*Федеральное государственное бюджетное образовательное учреждение высшего образования*

*«Владимирский государственный университет*

*имени Александра Григорьевича и Николая Григорьевича Столетовых»*

*Кафедра информационных систем и программной инженерии*

***ПОЯСНИТЕЛЬНАЯ ЗАПИСКА***

***к курсовому проекту по дисциплине   
"Распределённые программные системы"***

***на тему***

*Проектирование и разработка*

*Прототипа программной системы «Прокат автомобилей»*

*Выполнил: студент гр. ИСТ-120*

*Титов О.Э.*

*Принял: ст. преп. ИСПИ*

*Тимофеев А.А.*

*Владимир, 2023*

**Аннотация**

Курсовой проект посвящен разработке прототипа программной системы "Прокат автомобилей" на платформе Java с использованием Spring Boot Framework. Целью проекта является создание эффективной и удобной системы для управления процессом аренды автомобилей. В рамках работы были проведены анализ требований, проектирование архитектуры системы, разработка функциональных модулей, интеграция с базой данных MySQL. Проект включает в себя реализацию основных функций системы, таких как создание и управление контрактами, управление клиентской базой данных, а также административные функции для управления автомобилями и пользователями. В результате выполнения курсового проекта был создан прототип программной системы "Прокат автомобилей", который может быть дальнейшем использован в коммерческих целях. Весь код проекта представлен в GIT репозитории по ссылке: <https://github.com/sixxG/CarRental-.git> .

The term project is dedicated to the development of a prototype of a software system called "Car Rental" using the Java platform and the Spring Boot Framework. The goal of the project is to create an efficient and user-friendly system for managing the car rental process. Within the scope of the work, requirements analysis, system architecture design, development of functional modules, and integration with the MySQL database were carried out.

The project includes the implementation of key system functions such as contract creation and management, customer database management, as well as administrative functions for managing cars and users. As a result of completing the term project, a prototype of the "Car Rental" software system was created, which can be further used for commercial purposes. The entire project code is available in the GIT repository at the following link: <https://github.com/sixxG/CarRental-.git>.

Курсовой проект представлен на 95 страницах, рисунков – 23, таблиц – 6, использованных источников – 5, приложений – 3.

Оглавление

[1 ВВЕДЕНИЕ 5](#_Toc136240751)

[2 Постановка задачи 7](#_Toc136240752)

[2.1 Цели и задачи 7](#_Toc136240753)

[2.2 Описание предметной области 8](#_Toc136240754)

[2.3 Основные понятия предметной области 9](#_Toc136240755)

[2.4 Функциональные требования к системе 11](#_Toc136240756)

[2.5 State of Art 13](#_Toc136240757)

[2.6 Нефункциональные требования к системе 13](#_Toc136240758)

[2.7 Сравнительный анализ аналогов 14](#_Toc136240759)

[3 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ 19](#_Toc136240760)

[3.1 Анализ бизнес-процесса «Прокат автомобилей» 20](#_Toc136240761)

[3.2 Сценарий взаимодействия пользователя с системой 27](#_Toc136240762)

[3.3 Поиск авто 31](#_Toc136240763)

[3.4 Добавление авто в каталог 32](#_Toc136240764)

[3.5 Завершение аренды 33](#_Toc136240765)

[4 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ 35](#_Toc136240766)

[4.1 Организация доступа к данным 37](#_Toc136240767)

[4.2 Организация бизнес-логики 38](#_Toc136240768)

[4.3 Организация веб-интерфейса 38](#_Toc136240769)

[4.4 Взаимодействие компонентов системы для процесса «Найти авто» 44](#_Toc136240770)

[4.5 Взаимодействие компонентов системы для процесса «Отменить аренду» 45](#_Toc136240771)

[5 реализация компонентов системы 47](#_Toc136240772)

[5.1 Компоненты бизнес-логики 47](#_Toc136240773)

[5.1.1 Класс CarService 47](#_Toc136240774)

[5.1.2 Класс UserService 48](#_Toc136240775)

[5.2 Веб-компоненты 50](#_Toc136240776)

[5.2.1 Классы контроллеры 50](#_Toc136240777)

[5.2.2 FTL-страница Main.ftl 51](#_Toc136240778)

[5.2.3 Конфигурирование приложения с использованием Freemarker 55](#_Toc136240779)

[5.3 Организация обработки ошибок 56](#_Toc136240780)

[5.4 Организация управления доступом 59](#_Toc136240781)

[5.5 Физическая структура базы данных 61](#_Toc136240782)

[5.6 Обеспечение целостности данных 62](#_Toc136240783)

[5.6.1 Ограничение целостности 62](#_Toc136240784)

[5.6.2 Ограничение распределённых транзакций 63](#_Toc136240785)

[5.7 Реализация взаимодействия с внешним сервисом 64](#_Toc136240786)

[6 Инструкция по установке приложения 67](#_Toc136240787)

[7 оРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ 69](#_Toc136240790)

[7.1 Распределение ролей в команде/зоны ответственности 69](#_Toc136240791)

[7.2 Календарный план работ 69](#_Toc136240792)

[8 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ 71](#_Toc136240793)

[8.1 Тестирование регистрации и авторизации 71](#_Toc136240794)

[8.2 Тестирование функционала неавторизованного пользователя 73](#_Toc136240795)

[8.3 Тестирование функционала авторизованного пользователя 75](#_Toc136240796)

[8.4 Тестирование функционала менеджера 77](#_Toc136240797)

[8.5 Тестирование функционала администратора 80](#_Toc136240798)

[8.6 Анализ результатов нагрузочного тестирования 82](#_Toc136240799)

[9 заключение 83](#_Toc136240800)

[10 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 84](#_Toc136240801)

[ПРИЛОЖЕНИЕ А Листинг компонентов системы 85](#_Toc136240802)

[ПРИЛОЖЕНИЕ б Скрипт создания базы данных 93](#_Toc136240803)

[ПРИЛОЖЕНИЕ В IDEF0 модели as-is 95](#_Toc136240804)

# 1 ВВЕДЕНИЕ

Несмотря на стремительную информатизацию бизнес-процессов всё ещё остаются компании, выполняющие множество рутинных операций различными средствами, будь то MS Office или простая толстая тетрадь.

В частности, можно это можно сказать и про отрасль проката автомобилей. Процессы в данной сфере деятельности могут занимать много времени, а вся информация хранится в виде файлов и каталогов, что усложняет работу в многопользовательском режиме.

Целью данного курсового проекта является разработка прототипа программной системы "Прокат автомобилей", предназначенной для автоматизации процессов проката автомобилей, обеспечивая удобный интерфейс для клиентов и эффективное управление автомобильным парком для администраторов. Удобный интерфейс системы для клиентов необходим для обеспечения удобства и эффективности процесса аренды автомобилей. Клиенты должны иметь возможность легко найти, выбрать и забронировать автомобиль, а также получить всю необходимую информацию о условиях проката. Под эффективным управлением автомобильным парком понимается оптимизация процессов управления автомобилями, чтобы обеспечить их максимальную загруженность, минимизировать время простоя. Измерение эффективности управления будет основано на таких показателях, как загруженность автомобилей и количество успешно выполненных заказов.

В настоящее время прокат автомобилей является популярной услугой, и все больше компаний стремятся автоматизировать свои бизнес-процессы для повышения эффективности и удовлетворения потребностей клиентов. Ручное управление и отслеживание прокатных операций могут быть трудоемкими и подверженными ошибкам. Поэтому разработка программной системы, которая позволяет автоматизировать основные процессы проката автомобилей, является актуальной и значимой задачей.

Прототип программной системы "Прокат автомобилей" будет разработан с использованием платформы Java, широко применяемой в разработке программного обеспечения, и фреймворка Spring Boot, который облегчает создание веб-приложений и управление зависимостями. Использование Spring Boot позволит создать эффективное и масштабируемое веб-приложение с минимальными усилиями.

**2 Постановка задачи**

## 2.1 Цели и задачи

Основными целями создания ИС «Прокат автомобилей» являются:

1. Повышение эффективности работы компании путём:

* Сокращения операций, выполняемых вручную: Целью является автоматизация рутинных операций, таких как заполнение бумажных форм, расчеты и отслеживание данных. Это позволит снизить вероятность ошибок и ускорить выполнение задач.
* Оптимизации информационного взаимодействия участников процессов: Целью является создание единой централизованной системы, которая обеспечит удобство обмена информацией между сотрудниками и клиентами. Это поможет улучшить координацию действий и сократить время на передачу и обработку данных.
* Сокращение времени, затрачиваемого на выполнение операций: Целью является оптимизация бизнес-процессов и использование автоматизированных систем для ускорения выполнения задач, таких как бронирование автомобилей, расчет стоимости услуг и формирование отчетов.

1. Повышение качества принятия управленческих решений за счёт:

* Полноты, удобства и оперативности предоставления информации: Целью является создание системы, которая предоставит полную и актуальную информацию о состоянии автомобильного парка, бронированиях, финансовых показателях и других важных данных. Это поможет улучшить прозрачность и оперативность принятия управленческих решений.

Для реализации поставленных целей система должна решать следующие задачи:

• Ввод данных;

• Редактирование данных;

• Хранение наиболее полной информации об автомобилях, их состоянии и обо всех связанных с ними документах, а также о тарифах для всех моделей автомобилей.

• Составление договора проката;

• Изменение и завершение договора проката;

• Возможность удалённой работы с программой через интернет.

## 2.2 Описание предметной области

В данной работе рассматривается предметная область «Прокат автомобилей». Этим вопросом обычно занимается один человек – менеджер, менеджер должен знать кто взял на прокат авто, какой марки и на какой срок.

Специфика работы пункта проката авто состоит в следующем: когда в пункт приходят новые автомобили (процесс покупки авто в данной ИС не рассматривается) требуется занести в базу данные о каждом автомобиле (марка, модель, цвет автомобиля, год выпуска, в соответствии с экономическими требованиями, необходимо назначить цену за один день аренды).

Клиенты могут арендовать автомобиль для любых целей (например поездка с семьёй на природу ил встреча с друзьями) сроком от 1 дня на территории Владимирской обл. с неограниченным пробегом, для этого им необходимо будет иметь с собой паспорт и водительское удостоверение. Также клиент не должен быть младше 21 года и старше 65, и иметь стаж вождения от 2-х лет.

После выбора клиентом авто нужно составить договор аренды, в котором должны быть указаны данные о клиенте (фамилия, имя, адрес проживания, дата рождения, номер ВУ) и арендуемом авто (код авто, присвоенный ему при взятии на учет), а также срок проката. Клиент может выбрать дополнительные опции к выбранному авто (детское кресло, автобокс и т.д.) плата за них добавляется к основной цене аренды. Также клиент может выбрать способ получения авто: из офиса проката (клиенту необходимо будет явиться в офис компания для оплаты аренды и получения автомобиля) или в любом населённом пункте Владимирской области (клиенту необходимо будет заплатить за аренду и доставку автомобиля, которую будет осуществлять курьер). Аналогичными способами клиент может вернуть автомобиль. При возврате авто его осматривает специалист, каких-либо нарушений клиент будет обязан выплатить штраф.

После это необходимо занести эти данные в базу данных, убрать выбранный автомобиль из списка доступных авто, сохранить файл отчета (при необходимости распечатать его) и выписать клиенту чек оплаты.

Все автомобили предоставляются клиентам с полным баком и клиент обязан вернуть авто с таким же количеством топлива. Также все автомобили в компании имеют страховой полис Каско, но при ДТП клиенту могут назначить штраф исходя из привил компании.

Проанализировав, таким образом, предметную область, можно предложить следующий проект ИС, автоматизирующий и значительно упрощающий работу менеджера в рассматриваемой области. Все данные об автомобилях, клиентах и заказах будут храниться в таблицах базы данных, редактирование и просмотр которых будет осуществляться в наглядной и простой форме. Практически никаких записей на бумаге делать не придётся. Клиент будет иметь возможность удалённо просматривать и выбирать авто для аренды, и являться в офис только для оплаты аренды и получения автомобиля.

Одной из задачей, выполняемой проектом, будет являться показ всех автомобилей, которые могут быть арендованы (не арендуются в данный момент). Для этого нужно обновлять состояние автомобиля в БД при его аренде и возврате.

Также актуально обеспечить обработку таких запросов как показ всех клиентов (зарегистрированных в системе), заказов (с данными о клиенте, авто, времени аренды).

## 2.3 Основные понятия предметной области

Прокат автомобилей – это деятельность компании по предоставлению автомобилей на ограниченный срок эксплуатации с целью получения финансовой выгоды.

Автомобиль – транспортное средство, являющиеся предметом аренды.

Клиент – лицо, которое арендует автомобиль на ограниченный срок эксплуатации.

Менеджер– работник, занимающийся оформлением договора аренды ТС, контролем данных о автомобилях и клиентах.

Администратор – управляющий филиала проката автомобилей, заведующий менеджерами.

Контракт – договор об аренде, состоящий из данных клиента, автомобиля и сроков аренды.

Возврат авто – сдача клиентом автомобиля обратно компании.

Особенностью концептуальной диаграммы классов являются связи между сущностями.

* Связи между сущностями «Менеджер», «Администратор», «Клиент» и «Пользователь» - является. Т.е. сущности наследуются от класса «Пользователь»;
* Связи между сущностями «Менеджер», «Пользователь», «Автомобиль» и «Аренда» являются – композицией. Экземпляр сущности «Аренда» не может быть создан без экземпляров этих сущностей;
* Связь между сущностями «Клиент» и «Отзыв» - агрегация. Оба объекта могут существовать независимо друг от друга, т.е. отзыв может быть оставлен пользователем или пользователь предпочел остаться анонимным при оставлении отзыва, тогда отзыв будет от анонимного пользователя.

Концептуальная диаграмма классов представлена на рисунке 1.1.

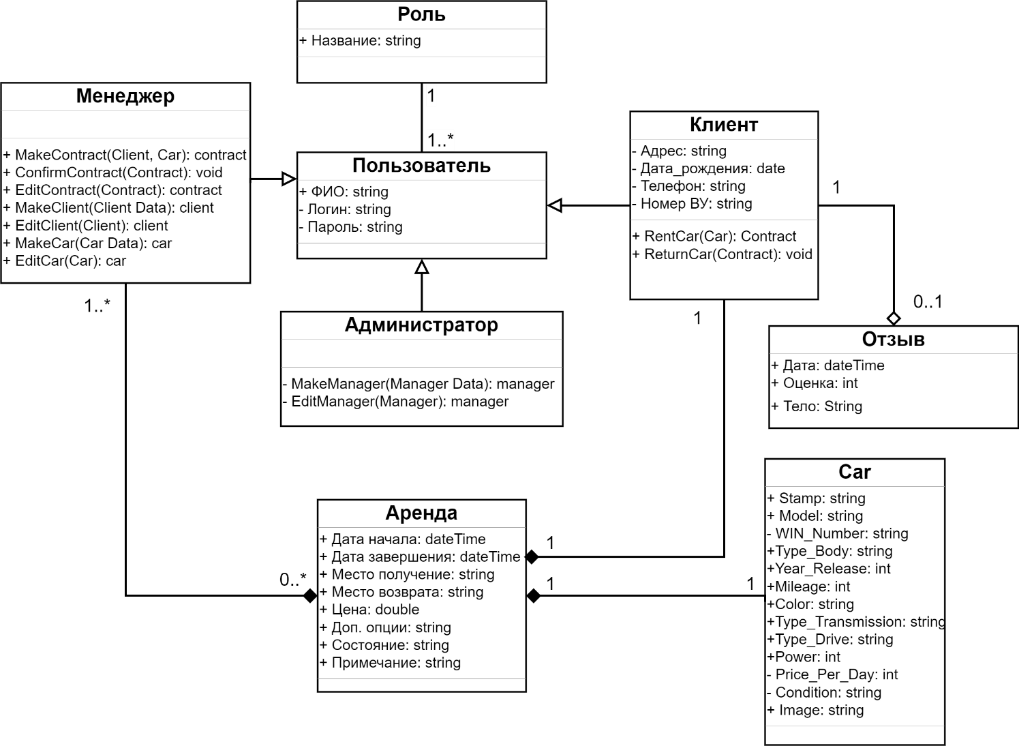


Рисунок 1.1 – Концептуальная диаграмма классов.

## 2.4 Функциональные требования к системе

В процессе анализа предметной области, были выделены следующие функциональные требования к разрабатываемому прототипу информационной системы:

* Система должна обеспечивать возможность регистрации / авторизации / аутентификации пользователей;
* Система должна обеспечивать поддержку различных ролей пользователей (таких как клиент, менеджер, администратор) с соответствующем функционалом;
* Система должна обеспечивать автоматическое установление роли «клиент», зарегистрированному пользователю;
* Система должна обеспечивать возможность установления роли пользователю для пользователей с ролью администратор;
* Система должна обеспечивать возможность добавление, удаления и редактирования автомобилей в каталоге с прикреплением изображения для пользователей с ролями менеджер и администратор;
* Система должна обеспечивать возможность добавления, удаления и редактирования клиентов для пользователей с ролями менеджер и администратор;
* Система должна обеспечивать демонстрацию каталога автомобилей для всех пользователей (в том числе не авторизованных);
* Система должна обеспечивать поиск авто по критериям (таким как, марка, модель, год выпуска, класс, тип кузова, тип КП, цена) для всех пользователей;
* Система должна обеспечивать демонстрацию подробной информации об автомобиле для всех пользователей;
* Система должна обеспечивать возможность арендовать автомобиль со страницы с подробной информацией об авто для пользователей с ролью клиент;
* Система должна обеспечивать возможность создания договора аренды, для клиентов, не имеющих активных или ожидающих подтверждения аренд;
* Система должна обеспечивать выбор дополнительных опций к аренде, а также место доставки авто, при оформлении аренды пользователем;
* Система должна обеспечивать выполнение функции для расчёта стоимости аренды исходя из параметров: цена посуточной аренды авто, срок аренды (не менее 1 дня), наличие доп. опций, способ получения авто (из офиса или доставка);
* Система должна обеспечивать возможность редактирования и удаления договоров аренды для пользователей с ролями менеджер и администратор;
* Система должна обеспечивать демонстрацию всех аренд для пользователей с ролями менеджер и администратор, а также демонстрацию личных аренд для клиентов;
* Система должна обеспечивать возможность удалённого завершения аренды т.е. клиент может завершить аренду, не приходя в офис, а дождавшись курьера, сдать автомобиль ему;
* Система должна обеспечивать демонстрацию всех отзывов клиентов;
* Система должна обеспечивать возможность клиентам оставить отзыв о компании;

## 2.5 State of Art

В качестве СУБД для организации хранения данных была выбрана MySQL. Для подключения приложения к БД будет использоваться JDBC. В качестве реализации карты на странице «Контакты» выбран сервис от Яндекс – Яндекс Карта.

Также будет использовано MapAPI MapQuest для расчёта стоимости доставки авто.

Само приложение будет реализовано на ЯП Java v18, с использованием Spring Boot.

## 2.6 Нефункциональные требования к системе

Были выделены следующие нефункциональные требования:

* реализация на платформе Java;
* использование Spring framework;
* база данных MySQL;
* использование Hibernate для работы с данными.
* Система должен иметь привлекательный дизайн;
* Система должна иметь понятный русскоязычный интерфейс;
* Система должна открываться в течение не больше 5 секунд;
* Отклик системы для типовых действий не должен превышать 3 секунд для средней нагрузки (10 – 15 пользователей);
* Система должна обеспечивать стабильную работу при одновременном использовании 50 пользователями;
* Клиент должен взаимодействовать с системой посредством протокола HTTP.
* Система должна корректно работать на браузерах: Yandex, Google, Opera, Firefox;

## 2.7 Сравнительный анализ аналогов

Анализируя похожие информационные системы, мной были выделены следующие аналоги, выполняющие те же функции что и разрабатываемая система:

* <https://rentride.ru/>;
* <https://gidprokat.ru/>;
* <https://goodokrent.ru/>;
* <https://localrent.com/>;

Был проведён анализ на основе методики с использованием весовых коэффициентов, для определения наилучшей системы из аналогов. Для получения наиболее обоснованного результата была использована методика определения обобщённой степени соответствия альтернатив.

Для оценки альтернатив был выделен ряд критериев, по которым была произведена оценка продуктов.

Значения выбранных критериев оценки: оценка может быть в интервале от [1; 10].

**Удобность навигации по сайту –** в данном случае рассматривается сложность поиска необходимого раздела/страницы на сайте продукта пользователем.

Значения:

– 8 – 10 любую страницу или раздел сайта легко найти из любого раздела продукта;

– 5 – 7 поиск некоторых страниц или разделов может вызвать затруднение у пользователя;

– 0 – 4 совершенно непонятная организация навигации по сайту.

**Возможность регистрации/авторизации –** возможность пользователю создавать личный кабинет, с его историей аренд.

Значения:

– 5 – 10 пользователю доступен личный кабинет с информацией о его арендах;

– 0 – 4 личный кабинет пользователя отсутствует или не обладает необходимыми данными.

**Простота и удобство пользовательского интерфейса –** оценка графического интерфейса пользователя.

Значения:

– 8 – 10 простой и понятный пользовательский интерфейс с приятным дизайном;

– 5 – 7 нагруженный или недостаточно информативный пользовательский интерфейс;

– 3 – 4 неприятный дизайн пользовательского интерфейса и сложность его понимания.

**Функциональность поиска авто по критериям –** возможность поиска авто по различным критериям и наличие необходимых критериев для нахождения интересуемого автомобиля.

Значения:

– 8 – 10 наличие широкого поиска автомобиля по большому числу необходимых критериев;

– 2 – 7 скудный на критерии поиск или его полное отсутствие.

**Сложность формы аренды для аренды авто –** оценка загруженности формы для оформления аренды автомобиля.

Значения:

– 8 – 10 присутствие только необходимых полей для оформления договора аренды, приятное визуальное отображение формы;

– 5 – 7 перегруженность формы излишними полями;

– 3 – 4 недостаточное количество полей для оформления договора аренды и непривлекательный интерфейс пользователя.

**Наличие дополнительных услуг –** оценка количества доп. услуг и их цены.

Значения:

– 5 – 10 наличие основных дополнительных услуг и адекватное ценообразование;

– 0 – 4 полное или частичное отсутствие основных доп. услуг или завышение цен за их использование.

**Возможность доставки автомобиля –** возможность получения/возврата авто в необходимом месте.

Значения:

– 8 – 10 возможность получать и забирать автомобиль в необходимом месте;

– 5 – 7 получение авто возможно доставкой непосредственно к клиенту, а возврат автомобиля можно осуществить только в офис компании;

– 0 – 4 получение и возврат автомобиля возможны только из офиса компании;

Результат оценки аналогов по каждому критерию представлен в таблице 2.1.

Таблица 2.1 – Оценка систем по ряду критериев:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Критерии \  Аналоги | Удобство навигации | Возможность рег./автор. | Удобство UI | Функциональность поиска | Сложность формы аренды | Наличие доп. услуг | Получение/возврат авто в любом месте |
| rentride.ru | 9 | 10 | 10 | 10 | 6 | 5 | 9 |
| gidprokat.ru | 9 | 0 | 9 | 8 | 4 | 0 | 6 |
| goodokrent.ru | 7 | 2 | 6 | 4 | 8 | 2 | 9 |
| localrent.com | 7 | 0 | 7 | 7 | 8 | 10 | 9 |
| dostupnoeauto.ru | 7 | 0 | 5 | 2 | 4 | 0 | 3 |

Для оценки продуктов, каждому из критериев будет сопоставлен весовой коэффициент (Ci).

Критерии и их веса представлены в таблице 2.2.

Вес критерия – определяет, на сколько сильное влияние оказывает тот или иной параметр на итоговое значение.

Взвесим критерии, т.е. определим, значение каких из них наиболее влияют на выбор системы.

Таблица 2.2. Критерии и весовые коэффициенты.

|  |  |
| --- | --- |
| Критерии | Вес (Ci) |
| Удобство навигации | 0,15 |
| Возможность регистрации/авторизации | 0,17 |
| Удобство пользовательского интерфейса | 0,02 |
| Функциональность поиска | 0,2 |
| Сложность формы оформления аренды | 0,2 |
| Наличие дополнительных услуг | 0,11 |
| Возможность получения/возврата авто где угодно | 0,15 |

Выполним оценку систем, подсчитая взвешенные оценки. Для этого для каждого аналога суммируем оценки по критериям х умноженным на веса этих критериев (Ci).

Таблица с результатами подсчёта взвешенных оценок представлена в таблице 2.3.

Таблица 2.3. Результат подсчёта взвешенных оценок.

|  |  |
| --- | --- |
| Аналог | Взвешенные оценки |
| rentride.ru | 8,35 |
| gidprokat.ru | 4,83 |
| goodokrent.ru | 5,48 |
| localrent.com | 6,64 |
| dostupnoeauto.ru | 2,8 |

Исходя из полученных данных можно сделать вывод, что система, использующаяся компанией rentride.ru имеет аналогичный функционал с проектируемом нами программным решением.

Таким образом, при разработке прототипа программной системы «Прокат автомобилей» следует сделать акцент на следующие аспекты, выделенные у фаворита среди аналогов:

* Наличие личного кабинета с возможностью просмотра истории аренд и информации о текущей аренде пользователя;
* Сложность формы оформления аренды (форма не должна содержать 100500 полей для заполнения, но и не нужно останавливаться только на номере телефона, для обсуждения деталей по звонку);
* И последнем, но не по значению, следует выделить UI и удобство навигации по системе (пользователю должно быть приятно находится на сайте, и не должно возникать никаких проблем с нахождением той или иной информации).

# 3 АНАЛИЗ ЗАДАЧИ И ФУНКЦИОНАЛЬНАЯ ДЕКОМПОЗИЦИЯ СИСТЕМЫ

После описания предметной области и сравнительного обзора аналогов проектируемого прототипа программной системы требуется определить основные бизнес-процессы для рассматриваемой предметной области, требующие автоматизации.

В данном разделе бизнес-процессы компании по прокату авто были рассмотрены с двух точек зрения:

* Точка зрения «как есть»;
* Точка зрения «как должно быть» после частичной или полной автоматизации.

Также были построены функциональные IDEF0 модели с данных точек зрения, выделены основные сценарии взаимодействия пользователя с системой и подробно рассмотрены некоторые из них.

## 3.1 Анализ бизнес-процесса «Прокат автомобилей»

**Участники процесса**

Пользователь – человек, который имеет водительское удостоверение категории В, и нуждается в предоставлении автомобиля для использования в своих целях.

Основная целевая аудитория разрабатываемой ИС – это мужчины и женщины от 21 до 65 лет со стажем вождения от 2-х лет, не имеющие личного транспорта.

Второстепенная целевая аудитория – люди, имеющие свой автомобиль, которым необходим дополнительный для удовлетворения их потребностей.

Из всей аудитории можно выделить несколько типов пользователей:

1) Иван 21 – 35 лет. Женат. Имеет личный автомобиль «Lada PRIORA». Доход около 35 000 рублей в месяц. Любит встречаться с друзьями и отдыхать с семьёй. Часто пользуется услугами ИС для аренды автомобиля на короткий срок (поездка с друзьями/семьёй либо кичиться на встрече выпускников).

2) Михаил 30-55 лет. Женат. Имеет 3 детей. Личного авто нет. Семейный доход более 70 000. Каждый день ездит в город на работу, часто ездит с детьми (в больницу, магазин и т.п.) и к тёще в деревню по выходным. Предпочитает личному автомобилю арендованный, часто пользуется услугами ИС на длительный период.

3) Владимир 35-65 лет. Не женат. Доход более 100 тыс. рублей в месяц. 2 раза в месяц бывает в городе в командировки. Любит сам управлять автомобилем. Редко арендует автомобиль на короткий срок (деловая поездка и т.п.).

**Перечень ролей.**

Выделенные роли пользователей ИС:

1) Клиент

2) Менеджер

3) Администратор

Описание каждой роли представлено в словаре предметной области пункт [2.3](#_2.3_Основные_понятия). Функционал каждой роли представлен ниже:

Функционал клиента:

• Просмотр и поиск авто;

• Аренда/возврат авто;

• Возможность оставлять отзыв о компании.

Функционал менеджера:

• Подтверждение/отмена/изменение аренды;

• Добавление/редактирование авто;

• Добавление/редактирование пользователей;

Функционал администратора:

• Удаление отзывов;

• Изменение ролей пользователям;

**Область расширения.**

Ожидаемая область расширения: г. Владимир и Владимирская обл. Ожидаемое количество обслуживаемых клиентов: 50 человек в месяц.

**Описание документов.**

Основными документами предметной области являются:

• Клиент – атрибуты: ФИО, логин, пароль, адрес проживания, телефон, дату рождения, номер ВУ.

Данная сущность создаётся пользователем после регистрации в системе.

Он может быть изменён как самим клиентом, так и администратором в процессе жизни. Документ может быть уничтожен по просьбе пользователя при удалении аккаунта.

Данный документ вовлечен в процессы: «Арендовать авто» и «Завершить аренду», «Оставить отзыв».

Возможные состояния у документа отсутствуют.

• Администратор – атрибуты: ФИО, логин, пароль.

Данная сущность создаётся при регистрации нового пользователя и назначения ей роли «Администратор» непосредственно через систему.

Изменения над данным документом может производиться только самим администратором. Документ не может быть уничтожен, только изменён в случае увольнения сотрудника.

Данная сущность вовлечена в процессы: «Назначить роль пользователю», «Удалить пользователя», «Добавить пользователя», а также во все процессы, в которые вовлечён документ Менеджер.

Возможные состояния у документа отсутствуют.

• Менеджер – атрибуты: ФИО, логин, пароль.

Данная сущность создаётся при регистрации нового пользователя и назначения ей роли «Менеджер» администратором.

Изменение над данным документом могут производить только администраторы. Документ может быть уничтожен администратором (путём удаления пользователя из системы) при увольнении сотрудника.

Данный документ вовлечен в процессы: «Редактировать информацию об авто», «Удалить авто из каталога», «Добавить авто в каталог», «Подтвердить аренду», «Отменить аренду», «Завершить аренду», «Посмотреть список аренд».

Возможные состояния у документа отсутствуют.

• Автомобиль – атрибуты: марка, модель, WIN-номер, тип кузова, год выпуска, пробег, цвет, тип КП, тип привода, кол-во лс, цена за день аренды, состояние, изображение.

Данная сущность создаётся после выполнения процесса «Добавить авто в каталог».

Изменение над данным документом могут производить администраторы и менеджеры. Документ может быть уничтожен после выполнения прецендента «Удалить авто из каталога».

Данный документ вовлечён в процессы: «Найти авто», «Редактировать информацию об авто», «Удалить авто», «Посмотреть информацию об авто», «Арендовать авто», «Подтвердить аренду», «Отменить аренду», «Завершить аренду».

Данный документ может иметь два возможных состояния:

Свободен – не арендуется в данный момент.

Арендуется – находиться в аренде.

• Контракт – атрибуты: ФИО клиента, ФИО менеджера, марку авто, модель, WIN-номер, дополнительные опции, дату начала аренды, дату конца аренды, стоимость аренды, место получения авто, место возраста авто, состояние (подтверждён/не подтверждён/действует), примечания.

Данная сущность создаётся после выполнения процесса «Арендовать авто».

Изменение над данным документом могут производить администраторы и менеджеры. Документ может быть уничтожен после прошествия 1 года с момента выполнения прецендента «Завершить аренду».

Данный документ вовлечён в процессы: «Подтвердить аренду», «Отменить аренду», «Завершить аренду», «Посмотреть список аренд».

Данный документ может иметь 6 возможных состояния:

Не подтверждён – клиент нажал кнопку «Арендовать», договор был создан и ждёт подтверждения менеджера/администратора.

Подтверждён – менеджер проверил все данные и нажимает кнопку «Подтвердить».

Отменён – менеджер отменил аренду из-за неправильности введённых данных или клиент сам отменил аренду.

Действует – когда клиент оплачивает аренду и забирает автомобиль, контракт переходит в состояние Действует.

Ожидает оплаты штрафа – клиент вернул автомобиль позже срока завершения аренды или состояние автомобиля после аренды не соответствует состоянию до аренды.

Завершён – клиент вернул автомобиль к сроку истечения аренды, и менеджер завершил аренду.

• Отзыв – атрибуты: ФИО клиента, дата, счёт, тело.

Данная сущность создаётся после выполнения процесса «Задать вопрос».

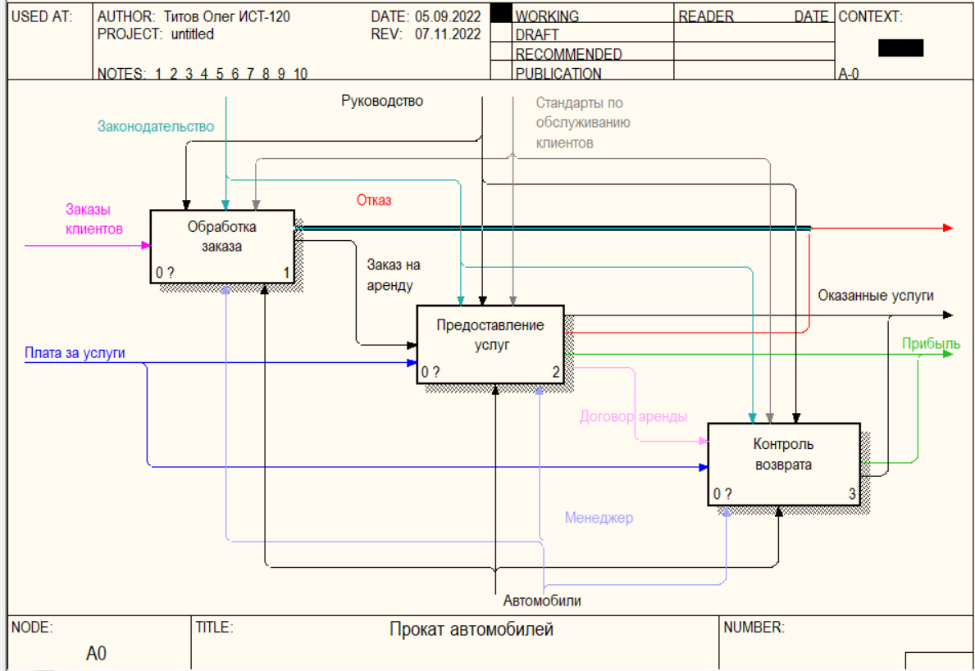
Изменение над данным документом могут производить администраторы и менеджеры. Документ может быть уничтожен в любое время по желанию администратора.

Данный документ вовлечён в процессы: «Оставить отзыв», «Просмотр отзывов».

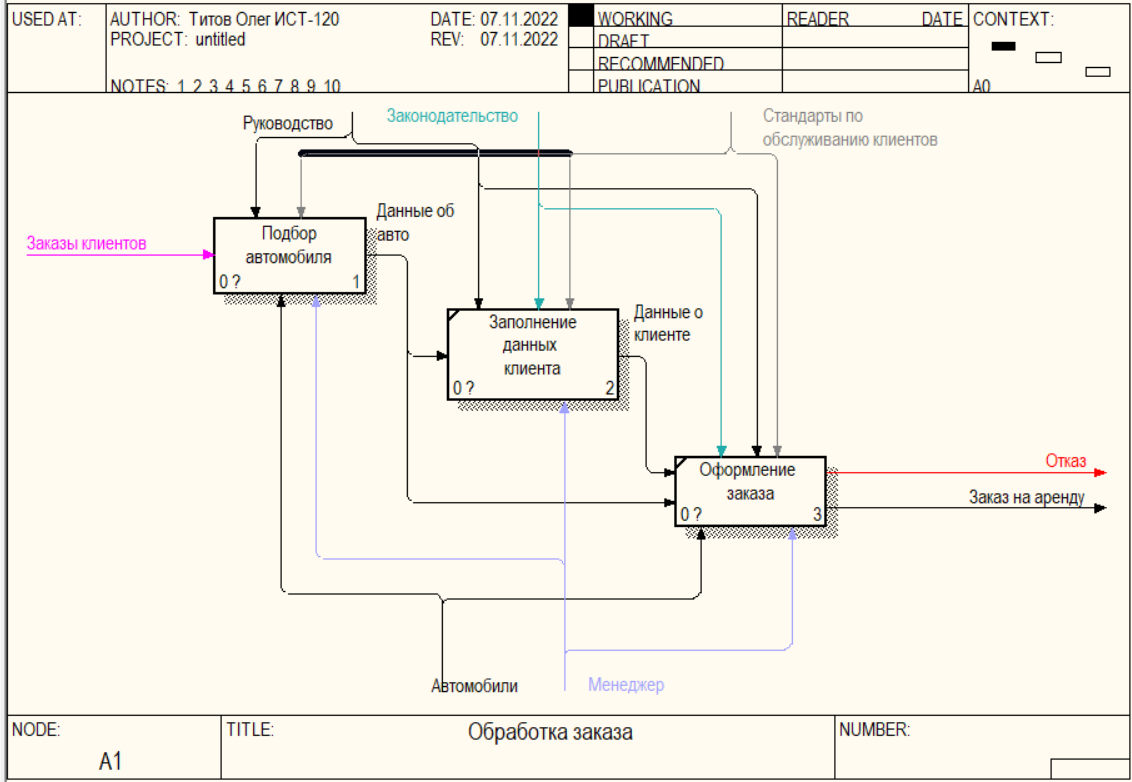
Возможные состояния у документа отсутствуют.

**Диаграммы бизнес-процессов**

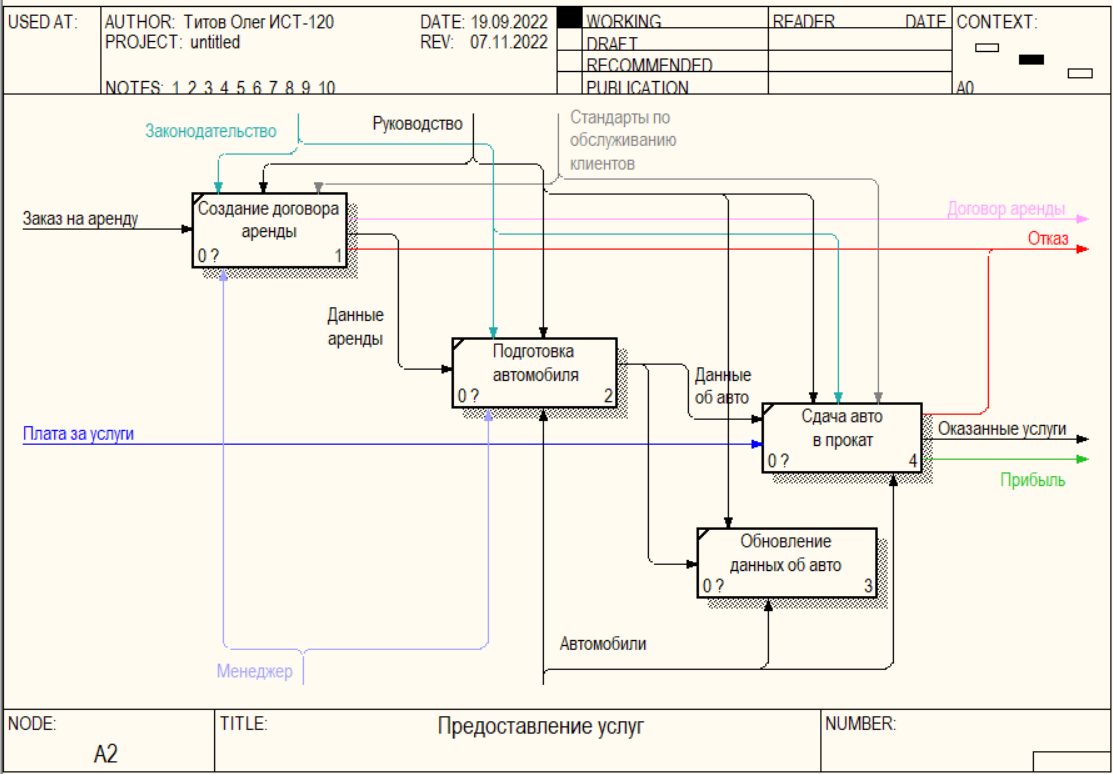
AS-IS диаграмма автоматизируемого бизнес-процесса «Прокат автомобилей» представлена в виде основных IDEF0 диаграмм на рисунке 2.1. Остальные диаграммы приведены в приложении В.



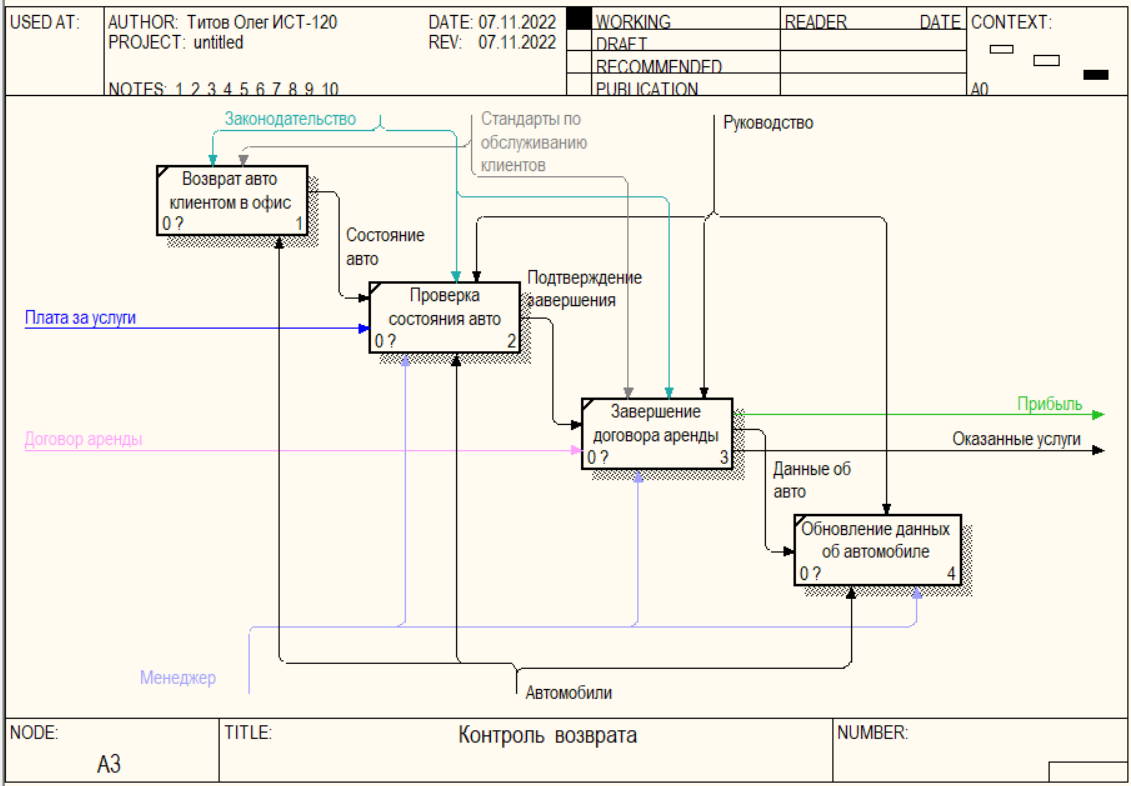
а



б



в



г

Рисунок 2.1. – AS-IS диаграмма автоматизируемого бизнес-процесса Название: *а – Диаграмма А0;б – А1 Диаграмма работы «Обработка заказа»; в – А2 Диаграмма работы «Предоставление услуг»; г – А3 Диаграмма работы «Контроль возврата».*

TO-BE диаграмма автоматизируемого бизнес-процесса «Прокат автомобилей» представлена в виде BPMN диаграмм на рисунке 2.2.

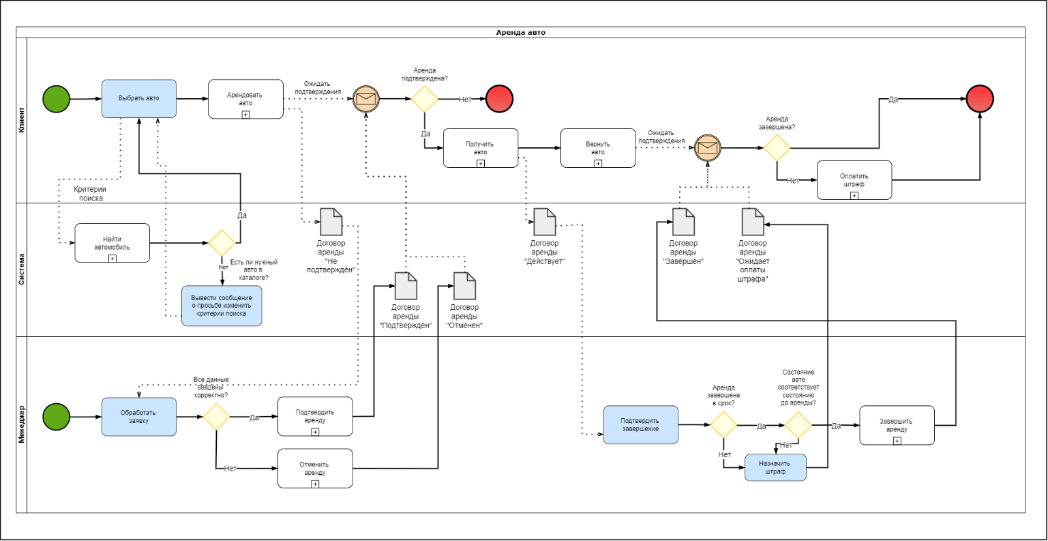


Рисунок 2.2 – TO-BE диаграмма автоматизируемого бизнес-процесса.

## 3.2 Сценарий взаимодействия пользователя с системой

Диаграмма прецедентов разрабатываемого прототипа ПС «Прокат автомобилей» представлена на рисунке 2.3.

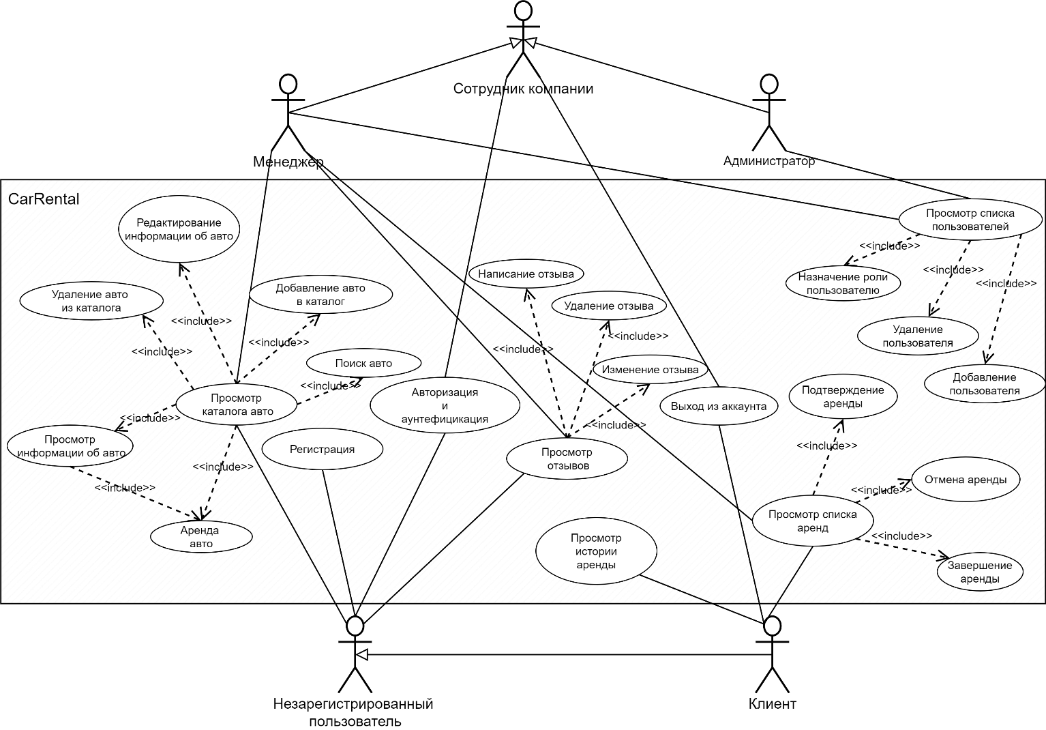


Рисунок 2.3 – Диаграмма прецедентов ПС «Прокат автомобилей».

Данная диаграмма демонстрирует возможные сценарии действия, поддерживаемые разрабатываемым прототипом программной системы «Прокат автомобилей».

Диаграммы состояния документов «Контракт» и «Автомобиль», демонстрирующие состояния и условия изменения данных состояний у документов представлены на рисунке 2.4.

Один из возможных бизнес-процессов работы ПС с арендой представлен ниже:

* Один из возможных бизнес-процессов работы ИС с заказом представлен ниже:

1. клиент (арендатор) оформил заказ на аренду, указав свои данные. Объект «Договор аренды» создан в системе (запись в БД). Поскольку оплата заказа производится непосредственно перед получением авто, заказ частично обрабатывается до момента оплаты;
2. Менеджер отменил заказ из-за неверных данных. Пользователю на сайте в разделе «Мои аренды» аренда отмечается как отменённая, после чего переходим к п.8;
3. Менеджер подтвердил заказ. Пользователю на указанный им номер поступает звонок с подтверждением аренды. Данные об авто, выбранном клиентом, при оформлении аренды, передаются в систему для обновления состояния автомобиля; заказ ожидает оплаты.
4. Клиент оплатил аренду. Заказ помечается как действующий и переходит в состояние ожидания завершения аренды.
5. Клиент возвращает автомобиль в срок завершения аренды. Аренда помечается как завершённая.
6. Клиент не возвращает автомобиль в срок. В этом случае устанавливается штраф за просроченную аренду.
7. Клиент возвращает автомобиль в срок/не в срок с повреждениями. В этом случае устанавливается штраф за повреждение авто/задержку аренды.
8. Аренда помечается как отменённая;



а



б

Рисунок 2.4 – Диаграммы состояния Название: *а – Диаграмма состояния документа «Контракт»; б – Диаграмма состояния документа «Автомобиль».*

## 3.3 Поиск авто

У актёра «Клиент» рассмотрим прецедент «Поиск авто», который охватывает два подсценария: успешного нахождения нужного авто и неудачного поиска. Выбранный прецедент подробно описан в таблице 3.1.

Таблица 3.1. Расширенное описание прецедента «Поиск авто».

|  |
| --- |
| **Название:** «Поиск авто»  **Предусловие:** клиент зарегистрирован в ИС «Прокат авто», т.е. имеет присвоенный ему индивидуальный номер, он авторизован в системе.  **Действующее лицо:** клиент  **Основной поток:** *поиск автомобиля*  Пользователь открывает страницу приложения, отображающую каталог автомобилей. Пользуется средствами поиска по каталогу (вводит марку или тип авто в соотв. поля).  Выбирает необходимый автомобиль и нажимает кнопку «Арендовать» - данные выбранного авто (прецедент «Аренда авто»).  Система обновляет данные авто и сохраняет требование в БД и перенаправляет его менеджеру.  **Альтернативный поток:** автомобиль в каталоге отсутствует.  На 2 шаге пользователь не находит требующийся автомобиль. В этом случае он:  либо начинает поиск другого авто,  либо корректирует условия поиска. |

## 3.4 Добавление авто в каталог

У актёра «Менеджер» рассмотрим прецедент «Добавление авто в каталог», который охватывает два подсценария: успешного добавления нужного авто и неудачного добавления. Выбранный прецедент подробно описан в таблице 3.2.

Таблица 3.2. Расширенное описание прецедента «Добавление авто в каталог».

|  |
| --- |
| **Название:** «Добавление авто в каталог»  **Предусловие:** менеджер авторизован в ИС «Прокат авто».  **Действующее лицо:** пользователь с ролью менеджера  **Основной поток:** *добавление автомобиля в каталог*  Менеджер открывает страницу приложения, отображающую список всех авто.  Водит данные о новом автомобиле в соотв. поля.  Система сохраняет данные в БД и обновляет каталог авто.  **Альтернативный поток:** нет соединения с БД.  На 3 шаге система выдаёт менеджеру сообщение об ошибки соединения с БД.  Менеджер перезагружает страницу и повторяет действия основного потока заново.  **Постусловие:** если авто добавлено успешно должна появиться соответствующая запись в каталоге всех авто. |

## 3.5 Завершение аренды

У актёра «Клиент» рассмотрим прецедент «Завершение аренды», который охватывает два подсценария: успешного возврата авто и неудачного возврата. Выбранный прецедент подробно описан в таблице 3.3.

Таблица 3.3. Расширенное описание прецедента «Завершение аренды».

|  |
| --- |
| **Название:** «Завершение аренды»  **Предусловие:** клиент зарегистрирован в ИС «Прокат авто», т.е. имеет присвоенный ему индивидуальный номер, он авторизован в системе, а также имеет активную аренду авто.  **Действующее лицо:** пользователь с ролью клиент  **Основной поток:** *завершение аренды*  Пользователь открывает страницу приложения, отображающую его активную аренду.  Выбирает запись и нажимает кнопку «Вернуть».  Система обновляет данные об авто в БД и перенаправляет их менеджеру.  **Альтернативный поток:** автомобиль был повреждён.  На 2 шаге пользователь отмечает неисправность авто. В этом случае он:  Выплачивает сумму ущерба, определённую менеджером.  **Альтернативный поток:** автомобиль был сдан не в срок.  На 2 шаге при попытке сдачи авто, если дата сдачи уже прошла, то пользователь:  выплачивает штраф за просроченный период.  **Постусловие:** если авто возвращено успешно должны обновиться данные об авто в БД, и добавлена запись в историю аренды пользователя. |

На данном этапе проведён достаточный анализ предметной области для проектирования прототипа программной системы «Прокат автомобилей».

# 4 СТРУКТУРНАЯ ОРГАНИЗАЦИЯ СИСТЕМЫ

В данной главе рассматривается архитектура разрабатываемого прототипа программной системы "Прокат автомобилей". Архитектура системы определяет общую структуру и организацию компонентов, их взаимодействие и роли в системе. Основная цель архитектуры системы состоит в обеспечении гибкости, расширяемости и поддерживаемости системы.

При проектировании программной системы "Прокат автомобилей" была выбрана клиент-серверная архитектура. Данная архитектура позволяет логически разделить функциональность системы на клиентскую и серверную части, обеспечивая эффективное взаимодействие между ними.

Клиентская часть представляет собой веб-приложение, доступное пользователям через веб-браузер. Она обеспечивает удобный и интуитивно понятный интерфейс для клиентов, позволяя им просматривать доступные автомобили, оформлять аренду, оставлять отзывы и управлять своими данными. Клиентская часть разработана с использованием технологии Freemarker для реализации динамических шаблонов.

Серверная часть системы состоит из компонентов бизнес-логики, отвечающих за обработку запросов клиентов, взаимодействие с базой данных и выполнение необходимых операций. Для реализации бизнес-логики был использован Spring Boot framework.

Выбор веб-приложения в качестве варианта реализации системы "Прокат автомобилей" обусловлен рядом факторов. Веб-приложение обеспечивает широкую доступность и удобство использования для пользователей, поскольку они могут получить доступ к системе через стандартный веб-браузер на различных платформах.

Кроме того, веб-приложение позволяет легко масштабировать систему, добавлять новые функциональные возможности и поддерживать ее с помощью веб-сервера. Это дает возможность разрабатывать и поддерживать систему независимо от конкретной операционной системы на стороне клиента.

Таким образом, выбор веб-приложения в качестве варианта реализации системы "Прокат автомобилей" обеспечивает удобство использования для клиентов, гибкость и масштабируемость системы, а также облегчает ее разработку и поддержку.

## 4.1 Организация доступа к данным

В процессе разработки прототипа программной системы была разработана диаграмма-сущностей в нотации UML.

Данная диаграмма, представляет основные классы разрабатываемой системы и их взаимоотношения на программном уровне.

Основными отличиями от концептуальной диаграммы классов являются:

* Наличие в таблицах сущностей первичных ключей (поля id), для организации целостности и непротиворечивости данных;
* Также было решено, не выделять отдельные сущности под разные роли пользователей системы, а сделать для них единую сущность «user», для упрощения работы с пользователями.

Разработанная диаграмма представлена на рисунке 4.1.

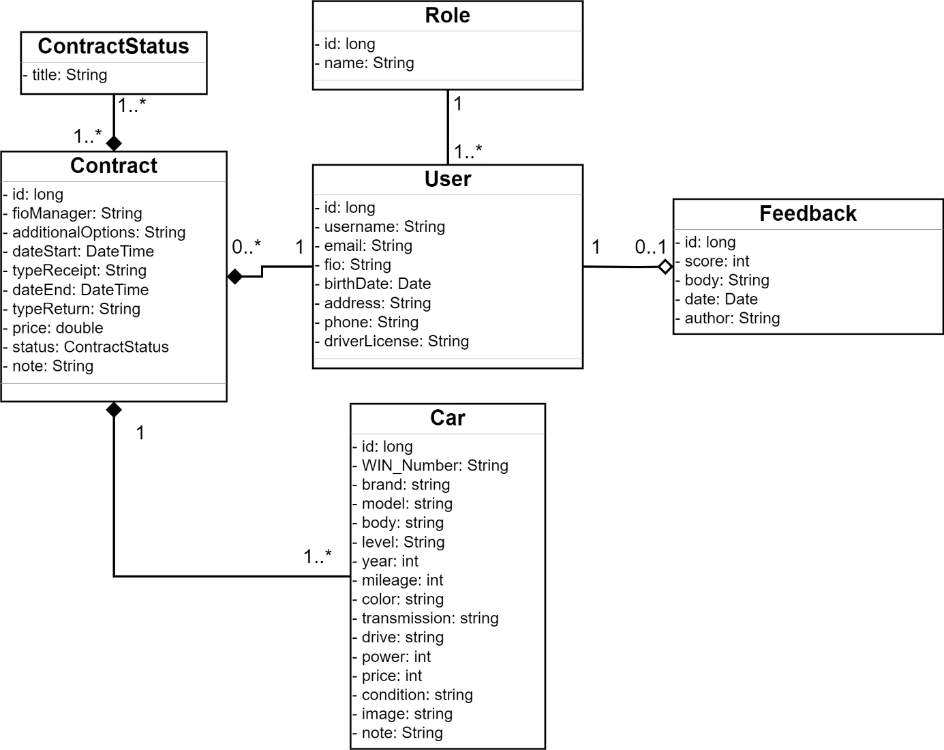


Рисунок 4.1 – Диаграмма классов-сущностей ULM*.*

Классами-сущностями разрабатываемого прототипа программной системы являются: роль, пользователь, отзыв, контракт аренды и автомобиль. Описание данных сущностей представлено в пункте [3.1](#_3.1_Анализ_бизнес-процесса).

## 4.2 Организация бизнес-логики

В процессе разработки прототипа программной системы была разработана диаграмма классов уровня бизнес-логики. Разработанная диаграмма представлена на рисунке 4.2.



Рисунок 4.2 – Диаграмма классов уровня бизнес-логики*.*

## 4.3 Организация веб-интерфейса

Разработка макета UI производилась в приложении Figma.

Дизайн страниц разрабатывался с оглядкой на аналоги и выбирались самые лучшие дизайнерские решения из них.

На каждой странице сайта будет находится верхнее меню навигации, и блок footer внизу страницы, с помощью которых можно будет перейти на основные разделы сайта такие как:

• Страница «О нас»;

• Страница «Контакты»;

• Страница «Автопарк»;

• Страница «Вопросы»;

• Страница «Аренды»;

• Страницы «Регистрация» или «Авторизация».

а также вернуться обратно.

Были разработаны следующие экранные формы:

• Экранная форма главной страницы;

• Экранная форма каталога авто;

• Экранная форма страницы «О нас»;

• Экранная форма страницы «Контакты»;

• Экранная форма страницы с подробной информацией об авто;

• Экранная форма «Регистрации»;

• Экранная форма «Авторизации»;

• Экранная форма страницы оформления аренды;

• Экранная форма страницы с историей аренд пользователя;

• Экранная форма страницы со списком всех аренд;

• Экранная форма страницы с подробной информации об аренде;

• Экранная форма страницы с вопросами;

Также была разработана диаграмма навигации сайта, которая представлена на рисунке 4.2.



Рисунок 4.2 – Диаграмма навигации*.*

Разработанные шаблоны основных экранных форм представлены на рисунках 4.3-4.6.

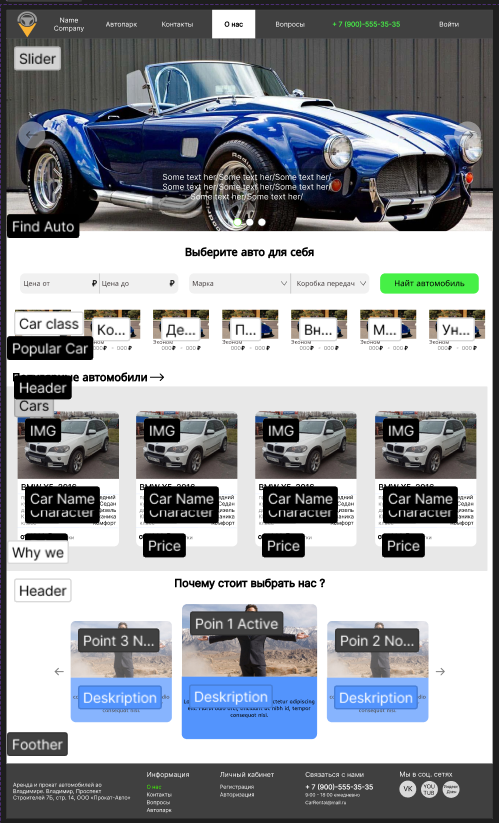


Рисунок 4.3 – Экранная форма Главной страницы.

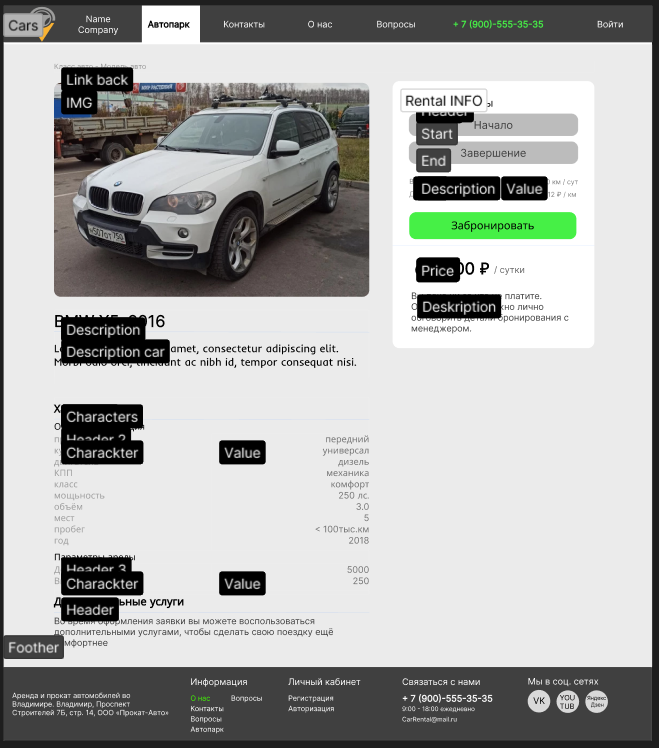


Рисунок 4.4 – Экранная форма страницы с подробной информацией об авто.

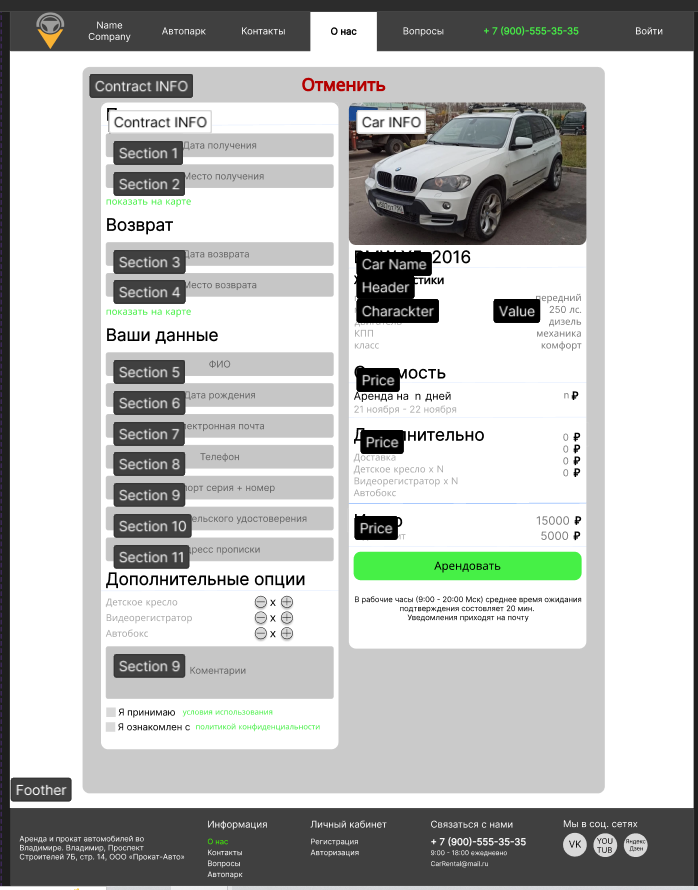


Рисунок 4.5 – Экранная форма страницы оформления аренды.

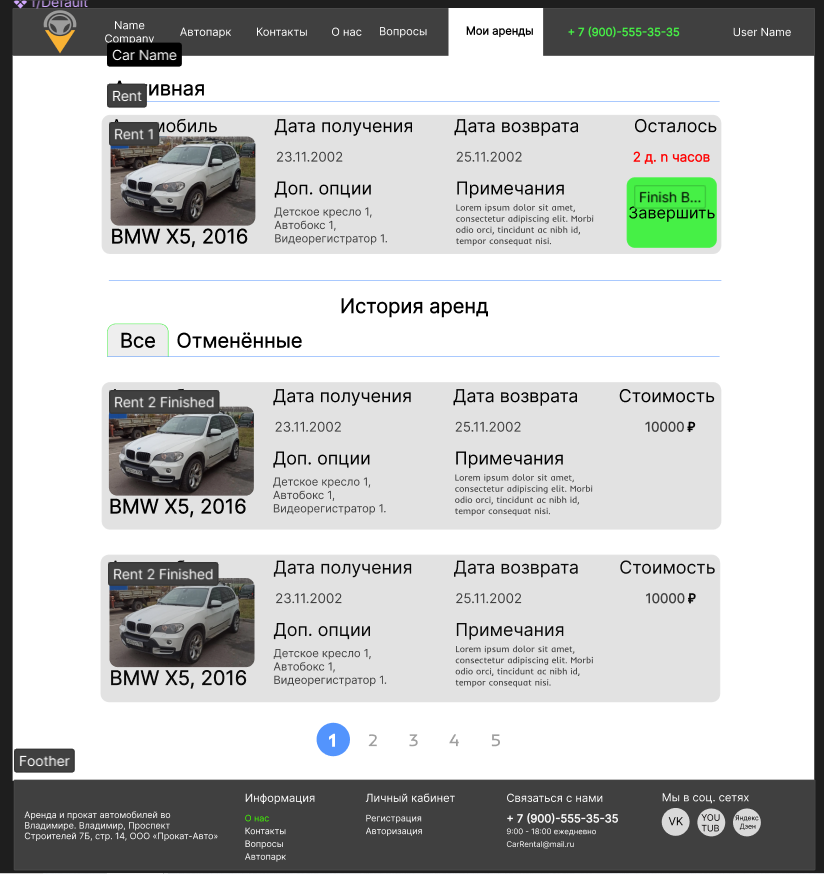


Рисунок 4.6 – Экранная форма страницы с историей аренд пользователя.

## 4.4 Взаимодействие компонентов системы для процесса «Найти авто»

Для бизнес-процесса «Найти авто» была построена диаграмма последовательности, представленная на рисунке 4.14.

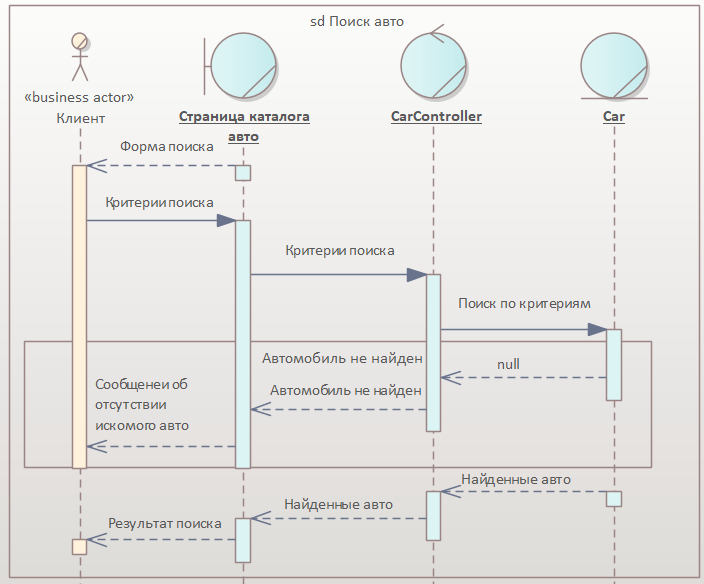


Рисунок 4.14 – Диаграмма последовательностей для бизнес-процесса «Найти авто».

Данная диаграмма демонстрирует последовательность действий, происходимых в системе, в процессе выполнения бизнес-процесса «Найти авто».

## 4.5 Взаимодействие компонентов системы для процесса «Отменить аренду»

Для бизнес-процесса «Отменить аренду» была построена диаграмма последовательности, представленная на рисунке 4.15.

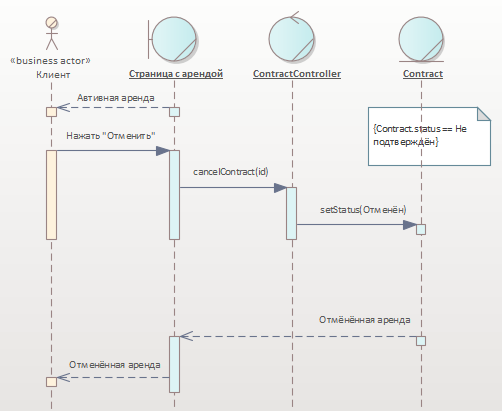


Рисунок 4.15 – Диаграмма последовательностей для бизнес-процесса «Отменить аренду».

# 5 реализация компонентов системы

## 5.1 Компоненты бизнес-логики

В данном разделе описываются компоненты бизнес-логики, которые отвечают за основную функциональность системы и обработку бизнес-процессов.

В процессе разработки прототипа программной системы «Прокат автомобилей» было создано шесть основных компонентов, отвечающих за бизнес-логику приложения:

* CarService;
* ContractService;
* FeedbackService;
* RegistrationService;
* ReportsService;
* UserService.

Компоненты реализованы с использованием Spring Framework и аннотации @Service.

Компоненты бизнес-логики взаимодействуют с другими компонентами системы, такими как репозитории, сервисы и контроллеры, для обеспечения полной функциональности системы. Интеграция компонентов осуществляется с помощью механизмов внедрения зависимостей, предоставляемых Spring Framework.

Данные компоненты обеспечивают реализацию бизнес-логики системы и являются ключевыми элементами функциональности, предоставляемой приложением.

Подробно разберём некоторые из представленных компонентов.

## 5.1.1 Класс CarService

Данный класс отвечает за управление операциями, связанными с автомобилями. Включает в себя методы для добавления, изменения, удаления и получения информации об автомобилях. Код данного компонента представлен в приложении А.

Подробно разберём несколько методов из CarService:

1. Название метода: findCar.

Входные параметры:

* Map<String, String> form, включающая в себя параметры поиска (цена от/до, марка, трансмиссия, привод, кузов, мощность, год, пробег).

Возвращаемый результат: Map<String, Object>, включающая в себя (количество страниц для реализации пагинации данных, список с найденными автомобилями, список с марками автомобилей).

Краткое описание метода: данный метод получает параметры, по которым следует произвести поиск автомобиля от контроллера CarController (который в свою очередь получает их от пользователя), затем ищет автомобили, удовлетворяющие данным критериям, и возвращает контроллеру результаты поиска.

1. Название метода: getCarList.

Входные параметры:

* int numberPage – номер страницы;
* int countItemOnPage – кол-во элементов на странице.

Возвращаемый результат: Map<String, Object>, включающая в себя (количество страниц для реализации пагинации данных, список с найденными автомобилями, список с марками автомобилей).

Краткое описание метода: данный метод получает параметры: номер страницы и кол-во элементов на странице, от контроллера CarController. Затем производит выборку из БД автомобилей, используя пагинацию, и возвращает полученный список обратно контроллеру.

## 5.1.2 Класс UserService

Данный класс отвечает за управление операциями, связанными с пользователями системы. Включает в себя методы для изменения пользователей, управления правами доступа и другими операциями, связанными с пользователями. Код данного компонента представлен в приложении А.

Подробно разберём несколько методов из UserService:

1. Название метода: editeUser.

Входные параметры:

* int userId;
* Map<String, String> rolesList, включающая в себя установленных ролей пользователю и имя пользователя.

Возвращаемый результат: boolean.

Краткое описание метода: данный метод получает параметры: ID пользователя, список установленных ему ролей и изменённое имя пользователя, затем находит пользователя в БД по переданному идентификатору и устанавливает ему новую роль и имя пользователя. Возвращает true, если пользователь был успешно изменён и false в противном случае.

1. Название метода: changePassword.

Входные параметры:

* User user;
* Map<String, String> form, включающая в себя старый пароль пользователя, новый пароль и подтверждение пароля.

Возвращаемый результат: Map<String, Object>, включающая в себя статус ответа, ошибку или сообщение о успешном выполнении.

Краткое описание метода: данный метод получает параметры: пользователь, старый пароль, новый пароль и подтверждения пароля, от контроллера UserController. Затем проверяет на соответствие полученного старого пароля и пароля из БД. Возвращает статус ответа – false и сообщение об ошибке соответствия старого и нового паролей, если пароли не соответствуют друг другу. В случае, если пароли верны, происходит проверка нового пароля и подтверждения пароля. Аналогичным образом, если пароли не совпадают, статус ответа становится – false, и добавляется сообщение об ошибке несоответствия паролей. Иначе пароль у пользователя изменяется, статус ответа устанавливается – true и добавляется сообщение об успешном изменении пароля. Затем данные передаются обратно в UserController.

## 5.2 Веб-компоненты

## 5.2.1 Классы контроллеры

В данном разделе описываются классы контроллеры, получающие запрос от клиента и возвращающие ответ в виде FTL страниц с данными от сервера.

Всего в процессе разработки прототипа программной системы «Прокат автомобилей» было разработано 9 контроллеров:

* AccountController;
* CarController;
* ContractController;
* CustomErrorController;
* FeedbackController;
* MainController;
* RegistrationController;
* ReportsController;
* UserController.

Подробно разберём несколько методов из CarController:

1. Название метода: getCarList.

Тип запроса: @GetMapping("/car/all")

Входные параметры:

* @RequestParam(required = false, defaultValue = "0") int numberPage;
* @RequestParam(required = false,defaultValue = "") String status;
* Model model.

Возвращаемый результат: String "Car/carList".

Краткое описание метода: данный метод получает от клиента GET запрос на получение всех автомобилей. Получает данные, по средствам класса CarService и возвращает их в виде FTL страницы пользователю.

1. Название метода: deleteCar.

Тип запроса: @PostMapping("/car/delete")

Входные параметры:

* @RequestParam int id;

Возвращаемый результат: String "redirect:/car".

Краткое описание метода: данный метод получает от клиента POST запрос на удаление автомобиля по переданному id. По средствам класса CarService производит удаление автомобиля из БД. Затем перенаправляет пользователя на url “/car”

## 5.2.2 FTL-страница Main.ftl

Разработка front-end части прототипа программной системы «Прокат автомобилей» осуществлялась с использованием шаблонизатора Freemarker. Freemarker — это движок шаблонов на основе Java: предназначен для поддержки HTML-веб-страниц в приложениях, следующих шаблону MVC.

Код главной страницы приложения, написанный с использованием шаблонизатора Freemarker представлен ниже.

Код страницы Main.ftl:

<#import "../parts/common.ftl" as c>

<#import "../parts/nav-bar.ftl" as nav\_bar>

<#import "../parts/foother.ftl" as futher>

<#import "../parts/car.ftl" as carParts>

<@c.page>

<link href="/static/css/Contact\_CSS.css" rel="stylesheet" />

<link href="/static/css/Button.css" rel="stylesheet" />

<link href="/static/css/About.css" rel="stylesheet" />

<div class="container body-content" style="display: flex; flex-direction: column; min-height: 100%; width: 80%;">

<br/>

<!--Slider-->

<div id="myCarousel" class="carousel slide" data-ride="carousel" style="margin-top: 4%;">

<!-- Indicators -->

<ol class="carousel-indicators">

<li data-target="#myCarousel" data-slide-to="0" class="active"></li>

<li data-target="#myCarousel" data-slide-to="1"></li>

<li data-target="#myCarousel" data-slide-to="2"></li>

</ol>

<!-- Wrapper for slides -->

<div class="carousel-inner">

<div class="item active" style="height: 500px;">

<img src="/static/img/slider/slider1.jpg" alt="Chania" style="height: 500px; width: 100%"/>

<div class="carousel-caption" style="background: rgba(40, 40, 40, 0.5)">

<h3>Аренда</h3>

<p>Прозрачные условия аренды для всех!</p>

</div>

</div>

<div class="item">

<img src="/static/img/slider/slider6.jpg" alt="Chicago" style="height: 500px; width: 100%"/>

<div class="carousel-caption" style="background: rgba(40, 40, 40, 0.5)">

<h3>Автопарк</h3>

<p>У нас огромный автопарк, где вы можете выбрать авто под любые нужды!</p>

</div>

</div>

<div class="item">

<img src="/static/img/slider/slider3.jpeg" alt="New York" style="height: 500px; width: 100%;"/>

<div class="carousel-caption" style="background: rgba(40, 40, 40, 0.5)">

<h3>Почему мы?</h3>

<p>Почему стоит выбрать именно нас? Потому что!</p>

</div>

</div>

</div>

<!-- Left and right controls -->

<a class="left carousel-control" href="#myCarousel" data-slide="prev">

<span class="sr-only">Previous</span>

</a>

<a class="right carousel-control" href="#myCarousel" data-slide="next">

<span class="sr-only">Next</span>

</a>

</div>

<!--Slider-->

<p class="headerSelect">Выберите авто для себя</p>

<@carParts.search></@carParts.search>

<@carParts.classes></@carParts.classes>

<br />

<br />

<!--Popular cars-->

<a href="/car" class="carHeader" style="text-align: left">Популярные авто →</a>

<div style="width: 100%" id="popular\_cars">

<#list cars as car>

<@carParts.carCardForMainPage car=car></@carParts.carCardForMainPage>

</#list>

</div>

<!--Popular cars-->

<!--Why us-->

<h1 class="whyUsHeader">Почему стоит выбрать нас?</h1>

<br />

<div class="WHYUs\_container">

<div class="WHY\_Us\_block\_notAtive">

<img src="/static/img/slider/slider4.jpg" class="WHY\_US\_img"/>

<div class="WHY\_US\_Text">

Наша компания ещё не вышла на рынок, но уже имеет огромную известность среди автолюбителей!

</div>

</div>

<div class="WHY\_Us\_block\_notAtive">

<img src="/static/img/slider/slider5.jpg" class="WHY\_US\_img" />

<div class="WHY\_US\_Text">

Все наши автомобили имеют страховку КАСКО и ОСАГО. Так что, вы с лёгкостью можете повторять все безумные трюки из GTA!

</div>

</div>

<div class="WHY\_Us\_block\_notAtive">

<img src="/static/img/slider/slider6.jpg" class="WHY\_US\_img" />

<div class="WHY\_US\_Text">

Нашему автопарку позавидуют даже шейхи из АОЭ! У них точно нет ГАЗ 21 волга в таком же состоянии как у нас!

</div>

</div>

</div>

<!--Why us-->

<br />

</div>

<marquee direction="right">

<h1 style="color: #404040">Some text! Some text! Some text!</h1>

</marquee>

<@futher.foother></@futher.foother>

</@c.page>

Шаблонизатор Freemarker предоставляет возможность создавать шаблон для множественного использования их в проекте.

Пример использования шаблона на странице Main.ftl представлен ниже:

<#import "../parts/car.ftl" as carParts>

…

<@carParts.carCardForMainPage car=car>

</@carParts.carCardForMainPage>

Содержимое шаблона car.ftl:

<#macro carCardForMainPage car>

<div class="col-xl-3 col-lg-4" data-id="${car.id}">

<div class="bg-info img-rounded shadow-sm">

<div style="padding: 15px; margin: 10px">

<img src="/imageCar/${car.image}" width="100%" height="220px" />

<p>

<h4>

<b>

${car.brand} ${car.model}

, ${car.year?c}

</b>

</h4>

<h4>

<b style="color: #5394FD; font-weight: 600">

<a href="/carbyclass?carClass=${car.level}&numberPage=0">${car.level}

</a>

</b>

</h4>

</p>

<hr />

<div class="auto-description-start" style=" vertical-align: top; font-size: 14px; width: 30%; display: inline-block; margin-right: 50px;">

<p class="auto-description">Коробка</p>

<p class="auto-description">Привод</p>

<p class="auto-description">Пробег</p>

<p class="auto-description">Мощность</p>

<p class="auto-description">Тип кузова</p>

</div>

<div class="auto-description-edn" style=" vertical-align: top; font-size: 14px; width: 47%; display: inline-block; text-align: right;">

<p class="auto-description">${car.transmission}</p>

<p class="auto-description">${car.drive}</p>

<p class="auto-description">&gt;${car.mileage?c}</p>

<p class="auto-description">${car.power?c} лс.</p>

<p class="auto-description">${car.body}</p>

</div>

<div class="bg-info">

<h4><b>от ${car.price?c}</b> / сутки</h4>

</div>

<h4 class="text-success text-uppercase" style="text-align: center">

</h4>

<div style="width:100%; height:100%; text-align:center;">

<a class="btn-details" href="/car/details?id=${car.id}">Подробнее

</a>

</div>

</div>

</div>

</div>

</#macro>

## 5.2.3 Конфигурирование приложения с использованием Freemarker

В данном разделе рассмотрено конфигурирование и настройка приложения с использованием Freemarker в качестве шаблонизатора для представления пользовательского интерфейса.

Для использования Freemarker в проекте необходимо добавить соответствующую зависимость в файле pom.xml:

<dependency>

<groupId>no.api.freemarker</groupId>

<artifactId>freemarker-java8</artifactId>

<version>2.1.0</version>

</dependency>

Шаблоны Freemarker хранятся в директории templates внутри ресурсов проекта.

Также для использования шаблонов Freemarker в качестве страниц, возвращаемых контроллерами, необходимо добавить в файл application.properties следующие строки:

spring.freemarker.template-loader-path=classpath:/templates

spring.freemarker.suffix=.ftl

## 5.3 Организация обработки ошибок

Организация обработки ошибок при разработке прототипа программной системы «Прокат автомобилей» сводилась к простой проверке на null на уровнях сервиса, контроллера и представления.

Пример такой обработки:

В классе CarService в методе findCarsByClass происходит поиск автомобилей через JPA репозиторий, который возвращает список авто или null в случае, если данные не найдены. Затем результат запроса заносится в Map <String, Object>.

Пример кода:

public Map<String, Object> findCarsByClass(String carClass, int numberPage, int countItemOnPage) {

Map<String, Object> response = new HashMap<> ();

List<Car> cars = carRepository.findByLevel(carClass, PageRequest.of(numberPage, 10)).stream().toList();

response.put("cars", cars);

Затем в классе CarController ответ от сервиса добавляется к представлению.

Пример кода:

@GetMapping("/carbyclass")

public String carByClass(@RequestParam String carClass, @RequestParam(defaultValue = "0") int numberPage, Model model) {

Map<String, Object> response = carService.findCarsByClass(carClass, numberPage, 10);

model.addAttribute("cars", response.get("cars"));

После этого, в представлении по средствам шаблонизатора freemarker происходит вывод результата или сообщения о том, что данные не найдены.

Пример кода:

<#list cars?sort\_by("price") as car>

<@carParts.carCard car=car></@carParts.carCard>

<#else >

<h1 style="display: block; width: 100%; text-align: center">Таких авто нет(</h1>

</#list>

Обработка ошибок валидации происходила на трех уровнях: уровне сервисов, уровне контроллеров и уровне представления.

На уровне сервисов использовался spring-boot-starter-validation, который позволял определять ошибки в сущностях при помощи аннотаций над полями сущностей и объектом BindingResult, содержащим ошибки валидации.

Пример кода аннотации для сущности Car:

@NotBlank(message = "WIN номер авто обязателен!")

@Size(min = 17, max = 17, message = "Размер WIN номера должен составлять 17 символов!")

private String WIN\_Number;

@NotBlank(message = "Укажите бренд автомобиля!")

@Size(min = 1, max = 20, message = "Название производителя авто не может превышать 20 символов!")

private String brand;

Пример кода валидации на уровне сервисов:

public Map<String, Object> addCar(Car car, BindingResult bindingResult, MultipartFile file) throws IOException {

Map<String, Object> response = new HashMap<>();

if (bindingResult.hasErrors()) {

Map<String, List<String>> errorMap = bindingResult.getFieldErrors()

.stream()

.collect(Collectors.groupingBy(

FieldError::getField,

Collectors.mapping(FieldError::getDefaultMessage,Collectors.to List())

));

response.put("ifError", true);

response.put("map", errorMap);

response.put("car", car);

return response;

}

Пример кода валидации на уровне контроллера:

@PreAuthorize("hasAnyAuthority('ADMIN', 'MANAGER')")

@PostMapping("/car")

public String add(@Valid Car car, BindingResult bindingResult,

@RequestParam("newImage") MultipartFile file, Model model) throws IOException {

Map<String, Object> response = carService.addCar(car, bindingResult, file);

if ((boolean) response.get("ifError")) {

model.addAttribute("map", response.get("errorMap"));

return "Car/carCreate";

}

Валидация на уровне представления производилась базовыми средствами HTML, такими как: required, maxlength, minlength, max, min.

Отображение ошибок валидации в представлении выполнялся средствами шаблонизатора freemarker следующим образом:

<#if map??>

<#if map['WIN\_Number']??>

<div class="invalid-feedback">

<#list map['WIN\_Number'] as error>

${error}

</#list>

</div>

</#if>

</#if>

Аналогичным сценарием обрабатывались и остальные ошибки разрабатываемого прототипа программной системы «Прокат автомобилей».

## 5.4 Организация управления доступом

Организация управления доступом осуществлялась по средствам Spring Security. Настройка управления доступом проводилась в классе SecurityConfig.

Фрагмент кода класса:

@Override

protected void configure(HttpSecurity http) throws Exception {

http.authorizeRequests()

.antMatchers("/", "/static/\*\*", "/imageCar/\*\*", "/image/\*\*").permitAll()

.antMatchers( "/login/\*\*", "/about", "/contacts", "/car/\*\*", "/appeal/\*\*", "/car",

"/findCar","/carbyclass","/registration "/error").permitAll()

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.logout()

.logoutSuccessUrl("/")

.permitAll();

//http.csrf().disable();

}

Организация управления доступом на уровне контроллеров осуществлялась с использованием аннотации @PreAuthorize.

Пример использования аннотации:

@PreAuthorize("hasAnyAuthority('ADMIN', 'MANAGER')")

@PostMapping("/car/delete")

public String deleteCar(@RequestParam int id) {…}

Организация управления доступом на уровне представления осуществлялась с помощью шаблонизатора freemarker. Для этого был создан шаблон security.ftl, который получал данные о пользователе, позволяя использовать их в представлении.

Код файла security.ftl:

<#assign

known = Session.SPRING\_SECURITY\_CONTEXT??

>

<#if known>

<#assign

user = Session.SPRING\_SECURITY\_CONTEXT.authentication.principal

name = user.getUsername()

isAdmin = user.getAuthorities()?seq\_contains('ADMIN')

isManager = user.getAuthorities()?seq\_contains('MANAGER')

isUser = user.getAuthorities()?seq\_contains('USER')

>

<#else>

<#assign

name = ""

isAdmin = false

isManager = false

isUser = false

>

</#if>

Затем полученные данные использовались в представлениях для разграничения функционала по ролям.

Пример разграничения функционала:

<#if isAdmin>

<li class="navBar-item">

<a id="/user" href="/user">Пользователи</a>

</li>

<li class="navBar-item">

<a id="/reports" href="/reports/cars">Отчёты</a>

</li>

</#if>

## 5.5 Физическая структура базы данных

В процессе разработки прототипа программной системы «Прокат автомобилей» использовалась СУБД MySQL Workbench. После разработки уровня доступа к данным средствами СУБД была получена ER диаграмма созданной БД.

Отличия данной диаграммы от диаграммы классов-сущностей UML уровня проектирования заключается в следующем:

* Было изменено отношение между таблицами «users» и «roles» с один-ко-многим на многие-ко-многим для поддержки нескольких ролей у пользователя. В результате чего была добавлена промежуточная таблица «users\_role».
* Было принято решение убрать связь между таблицами «users» и «feedbacks» для поддержки анонимных отзывов. Однако в таблице «feedbacks» осталось поле «author», которое будет идентифицировать отзыв с пользователем в случае, если пользователь оставил отзыв не анонимно.

Полученная диаграмма представлена на рисунке 5.3.1.

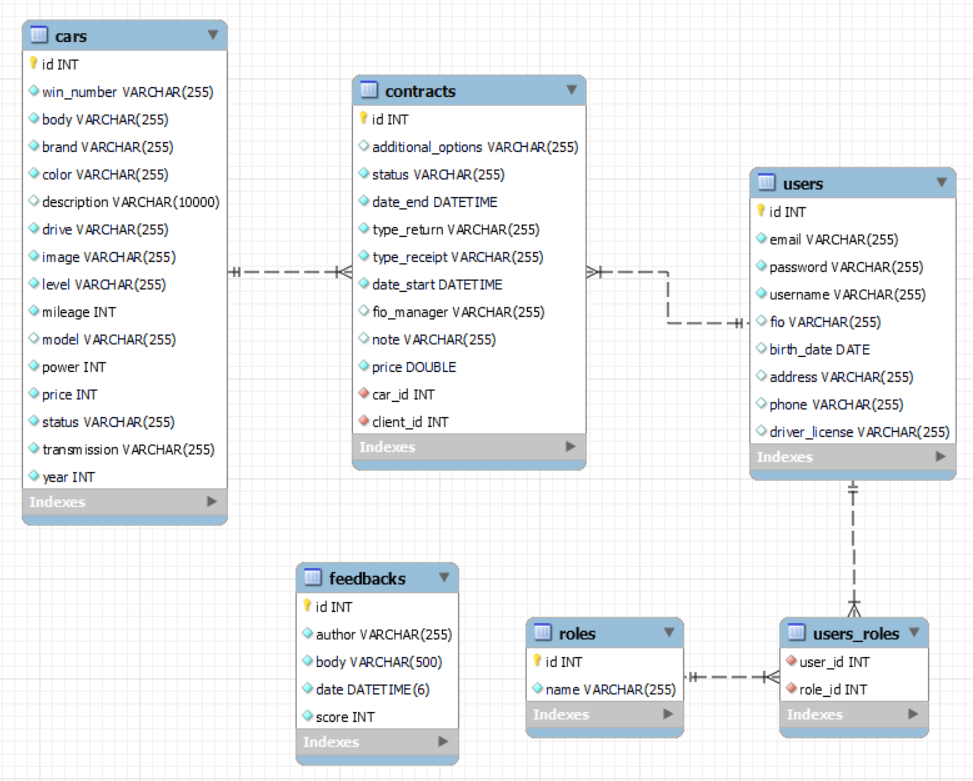


Рисунок 5.3.1 – Физическая структура БД.

## 5.6 Обеспечение целостности данных

## 5.6.1 Ограничение целостности

В данном разделе описываются ограничения и меры, принимаемые для обеспечения целостности данных в системе. Ограничения целостности гарантируют, что данные сохраняются в правильном и согласованном состоянии.

В процессе разработки прототипа программной системы «Прокат автомобилей» ограничения целостности проводилось на нескольких уровнях: базы данных и приложения.

***Ограничения на уровне базы данных.***

Использовались следующие принципы обеспечения ограничения целостности на уровне БД:

1. Уникальность: для обеспечения уникальности данных в базе данных, применены ограничения уникальности на определенные столбцы таблиц. Например, столбец email в таблице users имеет ограничение уникальности, чтобы каждый пользователь имел уникальный адрес электронной почты.
2. Внешние ключи: используются внешние ключи для связи данных между таблицами и обеспечения ссылочной целостности. Например, в таблице contract есть внешний ключ client\_id, который ссылается на идентификатор пользователя в таблице users.

Также для обеспечения целостности данных все таблицы были приведены к третьей нормальной форме.

***Ограничения на уровне приложения.***

1. Валидация данных: при обработке входящих запросов веб-приложение выполняет валидацию данных, чтобы убедиться, что они соответствуют определенным правилам и форматам. Например, проверяется правильность формата электронной почты или ограничения на минимальную и максимальную длину строки. Процесс валидации подробно описан в пункте [4.6](#_4.6_Организация_обработки).
2. Бизнес-правила: В приложении реализованы бизнес-правила, которые определяют допустимые операции и состояния данных. Например, при попытке пользователя арендовать автомобиль проверяется наличие активной аренды у пользователя или соблюдение временных ограничений для оформления аренды.

## 5.6.2 Ограничение распределённых транзакций

В данном разделе описываются ограничения, связанные с использованием распределенных транзакций в системе. Распределенные транзакции используются для координации изменений данных на нескольких распределенных ресурсах или сервисах.

В процессе разработки прототипа программной системы «Прокат автомобилей» были выявлены места, которые могут привести к неправильной работе системы в случае одновременного обращения нескольких пользователей.

Одним из таких мест является функционал аренды автомобилей. Несколько пользователей могу одновременно арендовать один и тот же автомобиль, и тогда создаться две аренды на один автомобиль чего по бизнес-требованиям быть не может.

Решение данной проблемы было выполнено с использованием механизма транзакций, поддерживаемых Spring framework (аннотация @Transactional). Фрагмент кода решения проблемы двойной аренды представлен ниже:

@Transactional(propagation = Propagation.REQUIRES\_NEW, isolation = Isolation.REPEATABLE\_READ)

public synchronized Map<String, Object> createContract(Map<String, String> form) {…}

## 5.7 Реализация взаимодействия с внешним сервисом

В процессе разработки прототипа программной системы «Прокат автомобилей» при реализации функционала создания аренды, возникла необходимость рассчитывать цену доставки автомобиля к клиенту в случае получения автомобиля и от клиента в случае возврата авто.

Для реализации поставленной цели был выбран бесплатный сервис MapQuest предоставляющий 15 000 бесплатных запросов на использование Map API.

Процесс работы с данным API заключался в следующем: пользователь вводит данные доставки авто (или место возврата), затем на стороне клиента выполняется Axios запрос на API, результат запроса обрабатывается (API возвращает расстояние между точками в милях, поэтому приходится переводить его в километры, затем умножать на цену доставки за 1км), обработанные данные показываются пользователю.

Пример работы с API в виде JS функции представлен ниже:

function changePlaceReturn() {

let toPlace = document.getElementById('placeReturnYourselfOption')

.value;

document.getElementById('mapForDeliveryCar').setAttribute("href",

'https://yandex.ru/maps/?rtext=56.141136,40.372616~' + toPlace);

axios.get('https://www.mapquestapi.com/directions/v2/route', {

params: {

key: 'cjlAEQkMSLijqu8uGq6Bci4Xg6R3jGEn',

from: 'Владимир,проспект Строителей,7',

to: toPlace

}

})

.then(response => {

const distance = response.data.route.distance;

const dinstanceFromOffice = Math.ceil(distance \* 1.609344);

const priceDelivery = Math.ceil(dinstanceFromOffice \* 20)

document.getElementById('priceForDelivery').style.display = "block";

if (priceDelivery < 300) {

document.getElementById('priceForDelivery').textContent = "Цена: 300 руб." + "Расстояние: " + dinstanceFromOffice + "km";

//устанавливаем цену доставки в прайс-лист

this.changeDeliveryPriceList("ForReturnToOwnVersion",300);

} else {

document.getElementById('priceForDelivery').textContent = "Цена: " + priceDelivery + "руб. Расстояние: " + dinstanceFromOffice + "km";

//устанавливаем цену доставки в прайс-лист

this.changeDeliveryPriceList("ForReturnToOwnVersion", priceDelivery);

}

calculateDeliveryPrice();

})

.catch(error => {

console.log(error);

});

}

В результате реализации компонентов системы "Прокат автомобилей" было успешно разработано веб-приложение, которое обеспечивает удобный интерфейс для клиентов и эффективное управление автомобильным парком для администраторов. Были разработаны и интегрированы различные компоненты, такие как контроллеры и сервисы, для реализации бизнес-логики системы.

В результате применения выбранной архитектуры и инструментов, система "Прокат автомобилей" обладает модульной структурой, позволяющей легко масштабировать и расширять функциональность системы. Оптимизация взаимодействия между клиентской и серверной частями системы позволяет достичь эффективного управления автомобильным парком, сокращения времени выполнения операций и повышения удобства использования для клиентов.

# 6 Инструкция по установке приложения

Для получения полнофункционального приложения с тестовыми данными необходимо выполнить следующие действия:

*Шаг 1: Загрузка исходного кода*

1. Перейдите на публичный репозиторий приложения на GitHub по ссылке: <https://github.com/sixxG/CarRental-.git> и найдите ссылку для загрузки исходного кода (обычно в виде кнопки "Code").
2. Щелкните на ссылке загрузки исходного кода, чтобы скачать его на ваш компьютер. Распакуйте архив.

*Шаг 2: Установка и настройка MySQL*

1. Установите MySQL на свою систему (если ещё не установлено), следуя официальной документации MySQL: <https://dev.mysql.com/doc/> .
2. Создайте базу данных для вашего приложения. Название БД должно быть таким же, как и в свойстве: spring.datasource.url.

*Шаг 3: Настройка проекта в среде разработки*

1. Откройте вашу среду разработки (например, IntelliJ IDEA, Eclipse или другую) и импортируйте загруженный исходный код проекта в вашу среду.
2. Убедитесь, что на вашем ПК установлен Maven для сборки проекта (если нет, то установите его по ссылке: <https://maven.apache.org/download.cgi>).
3. Внесите следующие изменения в файл настроек application.properties:

* spring.datasource.username=ИМЯ\_ПОЛЬЗОВАТЕЛЯ\_БД
* spring.datasource.password=ПАРОЛЬ\_ОТ\_БД
* upload.path=ПУТЬ\_НА\_ВАШЕМ\_ПК\_ДЛЯ\_СОХРАНЕНИЯ\_ИЗОБРАЖЕНИЙ

*Шаг 4: Сборка и запуск приложения*

1. В вашей среде разработки выполните сборку проекта, чтобы получить исполняемый JAR-файл. В большинстве сред разработки это можно сделать с помощью команды "Build" или "Compile" (Также можно через командную консоль, выполнив команду “mvn clean install” в папке проекта).
2. После успешной сборки проекта найдите полученный JAR-файл. Обычно он находится в папке target проекта.
3. Перейдите к папке, содержащей JAR-файл, в командной строке или терминале.
4. Запустите приложение, выполнив команду java -jar <имя\_файла.jar> для JAR-файла.
5. После запуска приложения оно будет доступно по адресу [http://localhost:8079](http://localhost:8079%20) (если используется встроенный Tomcat). Убедитесь, что сервер Apache Tomcat работает и принимает запросы.

Поздравляю! Теперь у вас должно быть полнофункциональное приложение на Spring Boot, использующее MySQL в качестве базы данных и работающее на сервере Apache Tomcat v8.6. Вы можете продолжить разработку, улучшение и тестирование данного приложения согласно вашим потребностям.



# 7 оРГАНИЗАЦИЯ УПРАВЛЕНИЯ ПРОЕКТОМ

## 7.1 Распределение ролей в команде/зоны ответственности

Работа будет выполняться одним человеком, отвечающим за все этапы разработки и ответственным за все зоны.

## 7.2 Календарный план работ

Был создан календарный план работ в виде диаграммы Ганта с указанием задач и временным периодом их выполнения. Календарный план представлен на рисунке 7.1.

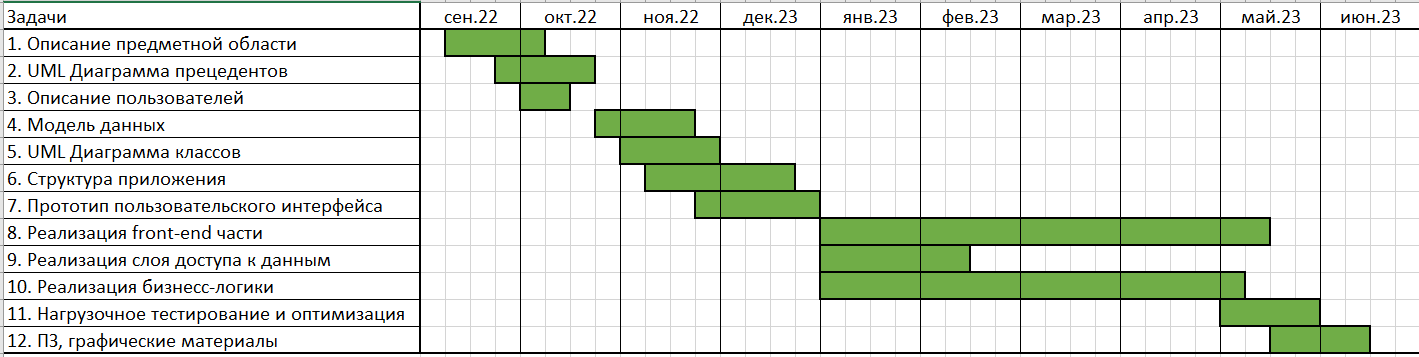


Рисунок 7.1 – Календарный план проекта.

В ходе работы по организации управления проектом были выполнены следующие задачи: определение целей и требований проекта, планирование и распределение ресурсов, установление ролей и ответственностей в команде, разработка графика работ и мониторинг прогресса проекта.

Основными результатами организации управления проектом стали:

* Определение конкретных целей и требований проекта, что способствует ясному пониманию его области и ориентации команды разработчиков.
* Разработка подробного плана проекта, включающего этапы работ, сроки и ресурсы, что обеспечивает структурированность и системность в выполнении проекта.
* Назначение ролей и ответственностей в команде, что способствует эффективному распределению задач и координации работы.
* Создание графика работ и мониторинг прогресса проекта, что позволяет отслеживать выполнение задач в срок и принимать меры по управлению рисками и изменениями.

В результате организации управления проектом достигнута более структурированная и организованная работа команды разработчиков, что способствует повышению эффективности и качества работы. Контроль прогресса проекта и своевременное реагирование на возникающие проблемы позволяют минимизировать риски и обеспечить успешную реализацию проекта.

# 8 НАГРУЗОЧНОЕ ТЕСТИРОВАНИЕ

Нагрузочное тестирование проводилось с целью проверки соответствия разработанного приложения установленным при проектировании системы нефункциональным требованиям [2.5](#_2.5_Нефункциональные_требования).

Основная задача заключалась в проверки работы приложения одновременно с множеством пользователей (от 100 до 250).

Нагрузочное тестирование проводилось по следующему плану:

1. Тестирование регистрации и авторизации;
2. Тестирование функционала неавторизованного пользователя;
3. Тестирование функционала авторизованного пользователя;
4. Тестирование функционала менеджера;
5. Тестирование функционала администратора.

Нагрузочное тестирование разработанного приложения производилось с использованием инструмента JMeter v5.5.

## 8.1 Тестирование регистрации и авторизации

Данный тест план соответствует следующему сценарию:

1. Авторизация пользователя.

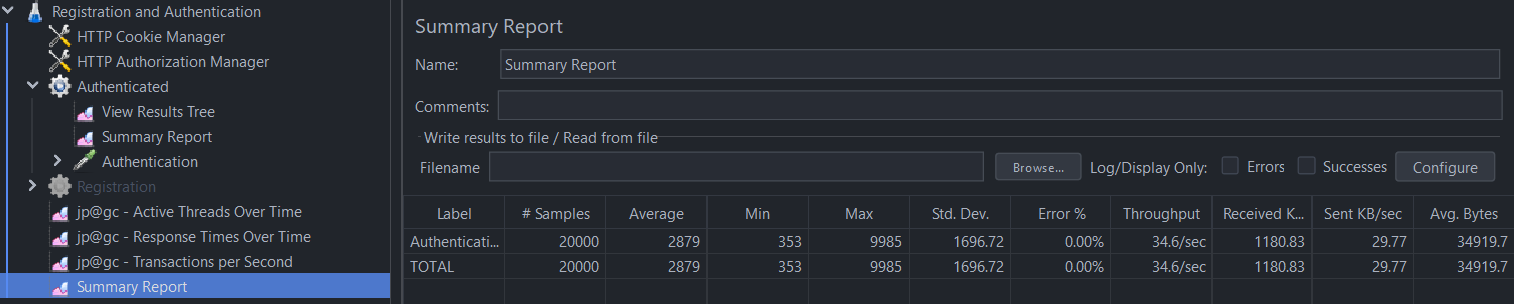
Нагрузочное тестирование было проведено со следующими параметрами:

• Количество пользователей: 200;

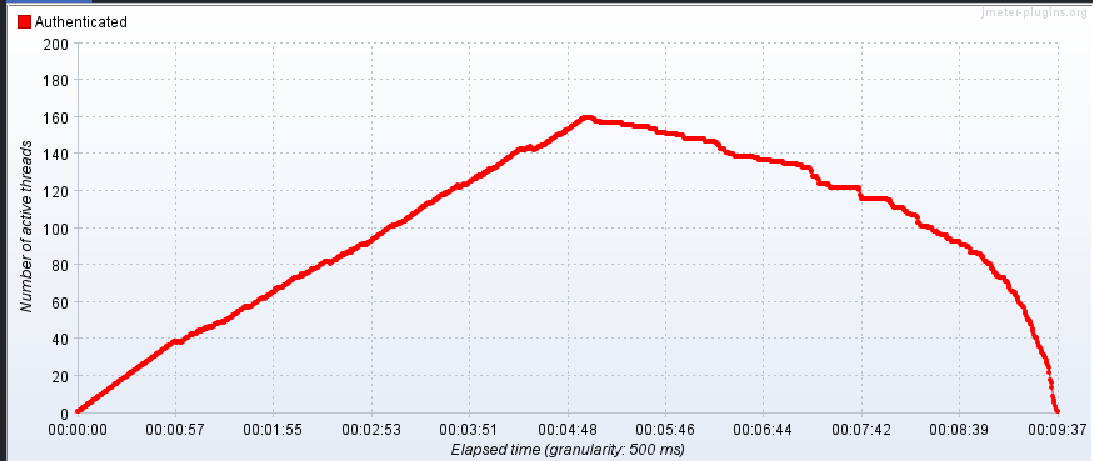
• Период наращивания: 5 мин;

• Количество циклов: 100.

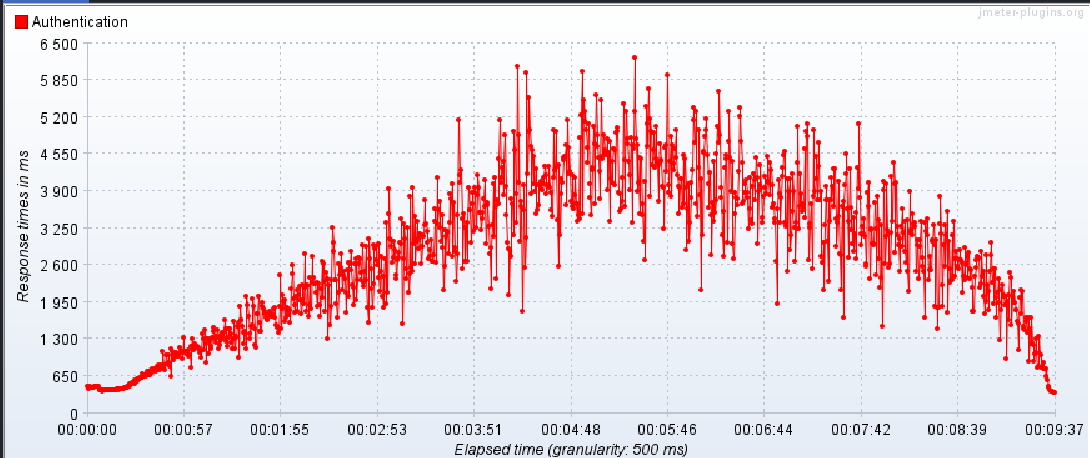
Результаты тестирования представлены на рисунке 8.1.



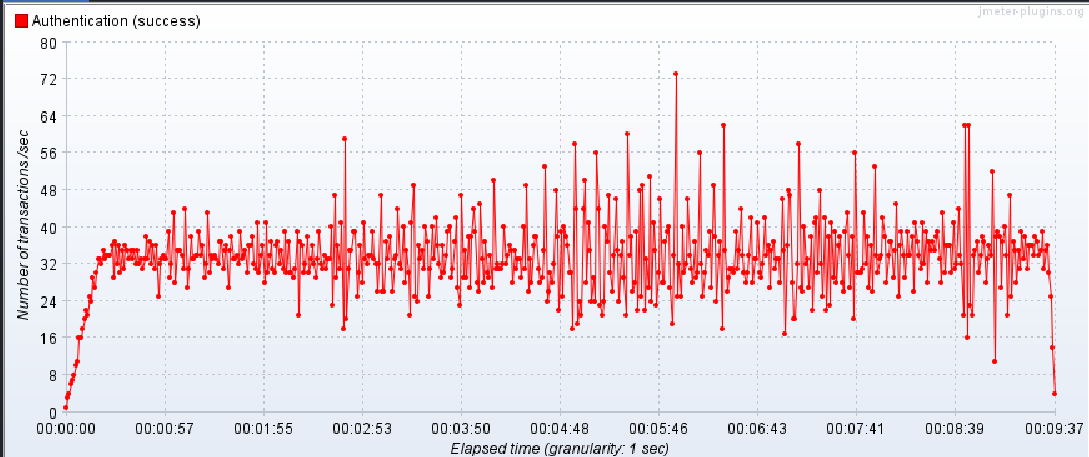
а



б



в



г

Рисунок 8.1 – Результаты нагрузочного тестирования регистрации и авторизации. Название: *а – краткий отчёт по тестированию; б – активные потоки с течением времени; в – времена ответов сервера; г – кол-во транзакции в секунду.*

## 8.2 Тестирование функционала неавторизованного пользователя

Данный тест план соответствует следующему сценарию действий неавторизованного пользователя:

1. Посещение главной страницы приложения;
2. Посещение страницы с информацией о компании;
3. Посещение страницы с контактами;
4. Посещение страницы с каталогом автомобилей;
5. Посещение страницы с отзывами;
6. Посещение страницы с автомобилями по классам;
7. Поиск автомобилей по цене;
8. Поиск автомобилей по марке;
9. Поиск автомобилей по марке и трансмиссии;
10. Посещение страницы с детальной информацией об автомобиле.

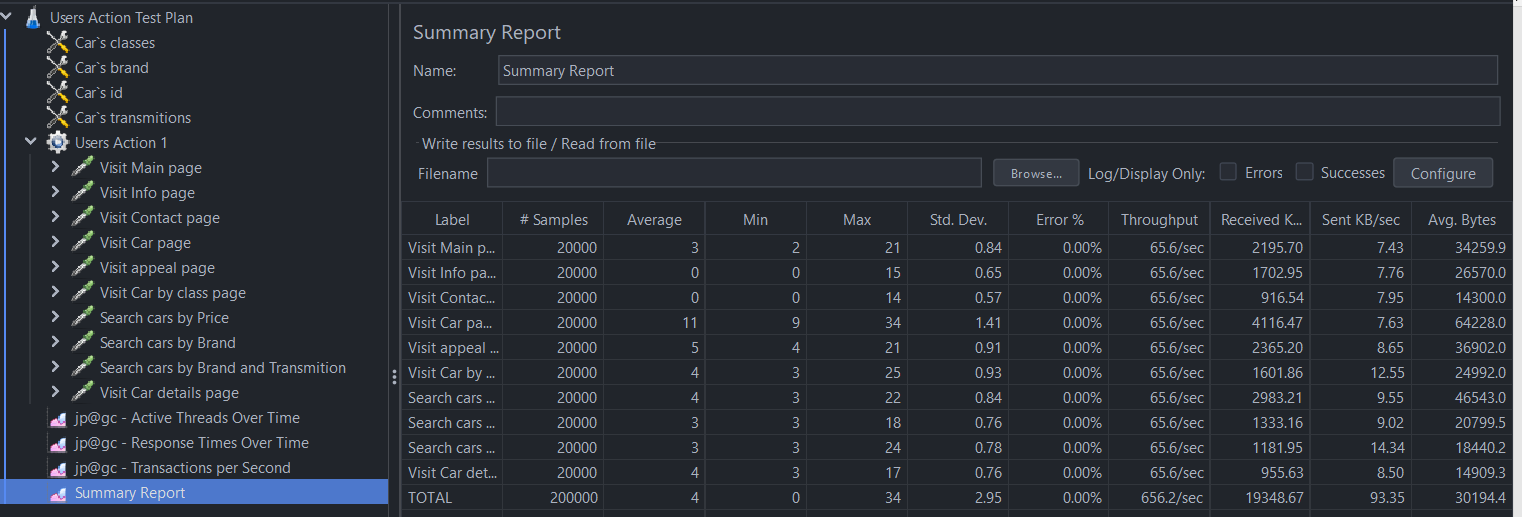
Нагрузочное тестирование было проведено со следующими параметрами:

• Количество пользователей: 200;

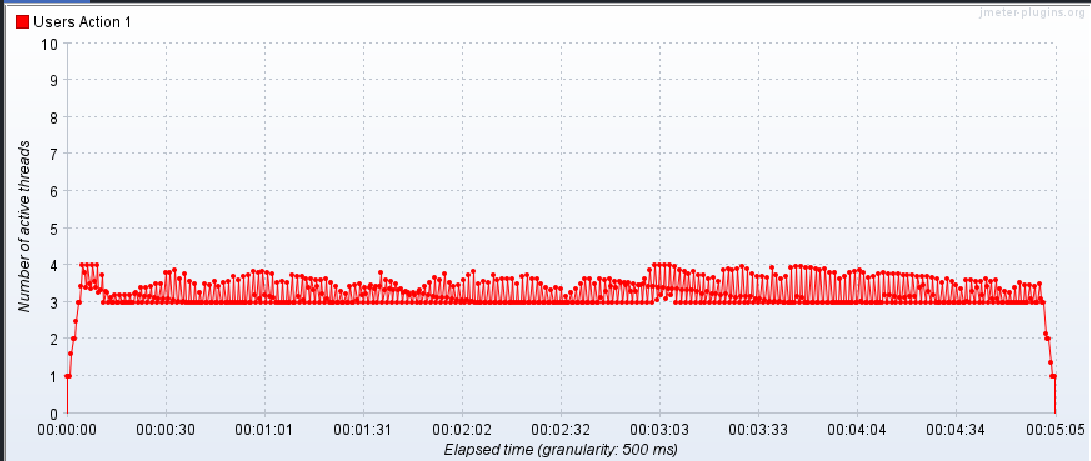
• Период наращивания: 5 мин;

• Количество кругов: 100.

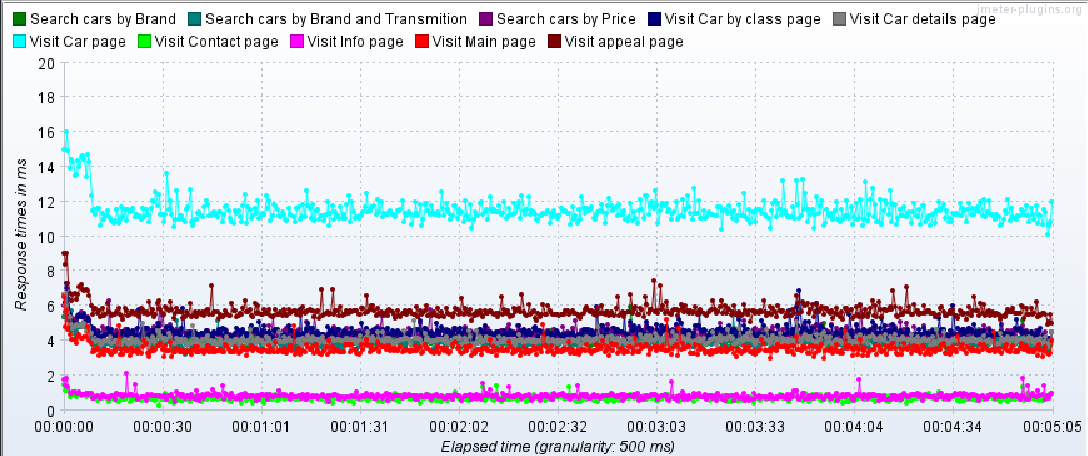
Результаты тестирования представлены на рисунке 8.2.



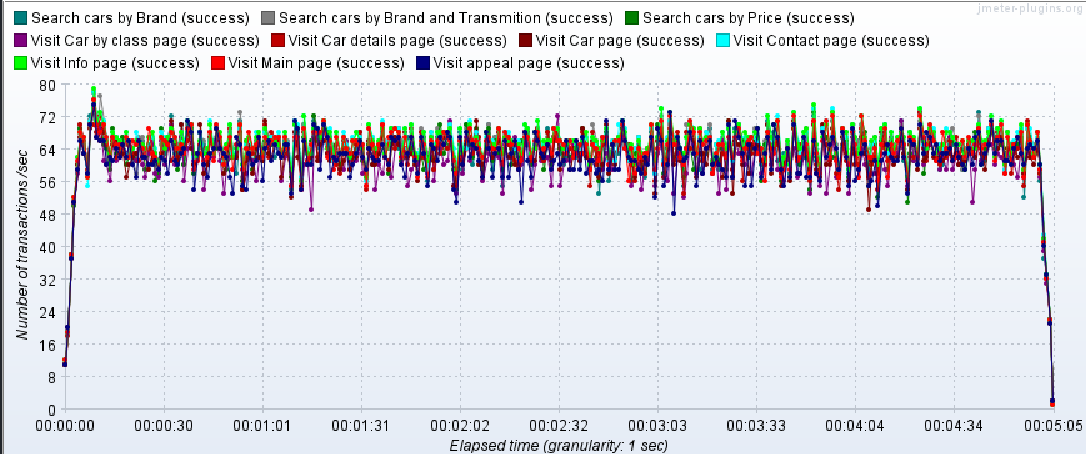
а



б



в



г

Рисунок 8.2 – Результаты нагрузочного тестирования функционала неавторизованного пользователя. Название: *а – краткий отчёт по тестированию; б – активные потоки с течением времени; в – времена ответов сервера; г – кол-во транзакции в секунду.*

## 8.3 Тестирование функционала авторизованного пользователя

Данный тест план соответствует следующему сценарию действий авторизованного пользователя:

1. Авторизация;
2. Посещение страницы с аккаунтом пользователя;
3. Аренда автомобиля;
4. Посещение страницы с арендами пользователя;
5. Посещение страницы с подробной информацией об аренде;
6. Отмена аренды;
7. Добавление отзыва о компании;
8. Посещение страницы с отзывами;
9. Изменение личных данных пользователя.

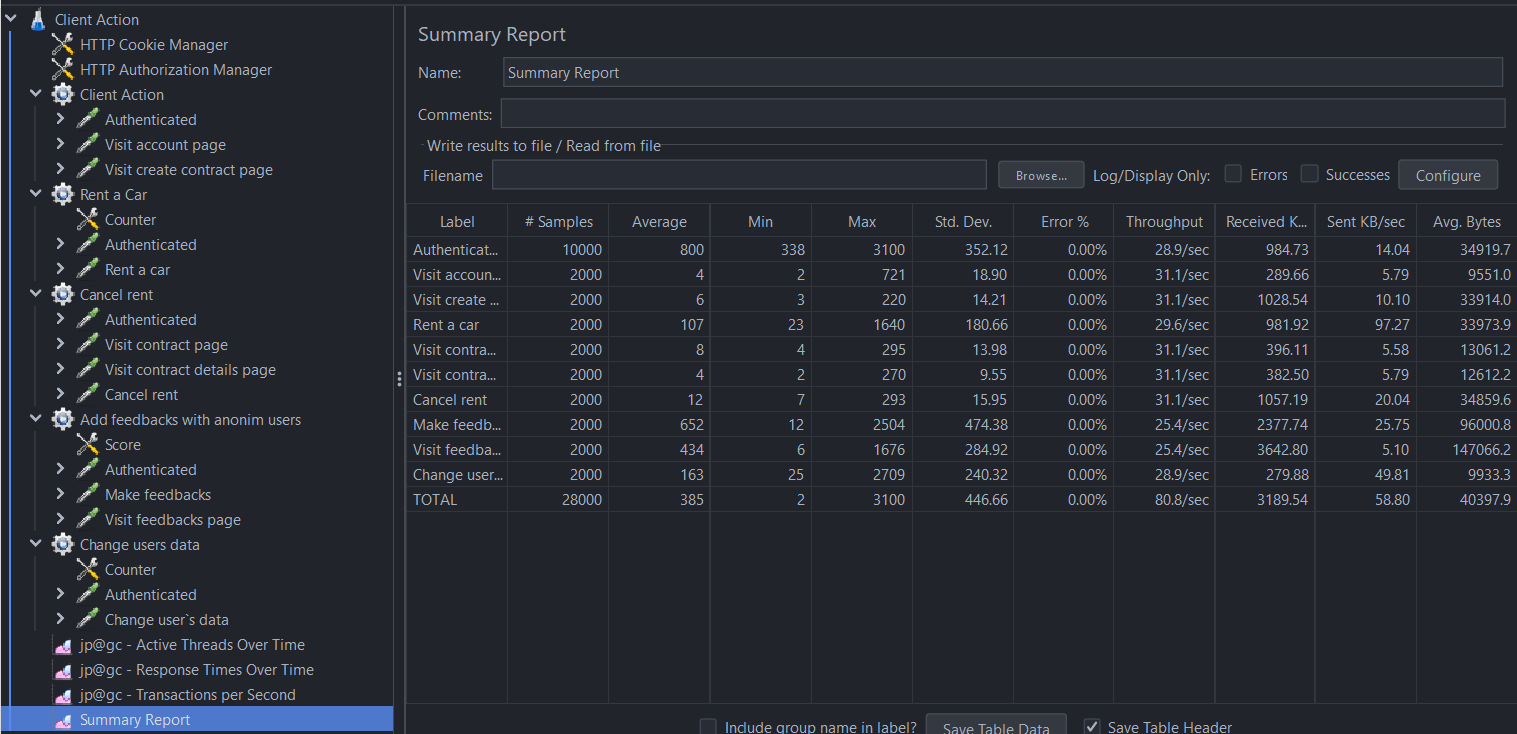
Нагрузочное тестирование было проведено со следующими параметрами:

• Количество пользователей: 200;

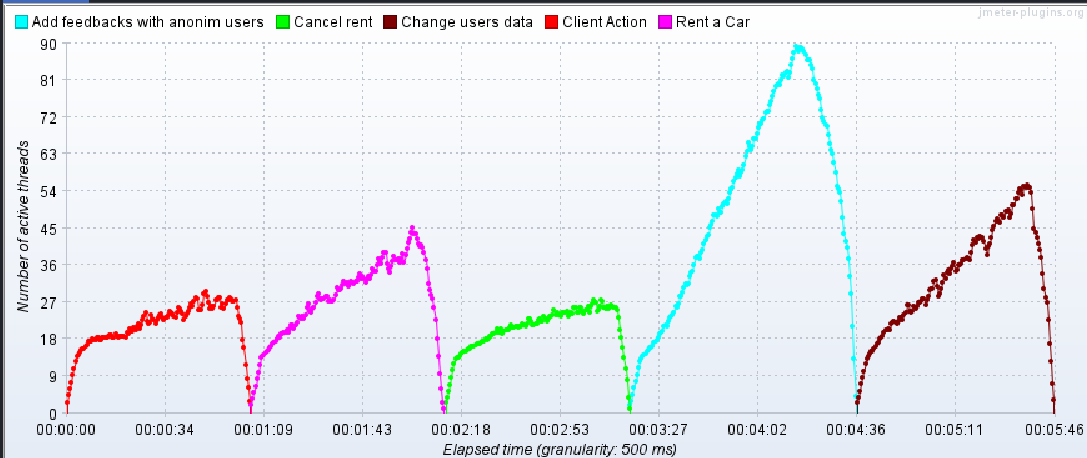
• Период наращивания: 60 сек;

• Количество кругов: 10.

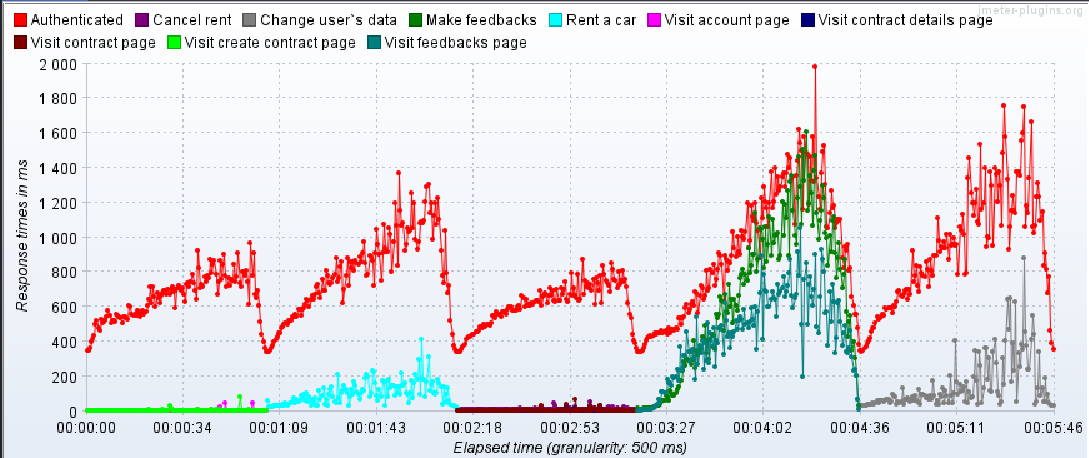
Результаты тестирования представлены на рисунке 8.3.



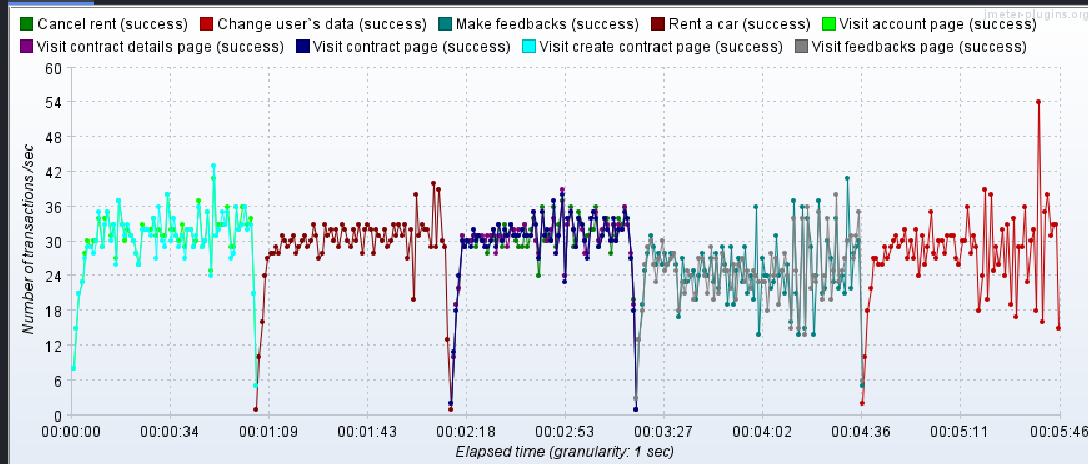
а



б



в



г

Рисунок 8.3 – Результаты нагрузочного тестирования функционала авторизованного пользователя. Название: *а – краткий отчёт по тестированию; б – активные потоки с течением времени; в – времена ответов сервера; г – кол-во транзакции в секунду.*

## 8.4 Тестирование функционала менеджера

Данный тест план соответствует следующему сценарию действий менеджера:

1. Авторизация;
2. Посещение страницы со списком аренд;
3. Посещение страницы со списком аренд по статусу;
4. Поиск аренд по автомобилям;
5. Поиск аренд за период;
6. Добавление автомобиля;
7. Поиск добавленного авто;
8. Изменение автомобиля;
9. Поиск добавленного авто;
10. Удаление автомобиля;
11. Изменение аренды.

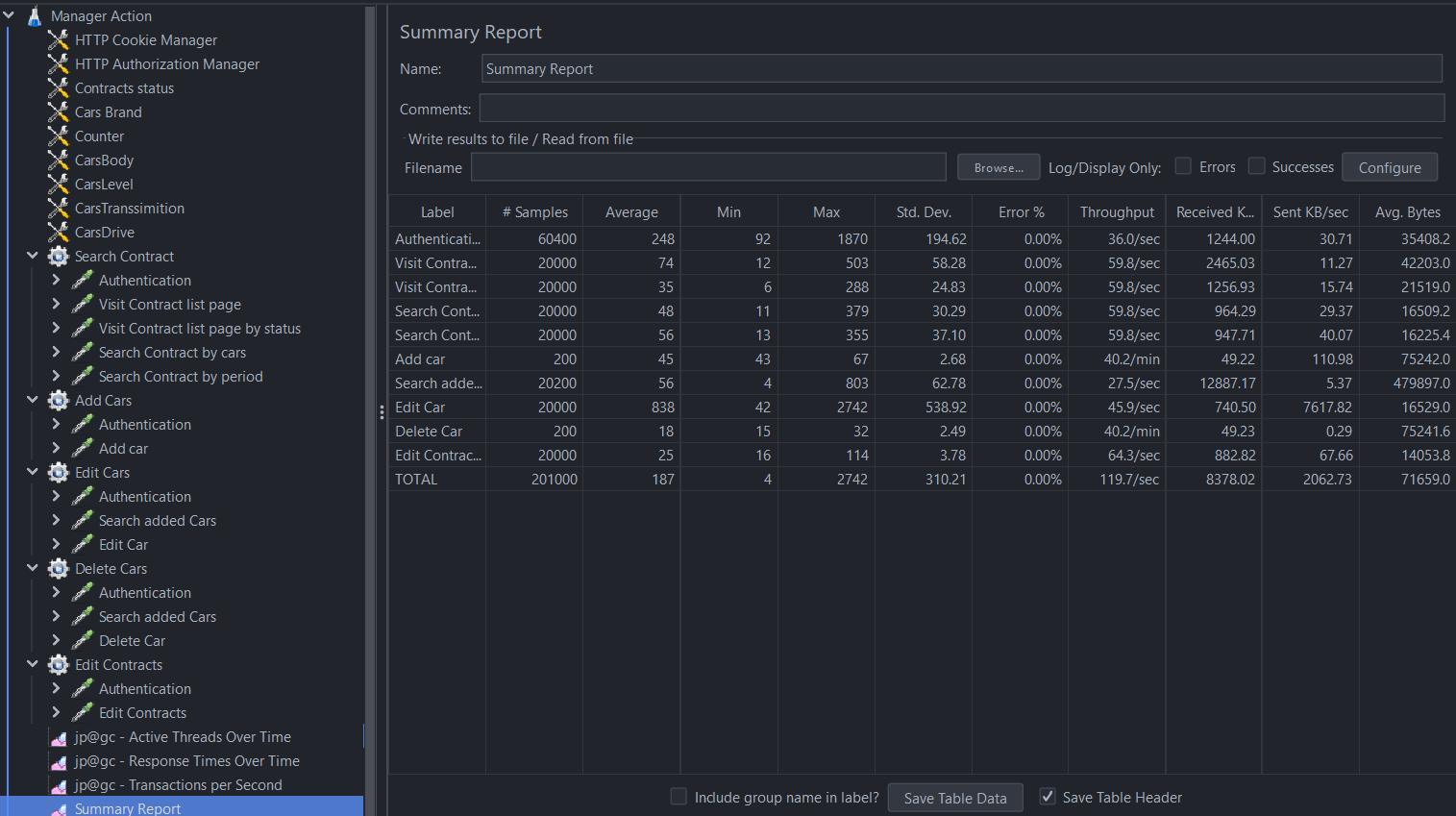
Нагрузочное тестирование было проведено со следующими параметрами:

• Количество пользователей: 200;

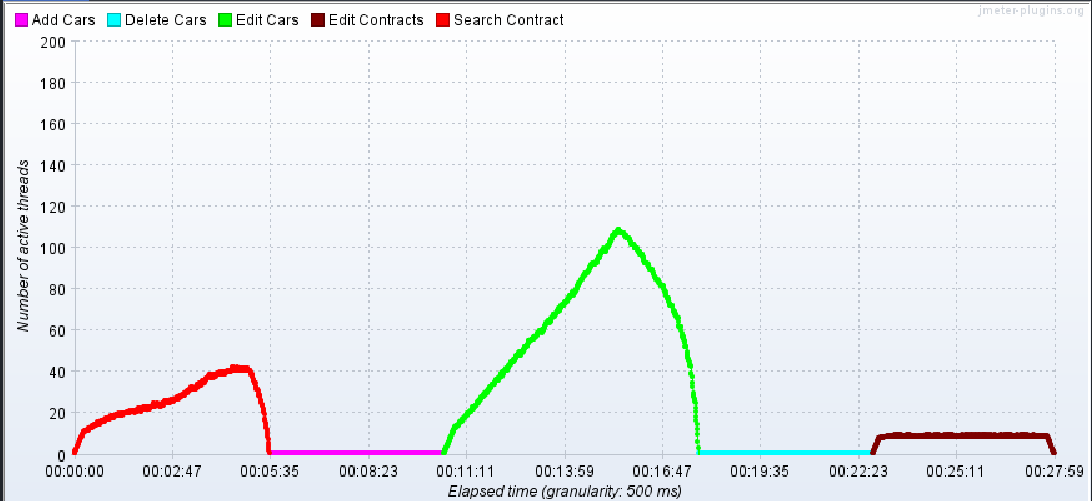
• Период наращивания: 5 мин;

• Количество кругов: 100.

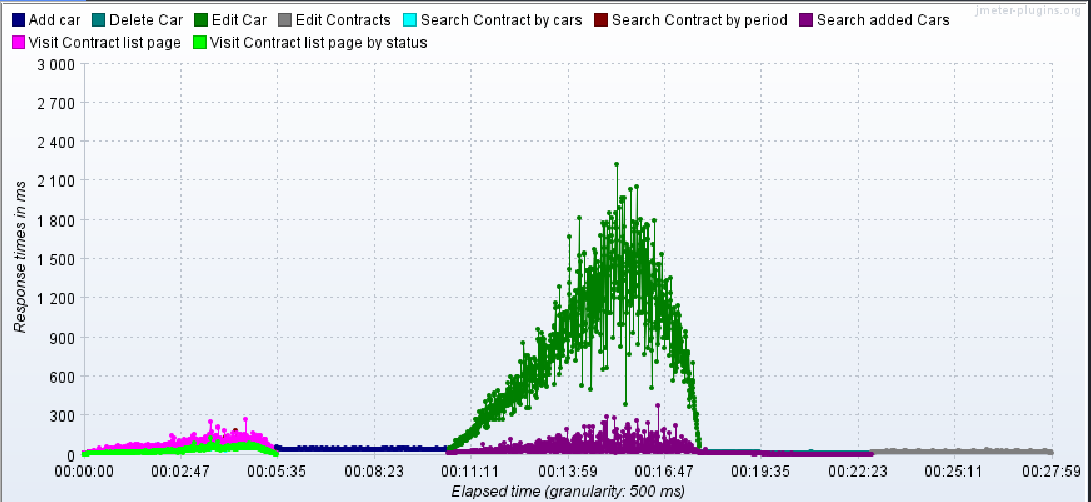
Результаты тестирования представлены на рисунке 8.4.



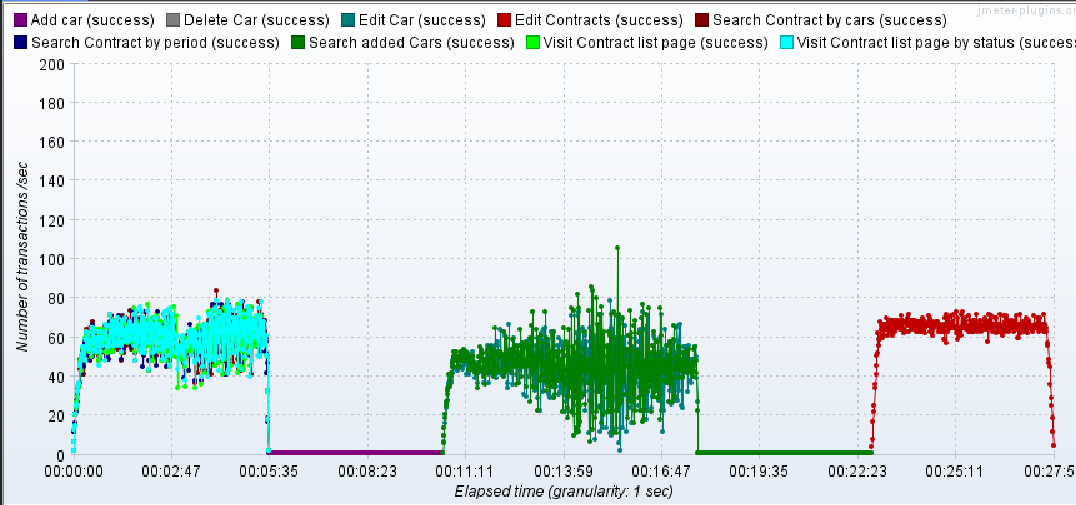
а



б



в



г

Рисунок 8.4 – Результаты нагрузочного тестирования функционала менеджера. Название: *а – краткий отчёт по тестированию; б – активные потоки с течением времени; в – времена ответов сервера; г – кол-во транзакции в секунду.*

## 8.5 Тестирование функционала администратора

Данный тест план соответствует следующему сценарию действий администратора:

1. Авторизация;
2. Посещение страницы с отчётами по автомобилям;
3. Посещение страницы с отчётами по арендам;
4. Посещение страницы с отчётами по пользователям;
5. Посещение страницы со списком пользователей;
6. Изменение информации о пользователе;

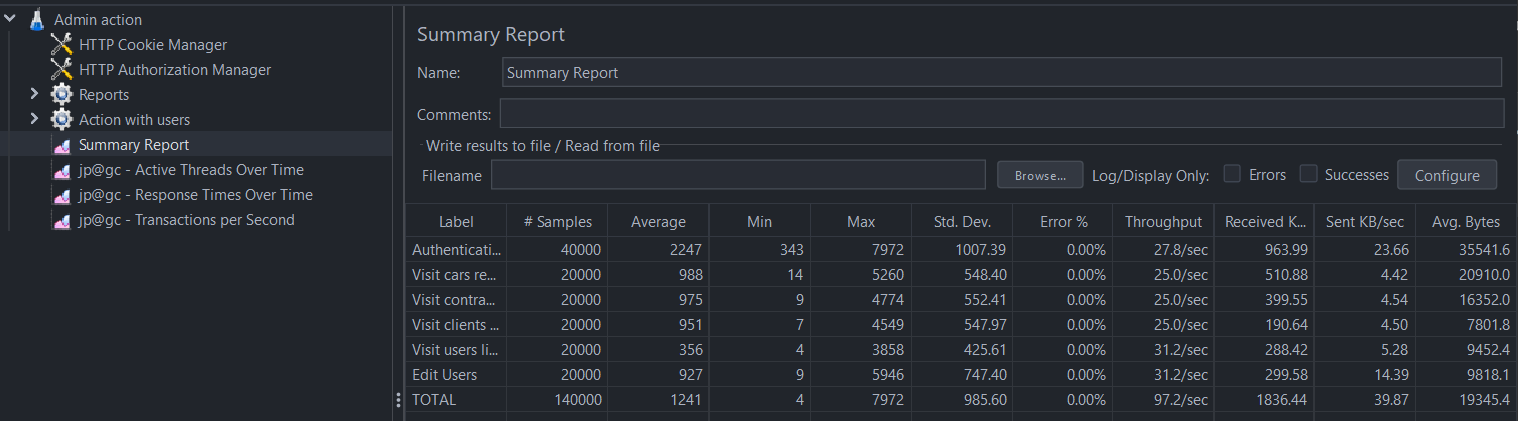
Нагрузочное тестирование было проведено со следующими параметрами:

• Количество пользователей: 200;

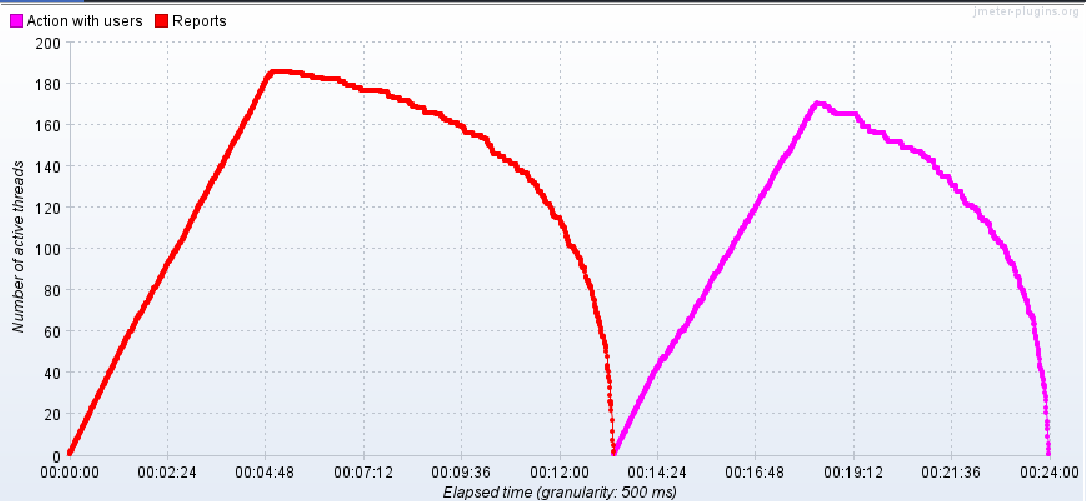
• Период наращивания: 5 мин

• Количество кругов: 100.

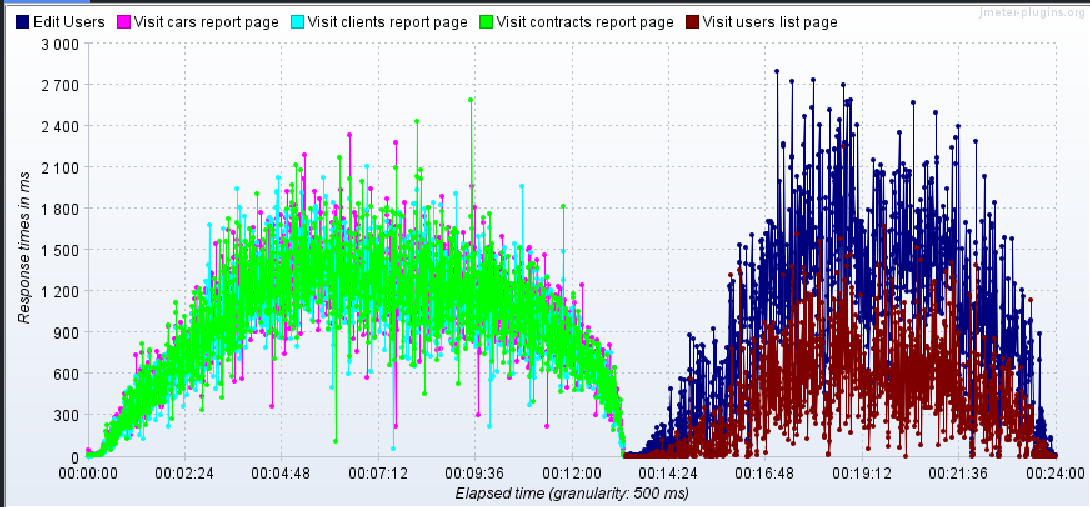
Результаты тестирования представлены на рисунке 8.5.



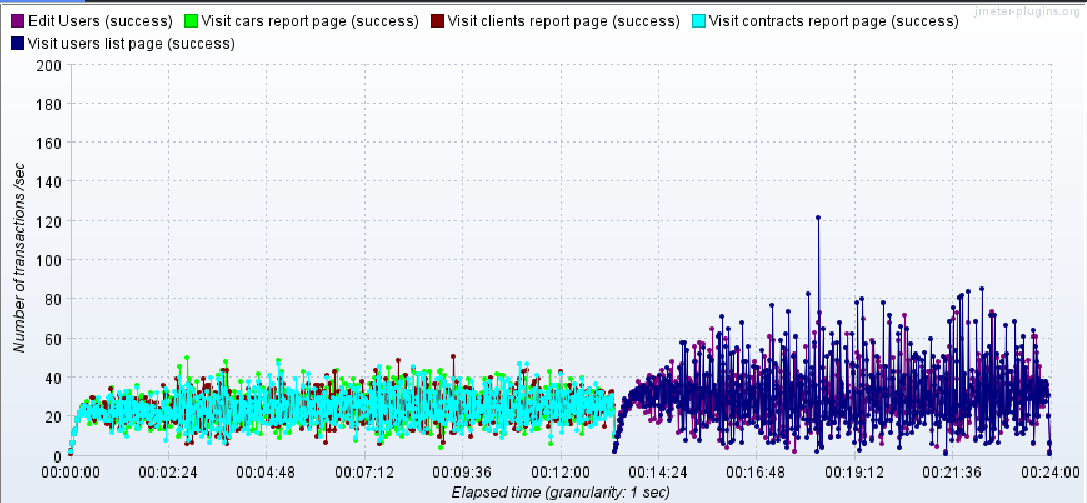
а



б



в



г

Рисунок 21 – Результаты нагрузочного тестирования функционала менеджера. Название: *а – краткий отчёт по тестированию; б – активные потоки с течением времени; в – времена ответов сервера; г – кол-во транзакции в секунду.*

## 8.6 Анализ результатов нагрузочного тестирования

После проведения нагрузочного тестирования разного функционала для различных ролей разработанного приложения результаты тестирования показали, что приложение способно спокойно справляться с нагрузкой от 100 до 250 пользователей.

Среднее время отклика системы не превышает 5 сек. Пропускная способность колеблется от 25/сек до 100/сек.

Эти результаты удовлетворяют нефункциональным требованием, установленным при проектировании системы.

# 9 заключение

В рамках данного проекта было разработано программное обеспечение "Прокат автомобилей" на платформе Java с использованием Spring Boot Framework. Целью данного проекта было создание прототипа системы, которая позволяет управлять процессом проката автомобилей, обеспечивая удобство использования для клиентов и эффективность работы для администраторов.

В процессе разработки были определены требования к системе, проведен анализ существующих аналогов и выбраны технологии, наиболее подходящие для реализации поставленных задач. Были применены современные подходы и принципы разработки, такие как модульность, масштабируемость и безопасность, что позволило создать гибкую и надежную систему.

В ходе разработки были реализованы основные функциональные возможности системы, включая регистрацию пользователей, просмотр доступных автомобилей, оформление и отслеживание заказов, а также административные функции, такие как управление автомобилями и клиентами.

Полученный прототип системы был протестирован, и в ходе тестирования выявлены и исправлены некоторые ошибки и недочеты. Были проведены тесты на производительность, которые показали хорошую отзывчивость системы даже при большом количестве одновременных запросов.

В заключение можно сказать, что разработанная система "Прокат автомобилей" полностью соответствует поставленным целям и требованиям. Она предоставляет удобный интерфейс для клиентов, эффективное управление прокатом автомобилей и администрирование системы. Прототип системы готов к дальнейшей разработке и расширению функциональности, и может служить основой для создания полноценного коммерческого приложения.

10 СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Learn Spring Boot [Электронный ресурс]: www.baeldung.com. – Режим доступа: <https://www.baeldung.com/spring-boot/>

2. Spring Framework - официальная документация [Электронный ресурс]: docs.spring.io. – Режим доступа <https://docs.spring.io/spring-framework/docs/current/reference/html/index.html>

3. JDK 18 Documentation [Электронный ресурс]: docs.oracle.com. – Режим доступа: <https://docs.oracle.com/en/java/javase/18/index.html>

4. Bootstrap Documentation [Электронный ресурс]: Bootstrap.com. – Режим доступа: <https://getbootstrap.com/docs/4.0/>

5. MapQuest Documentation [Электронный ресурс]: developer.mapquest.com. – Режим доступа: <https://developer.mapquest.com/documentation>

ПРИЛОЖЕНИЕ А Листинг компонентов системы

Весь код проекта представлен в GIT репозитории по ссылке: <https://github.com/sixxG/CarRental-.git>.

Код сервиса CarService:

package com.example.spring.security.services;  
  
import com.example.spring.security.models.Car;  
import com.example.spring.security.models.Contract;  
import com.example.spring.security.models.ContractCondition;  
import com.example.spring.security.models.User;  
import com.example.spring.security.repositories.CarRepository;  
import com.example.spring.security.repositories.ContractRepository;  
import com.example.spring.security.repositories.UserRepository;  
import org.springframework.beans.factory.annotation.Value;  
import org.springframework.data.domain.PageRequest;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Isolation;  
import org.springframework.transaction.annotation.Propagation;  
import org.springframework.transaction.annotation.Transactional;  
import org.springframework.validation.BindingResult;  
import org.springframework.validation.FieldError;  
import org.springframework.web.multipart.MultipartFile;  
  
import java.io.File;  
import java.io.IOException;  
import java.security.Principal;  
import java.util.\*;  
import java.util.stream.Collectors;  
  
@Service  
public class CarService {  
  
 @Value("${upload.path}")  
 private String uploadPath;  
 private final CarRepository carRepository;  
 private final ContractRepository contractRepository;  
 private final UserRepository userRepository;  
  
 public CarService(CarRepository carRepository, ContractRepository contractRepository, UserRepository userRepository) {  
 this.carRepository = carRepository;  
 this.contractRepository = contractRepository;  
 this.userRepository = userRepository;  
 }  
  
 public long count() {  
 return carRepository.count();  
 }  
 public int countByStatus(String status) {  
 return carRepository.countByStatus(status);  
 }  
 public List<Object[]> countByCarTransmission() {  
 return carRepository.countByCarTransmission();  
 }  
 public List<Object[]> countByCarDrive() {  
 return carRepository.countByCarDrive();  
 }  
 public List<Object[]> countByCarBody() {  
 return carRepository.countByCarBody();  
 }  
 public List<Object[]> countByCarLevel() {  
 return carRepository.countByCarLevel();  
 }  
 public int getAveragePrice() {  
 return (int) Math.*ceil*(carRepository.getAveragePrice());  
 }  
 public int getAverageYear() {  
 return carRepository.getAverageYear();  
 }  
 public int getAverageMileage() {  
 return carRepository.getAverageMileage();  
 }  
 @Transactional  
 public boolean saveCar(Car car, MultipartFile file) throws IOException {  
  
 if (file != null && !file.getOriginalFilename().isEmpty()) {  
 File uploadDir = new File(uploadPath);  
  
 if (!uploadDir.exists()) {  
 uploadDir.mkdir();  
 }  
  
 String uuidFile = UUID.*randomUUID*().toString();  
 String resultFileName = uuidFile + "." + file.getOriginalFilename();  
  
 file.transferTo(new File(uploadPath + "/" + resultFileName));  
  
 car.setImage(resultFileName);  
 }  
  
 carRepository.save(car);  
  
 if (carRepository.findById(car.getId()) == null) {  
 return false;  
 }  
 return true;  
 }  
 @Transactional  
 public void saveCar(Car car) {  
 carRepository.save(car);  
 }  
 public List<Object[]> findNotRentalCars() {  
 return carRepository.findNotRentalCars();  
 }  
 public Car findById(int id) {  
 return carRepository.findById(id);  
 }  
 public Car findOldestCar() {  
 return carRepository.findFirstByOrderByYearAsc();  
 }  
 public Car findYoungestCar() {  
 return carRepository.findFirstByOrderByYearDesc();  
 }  
 public Car findCarByMaxPrice() {  
 return carRepository.findFirstByOrderByPriceDesc();  
 }  
 public Car findCarByMinPrice() {  
 return carRepository.findFirstByOrderByPriceAsc();  
 }  
 public Car findCarByMaxMileage() {  
 return carRepository.findFirstByOrderByMileageDesc();  
 }  
 public Car findCarByMinMileage() {  
 return carRepository.findFirstByOrderByMileageAsc();  
 }  
  
 public List<Car> findAll() {  
 return carRepository.findAll();  
 }  
  
 public Map<String, Object> getCarList(int numberPage, int countItemOnPage) {  
 Map<String, Object> response = new HashMap<>();  
 List<Car> cars = carRepository.findAll(PageRequest.*of*(numberPage, countItemOnPage)).stream().toList();  
 Set<String> brandList = carRepository.getCarsBrand();  
  
 response.put("countPage", Math.*ceil*((double) carRepository.count()/countItemOnPage));  
 response.put("cars", cars);  
 response.put("carsBrand", brandList);  
  
 return response;  
 }  
 public Map<String, Object> getCarListByStatus(String status) {  
 Map<String, Object> response = new HashMap<>();  
 List<Car> cars = carRepository.findAllByStatus(status);  
 Set<String> brandList = carRepository.getCarsBrand();  
  
 response.put("cars", cars);  
 response.put("carsBrand", brandList);  
  
 return response;  
 }  
  
 public Map<String, Object> getCarsByClasses() {  
 Map<String, Object> response = new HashMap<>();  
  
 response.put("carsBrand", carRepository.getCarsBrand());  
  
 response.put("EconomyCar", carRepository.findByLevel("Эконом",PageRequest.*of*(0, 3)).stream().toList());  
 response.put("ComfortCar", carRepository.findByLevel("Комфорт",PageRequest.*of*(0, 3)).stream().toList());  
 response.put("BusinessCar", carRepository.findByLevel("Бизнес",PageRequest.*of*(0, 3)).stream().toList());  
 response.put("PremiumCar", carRepository.findByLevel("Premium",PageRequest.*of*(0, 3)).stream().toList());  
 response.put("SuvCar", carRepository.findByLevel("Внедорожники",PageRequest.*of*(0, 3)).stream().toList());  
 response.put("BusCar", carRepository.findByLevel("Минивэны",PageRequest.*of*(0, 3)).stream().toList());  
 response.put("UniqueCar", carRepository.findByLevel("Уникальные авто",PageRequest.*of*(0, 3)).stream().toList());  
  
 return response;  
 }  
  
 public Map<String, Object> getDataForDeleteCarPage(int id) {  
 Map<String, Object> response = new HashMap<>();  
 Car car = carRepository.findById(id);  
 List<Contract> activeContract = contractRepository.findByCar(car).stream()  
 .filter(contract1 -> !contract1.getStatus().equals(ContractCondition.*COMPLETED*.toString())  
 && !contract1.getStatus().equals(ContractCondition.*CANCELED*.toString())  
 && !contract1.getStatus().equals(ContractCondition.*AWAITING\_PAYMENT\_FINE*.toString())).toList();  
  
 response.put("car", car);  
 if (activeContract.size() != 0) {  
 response.put("isHasActiveContract", true);  
 response.put("contractId", activeContract.get(0).getId());  
 }  
 else {  
 response.put("isHasActiveContract", false);  
 }  
  
 return response;  
 }  
 public Map<String, Object> getDataForDetailsCarPage(int id, Principal user) {  
 Map<String, Object> response = new HashMap<>();  
 if (user != null) {  
 User client = userRepository.findByUsername(user.getName());  
  
 List<Contract> activeContract = contractRepository.findByUser(client).stream()  
 .filter(contract1 -> !contract1.getStatus().equals(ContractCondition.*COMPLETED*.toString())  
 && !contract1.getStatus().equals(ContractCondition.*CANCELED*.toString())).toList();  
  
 response.put("isHasActiveContract", (activeContract.size() != 0));  
 }  
  
 Contract contract = contractRepository.findByCarAndStatusNotAndStatusNot(carRepository.findById(id), "Завершён", "Отменён");  
  
 response.put("car", findById(id));  
 response.put("activeContract", contract);  
  
 if (contract != null) {  
 response.put("carRentingForDate", contract.getDateEnd());  
 }  
  
 return response;  
 }  
  
 @Transactional(propagation = Propagation.*REQUIRES\_NEW*, isolation = Isolation.*READ\_UNCOMMITTED*)  
 public synchronized void deleteCar(int id) {  
 if (carRepository.findById(id) == null) {  
 return;  
 }  
 Car carToDelete = carRepository.findById(id);  
 File fileToDelete = new File(uploadPath + "/" + carToDelete.getImage());  
  
 if (fileToDelete.exists()) {  
 try {  
 fileToDelete.delete();  
 } catch (Exception e) {  
 System.*out*.println(e.getMessage());  
 }  
 }  
  
 carRepository.deleteById(id);  
 }  
  
 @Transactional  
 public void editeCar(Car car, MultipartFile file) throws IOException {  
 if (file.getOriginalFilename().isEmpty()) {  
 carRepository.save(car);  
 }  
 else {  
 File fileToDelete = new File(uploadPath + "/" + carRepository.findById(car.getId()).getImage());  
  
 if (fileToDelete.exists()) {  
 fileToDelete.delete();  
 }  
  
 saveCar(car, file);  
 }  
 }  
  
 public Map<String, Object> addCar(Car car, BindingResult bindingResult, MultipartFile file) throws IOException {  
 Map<String, Object> response = new HashMap<>();  
 if (bindingResult.hasErrors()) {  
 Map<String, List<String>> errorMap = bindingResult.getFieldErrors()  
 .stream()  
 .collect(Collectors.*groupingBy*(  
 FieldError::getField,  
 Collectors.*mapping*(FieldError::getDefaultMessage, Collectors.*toList*())  
 ));  
 response.put("ifError", true);  
 response.put("map", errorMap);  
 response.put("car", car);  
  
 return response;  
 }  
 else {  
 boolean ifAdded = saveCar(car, file);  
  
 response.put("ifAdded", ifAdded);  
 response.put("ifError", false);  
  
 return response;  
 }  
 }  
  
 public Map<String, Object> findCar(Map<String, String> form) {  
 Map<String, Object> response = new HashMap<>();  
 int priceOT = form.get("PriceOT") == null || form.get("PriceOT").equals("") ? -1 : Integer.*parseInt*(form.get("PriceOT"));  
 int priceDO = form.get("PriceDO") == null || form.get("PriceDO").equals("") ? -1 : Integer.*parseInt*(form.get("PriceDO"));  
  
 String brand = form.get("ListBrand");  
 String transmission = form.get("ListTypeTransmition");  
  
 String drive = form.get("typeDrive");  
 String body = form.get("typeBody");  
  
 int power = form.get("power") == null || form.get("power").equals("") ? -1 : Integer.*parseInt*(form.get("power"));  
 int year = form.get("year") == null || form.get("year").equals("") ? -1 : Integer.*parseInt*(form.get("year"));  
 int mileage = form.get("mileage") == null || form.get("mileage").equals("") ? -1 : Integer.*parseInt*(form.get("mileage"));  
  
 List<Car> cars = findAll();  
 List<Car> carsByParam = cars;  
  
 if (priceOT > 0) {  
 carsByParam = carsByParam.stream().filter(car -> car.getPrice() >= priceOT).collect(Collectors.*toList*());  
 }  
 if (priceDO > 0) {  
 carsByParam = carsByParam.stream().filter(car -> car.getPrice() <= priceDO).collect(Collectors.*toList*());  
 }  
 if (power > 0) {  
 carsByParam = carsByParam.stream().filter(car -> car.getPower() >= power).collect(Collectors.*toList*());  
 }  
 if (year > 0) {  
 carsByParam = carsByParam.stream().filter(car -> car.getYear() >= year).collect(Collectors.*toList*());  
 }  
 if (mileage > 0) {  
 carsByParam = carsByParam.stream().filter(car -> car.getMileage() <= mileage).collect(Collectors.*toList*());  
 }  
 if (brand != null && !brand.equals("")) {  
 carsByParam = carsByParam.stream().filter(car -> car.getBrand().equals(brand)).collect(Collectors.*toList*());  
 }  
 if (transmission != null && !transmission.equals("")) {  
 carsByParam = carsByParam.stream().filter(car -> car.getTransmission().equals(transmission)).collect(Collectors.*toList*());  
 }  
 if (drive != null && !drive.equals("")) {  
 carsByParam = carsByParam.stream().filter(car -> car.getDrive().equals(drive)).collect(Collectors.*toList*());  
 }  
 if (body != null && !body.equals("")) {  
 carsByParam = carsByParam.stream().filter(car -> car.getBody().equals(body)).collect(Collectors.*toList*());  
 }  
  
 response.put("countPage", Math.*ceil*((double) carsByParam.size()/10));  
 response.put("cars", carsByParam);  
 response.put("carsBrand", carRepository.getCarsBrand().stream().sorted().collect(Collectors.*toList*()));  
  
 return response;  
 }  
  
 public Map<String, Object> findCarsByClass(String carClass, int numberPage, int countItemOnPage) {  
 Map<String, Object> response = new HashMap<>();  
 List<Car> cars = carRepository.findByLevel(carClass, PageRequest.*of*(numberPage, 10)).stream().toList();  
 response.put("cars", cars);  
  
 response.put("countPage", Math.*ceil*((double) carRepository.count()/countItemOnPage));  
 response.put("carsBrand", carRepository.getCarsBrand());  
  
 return response;  
 }  
}

Код сервиса UserService:

package com.example.spring.security.services;  
  
import com.example.spring.security.models.Role;  
import com.example.spring.security.models.User;  
import com.example.spring.security.repositories.RoleRepository;  
import com.example.spring.security.repositories.UserRepository;  
import org.springframework.security.crypto.password.PasswordEncoder;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
import java.util.Set;  
import java.util.stream.Collectors;  
  
@Service  
public class UserService {  
  
 private final UserRepository userRepository;  
 private final RoleRepository roleRepository;  
 private final PasswordEncoder passwordEncoder;  
  
 public UserService(UserRepository userRepository, RoleRepository roleRepository, PasswordEncoder passwordEncoder) {  
 this.userRepository = userRepository;  
 this.roleRepository = roleRepository;  
 this.passwordEncoder = passwordEncoder;  
 }  
  
 public long count() {  
 return userRepository.count();  
 }  
 public long countByRoleName(String roleName) {  
 return userRepository.countByRoleName(roleName);  
 }  
 public List<User> findAll() {  
 return userRepository.findAll();  
 }  
 public User findByUsername(String username) {  
 return userRepository.findByUsername(username);  
 }  
 public User findById(int id) {  
 return userRepository.findById(id);  
 }  
 public User findByEmail(String email) {  
 return userRepository.findByEmail(email);  
 }  
 public User findOldestUser() {  
 return userRepository.findOldestUser();  
 }  
 public User findYoungestUser() {  
 return userRepository.findYoungestUser();  
 }  
 @Transactional  
 public boolean editeUser(int userId, Map<String, String> rolesList) {  
  
 User user = userRepository.findById(userId);  
 if (rolesList.get("userName") == null) {  
 return false;  
 }  
 user.setUsername(rolesList.get("userName"));  
  
 Set<String> roles = roleRepository.findAll().stream()  
 .map(Role::getName)  
 .collect(Collectors.*toSet*());  
  
 user.getRoles().clear();  
  
 for (String key: rolesList.keySet()) {  
 if (roles.contains(key)) {  
 user.getRoles().add(roleRepository.findByName(key));  
 }  
 }  
 userRepository.save(user);  
  
 if (userRepository.findByUsername(user.getUsername()) == null) {  
 return false;  
 }  
  
 return true;  
 }  
 @Transactional  
 public void saveUser(User user) {  
 userRepository.save(user);  
 }  
 @Transactional  
 public Map<String, Object> changePassword(Map<String, String> form, User user) {  
 String oldPassword = form.get("oldPassword");  
 String newPassword = form.get("newPassword");  
 String confirmPassword = form.get("confirmPassword");  
  
 Map<String, Object> response = new HashMap<>();  
  
 boolean result = passwordEncoder.matches(oldPassword, user.getPassword());  
  
 if (!result) {  
 response.put("error", "Неправильный старый пароль!");  
 response.put("response", false);  
 } else if (newPassword.equals(confirmPassword)){  
 user.setPassword(passwordEncoder.encode(newPassword));  
 userRepository.save(user);  
  
 response.put("successPassword", "Пароль бы изменён!");  
 response.put("response", true);  
 } else {  
 response.put("error", "Новый пароль и подтверждение пароля не совпадают!");  
 response.put("response", false);  
 }  
 return response;  
 }  
 public double getAverageAge() {  
 return userRepository.getAverageAge();  
 }  
}

ПРИЛОЖЕНИЕ б Скрипт создания базы данных

Весь код проекта представлен в GIT репозитории по ссылке: <https://github.com/sixxG/CarRental-.git>.

Код модели cars:

create table cars (  
 id integer not null auto\_increment,  
 win\_number varchar(255) not null,  
 body varchar(255) not null,  
 brand varchar(255) not null,  
 color varchar(255) not null,  
 description varchar(10000),  
 drive varchar(255) not null,  
 image varchar(255) not null,  
 level varchar(255) not null,  
 mileage integer not null,  
 model varchar(255),  
 power integer not null,  
 price integer not null,  
 status varchar(255) not null,  
 transmission varchar(255) not null,  
 year integer not null,  
 primary key (id)  
);

Код модели feedback:

create table feedbacks (  
 id integer not null auto\_increment,  
 author varchar(255) not null,  
 body varchar(500) not null,  
 date datetime(6) not null,  
 score integer not null,  
 primary key (id)  
);

Код модели roles:

create table roles (  
 id integer not null auto\_increment,  
 name varchar(255) not null,  
 primary key (id)  
);

Код модели users:

create table users (  
 id integer not null auto\_increment,  
 email varchar(255) not null,  
 password varchar(255) not null,  
 username varchar(255) not null,  
 fio varchar(255) default '',  
 birth\_date date DEFAULT (*CURRENT\_DATE*),  
 address varchar(255) default '',  
 phone varchar(255) default '',  
 driver\_license varchar(255) default '',  
  
 primary key (id)  
);

Код модели users\_roles:

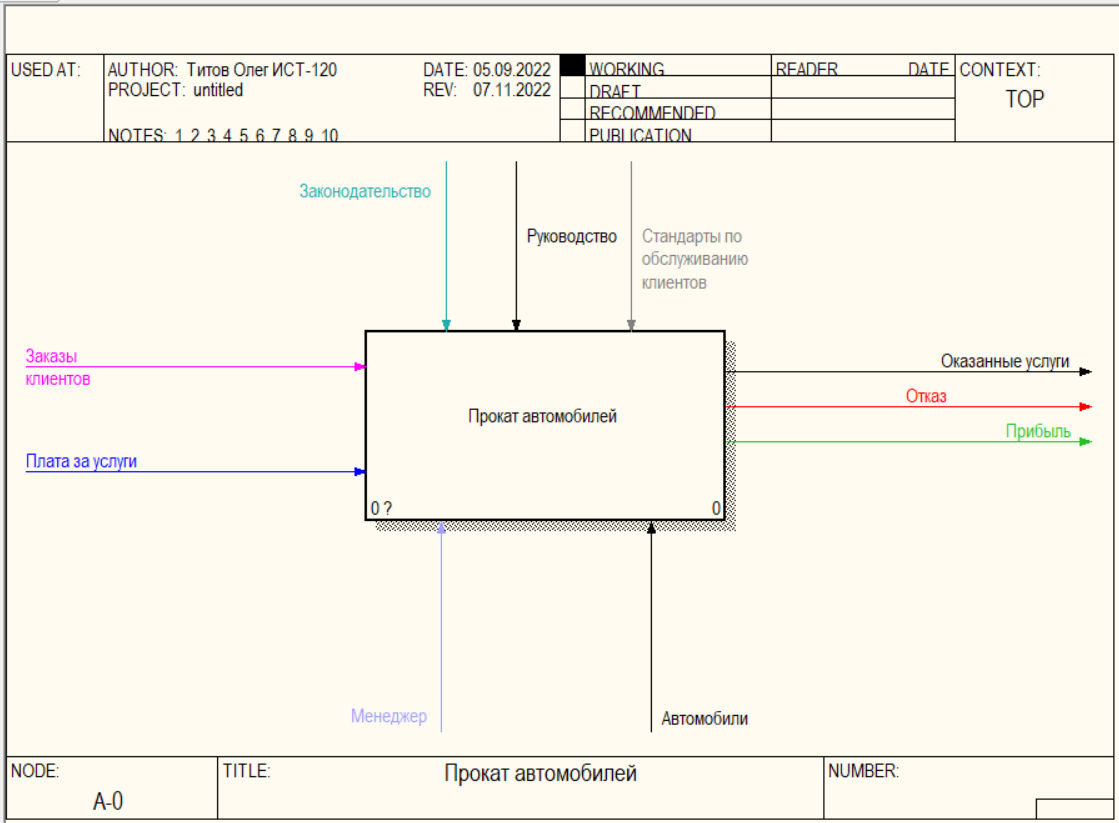
create table users\_roles (  
 user\_id integer not null,  
 role\_id integer not null  
);  
  
alter table users\_roles  
 add constraint FKj6m8fwv7oqv74fcehir1a9ffy  
 foreign key (role\_id) references roles (id) on DELETE cascade;  
  
alter table users\_roles  
 add constraint FK2o0jvgh89lemvvo17cbqvdxaa  
 foreign key (user\_id) references users (id) on DELETE cascade;

Код модели contracts:

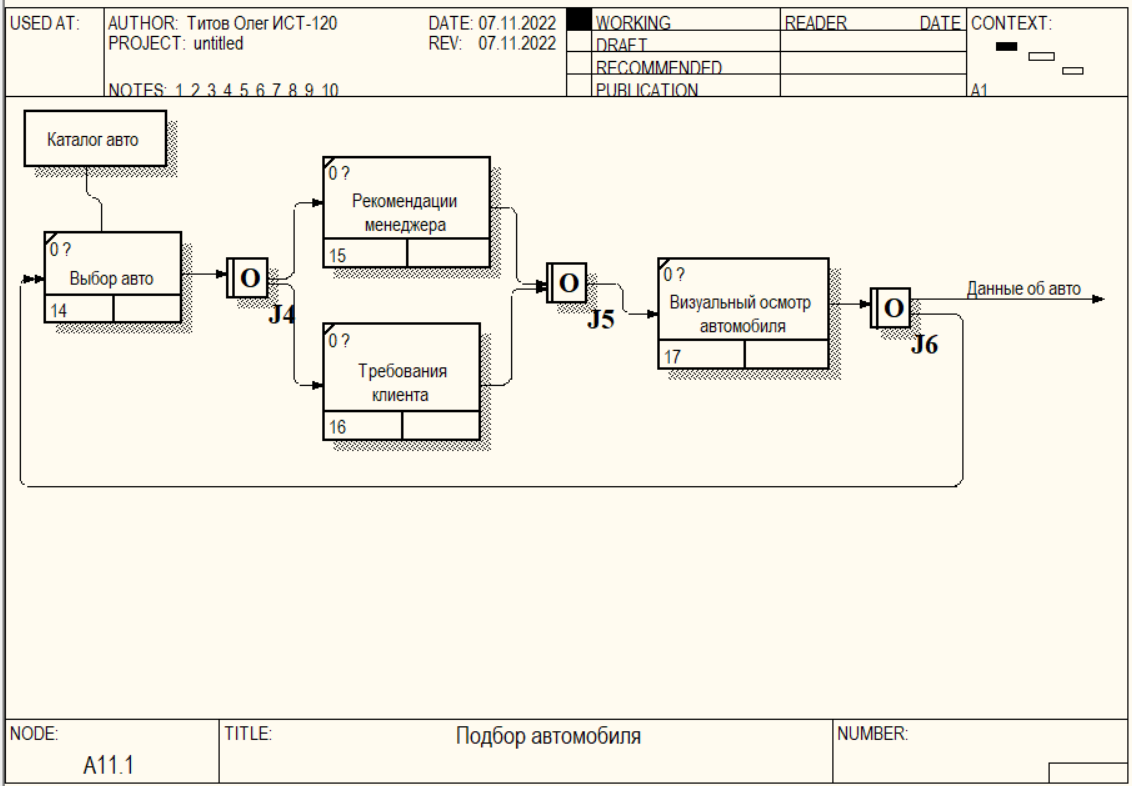
create table contracts (  
 id integer not null auto\_increment,  
 additional\_options varchar(255),  
 status varchar(255) not null,  
 date\_end datetime not null,  
 type\_return varchar(255) not null,  
 type\_receipt varchar(255) not null,  
 date\_start datetime not null,  
 fio\_manager varchar(255),  
 note varchar(255),  
 price double precision not null,  
 car\_id integer not null,  
 client\_id integer not null,  
 primary key (id)  
);  
  
alter table contracts  
 add constraint contracts\_car\_id  
 foreign key (car\_id) references cars (id);  
  
alter table contracts  
 add constraint contracts\_client\_id  
 foreign key (client\_id) references users (id);

ПРИЛОЖЕНИЕ В IDEF0 модели as-is

*А-0 Диаграмма верхнего уровня бизнес-процесса «Прокат автомобилей»:*



*А11.1 Диаграмма работы «Подбор автомобиля»:*



*Диаграмма дерева узлов:*

