МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

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

Факультет	*Кафедра*
вычислительной техники	МОиПЭВМ

Направление подготовки 09.03.02 «Информационные системы и технологии»

БАКАЛАВРСКАЯ РАБОТА

на тему

Клиент-серверное приложение для подбора поддержанных автомобилей с учетом предпочтений пользователей

Студент _____ Молотков Вячеслав Николаевич
(подпись, дата) (ФИО полностью)

Руководитель _____ Такташкин Д.В.
(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, 2023

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

Факультет	*Кафедра*
Вычислительной техники	МО и ПЭВМ

«Утверждаю»
Заведующий кафедрой
_____А.Ю. Козлов
«__»_____2023 г

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

БАКАЛАВРА

1. Студент Молотков Вячеслав Николаевич гр. 19ВИ1 факультета ВТ
<Фамилия, имя, отчество полностью>

2. Тема работы Клиент-серверное приложение для подбора поддержанных автомобилей с учетом предпочтений пользователей

Тема утверждена приказом ПГУ №_____от "_____"_____20_____

3. Руководитель работы доцент «МОиПЭВМ», к.т.н. Такташкин Д.В

4. Задание на работу (назначение разработки, исходные данные и т.п.)
1) Разработать клиент-серверное приложение для подбора поддержанных автомобилей с учетом предпочтений пользователей
2) Основные функции системы:
а) вывод объявлений о продаже автомобилей;
б) вывод персональной подборки объявлений;
в) добавление новых объявлений
г) удаление объявлений

5. Перечень подлежащих разработке вопросов
1) Анализ предметной области и требований к системе
2) Проектирование программных средств
3) Реализация программных средств
4) Тестирование программных средств
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%	12.04.2023 – 16.04.2023	
2	Проектирование программных средств	20%	17.04.2023 – 26.04.2023	
3	Реализация программных средств	30%	27.04.2023 – 11.05.2023	
4	Тестирование программных средств	20%	12.05.2023 – 21.05.2023	
5	Оформление пояснительной записки	20%	22.05.2023 – 01.06.2023	

Дата выдачи задания «22» декабря 2022 г.

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 77 листов, 13 рисунка, 11 таблиц, 18 использованных источников, 3 приложения.

WEB-ПРИЛОЖЕНИЕ, ПОДБОР ПОДДЕРЖАННЫХ АВТОМОБИЛЕЙ, MONGODB, JAVA, JAVASCRIPT, HTML, CSS, REACTJS, UML

Объектом разработки является клиент-серверное приложение для подбора поддержанных автомобилей с учетом предпочтений пользователей.

Цель работы – разработка клиент-серверного приложения для подбора поддержанных автомобилей, позволяющее добавлять, удалять и редактировать объявления.

Разработка серверной части приложения осуществлялась на языке программирования Java. Для реализации серверной части использовалась среда разработки Intellij IDEA. В качестве СУБД использовалась MongoDB.

Разработка клиентской части приложения осуществлялась на языке программирования JavaScript. Для реализации клиентской части приложения использовалась среда разработки JetBrains Webstorm. В качестве вспомогательных языковых средств разработки использовались языки HTML и CSS. Приложение предназначено для работы на устройствах под управлением ОС семейства Windows NT/UNIX.

Оглавление

Введение 7
1 Анализ предметной области и требований 8
1.1 Анализ предметной области и постановка задачи 8
1.2 Анализ доступных сервисов для продажи автомобилей 10
1.2.1 Интернет-портал «Дром» 10
1.2.2 Автомобильный сайт «Авто.ру» 10
1.2.3 Интернет-сервис «Авито» 11
1.3 Сравнительный анализ сервис для продажи автомобилей 11
1.4 Функционально-стоимостный анализ 14
2 Анализ информационных технологий разработки 18
2.1 Инструментальные средства проектирования 18
2.2 Инструментальные средства разработки 19
2.2.1 Язык программирования Java 19
2.2.2 Фреймворк Spring 20
2.2.3 Язык программирования JavaScript 21
2.2.4 Фреймворк React 22
2.2.5 Язык гипертекстовой разметки HTML 23
2.2.6 Каскадные таблицы стилей CSS 24
2.2.7 Выбор СУБД 25
3 Анализ функциональных требований к разрабатываемому приложению 27
4 Проектирование клиент-серверного приложения 29
4.1 Проектирование структуры клиент-серверного приложения 29
4.2 Проектирование пользовательского интерфейса 35
5 Реализация клиент-серверного приложения 40
5.1 Реализация серверной части приложения 40
5.2 Реализация клиентской части приложения 41
6 Тестирование «Клиент-серверного приложения для подбора поддержанных автомобилей с учетом предпочтений пользователей» 43
6.1 Тестирование формы для ввода марки автомобиля 43
6.2 Тестирование метрики кода 48

6.3 Автоматизирование тестирование репозитория для хранения пользователей 51
Заключение 54
Список используемых источников 55
Приложение А. Исходный код серверной части 57
Приложение Б. Исходный код клиентской части 66
Приложение В. Результаты тестирования 72

Введение

Сервисы для продажи подержанных автомобилей остаются актуальными и востребованными. Они предоставляют большой выбор автомобилей различных марок, моделей, годов выпуска и ценовых категорий. Покупателям это дает возможность найти автомобиль, который соответствует их требованиям и бюджету. Эти сервисы также предоставляют удобство и доступность, позволяя покупателям просматривать и сравнивать автомобили онлайн из своего дома или офиса. Это экономит время и усилия, которые тратятся на посещение автосалонов или частных продавцов.

Одна из важных преимуществ сервисов для продажи подержанных автомобилей – информация о состоянии автомобиля. Многие сервисы предоставляют подробные данные о техническом состоянии, пробеге, истории обслуживания и аварий автомобилей. Это помогает покупателям принимать осознанные решения и избегать потенциальных проблем.

Безопасность и доверие также являются важными аспектами сервисов продажи подержанных автомобилей. Надежные сервисы обеспечивают проверку и проверку автомобилей перед их размещением на платформе. Это создает доверие у покупателей и позволяет им чувствовать себя уверенно при совершении покупки.

Дополнительные услуги, предоставляемые некоторыми сервисами, такие как финансирование, страхование и оформление документов, делают процесс покупки более удобным для клиентов. Продавцам также предоставляются возможности размещать объявления и эффективно привлекать потенциальных покупателей.

Целью выполнения выпускной квалификационной работы является создание сервиса, который позволяет размещать объявления о продаже автомобилей, а также осуществляет подбор автомобилей в зависимости от предпочтений пользователя.

1 Анализ предметной области и требований

1.1 Анализ предметной области и постановка задачи

Сервис продажи автомобилей – это виртуальная платформа или веб-сайт, где покупатели и продавцы могут взаимодействовать и совершать сделки по продаже или покупке автомобилей через интернет. Данные сервис предоставляет удобную и эффективную альтернативу традиционным автосалонам и частным продавцам.

Анализ источников информации позволяет утверждать, что любой сервис для продажи автомобилей содержит следующие элементы:

- список объявлений о продаже автомобилей;
- инструмент для отбора определенных объявлений;
- форма авторизации;
- форма регистрации;
- личный кабинет пользователя;
- личный кабинет модератора.

Кратко рассмотрим каждый из представленных выше элементов.

Список объявлений состоит из карточек, которые содержат краткое описание автомобиля, информацию о продавце, цену, год выпуска, а также одно или несколько изображений автомобиля.

Инструмент для отбора определенных объявлений представляет из себя набор параметров, позволяющие отфильтровать выдаваемый список автомобилей.

Форма авторизации содержит поля для ввода имени пользователя и пароля, которые требуются для входа в личный кабинет.

Форма регистрации содержит те же поля, что и форма авторизации, но в нее также входят поля для ввода ФИО, возраста пользователя и контактных данных. Форма регистрации требуется для создания новых учетных записей пользователей.

Личный кабинет пользователя позволяет менять данные учетной записи, а также предоставляет доступ для создания объявления о продаже автомобиля.

Личный кабинет модератора имеет ту же структуру, что и личный кабинет пользователя, но также содержит в себе инструмент, позволяющий одобрять и отклонять объявления, которые размещают обычные пользователи.

Кроме того, что сервисы для продажи автомобилей имеют общие элементы, многие из них могут содержать дополнительные возможности, отличающие их от других сервисов.

Определим основные требования к разрабатываемому приложению.

Клиентская часть сервиса распознавания должна содержать:

– список объявлений, которые размещают пользователи;
– инструмент для настройки отбора объявлений по определенным параметрам;
– форма авторизации;
– форма регистрации;
– раздел с персональной подборкой объявлений для пользователя;
– личный кабинет пользователя;
– личный кабинет модератора;
– анкета пользователя, на основе которой составляется персональная подборка;
– панель для добавления объявления.

Серверная часть состоит из следующих элементов:

– сервис для обработки запросов клиента;
– сервис авторизации пользователя;
– сервис для получения и редактирования объявлений;
– сервис для получения персональной подборки объявлений.

Данные требования позволят рассмотреть аналоги сервисов для продажи автомобилей и провести их сравнительный анализ.

1.2 Анализ доступных сервисов для продажи автомобилей

В ходе изучения предметной области был проведен анализ популярных сервисов, специализирующихся на продаже автомобилей. Далее представлен обзор нескольких из них.

1.2.1 Интернет-портал «Дром»

Дром – это автомобильный Интернет-портал, сочетающий многогранную и обширную базу информации по автомобилям с современными технологиями общения. Изначально проект охватывал преимущественно японские автомобили, но со временем, по мере роста и переплетения аудитории, акцент начал смещаться в сторону всех авто [1].

Здесь можно найти как новые, так и подержанные автомобили различных марок и моделей. Основными преимуществами «Дром.ру» являются широкий выбор объявлений, подробная информация о каждом автомобиле, а также удобный интерфейс и простота использования. Однако, стоит отметить, что некоторые пользователи могут столкнуться с повторяющимися объявлениями и ограниченными функциональными возможностями, такими как отсутствие чата с продавцом или возможности сравнения нескольких автомобилей.

Особенностью данного сервиса является то, что при переходе на страницу определенного автомобиля, в нижней части присутствует раздел, в котором представлены объявления о продаже запчастей, которые подходят этому автомобилю. Данный интернет-портал представляет из себя не только сервис для продажи автомобилей, но и активный форум автолюбителей, а также портал, на котором публикуются статьи об автомобилях и путешествиях.

1.2.2 Автомобильный сайт «Авто.ру»

Авто.ру – Автомобильный сайт в Рунете, через который ежегодно продаётся более полутора миллионов подержанных машин. Создан в 1996 году Михаилом Рогальским. В 2014 году был выкуплен компанией «Яндекс». Весной 2016 года в состав Auto.ru вошёл агрегатор «Яндекс.Авто», затем под этим

именем была запущена информационно-развлекательная платформа для автомобилистов [2].

Среди преимуществ «Авто.ру» следует отметить обширную базу объявлений, возможность ознакомиться с новостями и экспертными обзорами, удобный поиск и фильтрация, а также подробную информацию о каждом автомобиле.

К особенностям данного сайта также можно отнести наличие чата с продавцом, возможность сравнить несколько автомобилей между собой и историю поиска объявлений.

1.2.3 Интернет-сервис «Авито»

Авито – российский интернет-сервис для размещения объявлений о товарах, недвижимости, вакансиях и резюме на рынке труда, а также услугах от частных лиц и компаний, занимающий первое место в мире среди сайтов объявлений. Товары, предлагаемые к продаже на «Авито», могут быть новыми и бывшими в использовании. По результатам апреля 2023 года количество активных объявлений на платформе превысило 150 миллионов, увеличившись за год почти в два раза [3].

К особенностям «Авито» можно отнести возможность посмотреть состояние кузова автомобиля на специальной схеме, представленной на странице с автомобилем, а также наличие раздела, в котором представлена информации о приблизительном расходе денежных средств на обслуживание данного автомобиля.

1.3 Сравнительный анализ сервис для продажи автомобилей

В ходе анализа существующих аналогов был составлен перечень основных функций, которые должны присутствовать в сервисе для продажи автомобилей, а также определены критерии, которым сервис должен соответствовать.

Каждый сервис для продажи автомобилей должен содержать в себе следующие элементы:

– список объявлений о продаже автомобилей;

– панель для настройки отбора объявлений по определенным параметрам;

– форма создания нового объявления;

– личный кабинет пользователя.

Для сравнения вышеперечисленных сервисов выделен ряд критериев, которые позволят определить наиболее удобный сервис для пользователя:

– возможность сравнить объявления;

– возможность добавить объявление в избранное;

– возможность предложить свою цену;

– наличие чата с продавцом;

– персональная подборка объявлений;

– наличие виртуального помощника для расчета кредита.

Сравнение объявлений является важной функцией, так как позволяет пользователю совершить наиболее выгодную покупку, не сопоставляя вручную каждый параметр объявления.

Возможность добавления объявления в избранное позволяет сохранять понравившиеся объявления.

Возможность предложить свою цену зачастую помогает покупателю более выгодно приобрести автомобиль.

Наличие чата значительно упрощает коммуникацию между покупателем и продавцом, так как каждый из участников диалога может написать сообщение в любое время, а собеседник ответит тогда, когда ему удобно.

Персональная подборка представляет из себя список объявлений, который были отобраны в соответствии с предпочтениями пользователя.

Виртуальный помощник для расчета кредита подбирает различные предложения о получении денежных средств взаем для покупки автомобиля. Это полезная и актуальная функция, так как финансовое положение пользователей не всегда позволяет купить автомобиль за полную стоимость.

Для удобного представления результатов сравнения была составлена таблица, в которой каждому сервису сопоставлен каждый критерий.

Результат сравнительного анализа представлен в таблице 1.

Таблица 1 – Результат сравнительного анализа представленных сервисов

Критерии	Сервисы для продажи автомобилей		
	«Дром»	«Авто.ру»	«Авито»
Возможность сравнения объявлений	–	+	+
Избранные объявления	+	+	+
Возможность предложить свою цену	–	–	+
Чат с продавцом	–	+	+
Персональная подборка объявлений	–	–	–
Виртуальный помощник для расчета кредита	+	+	+

Исходя из результата сравнения сервисов для продажи автомобилей, можно сделать вывод, что сервис «Авито» является наиболее удобным для пользователя, но ни один из них не имеет функции подбора объявлений на основе предпочтений пользователя. Именно это и послужило причиной разработки клиент-серверного приложения для подбора поддержанных автомобилей с учетом предпочтений пользователя.

1.4 Функционально-стоимостный анализ

Под функционально-стоимостным анализом понимается метод системного исследования функции отдельного изделия или определенного производственно-хозяйственного процесса, или же управленческой структуры, направленный на минимизацию затрат в сферах проектирования, освоения производства, сбыта, промышленного и бытового потребления при высоком качестве, предельной полезности и долговечности [4].

Эта технология позволяет определить стоимость товара или услуги, независимо от того, кем они создаются – организацией или предприятием. В рамках функционально-стоимостного анализа, деятельность, выполняемая на разных этапах производства продукта, называется функциями.

Преимущества функционально-стоимостного анализа заключаются в следующем:

– обеспечение доступности результатов для любого персонала в предприятии или организации;

– распределение расходов в соответствии с использованными ресурсами и представлением о процессах на производстве.

Для определения стоимости и целесообразности разработки данного проекта, проведем функционально-стоимостной анализ. Этот анализ позволит определить время, затрачиваемое на реализацию проекта, а также стоимость каждого из этапов разработки проекта.

Для планирования процесса разработки и расчета бюджета была использована диаграмма Ганта.

Диаграмма Ганта – это популярный тип столбчатых диаграмм (гистограмм), который используется для иллюстрации плана, графика работ по какому-либо проекту. Является одним из методов планирования проектов [5].

Для построения диаграммы были выделены следующие задачи: «Анализ требований», «Проектирование», «Реализация», «Тестирование», и «Оформление ПЗ».

План разработки клиент-серверного приложения для подбора поддержанных автомобилей с учетом предпочтений пользователей представлен в таблице 2.

Таблица 2 – План разработки

Наименование задачи	Длительность задачи в днях	Дата начала	Дата окончания
Анализ требований	5	12.04.2023	16.04.2023
Проектирование	10	17.04.2023	26.04.2023
Реализация	15	27.04.2023	11.05.2023
Тестирование	10	12.05.2023	21.05.2023
Оформление ПЗ	11	22.05.2023	01.06.2023

На основе плана разработки была составлена диаграмма Ганта, представленная на рисунке 1.

Рисунок 1 – Диаграмма Ганта

Для определения стоимости разработки проекта, требуется определить среднюю заработную плату таких специалистов как системный аналитик, Web-дизайнер, программист и тестировщик.

Сервис «ГородРабот.ру» предоставляет статистику заработной платы разных профессий в разные года. Данный сервис содержит информацию о средней, медианной и модальной зарплатах. «ГородРабот.ру» считает зарплату

на основе российских вакансий. Статистика показывает предложения работодателей, а не реальные зарплаты, которые получают работники с учетом премий и надбавок.

Данные о средних зарплатах специалистов представлены в таблице 3.

Таблица 3 – Средняя зарплата специалистов в 2023 году

Наименование специалиста	Средняя заработная плата в месяц, руб.	Средняя почасовая ставка, руб.
Системный аналитик	76180	476
Web-дизайнер	49582	310
Программист	96060	600
Тестировщик	65037	406

Так как в одном месяце 160 рабочих часов, отсюда следует, что почасовая ставка рассчитывается по формуле:

$$Y = \frac{X}{160}, \quad (1)$$

где Y – средняя почасовая ставка;

X – средняя заработная плата в месяц.

Для вышеперечисленных задач была рассчитана стоимость, которая зависит от количества затраченных дней и специалистов, которые участвуют в выполнении конкретной задачи. Стоимость задач представлена в таблице 4.

Таблица 4 – Стоимость задач проекта

Наименование задачи	Длительность задачи в днях	Участвующие специалисты	Затраты, руб.
Анализ требований	5	Системный аналитик	19040
Проектирование	10	Web-дизайнер, программист	72800
Реализация	15	Программист	72000
Тестирование	10	Тестировщик	32480
Оформление ПЗ	11	Системный аналитик	41888

Результатом проведенного анализа являются стоимость каждой задачи и суммарная стоимость проекта. Для выполнения этапа «Анализ требований» потребуется 5 дней, что при средней зарплате системного аналитика будет стоить 19040 рублей. На выполнение этапа «Проектирование» выделено 10 дней совместной работы Web-дизайнера и программиста, что, в свою очередь, стоит 72800 рублей. Этапом «Реализация» полностью занимается программист на протяжении 15 дней, из-за чего стоимость данного этапа равна 72000 рублей. Этап «Тестирование» выполняется тестировщиком в течение 10 дней, что при его средней зарплате будет стоить 32480 рублей. Этапом «Оформление ПЗ» также занимается системный аналитик. На данный этап будет потрачено 11 дней, что в денежном эквиваленте равно 41888 рублям.

Таким образом, можно сделать вывод, что суммарно на проект будет затрачен 41 день, а его стоимость будет равна 238208 рублям.

2 Анализ информационных технологий разработки

2.1 Инструментальные средства проектирования

UML – язык графического описания для объектного моделирования в области разработки программного обеспечения, для моделирования бизнес-процессов, системного проектирования и отображения организационных структур.

UML является языком широкого профиля, это – открытый стандарт, использующий графические обозначения для создания абстрактной модели системы, называемой UML-моделью. UML был создан для определения, визуализации, проектирования и документирования, в основном, программных систем. UML не является языком программирования, но на основании UML-моделей возможна генерация кода [6].

Преимущества использования UML заключаются в том, что он способствует улучшению понимания системы, увеличивает коммуникацию и сотрудничество между членами команды разработки, упрощает документирование и обеспечивает основу для автоматизированных инструментов, поддерживающих разработку и анализ систем.

Так как UML является языком, то для каждого языка свойственен стандартный набор элемент, в который входит словарь.

Для языка моделирования UML словарь представлен в виде трех групп элементов:

– диаграммы;
– отношения;
– сущности.

У каждого типа элемента и отношения в UML есть свои семантические правила и нотации, которые позволяют точно описать и визуализировать модель системы. Комбинация различных сущностей, диаграмм и отношений позволяет создавать полное и понятное представление о системе, ее структуре, поведении и взаимодействии.

2.2 Инструментальные средства разработки

Рассмотрим инструментальные средства, которые будут использованы в разработке клиент-серверного приложения для подбора поддержанных автомобилей с учетом предпочтений пользователя.

2.2.1 Язык программирования Java

Java – это кроссплатформенный объектно-ориентированный язык программирования общего назначения. Язык проектировался с целью минимизации зависимостей от реализации так, чтобы однажды написанное и скомпилированное приложение работало на всех платформах, поддерживаемых Java, без надобности перекомпиляции. Java популярна благодаря своей простоте, надежности и безопасности. Ее дизайн позволяет создавать модульные программы из повторно используемого кода. Она широко используется в приложениях корпоративного уровня [7].

Важной характеристикой Java является ее объектно-ориентированный подход. Язык полностью основан на принципах объектно-ориентированного программирования (ООП), что позволяет создавать модульные, гибкие и легко поддерживаемые приложения. ООП позволяет разбить программу на небольшие, самодостаточные и повторно используемые блоки кода, называемые объектами, которые взаимодействуют друг с другом для выполнения задач.

Безопасность является еще одним важным аспектом Java. Язык предлагает механизмы безопасности, которые помогают предотвращать ошибки и уязвимости в приложениях. Это включает в себя контроль границ массивов, автоматическое управление памятью, проверку типов на этапе компиляции и исполнения, что обеспечивает надежность и защиту данных.

Java также обладает встроенной поддержкой многопоточности, что позволяет создавать параллельные приложения, способные выполнять несколько задач одновременно. Это особенно полезно для разработки масштабируемых и отзывчивых приложений, которые могут эффективно использовать ресурсы системы.

2.2.2 Фреймворк Spring

Spring Framework – это мощный и гибкий инструмент для разработки Java-приложений. Он предоставляет обширный набор модулей и интеграционных возможностей, которые значительно упрощают процесс разработки и поддержки приложений [8].

Spring состоит из нескольких ключевых компонентов, каждый из которых имеет свою роль и функциональность. Одним из основных элементов Spring является контейнер приложения, известный как Spring IoC (Inversion of Control). IoC контейнер управляет созданием и жизненным циклом объектов в приложении, освобождая разработчика от необходимости явно создавать экземпляры объектов. Это обеспечивает гибкость и легкость интеграции различных компонентов приложения.

Еще одним важным компонентом Spring является модуль Spring MVC (Model-View-Controller). Spring MVC предоставляет структуру и инструменты для разработки веб-приложений, используя паттерн MVC. Он разделяет приложение на три основных компонента: модель, представление и контроллер. Модель представляет данные, представление отвечает за отображение данных пользователю, а контроллер обрабатывает входящие запросы и управляет потоком данных между моделью и представлением. Spring MVC упрощает разработку веб-приложений, обеспечивая гибкую маршрутизацию запросов, управление сессиями, обработку форм и другие функции.

Spring также предоставляет инструменты для работы с базами данных через модуль Spring JDBC (Java Database Connectivity). Он упрощает взаимодействие с базами данных, предоставляя удобные API для выполнения SQL-запросов, управления транзакциями и обработки результатов. Spring JDBC обеспечивает абстракцию от специфических драйверов баз данных и позволяет разработчикам легко интегрировать базы данных в свои приложения.

Другим важным компонентом Spring является модуль Spring Security. Он предоставляет инструменты для обеспечения безопасности приложений,

включая аутентификацию и авторизацию пользователей, управление правами доступа, защиту от атак и другие функции. Spring Security позволяет разработчикам легко внедрять механизмы безопасности в свои приложения и защищать их от потенциальных угроз.

Spring также имеет множество других модулей и инструментов, таких как Spring AOP (Aspect-Oriented Programming) для аспектно-ориентированного программирования, Spring ORM (Object-Relational Mapping) для работы с объектно-реляционными отображениями, Spring Boot для упрощения создания самостоятельных приложений и многое другое. В целом, Spring предоставляет разработчикам мощный и гибкий фреймворк для разработки разнообразных приложений на языке Java.

2.2.3 Язык программирования JavaScript

JavaScript – это кроссплатформенный, интерпретируемый язык программирования, который используется для создания интерактивных веб-страниц и приложений [9].

JavaScript имеет ряд особенностей, которые делают его уникальным и мощным инструментом для создания интерактивных веб-приложений. Во-первых, он является скриптовым языком, что означает, что программы на JavaScript выполняются непосредственно в браузере клиента. Это позволяет веб-страницам реагировать на действия пользователя и динамически изменять свое содержимое без необходимости обращения к серверу.

Одной из главных особенностей JavaScript является его возможность манипулировать элементами веб-страницы и изменять их свойства и содержимое. С помощью JavaScript можно создавать анимации, выполнять валидацию форм, изменять стили элементов, добавлять и удалять элементы из DOM (Document Object Model), что позволяет динамически обновлять страницу без ее полной перезагрузки.

JavaScript также поддерживает объектно-ориентированное программирование (ООП), что позволяет разрабатывать сложные и

масштабируемые приложения. Он позволяет создавать классы, объекты и наследование, что способствует повторному использованию кода и упрощает его структурирование. ООП в JavaScript отличается от классического ООП, так как использует прототипное наследование.

Другой важной особенностью JavaScript является его поддержка асинхронного программирования. Веб-страницы могут содержать множество асинхронных операций, таких как загрузка данных с сервера, обращение к API или выполнение сложных вычислений. JavaScript предоставляет различные механизмы, такие как колбэки, промисы и асинхронные функции, для эффективного управления асинхронными задачами и предотвращения блокировки интерфейса пользователя.

JavaScript также является мультипарадигменным языком, что означает, что он поддерживает не только процедурное и объектно-ориентированное программирование, но и функциональное программирование. Это позволяет разработчикам выбирать подход, который лучше всего подходит для конкретной задачи или стиля программирования.

2.2.4 Фреймворк React

React – это библиотека JavaScript с открытым кодом для создания внешних пользовательских интерфейсов. В отличие от других библиотек JavaScript, предоставляющих полноценную платформу приложений, React ориентируется исключительно на создание представлений приложений через инкапсулированные единицы (называются компонентами), которые сохраняют состояние и генерируют элементы пользовательского интерфейса [10].

Одной из особенностей React является его виртуальный DOM (Document Object Model), который представляет собой легковесное представление реального DOM. Вместо непосредственного обновления DOM при каждом изменении данных, React использует виртуальный DOM для эффективного рендеринга только тех компонентов, которые изменились. Это позволяет снизить нагрузку на браузер и повысить производительность приложения.

Еще одной особенностью React является его однонаправленный поток данных, известный как "управляемый компонентами". Данные в React передаются вниз по иерархии компонентов с помощью пропсов (props). Компоненты не могут изменять пропсы напрямую, они могут только передавать их дочерним компонентам. Это способствует более предсказуемому и контролируемому потоку данных, что упрощает отладку и сопровождение кода.

React также обладает модульной структурой, которая позволяет разработчикам создавать независимые компоненты с собственным состоянием (state) и методами жизненного цикла. Это позволяет разделить сложные пользовательские интерфейсы на более мелкие и управляемые части, что облегчает разработку и тестирование.

Другой важной особенностью React является его возможность работать с другими библиотеками и фреймворками, такими как Redux или React Router. React является гибким и расширяемым, что позволяет разработчикам выбирать инструменты, которые лучше всего соответствуют их потребностям и предпочтениям.

2.2.5 Язык гипертекстовой разметки HTML

HTML – стандартизированный язык гипертекстовой разметки документов для просмотра веб-страниц в браузере. Веб-браузеры получают HTML документ от сервера по протоколам HTTP/HTTPS или открывают с локального диска, далее интерпретируют код в интерфейс, который будет отображаться на экране монитора [11].

Одной из основных особенностей HTML является его простота и легкость изучения. Синтаксис HTML интуитивно понятен и не требует особых навыков программирования. Это делает HTML доступным для широкого круга пользователей и способствует созданию веб-страниц без необходимости обширных знаний в программировании.

HTML также обладает своей структурой и семантикой. Он использует теги для определения различных элементов, таких как заголовки, параграфы, списки,

таблицы и медиа-элементы. Это позволяет разработчикам ясно определить структуру и смысловую иерархию веб-страницы, что полезно для поисковых систем и доступности.

HTML также предоставляет возможность создания гиперссылок. Гиперссылки позволяют связывать веб-страницы между собой и создавать навигацию в интернете. Это основной механизм для перехода по страницам, открытия новых вкладок или загрузки файлов.

2.2.6 Каскадные таблицы стилей CSS

CSS – формальный язык декорирования и описания внешнего вида документа (веб-страницы), написанного с использованием языка разметки (чаще всего HTML или XHTML) [12].

Одной из основных особенностей CSS является его возможность разделения стилей и содержимого. С помощью CSS разработчики могут определить стили для различных элементов веб-страницы, независимо от их содержимого. Это позволяет легко изменять внешний вид веб-сайта, не затрагивая его структуру и содержимое.

CSS также предоставляет различные методы для выбора и настройки элементов на странице. Селекторы CSS позволяют разработчикам выбирать элементы по их типу, классу, идентификатору и другим атрибутам. Это позволяет применять стили к определенным элементам или группам элементов, что обеспечивает гибкость и контроль над оформлением страницы.

Еще одной важной особенностью CSS является его каскадность. CSS-правила могут быть применены к элементам, и, если возникают конфликты между правилами, CSS использует каскадную модель для определения, какие стили будут применены. Это позволяет легко управлять приоритетом стилей и создавать различные эффекты и состояния элементов.

CSS также предоставляет множество свойств и значений для настройки внешнего вида элементов. Разработчики могут устанавливать цвета, шрифты, размеры, отступы, границы, фоны и другие атрибуты элементов, чтобы достичь

желаемого визуального эффекта. Это позволяет создавать уникальные и привлекательные дизайны веб-страниц.

2.2.7 Выбор СУБД

Система управления базами данных (СУБД) – это набор программ, которые управляют структурой БД и контролируют доступ к данным, хранящимся в БД.

СУБД служит посредником между пользователем и БД. Сама структура БД хранится в виде набора файлов, и единственный способ получить доступ к данным в этих файлах – через СУБД. Она представляет конечному пользователю (или прикладной программе) единое интегрированное представление данных в БД. СУБД получает запросы приложений и переводит их в сложные операции, необходимые для выполнения [13].

SQL базы данных, основанные на реляционной модели, предлагают структурированный и строго организованный подход к хранению и обработке данных. Они используют таблицы, связи и схемы, чтобы обеспечить целостность данных и поддерживать ACID) – набор свойств, гарантирующих надежность и непрерывность операций. SQL базы данных хорошо подходят для проектов, где требуется строгая схема данных, сложные запросы и связи между таблицами.

NoSQL базы данных предлагают гибкий и распределенный подход к хранению данных. Они обычно не имеют строгой схемы и позволяют хранить данные в виде документов, ключ-значение пар, графов или широких столбцов. Это делает их более масштабируемыми и гибкими, особенно при работе с большими объемами данных и приложениями, где требуется высокая производительность и отказоустойчивость. NoSQL базы данных также часто предоставляют горизонтальное масштабирование, что позволяет легко добавлять новые узлы для обработки данных.

Так как объект объявления может менять свою структуру и количество атрибутов, для клиент-серверного приложения для подбора поддержанных

автомобилей с учетом предпочтений пользователя наиболее подходящим вариантом является NoSQL база данных.

В разработке проекта используется СУБД MongoDB, так как она проста в установке и использовании, а также имеет бесплатную версию.

3 Анализ функциональных требований к разрабатываемому приложению

Функциональные требования к системе – это набор функций, которые должно выполнять разрабатываемое приложение.

Функции представляются в виде прецедентов, а функциональные требования отображаются на диаграмме вариантов использования.

Диаграмма прецедентов или диаграмма вариантов использования в UML – диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне [14].

Для отражений функциональный требований системы была создана диаграмма вариантов использования, представленная на рисунке 2.

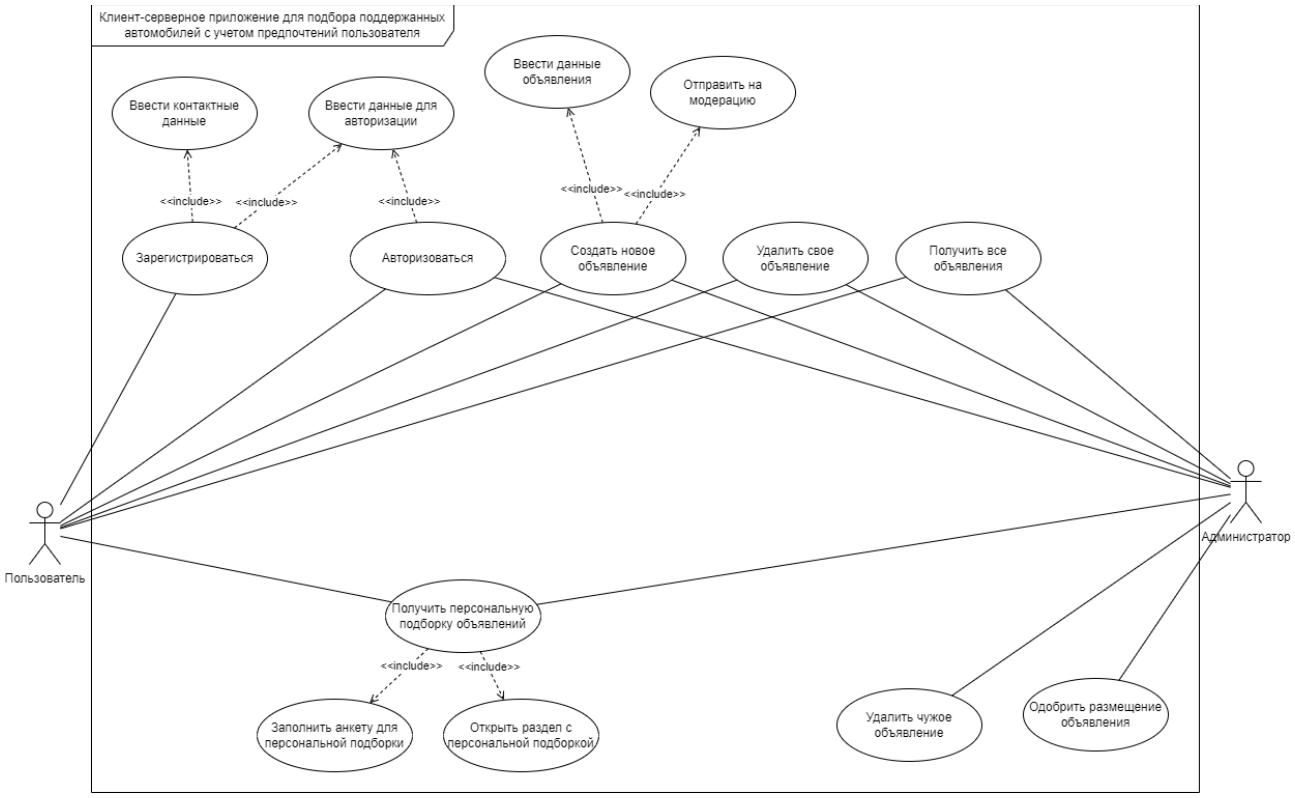

Рисунок 2 – Диаграмма вариантов использования

В данном приложении участвуют два основных актера: пользователь и администратор.

Пользователь имеет возможность зарегистрировать учетную запись, а также авторизоваться под существующей учетной записью. Варианты использования «Авторизоваться» и «Зарегистрироваться» включают в себя вариант использования «Ввести данные авторизации», которые представляют из себя имя пользователя и пароль. При регистрации также пользователь должен ввести свои контактные данные, которые будут отображаться при публикации объявления о продаже.

Пользователь также может создавать новые и удалять существующие объявления, но только те, которые ему принадлежат.

У пользователя есть возможность получить все объявления или персональную подборку, для которой нужно предварительно заполнить анкету пользователя, а после перейти в соответствующий раздел.

Администратор имеет те же функции, что и пользователь, за исключением возможности регистрации, так как подразумевается, что администратором может стать только зарегистрированный пользователь. Также администратор имеет дополнительные возможности: удаление чужих объявление и одобрение объявлений для публикации.

4 Проектирование клиент-серверного приложения

Проектирование приложения направлено на достижение эффективности и оптимизации его работы. При этом уделяется внимание выбору наиболее подходящих алгоритмов, структур данных и архитектурных решений, чтобы обеспечить оптимальное использование ресурсов и достижение высокой производительности.

Одним из важных аспектов проектирования приложения является его гибкость и масштабируемость. Правильное проектирование учитывает возможные изменения требований, рост объемов данных и пользовательской нагрузки. Для этого выбираются архитектурные решения и используются соответствующие технологии, позволяющие легко модифицировать и масштабировать приложение без больших затрат.

Проектирование приложения также направлено на облегчение его сопровождения и улучшения в долгосрочной перспективе. Четкая структура, модульность и понятный код помогают команде разработчиков легко понимать и вносить изменения в приложение. Это особенно важно в случае постоянного развития и обновления приложения, когда требуется внедрять новые функциональные возможности или исправлять ошибки.

4.1 Проектирование структуры клиент-серверного приложения

Структура данного приложения была спроектирована с использованием шаблона проектирования Repository.

Паттерн проектирования Repository (репозиторий) является одним из популярных структурных паттернов, используемых в разработке программного обеспечения. Он предоставляет абстракцию для доступа к данным и скрывает детали их хранения и взаимодействия с хранилищем данных.

Основная цель паттерна Repository состоит в создании единообразного интерфейса для работы с данными, независимо от источника данных (например, базы данных, файловой системы, веб-службы и т.д.). Он позволяет изолировать

клиентский код от конкретных реализаций хранилища данных и упрощает поддержку и модификацию приложения в случае изменения источника данных.

Структура приложения состоит из сервисов, репозиториев, сущностей и контроллеров. Вышеописанный фреймворк Spring имеет предопределенную реализацию репозиториев. Они позволяют получить одну или все записи, создать и удалить запись базы данных.

Также структура приложения разделена на 2 подсистемы:

– подсистема авторизации;

– подсистема для работы с объявлениями.

Подсистема авторизации включает в себя репозиторий, сервис и контроллер, которые манипулируют данными пользователя, а также создают, удаляют и проверяют текущую сессию учетной записи.

Диаграмма классов подсистемы авторизации представлена на рисунке 3.

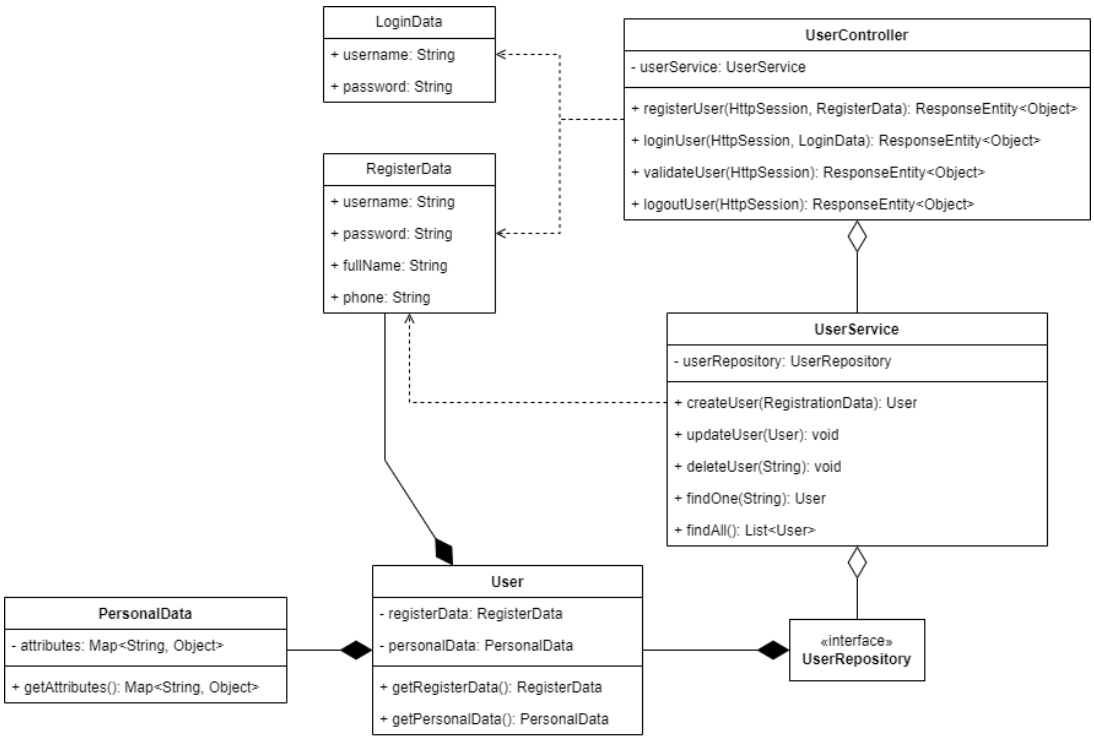

Рисунок 3 – Диаграмма классов подсистемы авторизации

Основным классом подсистемы является «UserController». Он отвечает за обработку полученных запросов от клиента и взаимодействует с сервисом пользователей.

Класс «UserController» содержит следующие методы:

– registerUser, типы данных аргументов – HttpSession, RegisterData;

– loginUser, типы данных аргументов – HttpSession, LoginData;

– validateUser, типы данных аргументов – HttpSession;

– logoutUser, типы данных аргументов – HttpSession.

Каждый из вышеперечисленных методов возвращает тип данных – ResponseEntity.

Метод registerUser обрабатывает клиентские запросы, отправленные по REST-пути «/account/register» и регистрирует пользователя по тем данным, которые указаны в классе «RegisterData».

Метод loginUser обрабатывает клиентские запросы, отправленные по REST-пути «/account/login» и создает сессию учетной записи пользователя, если класс «LoginData» содержит корректные данные.

Метод validateUser обрабатывает клиентские запросы, отправленные по REST-пути «/account/validate» и возвращает клиенту объект пользователя, если на данный момент времени существует корректная сессия учетной записи.

Метод logoutUser обрабатывает клиентские запросы, отправленные по REST-пути «/account/logout» и сбрасывает текущую сессию учетной записи, если она существует.

Также данная подсистема содержит следующие классы:

– класс «UserService», назначение – содержит методы для создания, удаления, обновления и получения пользователей;

– интерфейс «UserRepository», назначение – содержит методы для взаимодействия с базой данных пользователей;

– класс «User», назначение – является сущностью пользователя;

– класс «RegisterData», назначение – вспомогательный класс, содержащий данные для регистрации пользователя, которые приходят от клиента;

– класс «LoginData», назначение – вспомогательный класс, содержащий данные для авторизации пользователя, которые приходят от клиента;

– класс «PersonalData», назначение – содержит данные анкеты пользователя, которая используется для персональной подборки.

Методы класса «UserService»:

– createUser, типы данных аргументов – RegisterData, тип возвращаемого значения – User;

– updateUser, типы данных аргументов – User;

– deleteUser, типы данных аргументов – String;

– findOne, типы данных аргументов – String, тип возвращаемого значения – User;

– findAll, тип возвращаемого значения – List<User>.

Метод createUser создает нового пользователя с учетными данными, которые были переданы в классе «RegisterDate» от клиентской части приложения. Внутри сервиса используется метод интерфейса «UserRepository» под названием save, который принимает объект пользователя и возвращает такой же объект с измененным идентификатором пользователя.

Метод updateUser обновляет уже существующие данные о пользователе, который был передан в качестве аргумента метода.

Метод deleteUser принимает строку в качестве аргумента, этой строкой является идентификатор пользователя, которого требуется удалить. Внутри сервиса вызывается метод интерфейса «UserRepository», который удаляет запись базы данных по идентификатору.

Метод findOne также принимает идентификатор пользователя в качестве аргумента, после чего интерфейс репозитория ищет в базе данных соответствующую запись. Если запись была найдена, то она и является возвращаемым значением, иначе будет возвращен «null».

Метод findAll не имеет аргументов и возвращает все объекты пользователей.

Подсистема для работы с объявлениями включает в себя репозиторий, сервис и контроллер, которые взаимодействуют с сущностью объявления о продаже автомобиля.

Диаграмма классов данной подсистемы представлена на рисунке 4.

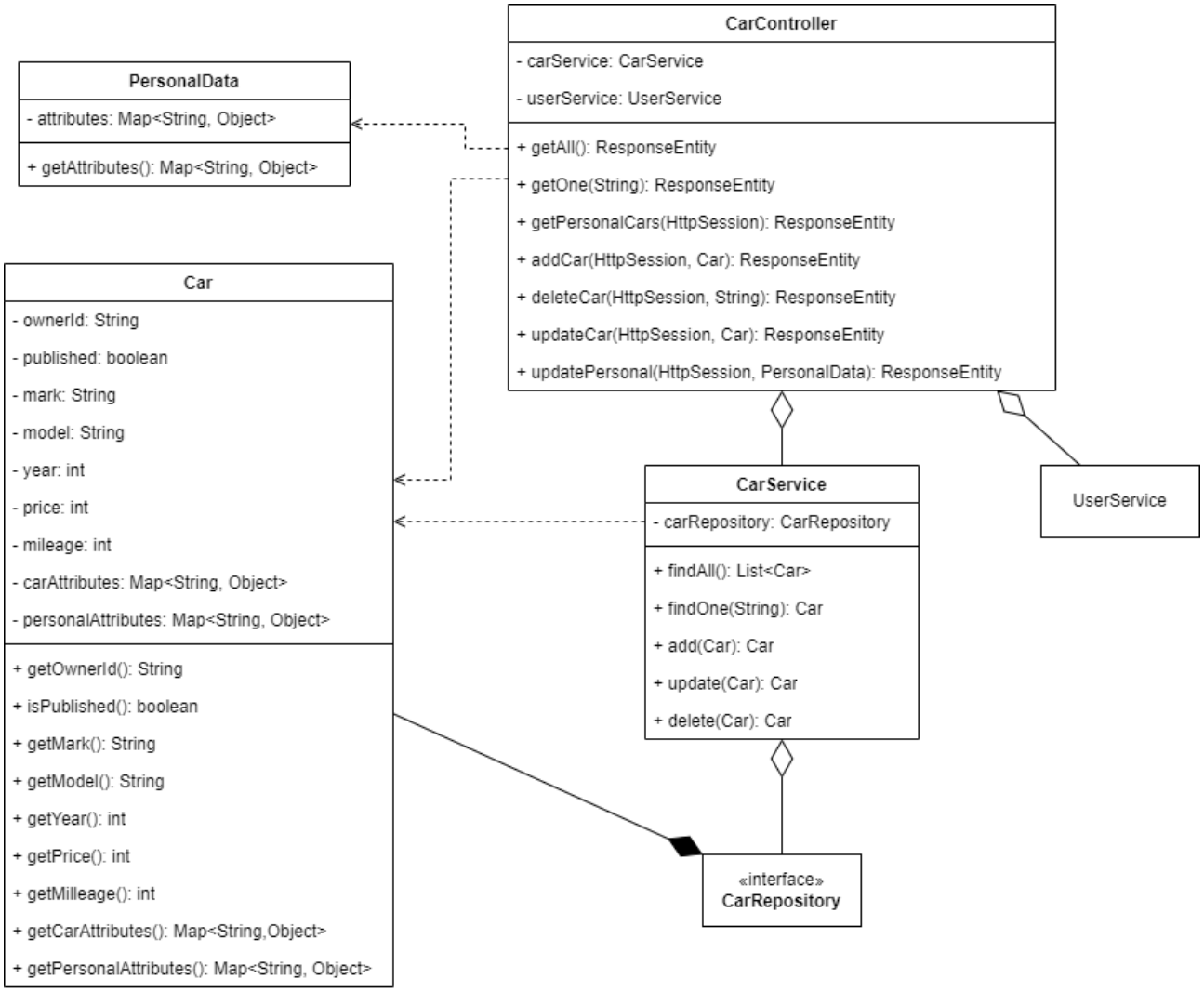

Рисунок 4 – Диаграмма классов подсистемы для работы с объявлениями

Данная подсистема включает в себя следующие классы:

– класс «CarController», назначение – обрабатывает запросы, связанные с получением и изменением объявлений, которые приходят от клиента;

– класс «CarService», назначение – предоставляет методы, упрощающие работу с репозиторием;

– интерфейс «CarRepository», назначение – содержит методы для взаимодействия с базой данных объявлений;

– класс «Car», назначение – является сущностью объявления о продаже автомобиля;

– классы «UserService» и «PersonalData» являются классами из вышеописанной подсистемы авторизации.

Методы класса «CarController»:

– getAll, аргументы отсутствуют;

– getOne, типы данных аргументов – String;

– getPersonalCars, типы данных аргументов – HttpSession;

– addCar, типы данных аргументов – HttpSession, Car;

– deleteCar, типы данных аргументов – HttpSession, String;

– updateCar, типы данных аргументов – HttpSession, Car;

– updatePersonal, типы данных аргументов – HttpSession, PersonalData.

Каждый из методов контроллера возвращает тип данных – ResponseEntity.

Метод getAll обрабатывает запрос по REST-пути «/cars» и возвращает все объявления о продаже автомобилей.

Метод getOne обрабатывает запрос по REST-пути «/cars/:id», принимает идентификатор объявления в качестве аргумента и возвращает объект объявления, если запись в базе данных существует, иначе будет возвращен «null».

Метод getPersonalCars обрабатывает запрос по REST-пути «/cars/personal» и возвращает персонально подобранные объявления. Если текущая сессия существует и корректно, то система получает текущего пользователя, проверяет, заполнена ли у него анкета, если да – выбирает объявления, которые соответствуют данным, указанным в анкете, в противном случае возвращается ошибка.

Метод addCar обрабатывает запрос по REST-пути «/cars/add» и добавляет новое объявление о продаже автомобиля. После этого администратор может опубликовать данное объявление.

Метод updateCar обрабатывает запрос по REST-пути «/cars/:id» и изменяет данные объявления.

Метод deleteCar обрабатывает запрос по REST-пути «/cars/:id» и удаляет объявление, если оно существует.

Методы класса «CarService» аналогичны методам класса «UserService», но в качестве аргументов вместе класса «User» принимают класс «Car» соответственно.

4.2 Проектирование пользовательского интерфейса

Пользовательский интерфейс представляет собой средство, с помощью которого конечный пользователь может взаимодействовать с разрабатываемым приложением и осуществлять обмен информацией с ним.

Разработка интерфейса является важным аспектом, поскольку качественно спроектированный интерфейс обеспечивает оптимальное взаимодействие пользователя с системой, позволяя ему эффективно использовать ее возможности.

Рассмотрим основные элементы интерфейса приложения.

В клиент-серверном приложении для подбора поддержанных автомобилей с учетом предпочтений пользователя главной страницей является меню выбора марки автомобиля и раздел с персональной подборкой объявлений.

При входе пользователя на веб-страницу, проверяется, авторизован он или нет. В случае, если пользователь авторизован, проверятся, заполнена ли его анкета для получения персональной подборки, если заполнена, то пользователю выводится список объявлений, подобранный персонально для него, иначе – выводится сообщение о том, что сначала нужно заполнить анкету, и кнопка, при нажатии на которую, пользователь переходит на страницу для заполнения анкеты.

Если же пользователь не авторизован, то система также информирует его об этом и предлагает войти в учетную запись.

Макет начальной страницы сервиса, когда игрок не авторизован, представлен на рисунке 5.

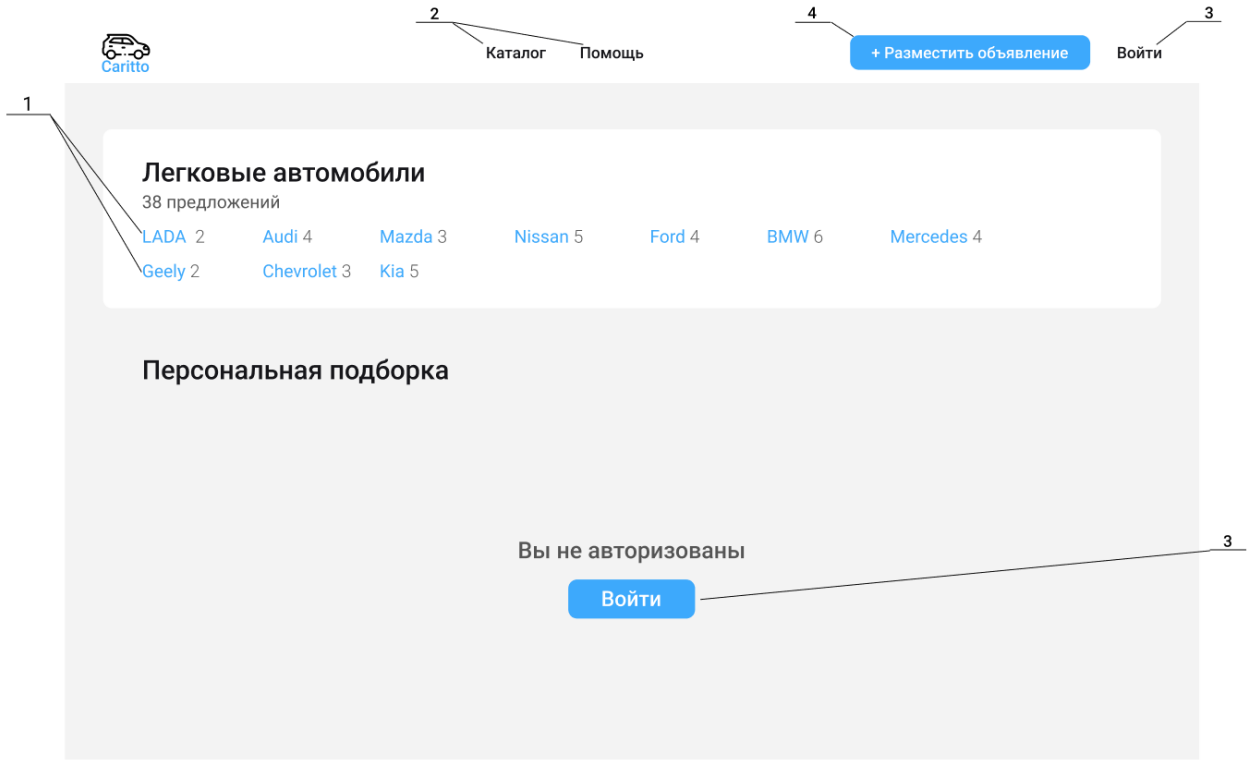

Рисунок 5 – Начальная страница, когда пользователь не авторизован

Рассмотрим основные элементы интерфейса начальной страницы, представленные на рисунке 5:

1 – меню навигации по маркам автомобиля;

2 – меню навигации по страницам сайта;

3 – кнопка перехода на форму авторизации;

4 – кнопка перехода на форму создания объявления.

Меню навигации по маркам автомобиля позволяет пользователю перейти на страницу, на которой размещены объявления по конкретной марке автомобиля.

Меню навигации по страницам сайта позволяет перейти в каталог всех объявлений или открыть страницу со справочной информацией.

Макет начальной страницы, когда пользователь авторизован, но не имеет заполненной анкеты, представлен на рисунке 6.

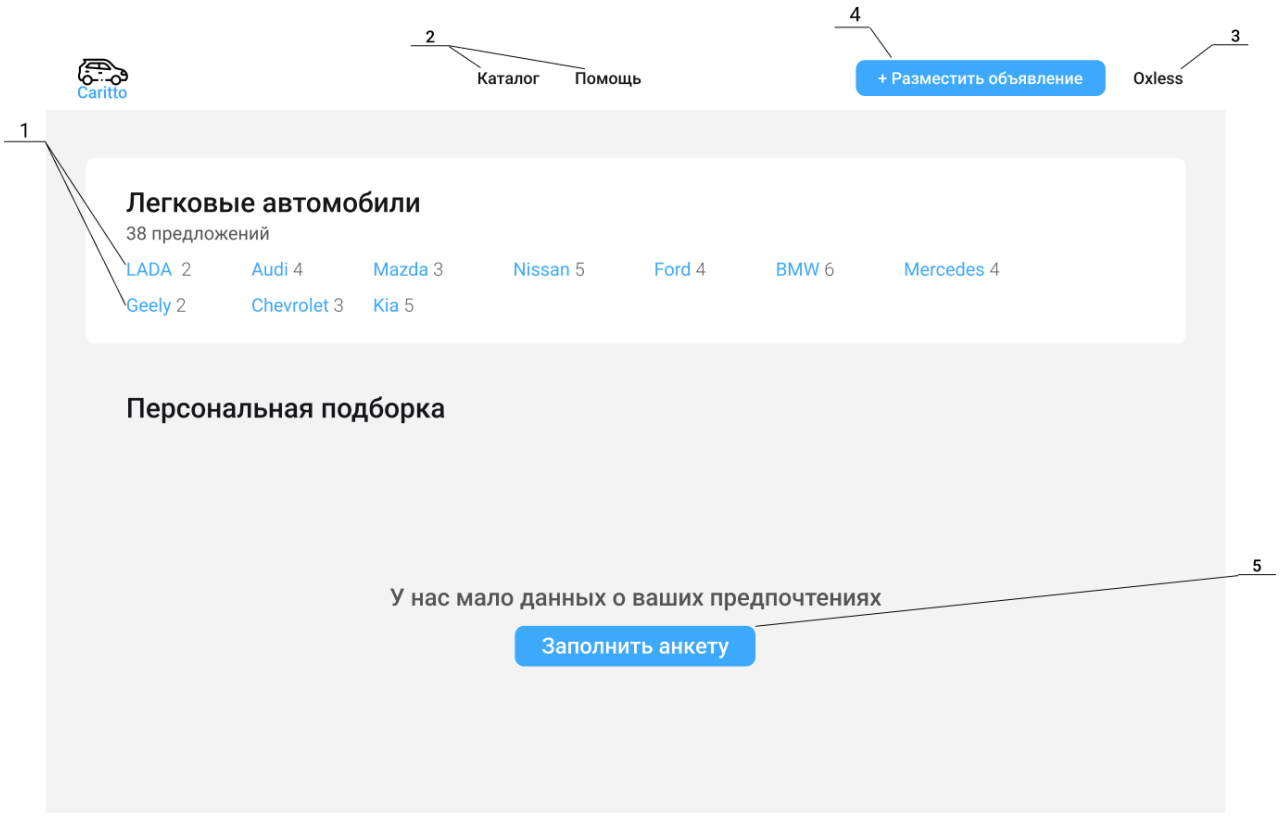

Рисунок 6 – Начальная страница, когда пользователь не заполнил анкету

Рассмотрим основные элементы интерфейса начальной страницы, представленные на рисунке 6:

1 – меню навигации по маркам автомобиля;

2 – меню навигации по страницам сайта;

3 – кнопка перехода в личный кабинет пользователя;

4 – кнопка перехода на форму создания объявления;

5 – кнопка перехода на форму заполнения анкеты.

При нажатии на имя пользователя, будет открыть личный кабинет, в котором можно увидеть опубликованные пользователем объявления.

При нажатии на кнопку «Заполнить анкету» пользователь будет перемещен на страницу с формой анкеты.

Макет начальной страницы с персональной подборкой объявлений представлен на рисунке 7.

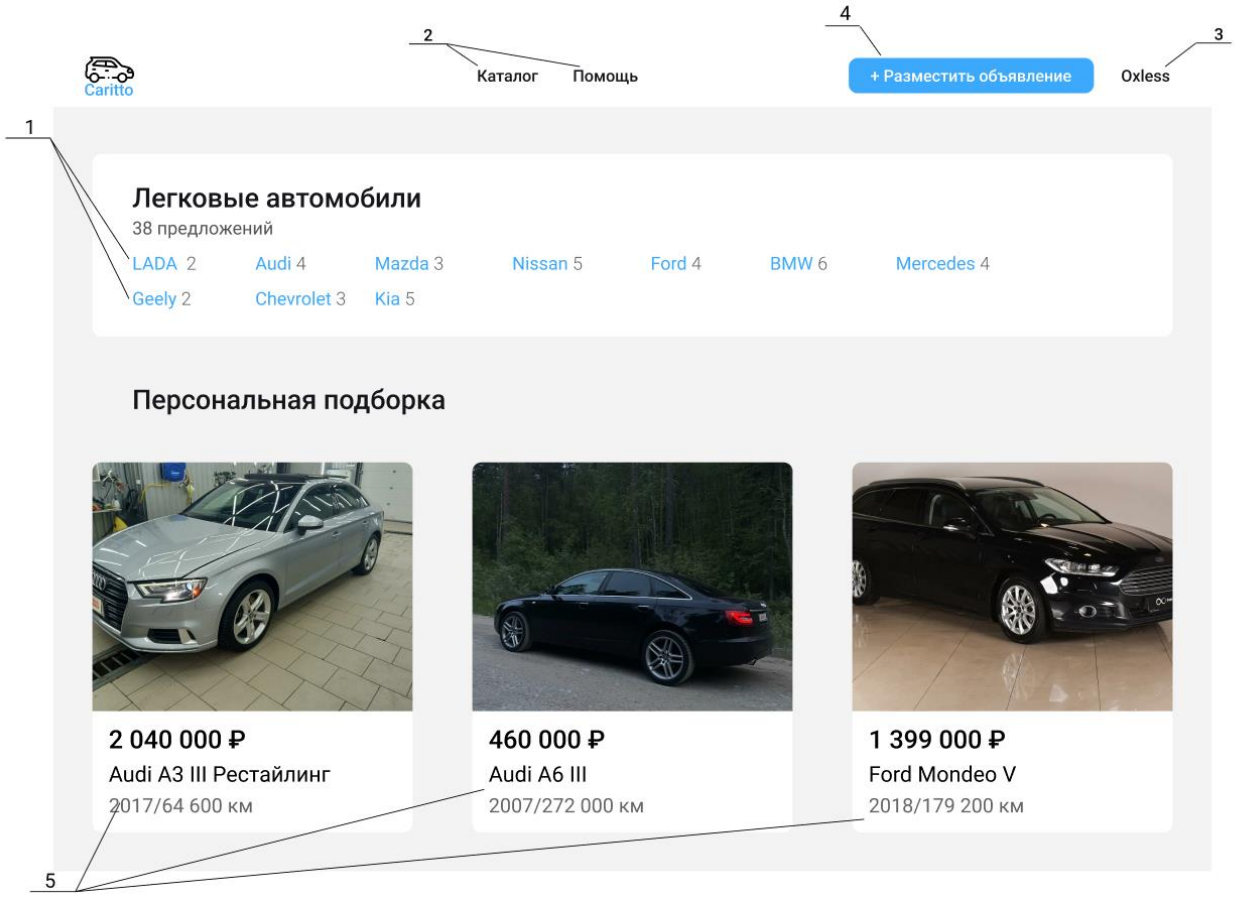

Рисунок 7 – Начальная страница с персональной подборкой объявлений

Рассмотрим основные элементы интерфейса начальной страницы с разделом персональной подборки, представленные на рисунке 7:

1 – меню навигации по маркам автомобиля;

2 – меню навигации по страницам сайта;

3 – кнопка перехода в личный кабинет пользователя;

4 – кнопка перехода на форму создания объявления;

5 – карточка автомобиля.

При нажатии по карточке автомобиля, пользователь будет перемещен на страницу с подробной информацией об автомобиле, на которой указаны основные характеристики, описание, адрес продажи, а также контактные данные продавца.

Макет формы заполнения анкеты для получения персональной подборки объявлений представлен на рисунке 8.

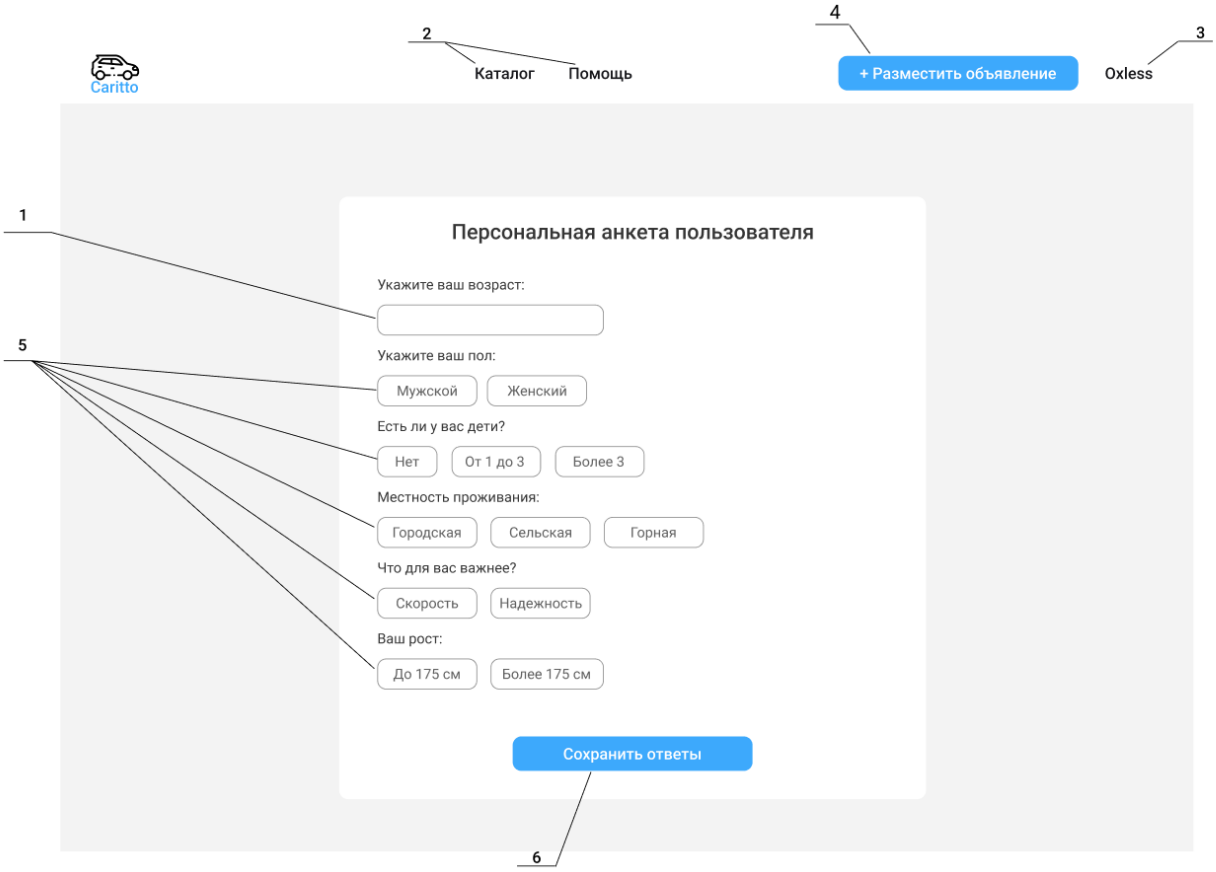

Рисунок 8 – Персональная анкета пользователя

Подробнее рассмотрим основные элементы интерфейса персональной анкеты пользователя, представленные на рисунке 8:

1 – поле для ввода возраста пользователя;

2 – меню навигации по страницам сайта;

3 – кнопка перехода в личный кабинет пользователя;

4 – кнопка перехода на форму создания объявления;

5 – варианты ответов;

6 – кнопка сохранения выбранных ответов.

Таким образом, были рассмотрены основные страницы клиент-серверного приложения для подбора поддержанных автомобилей с учетом предпочтений пользователей.

5 Реализация клиент-серверного приложения

Отдельно рассмотрим реализацию серверной и клиентской частей приложения.

5.1 Реализация серверной части приложения

Серверная часть клиент-серверного приложения для подбора поддержанных автомобилей представляет собой совокупность классов на языке программирования Java, размещенных в файлах с расширением java. Данные файлы используются для сборки исполняемого файла с расширением jar. Данный файл используется для запуска серверной части приложения.

Таким образом, серверная часть приложения была разделена на 12 модулей, представленных на диаграмме компонентов (рисунок 9).

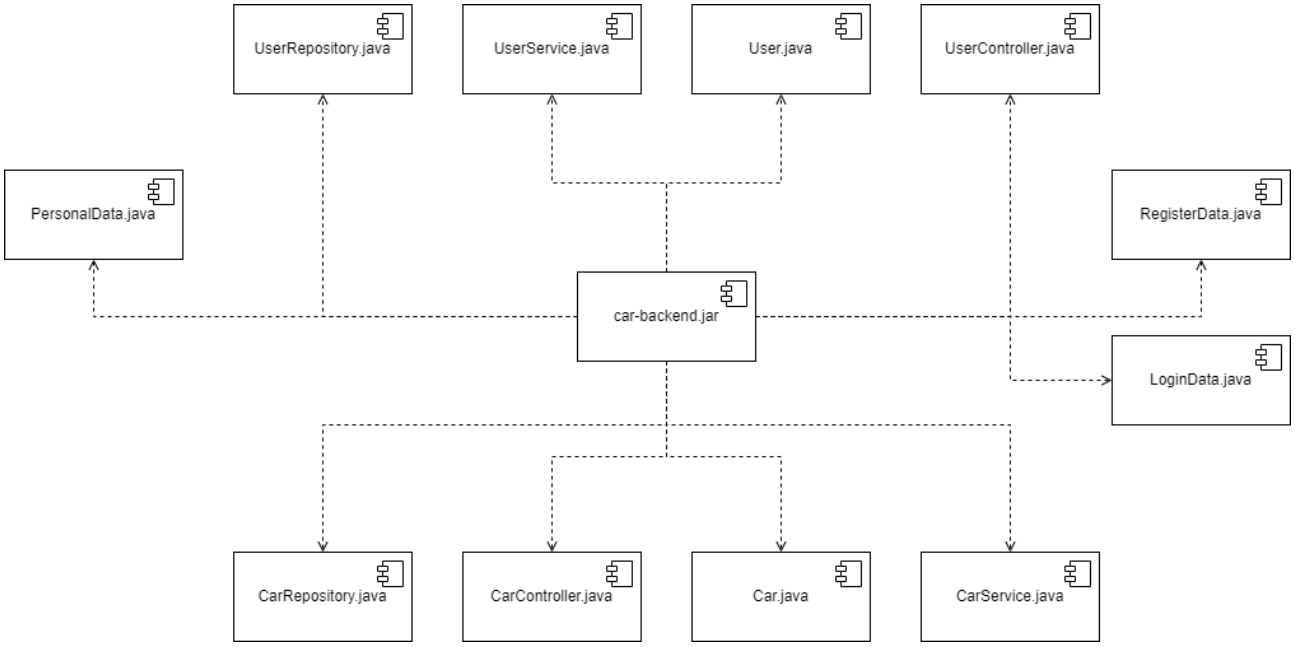

Рисунок 9 – Диаграмма компонентов серверной части

В результате сборки проекта, данные модули будут объединены в конечный файл car-backend.jar. Он включает в себя скомпилированные файлы-классы с расширением class.

Рассмотрим подробнее процесс подбора объявлений. После того, как пользователь заполнил персональную анкету и перешел в раздел с персональной

подборкой, клиентская часть приложения отправляет запрос на серверную часть, где происходит сравнение пользовательских данных с данными автомобилей, которые хранятся в базе данных, и в результате серверная часть возвращает список подходящих пользователю автомобилей.

5.2 Реализация клиентской части приложения

Клиентская часть приложения разработана на языке JavaScript с применением фреймворка React. Страницы, которые видит пользователь, полностью состоят из компонентов, которые приведены в диаграмме компонентов на рисунке 10.

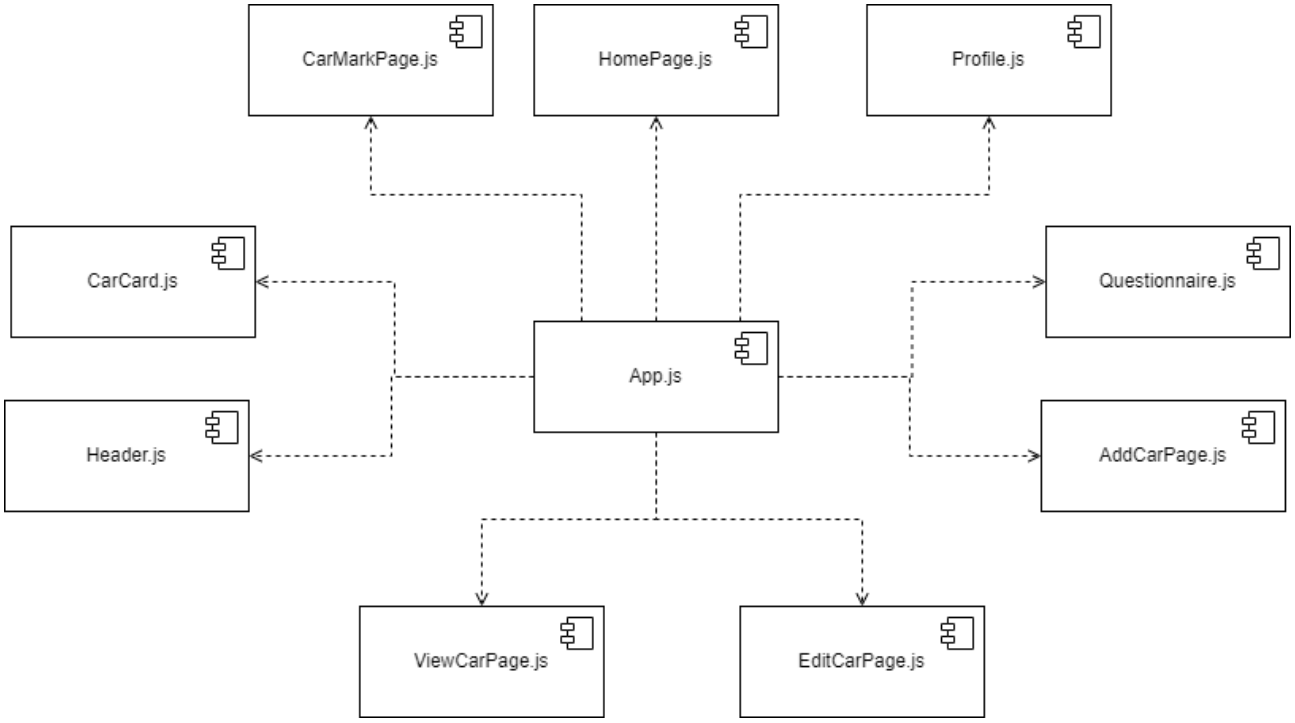

Рисунок 10 – Диаграмма компонентов клиентской части

Модуль HomePage является начальной страницей клиент-серверного приложения и содержит в себе раздел с персональной подборкой объявлений.

Модуль CarMarkPage содержит список автомобилей определенной марки.

Модуль Profile является личным кабинетом пользователя.

Модуль Questionnaire реализует форму персональной анкеты пользователя.

Модули EditCarPage и AddCarPage служат для редактирование и добавления новых объявлений соответственно.

Модуль Header содержит в себе меню навигации, которое расположено вверху экрана.

Модуль ViewCarPage является страницей автомобиля, на котором подробно описаны его характеристики.

Модуль CarCard содержит короткую информацию об объявлении.

Модуль App является основным, которые объединяет все вышеперечисленные модули.

6 Тестирование «Клиент-серверного приложения для подбора поддержанных автомобилей с учетом предпочтений пользователей»

Тестирование программного обеспечения – процесс анализа программного средства и сопутствующей документации с целью выявления дефектов и повышения качества продукта [15].

В ходе анализа способов тестирования программного обеспечения было определено, что более подходящими из них для данного приложения являются: тестирование методом черного ящика, тестирование метрики кода, а также автоматизированное тестирование.

Для определения сложности программного кода обработки пользовательских данных при авторизации были использованы метрики Холстеда.

6.1 Тестирование формы для ввода марки автомобиля

Метод черного ящика применятся в тех случаях, когда тестировщику неизвестны структура системы и применяющиеся в ней алгоритмы. Этот способ был выбран для тестирования формы ввода марки автомобиля, так как нужно сымитировать действия пользователя, который не обладает информацией о том, как именно работает этот компонент приложения.

Для реализации тестирования методом черного ящика были использованы функциональная диаграмма, классы эквивалентности и таблица решений.

Классом эквивалентности называется набор данных, который запускает одни и те же модули и должен приводить к одним и тем же результатам [16].

В приведенном примере тестируется форма ввода марки автомобиля, которая предусматривает, что введенное значение должно быть длиной от 3 до 20 символов включительно, должно начинаться с буквы и состоять из букв латинского алфавита или кириллицы, а также может содержать пробел.

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

Представление классов эквивалентности для тестирования формы ввода марки автомобиля представлено в таблице 5.

Таблица 5 – Классы эквивалентности: табличное представление

	Правильные КЭ	Неправильные КЭ
Длина названия	3-20 (1)	2 (2) 21- ∞ (3)
Структура	Начинается с буквы (4)	Начинается не с буквы (5)
Содержание	Буквы Кириллица (6) Буквы Латиница (7) Пробел (8)	Остальное (9)

Классы эквивалентности в графическом виде представлены на рисунке 11.

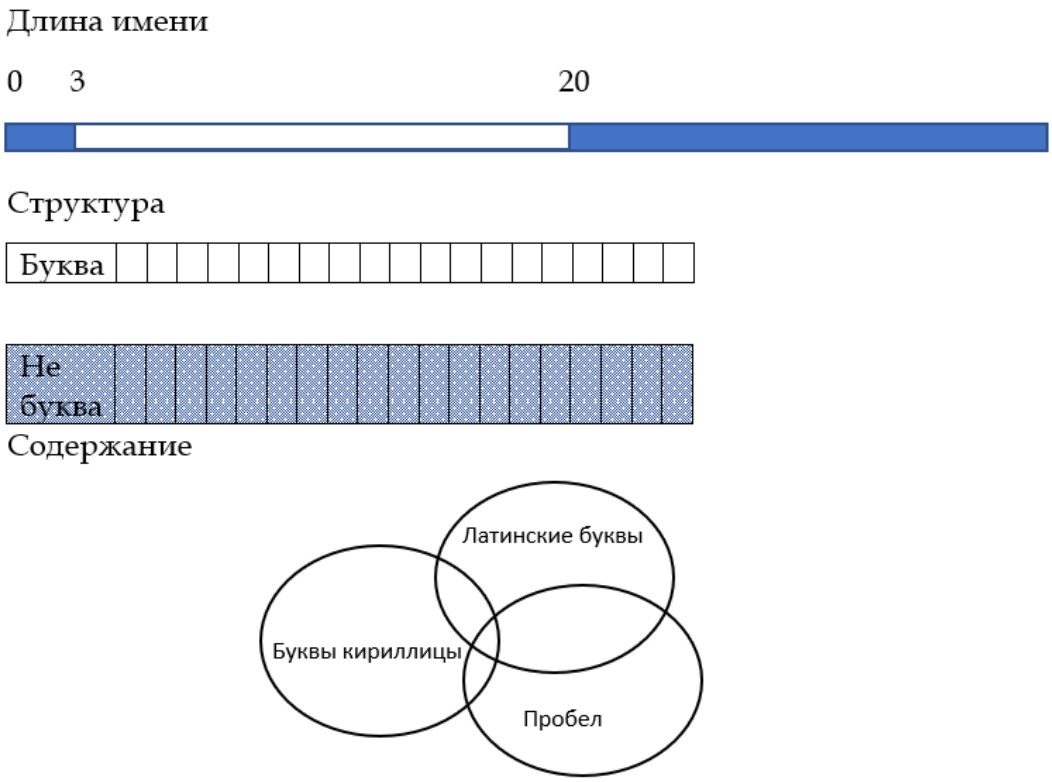

Рисунок 11 – Графическое представление классов эквивалентности

Графическое представление классов эквивалентности показывает структуру, содержание, а также другие ограничения вводимого значения.

На основе классов эквивалентности были составлены тест-кейсы, которые состоят из списка входящих классов эквивалентности, а также из входных и выходных данных. Тест-кейс представлены в таблице 6.

Таблица 6 – Классы эквивалентности: тест-кейсы

№	КЭ	Входные данные	Выходные данные
0	Все правильные 1, 4, 6, 7, 8	Mazda	Успешно
1	2, 4, 6, 7	P1	Ошибка: меньше 3х символов
2	3, 4, 6, 7, 8	Zbshudsujгрвгыаргвацу123432	Ошибка: больше 20 символов
3	1, 5, 6, 7, 8	3LadaЛада	Ошибка: начинается не с буквы
4	1, 4, 9	Ла-да	Ошибка: имеет неразрешенный символ

Таблица причин и следствий, построенная для тестирования формы добавления нового автомобиля, представлена в таблице 7.

Таблица 7 – Причины и следствия

Причины	
П1	Название введено
П2	Название корректно
П3	Год выпуска введен
П4	Год выпуска корректен
Следствия	
С1	Сообщение: «Название не введено»
С2	Сообщение: «Название введено не корректно»
С3	Сообщение: «Год выпуска не введен»
С4	Сообщение: «Год выпуска введен не корректно»
С5	Сообщение: «Вы будете перемещены на следующую страницу»

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

Функциональная диаграмма представлена на рисунке 12.

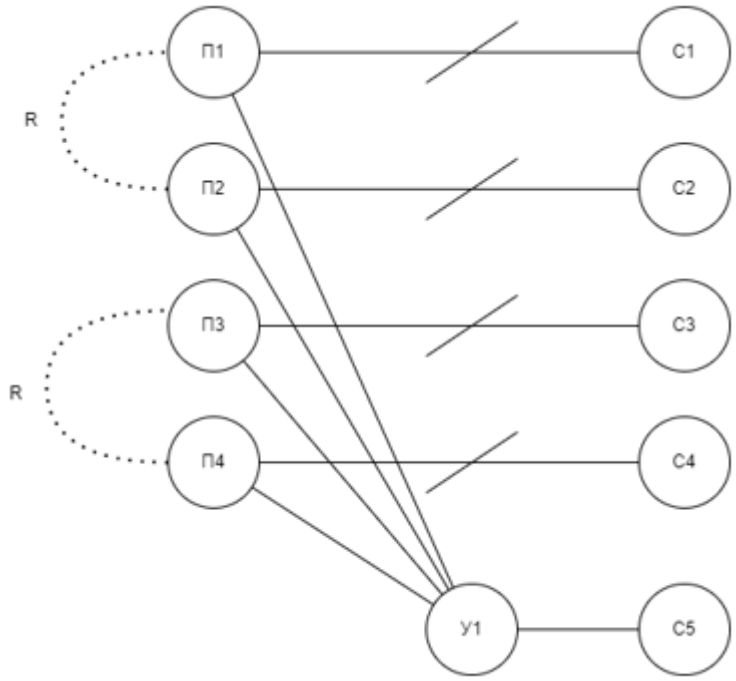

Рисунок 12 – Функциональная диаграмма причин и следствий

На диаграмме показано, что если первая причина не выполняется, то выполняется первое следствие. Таким же образом, если вторая причина не выполняется, то выполняется второе следствие. Также вторая причина зависит от первой: если первая причина не будет выполнена, то вторая тоже не может быть выполненной, так как название марки автомобиля не может быть корректным, если он не введено вовсе.

Аналогично этому, если третья причина не выполняется, то будет выполнено третье следствие, и, если четвертая причина не выполняется, то выполняется четвертое следствие. Четвертая причина также зависит от третьей причины, так как дата выпуска не может быть корректной, если она не заполнена.

В случае, если все причины успешно выполнены, то выполняется пятое следствие, которое сообщает о том, что автомобиль добавлен.

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

Таблица решений представлена в таблице 8.

Таблица 8 – Таблица решений

	1	2	3	4	5	6	7	8	9
П1	0	1	1	0	0	1	1	1	1
П2	0	0	1	0	0	0	0	1	1
П3	0	0	0	1	1	1	1	1	1
П4	0	0	0	0	1	0	1	0	1
С1	1	0	0	1	1	0	0	0	0
С2	1	1	0	1	1	1	1	0	0
С3	1	1	1	0	0	0	0	0	0
С4	1	1	1	1	0	1	0	1	0
С5	0	0	0	0	0	0	0	0	1

Далее были составлены тест-кейсы, представленные в таблице 9.

Таблица 9 – Таблица решений: тест-кейсы

№	Входные данные	Выходные данные
1	Марка: Год выпуска:	Сообщение: «Название не введено» Сообщение: «Название не корректно» Сообщение: «Год выпуска не введен» Сообщение: «Год выпуска не корректен»
2	Марка: Mazd1 Год выпуска:	Сообщение: «Название не корректно» Сообщение: «Год выпуска не введен» Сообщение: «Год выпуска не корректен»
3	Марка: Mazda Год выпуска:	Сообщение: «Год выпуска не введен» Сообщение: «Год выпуска не корректен»
4	Марка: Год выпуска: 1799	Сообщение: «Название не введено» Сообщение: «Название не корректно» Сообщение: «Год выпуска не корректен»
5	Марка: Год выпуска: 1945	Сообщение: «Название не введено» Сообщение: «Название не корректно»
6	Марка: Mazd1 Год выпуска: 1700	Сообщение: «Название не корректно» Сообщение: «Год выпуска не корректен»
7	Марка: Mazd1 Год выпуска: 1999	Сообщение: «Название не корректно»
8	Марка: Mazda Год выпуска: 1700	Сообщение: «Год выпуска не корректен»
9	Марка: Mazda Год выпуска: 2001	Сообщение: «Вы будете перемещены на следующую страницу»

В таблице перечислены входные и соответствующие выходные данные.

В ходе тестирование методом черного ящика было выяснено, что форма добавления нового автомобиля работает правильно, а при некорректных данных оповещает об этом пользователя.

6.2 Тестирование метрики кода

Для оценки сложности программного кода обработки авторизации пользователей была использована метрика Холстеда.

Основу метрики Холстеда составляют четыре измеряемых характеристики программы:

- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов);
- число уникальных операндов программы (словарь операндов);
- общее число операторов в программе;
- общее число операндов в программе [17].

Программный код обработки авторизации пользователей представлен в листинге 1.

Листинг 1 – Функция для обработки авторизации пользователей

```
private ResponseEntity<Object> handleAuth(String username, String
password) {
    UserAccount userAccount = authService.findAccount(username);
    if(userAccount == null) {
        return
ResponseEntity.status(404).body("account_not_exists");
    }
    if(!userAccount.getPassword().equals(password)) {
        return ResponseEntity.status(401).body("wrong_password");
    }
    return ResponseEntity.ok(userAccount);
}
```

В качестве входных данных функция принимает имя пользователя и пароль, после чего по первому параметру система ищет объект пользователя.

Далее проверяется, существует ли пользователь. Если не существует – возвращается ошибка с кодом 404, в которой обозначено, что пользователь не найден. Если пользователь существует, то сравнивается полученный пароль с

паролем пользователя. Если пароли совпали, то сервер возвращает объект пользователя, а противном случае – ошибку.

Для использования метрики Холстеда требуется подсчитать количество операторов и операндом. Искомые данные представлены в таблицах 10 и 11 соответственно.

Таблица 10 – Операторы

Операторы	Число вхождений
=	1
!	1
.	8
if	2
==	1
return	3
;	4
,	1
{	3
}	3
findAccount	1
status	2
getPassword	1
equals	1
ok	1
body	2
n1 = 16	N1 = 33

Таблица 11 – Операнды

Операнды	Число вхождений
username	2
password	2
userAccount	4
“account_not_exists”	1
“wrong_password”	1
401	2
n2 = 6	N2 = 12

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

Длина функции рассчитывается по формуле:

$$N = N1 + N2 = 33 + 12 = 45, \quad (2)$$

где N1 – число вхождений операторов;

N2 – число вхождений операндов.

Словарь функции рассчитывается по формуле:

$$n = n1 + n2 = 16 + 6 = 22, \quad (3)$$

где n1 – количество операторов;

n2 – количество операндов.

Объем функции рассчитывается по формуле:

$$HV = N * \log_2(n) = 45 * 4.4594 = 200.7, \quad (4)$$

где N – длина функции;

n – словарь функции.

Потенциальный объём функции:

$$V^* = (2 + n`) * \log_2(2 + n`) = 4 * \log_2 4 = 8, \quad (5)$$

где n` – количество уникальных операндов.

Теоретический уровень программы рассчитывается по формуле:

$$L` = \frac{V^*}{HV} = \frac{8}{200.7} = 0.03986, \quad (6)$$

где V* – потенциальный объем функции;

HV – объем функции.

Сложность функции рассчитывается по формуле:

$$HD = \frac{n1}{2} * \frac{N2}{n2} = \frac{16}{2} * \frac{12}{6} = 16, \quad (7)$$

где n1 – количество операторов;

N2 – число вхождений операндов;

n2 – количество операндов.

Исходя из расчетов, было определено, что сложность функции handleAuth равна 16, что является показателем простой функции.

6.3 Автоматизирование тестирование репозитория для хранения пользователей

Автоматизированное тестирование или автоматизация тестирования – это метод тестирования программного обеспечения, который выполняется с использованием специальных программных средств, которые, в свою очередь необходимы для выполнения набора тестовых примеров [18].

В качестве примера, был взят репозиторий для хранения пользователей, код которого представлен в листинге 2.

Листинг 2 – Структура репозитория и хранимой сущности

```
package ru.oxless.carbackend.model;

import lombok.Data;
import org.springframework.data.mongodb.core.mapping.Document;
import ru.oxless.carbackend.util.PersonalData;
import ru.oxless.carbackend.util.RegisterData;

@Data
@Document(collection = "cars")
public class User {

    private final RegisterData registerData;
    private PersonalData personalData;

}
package ru.oxless.carbackend.repository;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import ru.oxless.carbackend.model.User;

@Repository
public interface UserRepository extends MongoRepository<User,
String> {

}
```

Автоматизированные модульные тесты представлены в листинге 3.

Листинг 3 – Автоматизированные тесты репозитория

```
package ru.oxless.carbackend;

import ru.oxless.carbackend.model.User;
```

```
import ru.oxless.carbackend.repository.UserRepository;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.Test;
import org.springframework.boot.test.context.SpringBootTest;
import static org.junit.jupiter.api.Assertions.*;

@SpringBootTest
class CarBackendApplicationTests {

        @BeforeAll
    static void loadRepository() {
        userRepository.save(new   User(new   RegisterData("Oxless",
"test", "", "")));
        userRepository.save(new    User(new    RegisterData("admin",
"admin", "", "")));
        userRepository.save(new    User(new    RegisterData("test",
"test", "", "")));
    }

    @Test
    void findUserTest() {
        assertNotNull(userRepository.findUserByName("oxless"));
    }

    @Test
    void createUserTest() {
        User  userEntity  =  new  User(new  RegisterData("kipolive",
"test", "", ""));
        userRepository.save(userEntity);
        assertNotNull(userRepository.findUserByName("kipolive"));
    }

    @Test
    void deleteUserTest() {
        userRepository.deleteUser("test");
        assertNull(userRepository.findUserByName("test"));
    }

    @Test
    void updateUserTest() {
        User admin = userRepository.findUserByName("admin");
        admin.setPassword("test");
        userRepository.save(admin);
        User updatedAdmin = userRepository.findUserByName("admin");
        assertEquals("test", updatedAdmin.getPassword());
    }
}
```

Функция loadRepository использует для инициализации тестовых данных. В теле функции создается репозиторий, который содержит три разных пользователя.

Тест findUserTest демонстрирует работу получение пользователя по его имени. После получения пользователя, результат сравнивается с ожидаемым. Ожидаемым результатом будет существование пользователя в репозитории.

Тест createUserTest позволяет протестировать создание нового пользователя. После создания пользователя, приложение находит его по имени и проверяет на существование. Ожидаемым результатом будет существование пользователя в репозитории, так как он был только что добавлен.

Тест deleteUserTest демонстрирует удаление существующего пользователя.

Тест updateUserTest проверяет работоспособность обновления данных пользователя. В текущем примере обновляется пароль пользователя, после приложение получает пользователя по этому имени из репозитория и проверяет, совпадают ли входные данные с теми, чтобы были обновлены.

Результаты тестирования приведены на рисунке 13.

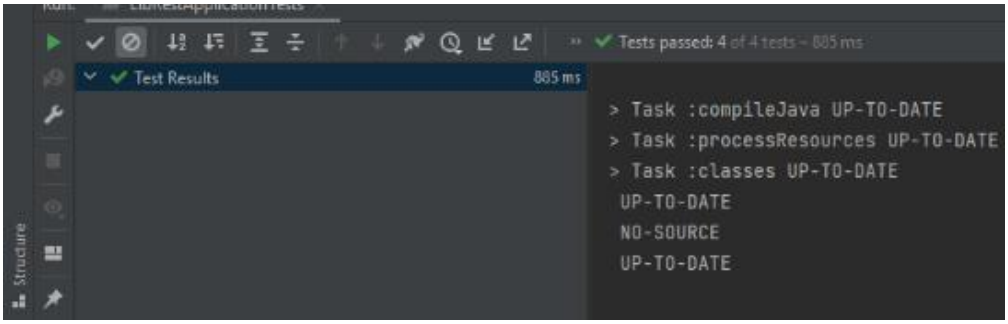

Рисунок 13 – Результаты автоматизированного тестирования

Автоматизированное тестирование позволило проверить ожидаемые данные с полученными путем написания программного кода.

Заключение

В результате выполнения выпускной квалификационной работы было разработано клиент-серверное приложение для подбора поддержанных автомобилей с учетом предпочтений пользователя.

На этапе анализа требований и предметной области были подобраны и проанализированы продукты-аналоги. В результате анализа была проведена сравнительная характеристика, в ходе которой были выявлены основные преимущества и недостатки продуктов-аналогов, на основе которых были определены требования, которым должна соответствовать разрабатываемая система.

Был осуществлен функционально-стоимостной анализ, в ходе которого был определен бюджет и затраты разработки. В результате анализа было определено, что всего разработка приложения займет 41 день, а затраты на реализацию системы составят 238 000 рублей.

Серверная часть приложения разработана с помощью языка программирования Java. Клиентская часть была разработана с помощью языка программирования JavaScript и фреймворка React.

Для разработанного приложения было осуществлено функциональное и модульное тестирование, в ходе которых ошибок работы приложения обнаружено не было. Для исходного кода системы были рассчитаны метрики, чтобы оценить его качество.

Таким образом, в результате выполнения выпускной квалификационной работы было успешно разработано клиент-серверное приложение для подбора поддержанных автомобилей с учетом предпочтений пользователя.

Список используемых источников

1. О сайте Дром // Дром.ру [Электронный ресурс]. URL: https://www.drom.ru/about/ (дата обращения: 23.06.2023).
2. Auto.ru // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Auto.ru (дата обращения: 23.06.2023).
3. Авито // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Авито (дата обращения: 23.06.2023).
4. Методика функционально-стоимостного анализа // НОУ ИНТУИТ [Электронный ресурс]. URL: https://intuit.ru/studies/courses/3636/878/lecture/32286 (дата обращения: 23.06.2023).
5. Диаграмма Ганта // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_Ганта (дата обращения: 23.06.2023).
6. UML // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/UML (дата обращения: 23.06.2023).
7. Введение в Java // Ravesli [Электронный ресурс]. URL: https://ravesli.com/introduction-to-java/ (дата обращения: 23.06.2023).
8. Что такое Spring Framework // Skypro [Электронный ресурс]. URL: https://sky.pro/media/chto-takoe-spring-framework/ (дата обращения: 23.06.2023).
9. Введение в JavaScript // Ravesli [Электронный ресурс]. URL: https://ravesli.com/intro-javascript/ (дата обращения: 23.06.2023).
10. React в Windows // Microsoft Learn [Электронный ресурс]. URL: https://learn.microsoft.com/ru-ru/windows/dev-environment/javascript/react-overview (дата обращения: 23.06.2023).
11. HTML // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/HTML (дата обращения: 23.06.2023).
12. CSS // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/CSS (дата обращения: 23.06.2023).

13. Р. Мамедли. Системы управления базами данных. Учебное пособие. – 2021. – С. 7.
14. Диаграмма прецедентов // Википедия [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/Диаграмма_прецедентов (дата обращения: 06.04.2023).
15. С. Куликов. Тестирование программного обеспечения. Базовый курс. – 2023. – С. 6.
16. Тестирование методом черного ящика // Timofeev.ru [Электронный ресурс] URL: https://temofeev.ru/info/articles/testirovanie-metodom-chernogo-yashchika/ (дата обращения: 06.04.2023).
17. Общие сведения о программном обеспечении // НОУ ИНТУИТ [Электронный ресурс] URL: https://intuit.ru/studies/professional_skill_improvements/17031/courses/874/lecture/14289?page=2 (дата обращения: 06.04.2023).
18. Автоматизированное тестирование: что это? // LOGROCON [Электронный ресурс] URL: https://logrocon.ru/news/automation_testing (дата обращения: 06.04.2023).

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД СЕРВЕРНОЙ ЧАСТИ
(рекомендуемое)

Листинг А.1 – Файл CarBackendApplication.java

```
package ru.oxless.carbackend;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CarBackendApplication {

    public static void main(String[] args) {
        SpringApplication.run(CarBackendApplication.class, args);
    }

}
```

Листинг А.2 – Файл Car.java

```
package ru.oxless.carbackend.model;

import lombok.Data;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;

import java.util.Map;

@Data
@Document(collection = "cars")
public class Car {

    @Id
    private final String id;
    private final String ownerId;
    private boolean published;
    private final String mark;
    private final String model;
    private final int year;
    private int price;
    private int mileage;
    private final Map<String, Object> carAttributes;
    private final Map<String, Object> personalAttributes;

}
```

Листинг А.3 – Файл User.java

```
package ru.oxless.carbackend.model;

import lombok.Data;
```

```
import org.springframework.data.mongodb.core.mapping.Document;
import ru.oxless.carbackend.util.PersonalData;
import ru.oxless.carbackend.util.RegisterData;

@Data
@Document(collection = "cars")
public class User {

    private final RegisterData registerData;
    private PersonalData personalData;

}
```

Листинг А.4 – Файл CarRepository.java

```
package ru.oxless.carbackend.repository;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import ru.oxless.carbackend.model.Car;

@Repository
public interface CarRepository extends MongoRepository<Car, String>
{

}
```

Листинг А.5 – Файл UserRepository.java

```
package ru.oxless.carbackend.repository;

import org.springframework.data.mongodb.repository.MongoRepository;
import org.springframework.stereotype.Repository;
import ru.oxless.carbackend.model.User;

@Repository
public interface UserRepository extends MongoRepository<User,
String> {

}
```

Листинг А.6 – Файл CarService.java

```
package ru.oxless.carbackend.service;

import org.springframework.stereotype.Service;
import ru.oxless.carbackend.model.Car;
import ru.oxless.carbackend.repository.CarRepository;

import java.util.List;
```

```
@Service
public class CarService {

    private final CarRepository carRepository;

    public CarService(CarRepository carRepository) {
        this.carRepository = carRepository;
    }

    public List<Car> findAll() {
        return carRepository.findAll();
    }

    public Car findOne(String carId) {
        return carRepository.findById(carId).orElse(null);
    }

    public Car add(Car car) {
        return carRepository.save(car);
    }

    public Car update(Car car) {
        return add(car);
    }

    public Car delete(Car car) {
        carRepository.deleteById(car.getId());
        return car;
    }

}
```

Листинг А.7 – Файл UserService.java

```
package ru.oxless.carbackend.service;

import org.springframework.stereotype.Service;
import ru.oxless.carbackend.model.User;
import ru.oxless.carbackend.repository.UserRepository;
import ru.oxless.carbackend.util.RegisterData;

import java.util.List;

@Service
public class UserService {

    private final UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }
```

```
    public User createUser(RegisterData registerData) {
        User user = new User(registerData);
        return userRepository.save(user);
    }

    public void updateUser(User user) {
        userRepository.save(user);
    }

    public void deleteUser(String userId) {
        userRepository.deleteById(userId);
    }

    public User findOne(String userId) {
        return userRepository.findById(userId).orElse(null);
    }

    public List<User> findAll() {
        return userRepository.findAll();
    }

}
```

Листинг А.8 – Файл MongoConfig.java

```
package ru.oxless.carbackend.config;

import com.mongodb.client.MongoClient;
import com.mongodb.client.MongoClients;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.data.mongodb.core.MongoTemplate;

@Configuration
public class MongoConfig {

    @Bean
    public MongoClient mongoClient() {
        return MongoClients.create();
    }

    @Bean
    public MongoTemplate mongoTemplate() {
        return new MongoTemplate(mongoClient(), "local");
    }

}
```

Листинг А.9 – Файл LoginData.java

```
package ru.oxless.carbackend.util;

import lombok.AllArgsConstructor;

@AllArgsConstructor
public class LoginData {

    public String username;
    public String password;

}
```

Листинг А.10 – Файл PersonalData.java

```
package ru.oxless.carbackend.util;

import lombok.Data;

import java.util.Map;

@Data
public class PersonalData {

    private final Map<String, Object> attributes;

}
```

Листинг А.11 – Файл RegisterData.java

```
package ru.oxless.carbackend.util;

import lombok.AllArgsConstructor;

@AllArgsConstructor
public class RegisterData {

    public String username;
    public String password;
    public String fullName;
    public String phone;

}
```

Листинг А.12 – Файл CarController.java

```
package ru.oxless.carbackend.controller;

import jakarta.servlet.http.HttpSession;
```

```
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import ru.oxless.carbackend.model.Car;
import ru.oxless.carbackend.model.User;
import ru.oxless.carbackend.service.CarService;
import ru.oxless.carbackend.service.UserService;
import ru.oxless.carbackend.util.PersonalData;

import java.util.stream.Collectors;

@RestController
public class CarController {

    private final CarService carService;
    private final UserService userService;

    public CarController(CarService carService, UserService
userService) {
        this.carService = carService;
        this.userService = userService;
    }

    @GetMapping("/cars")
    public ResponseEntity getAll() {
        return ResponseEntity.ok(carService.findAll());
    }

    @GetMapping("/cars/{id}")
    public ResponseEntity getOne(@PathVariable String id) {
        return ResponseEntity.ok(carService.findOne(id));
    }

    @GetMapping("/cars/personal")
    public ResponseEntity getPersonalCars(HttpSession httpSession)
{
        User user = userService.findOne((String)
httpSession.getAttribute("id"));
        return ResponseEntity.ok(carService.findAll()
                .stream().filter(car ->
car.getPersonalAttributes().equals(user.getPersonalData().getAttri
butes()))
                .collect(Collectors.toList())
        );
    }

    @PostMapping("/cars/add")
    public ResponseEntity addCar(HttpSession httpSession, Car car)
{
        if(httpSession.getAttribute("id") != null) {
            return ResponseEntity.ofNullable(carService.add(car));
        }
        return ResponseEntity.status(403).build();
    }
```

```
    @PostMapping("/cars/{id}")
    public ResponseEntity updateCar(HttpSession httpSession, Car
car) {
        if(httpSession.getAttribute("id") != null) {
            return ResponseEntity.ofNullable(carService.add(car));
        }
        return ResponseEntity.status(403).build();
    }

    @DeleteMapping("/cars/{id}")
    public ResponseEntity deleteCar(HttpSession httpSession,
@PathVariable String id) {
        if(httpSession.getAttribute("id") != null) {
            return
ResponseEntity.ofNullable(carService.delete(carService.findOne(id)
));
        }
        return ResponseEntity.status(403).build();
    }

    @PostMapping("/personal")
    public ResponseEntity updatePersonal(HttpSession httpSession,
@RequestBody PersonalData personalData) {
        if(httpSession.getAttribute("id") != null) {
            User user = userService.findOne((String)
httpSession.getAttribute("id"));
            user.setPersonalData(personalData);
            userService.updateUser(user);
            return ResponseEntity.ok(user);
        }
        return ResponseEntity.status(403).build();
    }

}
```

Листинг А.13 – Файл UserController.java

```
package ru.oxless.carbackend.controller;

import jakarta.servlet.http.HttpSession;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
import ru.oxless.carbackend.model.User;
import ru.oxless.carbackend.service.UserService;
import ru.oxless.carbackend.util.LoginData;
import ru.oxless.carbackend.util.RegisterData;

@RestController
```

```
public class UserController {

    private final UserService userService;

    public UserController(UserService userService) {
        this.userService = userService;
    }

    @PostMapping("/account/register")
    public     ResponseEntity<Object>     registerUser(HttpSession
httpSession, @RequestBody RegisterData registerData) {
        if(httpSession != null)
            return ResponseEntity.status(400).build();
        return
ResponseEntity.ok(userService.createUser(registerData));
    }

    @PostMapping("/account/login")
    public       ResponseEntity<Object>       loginUser(HttpSession
httpSession, @RequestBody LoginData loginData) {
        if(httpSession != null)
            return ResponseEntity.status(400).build();
        User foundUser = userService.findAll().stream()
                .filter(user                                        ->
user.getRegisterData().username.equals(loginData.username))
                .findFirst().orElse(null);
        if(foundUser == null)
            return ResponseEntity.status(404).build();

if(!foundUser.getRegisterData().password.equals(loginData.password
)) {
            return ResponseEntity.ok(foundUser);
        }
        return ResponseEntity.status(400).build();
    }

    @PostMapping("/account/logout")
    public       ResponseEntity<Object>       logoutUser(HttpSession
httpSession) {
        if(httpSession == null)
            return ResponseEntity.status(401).build();
        httpSession.invalidate();
        return ResponseEntity.ok("logout");
    }

    @GetMapping("/account/validate")
    public     ResponseEntity<Object>     validateUser(HttpSession
httpSession) {
        if(httpSession == null)
            return ResponseEntity.status(401).build();
        User       user       =       userService.findOne((String)
httpSession.getAttribute("id"));
        return ResponseEntity.ok(user); }}
```

ПРИЛОЖЕНИЕ Б.
ИСХОДНЫЙ КОД КЛИЕНТСКОЙ ЧАСТИ
(рекомендуемое)

Листинг Б.1 – Файл App.js

```
import './App.css';
import {BrowserRouter, Route, Routes} from "react-router-dom";
import HomePage from "./page/HomePage";
import CarMarkPage from "./page/CarMarkPage";
import Profile from "./page/Profile";
import ViewCarPage from "./page/ViewCarPage";

function App() {
  return (
    <BrowserRouter>
      <Routes>
        <Route path="/login" element={<LoginPage/>}/>
        <Route path="/cars" element={<HomePage/>}/>
        <Route path="/" element={<HomePage/>}/>
        <Route path="/cars/:model" element={<CarMarkPage/>}/>
        <Route path="/profile" element={<Profile/>}/>
        <Route path="/car" element={<ViewCarPage/>}/>
        <Route path="/help" element={<HelpPage/>}/>
      </Routes>
    </BrowserRouter>
  );
}

export default App;
```

Листинг Б.2 – Файл Questionnaire.js

```
import React from 'react';
import Header from "../component/Header";
import "./Questionnaire.css"

const Questionnaire = (props) => {

    return (
        <div className="questionnaire">
            <Header/>
            <div>
                <h2>Персональная анкета пользователя</h2>
                <p>Укажите ваш возраст:</p>
                <input/>
                <p>Укажите ваш пол:</p>
                <input type="checkbox" value="Мужской"/>
                <input type="checkbox" value="Женский"/>
                <p>Есть ли у вас дети?</p>
                <input type="checkbox" value="Нет"/>
                <input type="checkbox" value="От 1 до 3"/>
                <input type="checkbox" value="Более 3"/>
                <p>Местность проживания:</p>
```

```
                <input type="checkbox" value="Городская"/>
                <input type="checkbox" value="Сельская"/>
                <input type="checkbox" value="Горная"/>
                <p>Что для вас важнее?</p>
                <input type="checkbox" value="Скорость"/>
                <input type="checkbox" value="Надежность"/>
                <p>Ваш рост:</p>
                <input type="checkbox" value="До 175 см"/>
                <input type="checkbox" value="Более 175 см"/>
                <button>Сохранить ответы</button>
            </div>
        </div>
    );
}

export default Questionnaire;
```

Листинг Б.3 – Файл CarMarkPage.js

```
import React, {useEffect, useState} from 'react';
import Header from "../component/Header";
import axios from "axios";
import {Link} from "react-router-dom";
import CarCard from "../component/CarCard";

const CarMarkPage = (props) => {

    const [cars, setCars] = useState([]);

    useEffect(() => {
        axios.get("http://localhost:8080/cars")
            .then(res => setCars(res.data))
    }, [])

    return (
        <div className="mark-page">
            <Header/>
            <div>
                {cars.map(car      =>      <Link      to={"/"      +
car.model}>{car.model}</Link>)}
            </div>
            <div>
                <Filters/>
                {cars.map(car => <CarCard car={car}/>)}
            </div>
        </div>
    );
}

export default CarMarkPage;
```

Листинг Б.4 – Файл CarCard.js

```
import React from 'react';
import "./CarCard.css"

const CarCard = (props) => {

    const car = props.car;

    return (
        <div>
            <img src={car.src} alt=""/>
            <h2>{car.price}</h2>
            <h3>{car.name}</h3>
            <h3>{car.year + "/" + car.mileage}</h3>
        </div>
    );
}

export default CarCard;
```

Листинг Б.5 – Файл Header.js

```
import React, {useEffect, useState} from 'react';
import Logo from "./Logo";
import {Link} from "react-router-dom";
import axios from "axios";
import "./Header.css"

const Header = (props) => {

    const [user, setUser] = useState();

    useEffect(() => {
        axios.get("http://localhost:8080/account/validate")
            .then(res => setUser(res.data))
    }, [])

    return (
        <div className="header">
            <Logo/>
            <Link to="/cars">Каталог</Link>
            <Link to="/help">Помощь</Link>
            <div className="left">
                <Link to="/create">+ Разместить объявление</Link>
                <Link to={user ? "/profile" : "/login"}>{user ?
user.username : "Войти"}</Link>
            </div>
        </div>
    );
}
```

```
export default Header;
```

Листинг Б.6 – Файл ViewCarPage.js

```
import React from 'react';
import Header from "../component/Header";
import "./ViewCarPage"

const ViewCarPage = (props) => {

    const car = props.car;

    return (
        <div className="car-view">
            <Header/>
            <div>
                <h2>{car.name}</h2>
                <h3>{car.price}</h3>
                <p>{car.description}</p>
                <p>{car.mileage}</p>
                <p>{car.mark}</p>
                <p>{car.model}</p>
                <p>{car.year}</p>
            </div>
        </div>
    );
}

export default ViewCarPage;
```

Листинг Б.7 – Файл HomePage.js

```
import React, {useEffect, useState} from 'react';
import Header from "../component/Header";
import {Link} from "react-router-dom";
import CarCard from "../component/CarCard";
import "./HomePage.css"

const HomePage = (props) => {

    const [cars, setCars] = useState([]);
    const [personal, setPersonal] = useState([]);
    const [user, setUser] = useState(null);

    useEffect(() => {
        axios.get("http://localhost:8080/account/validate")
            .then(res => setUser(res.data))
        axios.get("http://localhost:8080/cars")
            .then(res => setCars(res.data))
        axios.get("http://localhost:8080/cars/personal")
```

```
            .then(res => setPersonal(res.data))
    })

    const renderNotAuth = () => {
        return (
            <div>
                <h2>Вы не авторизованы</h2>
                <Link to="/login">Войти</Link>
            </div>
        )
    }

    const renderNotInfo = () => {
        return (
            <div>
                <h2>У нас мало данных о ваших предпочтениях</h2>
                <Link to="/questionnaire">Заполнить анкету</Link>
            </div>
        )
    }

    const renderPersonal = () => {
        return (
            <div>
                {personal.map(car => <CarCard car={car}/>)}
            </div>
        )
    }

    return (
        <div>
            <Header/>
            <div className="marks">
                <h1>Легковые автомобили</h1>
                <h2>{cars.length} предложений</h2>
                {cars.map(car      =>      <Link      to={"/"      +
car.mark}>{car.mark}</Link>)}
            </div>
            <div>
                <h1>Персональная подборка</h1>
                {user  ?  user.hasPersonal  ?  renderPersonal()  :
renderNotInfo() : renderNotAuth()}
            </div>
        </div>
    );
}

export default HomePage;
```

ПРИЛОЖЕНИЕ В.
РЕЗУЛЬТАТЫ ТЕСТИРОВАНИЯ
(рекомендуемое)

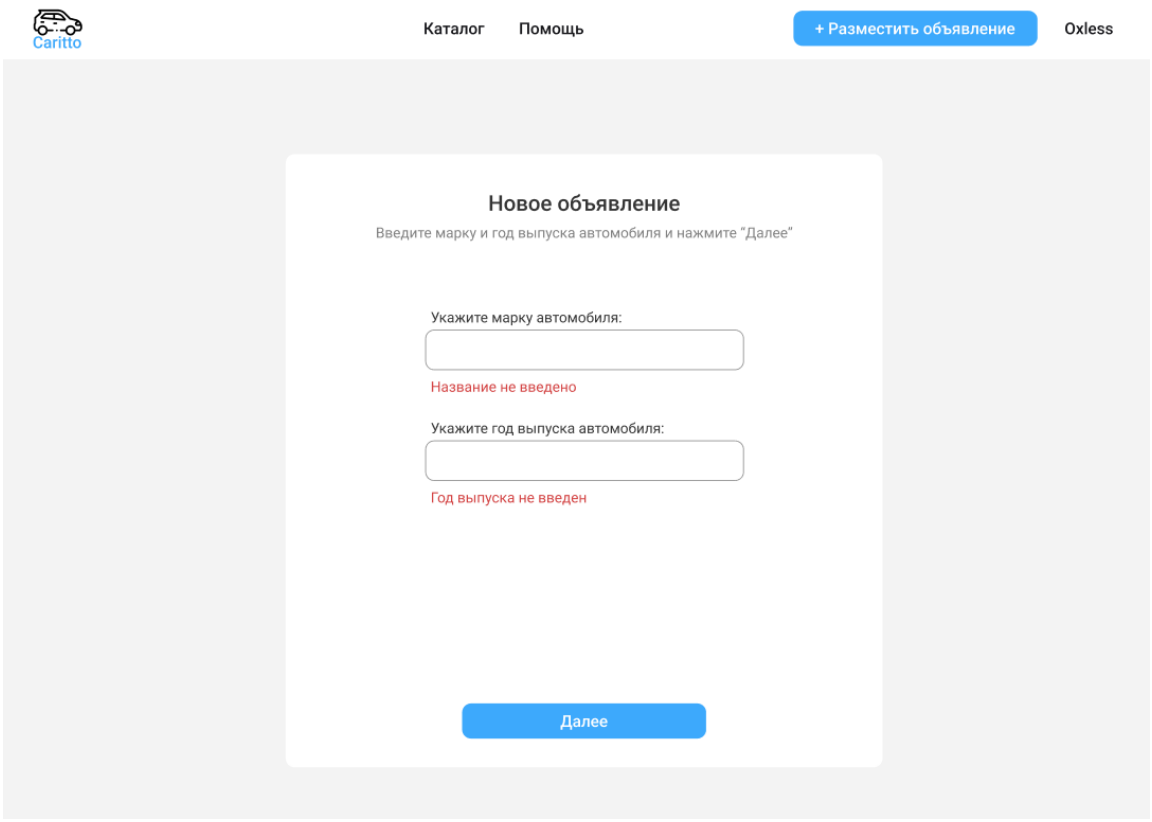

Рисунок В.1 – Результат теста №1

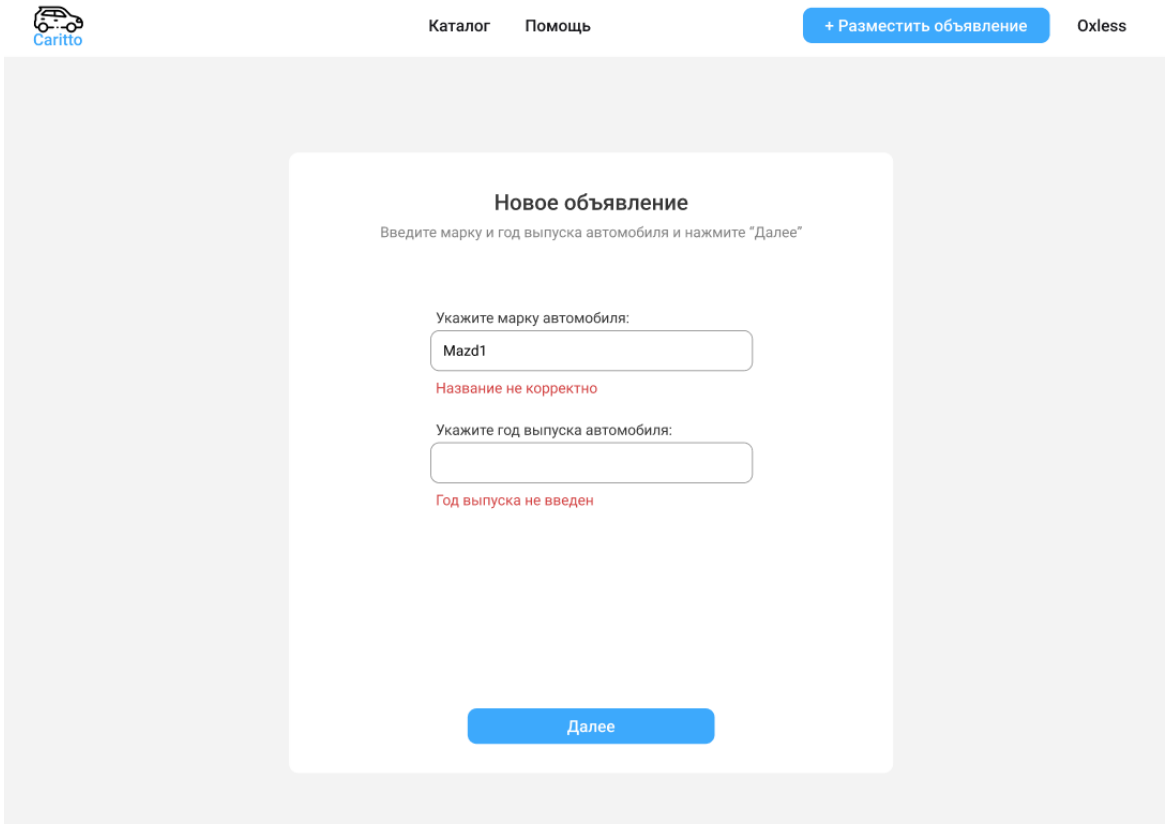

Рисунок В.2 – Результат теста №2

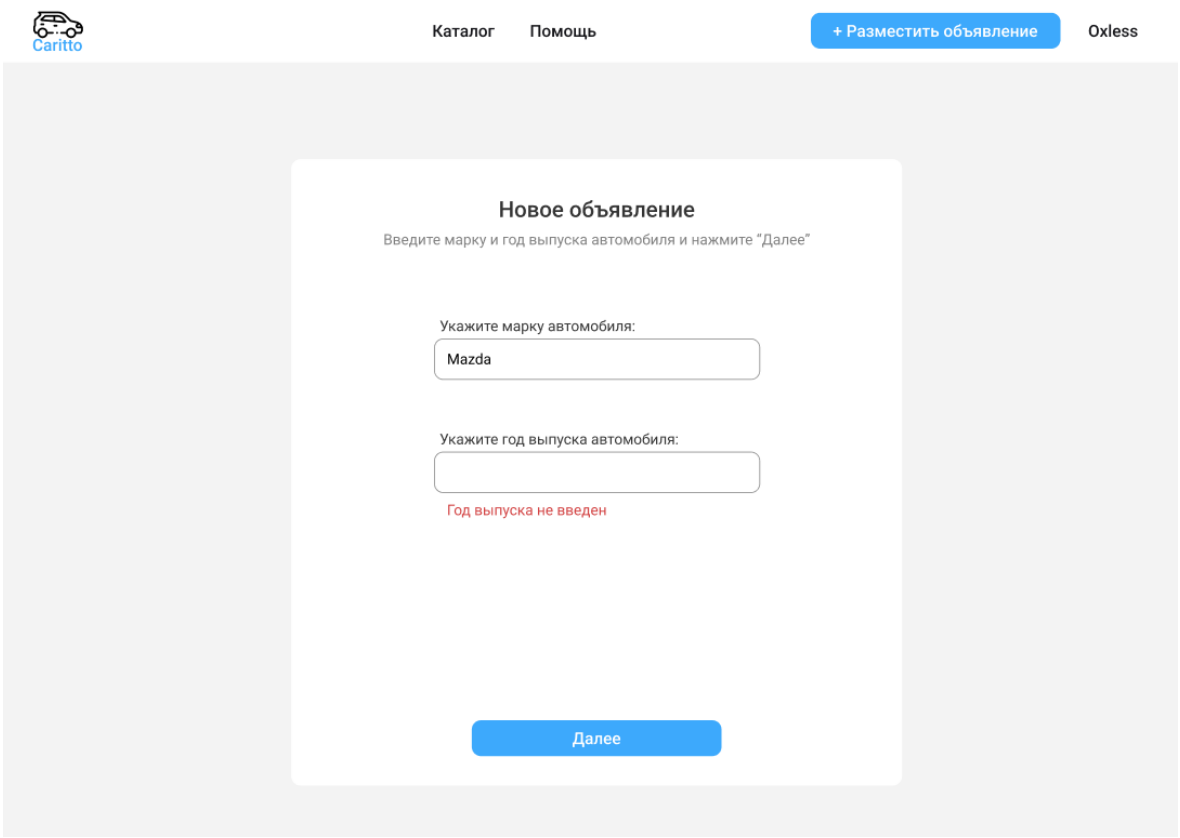

Рисунок В.3 – Результат тест №3

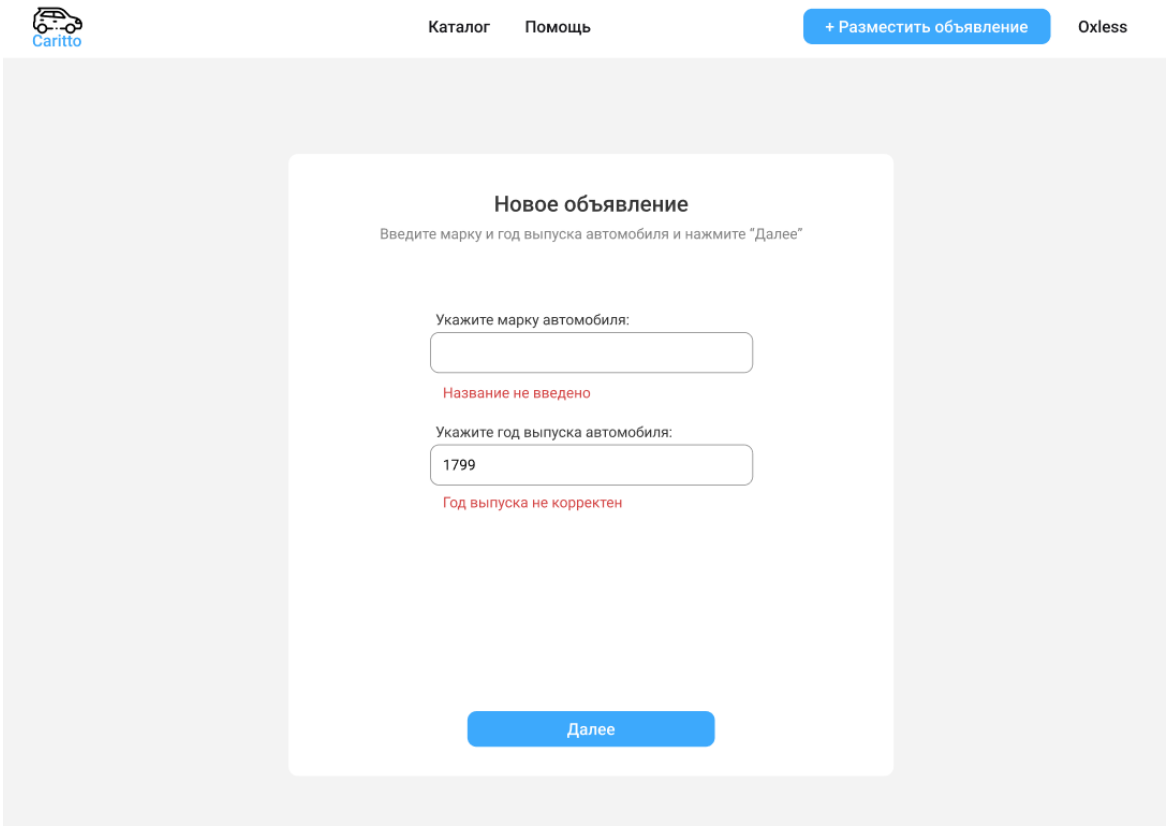

Рисунок В.4 – Результат теста №4

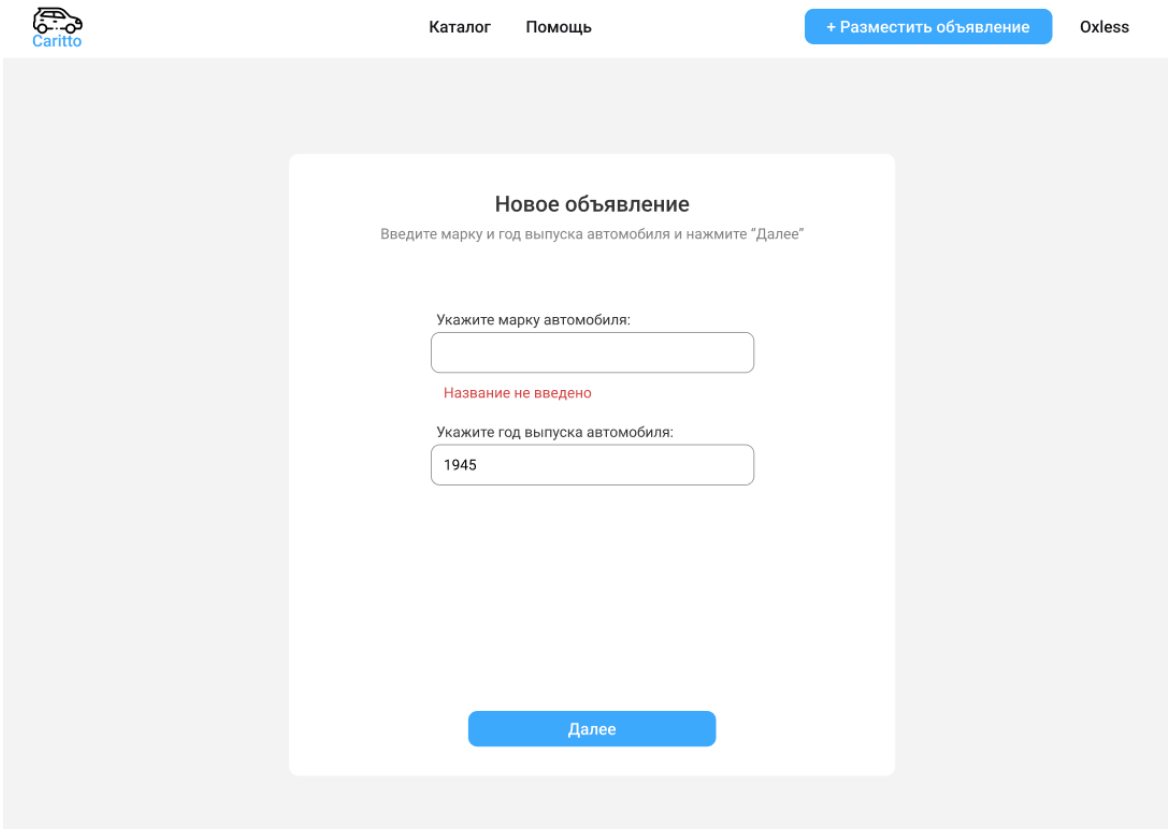

Рисунок В.5 – Результат теста №5

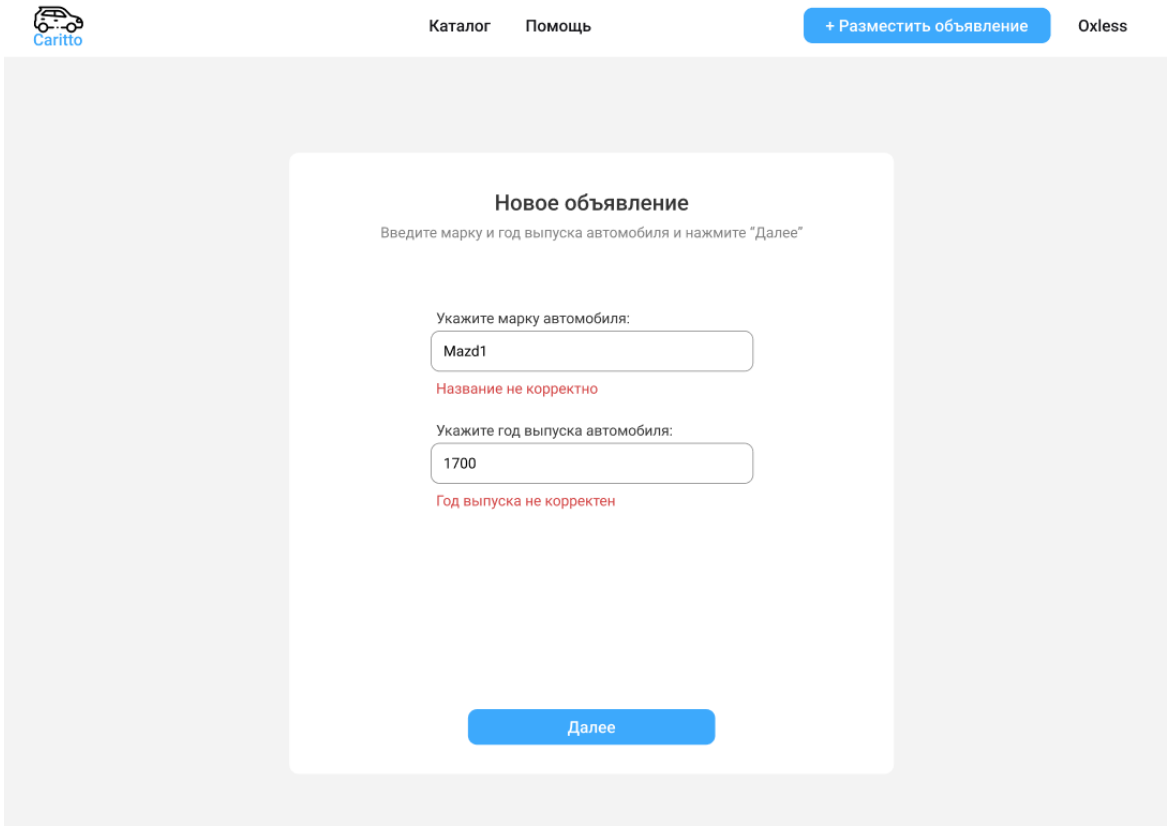

Рисунок В.6 – Результат теста №6

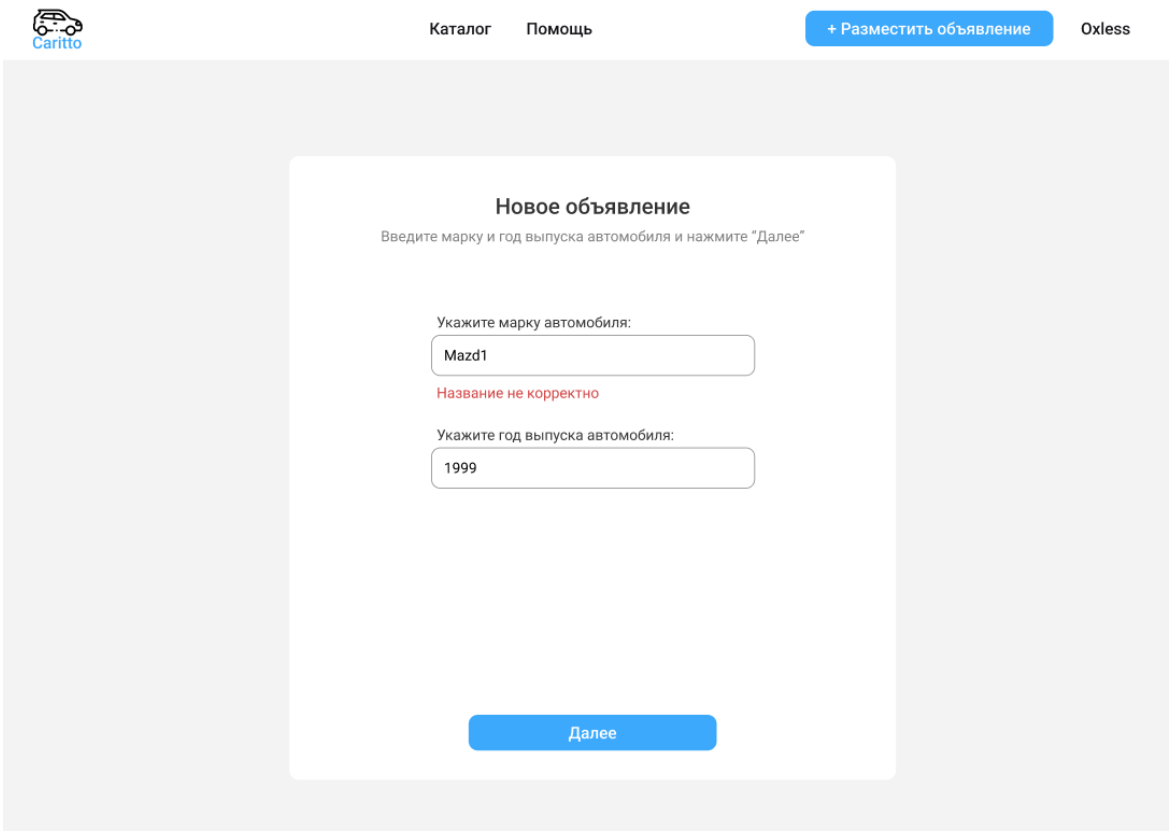

Рисунок В.7 – Результат теста №7

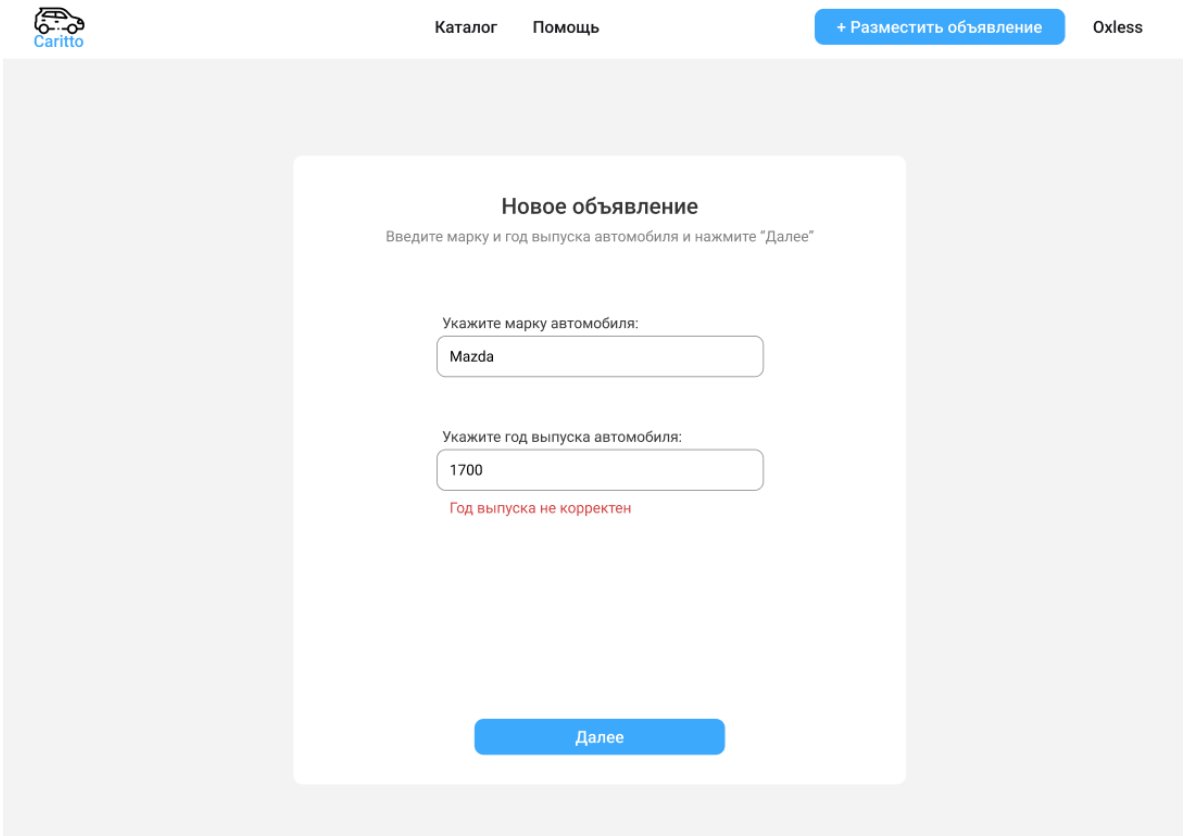

Рисунок В.8 – Результат теста №8

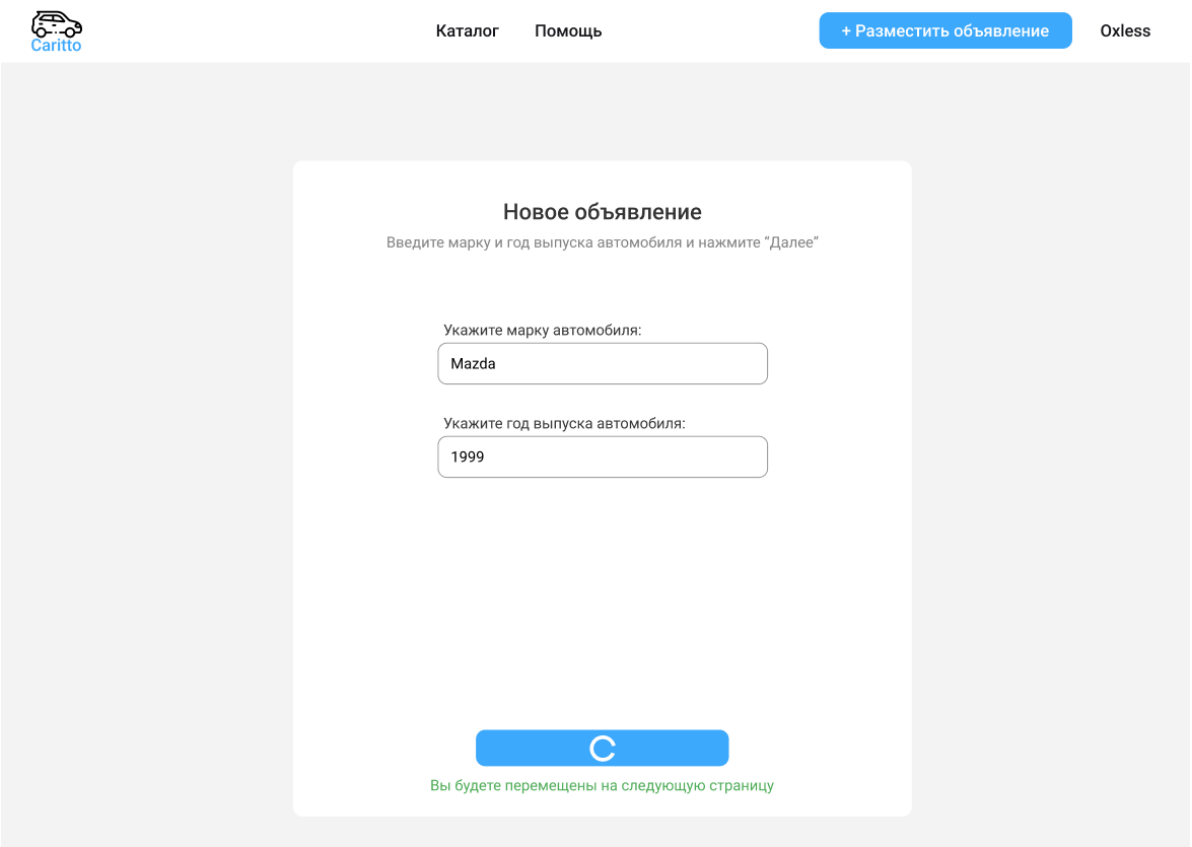

Рисунок В.9 – Результат теста №9