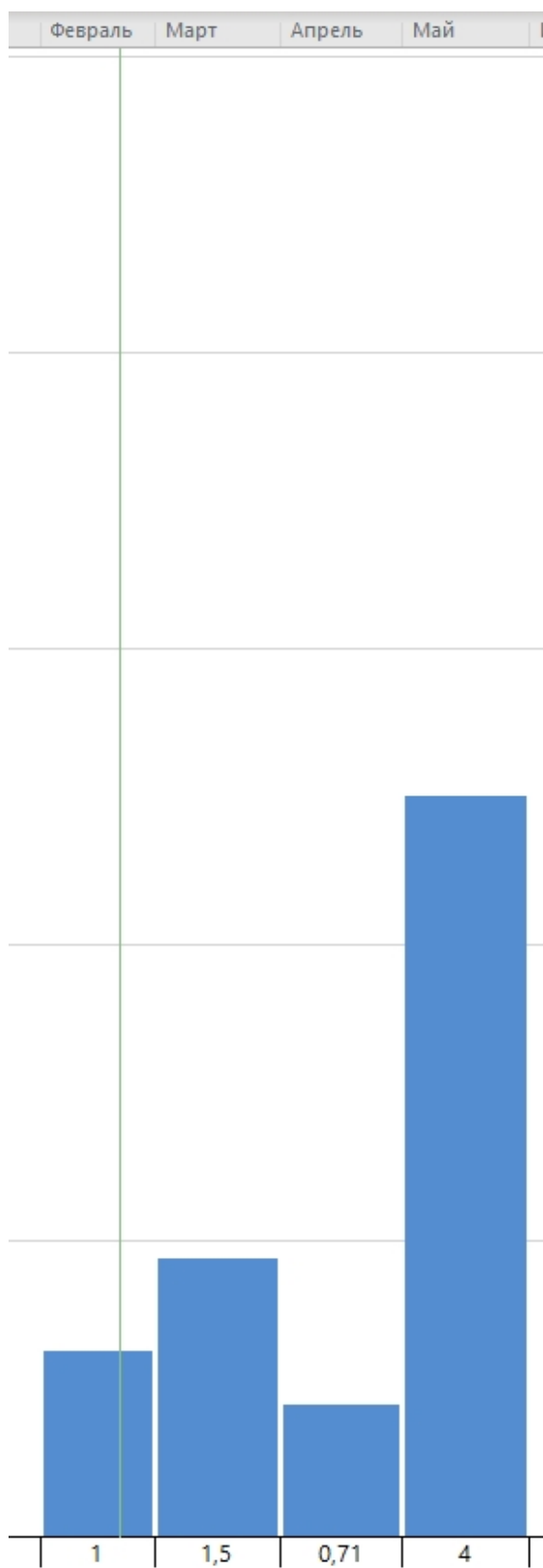


Рисунок 24 – График ресурсов для ресурса «Компьютер»

На рисунке 25 представлен график трудового ресурса «Мобильное устройство».



Рисунок 25 – График ресурсов для ресурса «Мобильное устройство»

На рисунке 26 представлен график трудового ресурса «Бумага».

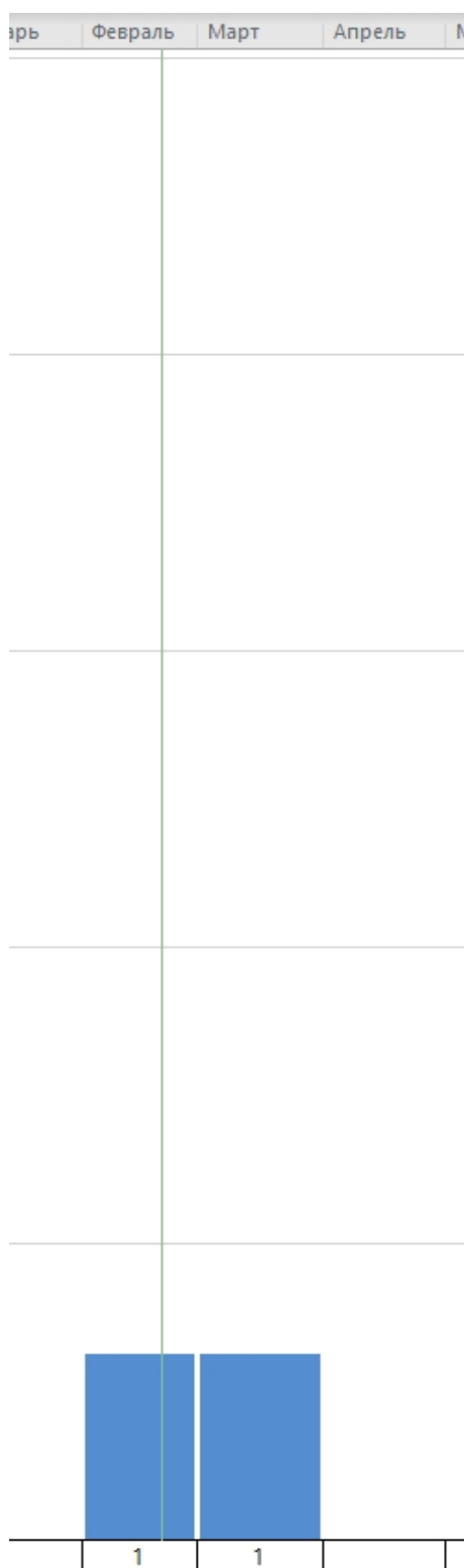


Рисунок 26 – График ресурсов для ресурса «Бумага»

На рисунке 27 представлен график трудового ресурса «Программист 1».

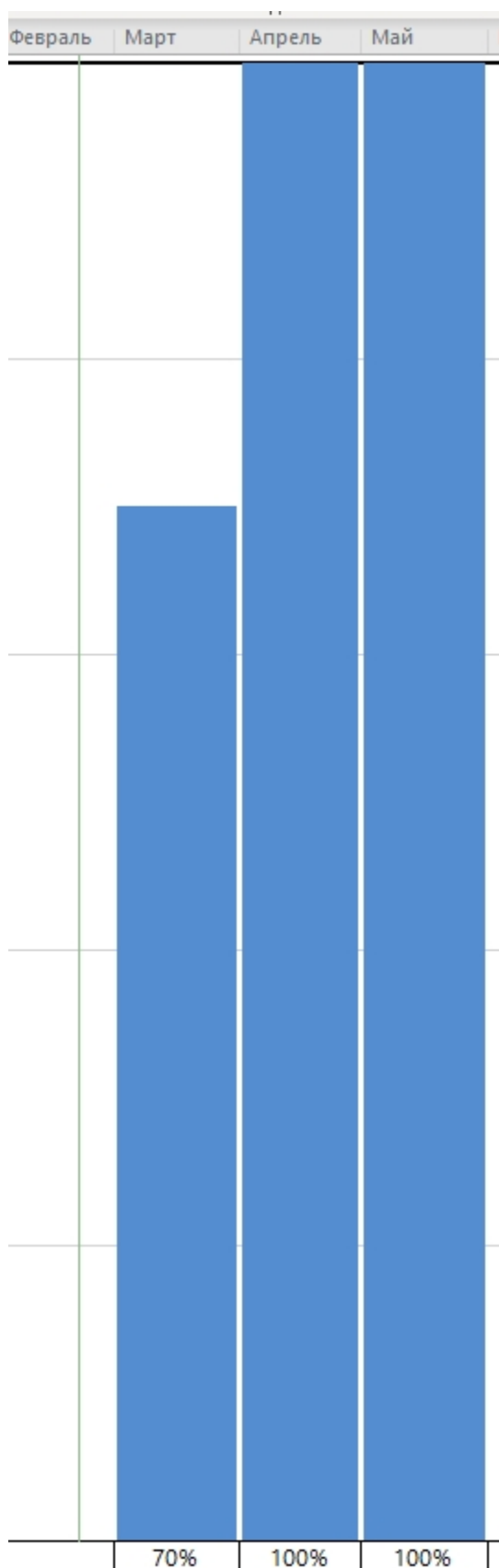


Рисунок 27 – График ресурсов для ресурса «Программист 1»

На рисунке 28 представлен график трудового ресурса «Программист 2».

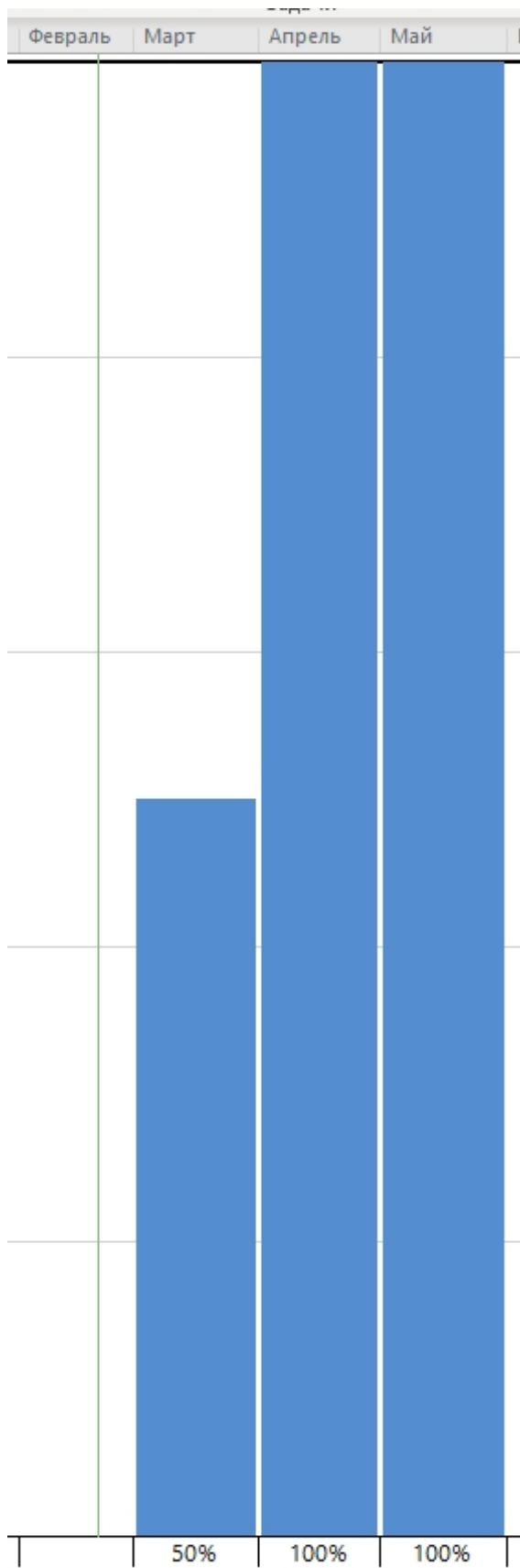


Рисунок 28 – График ресурсов для ресурса «Программист 2»

На рисунке 29 представлен обзор проекта. Из рисунка видно, что общая длительность проекта составляет 59 дней (что совпадает со значением, вычисленным вручную), а предполагаемые затраты – 450 320 руб.

Статистика проекта для 'Проект1'

	Начало	Окончание
Текущее	Пн 23.02.26	Чт 14.05.26
Базовое	НД	НД
Фактическое	НД	НД
Отклонение	0д	0д

	Длительность	Трудозатраты	Затраты
Текущие	59д	782,8ч	450 320,00 Р
Базовые	0д	0ч	0,00 Р
Фактические	0д	0ч	0,00 Р
Оставшиеся	59д	782,8ч	450 320,00 Р

Процент завершения

Длительность: 0%Трудозатраты: 0%

Заккрыть

Рисунок 29 – Обзорный отчёт проекта

На основе произведённых оценок стоимости и сроков проекта можно сделать вывод о его экономической целесообразности.

2 Проектирование мобильного приложения

2.1 Проектирование архитектуры программного обеспечения

Было принято решение использовать в приложении трехуровневую архитектуру «клиент-сервер». Выбор данной архитектуры обусловлен необходимостью разделения ответственности между компонентами системы, что особенно важно для приложений, работающих с динамически обновляемыми данными [20].

В данной архитектуре приложение делится на 3 отдельных уровня:

- уровень представления – отвечает за ввод и вывод данных пользователю;
- уровень приложения – отвечает за обработку запросов пользователей, выполняет вычисления и принимает решения, выступает в качестве посредника между уровнями представления и данных;
- уровень данных – отвечает за управление и хранение данных.

Данная архитектура была выбрана на основе ее преимуществ. Она легко масштабируется независимо от другого уровня, что позволяет при росте числа пользователей увеличивать мощность серверной части без изменений в клиентском приложении. Архитектура удобна в обслуживании: обновление или изменение одного уровня не затрагивает другие. Например, доработка пользовательского интерфейса не влияет на серверную логику. Также главными достоинствами данной архитектуры являются ее гибкость, производительность и безопасность [20]. Ее схема представлена на рисунке 30.



Рисунок 30 – Трехуровневая архитектура «Клиент-сервер»

Взаимодействие в данной архитектуре происходит следующим образом: клиент отправляет HTTP-запрос серверу, сервер обрабатывает этот запрос, выполняет необходимые вычисления и при необходимости отправляет SQL-запрос базе данных. СУБД обрабатывает SQL-запрос и возвращает его результаты серверу. Сервер на основе полученных данных формирует ответ на запрос клиента и отправляет его обратно. Клиент получает ответ и отображает данные в удобочитаемой форме пользователю приложения. Такая схема обеспечивает четкое разделение ответственности и упрощает поддержку системы мониторинга объявлений.

2.2 Проектирование базы данных

2.2.1 Концептуальное проектирование базы данных

На основе анализа предметной области и диаграммы вариантов использования была построена концептуальная модель базы данных. Построение модели выполнено с использованием инструмента Visual Paradigm Community Edition, позволяющего создавать диаграммы в нотации «сущность-связь». Графическое представление концептуальной модели БД в виде диаграммы «сущность-связь» представлено на рисунке 31.

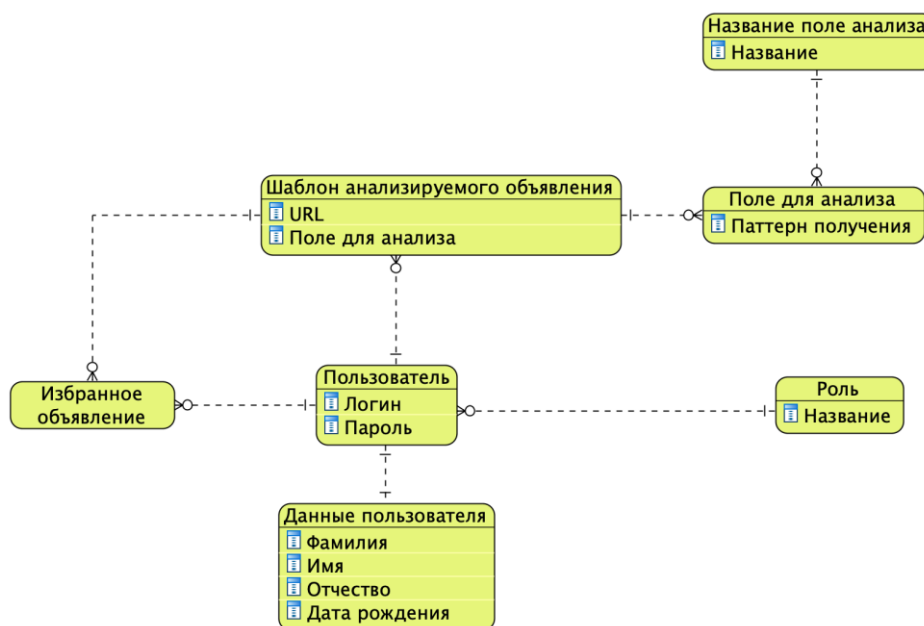


Рисунок 31 – Графическое представление концептуальной модели базы данных

Сущности концептуальной модели данных представлены в таблице 8.

Таблица 8 – Сущности концептуальной модели базы данных

№	Название	Описание	Количество экземпляров	Режим обновления
1	Пользователь	Список пользователей, менеджеров и администраторов	10000	При регистрации нового пользователя в системе
2	Роль	Список возможных ролей	3	При добавлении нового функционала требующего нового типа пользователей
3	Шаблон анализируемого объявления	Список объявлений с объявлениями о продаже авто	300000	При добавлении, удалении машин на сайтах
4	Избранное объявление	Список объявлений, которые пользователи добавили в избранное	300000	При добавлении, удалении объявлений в избранное
5	Поле для анализа	Список данных, получаемых с сайтов	900000	При добавлении, удалении, изменении объявлений
6	Название поле анализа	Список название полей для анализа	20	При добавлении, изменении анализируемых данных для всех объявлений
7	Данные пользователя	Данные пользователя	10000	При регистрации нового пользователя в системе
8	Полученные данные	Данные, полученные по шаблону с сайта	900000	При актуализации данных

В таблице 9 представлено описание связей между сущностями.

Таблица 9 – Описание связей между сущностями концептуальной модели

№	Связь	Описание	Мощность	Обязательность
1	Пользователь – роль	Определяет роль пользователя	1:M	Да
2	Пользователь – Шаблон анализируемого объявления	Определяет шаблоны объявлений, добавленные пользователем(модератором)	1:M	Нет
3	Пользователь – избранное объявление	Определяет объявления, избранные пользователем	1:M	Нет
4	Избранный объявление – Шаблон анализируемого объявления	Определяет каким является избранное объявление	1:M	Да

Продолжение таблицы 9

№	Связь	Описание	Мощность	Обязательность
5	Шаблон анализируемого объявления – Поле для анализа	Определяет какие поля будут анализироваться на каком сайте	1:M	Да
6	Поле для анализа – Название поле анализа	Определяет название типа поля для названия для последующей фильтрации	1:M	Да
7	Пользователь – данные пользователя	Определяет данные конкретного пользователя	1:1	Да
8	Шаблон анализируемого объявления – Полученные данные	Определяет данные полученные при анализе объявления	1:M	Да

Таким образом, были спроектированы основные сущности с атрибутами и связями между ними, без включения атрибутов первичных и внешних ключей.

2.2.2 Логическое проектирование базы данных

На основании концептуальной модели была спроектирована логическая модель базы данных (рисунок 32). Построение логической модели также выполнено в среде Visual Paradigm Community Edition.

На данной логической модели каждая сущность представлена в виде таблицы с чётко определённым набором столбцов, для которых указаны наименования, типы данных и признак обязательности заполнения. Нормализация структуры выполнена с целью устранения избыточности данных и предотвращения аномалий при операциях вставки, обновления и удаления записей. Связи между таблицами классифицированы по типу – «один к одному», «один ко многим» и «многие ко многим» – и реализованы посредством соответствующих ключевых атрибутов. Такой уровень детализации позволяет однозначно перейти от логической модели к физической реализации базы данных в конкретной СУБД.

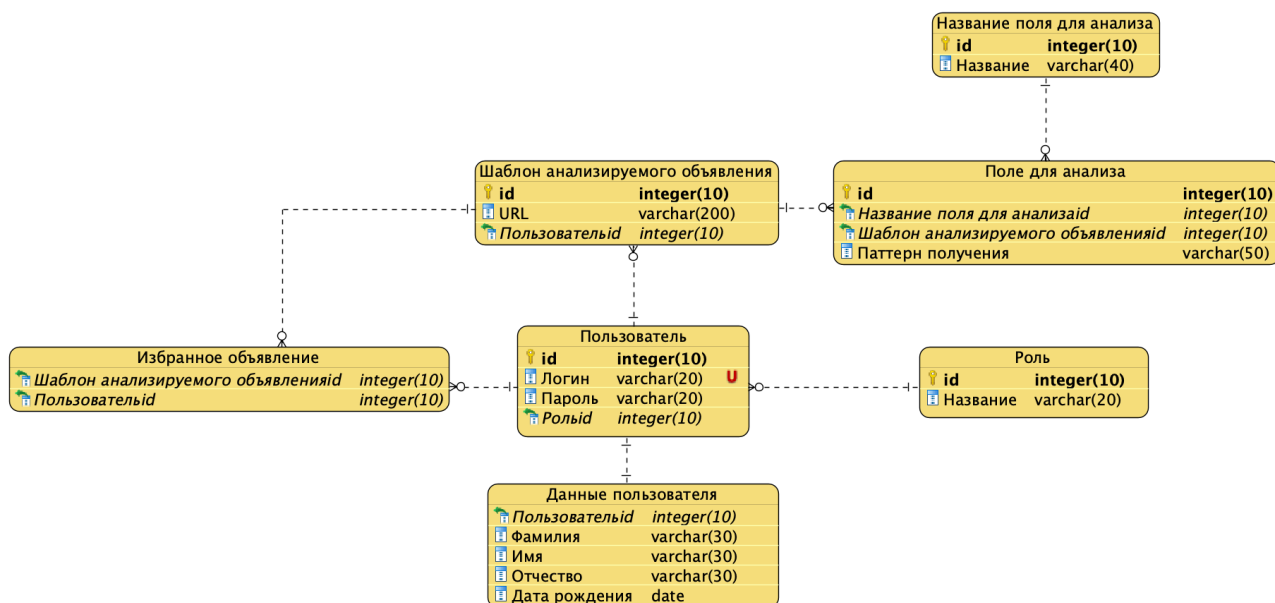


Рисунок 32 – Графическое представление логической модели базы данных

На данном представлении модели базы данных сущности, связи и атрибуты описаны более подробно по сравнению с концептуальной моделью, однако структура по-прежнему остаётся независимой от конкретной СУБД и не содержит платформенно-специфичных решений, таких как индексы, хранимые процедуры или особенности физического хранения данных.

2.3 Проектирование серверной части приложения

Серверная часть разрабатываемого мобильного приложения будет реализована на основе архитектурного стиля REST (Representational State Transfer). REST – это подход к организации взаимодействия между клиентом и сервером, который использует протокол HTTP и его методы: GET – для получения данных, POST – для создания нового ресурса, PUT – для полного обновления, PATCH – для частичного обновления, DELETE – для удаления ресурса. Каждый ресурс имеет уникальный идентификатор – URI (Uniform Resource Identifier). Такой подход обеспечивает масштабируемость, простоту интеграции с клиентскими приложениями и независимость серверной логики от клиентской платформы [21].

В качестве основного архитектурного шаблона серверной части выбран Model-View-Controller (MVC). Схема данной архитектуры представлена на рисунке 33. Выбор MVC обусловлен необходимостью чёткого разделения логики обработки данных, визуального представления и управления запросами. Это соответствует ранее сделанному выбору в разделе 1.5 и требованиям к масштабируемости и поддержке кода [21, 22].

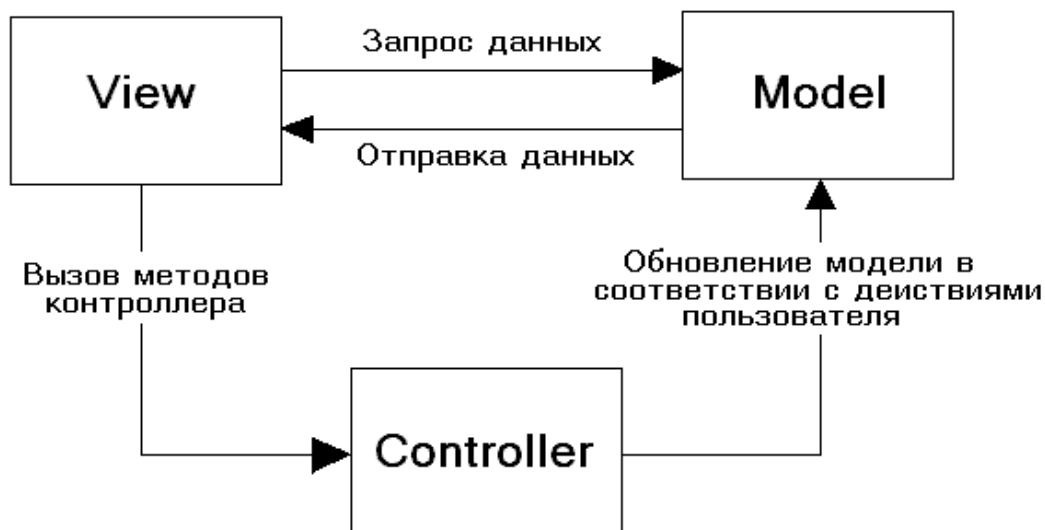


Рисунок 33 – Схема архитектуры MVC

Данная архитектура состоит из трёх отдельных модулей:

Model (модель) – основная логика приложения и данные. Модель представляет собой набор классов (сущностей), которые отражают предметную область. Модель взаимодействует с базой данных через Entity Framework Core, реализует бизнес-правила и уведомляет контроллер об изменениях состояния. Модель ничего не знает о представлении и контроллере, что обеспечивает её переиспользование [23].

View (представление) – в REST API роль представления выполняют форматы передачи данных, чаще всего JSON. Контроллер сериализует объекты модели в JSON и отправляет клиенту. Таким образом, клиентское мобильное приложение получает структурированные данные для отображения [24].

Controller (контроллер) – центральный элемент, обрабатывающий входящие HTTP-запросы от клиента. Контроллер проверяет входные параметры,

вызывает соответствующие методы модели для выполнения бизнес-логики, выбирает нужное представление и возвращает ответ клиенту с соответствующим HTTP-статусом.

2.4 Проектирование интерфейса пользователя

На основе выявленных функциональных требований был спроектирован интерфейс мобильного приложения. Приложение содержит следующие основные экраны: страница объявлений, страница профиля, а также экраны авторизации и регистрации.

Навигация организована с помощью нижней таб-панели. Ролевая модель определяет доступность отдельных функций: обычные пользователи могут просматривать объявления, добавлять их в избранное; администратор дополнительно может создавать, редактировать и удалять объявления.

Макет интерфейса страницы авторизации представлен на рисунке 34.

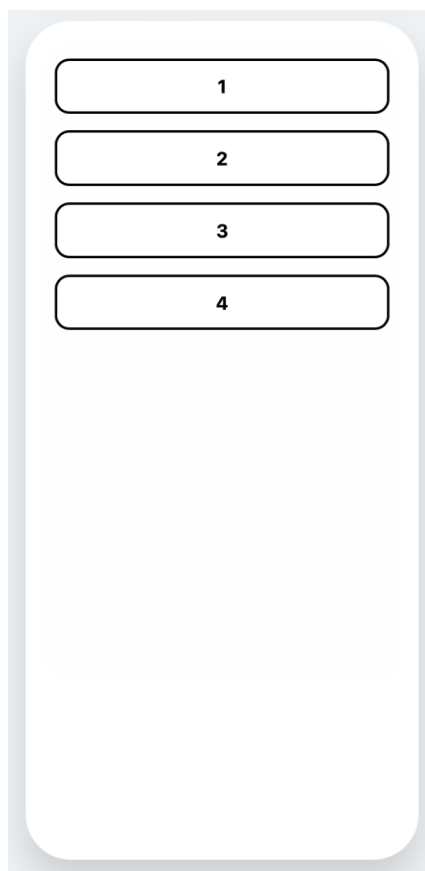


Рисунок 34 – Макет интерфейса страницы авторизации

На макете страницы авторизации имеются следующие элементы:

- 1 – поле, для ввода логина;
- 2 – поле, для ввода пароля;
- 3 – кнопка для осуществления входа;
- 4 – кнопка для перехода на страницу регистрации.

Макет интерфейса страницы регистрации представлен на рисунке 35.

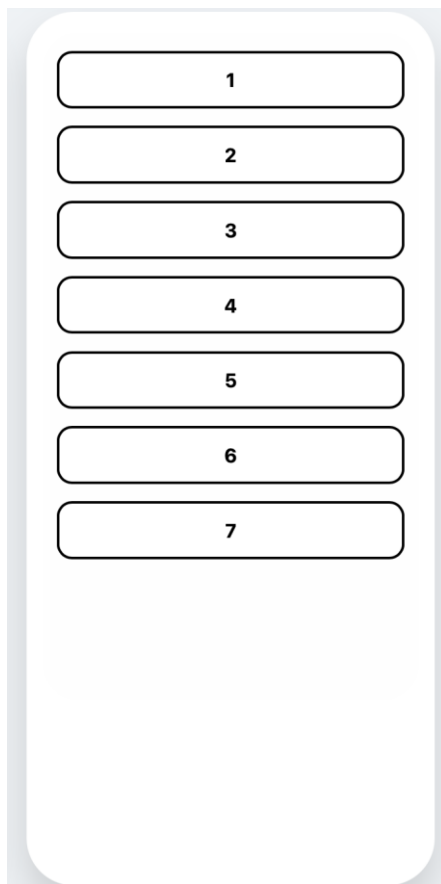


Рисунок 35 – Макет интерфейса страницы регистрации

На макете страницы регистрации имеются следующие элементы:

- 1 – поясняющий текст для пользователя;
- 2 – поле для ввода логина;
- 3 – поле для ввода пароля;
- 4 – поле для подтверждения пароля;
- 5 – кнопка для осуществления регистрации;
- 6 – поясняющий текст для пользователя;
- 7 – кнопка для перехода на страницу авторизации.

Макет интерфейса страницы объявлений представлен на рисунке 36.

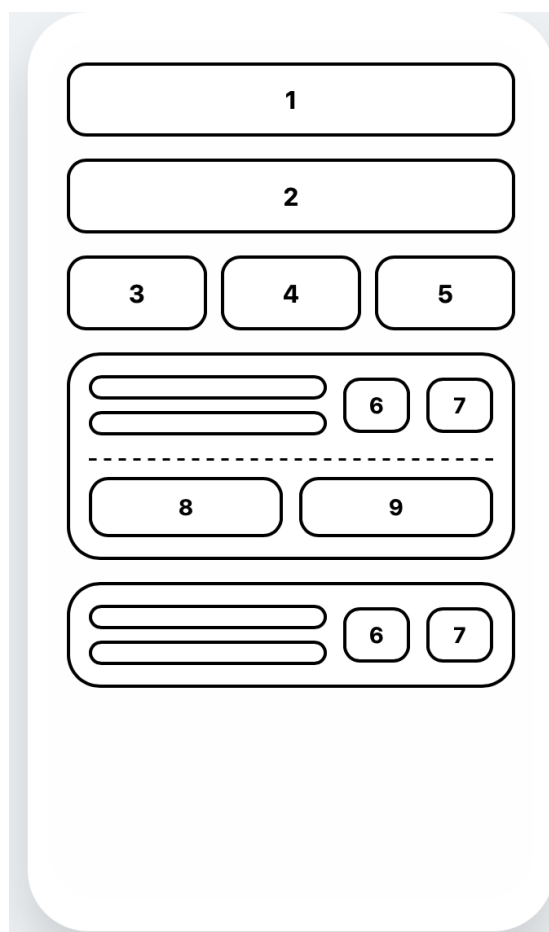


Рисунок 36 – Макет интерфейса страницы объявлений

На макете страницы каталога имеются следующие элементы:

- 1 – заголовок «Объявления»;
- 2 – строка поиска;
- 3 – кнопка вызова панели фильтрации;
- 4 – кнопка выбора сортировки;
- 5 – кнопка «Добавить объявление» (отображается только для администратора);
- 6 – кнопка добавления/удаления объявления в избранное (звезда);
- 7 – кнопка раскрытия карточки для отображения дополнительной информации (технические характеристики, описание, URL источника);
- 8 – кнопка редактирования объявления (для администратора, отображается внутри раскрытой карточки);

9 – кнопка удаления объявления (для администратора, внутри раскрытой карточки).

Макет панели фильтрации (выезжающее окно) представлен на рисунке 37.

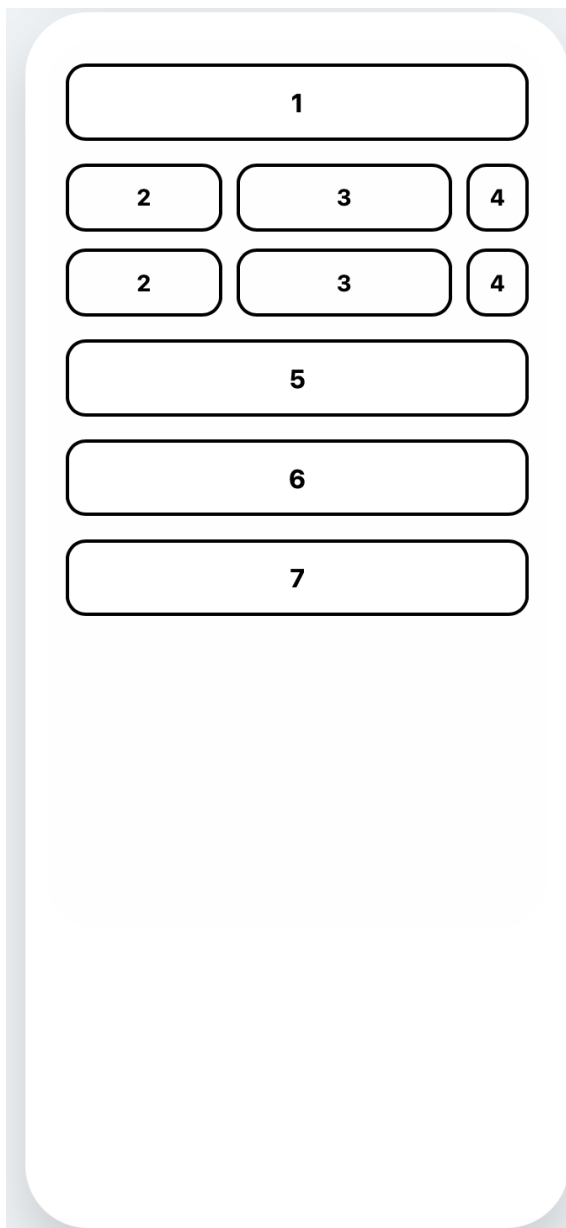


Рисунок 37 – Макет панели фильтрации (динамические условия)

На макете панели фильтрации имеются следующие элементы:

- 1 – заголовок «Фильтры»;
- 2 – выбор параметра фильтрации;
- 3 – поле ввода значения для выбранного параметра;
- 4 – кнопка удаления данного условия;

5 – кнопка «Добавить условие»;

6 – кнопка «Применить фильтры»;

7 – кнопка «Сбросить все фильтры».

Макет интерфейса страницы добавления/редактирования объявления представлен на рисунке 38.

The wireframe shows a vertical stack of UI elements on a light gray background. At the top is a white rounded rectangle labeled '1'. Below it is another white rounded rectangle labeled '2'. The third element is a container with a rounded top and bottom, containing a white rounded rectangle labeled '3'. Inside this container is a dashed-line box containing two yellow rounded rectangles. The top yellow rectangle is labeled '4' and has a small white pill-shaped label above it that says 'Выделено: цена'. The bottom yellow rectangle is labeled '5' and has a small white pill-shaped label above it that says 'Выделено: пробег'. Below the dashed box are three more white rounded rectangles labeled '6', '7', and '8' in descending order.

Рисунок 38 – Макет интерфейса страницы добавления/редактирования объявления

На макете страницы добавления/редактирования объявления имеются следующие элементы:

1 – поле для ввода URL страницы с объявлением;

- 2 – кнопка «Загрузить страницу»;
- 3 – область предпросмотра загруженной страницы;
- 4 – выделенный элемент в области предпросмотра – визуально обведён рамкой;
- 5 – другой выделенный элемент;
- 6 – кнопка «Добавить новое выделение»;
- 7 – кнопка «Сохранить объявление»;
- 8 – кнопка «Отмена».

Макет интерфейса страницы профиля представлен на рисунке 39.

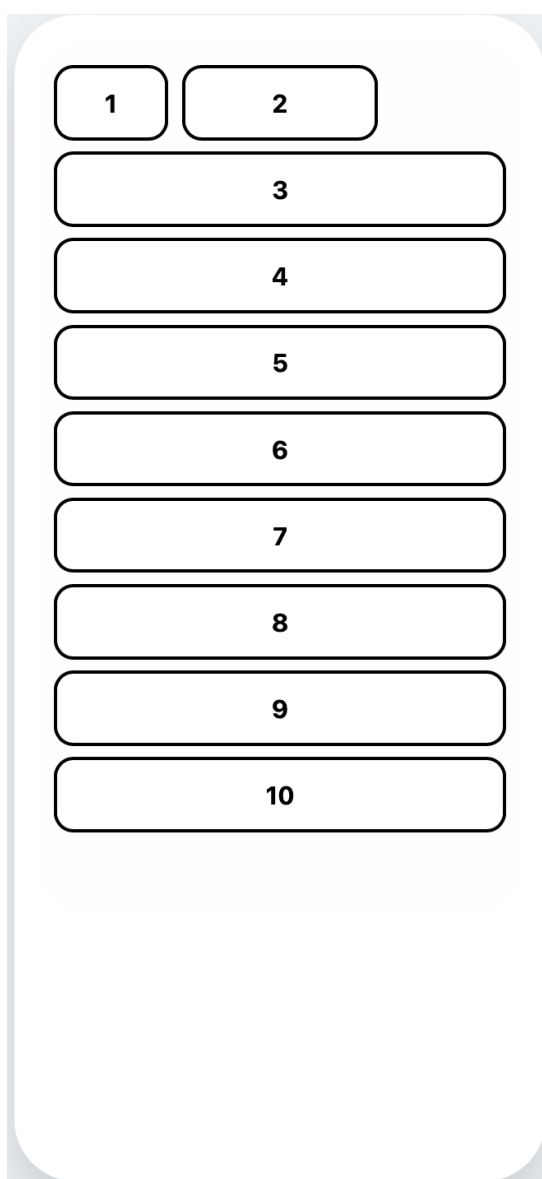


Рисунок 39 – Макет интерфейса страницы профиля

На макете страницы профиля имеются следующие элементы:

- 1 – кнопка возврата на главную страницу;
- 2 – заголовок «Профиль»;
- 3 – имя пользователя (логин);
- 4 – отображение роли (пользователь/администратор);
- 5 – блок информации (логин, роль, статус);
- 6 – кнопка «Перейти к объявлениям»;
- 7 – кнопка «Выйти» (для авторизованных);
- 8 – для гостя: приглашение войти, преимущества регистрации, кнопки «Войти» и «Зарегистрироваться»;
- 9 – блок информации о приложении.

Таким образом, спроектированный интерфейс мобильного приложения обеспечивает удобный доступ ко всем функциям системы. Экран объявлений предоставляет пользователю каталог с возможностью поиска, сортировки и фильтрации, экран профиля позволяет управлять учётной записью и просматривать избранное, а экраны авторизации и регистрации обеспечивают безопасный доступ к функционалу приложения. Ролевая модель разграничивает возможности обычного пользователя и администратора, предоставляя последнему инструменты добавления, редактирования и удаления объявлений. Разделение функциональности по экранам с навигацией через нижнюю панель вкладок соответствует общепринятым паттернам проектирования мобильных интерфейсов и обеспечивает интуитивно понятное взаимодействие пользователя с системой.

3 Реализация программного обеспечения

3.1 Реализация базы данных

На основании концептуальной и логической модели для СУБД PostgreSQL [24] была разработана физическая модель базы данных (рисунок 40).

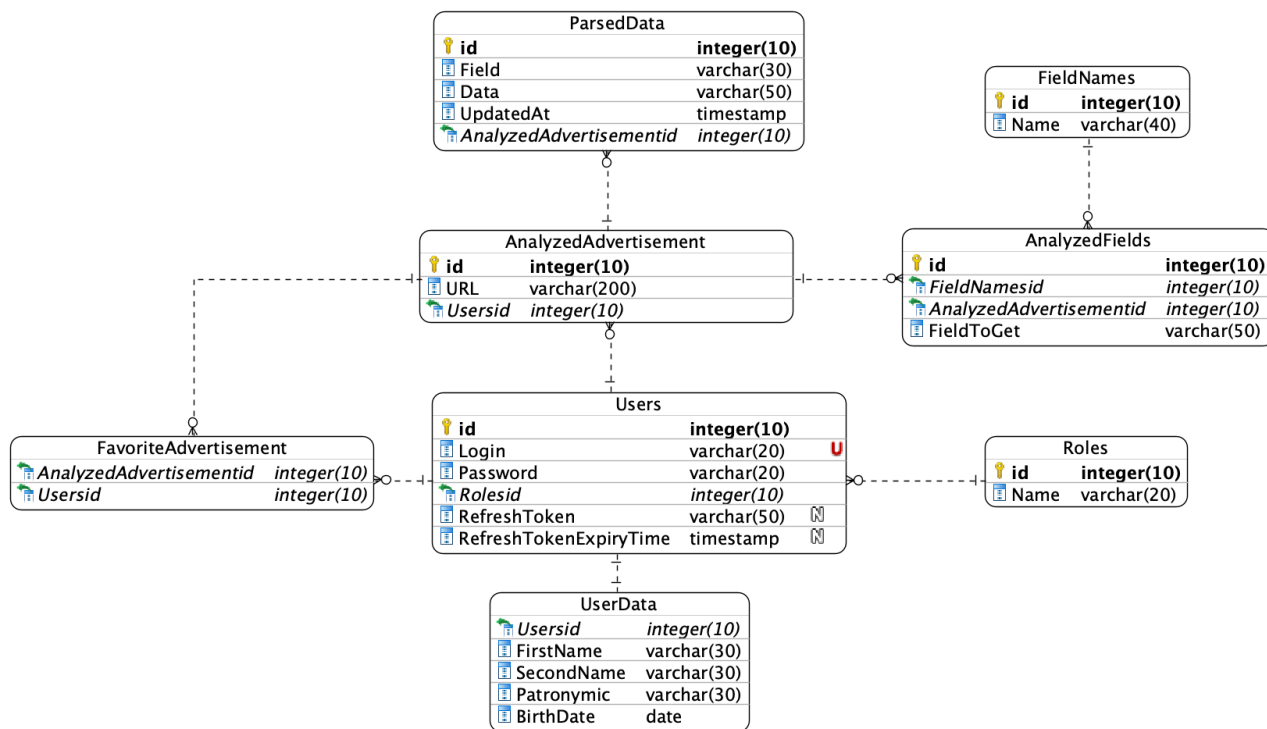


Рисунок 40 – Графическое представление физической модели базы данных

Описание атрибутов сущности «Полученные данные» представлено в таблице 10.

Таблица 10 – Описание атрибутов сущности «Полученные данные»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer(10)	Нет	Да	PK
Field	Название поля	varchar(30)	Нет	Нет	-
Data	Данные	varchar(50)	Нет	Нет	-
UpdatedAt	Дата обновления	timestamp	Нет	Нет	-
AnalyzedAdvertisementid	Идентификатор объявления	integer(10)	Нет	Нет	FK

Описание атрибутов сущности «Название поля для анализа» представлено в таблице 11.

Таблица 11 – Описание атрибутов сущности «Название поля для анализа»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer(10)	Нет	Да	PK
Name	Наименование поля	varchar(40)	Нет	Нет	-

Описание атрибутов сущности «Шаблон анализируемого сайта» представлено в таблице 12.

Таблица 12 – Описание атрибутов сущности «Шаблон анализируемого объявления»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer(10)	Нет	Да	PK
URL	Адрес сайта	varchar(200)	Нет	Нет	-
UserId	Идентификатор пользователя	integer(10)	Нет	Нет	FK

Описание атрибутов сущности «Поле для анализа» представлено в таблице 13.

Таблица 13 – Описание атрибутов сущности «Поле для анализа»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer(10)	Нет	Да	PK
FieldNameId	Идентификатор названия поля	integer(10)	Нет	Нет	FK
AnalyzedAdvertisementId	Идентификатор анализируемого сайта	integer(10)	Нет	Нет	FK
FieldToGet	Поле для получения	varchar(50)	Нет	Нет	-

Описание атрибутов сущности «Избранное объявление» представлено в таблице 14.

Таблица 14 – Описание атрибутов сущности «Избранное объявление»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
AnalyzedAdvertisementId	Идентификатор анализируемого сайта	integer(10)	Нет	Нет	FK
UserId	Идентификатор пользователя	integer(10)	Нет	Нет	FK

Описание атрибутов сущности «Пользователи» представлено в таблице 15.

Таблица 15 – Описание атрибутов сущности «Пользователь»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer(10)	Нет	Да	PK
Login	Логин пользователя	varchar(20)	Нет	Да	-
Password	Пароль пользователя	varchar(20)	Нет	Нет	-
RoleId	Идентификатор роли	integer(10)	Нет	Нет	FK
RefreshToken	Токен обновления	varchar(50)	Нет	Нет	-
RefreshTokenExpiryTime	Время истечения токена	timestamp	Нет	Нет	-

Описание атрибутов сущности «Роль» представлено в таблице 16.

Таблица 16 – Описание атрибутов сущности «Роль»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
Id	Идентификатор	integer(10)	Нет	Да	PK
Name	Наименование роли	varchar(20)	Нет	Нет	-

Описание атрибутов сущности «Данные пользователя» представлено в таблице 17.

Таблица 17 – Описание атрибутов сущности «Данные пользователя»

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
UserId	Идентификатор пользователя	integer(10)	Нет	Нет	FK
FirstName	Имя	varchar(30)	Нет	Нет	-

Продолжение таблицы 17

Наименование	Описание	Тип данных	Возможность значения NULL	Unique	Роль
SecondName	Фамилия	varchar(30)	Нет	Нет	-
Patronymic	Отчество	varchar(30)	Нет	Нет	-
BirthDate	Дата рождения	date	Нет	Нет	-

На данной физической модели структура базы данных отображена с учетом спецификации типов СУБД PostgreSQL.

3.2 Разработка серверной части приложения

Разработка серверной части приложения производилась в среде программирования JetBrains Rider на языке C# с применением технологии ASP.NET Web API. Для взаимодействия с базой данных использовалась технология Entity Framework Core [21, 25]. Диаграмма классов серверной части представлена на рисунке 41.

В соответствии с архитектурным шаблоном все классы серверной части можно разделить на три части: модели, предназначенные для описания данных предметной области; сервисы, реализующие бизнес-логику приложения; и контроллеры, для обработки HTTP-запросов к серверу.

Диаграмма классов моделей представлена на рисунке 41. Эти классы были написаны при разработке серверной части приложения, и на основе них была автоматически сгенерирована база данных с использованием подхода Code First.

«Code First» (Сначала код) – это подход к разработке программного обеспечения, в котором сначала создаётся код реализации системы, в частности описываются модели предметной области, а затем на его основе автоматически генерируется соответствующая схема базы данных. Преимущества данного подхода заключаются в его простоте и скорости.

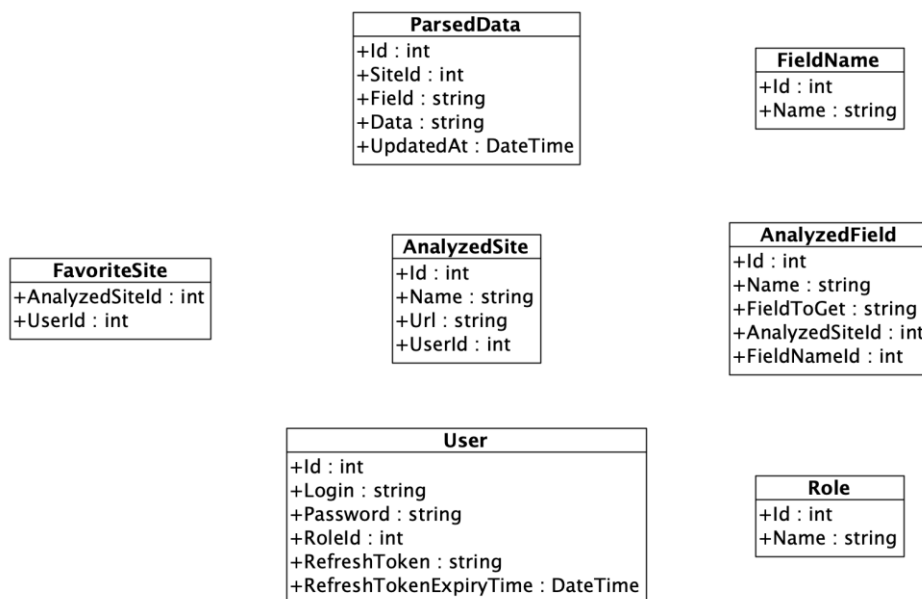


Рисунок 41 – Диаграмма классов серверной части (модели)

Диаграмма классов слоя сервисов представлена на рисунке 42.

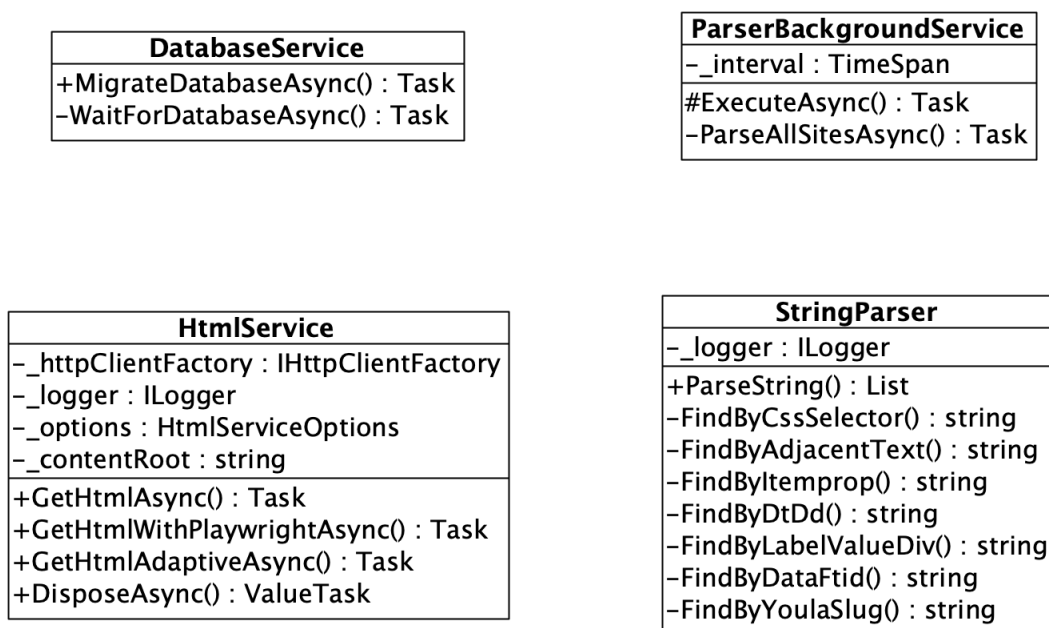


Рисунок 42 – Диаграмма классов серверной части (сервисы)

Диаграмма классов слоя контроллеров представлена на рисунке 43.

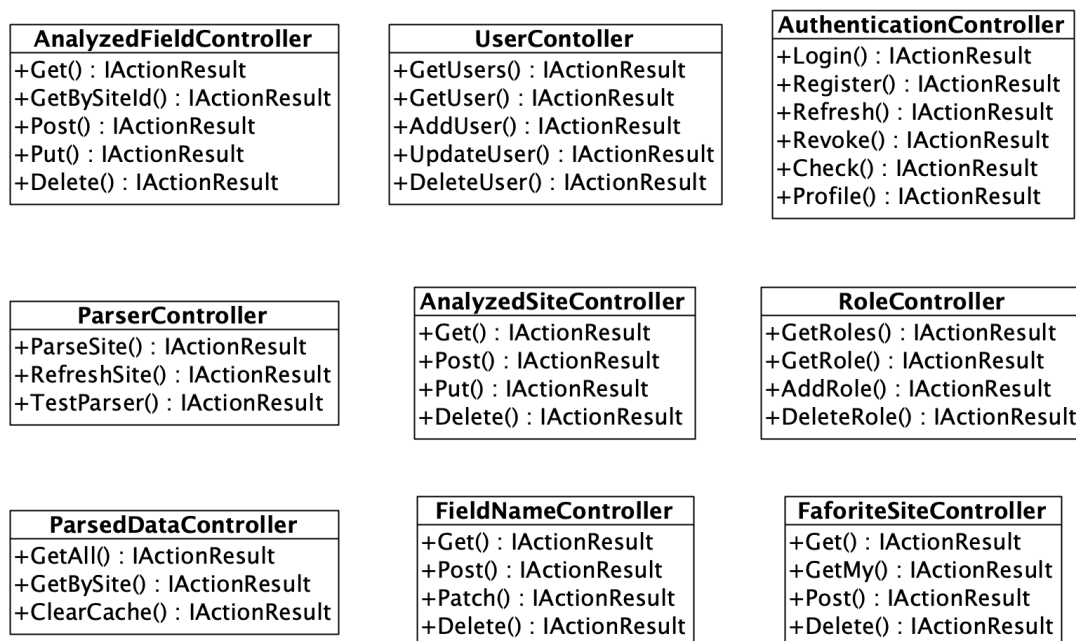


Рисунок 43 – Диаграмма классов серверной части (контроллеры)

Более подробное описание классов слоя сервисов и их назначений представлено в таблице 18.

Таблица 18 – Описание классов сервисов

Класс	Назначение
HtmlService	Отвечает за загрузку HTML-содержимого веб-страниц, в том числе с поддержкой JavaScript-рендеринга посредством библиотеки Playwright.
StringParser	Отвечает за разбор HTML-разметки и извлечение из неё данных по заданным полям.
ParserBackgroundService	Реализует фоновый сервис, выполняющий периодическое (каждые 6 часов) обновление кэша спарсенных данных для всех отслеживаемых сайтов.
DatabaseService	Отвечает за применение миграций и подготовку базы данных при запуске приложения.

Более подробное описание классов слоя контроллеров и их назначений представлено в таблице 19.

Таблица 19 – Описание классов контроллеров

Класс	Назначение
AuthenticationController	Отвечает за обработку запросов по регистрации, аутентификации пользователей и обновлению JWT-токенов.
UserContoller	Отвечает за обработку запросов на получение, добавление и управление пользователями.

Продолжение таблицы 19

Класс	Назначение
RoleController	Отвечает за обработку запросов на получение и управление ролями пользователей.
AnalyzedSiteController	Отвечает за обработку запросов на добавление, получение, редактирование и удаление анализируемых сайтов.
AnalyzedFieldController	Отвечает за обработку запросов на добавление, получение, редактирование и удаление полей, извлекаемых с анализируемых сайтов.
ParsedDataController	Отвечает за обработку запросов на получение спарсенных данных по всем сайтам или по конкретному сайту.
ParserController	Отвечает за обработку запросов на запуск парсинга конкретного сайта с возвратом результата или кэшированных данных.

Листинг серверной части представлен в публичном репозитории GitHub [26].

3.3 Реализация клиентской части приложения

Разработка мобильного приложения производилась на языке TypeScript с использованием фреймворка React Native и платформы Expo [27, 28, 29]. В качестве среды программирования использовался Visual Studio Code.

Диаграмма классов клиентской части приложения представлена на рисунке 44.

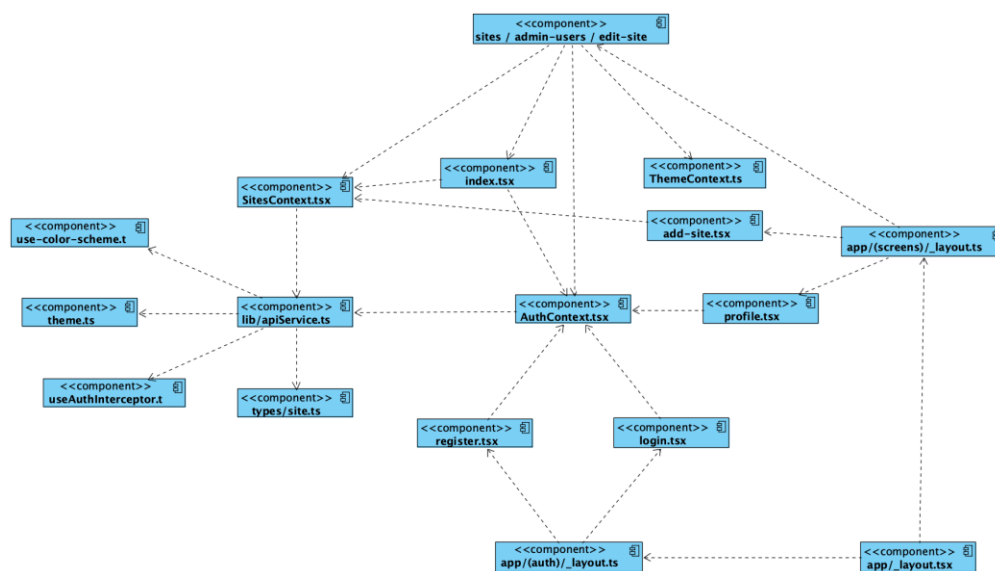


Рисунок 44 – Диаграмма компонентов мобильного приложения

Описание компонентов представлено в таблице 20.

Таблица 20 – Компоненты клиентского приложения

Компонент	Назначение
app/_layout.tsx	Является начальной точкой загрузки приложения, оборачивает всё дерево компонентов в провайдеры AuthProvider, SitesProvider и ThemeProvider
app/(auth)/_layout.tsx	Определяет структуру группы экранов, доступных без токена авторизации
login.tsx	Описывает интерфейс и логику страницы авторизации пользователя
register.tsx	Описывает интерфейс и логику страницы регистрации нового пользователя
app/(screens)/_layout.tsx	Описывает структуру нижней панели вкладок приложения, управляет видимостью вкладок в зависимости от роли пользователя
index.tsx	Описывает интерфейс и логику главного экрана – ленты спарсенных объявлений
add-site.tsx	Описывает интерфейс страницы добавления нового сайта-источника; содержит встроенный WebView с инъекцией JavaScript для визуального выбора CSS-селекторов
profile.tsx	Описывает интерфейс и логику страницы профиля пользователя
sites.tsx	Описывает интерфейс и логику страницы управления сайтами-источниками
AuthContext.tsx	Управляет состоянием аутентификации: хранит JWT-токен в защищённом хранилище expo-secure-store, данные пользователя и его роль, реализует вход и выход из системы
ThemeContext.tsx	Управляет темой оформления приложения, предоставляет переключение между тёмным и светлым режимами
lib/apiService.ts	Реализует все HTTP-запросы к серверу через метод fetchWithTokenRefresh(), обеспечивающий автоматическое обновление токена при получении ответа с кодом 401
useAuthInterceptor.ts	Отслеживает истечение пользовательской сессии и выполняет перенаправление на экран авторизации
use-color-scheme.ts	Определяет системную цветовую схему устройства и передаёт её в ThemeContext
theme.ts	Содержит константы цветовой палитры приложения для светлой и тёмной тем
types/site.ts	Содержит TypeScript-типы, описывающие структуру объектов сайта-источника и спарсированных полей

Диаграмма развертывания, на которой описывается физическое расположение компонентов системы, представлена на рисунке 45.

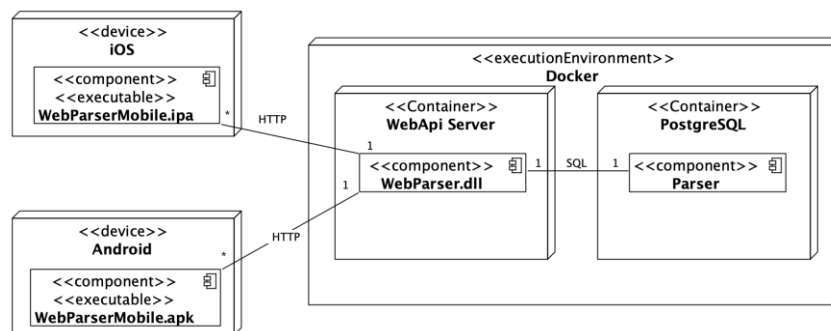


Рисунок 45 – Диаграмма развертывания мобильного приложения

Клиентское приложение существует в двух исполняемых формах: WebParserMobile.ipa для устройств под управлением iOS и WebParserMobile.apk для устройств под управлением Android. API и база данных располагаются на сервере в среде Docker [30]: серверная часть приложения WebParser.dll развёртывается в контейнере WebApi Server, а база данных Parser – в контейнере PostgreSQL. Взаимодействие между клиентской и серверной частями приложения осуществляется посредством протокола HTTP, при этом один экземпляр сервера обслуживает множество клиентских устройств. Взаимодействие между сервером и базой данных происходит при помощи SQL-запросов.

Листинг клиентской части программы представлен в публичном репозитории GitHub [26].

4 Тестирование программного продукта

4.1 Функциональное тестирование черным ящиком

Тестирование методом черного ящика – это вид тестирования, при котором проверяется функциональность приложения, с учетом незнания того, как устроена внутренняя структура и исходный код. Смысл данного подхода подавать на вход различные входные значения, наблюдать за выходными результатами и сравнивать их с ожидаемыми [31].

Рассмотрим данный вид тестирования на примере вводимого поля названия блока, где содержатся данные. Ниже в таблице 21 определены границы для поля название блока, где содержатся данные.

Таблица 21 – Границы тестирования поля названия блока, где содержатся данные

Границы	
Физические	Строка содержит более 0 символов
Логические	Л1 – После «id=» стоит число Л2 – В поле «id=» содержится пробел
Технологические	Максимальный теоретический размер строки
Произвольные	Начинается с «<» и заканчивается «>»; содержит внутри блок id;

Результаты функционального тестирования на примере поля названия блока, где содержатся данные, представлены в таблице 22.

Таблица 22 – Результаты тестирования ввода поля названия блока, где содержатся данные

Границы	Входные данные	Выходные данные
Ф1	Пустая строка	Ошибка – строка пуста
Л1	<div id="1">	Ошибка – невозможно получить данные с парсера
Л2	<div id="dsc sdad">	Ошибка – невозможно получить данные с парсера

Продолжение таблицы 22

Гран ицы	Входные данные	Выходные данные
T1	Строка длины int.MaxValue	Ошибка получения данных с приложения
П1	Блацка	Ошибка – неправильный формат ввода данных

Все тест-кейсы были пройдены успешно, что говорит о корректной работе программы.

4.2 Модульное тестирование

Модульное тестирование – это тестирование, при котором тестируются отдельные модули или компоненты программы. Тестирование проводится изолировано, автоматизировано с использованием заглушек. Цель модульного тестирования заключается в проверке работоспособности отдельного модуля на корректность работы [31].

В данном проекте было принято решение написать модульные тесты для серверной части приложения. Рассмотрим применение данного вида тестирования на примере класса StringParser, код которого представлен в листинге Б.1 в приложении Б, который отвечает за обработку полученных html для получения из них необходимой информации.

Для проведения модульных тестов был разработан тестовый класс StringParserTests, код которого представлен в приложении Д.

Результат проведения модульных тестов представлен на рисунке 46.

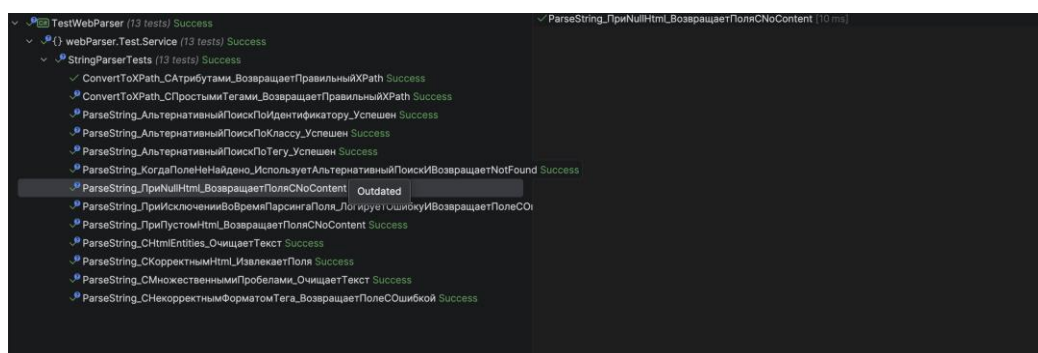


Рисунок 46 – Результаты модульного тестирования

Все тесты были успешно пройдены, что свидетельствует о корректной работе тестируемых методов.

4.3 Метрики кода

Метриками кода называются некоторые количественные величины, позволяющие определить характеристики программного обеспечения и определить сложность, качество и читаемость кода.

Рассмотрим метод «ConvertToXPath», отвечающий за преобразование html блока в строку для последующего анализа полей с его помощью. Его код представлен в листинге 1.

Листинг 1 – Метод ConvertToXPath

```
private string ConvertToXPath(string htmlElement)
{
    try
    {
        //проверка формата html тега
        var tagMatch = Regex.Match(htmlElement, @"^<(\w+)");
        if (!tagMatch.Success)
            throw new ArgumentException("Invalid HTML tag
format");
        //сохранение названия тега
        string tagName = tagMatch.Groups[1].Value;
        //выделение и сохранение атрибутов тега для дальнейшего
поиска
        var attributes = new Dictionary<string, string>();
        var attrMatches = Regex.Matches(htmlElement,
@"(\w+)=[\"'"]([^\\"'"]*)[\"'"]");

        foreach (Match match in attrMatches)
        {
            string attrName = match.Groups[1].Value;
            string attrValue = match.Groups[2].Value;
            attributes[attrName] = attrValue;
        }

        if (attributes.Count == 0)
            return $"//{tagName}";
        //формирование массива атрибутов тегв
        var conditions = new List<string>();
        foreach (var attr in attributes)
        {
            conditions.Add($"@{attr.Key}='{attr.Value}'");
        }
    }
}
```

```

        //формирование результирующей строки
        return $"//{tagName}[{string.Join(" and ", conditions)}]";
    }
    catch (Exception ex)
    {
        _logger.LogError(ex, "Error converting HTML to XPath: {HtmlElement}", htmlElement);
        throw;
    }
}

```

Данный метод находится в серверной части приложения и вызывается при добавлении нового сайта для анализа. Он принимает на вход строку с HTML-тегом, извлекает из неё тег и его атрибуты, а затем формирует XPath-выражение.

Для данного метода будут вычислены две группы метрик: метрики размера программы и метрики Холстеда.

4.3.1 Метрики размера программы

Данные метрики представляют собой подсчет количества строк программного кода, а также числа строк с комментариями. Данный метод легок для оценки, а также хорошо подходит для программных продуктов, которые написаны на одном языке программирования. Результаты вычисления метрик представлены в таблице 23.

Таблица 23 – Метрики размера программы

Метрика	Количество
Количество строк кода	6
Строки-комментарии	38

Также вычислим процент комментариев. Он равен $(5/38) \cdot 100 = 15,789$. Данный показатель указывает на небольшой размер метода, что говорит о том, что данный метод не имеет избыточный или плохо структурированный код.

4.3.2 Метрики Холстеда

Данный метод помогает оценить сложность программы. Полученные, благодаря подсчету, метрики помогают разработчикам понимать и контролировать сложность программного обеспечения, выявлять

потенциальные области для оптимизации и повышать общее качество программного обеспечения. Для вычисления метрик Холстеда, подсчитаем необходимые характеристики программы [32]:

- число уникальных операторов $n1$;
- число уникальных операндов $n2$;
- общее число операторов $N1$;
- общее число операндов $N2$.

Составим словарь операторов и операндов для вычисления метрик. Словарь операторов представлен в таблице 24.

Таблица 24 – Словарь операторов

Операторы	Число вхождений
Regex.Match	1
=	7
!	1
throw	2
_logger.LogError	1
Regex.Matches	1
string.Join	1
logger.LogError	1
in	2
foreach	2
return	2
if	2
.Add	1
{	11
}	11
(	13
)	13
[	4
]	4

Словарь операндов представлен в таблице 25.

Таблица 25 – Словарь операндов

Операнды	Число вхождений
htmlElement	4
tagMatch	2

Продолжение таблицы 25

Операнды	Число вхождений
@'^<(\w+)"	1
"Invalid HTML tag format"	1
@'(\w+)=[''"]([^\''"]*)[''"]"	1
tagName	3
attributes	3
attrMatches	2
match	3
attrName	2
attrValue	2
conditions	2
attr	3
"Error converting HTML to XPath: {HtmlElement}"	1
" and "	1
ex	2

На основе метрик Холстеда можно вычислить целый ряд характеристик программы. Рассмотрим некоторые из них.

Расчёт словаря программы производится по формуле:

$$n = n1 + n2 = 19 + 16 = 35,$$

где $n1$ – количество операторов в программе;

$n2$ – количество операндов программе.

Расчёт длины программы производится по формуле:

$$N = N1 + N2 = 80 + 33 = 113,$$

где $N1$ – число вхождений всех операторов в программе;

$N2$ – число вхождений всех операндов программе.

Расчёт объёма программы производится по формуле:

$$V = N * \log_2 n = 113 * \log_2 35 = 579,6,$$

где N – длина программы;

n – словарь программы.

Расчёт сложности программы производится по формуле:

$$D = (n1/2) * (N2/n2) = (19 / 2) * (33/ 16) = 33,7,$$

где $n1$ – количество операторов в программе;

n_2 – количество операндов программе;

N_2 – число вхождений всех операндов программе.

Расчёт оценки интеллектуальных усилий производится по формуле:

$$E = D * V = 28,5 * 579,6 = 19523,7,$$

где D – сложность программы;

V – объем программы.

Высокое значение сложности $D = 33,7$ обусловлено значительным числом вхождений операндов ($N_2 = 33$) при относительно малом их уникальном количестве ($n_2 = 16$), что свидетельствует об интенсивном переиспользовании переменных в различных контекстах. Объём $V = 579,6$ указывает на высокую информационную насыщенность модуля. Итоговый показатель интеллектуальных усилий $E = 19\,523,7$ является следствием произведения высоких значений D и V , что свидетельствует о значительной трудоёмкости сопровождения, отладки и модификации данного программного модуля.

Заключение

В ходе выполнения выпускной квалификационной работы было разработано мобильное приложение для мониторинга объявлений по продаже автомобилей, агрегирующее данные с нескольких интернет-площадок в едином каталоге.

На этапе анализа были рассмотрены три существующих сервиса аналогичного назначения – «Autospot», «Fresh» и «Автокод Поиск». У каждого выявлены конкретные недостатки: ориентация только на дилеров, платный доступ или неудобный интерфейс. На основе этого анализа сформулированы требования к разрабатываемому продукту и определена целевая аудитория. Также было проведено планирование разработки: общий срок составил 59 рабочих дней, расчётный бюджет – 450 320 рублей. Данный бюджет охватывает затраты на оплату труда разработчиков, инфраструктуру и инструментарий, необходимые для реализации проекта в установленные сроки. Для стартапа или малого бизнеса такой объём финансирования является приемлемым при условии чёткого соблюдения плана разработки и минимизации рисков отклонения от графика, поскольку перерасход ресурсов при фиксированном бюджете данного уровня может поставить под угрозу завершение проекта.

На этапе проектирования выбраны технологии и инструменты: ASP.NET и PostgreSQL для серверной части, React Native с TypeScript для клиентской. Спроектированы концептуальная, логическая и физическая модели базы данных, архитектура сервера по шаблону MVC, а также макеты основных экранов приложения.

На этапе реализации разработана база данных под управлением PostgreSQL, серверная часть на C# с развёртыванием в Docker-контейнере и мобильное приложение на React Native с поддержкой Android 9.0 и выше, а также iOS 18.5 и выше.

В ходе тестирования проведено функциональное тестирование методом чёрного ящика и модульное тестирование серверного класса StringParser. Для

метода `ConvertToXPath` вычислены метрики размера программы и метрики Холстеда. Все выявленные ошибки устранены.

В дальнейшем планируется опубликовать приложение в Google Play и App Store, а серверную часть перевести на облачный хостинг для повышения доступности сервиса.

Список использованных источников

1. Распоряжение Правительства Российской Федерации от 28 июля 2017 г. № 1632-р «Об утверждении программы «Цифровая экономика Российской Федерации»». [Электронный ресурс]. URL: <http://static.government.ru/media/files/9gFM4FHj4PsB79I5v7yLVuPgu4bvR7M0.pdf> (дата обращения: 19.03.2026).
2. Указ Президента Российской Федерации от 9 мая 2017 г. № 203 «О Стратегии развития информационного общества в Российской Федерации на 2017–2030 годы». [Электронный ресурс]. URL: http://zmedu.hostedu.ru/files/ykaz_7668.pdf (дата обращения: 19.03.2026).
3. Годовой отчет ПАО «МТС» за 2025 год. Раздел «Ключевые операционные показатели». [Электронный ресурс] // Официальный сайт ПАО «МТС» для инвесторов. – URL: <https://ir.mts.ru/> (дата обращения: 19.03.2026).
4. Информация о компании Autospot. [Электронный ресурс] // Официальный сайт Autospot. – URL: <https://autospot.ru/about/> (дата обращения: 19.03.2026).
5. Информация о маркетплейсе Fresh. [Электронный ресурс] // Официальный сайт Fresh. – URL: <https://freshauto.ru/o-kompanii/> (дата обращения: 19.03.2026).
6. Информация о сервисе «Автокод Поиск». [Электронный ресурс] // Официальный сайт «Автокод». – URL: <https://avtocod.ru/o-servise> (дата обращения: 19.03.2026).
7. Черeda сюрпризов. Семь главных автотрендов в феврале 2026 года. [Электронный ресурс] // RBC Autonews. – 2026. – 3 марта. – URL: <https://www.rbcautonews.ru/news/69a7ef929a79473260d86e46> (дата обращения: 19.03.2026).
8. Продажи новых легковых автомобилей перешли к стагнации. [Электронный ресурс] // Ведомости. – 2026. – 4 марта. – URL: <https://www.vedomosti.ru/business/articles/2026/03/04/1180530-prodazhi-novih-legkovih-avtomobilei-pereshli-k-stagnatsii> (дата обращения: 19.03.2026).

9. Статистика доли мобильных операционных систем на мировом рынке. [Электронный ресурс] // Statcounter Global Stats. – URL: <https://gs.statcounter.com/os-market-share/mobile/worldwide> (дата обращения: 19.03.2026).

10. Статистика доли мобильных операционных систем на российском рынке. [Электронный ресурс] // Statcounter Global Stats. – URL: <https://gs.statcounter.com/os-market-share/mobile/russian-federation> (дата обращения: 19.03.2026).

11. Унифицированный язык моделирования (UML). [Электронный ресурс] // Материал из Википедии – свободной энциклопедии. – URL: <https://ru.wikipedia.org/wiki/UML> (дата обращения: 19.03.2026).

12. Ананьева, В. Я. UML-диаграммы. Основные возможности: учебно-методическое пособие / В. Я. Ананьева; Санкт-Петербургский государственный электротехнический университет «ЛЭТИ». – Санкт-Петербург: Изд-во СПбГЭТУ «ЛЭТИ», 2023. – 28 с. – ISBN 978-5-7629-3117-5. [Электронный ресурс]. – URL: <http://letiizdat.ru/izdaniya/uchebno-metodicheskaya-literatura/metodicheskie-ukazaniya/2023/uml-diagrammy.-osnovnye-vozmozhnosti> (дата обращения: 19.03.2026).

13. Проектирование диаграммы деятельности UML (Activity Diagram). [Электронный ресурс] // Национальная сборная Worldskills Россия / Агентство развития профессионального мастерства. – URL: <https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-deyatelnosti-uml-activity-diagram/> (дата обращения: 19.03.2026).

14. Проектирование диаграммы последовательности UML (Sequence Diagram). [Электронный ресурс] // Национальная сборная Worldskills Россия / Агентство развития профессионального мастерства. – URL: <https://nationalteam.worldskills.ru/skills/proektirovanie-diagrammy-posledovatelnosti-uml-sequence-diagram/> (дата обращения: 19.03.2026).

15. RUP (Rational Unified Process). [Электронный ресурс] // AppMaster Glossary. – 2020. – URL: <https://appmaster.io/ru/glossary/rup-rational-unified-process-ru> (дата обращения: 19.03.2026).

16. Что такое MVC (Model-View-Controller)? [Электронный ресурс] // Хекслет. – 2023. URL: <https://ru.hexlet.io/qna/glossary/questions/chto-takoe-mvc> (дата обращения: 19.03.2026).

17. Visual Paradigm Community Edition. Руководство пользователя. [Электронный ресурс] // Visual Paradigm. – URL: <https://www.visual-paradigm.com/support/> (дата обращения: 07.05.2026).

18. Руководство к своду знаний по управлению проектами (Руководство PMBOK). – 6-е изд. – Newtown Square: Project Management Institute, 2017. – 756 с.

19. Документация по Microsoft Project. [Электронный ресурс] // Microsoft Learn. – URL: <https://learn.microsoft.com/ru-ru/project/> (дата обращения: 07.05.2026).

20. Филдинг, Р. Т. Архитектурные стили и проектирование сетевых архитектур программного обеспечения / Р. Т. Филдинг; пер. с англ. – Диссертация, 2000. [Электронный ресурс]. – URL: https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm (дата обращения: 07.05.2026).

21. Документация по Entity Framework Core. [Электронный ресурс] // Microsoft Learn. – URL: <https://learn.microsoft.com/ru-ru/ef/core/> (дата обращения: 07.05.2026).

22. Трехуровневая архитектура клиент-серверного приложения. [Электронный ресурс] // Sky.pro. – URL: <https://sky.pro/wiki/sql/trehurovnevaya-klient-servernaya-arhitektura/> (дата обращения: 07.05.2026).

23. Спецификация JSON (JavaScript Object Notation). [Электронный ресурс] // ECMA International. – URL: <https://www.ecma-international.org/publications-and-standards/standards/ecma-404/> (дата обращения: 07.05.2026).

24. Документация по PostgreSQL 16. [Электронный ресурс] // The PostgreSQL Global Development Group. – URL: <https://www.postgresql.org/docs/16/> (дата обращения: 07.05.2026).

25. Документация по ASP.NET Core. [Электронный ресурс] // Microsoft Learn. – URL: <https://learn.microsoft.com/ru-ru/aspnet/core/> (дата обращения: 07.05.2026).

26. GitHub репозиторий. [Электронный ресурс] // Microsoft. – URL: <https://github.com/Zhivov/AutoParser.git/> (дата обращения: 01.06.2026).

27. React Native. Документация фреймворка. [Электронный ресурс] // Meta Platforms, Inc. – URL: <https://reactnative.dev/docs/getting-started> (дата обращения: 07.05.2026).

28. Expo. Документация платформы. [Электронный ресурс] // Expo. – URL: <https://docs.expo.dev/> (дата обращения: 07.05.2026).

29. Документация по TypeScript. [Электронный ресурс] // Microsoft. – URL: <https://www.typescriptlang.org/docs/> (дата обращения: 07.05.2026).

30. Документация по Docker. [Электронный ресурс] // Docker, Inc. – URL: <https://docs.docker.com/> (дата обращения: 07.05.2026).

31. Котляров, В. П. Основы тестирования программного обеспечения: учебное пособие / В. П. Котляров, Т. В. Коликова. – Москва: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2006. – 285 с. – ISBN 5-94774-522-4.

32. Halstead, M. H. Elements of Software Science / M. H. Halstead. – New York: Elsevier North-Holland, 1977. – 127 p.

Приложение А
Глоссарий
(справочное)

Android – мобильная операционная система для смартфонов и планшетов, разрабатываемая компанией Google и используемая на подавляющем большинстве современных мобильных устройств.

API (Application Programming Interface) – набор правил и протоколов, обеспечивающий взаимодействие между клиентским приложением и серверной частью системы посредством передачи структурированных запросов и ответов.

CSS-селектор – шаблон, используемый для выбора HTML-элементов на веб-странице по их атрибутам, классам, идентификаторам или положению в структуре документа; в данном приложении применяется для указания элементов, данные которых необходимо извлечь при парсинге.

Docker – платформа контейнеризации, позволяющая упаковывать приложение и все его зависимости в изолированный контейнер, обеспечивая воспроизводимое развёртывание на любом сервере.

Expo – платформа и набор инструментов для разработки мобильных приложений на React Native, упрощающая сборку, тестирование и публикацию приложений для iOS и Android.

HTTP (HyperText Transfer Protocol) – протокол прикладного уровня передачи данных, основанный на технологии «клиент-сервер» и используемый для обмена данными между мобильным приложением и серверным API.

iOS – мобильная операционная система компании Apple, применяемая на смартфонах iPhone и планшетах iPad.

JWT (JSON Web Token) – компактный, самодостаточный токен в формате JSON, используемый для безопасной передачи информации об аутентификации между клиентом и сервером; в данном приложении применяется для авторизации запросов к API.

PostgreSQL – свободная объектно-реляционная система управления базами данных с открытым исходным кодом, используемая в данной работе для хранения пользовательских данных, информации о сайтах-источниках и спарсированных объявлениях.

React Native – кроссплатформенный фреймворк с открытым исходным кодом от компании Meta, позволяющий разрабатывать мобильные приложения для iOS и Android на языке JavaScript и TypeScript с использованием нативных компонентов интерфейса.

SQL (Structured Query Language) – декларативный язык программирования, применяемый для создания, изменения и управления данными в реляционных базах данных.

TypeScript – строго типизированный язык программирования, являющийся надмножеством JavaScript; использован при разработке клиентской части приложения для повышения надёжности и сопровождаемости кода.

UML (Unified Modeling Language) – язык графического описания для объектного моделирования в области разработки программного обеспечения, использованный при проектировании архитектуры системы.

WebView – встроенный компонент мобильного приложения, отображающий веб-страницу внутри экрана приложения; в данной работе применяется для визуального выбора CSS-селекторов на страницах анализируемых сайтов.

Агрегатор – информационная система, собирающая и объединяющая данные из множества независимых источников в единый каталог для удобства просмотра и сравнения пользователем.

Аутентификация – процесс проверки подлинности учётных данных пользователя (логина и пароля) при входе в систему.

Авторизация – процесс предоставления пользователю прав доступа к функционалу системы на основании его роли (пользователь, редактор, администратор).

Избранное – персональный список сайтов-источников, отмеченных пользователем для быстрого доступа к их объявлениям.

Парсинг – автоматизированный процесс извлечения структурированных данных из HTML-содержимого веб-страниц с использованием заранее настроенных CSS-селекторов.

Токен обновления (Refresh Token) – долгоживущий токен, используемый для получения нового токена доступа без повторного ввода логина и пароля пользователем.

Приложение Б
Исходный код серверной части
(обязательное)

Листинг Б.1 – Файл Program.cs

```

using System.IdentityModel.Tokens.Jwt;
using System.Reflection;
using System.Security.Claims;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.OpenApi;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using webParser.config;
using webParser.Data;
using webParser.Service;

var builder = WebApplication.CreateBuilder(args);
builder.Services.AddCors(options =>
{
    options.AddPolicy("Mobile",
        policy =>
        {
            policy.WithOrigins(
                "http://localhost:8081",
                "http://192.168.31.54:8081",
                "http://192.168.31.200:8081")
                .AllowAnyHeader()
                .AllowAnyMethod()
                .AllowCredentials();
        });
});

Environment.SetEnvironmentVariable("PLAYWRIGHT_BROWSERS_PATH",
"/app/bin/.playwright");
Environment.SetEnvironmentVariable("DOTNET_PLAYWRIGHT_PATH",
"/app/bin/.playwright");
builder.Services.Configure<HtmlServiceOptions>(builder.Configuration.
GetSection("HtmlService"));
builder.Services.AddSingleton<HtmlService>();

builder.Services.Configure<HtmlServiceOptions>(
    builder.Configuration.GetSection("HtmlService"));

builder.Services.AddHttpClient();
builder.Services.AddSingleton<HtmlService>();
Console.WriteLine("Playwright browsers path: " +
Environment.GetEnvironmentVariable("PLAYWRIGHT_BROWSERS_PATH"));

builder.Configuration.AddEnvironmentVariables();
builder.Services.AddAuthorization();
builder.Services.AddAuthentication(JwtBearerDefaults.Authentication
nScheme)
    .AddJwtBearer(options =>
    {

```

```

        options.TokenValidationParameters = new
TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidIssuer = AuthOptions.ISSUER,
            ValidateAudience = true,
            ValidAudience = AuthOptions.AUDIENCE,
            ValidateLifetime = true,
            IssuerSigningKey =
AuthOptions.GetSymmetricSecurityKey(),
            ValidateIssuerSigningKey = true,
        };
    });
builder.Services.AddDbContext<AppDbContext>(options =>

options.UseNpgsql(builder.Configuration.GetConnectionString("DefaultConnection")));
builder.Services.AddScoped<HtmlService>();
builder.Services.AddScoped<StringParser>();
builder.Services.AddHostedService<ParserBackgroundService>();
builder.Services.AddControllers().AddJsonOptions(options =>
{
    options.JsonSerializerOptions.PropertyNamingPolicy = null;
    options.JsonSerializerOptions.WriteIndented = true;
});
builder.Services.AddHttpClient();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new OpenApiInfo
    {
        Version = "v1",
        Title = "WebParser API",
        Description = "Web Parser API"
    });

    c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = SecuritySchemeType.Http,
        Scheme = "Bearer",
        BearerFormat = "JWT",
        In = ParameterLocation.Header,
        Description = "Enter JWT token"
    });

    c.AddSecurityRequirement(document => new() { [new
OpenApiSecuritySchemeReference("Bearer", document)] = [] });
});

var app = builder.Build();

```

```

app.UseCors("Mobile");
using (var scope = app.Services.CreateScope())
{
    var db =
scope.ServiceProvider.GetRequiredService<AppDbContext>();
    db.Database.Migrate();

    if (!db.AnalyzedSites.Any())
    {
        db.AnalyzedSites.AddRange(
            new webParser.Models.Database.AnalyzedSite { Id = 1,
Name = "Toyota Camry 2018", Url = "https://example.com/camry-
2018", UserId = -1 },
            new webParser.Models.Database.AnalyzedSite { Id = 2,
Name = "BMW 3 Series 2020", Url = "https://example.com/bmw-3-
2020", UserId = -1 },
            new webParser.Models.Database.AnalyzedSite { Id = 3,
Name = "Hyundai Tucson 2019", Url = "https://example.com/tucson-
2019", UserId = -1 },
            new webParser.Models.Database.AnalyzedSite { Id = 4,
Name = "Kia Sportage 2021", Url = "https://example.com/sportage-
2021", UserId = -1 },
            new webParser.Models.Database.AnalyzedSite { Id = 5,
Name = "Volkswagen Passat 2017", Url =
"https://example.com/passat-2017", UserId = -1 }
        );
        db.SaveChanges();

        db.AnalyzedFields.AddRange(
            new webParser.Models.Database.AnalyzedField { Name =
"Цена", FieldToGet = "1 250 000 ₽", AnalyzedSiteId = 1,
FieldNameId = 1 },
            new webParser.Models.Database.AnalyzedField { Name =
"Год выпуска", FieldToGet = "2018", AnalyzedSiteId = 1,
FieldNameId = 2 },
            new webParser.Models.Database.AnalyzedField { Name =
"Марка", FieldToGet = "Toyota", AnalyzedSiteId = 1, FieldNameId =
3 },
            new webParser.Models.Database.AnalyzedField { Name =
"Модель", FieldToGet = "Camry", AnalyzedSiteId = 1, FieldNameId =
4 },
            new webParser.Models.Database.AnalyzedField { Name =
"Пробег", FieldToGet = "78 000 км", AnalyzedSiteId = 1,
FieldNameId = 5 },
            new webParser.Models.Database.AnalyzedField { Name =
"Тип топлива", FieldToGet = "Бензин", AnalyzedSiteId = 1,
FieldNameId = 8 },

            new webParser.Models.Database.AnalyzedField { Name =
"Цена", FieldToGet = "3 800 000 ₽", AnalyzedSiteId = 2,
FieldNameId = 1 },

```

```

        new webParser.Models.Database.AnalyzedField { Name =
"Год выпуска", FieldToGet = "2020", AnalyzedSiteId = 2,
FieldNameId = 2 },
        new webParser.Models.Database.AnalyzedField { Name =
"Марка", FieldToGet = "BMW", AnalyzedSiteId = 2, FieldNameId = 3
},
        new webParser.Models.Database.AnalyzedField { Name =
"Модель", FieldToGet = "3 Series", AnalyzedSiteId = 2, FieldNameId
= 4 },
        new webParser.Models.Database.AnalyzedField { Name =
"Мощность двигателя", FieldToGet = "184 л.с.", AnalyzedSiteId = 2,
FieldNameId = 6 },
        new webParser.Models.Database.AnalyzedField { Name =
"Коробка передач", FieldToGet = "Автомат", AnalyzedSiteId = 2,
FieldNameId = 9 },

        new webParser.Models.Database.AnalyzedField { Name =
"Цена", FieldToGet = "2 050 000 ₽", AnalyzedSiteId = 3,
FieldNameId = 1 },
        new webParser.Models.Database.AnalyzedField { Name =
"Год выпуска", FieldToGet = "2019", AnalyzedSiteId = 3,
FieldNameId = 2 },
        new webParser.Models.Database.AnalyzedField { Name =
"Марка", FieldToGet = "Hyundai", AnalyzedSiteId = 3, FieldNameId =
3 },
        new webParser.Models.Database.AnalyzedField { Name =
"Модель", FieldToGet = "Tucson", AnalyzedSiteId = 3, FieldNameId =
4 },
        new webParser.Models.Database.AnalyzedField { Name =
"Привод", FieldToGet = "Полный", AnalyzedSiteId = 3, FieldNameId =
10 },
        new webParser.Models.Database.AnalyzedField { Name =
"Цвет", FieldToGet = "Белый", AnalyzedSiteId = 3, FieldNameId = 11
},

        new webParser.Models.Database.AnalyzedField { Name =
"Цена", FieldToGet = "2 450 000 ₽", AnalyzedSiteId = 4,
FieldNameId = 1 },
        new webParser.Models.Database.AnalyzedField { Name =
"Год выпуска", FieldToGet = "2021", AnalyzedSiteId = 4,
FieldNameId = 2 },
        new webParser.Models.Database.AnalyzedField { Name =
"Марка", FieldToGet = "Kia", AnalyzedSiteId = 4, FieldNameId = 3
},
        new webParser.Models.Database.AnalyzedField { Name =
"Модель", FieldToGet = "Sportage", AnalyzedSiteId = 4, FieldNameId
= 4 },
        new webParser.Models.Database.AnalyzedField { Name =
"Кузов", FieldToGet = "SUV", AnalyzedSiteId = 4, FieldNameId = 12
},

```

```

        new webParser.Models.Database.AnalyzedField { Name =
"Тип топлива", FieldToGet = "Дизель", AnalyzedSiteId = 4,
FieldNameId = 8 },

        new webParser.Models.Database.AnalyzedField { Name =
"Цена", FieldToGet = "1 650 000 ₽", AnalyzedSiteId = 5,
FieldNameId = 1 },
        new webParser.Models.Database.AnalyzedField { Name =
"Год выпуска", FieldToGet = "2017", AnalyzedSiteId = 5,
FieldNameId = 2 },
        new webParser.Models.Database.AnalyzedField { Name =
"Марка", FieldToGet = "Volkswagen", AnalyzedSiteId = 5,
FieldNameId = 3 },
        new webParser.Models.Database.AnalyzedField { Name =
"Модель", FieldToGet = "Passat", AnalyzedSiteId = 5, FieldNameId =
4 },
        new webParser.Models.Database.AnalyzedField { Name =
"Пробег", FieldToGet = "112 000 км", AnalyzedSiteId = 5,
FieldNameId = 5 },
        new webParser.Models.Database.AnalyzedField { Name =
"Объём двигателя", FieldToGet = "1.8 л", AnalyzedSiteId = 5,
FieldNameId = 7 }
    );
    db.SaveChanges();
}
}
var configuration = builder.Configuration;

if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI(options =>
    {
        options.SwaggerEndpoint("/swagger/v1/swagger.json", "v1");
        options.RoutePrefix = String.Empty;

        options.OAuthClientId("swagger-ui");
        options.OAuthAppName("Swagger UI");
        options.OAuthClientId("swagger-ui");
        options.OAuthAppName("Swagger UI");
        options.OAuthUsePkce();
        options.EnablePersistAuthorization();

        options.OAuth2RedirectUrl($"{configuration["Swagger:RedirectUrl"]}
/swagger/oauth2-redirect.html");

        options.ConfigObject.AdditionalItems.Add("requestInterceptor",
"(req) => { console.log('Request:', req); return req; }");

        options.EnablePersistAuthorization();
        options.DefaultModelsExpandDepth(-1);
    }
}

```

```

        });
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        app.UseHsts();
    }

    app.UseHttpsRedirection();
    app.UseRouting();

    app.UseAuthentication();
    app.UseAuthorization();

    app.MapStaticAssets();
    app.MapSwagger();
    app.MapControllerRoute(
        name: "default",
        pattern: "{controller=Home}/{action=Index}/{id?}")
        .WithStaticAssets();

    app.Run();

```

Листинг Б.2 – Файл config/AuthOptions.cs

```

using System.Text;
using Microsoft.IdentityModel.Tokens;

namespace webParser.config;

public class AuthOptions
{
    public const string ISSUER = "WebParser"; // издатель токена
    public const string AUDIENCE = "WebParser"; // потребитель токена
    const string KEY = "mysupersecret_secretsecretsecretkey!123";
    // ключ для шифрации
    public const int ACCESS_TOKEN_LIFETIME_MINUTES = 15;
    public const int REFRESH_TOKEN_LIFETIME_DAYS = 7;

    public static SymmetricSecurityKey GetSymmetricSecurityKey()
    {
        return new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(KEY));
    }
}

```

Листинг Б.3 – Файл config/HtmlServiceOptions.cs

```

namespace webParser.config;

public class HtmlServiceOptions

```

```

{
    public int NavigationTimeoutMs { get; set; } = 60000;
    public int NetworkIdleTimeoutMs { get; set; } = 30000;
    public int AdditionalWaitMs { get; set; } = 2000;
    public int HttpTimeoutSeconds { get; set; } = 30;
    public int MinHtmlLengthForHttp { get; set; } = 3000;
    public string[] JsHeavySites { get; set; } = { "auto.ru",
"drom.ru", "avito.ru", "sberauto.com", "auto.drom.ru" };
    public bool ForcePlaywrightForJsSites { get; set; } = true;
    public string ProxyListPath { get; set; } = "config/Webshare
10 proxies.txt";

    public int RateLimitMinDelayMs { get; set; } = 5000;
    public int RateLimitMaxDelayMs { get; set; } = 8000;
    public int RetryDelayOnRateLimitMs { get; set; } = 30000;
}

```

Листинг Б.4 – Файл Data/AppDbContext.cs

```

using Microsoft.EntityFrameworkCore;
using webParser.Models.Database;

namespace webParser.Data;

public class AppDbContext : DbContext
{
    public AppDbContext(DbContextOptions<AppDbContext> options) :
base(options) { }

    public DbSet<Role> Roles => Set<Role>();
    public DbSet<User> Users => Set<User>();
    public DbSet<AnalyzedSite> AnalyzedSites =>
Set<AnalyzedSite>();

    public DbSet<FavoriteSite> FavoriteSites =>
Set<FavoriteSite>();

    public DbSet<AnalyzedField> AnalyzedFields =>
Set<AnalyzedField>();
    public DbSet<FieldName> FieldNames => Set<FieldName>();
    public DbSet<ParsedData> ParsedData => Set<ParsedData>();
    protected override void OnModelCreating(ModelBuilder
modelBuilder)
    {
        modelBuilder.Entity<User>(entity =>
        {
            entity.HasKey(e => e.Id);
            entity.Property(e => e.Login).IsRequired();
            entity.Property(e => e.Password).IsRequired();

            entity.HasOne<Role>()
                .WithMany()
                .HasForeignKey(u => u.RoleId)

```

```

        .onDelete(DeleteBehavior.Restrict);
    });

modelBuilder.Entity<Role>(entity =>
{
    entity.HasKey(e => e.Id);
    entity.Property(e => e.Name).IsRequired();
});

modelBuilder.Entity<AnalyzedSite>(entity =>
{
    entity.HasKey(e => e.Id);
    entity.Property(e => e.Url).IsRequired();

    entity.HasOne<User>()
        .WithMany()
        .HasForeignKey(a => a.UserId)
        .onDelete(DeleteBehavior.Restrict);
});

modelBuilder.Entity<FieldName>(entity =>
{
    entity.HasKey(e => e.Id);
    entity.Property(e =>
e.Name).IsRequired().HasMaxLength(40);
});

modelBuilder.Entity<ParsedData>(entity =>
{
    entity.HasKey(e => e.Id);
    entity.Property(e => e.Field).IsRequired();
    entity.Property(e => e.Data).IsRequired();
    entity.HasOne<AnalyzedSite>()
        .WithMany()
        .HasForeignKey(p => p.SiteId)
        .onDelete(DeleteBehavior.Cascade);
});

modelBuilder.Entity<AnalyzedField>(entity =>
{
    entity.HasKey(e => e.Id);
    entity.Property(e => e.Name).IsRequired();
    entity.Property(e => e.FieldToGet).IsRequired();

    entity.HasOne<AnalyzedSite>()
        .WithMany()
        .HasForeignKey(a => a.AnalyzedSiteId)
        .onDelete(DeleteBehavior.Cascade);

    entity.HasOne<FieldName>()
        .WithMany()
        .HasForeignKey(a => a.FieldNameId)

```

```

        .OnDelete(DeleteBehavior.SetNull)
        .IsRequired(false);
    });

modelBuilder.Entity<FavoriteSite>(entity =>
{
    entity.HasKey(e => new { e.AnalyzedSiteId, e.UserId

entity.HasOne<AnalyzedSite>()
    .WithMany()
    .HasForeignKey(f => f.AnalyzedSiteId)
    .OnDelete(DeleteBehavior.Cascade);

entity.HasOne<User>()
    .WithMany()
    .HasForeignKey(f => f.UserId)
    .OnDelete(DeleteBehavior.Cascade);
});
modelBuilder.Entity<Role>().HasData(
    new Role
    {
        Id = 1,
        Name = "Пользователь",
    },
    new Role
    {
        Id = 2,
        Name = "Редактор",
    },
    new Role
    {
        Id = 3,
        Name = "Администратор",
    }
);
modelBuilder.Entity<User>().HasData(
    new User
    {
        Id = -1,
        Login = "admin",
        Password = "admin",
        RoleId = 3
    }
);
modelBuilder.Entity<FieldName>().HasData(
    new FieldName { Id = 1, Name = "Цена" },
    new FieldName { Id = 2, Name = "Год выпуска" },
    new FieldName { Id = 3, Name = "Марка" },
    new FieldName { Id = 4, Name = "Модель" },
    new FieldName { Id = 5, Name = "Пробег" },
    new FieldName { Id = 6, Name = "Мощность двигателя" },

```

```

        new FieldName { Id = 7, Name = "Объём двигателя" },
        new FieldName { Id = 8, Name = "Тип топлива" },
        new FieldName { Id = 9, Name = "Коробка передач" },
        new FieldName { Id = 10, Name = "Привод" },
        new FieldName { Id = 11, Name = "Цвет" },
        new FieldName { Id = 12, Name = "Кузов" }
    );
}
}

```

Листинг Б.5 – Файл Models/Database/User.cs

```

using System.ComponentModel.DataAnnotations.Schema;

namespace webParser.Models.Database;

public class User
{
    public int Id { get; set; }
    public required string Login { get; set; }
    public required string Password { get; set; }
    [ForeignKey("Role")]
    public required int RoleId { get; set; }
    public string? RefreshToken { get; set; }
    public DateTime? RefreshTokenExpiryTime { get; set; }
}

```

Листинг Б.6 – Файл Models/Database/Role.cs

```

namespace webParser.Models.Database;

public class Role
{
    public int Id { get; set; }
    public required string Name { get; set; }
}

```

Листинг Б.7 – Файл Models/Database/AnalyzedSite.cs

```

using System.ComponentModel.DataAnnotations.Schema;

namespace webParser.Models.Database;

public class AnalyzedSite
{
    public int Id { get; set; }
    public required string Name { get; set; }
    public required string Url { get; set; }
    [ForeignKey("User")]
    public required int UserId { get; set; }
}

```

Листинг Б.8 – Файл Models/Database/AnalyzedField.cs

```

using System.ComponentModel.DataAnnotations.Schema;

namespace webParser.Models.Database;

public class AnalyzedField
{
    public int Id { get; set; }
    public required string Name { get; set; }
    public required string FieldToGet { get; set; }
    [ForeignKey("AnalyzedSite")]
    public required int AnalyzedSiteId { get; set; }
    [ForeignKey("FieldName")]
    public int FieldNameId { get; set; }
}

```

Листинг Б.9 – Файл Models/Database/FavoriteSite.cs

```

using System.ComponentModel.DataAnnotations.Schema;

namespace webParser.Models.Database;

public class FavoriteSite
{
    [ForeignKey("AnalyzedSite")]
    public required int AnalyzedSiteId { get; set; }
    [ForeignKey("User")]
    public required int UserId { get; set; }
}

```

Листинг Б.10 – Файл Models/Database/FieldName.cs

```

namespace webParser.Models.Database;

public class FieldName
{
    public int Id { get; set; }
    public required string Name { get; set; }
}

```

Листинг Б.11 – Файл Models/Database/ParsedData.cs

```

using System.ComponentModel.DataAnnotations.Schema;

namespace webParser.Models.Database;

public class ParsedData
{
    public int Id { get; set; }
    [ForeignKey("AnalyzedSite")]
    public required int SiteId { get; set; }
    public required string Field { get; set; }
}

```

```

        public required string Data { get; set; }
        public DateTime UpdatedAt { get; set; } = DateTime.UtcNow;
    }

```

Листинг Б.12 – Файл Models/DTO/DataField.cs

```

namespace webParser.Models.DTO;

public class DataField
{
    public required string Data { get; set; }
    public required string Field { get; set; }
}

```

Листинг Б.13 – Файл Models/DTO/TokenDto.cs

```

namespace webParser.Models.DTO;

public class TokenDto
{
    public string AccessToken { get; set; }
    public string RefreshToken { get; set; }
    public string? Username { get; set; }
}

```

Листинг Б.14 – Файл Models/DTO/FavoriteSiteDto.cs

```

namespace webParser.Models.DTO;

public class FavoriteSiteDto
{
    public required int AnalyzedSiteId { get; set; }
}

```

Листинг Б.15 – Файл Models/DTO/RoleDTO.cs

```

namespace webParser.Models.DTO;

public class RoleDto
{
    public required int Id { get; set; }
    public required string Name { get; set; }
    public required string Role { get; set; }
}

```

Листинг Б.16 – Файл Models/DTO/user/UserDTO.cs

```

namespace webParser.Models.DTO.user;

public class UserDto
{
    public required string Login { get; set; }
    public required string Password { get; set; }
}

```

```
}
```

Листинг Б.17 – Файл Models/DTO/user/ProfileUserDto.cs

```
namespace webParser.Models.DTO.user;

public class ProfileUserDto
{
    public required string Login { get; set; }
    public required string Role { get; set; }
}

```

Листинг Б.18 – Файл Models/DTO/user/UpdateUserDto.cs

```
namespace webParser.Models.DTO.user;

public class UpdateUserDto
{
    public int Id { get; set; }
    public string? Login { get; set; }
    public string? Password { get; set; }
    public int? RoleId { get; set; }
}

```

Листинг Б.19 – Файл Models/DTO/AnalyzedSite/CreateAnalyzedSiteDto.cs

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

namespace webParser.Models.DTO.AnalyzedSite;

public class CreateAnalyzedSiteDto
{
    public required string Name { get; set; }
    public required string Url { get; set; }
}

```

Листинг Б.20 – Файл Models/DTO/AnalyzedField/CreateAnalyzedFieldDto.cs

```
namespace webParser.Models.DTO.AnalyzedField;

public class CreateAnalyzedFieldDto
{
    public required string Name { get; set; }
    public required string FieldToGet { get; set; }
    public required int AnalyzedSiteId { get; set; }
    public int FieldNameId { get; set; }
}

```

Листинг Б.21 – Файл Models/DTO/AnalyzedField/UpdateAnalyzedFieldDto.cs

```
namespace webParser.Models.DTO.AnalyzedField;
```

```
public class UpdateAnalyzedFieldDto
{
    public string? Name { get; set; }
    public string? FieldToGet { get; set; }
    public int? AnalyzedSiteId { get; set; }
    public int? FieldNameId { get; set; }
}
```

Листинг Б.22 – Файл Models/DTO/FieldName/CreateFieldNameDto.cs

```
namespace webParser.Models.DTO.FieldName;

public class CreateFieldNameDto
{
    public required string Name { get; set; }
}
```

Листинг Б.23 – Файл Models/DTO/FieldName/UpdateFieldNameDto.cs

```
namespace webParser.Models.DTO.FieldName;

public class UpdateFieldNameDto
{
    public string? Name { get; set; }
}
```

Листинг Б.24 – Файл Controllers/AuthenticationController.cs

```
using System.IdentityModel.Tokens.Jwt;
using System.Security.Claims;
using System.Security.Cryptography;
using System.Text.Json;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.IdentityModel.Tokens;
using webParser.config;
using webParser.Data;
using webParser.Models.Database;
using webParser.Models.DTO;
using webParser.Models.DTO.user;

namespace webParser.Controllers;
[AllowAnonymous]
[Route("api/[controller]")]
[ApiController]
public class
AuthenticationController(ILogger<AuthenticationController> logger,
AppDbContext context) : Controller
{
    private string GenerateRefreshToken()
```

```

    {
        var randomNumber = new byte[64];
        using var rng = RandomNumberGenerator.Create();
        rng.GetBytes(randomNumber);
        return Convert.ToBase64String(randomNumber);
    }

    // Извлекает claims из протухшего access-токена
    private ClaimsPrincipal GetPrincipalFromExpiredToken(string
token)
    {
        var tokenValidationParameters = new
TokenValidationParameters
        {
            ValidateAudience = true,
            ValidAudience = AuthOptions.AUDIENCE,
            ValidateIssuer = true,
            ValidIssuer = AuthOptions.ISSUER,
            ValidateIssuerSigningKey = true,
            IssuerSigningKey =
AuthOptions.GetSymmetricSecurityKey(),
            ValidateLifetime = false
        };

        var tokenHandler = new JwtSecurityTokenHandler();
        var principal = tokenHandler.ValidateToken(token,
tokenValidationParameters, out SecurityToken securityToken);

        if (securityToken is not JwtSecurityToken jwtSecurityToken
||
!jwtSecurityToken.Header.Alg.Equals(SecurityAlgorithms.HmacSha256,
StringComparison.InvariantCultureIgnoreCase))
            throw new SecurityTokenException("Invalid token");

        return principal;
    }

    // два токена: access-токен (15 мин) + refresh-токен (7 дней,
хранится в БД).
    [HttpPost("Login")]
    public async Task<IActionResult> Login(UserDto user)
    {
        var findUser = await context.Users.FirstOrDefaultAsync(u
=> u.Login == user.Login && u.Password == user.Password);
        if (findUser is null)
            return Unauthorized("Invalid credentials");

        var roleName = context.Roles.Find(findUser.RoleId)?.Name;
        if (roleName is null)
            return Unauthorized("Role not found");
    }

```

```

        var claims = new List<Claim>
        {
            new Claim(ClaimTypes.NameIdentifier,
findUser.Id.ToString()),
            new Claim(ClaimTypes.Name, user.Login),
            new Claim(ClaimTypes.Role, roleName),
        };

        var accessToken = new JwtSecurityToken(
            issuer: AuthOptions.ISSUER,
            audience: AuthOptions.AUDIENCE,
            claims: claims,
            expires:
DateTime.UtcNow.AddMinutes(AuthOptions.ACCESS_TOKEN_LIFETIME_MINUT
ES),
            signingCredentials: new
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256));

        var encodedAccessToken = new
JwtSecurityTokenHandler().WriteToken(accessToken);

        var refreshToken = GenerateRefreshToken();
        findUser.RefreshToken = refreshToken;
        findUser.RefreshTokenExpiryTime =
DateTime.UtcNow.AddDays(AuthOptions.REFRESH_TOKEN_LIFETIME_DAYS);

        await context.SaveChangesAsync();

        var response = new TokenDto
        {
            AccessToken = encodedAccessToken,
            RefreshToken = refreshToken,
            Username = user.Login
        };

        return Ok(response);
    }

    [HttpGet("check")]
    [Authorize]
    public IActionResult Check()
    {
        return Ok();
    }

    [HttpGet("my")]
    [Authorize]
    public IActionResult Profile()
    {
        var name =User.FindFirst(ClaimTypes.Name)?.Value;
        if (name is null)

```

```

        return Unauthorized();
var role = User.FindFirst(ClaimTypes.Role)?.Value;
if (role is null)
    return Unauthorized();
return Ok(new ProfileUserDto()
{
    Login = name,
    Role = role
});
}
[HttpPost("Refresh")]
public async Task<IActionResult> Refresh(TokenDto tokenDto)
{
    if (string.IsNullOrEmpty(tokenDto.AccessToken) ||
string.IsNullOrEmpty(tokenDto.RefreshToken))
        return BadRequest("Invalid client request");

    var principal =
GetPrincipalFromExpiredToken(tokenDto.AccessToken);
    var userId =
int.Parse(principal.FindFirst(ClaimTypes.NameIdentifier)?.Value ??
"0");

    var user = await context.Users.FindAsync(userId);

    if (user == null || user.RefreshToken !=
tokenDto.RefreshToken || user.RefreshTokenExpiryTime <=
DateTime.UtcNow)
        return Unauthorized("Invalid refresh token");

    var newAccessToken = new JwtSecurityToken(
        issuer: AuthOptions.ISSUER,
        audience: AuthOptions.AUDIENCE,
        claims: principal.Claims,
        expires:
DateTime.UtcNow.AddMinutes(AuthOptions.ACCESS_TOKEN_LIFETIME_MINUT
ES),
        signingCredentials: new
SigningCredentials(AuthOptions.GetSymmetricSecurityKey(),
SecurityAlgorithms.HmacSha256));

    var encodedAccessToken = new
JwtSecurityTokenHandler().WriteToken(newAccessToken);

    var newRefreshToken = GenerateRefreshToken();

    user.RefreshToken = newRefreshToken;
    user.RefreshTokenExpiryTime =
DateTime.UtcNow.AddDays(AuthOptions.REFRESH_TOKEN_LIFETIME_DAYS);

    await context.SaveChangesAsync();

```

```

        var response = new TokenDto
        {
            AccessToken = encodedAccessToken,
            RefreshToken = newRefreshToken,
            Username = user.Login
        };

        return Ok(response);
    }

    [HttpPost("Revoke")]
    [Authorize]
    public async Task<IActionResult> Revoke()
    {
        var userId =
int.Parse(User.FindFirst(ClaimTypes.NameIdentifier)?.Value ??
"0");
        var user = await context.Users.FindAsync(userId);

        if (user == null)
            return NotFound();

        user.RefreshToken = null;
        user.RefreshTokenExpiryTime = null;

        await context.SaveChangesAsync();

        return NoContent();
    }

    [HttpPost("Register")]
    public async Task<IActionResult> Register(UserDto user)
    {
        if ((await context.Users.AnyAsync(u => u.Login ==
user.Login)))
            return BadRequest("Login is already occupied");

        User newUser = new User
        {
            Login = user.Login,
            Password = user.Password,
            RoleId = 1
        };

        context.Users.Add(newUser);
        await context.SaveChangesAsync();

        return Ok(new { Message = "Registration successful",
UserId = newUser.Id });
    }
}

```

Листинг Б.25 – Файл Controllers/AnalyzedSiteController.cs

```

using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using webParser.Data;
using webParser.Models.Database;
using webParser.Models.DTO.AnalyzedSite;

namespace webParser.Controllers;
[Route("api/[controller]")]
[ApiController]
[Authorize(Roles = "Редактор,Администратор")]
public class AnalyzedSiteController(ILogger<HomeController>
logger, AppDbContext context) : Controller
{
    [HttpGet("all")]
    [AllowAnonymous]

    public IActionResult Get()
    {
        return Ok(context.AnalyzedSites.ToList());
    }

    [HttpGet("{siteId}")]
    [AllowAnonymous]
    public IActionResult Get([FromRoute] int siteId)
    {
        var site = context.AnalyzedSites.FirstOrDefault(x => x.Id
== siteId);
        if (site == null)
            return NotFound();
        return Ok(site);
    }

    [HttpPost]
    public IActionResult Post([FromBody] CreateAnalyzedSiteDto
site)
    {
        if(site.Url=="")
            return BadRequest("Url is required");
        if(site.Name=="")
            return BadRequest("Name is required");
        var id =
Convert.ToInt32(User.FindFirst(ClaimTypes.NameIdentifier)?.Value);
        if (id == 0)
            return BadRequest("no user id");
        if(context.AnalyzedSites.Any(s=>s.Url==site.Url))
            return BadRequest("Site already exists");
        var entity = new AnalyzedSite()
        {
            Name = site.Name,

```

```

        Url = site.Url,
        UserId = Convert.ToInt32(id)
    };
    context.AnalyzedSites.Add(entity);
    context.SaveChanges();
    return Ok(entity);
}

[HttpPut("{siteId}")]
public IActionResult Put([FromRoute] int siteId, [FromBody]
CreateAnalyzedSiteDto site)
{
    if(site.Name=="")
        return BadRequest("Name is required");
    if(site.Url=="")
        return BadRequest("Url is required");
    if (siteId == 0)
        return BadRequest("no site id");
    var analyzedSite = context.AnalyzedSites.Find(siteId);
    if (analyzedSite == null)
        return NotFound("site not found");
    analyzedSite.Url=site.Url;
    analyzedSite.Name = site.Name;
    context.SaveChanges();
    return Ok(site);
}

[HttpDelete("{siteId}")]
public IActionResult Delete([FromRoute] int siteId)
{
    if (siteId == 0)
        return BadRequest("no site id");
    var analyzedSite = context.AnalyzedSites.Find(siteId);
    if (analyzedSite == null)
        return NotFound("site not found");
    context.AnalyzedSites.Remove(analyzedSite);
    context.SaveChanges();
    return Ok(siteId);
}
}

```

Листинг Б.26 – Файл Controllers/AnalyzedFieldController.cs

```

using System.Security.Claims;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using webParser.Data;
using webParser.Models.Database;
using webParser.Models.DTO.AnalyzedField;

namespace webParser.Controllers;

```

```

[Route("api/[controller]")]
[ApiController]
[Authorize(Roles = "Редактор,Администратор")]
public class AnalyzedFieldController(ILogger<HomeController>
logger, AppDbContext context) : Controller
{
    [HttpGet("all")]
    public IActionResult Get()
    {
        return Ok(context.AnalyzedFields.ToList());
    }

    [HttpGet("{id}")]
    public IActionResult Get([FromRoute] int id)
    {
        var site = context.AnalyzedFields.FirstOrDefault(x => x.Id
== id);
        if (site == null)
            return NotFound();
        return Ok(site);
    }

    [HttpGet("site/{siteId}")]
    public IActionResult GetBySiteId([FromRoute] int siteId)
    {
        var site = context.AnalyzedFields.Where(x =>
x.AnalyzedSiteId == siteId).ToList();
        if (site == null)
            return NotFound();
        return Ok(site);
    }

    [HttpPost]
    public IActionResult Post([FromBody] CreateAnalyzedFieldDto
dto)
    {
        if (dto == null)
            return BadRequest("no body");
        if (dto.Name == "")
            return BadRequest("name is required");
        if (dto.AnalyzedSiteId == 0)
            return BadRequest("site id is required");
        if (dto.FieldToGet == "")
            return BadRequest("field id is required");
        if (context.AnalyzedSites.Find(dto.AnalyzedSiteId) ==
null)
            return NotFound("site not found");
        var fieldName = context.FieldNames.Find(dto.FieldNameId);
        if (fieldName == null)
            return BadRequest($"field name with
id={dto.FieldNameId} not found");
        context.AnalyzedFields.Add(new AnalyzedField()

```

```

        {
            Name = dto.Name,
            FieldToGet = dto.FieldToGet,
            AnalyzedSiteId = dto.AnalyzedSiteId,
            FieldNameId = dto.FieldNameId
        });
        context.SaveChanges();
        return Ok(dto);
    }

    [HttpPatch("{id}")]
    public IActionResult Put([FromRoute] int id, [FromBody]
UpdateAnalyzedFieldDto dto)
    {
        if(id == 0)
            return BadRequest("id is required");
        if (dto is { Name: "", FieldToGet: "", AnalyzedSiteId:
null })
            return BadRequest("changed fields is required");
        var site = context.AnalyzedFields.Find(id);
        if (site == null)
            return NotFound("site not found");
        if(context.FieldNames.Find(dto.FieldNameId)==null)
            return NotFound("field not found");
        if (dto.AnalyzedSiteId != null)
            site.AnalyzedSiteId =(int)dto.AnalyzedSiteId;
        site.Name = dto.Name??site.Name;
        site.FieldToGet = dto.FieldToGet??site.FieldToGet;
        site.FieldNameId = dto.FieldNameId ?? site.FieldNameId;
        context.SaveChanges();
        return Ok(dto);
    }

    [HttpDelete("{id}")]
    public IActionResult Delete([FromRoute] int id)
    {
        if (id == 0)
            return BadRequest("no correct id");
        var analyzedSite = context.AnalyzedFields.Find(id);
        if (analyzedSite == null)
            return NotFound("field not found");
        context.AnalyzedFields.Remove(analyzedSite);
        context.SaveChanges();
        return Ok(id);
    }
}

```

Листинг Б.27 – Файл Controllers/ParsedDataController.cs

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;

```

```

using webParser.Data;

namespace webParser.Controllers;

[Route("api/[controller]")]
[ApiController]
public class ParsedDataController(AppDbContext context) :
    Controller
    {
        [HttpGet("all")]
        public async Task<IActionResult> GetAll()
        {
            var data = await context.ParsedData
                .Select(p => new { p.SiteId, p.Field, p.Data,
p.UpdatedAt })
                .ToListAsync();
            return Ok(data);
        }

        [HttpGet("site/{siteId}")]
        public async Task<IActionResult> GetBySite([FromRoute] int
siteId)
        {
            var data = await context.ParsedData
                .Where(p => p.SiteId == siteId)
                .Select(p => new { p.Field, p.Data, p.UpdatedAt })
                .ToListAsync();
            return Ok(data);
        }

        [HttpDelete("site/{siteId}")]
        [Authorize]
        public async Task<IActionResult> ClearCache([FromRoute] int
siteId)
        {
            var rows = context.ParsedData.Where(p => p.SiteId ==
siteId);
            context.ParsedData.RemoveRange(rows);
            await context.SaveChangesAsync();
            return Ok(new { deleted = true, siteId });
        }
    }

```

Приложение Г
Результаты тестирования серверной части
(рекомендуемое)

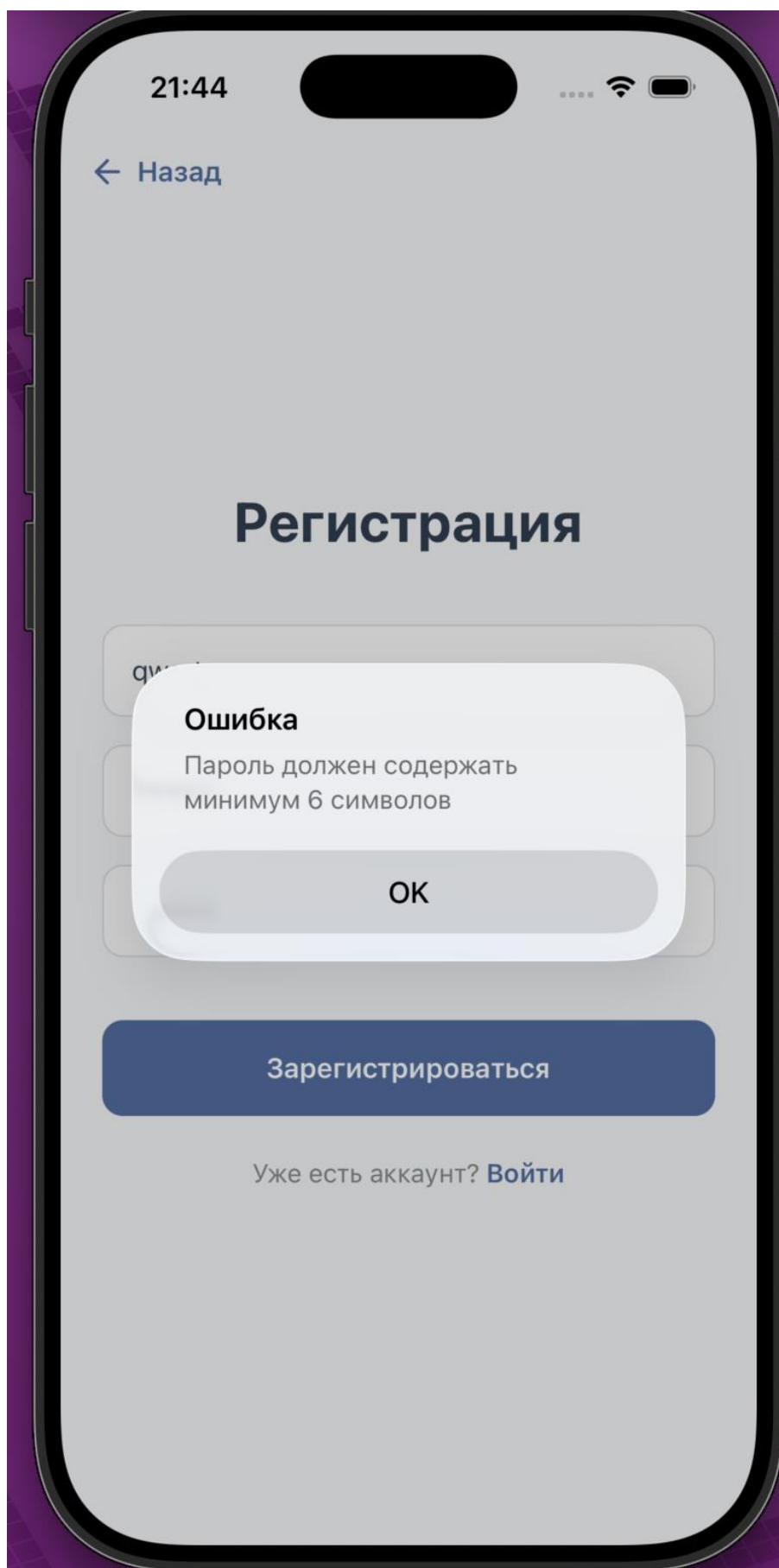


Рисунок Г.1 – Ввод пароля короче 6 символов

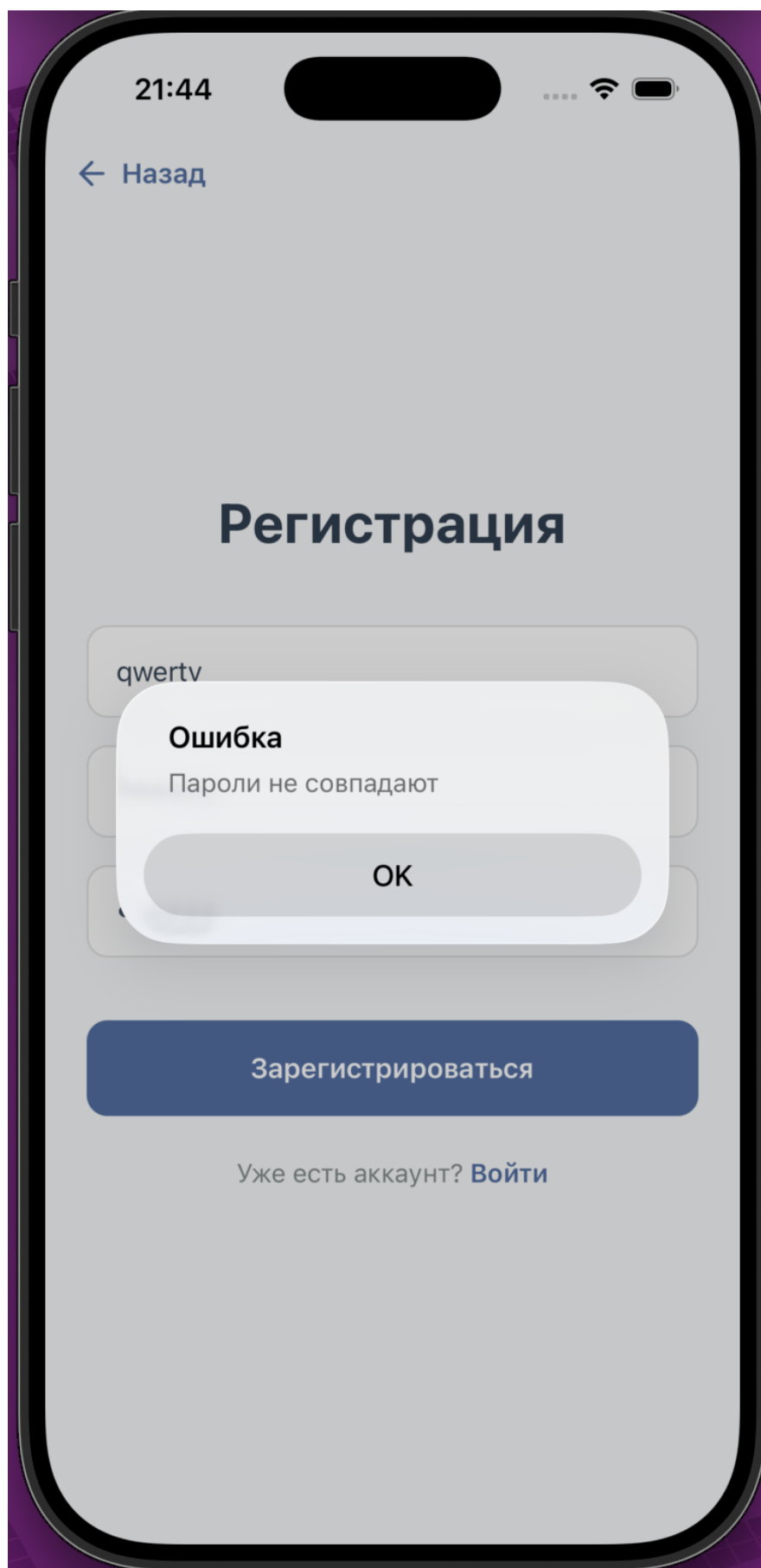


Рисунок Г.2 – Ввод несовпадающих паролей

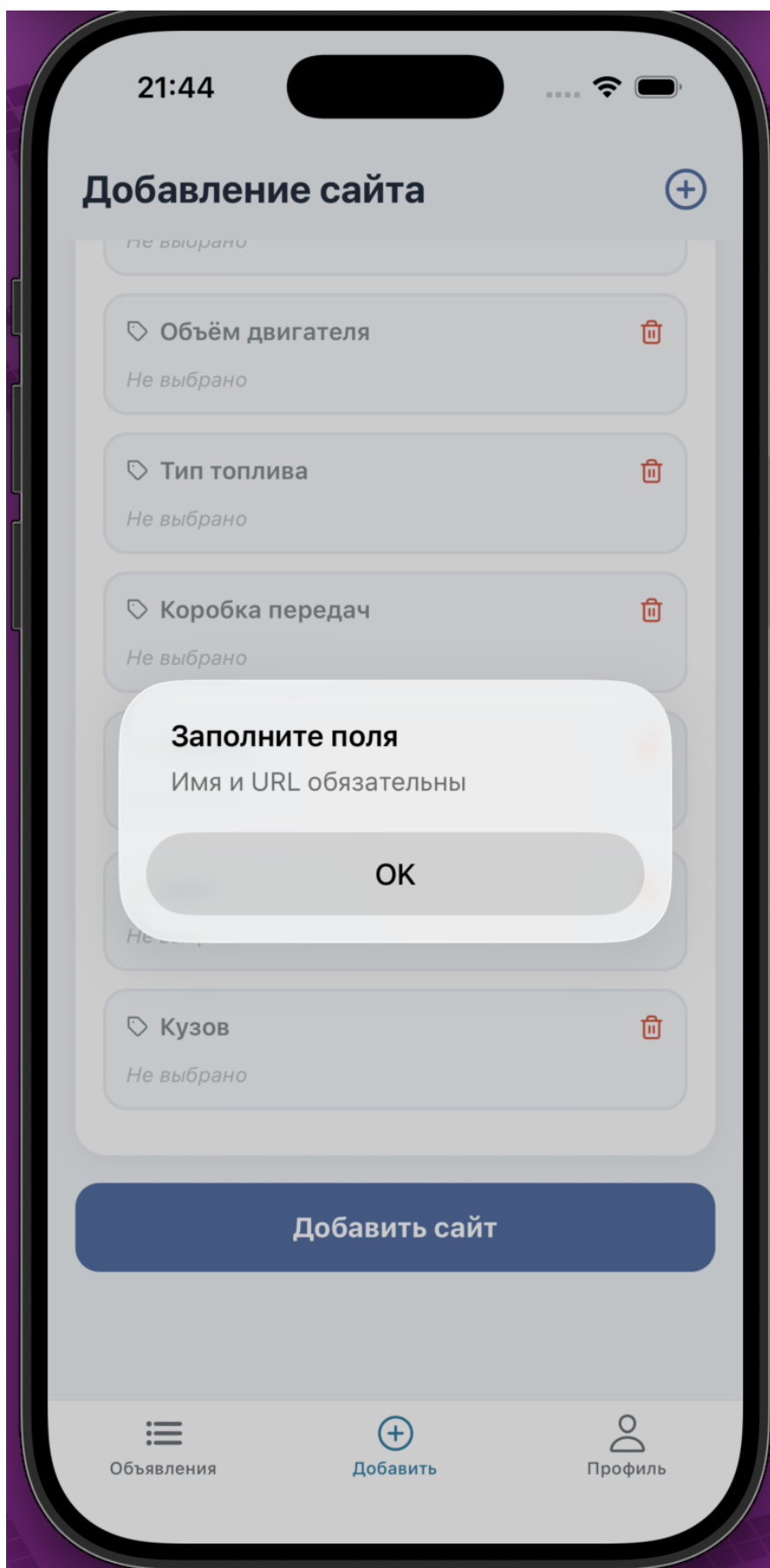


Рисунок Г.3 – Попытка сохранить сайт с пустым названием

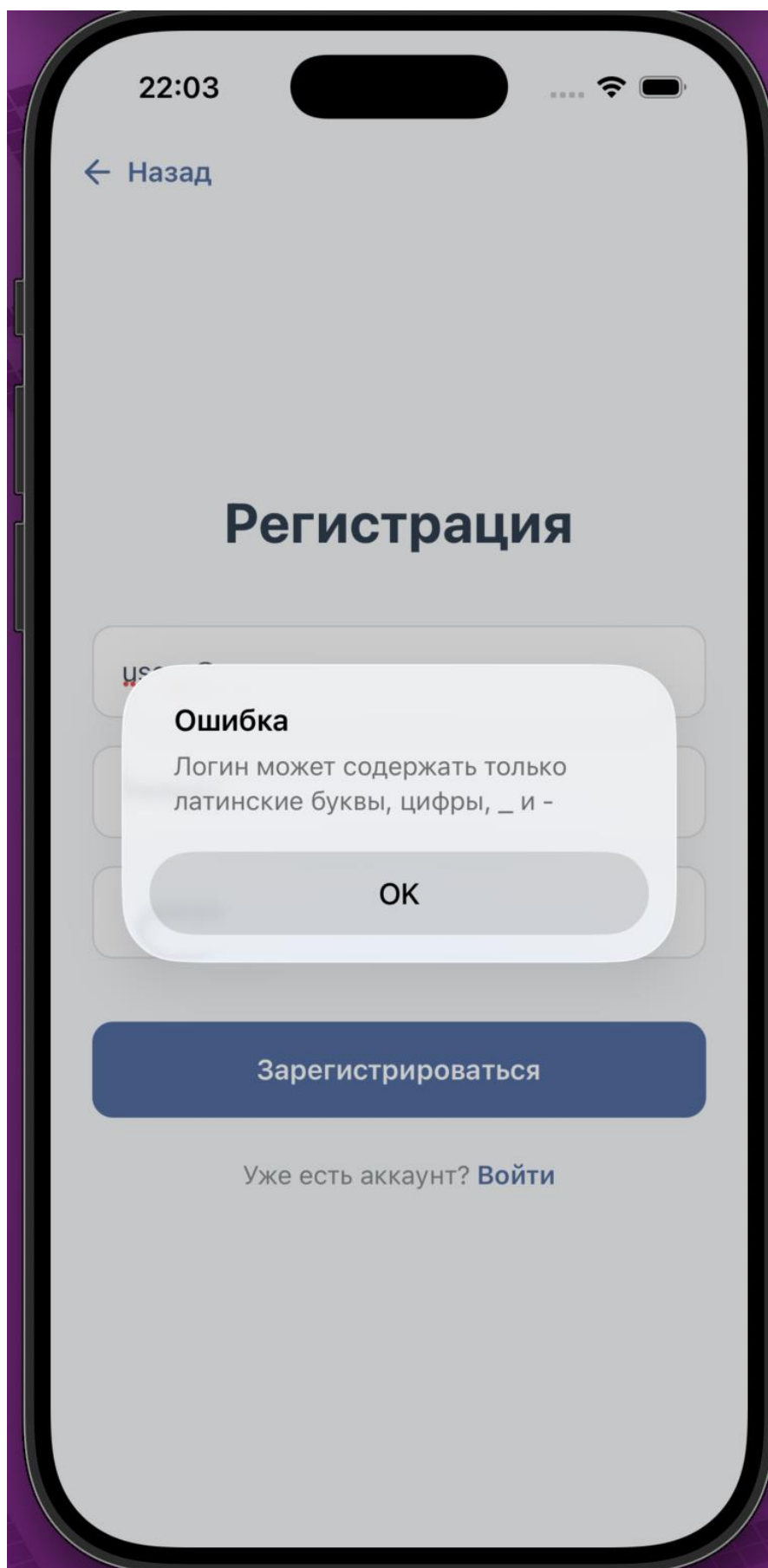


Рисунок Г.4 – Ввод логина с недопустимыми символами

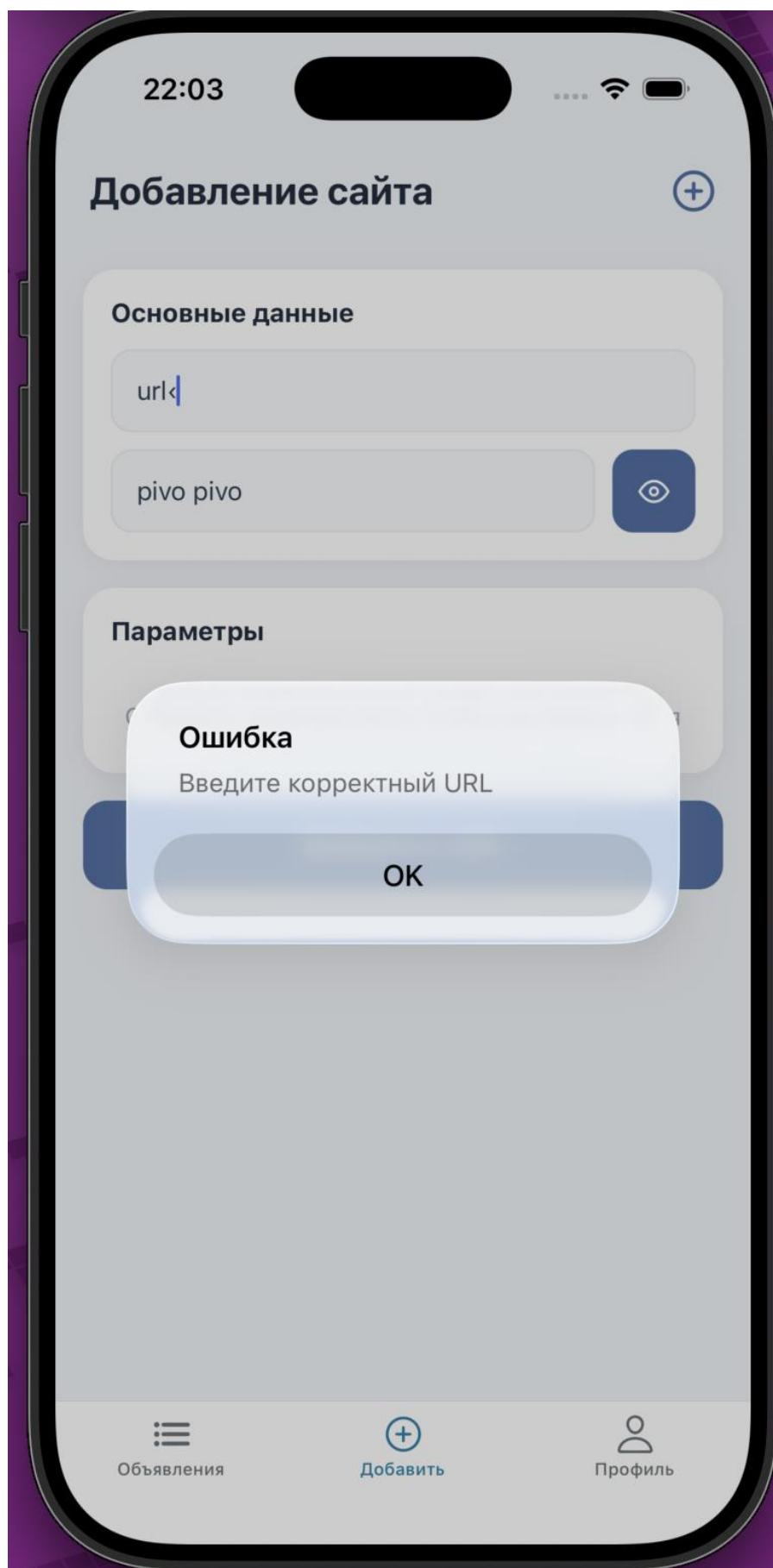


Рисунок Г.5 – Ввод некорректного URL

Приложение Д
Исходный код функций тестирования
(рекомендуемое)

Листинг Д.1 – Файл StringParserTests.cs

```

using FluentAssertions;
using Microsoft.Extensions.Logging;
using Moq;
using webParser.Models.DTO;
using webParser.Service;
using Xunit;

namespace webParser.Test.Service;

public class StringParserTests
{
    private readonly Mock<ILogger<StringParser>> _loggerMock;
    private readonly StringParser _parser;

    public StringParserTests()
    {
        _loggerMock = new Mock<ILogger<StringParser>>();
        _parser = new StringParser(_loggerMock.Object);
    }

    [Fact]
    public void
ParseString_ПриNullHtml_ВозвращаетПоляCNoContent()
    {
        var fields = new List<DataField>
        {
            new DataField { Field = "Title", Data = "<h1>" },
            new DataField { Field = "Price", Data = "<span
class='price'>" }
        };

        var result = _parser.ParseString(null, fields);

        result.Should().HaveCount(2);
        result.All(r => r.Data == "No
content").Should().BeTrue();
        result.Select(r =>
r.Field).Should().BeEquivalentTo(fields.Select(f => f.Field));
    }

    [Fact]
    public void
ParseString_ПриПустомHtml_ВозвращаетПоляCNoContent()
    {
        var fields = new List<DataField>
        {
            new DataField { Field = "Title", Data = "<h1>" }
        };

        var result = _parser.ParseString("", fields);
    }
}

```

```

        result.Should().ContainSingle()
            .Which.Data.Should().Be("No content");
    }

    [Fact]
    public void ParseString_СКорректнымHtml_ИзвлекаетПоля()
    {
        var html = @"
            <html>
                <body>
                    <h1>Product Title</h1>
                    <span class='price'>$19.99</span>
                    <div id='description'>A great
product</div>
                </body>
            </html>";

        var fields = new List<DataField>
        {
            new DataField { Field = "Title", Data = "<h1>" },
            new DataField { Field = "Price", Data = "<span
class='price'>" },
            new DataField { Field = "Description", Data =
"<div id='description'>" }
        };

        var result = _parser.ParseString(html, fields);

        result.Should().HaveCount(3);
        result.Single(r => r.Field ==
"Title").Data.Should().Be("Product Title");
        result.Single(r => r.Field ==
"Price").Data.Should().Be("$19.99");
        result.Single(r => r.Field ==
"Description").Data.Should().Be("A great product");
    }

    [Fact]
    public void
ParseString_КогдаПолеНеНайдено_ИспользуетАльтернативныйПоискИВозвр
ащаетNotFound()
    {
        var html = @"
            <html>
                <body>
                    <div class='old-class'>Some content</div>
                </body>
            </html>";

        var fields = new List<DataField>
        {

```

```

        new DataField { Field = "NotFound", Data = "<span
class='nonexistent'>" }
        };

        var result = _parser.ParseString(html, fields);

        result.Should().ContainSingle()
            .Which.Data.Should().Be("Not Found");
    }

    [Fact]
    public void
ParseString_АльтернативныйПоискПоКлассу_Успешен()
    {
        var html = @"
        <html>
            <body>
                <div class='price-tag'>$29.99</div>
            </body>
        </html>";

        var fields = new List<DataField>
        {
            new DataField { Field = "Price", Data = "<span
class='price-tag'>" }
        };

        var result = _parser.ParseString(html, fields);

        result.Should().ContainSingle()
            .Which.Data.Should().Be("$29.99");
    }

    [Fact]
    public void
ParseString_АльтернативныйПоискПоИдентификатору_Успешен()
    {
        var html = @"
        <html>
            <body>
                <div id='product-title'>Awesome
Product</div>
            </body>
        </html>";

        var fields = new List<DataField>
        {
            new DataField { Field = "Title", Data = "<h1
id='product-title'>" }
        };

```

```

        var result = _parser.ParseString(html, fields);

        result.Should().ContainSingle()
            .Which.Data.Should().Be("Awesome Product");
    }

    [Fact]
    public void
ParseString_АльтернативныйПоискПоТегу_Успешен()
    {
        var html = @"
            <html>
                <body>
                    <p>First paragraph</p>
                    <p>Second paragraph</p>
                </body>
            </html>";

        var fields = new List<DataField>
        {
            new DataField { Field = "FirstP", Data = "<p>" }
        };

        var result = _parser.ParseString(html, fields);

        result.Should().ContainSingle()
            .Which.Data.Should().Be("First paragraph");
    }

    [Fact]
    public void
ParseString_СНекорректнымФорматомТега_ВозвращаетПолеСОшибкой()
    {
        var html = "<html><body>Some content</body></html>";
        var fields = new List<DataField>
        {
            new DataField { Field = "Invalid", Data = "not a
tag" }
        };

        var result = _parser.ParseString(html, fields);

        result.Should().ContainSingle();
        result[0].Data.Should().StartWith("Error:");
    }

    [Fact]
    public void
ParseString_ПриИсключенииВоВремяПарсингаПоля_ЛогируетОшибкуИВозвра
щаетПолеСОшибкой()
    {
        var html = "<html><body></body></html>";

```

```

var fields = new List<DataField>
{
    new DataField { Field = "NullSelector", Data =
null }
};

var result = _parser.ParseString(html, fields);

result.Should().ContainSingle();
result[0].Data.Should().StartWith("Error:");
_loggerMock.Verify(
    x => x.Log(
        LogLevel.Error,
        It.IsAny<EventId>(),
        It.Is<It.IsAnyType>((v, t) => true),
        It.IsAny<Exception>(),
        It.Is<Func<It.IsAnyType, Exception,
string>>((v, t) => true)),
    Times.AtLeastOnce);
}

[Fact]
public void ParseString_CHtmlEntities_ОчищаетТекст()
{
    var html = @"
<html>
    <body>
        <p>Price & tax included</p>
    </body>
</html>";

    var fields = new List<DataField>
{
    new DataField { Field = "Description", Data =
"<p>" }
};

var result = _parser.ParseString(html, fields);

result.Should().ContainSingle()
    .Which.Data.Should().Be("Price & tax included");
}

[Fact]
public void
ParseString_СМножественнымиПробелами_ОчищаетТекст()
{
    var html = @"
<html>
    <body>
        <div>    Lots    of
                spaces
    </div>
    </body>
</html>";

```

```

        </div>
    </body>
</html>";

var fields = new List<DataField>
{
    new DataField { Field = "Text", Data = "<div>" }
};

var result = _parser.ParseString(html, fields);

result.Should().ContainSingle()
    .Which.Data.Should().Be("Lots of spaces");
}

[Fact]
public void
ConvertToXPath_СПростымиТегами_ВозвращаетПравильныйXPath()
{
    var html = "<html><body><h1>Title</h1></body></html>";
    var fields = new List<DataField> { new DataField {
Field = "Title", Data = "<h1>" } };
    var result = _parser.ParseString(html, fields);
    result.Single().Data.Should().Be("Title");
}

[Fact]
public void
ConvertToXPath_САтрибутами_ВозвращаетПравильныйXPath()
{
    var html = "<html><body><span class='price' data-
type='final'>$10</span></body></html>";
    var fields = new List<DataField> { new DataField {
Field = "Price", Data = "<span class='price' data-type='final'>" }
};
    var result = _parser.ParseString(html, fields);
    result.Single().Data.Should().Be("$10");
}
}

```