Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет инженерно-экономический
Кафедра экономической информатики
Дисциплина «Программирование сетевых приложений»

«К ЗАЩИТЕ ДОПУСТИТЬ»
Руководитель курсового проекта
Ассистент кафедры ЭИ
_____ Ю.В.Сильванович
___.____.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту
на тему:

«ПРОГРАММНАЯ ПОДДЕРЖКА ДЕЯТЕЛЬНОСТИ АВТОСАЛОНА»

БГУИР КР 1-40 05 01-08 026 ПЗ

Выполнил студент группы 272303
ХУДНИЦКИЙ Арсений Андреевич
_____
(подпись студента)
Курсовой проект представлен на проверку ___.____.2024
_____
(подпись студента)

Минск 2024

РЕФЕРАТ

БГУИР КР 1-40 05 01-08 026 ПЗ

Худницкий, А.А. Программная поддержка деятельности автосалона: пояснительная записка к курсовому проекту / А.А. Худницкий. – Минск: БГУИР, 2024. – 69 с.

Пояснительная записка 69 с., 60 рис., 10 источников, 4 приложения

ПРОГРАММНОЕ СРЕДСТВО, КЛИЕНТ-СЕРВЕР, БАЗА ДАННЫХ, ИНТЕРНЕТ-СЕРВИС, ОПТИМИЗАЦИЯ, МОДЕЛИРОВАНИЕ ДИАГРАММ, ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС.

Цель проектирования: снижение времени на актуализацию данных о предоставляемых услугах и ценах, увеличение скорости обслуживания клиентов, а также упрощение процесса оставления заявки на автомобиль.

Методология проведения работы: системные, статистические и общенаучные методы.

Результаты работы: выполнена постановка задачи и определены основные методы ее решения. Была детально исследована предметная область проекта, а именно основные бизнес-процессы работы автосалона. Была разработана функциональная и информационная модели системы; построен ряд UML-диаграмм; описаны алгоритмы работы сервиса; выполнено тестирование системы; разработано руководство пользователя. В результате проектирования были реализованы серверная и клиентская части приложения, позволяющие пользователю взаимодействовать с системой.

Область применения результатов: разработанная система решает задачу увеличения производительности при обработке информации. Приложение будет полезно для всех пользователей, которые хотят ознакомиться со всем перечнем автомобилей, предоставляемых автосалоном, а также упростить задачу оставления заявки на автомобиль.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В современном мире деятельность автосалонов становится все более динамичной и высококонкурентной. Услуги автосалонов пользуются большим спросом, что связано с ростом автопарка и потребностью клиентов в качественном обслуживании, покупке и техническом обслуживании автомобилей. Однако с увеличением объема предоставляемых услуг и расширением клиентской базы возникает необходимость в оптимизации и автоматизации процессов, связанных с управлением автосалоном.

Традиционные методы ведения бизнеса в автосалонах часто сопровождаются сложностями, связанными с управлением записью клиентов, координацией работы сотрудников, обновлением информации о наличии автомобилей и предоставляемых услугах. Эти процессы могут занимать значительное время как у персонала, так и у клиентов, что в конечном итоге может негативно сказаться на уровне обслуживания и удовлетворенности покупателей.

С целью повышения эффективности работы автосалона и улучшения взаимодействия с клиентами, разработка системы автоматизации становится актуальной задачей. Автоматизированная система позволит уменьшить время на обработку данных, ускорить процесс оставления заявки на автомобиль, а также обеспечить доступ клиентов к информации о доступных автомобилях в режиме онлайн.

Целью данного курсового проекта является повышение эффективности при работе с заявками пользователей на автомобиль, которая будет способствовать улучшению качества обслуживания и оптимизации бизнес-процессов.

Для достижения этой цели необходимо выполнить следующие задачи:

– провести анализ и моделирование предметной области;

– разработать необходимые функциональные, UML модели системы;

– проектировать программное обеспечение и разработать базу данных;

– реализовать удобный и интуитивно понятный интерфейс, затем протестировать и проверить работоспособность данной системы.

В результате реализации проекта ожидается создание удобного и интуитивно понятного интерфейса, который упростит процесс оформления заявки на автомобиль. Разработанная система будет способствовать повышению производительности работы автосалона и увеличению уровня удовлетворенности клиентов, что в свою очередь будет способствовать его успешному развитию в конкурентной среде.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ДЕЯТЕЛЬНОСТИ АВТОСАЛОНА

1.1 Описание деятельности автосалона

В современных условиях деятельность автосалонов играет важную роль в жизни многих людей, стремящихся приобрести, обслуживать или модернизировать свои автомобили. Автомобиль давно стал неотъемлемой частью повседневной жизни, обеспечивая комфорт и свободу передвижения. С учетом увеличивающегося количества владельцев автомобилей и растущего разнообразия предлагаемых услуг, автосалоны сталкиваются с необходимостью адаптироваться к новым требованиям рынка и ожиданиям клиентов. Это особенно актуально в условиях высококонкурентной среды, где успешность бизнеса напрямую зависит от качества предоставляемых услуг и уровня удовлетворенности клиентов [1].

Автосалоны могут существенно различаться по своему масштабу и спектру предлагаемых услуг. Некоторые из них представляют собой небольшие компании, работающие с ограниченным ассортиментом автомобилей, в то время как другие являются крупными сетевыми организациями, предлагающими комплексные решения, включающие продажу новых автомобилей, покупку подержанных автомобилей, сервисное обслуживание, а также программы финансирования и страхования. Несмотря на эти различия, общая задача для всех автосалонов – обеспечить высокое качество обслуживания, оперативное выполнение заявок клиентов и прозрачность бизнес-процессов. Однако традиционные методы управления бизнесом зачастую не позволяют справляться с этими задачами эффективно. Ручное ведение записей, недостаточная синхронизация данных между отделами и ограниченный доступ клиентов к информации могут привести к задержкам в обслуживании и снижению лояльности клиентов.

В условиях стремительного развития технологий актуальной задачей становится автоматизация бизнес-процессов в автосалонах. Внедрение современных информационных систем позволяет оптимизировать широкий спектр операций, начиная от обработки заявок на покупку автомобилей и заканчивая управлением складскими запасами запчастей. Например, автоматизация процесса записи на обслуживание обеспечивает клиентам возможность бронирования времени через интернет или мобильное приложение, исключая необходимость личного обращения в автосалон. Такой подход экономит время как клиента, так и сотрудников автосалона, а также сводит к минимуму вероятность ошибок, связанных с человеческим фактором.

Еще одним важным аспектом автоматизации является предоставление клиентам актуальной информации о наличии автомобилей и сервисных услугах. Онлайн-доступ к базе данных позволяет клиентам в любое время узнавать о доступных моделях, их характеристиках, стоимости, а также об акциях и специальных предложениях. Это повышает прозрачность и доверие к

автосалону, что особенно важно в условиях жесткой конкуренции. Клиенты могут не только оставить заявку на покупку, но и добавить понравившиеся автомобили в избранное, чтобы вернуться к выбору позже. Интеграция таких функций делает процесс взаимодействия с автосалоном более удобным и интуитивно понятным, что в конечном итоге положительно сказывается на уровне удовлетворенности клиентов.

Для сотрудников автосалонов автоматизация становится инструментом, который значительно упрощает управление операционной деятельностью. Центральная система позволяет координировать работу различных отделов, отслеживать текущие заявки, планировать загрузку сервиса и своевременно обновлять данные о наличии автомобилей. Это не только повышает производительность труда, но и снижает вероятность ошибок, связанных с устаревшей информацией или человеческими факторами. Более того, автоматизация позволяет анализировать клиентские данные, что способствует лучшему пониманию их потребностей и разработке персонализированных предложений.

Разработка автоматизированного приложения в рамках данного курсового проекта направлена на решение ключевых задач, стоящих перед современными автосалонами. Приложение предоставит все необходимые инструменты для организации взаимодействия между клиентами и сотрудниками, оптимизации процессов и повышения эффективности работы. Благодаря интеграции с базами данных, аналитическим модулям и инструментам управления, система станет не только средством автоматизации, но и мощным инструментом для развития бизнеса. Такой подход позволит не только минимизировать ручной труд, но и значительно улучшить качество обслуживания клиентов, что способствует укреплению их лояльности и повышению конкурентоспособности автосалона.

Таким образом, внедрение автоматизированной системы станет важным шагом на пути модернизации автосалона. Она не только оптимизирует текущие процессы, но и создаст платформу для дальнейшего развития, соответствующего современным стандартам рынка. Это особенно актуально в условиях возрастающего спроса на цифровые решения и необходимости быстрой адаптации к изменениям в потребительских предпочтениях.

1.2 Разработка функциональной модели деятельности автосалона

В данном курсовом проекте для разработки программной поддержки деятельности автосалона используется стандарт IDEF0 и инструментальная среда BpWin.

На рисунке 1.1 представлена контекстная диаграмма процесса «Продать автомобиль». Входные данные представляют собой запрос клиента, информация о доступных автомобилях, данные поставок автомобилей, история взаимодействия с клиентом, данные о складе. В данном процессе используются следующие методы управления: правила покупки, продажи, дарения автомобилей, внутренние регламенты автосалона, прайс-листы

автосалона. Выполняют данный процесс следующие механизмы: менеджеры, бухгалтер, оператор поддержки и складские сотрудники (рисунок 1.1).

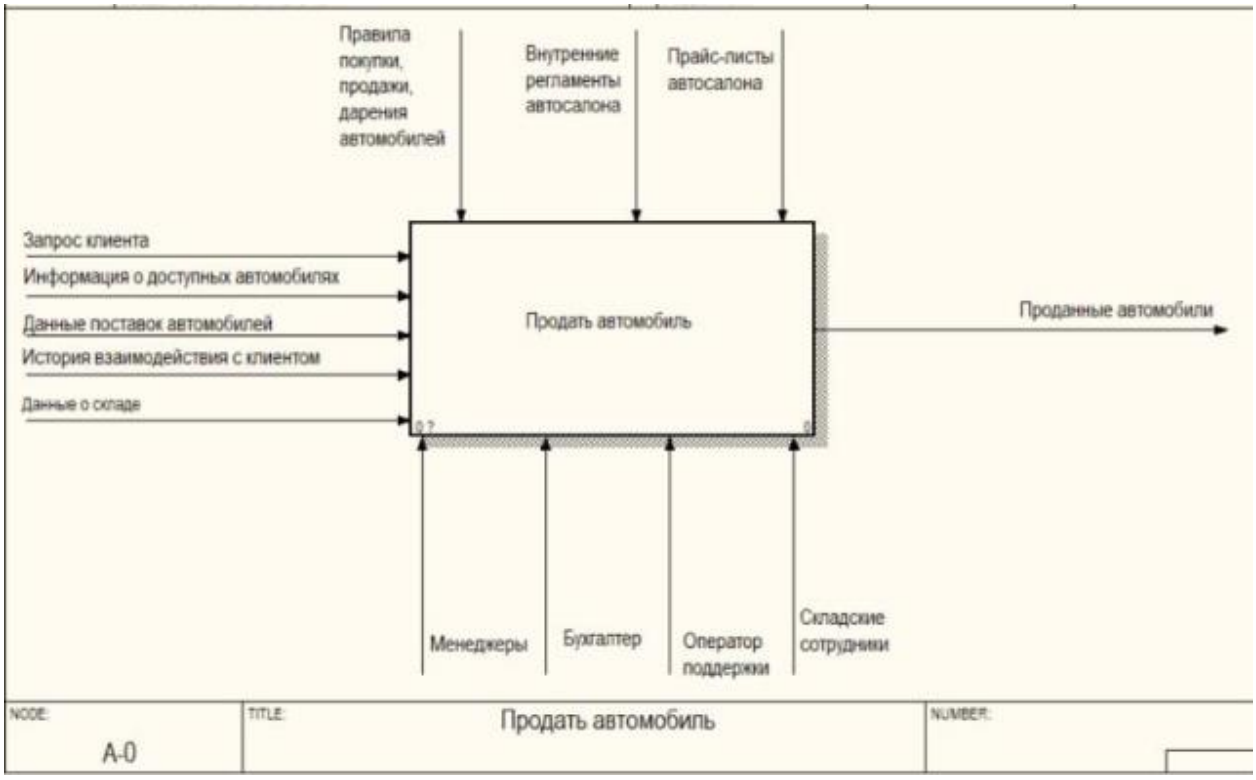

Рисунок 1.1 – Контекстная диаграмма

Для достижения поставленной цели разобьем нашу функциональную модель используя принцип декомпозиции. На первом уровне получается пять функциональных блока: «Обработать первичный запрос клиента», «Выбрать автомобиль», «Согласовать условия сделки», «Оформить и завершить сделку», «Выдать автомобиль» (рисунок 1.2). На выходе процесса «Обработать первичный запрос клиента» получится «Предварительный список автомобилей», процесса «Выбрать автомобиль» – «Рекомендованный автомобиль», процесса «Согласовать условия сделки» – «Окончательные условия продажи», процесса «Оформить и завершить сделку» – «Подписанный договор», «Выдать автомобиль» – «Проданный автомобиль».

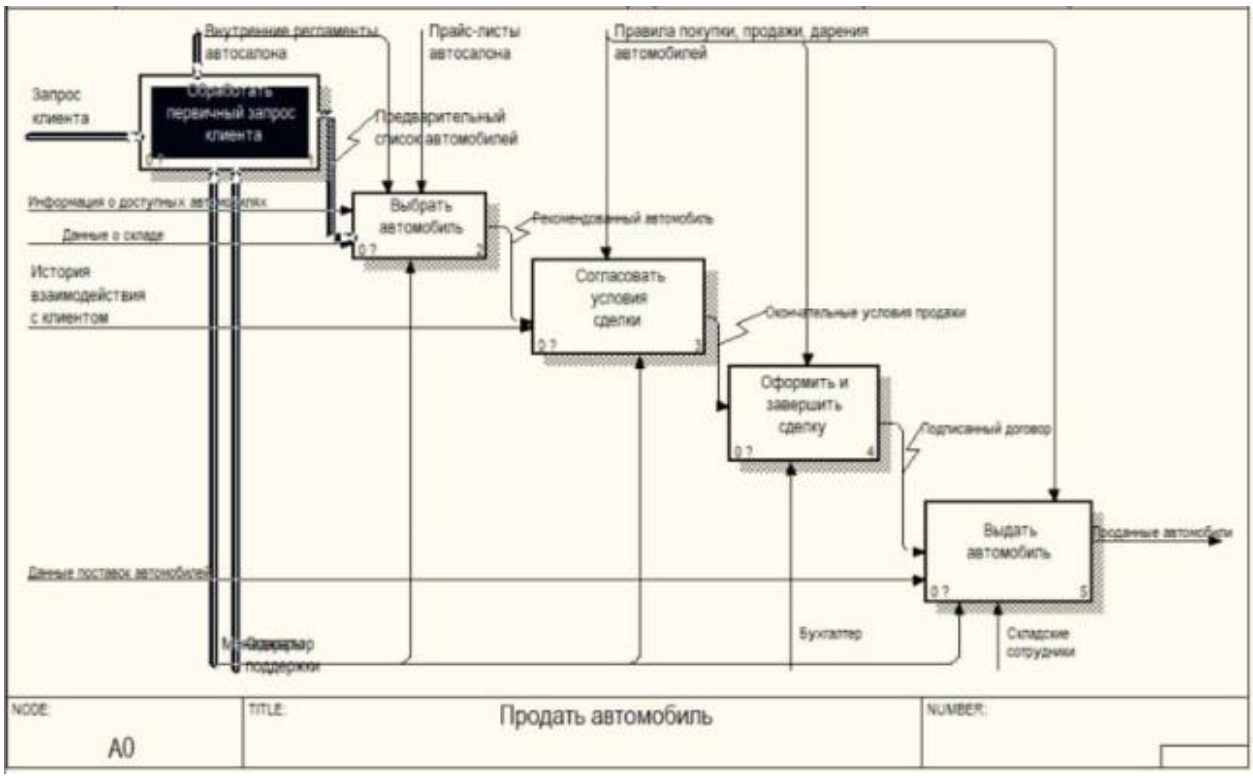

Рисунок 1.2 – Первый уровень декомпозиции

Для перехода к выбору автомобиля нужно составить предварительный список автомобилей. Для этого функциональный блок «Обработать первичный запрос клиента» разбивается на следующие функциональные блоки: «Принять запрос», «Анализировать предпочтения клиента», «Составить предварительный список автомобилей» (рисунок 1.3). На выходе получается предварительный список автомобилей, который используется в качестве входных данных для следующего функционального блока на первом уровне декомпозиции (рисунок 1.2).

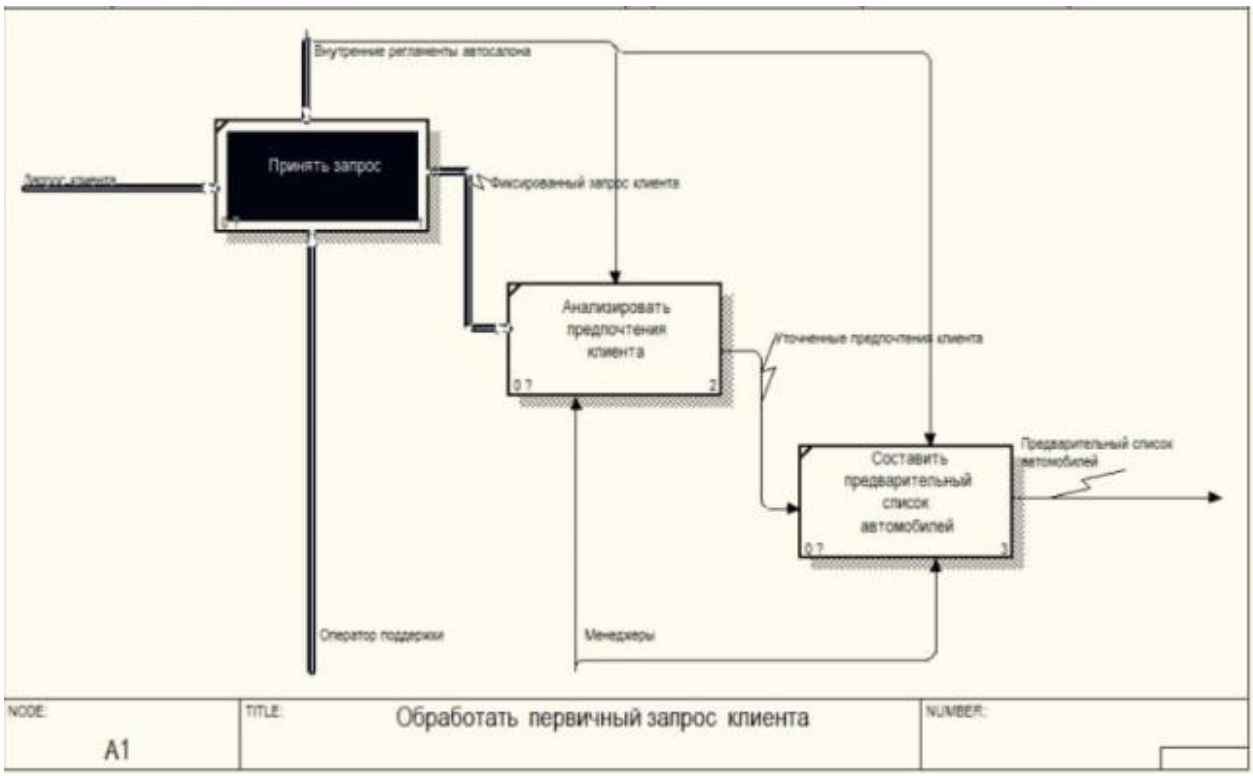

Рисунок 1.3 – Декомпозиция функционального блока «Обработать первичный запрос клиента»

Для начала согласования условий сделки необходимо выбрать конкретный автомобиль. Для этого блок «Выбрать автомобиль» разбивается на следующие функциональные блоки: «Анализировать наличие на складе», «Сравнить технические характеристики», «Подготовить рекомендации для клиента» (рисунок 1.4). На выходе получается рекомендованный автомобиль, который будет являться входными данными для следующего функционального блока на первом уровне декомпозиции (рисунок 1.2).

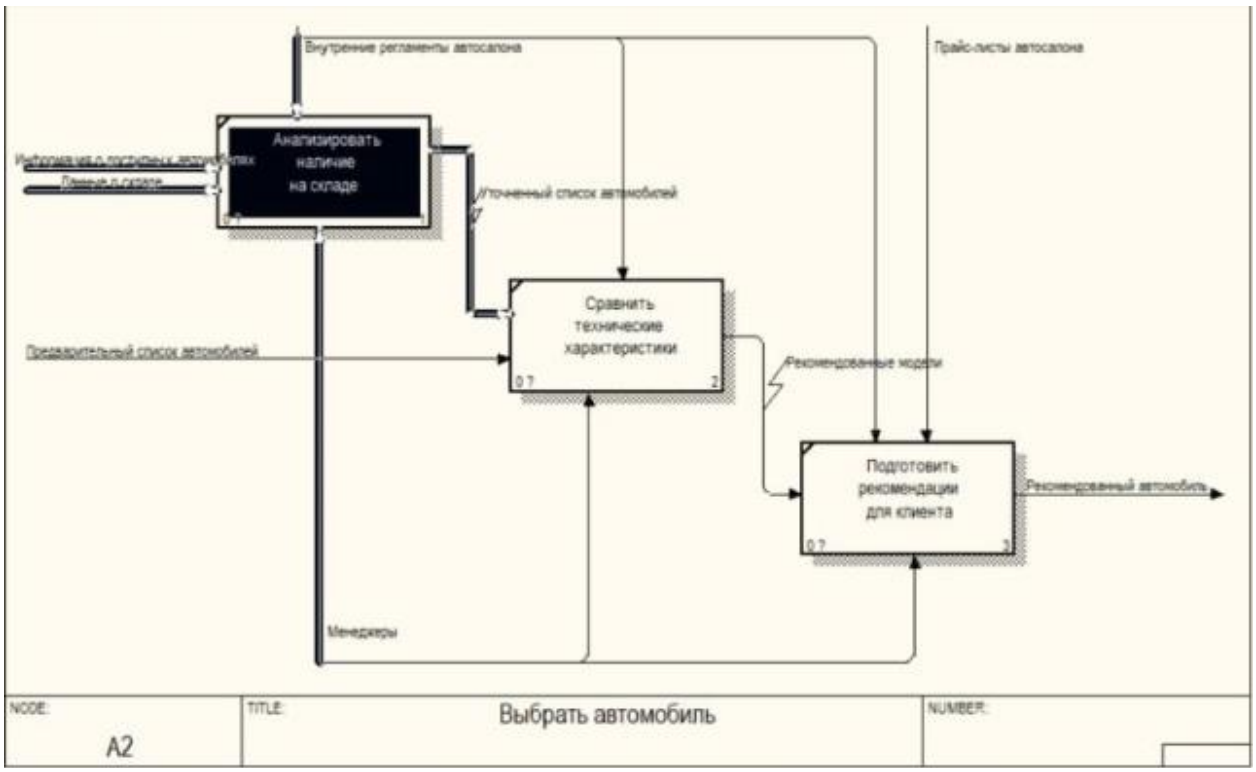

Рисунок 1.4 – Декомпозиция функционального блока «Выбрать автомобиль»

Для окончательных условий продажи разбить функциональный блок «Согласовать условия сделки» на следующие функциональные блоки: «Уточнить окончательную цену», «Согласовать скидки и акции», «Определить условия оплаты и доставки» (рисунок 1.5). На выходе получаются окончательные условия продажи, что является входными данными для следующего функционального блока на первом уровне декомпозиции (рисунок 1.2).

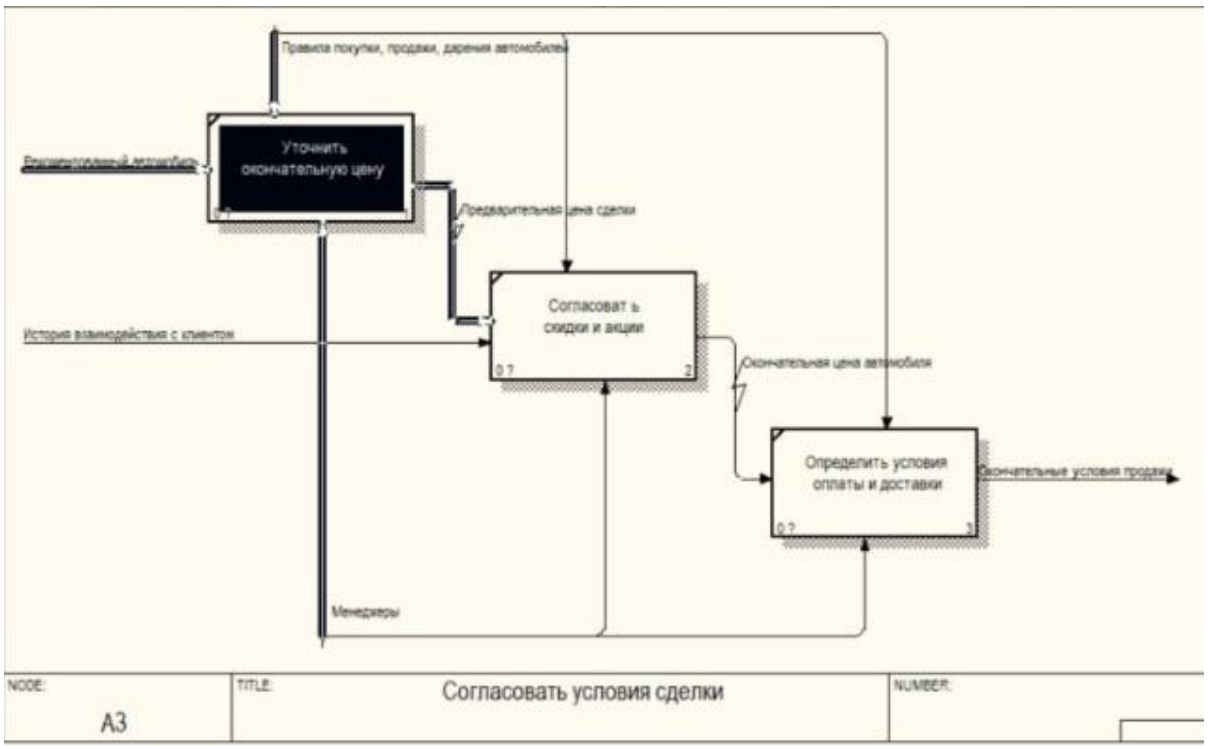

Рисунок 1.5 – Декомпозиции функционального блока «Рассчитать налоги и сборы»

Для создания подписанного договора необходимо разбить функциональный блок «Оформить и завершить сделку» на следующие функциональные блоки: «Подготовить договор», «Согласовать и подписать договор», «Оформить накладные» (рисунок 1.6). На выходе получается подписанный договор, который является входными данными для следующего функционального блока на первом уровне декомпозиции (рисунок 1.2).

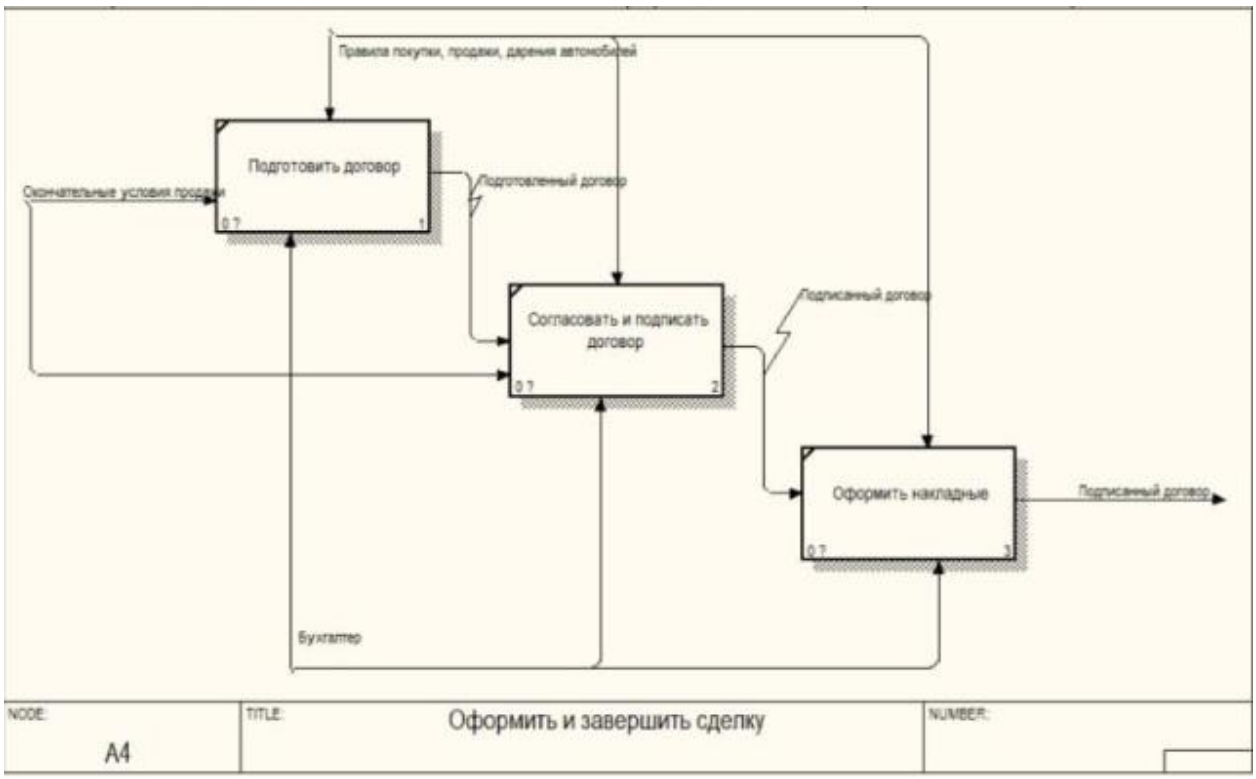

Рисунок 1.6 – Декомпозиции функционального блока «Оформить и завершить сделку»

На выходе процесса «Выдать автомобиль» имеем «Проданный автомобиль». Этот процесс можно разбить на три функциональных блока: «Подготовить автомобиль к выдаче», «Выдать автомобиль клиенту», «Окончательно подтвердить сделку» (рисунок 1.7). Также разобьем процесс «Окончательно подтвердить сделку» на три блока: «Проверить подписанные документы», «Сверить данные по сделке», «Обновить данные в учетных журналах» (рисунок 1.8). На выходе имеем «Проданный автомобиль».

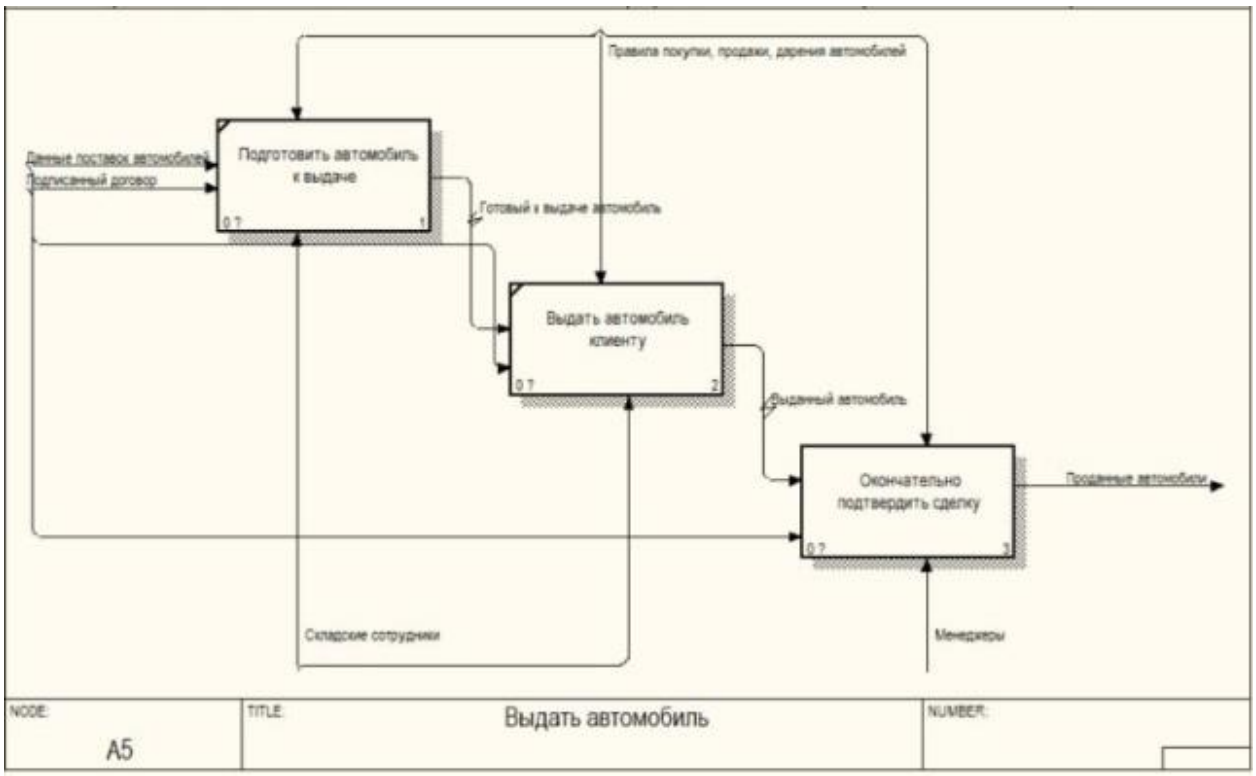

Рисунок 1.7 – Декомпозиции функционального блока «Выдать автомобиль»

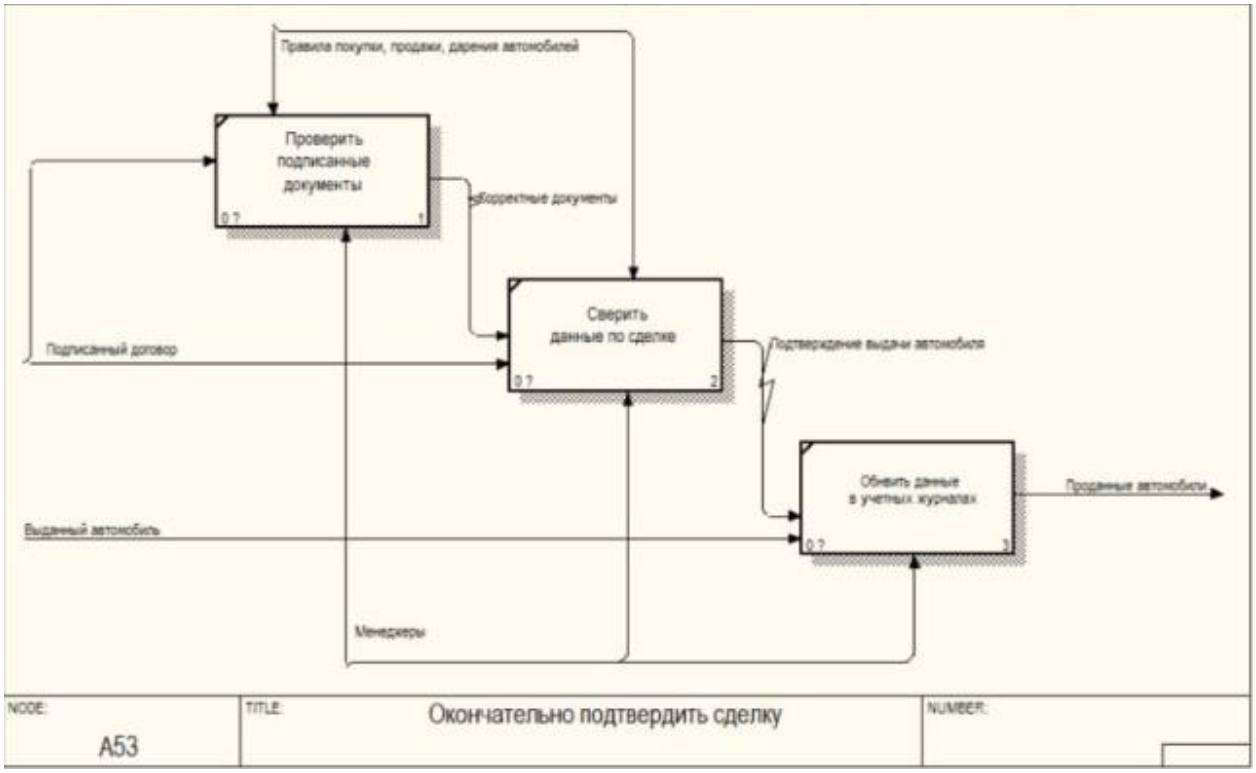

Рисунок 1.8 – Декомпозиции функционального блока «Окончательно подтвердить сделку»

Таким образом процесс продажи автомобиля является многоэтапным и имеет свои особенности.

1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований

Разрабатываемое приложение должно иметь удобный и интуитивно понятный интерфейс для того, чтобы администратор мог управлять записями о сотрудниках и каталогом доступных автомобилей, на которые пользователь может оставить заявку, чтобы пользователь мог без проблем зарегистрироваться, войти в учётную запись клиента и там мог изучить всю доступную информацию.

В данной системе будут существовать такие роли как:

– администратор;

– клиент;

– работник.

Администратор может добавлять и удалять данные о сотрудниках. Каждый работник регистрируется как обычный клиент, а далее из перечня клиентов администратор выбирает нужных для выдачи роли работника. Также администратор может управлять данными об автомобилях, то есть удалять и добавлять автомобили в систему.

Клиент в свою очередь имеет доступ к каталогу автомобилей, который позволяет ему:

– ознакомиться с полным списком доступных автомобилей;

– отфильтровать каталог по ценам;

– отправлять заявку на выбранный автомобиль.

Также клиент имеет возможность добавлять автомобили в избранное и записываться на тест-драйв.

Работник способен просматривать заявки от клиентов, отправляя в ответ отказ или соглашение, заблокировать пользователя в системе создавать отчет о подтвержденных заявках, а также одобрить или отклонить заявку на тест-драйв от пользователя (рис. 1.9).

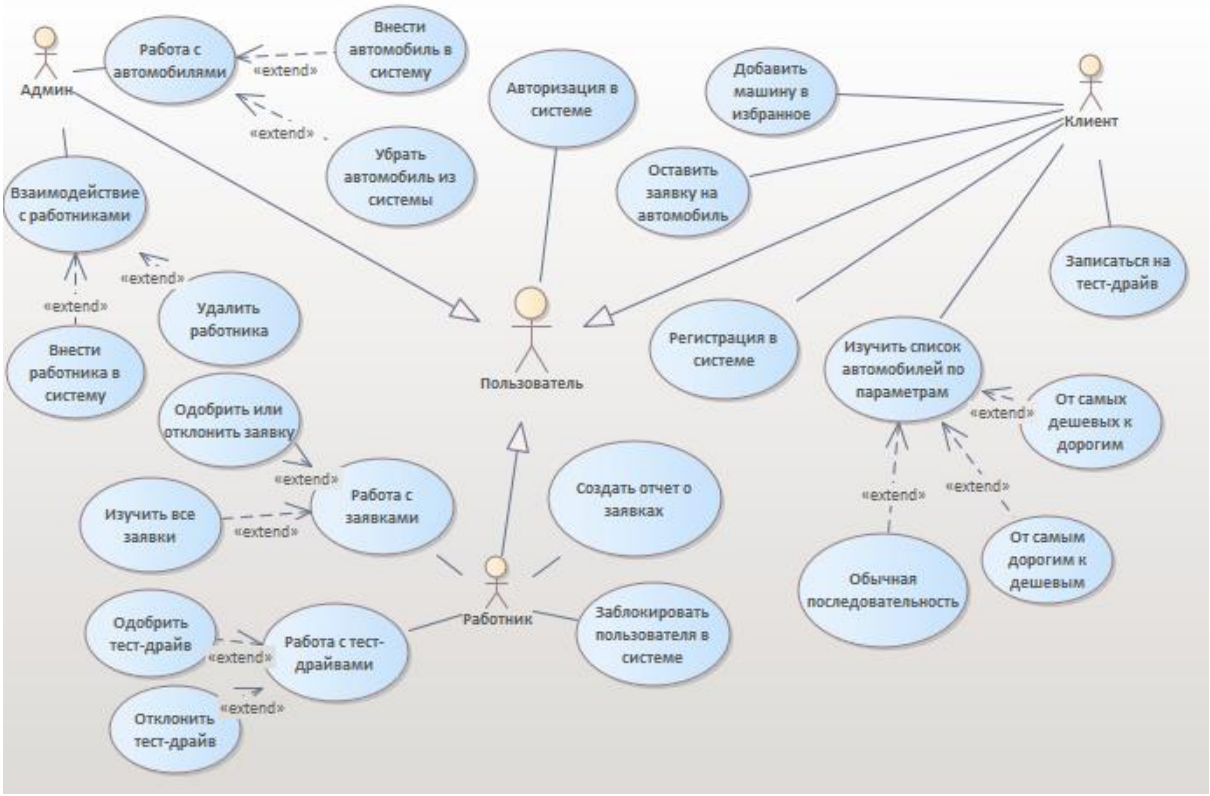

Рисунок 1.9 – Диаграмма Use case

Все данные в программном продукте проверяются на корректность и правильность (невозможно ввести строку вместо цифр, выбрать несуществующую дату и т.д.).

Помимо удобного интерфейса, для работы с приложением так же необходимо реализовать базу данных, настроить связь между клиентом и сервером (спроектировать клиент-серверную архитектур). В созданной БД будет хранится вся информация необходимая для реализации основных бизнес-процессов автосалона.

1.4 Разработка информационной модели предметной области

Информационная модель – модель объекта, представленная в виде информации, описывающей существенные для данного рассмотрения параметры и переменные величины объекта, связи между ними, входы и выходы объекта и позволяющая путем подачи на модель информации об изменениях входных величин моделировать возможные состояния объекта. Так же в более широком смысле информационная модель – совокупность информации, характеризующая существенные свойства и состояния объекта, процесса, явления, а также взаимосвязь с внешним миром.

Физическая модель базы данных содержит все детали, необходимые для создания базы: наименования таблиц и столбцов, типы данных, определения первичных и внешних ключей.

Информационная модель может быть представлена в виде схемы, диаграммы или формального описания. В ней могут использоваться различные концептуальные языки, такие как диаграммы классов, ER-диаграммы (диаграммы сущность-связь), UML (язык моделирования единых концепций) и другие.

Основные элементы информационной модели предметной области включают:

а) Сущности (объекты): представляют основные концепции в предметной области.

б) Атрибуты: описывают характеристики сущностей.

в) Связи: определяют отношения и взаимосвязи между сущностями.

г) Ограничения: определяют правила и ограничения, которые должны соблюдаться в предметной области.

Основным критерием качества разработанной модели данных является ее соответствие так называемым нормальным формам (далее НФ). Основная цель нормализации — устранение избыточности данных. Кодом были определены три нормальные формы, которые включают одна другую. Соответствие 3НФ подразумевает соответствие 1НФ и 2НФ.

Для хранения данных на сервере установлен сервер MySQL 8.0. На этом сервере будут храниться данные о пользователях, об автомобилях, о всех заявках и тест-драйвах. На рисунке 1.10 показана модель базы данных, на которой изображены все таблицы и их связи между собой.

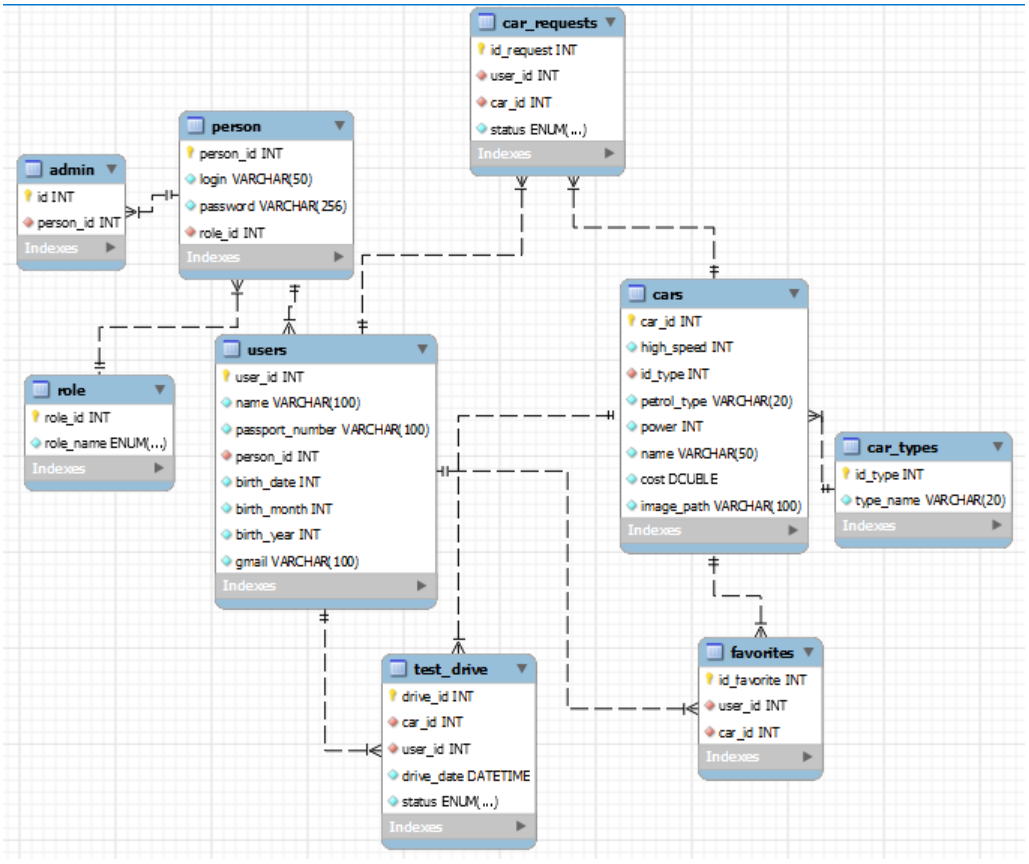

Рисунок 1.10 – Модель базы данных

В базе данных находятся следующие таблицы. Рассмотри каждую по отдельности.

Таблица users предназначена для хранения данных о зарегистрированных сотрудниках и клиентах системы. Связь с таблицей person 1:1. Рассмотрим все свойства таблицы users:

– user_id – хранит идентификатор пользователя, для каждого пользователя он уникальный;

– name – хранит ФИО работника или клиента;

– passport_number – хранит номер паспорта работника или клиента;

– person_id – хранит id связанного с ним объекта person;

– birth_date – хранит день рождения работника или клиента;

– birth_month – хранит месяц рождения работника или клиента;

– birth_year – хранит год рождения работника или клиента;

– gmail – хранит почту работника или клиента.

Таблица person предназначена для хранения логинов, паролей, а также ролей всех пользователей системы. Связь с таблицей role 1:1. Рассмотрим все свойства таблицы person:

– person_id – уникальный идентификатор;

– login – хранит логин пользователя;

– password – хранит пароль пользователя;

– role_id – хранит роль пользователя.

Таблица role предназначена для хранения всех ролей пользователей в системе. Рассмотрим все свойства таблицы role:

– role_id – хранит идентификатор роли, для каждой роли идентификатор уникальный;

– role_name – хранит все доступные роли в системе.

Таблица admin предназначена для хранения администратора, существует только один администратор. Связь с person 1:1. Рассмотрим все свойства таблицы admin:

– id – хранит идентификатор админа;

– person_id – хранит id связанного объекта person.

Таблица test_drive предназначена для хранения тест-драйвов. Связь с user 1:M. Связь с car 1:M. Рассмотрим все свойства таблицы test_drive:

– drive_id – хранит уникальный идентификатор тест-драйва;

– car_id – хранит id связанного объекта car;

– user_id – хранит id связанного объекта user;

– drive_date – хранит дату тест-драйва;

– status – хранит статус данного тест-драйва, может быть NONE, REJECT, ACCEPT.

Таблица cars предназначена для хранения автомобилей. Связь с type 1:1. Рассмотрим все свойства таблицы car:

– car_id – хранит уникальный идентификатор автомобиля;

– high_speed – хранит максимальную скорость, которую развивает данный автомобиль;

– id_type – хранит id связанного объекта car_type;

– petrol_type – хранит тип топлива данного автомобиля;

– power – хранит мощность (лошадиные силы) данного автомобиля;

– name – хранит название данного автомобиля;

– cost – хранит стоимость автомобиля;

– image_path – хранит путь к фото автомобиля.

Таблица car_types предназначена для хранения типов автомобилей. Рассмотрим все свойства таблицы car_types:

– id_type – хранит уникальный идентификатор типа автомобиля;

– type_name – хранит название типа автомобиля.

Таблица car_requests предназначена для хранения заявок на автомобили. Связь с user 1:M. Связь с car 1:M. Рассмотрим все свойства таблицы car_requests:

– id_request – хранит уникальный идентификатор заявки на автомобиль;

– car_id – хранит id связанного объекта car;

– user_id – хранит id связанного объекта user;

– status – хранит статус данной заявки на автомобиль, может быть NONE, WAIT, REJECT, ACCEPT.

Таблица favorites предназначена для хранения избранных автомобилей конкретного пользователя. Связь с user 1:M. Связь с car 1:M. Рассмотрим все свойства таблицы favorites:

– id_request – хранит уникальный идентификатор заявки на автомобиль;

– car_id – хранит id связанного объекта car;

– user_id – хранит id связанного объекта user.

Докажем, что данная база данных находится в третьей нормальной форме. Данная база данных находится в 1-й нормальной форме, так как в любом допустимом значении отношения каждый кортеж содержит только одно значение для каждого из атрибутов. Также база данных находится и во второй нормальной форме, так как находится в первой нормальной форме, и каждый неключевой атрибут функционально зависит от ее потенциального ключа. Так как база данных находится во второй нормальной форме и у неё отсутствуют функциональный зависимости неключевых атрибутов от ключевых, то она находится в третьей нормальной форме.

При запуске серверного приложения происходит подключение к этой базе данных. Если же окажется что базы данных не существует или таблиц, входящих в базу данных не будет, то будет выдана ошибка. SQL-скрипт, который создает эту бд, находится в проекте (см. Приложение Б).

1.5 UML-модели представления программного средства и их описание

В данном курсовом проекте рассматриваются следующие модели представления разрабатываемой системы:

− диаграмма развертывания;
− диаграмма состояний;
− диаграмма последовательности;
− диаграмма классов;
− диаграмма компонентов.

Рассмотрим подробнее каждую из них.

Диаграммы последовательностей являются эффективным средством для более подробного описания сценариев использования в рамках проекта. Они позволяют документировать взаимодействие объектов и сообщения, которыми они обмениваются, а также связанные с ними возвращаемые результаты. Данные диаграммы полезны для более глубокого понимания последовательности действий и взаимодействия между различными компонентами системы.

Объекты на диаграммах последовательностей обозначаются прямоугольниками с подчеркнутыми именами, что помогает отличить их от классов. Сообщения, представляющие вызовы методов, изображаются линиями со стрелками, а возвращаемые результаты отображаются пунктирными линиями со стрелками. Это позволяет ясно видеть взаимодействие между объектами и последовательность выполнения операций.

Использование диаграмм последовательностей помогает разработчикам и аналитикам лучше понять, как взаимодействуют различные компоненты системы в рамках определенных сценариев использования. Это дает возможность выявить потенциальные проблемы и улучшить процессы взаимодействия между объектами [10].

На рисунке 1.11 представлена диаграмма последовательности процесса просмотра пользователем его заявок.

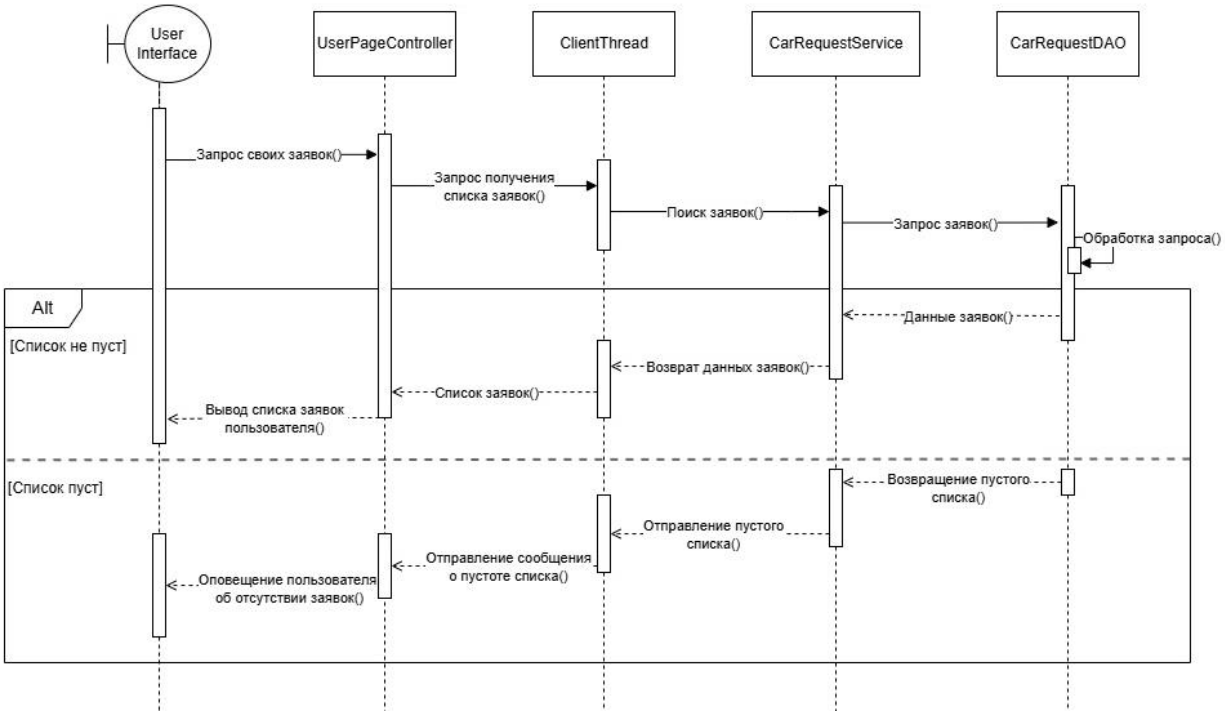

Рисунок 1.11 – Диаграмма последовательности отображения заявок пользователя

Диаграмма развертывания предназначена для визуализации элементов и компонентов системы, существующих лишь на этапе ее исполнения (runtime), к которым относятся исполнимые файлы, динамические библиотеки, таблицы БД и т.д. Те компоненты, которые не используются на этапе исполнения (например, исходные тексты программ), на диаграмме не показываются.

Основные цели, преследуемые при разработке диаграммы развертывания:

– распределение компонентов системы по ее физическим узлам;

– отображение физических связей между узлами системы на этапе исполнения;

– выявление узких мест системы и реконфигурация ее топологии для достижения требуемой производительности.

На рисунке 1.12 представлена UML-диаграмма развертывания, которая отображает конфигурацию компонентов приложения. Одним из узлов системы является база данных kursovayapsp, которая соединяется с сервером с помощью драйвера JDBC. Запускаемыми приложениями являются StartApplication на клиенте и Main на сервере.

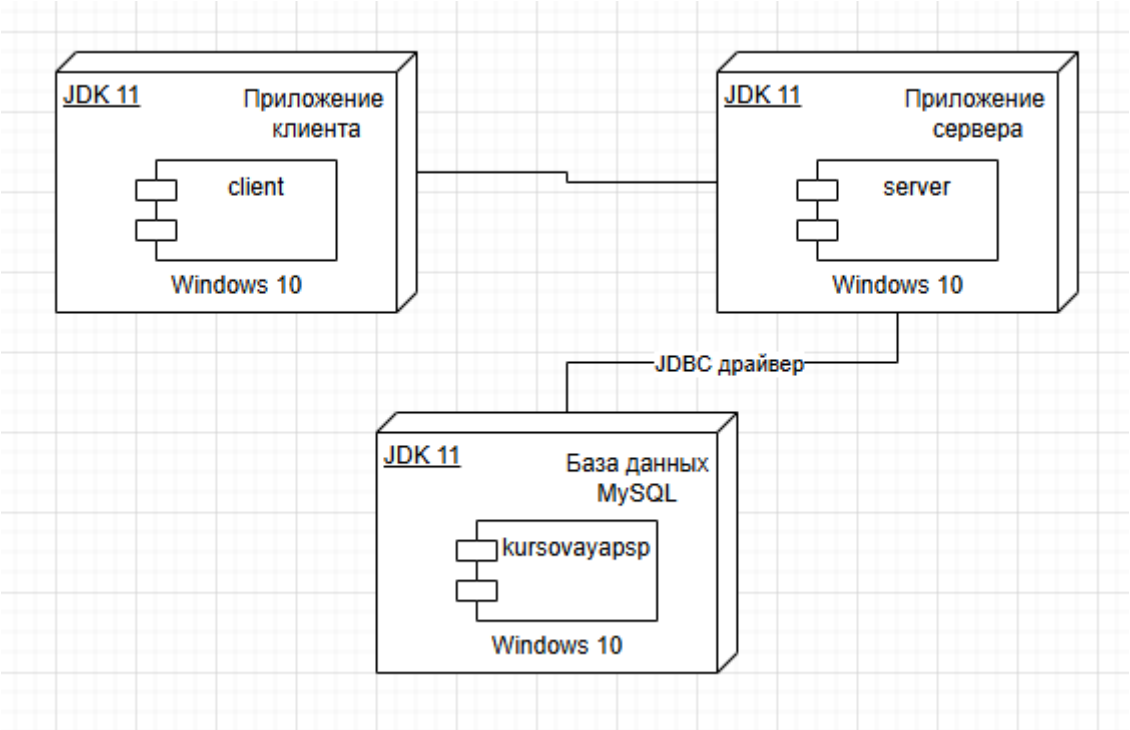

Рисунок 1.12 – Диаграмма развертывания системы программной поддержки деятельности автосалона

Диаграмма компонентов описывает особенности физического представления системы. Диаграмма компонентов позволяет определить архитектуру разрабатываемой системы, установив зависимости между программными компонентами, в роли которых может выступать исходный, бинарный и исполняемый код. Во многих средах разработки модуль или компонент соответствует файлу. Пунктирные стрелки, соединяющие модули, показывают отношения взаимозависимости, аналогичные тем, которые имеют место при компиляции исходных текстов программ. Основными графическими элементами диаграммы компонентов являются компоненты, интерфейсы и зависимости между ними.

Диаграмма компонентов разрабатывается для следующих целей:

– визуализации общей структуры исходного кода программной системы;

– спецификации исполнимого варианта программной системы;

– обеспечения многократного использования отдельных фрагментов программного кода;

– представления концептуальной и физической схем баз данных.

Диаграмма компонентов системы представлена на рисунке 1.13. Она отображает общую структуру исходного кода приложения.

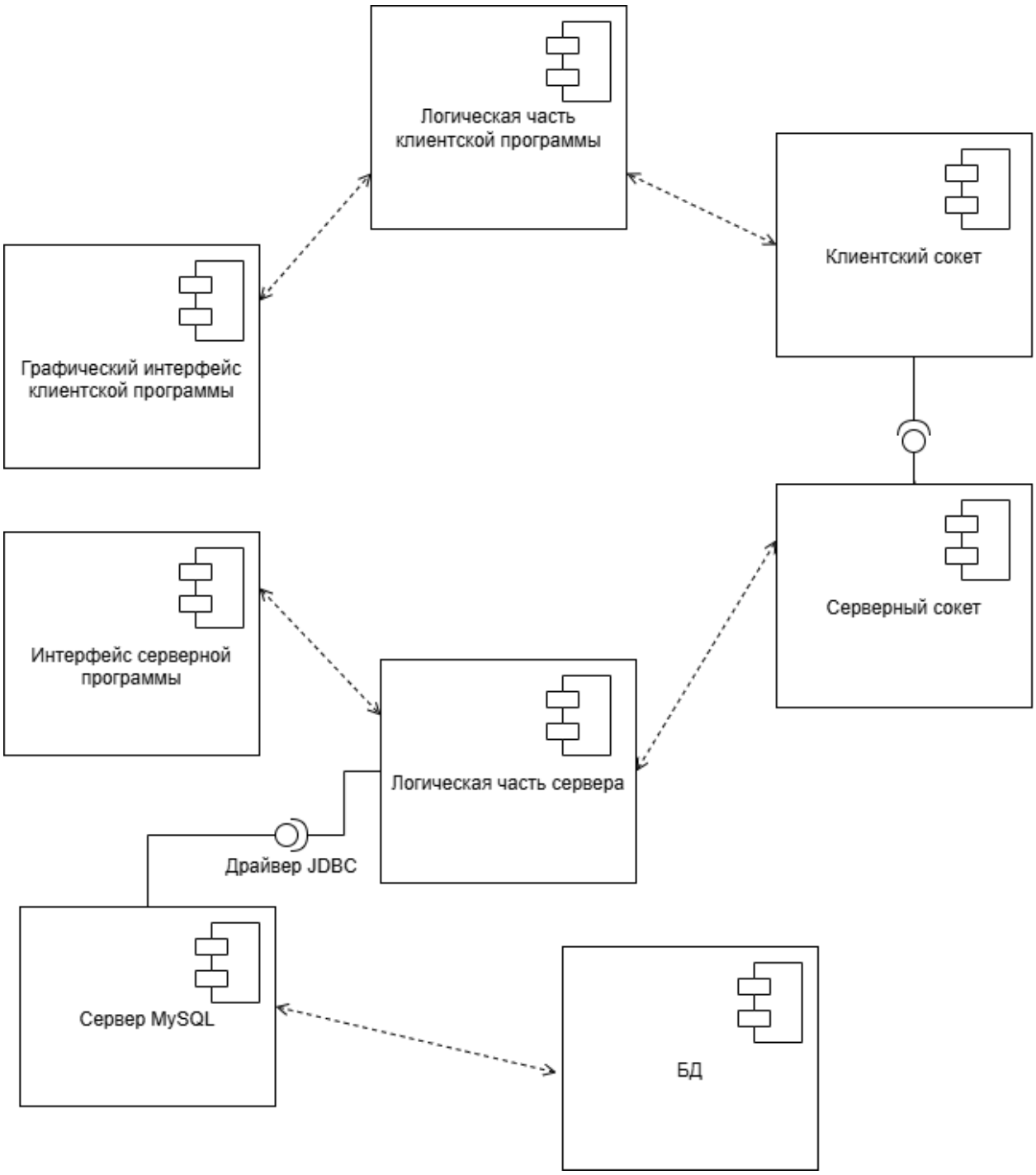

Рисунок 1.13 – Диаграмма компонентов системы

Диаграмма состояний — это один из ключевых инструментов в моделировании систем, используемый для отображения динамического поведения объектов в процессе их жизненного цикла. Она визуализирует различные состояния объекта и переходы между ними, что помогает понять, как объект реагирует на события и изменения. Диаграммы состояний широко используются в разработке программного обеспечения, особенно в таких областях, как системный анализ, проектирование и тестирование.

На рисунке 1.14 представлена диаграмма состояний тест-драйва в системе.

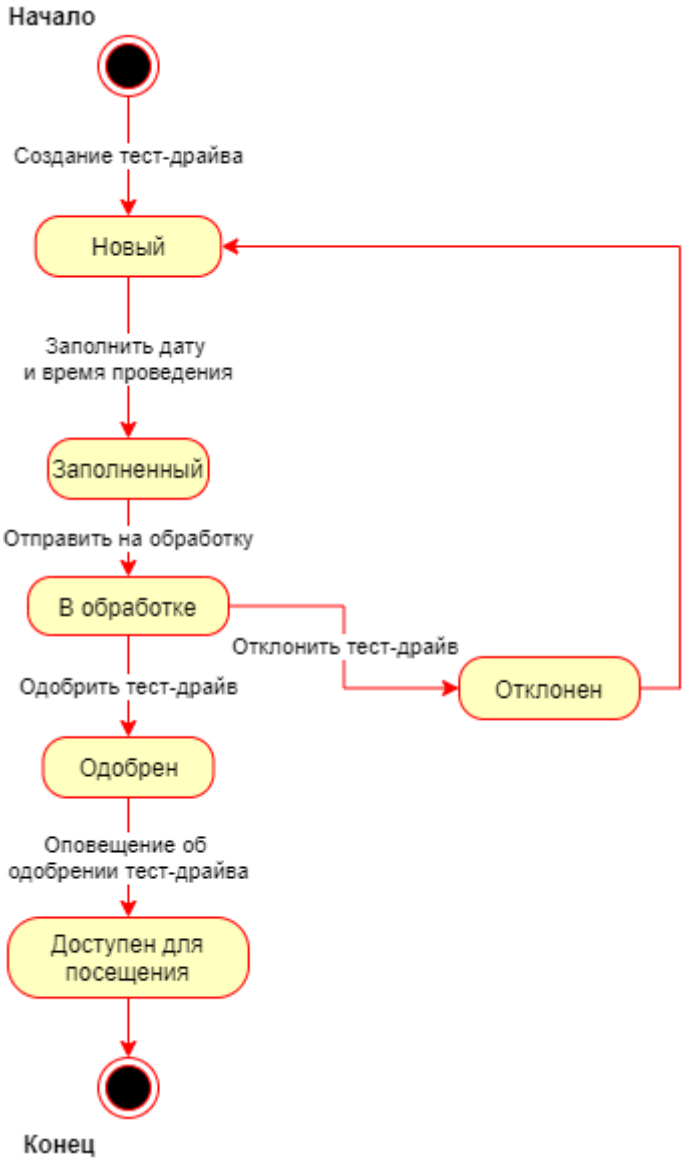

Рисунок 1.14 – Диаграмма состояний тест-драйва

На рисунках 1.15 – 1.20 представлены диаграммы классов системы.

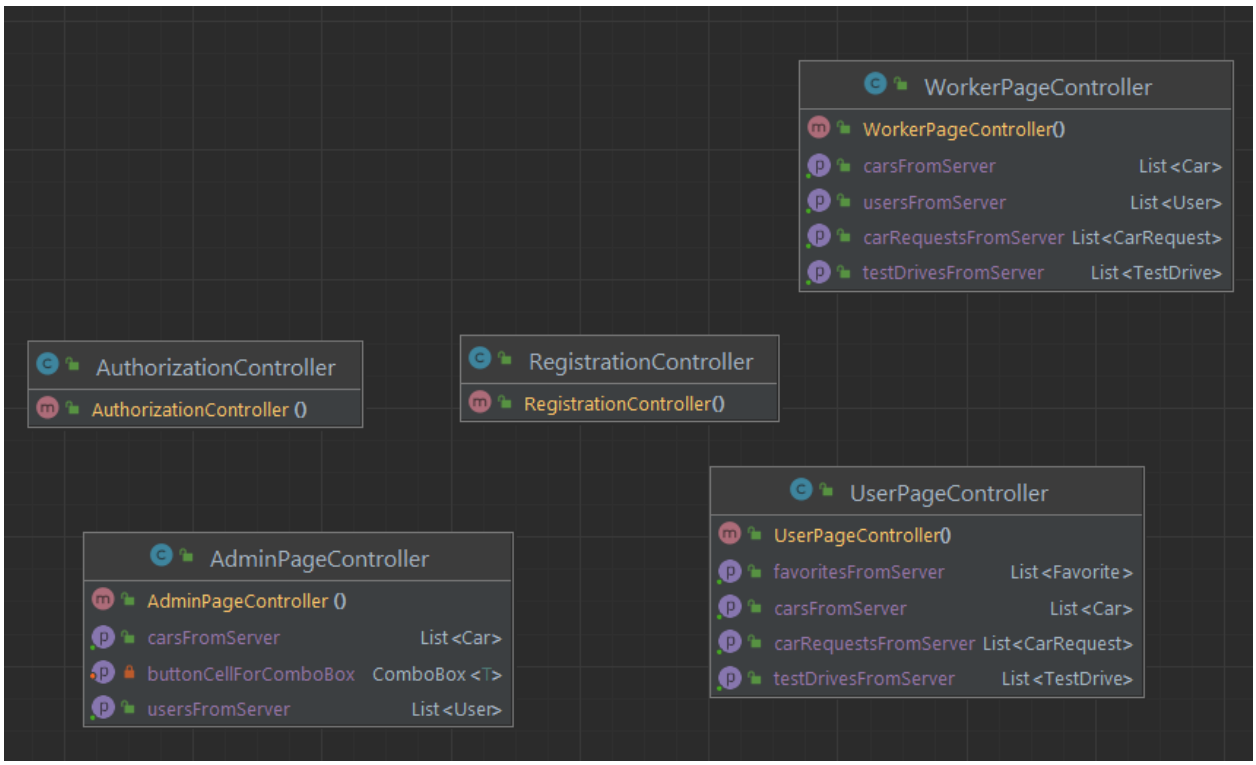

Рисунок 1.15 – Диаграмма классов контроллеров

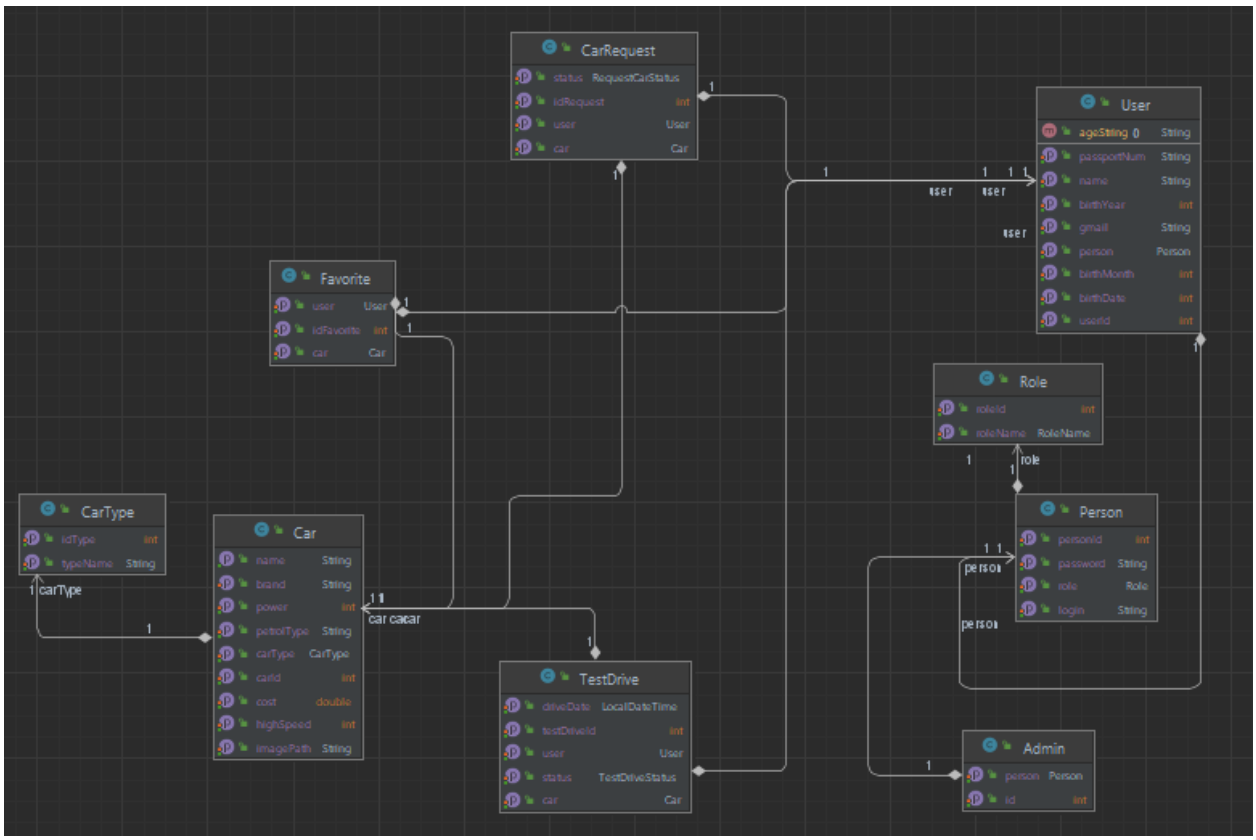

Рисунок 1.16 – Диаграмма классов сущностей

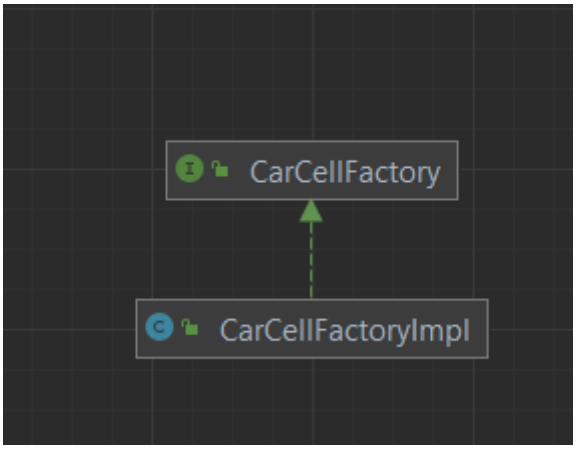

Рисунок 1.17 – Диаграмма классов паттерна фабрика

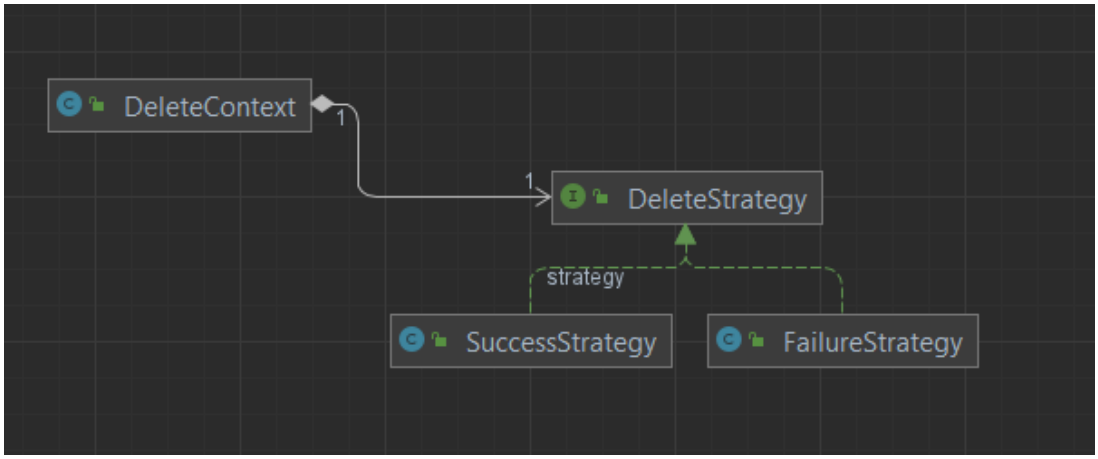

Рисунок 1.18 – Диаграмма классов паттерна стратегия

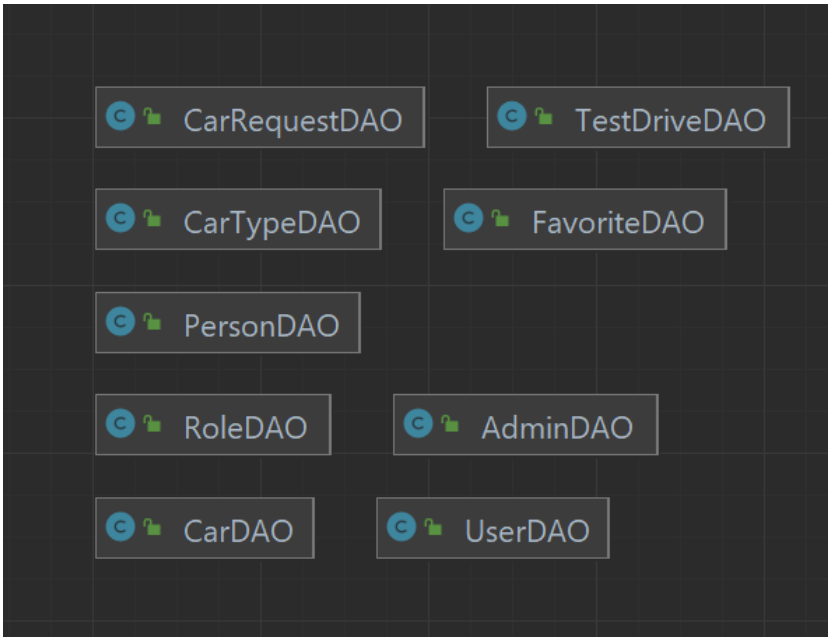

Рисунок 1.19 – Диаграмма классов DAO

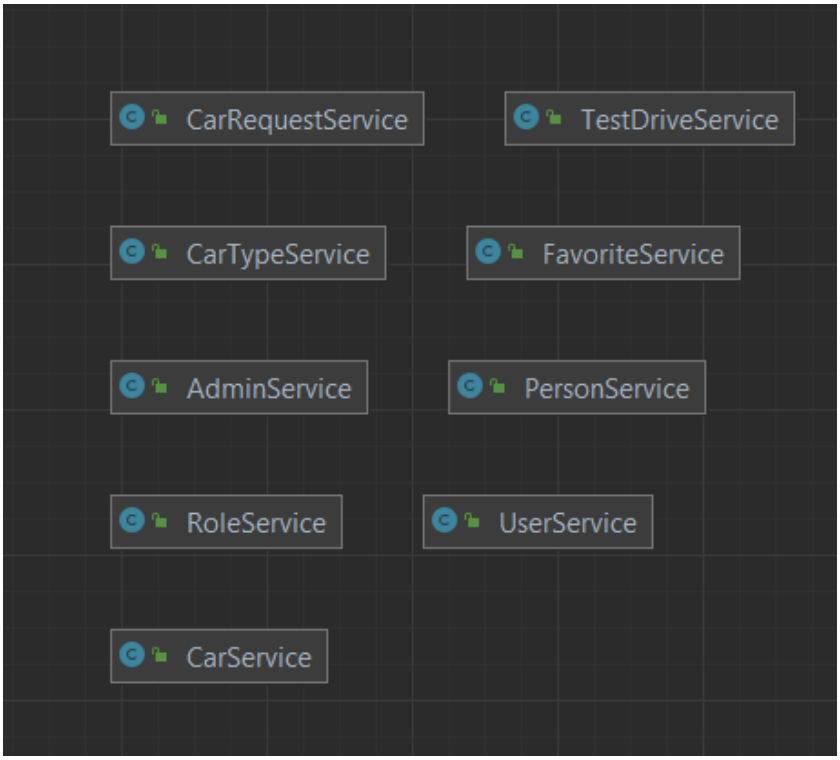

Рисунок 1.20 – Диаграмма классов Service

Таким образом, UML-диаграммы, которые на первый взгляд выглядят как набор фигур и стрелок, на самом деле помогают значительно упростить решение сложных задач в программировании, помочь при выборе оптимального решения и разработке технической документации.

Также в данной главе был проведен анализ предметной области, что позволило получить полное представление о системе и ее компонентах.

Была создана IDEF0 диаграмма, которая помогла визуализировать процессы и функциональность системы.

Также были проанализированы требования к разрабатываемому продукту, что позволило определить основные характеристики и функциональные возможности системы. Для более детального описания функциональных требований была разработана диаграмма вариантов использования, которая позволяет исследовать различные сценарии использования системы.

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

2.1 Постановка задачи

Автоматизация учета оформленных пользователям заявок предполагает использование современных информационных и коммуникационных технологий (ИКТ). Данная система охватывает полный цикл работы со всеми записями: от внесения данных об оформленных заявках до формирования отчётности. Она позволяет снизить ошибки, связанные с ручной обработкой данных и ускорить процесс работы с заявками.

Недостаточная автоматизация работы с заявками в автосалонах может создавать множество проблем, снижающих эффективность и увеличивающих риски в управлении клиентскими данными и процессами. Основные трудности включают:

1 Ручная обработка заявок. Без автоматизированной системы сотрудники вынуждены вручную регистрировать информацию о клиентах, транспортных средствах и других аспектах работы, что требует значительных временных затрат. Это повышает вероятность ошибок, таких как неправильное заполнение данных, что может привести к снижению уровня клиентского сервиса.

2 Отсутствие единой базы данных. Использование разрозненных систем или ведение учета в таблицах и на бумаге затрудняет доступ к актуальной информации о состоянии заявок, истории взаимодействий с клиентами и наличии автомобилей на складе. Это замедляет процессы обработки заявок и затрудняет координацию между отделами.

3 Сложности в управлении клиентской базой. Без автоматизированного решения трудно централизованно управлять информацией о клиентах, отслеживать историю их запросов и предпочтений, а также эффективно работать с повторными обращениями и рекомендациями.

Внедрение автоматизированной системы управления заявками предоставляет автосалонам ряд значительных преимуществ:

1 Автоматизация обработки заявок. Система фиксирует всю информацию о заявках и клиентских данных автоматически, что снижает вероятность ошибок и освобождает сотрудников от рутинных задач.

2 Единая база данных. Все сведения о клиентах, заявках и наличии автомобилей хранятся в централизованной базе, доступной в режиме реального времени. Это позволяет сотрудникам оперативно получать актуальную информацию и улучшает взаимодействие между отделами.

3 Инструменты для анализа и оптимизации. Система предоставляет аналитические возможности для оценки эффективности обработки заявок, анализа популярности моделей и оценки клиентских предпочтений. Это

помогает принимать обоснованные управленческие решения и улучшать сервис.

Для достижения поставленной цели будет разработано программное обеспечение, которое поможет сотрудникам без каких-либо отслеживать заявки и работать с ним. Программное обеспечение будет в себя включать клиентское приложение, которым будут пользоваться клиенты, работники и администратор, и также серверное приложение.

Серверное приложение будет реализовано в виде консольного приложения, в котором будет отображаться состояние сервера. Так как пользователей много, следовательно, серверное приложение должно быть в состоянии одновременно обрабатывать все запросы. Серверное приложение будет обрабатывать запросы на:

- авторизацию пользователей;
- регистрацию клиента;
- существование пользователя в системе;
- получение списка всех пользователей;
- выдачу роли;
- удаление пользователей;
- добавление автомобиля;
- удаление автомобиля;
- получение списка автомобилей;
- получение списка тест-драйвов;
- получение списка заявок;
- установление статуса заявки;
- регистрацию заявки;
- сохранение тест-драйва;
- добавление избранного автомобиля;
- получение избранных автомобилей;
- удаление тест-драйвов;
- удаление избранных автомобилей.

2.2 Архитектурные решения

Для хранения данных на сервере будет установлен сервер MySQL. База данных должна будет хранить следующие данные: информацию о пользователях, прошедших регистрацию, информацию о всех автомобилях системы, информацию о всех тест-драйвах и всю информацию о заявках.

Общение клиента с сервером будет осуществляться с использование сокетов и протокола TCP/IP. Общение клиента с базой данных будет осуществляться с помощью фреймворка Hibernate.

В программе будут выделены следующие сущности: пользователь (user), администратор(admin), тест-драйв (test_drive), автомобиль (cars), заявки на автомобили (car_requests), типы автомобилей (car_types), избранное (favorites),

персона (person), роль (role).

При написании программного обеспечения будут использоваться паттерны проектирования: Factory, Strategy и Adapter. Также использовалась схема MVC.

MVC (Model-view-controller) – схема использования нескольких шаблонов проектирования, с помощью которых модель приложения, пользовательский интерфейс и взаимодействие с пользователем разделены на три отдельных компонента таким образом, чтобы модификация одного из компонентов оказывала минимальное воздействие на остальные

Паттерн Адаптер (Adapter) используется для обеспечения совместимости между классами, которые по своей природе не могут работать вместе из-за различий в интерфейсах. Он создает "адаптер", который преобразует интерфейс одного класса в интерфейс, ожидаемый клиентом. Это позволяет использовать существующие классы без изменения их кода.

Паттерн Стратегия (Strategy) позволяет определить семейство алгоритмов, инкапсулировать их и сделать их взаимозаменяемыми. Это дает возможность изменять поведение объекта в зависимости от ситуации, не изменяя сам класс. Стратегия позволяет отделить алгоритм от его использования, что упрощает код и делает его более гибким [3].

Паттерн Фабрика (Factory) относится к созданию объектов. Он определяет интерфейс для создания объектов, но позволяет подклассам решать, какие классы инстанцировать. Фабрика скрывает детали создания объектов и предоставляет гибкость в изменении создаваемых объектов без изменения клиентского кода [4].

2.3 Описание алгоритмов, реализующих ключевую бизнес-логику разрабатываемого программного средства

Алгоритм работы многопоточного сервера. Рассмотрим алгоритм многопоточного сервера (см. Приложение В). Многопоточный сервер – это сервер который может одновременно обрабатывать запросы от нескольких клиентов. Ниже приведён пример кода, используемого в программе:

```
            serverSocket = new ServerSocket(PORT_NUMBER);
        while (true) {

            Iterator<Socket> iterator = currentSockets.iterator();
            while (iterator.hasNext()) {
                Socket socket = iterator.next();
                if (socket.isClosed()) {
                    continue;
                }
                String socketInfo = "Client: " + socket.getInetAddress() +
":" + socket.getPort();
                System.out.println(socketInfo);
            }

            Socket socket = serverSocket.accept();
            currentSockets.add(socket);
```

```
            clientHandler = new ClientThread(socket);
            thread = new Thread(clientHandler);
            thread.start();
            System.out.flush();
```

Код начинается с создания объекта ServerSocket, который прослушивает указанный порт PORT_NUMBER. Основной поток выполняет бесконечный цикл, обеспечивающий обработку подключений клиентов. В начале каждой итерации происходит проверка текущих подключений из списка currentSockets. С помощью итератора проверяется состояние каждого сокета, и если сокет закрыт, он пропускается. Для активных подключений выводится информация о клиенте, включая его IP-адрес и порт.

После обработки текущих подключений основной поток останавливается на вызове метода accept, ожидая нового подключения клиента. Когда клиент подключается, его сокет добавляется в список активных соединений, после чего создаётся экземпляр класса ClientThread для обработки запросов этого клиента. Для каждого нового соединения создаётся и запускается отдельный поток, отвечающий за взаимодействие с клиентом. Основной поток тем временем продолжает свою работу, возвращаясь к ожиданию новых подключений. Этот процесс повторяется до завершения программы.

Алгоритм обработки запроса на получение списка заявок клиентов. Рассмотрим алгоритм обработки одного из запросов работников, а именно получение списка заявок (см. Приложение В). Ниже приведён пример кода, используемого в программе:

```
      public List<CarRequest> getCarRequestsFromServer()
{
    List<CarRequest> carRequests = new ArrayList<>();

    Request request = new Request();
    request.setRequestMessage("");
    request.setRequestType(RequestType.GET_CAR_REQUESTS);
    try {
        ClientSocket.getInstance().getOut().println(new
Gson().toJson(request));
        ClientSocket.getInstance().getOut().flush();

        String responseJson = ClientSocket.getInstance().getIn().readLine();
        if (responseJson != null) {
            System.out.println("Response from server: " + responseJson);
            Response response = new Gson().fromJson(responseJson,
Response.class);

            if (response.getResponseStatus() == ResponseStatus.OK) {
                System.out.println("Successfully retrieved car requests.");
                Type carRequestListType = new TypeToken<List<CarRequest>>()
{}.getType();
                carRequests = new Gson().fromJson(new
Gson().toJson(response.getData()), carRequestListType);
} else {

 System.out.println("Failed to retrieve car requests: " +
response.getMessage());
```

```
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return carRequests;
}
```

Функция getCarRequestsFromServer() не принимает вхожных данных. Создается запрос на сервер, отправляется на сервер с типом запроса GET_CARS_REQUESTS. Далее получаем ответ от сервера и разбираем на части: если статус ответа OK, то проверяем на пустоту данные в response.data и если не пусто, то записываем данные в массив и возвращаем это значение.

Функция для записи на тест-драйв. Рассмотрим алгоритм обработки одного из запросов клиента, а именно, запись на тест-драйв (см. Приложение В). Ниже приведён пример кода, используемого в программе:

```
      private void handleTestDriveBooking(Car car)
{
    Stage modalStage = new Stage();
    modalStage.initModality(Modality.APPLICATION_MODAL);
    modalStage.setTitle("Запись на тест-драйв");

    VBox modalContainer = new VBox(10);
    modalContainer.setPadding(new Insets(20));
    modalContainer.setStyle("-fx-background-color: #282828; -fx-border-
radius: 10; -fx-background-radius: 10;");
    modalContainer.setAlignment(Pos.CENTER);

    Label headerLabel = new Label("Выберите дату и время для тест-драйва");
    headerLabel.setTextFill(Color.WHITE);
    headerLabel.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");

    DatePicker datePicker = new DatePicker();
    datePicker.setStyle("-fx-font-size: 14px;");
    datePicker.setDayCellFactory(param -> new DateCell() {
        @Override
        public void updateItem(LocalDate item, boolean empty) {
            super.updateItem(item, empty);
            if (item.isBefore(LocalDate.now().plusDays(7))) {
                setDisable(true);
                setStyle("-fx-background-color: #cccccc;");
            }
        }
    });

    TextField timeField = new TextField();
    timeField.setPromptText("Введите время (формат: 00:00)");
    timeField.setStyle("-fx-font-size: 14px;");

    Button bookButton = new Button("Записаться");
    bookButton.setStyle("-fx-font-size: 14px; -fx-font-weight: bold;");
    bookButton.setOnAction(e -> {
        if (datePicker.getValue() == null ||
timeField.getText().trim().isEmpty()) {
```

```
            Alert errorAlert = new Alert(Alert.AlertType.ERROR);
            errorAlert.setTitle("Ошибка");
            errorAlert.setHeaderText(null);
            errorAlert.setContentText("Пожалуйста, заполните все поля!");
            errorAlert.showAndWait();
            return;
        }

        String timePattern = "^([01]\\d|2[0-3]):[0-5]\\d$";
        if (!timeField.getText().matches(timePattern)) {
            Alert errorAlert = new Alert(Alert.AlertType.ERROR);
            errorAlert.setTitle("Ошибка");
            errorAlert.setHeaderText(null);
            errorAlert.setContentText("Неправильный формат времени! Введите
время в формате 00:00.");
            errorAlert.showAndWait();
            timeField.clear();
            return;
        }
        LocalDate selectedDate = datePicker.getValue();
        String time = timeField.getText();


        LocalDateTime driveDate = combineDateAndTime(selectedDate,time);
        TestDrive testDrive = new
TestDrive(car,Session.getUser(),driveDate,TestDriveStatus.NONE);

        boolean checker = sendTestDriveToServer(testDrive, SAVE_TEST_DRIVE);
        if (checker) {
            showSuccessAlert("Успешная запись", "Успешная запись на тест
драйв");
            modalStage.close();
        } else {
            showAlert("Ошибка", "Произошла ошибка при записи.");
        }
    });

    modalContainer.getChildren().addAll(headerLabel, datePicker, timeField,
bookButton);

    Scene modalScene = new Scene(modalContainer, 400, 300);
    modalStage.setScene(modalScene);
    modalStage.showAndWait();
}
```

Пользователь подтверждает желание записаться на тест-драйв. Далее в появившемся модальном окне заполняет данные даты и времени, удобные для него, далее после проверки на корректность данных объект отправляется на сервер. В случае положительного ответа от сервера, пользователь будет оповещен об успешности записи на тест-драйв.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс – это совокупность информационной модели проблемной области, средств и способов взаимодействия пользователя с информационной моделью, а также компонентов, обеспечивающих формирование информационной модели в процессе работы программной системы.

Под информационной моделью понимается условное представление проблемной области, формируемое с помощью компьютерных объектов, отражающих состав и взаимодействие реальных компонентов проблемной области.

Средства и способы взаимодействия с информационной моделью определяются составом аппаратного и программного обеспечения, имеющегося в распоряжении пользователя, и от характера решаемой задачи. Эффективность работы пользователя определяется не только функциональными возможностями имеющихся в его распоряжении аппаратных и программных средств, но и доступностью для пользователя этих возможностей. В свою очередь, полнота использования потенциальных возможностей имеющихся ресурсов зависит от качества пользовательского интерфейса.

Качество пользовательского интерфейса является самостоятельной характеристикой программного продукта, сопоставимой по значимости с такими его показателями, как надежность и эффективность использования вычислительных ресурсов.

Пользовательский интерфейс состоит из следующих основных компонентов, представленных в виде айсберга – подачи информации пользователю, взаимодействию и взаимосвязям между объектами. При этом «видимая» часть айсберга значительно меньше его «невидимой», скрытой части**.** Верх айсберга – информация для пользователей (цвет, анимация, звук, форма объектов, расположение информации на экране, графика) составляет всего 10% и является отнюдь не самой важной составляющей пользовательского интерфейса. Следующая часть пользовательского интерфейса (30% модели проектировщика) – это техника общения с пользователем и обратная связь с ним. И, наконец, нижняя часть айсберга (60%) модели проектировщика – это свойства объектов и связи между ними.

Основное достоинство хорошего интерфейса пользователя заключается в том, что пользователь всегда чувствует, что он управляет программным обеспечением, а не программное обеспечение управляет им**.**

Для создания у пользователя такого ощущения интерфейс должен обладать целым рядом свойств.

Естественность интерфейса. Естественный интерфейс – такой, который не вынуждает пользователя существенно изменять привычные для него способы решения задачи. Это означает, что сообщения и результаты, выдаваемые приложением, не должны требовать дополнительных пояснений.

Согласованность интерфейса. Согласованность позволяет пользователям переносить имеющиеся знания на новые задания, осваивать новые аспекты быстрее, и благодаря этому фокусировать внимание на решаемой задаче, а не тратить время на уяснение различий в использовании тех или иных элементов управления, команд и т.д. Обеспечивая преемственность полученных ранее знаний и навыков, согласованность делает интерфейс узнаваемым и предсказуемым.

Дружественность интерфейса. Пользователи обычно изучают особенности работы с новым программным продуктом методом проб и ошибок. Эффективный интерфейс должен принимать во внимание такой подход. На каждом этапе работы он должен разрешать только соответствующий набор действий и предупреждать пользователей о тех ситуациях, где они могут повредить системе или данным; еще лучше, если у пользователя существует возможность отменить или исправить выполненные действия.

Даже при наличии хорошо спроектированного интерфейса пользователи могут делать те или иные ошибки. Эти ошибки могут быть как «физического» типа (случайный выбор неправильной команды или данных), так и «логического» (принятие неправильного решения на выбор команды или данных). Эффективный интерфейс должен позволять предотвращать ситуации, которые, вероятно закончатся ошибками. Он также должен уметь адаптироваться к потенциальным ошибкам пользователя и облегчать ему процесс устранения последствий таких ошибок.

Простота интерфейса. Интерфейс должен быть простым. При этом имеется в виду обеспечение легкости в его изучении и в использовании. Кроме того, он должен предоставлять доступ ко всему перечню функциональных возможностей, предусмотренных данным приложением. Реализация доступа к широким функциональным возможностям и обеспечение простоты работы противоречат друг другу. Разработка эффективного интерфейса призвана сбалансировать эти цели.

Гибкость интерфейса. Гибкость интерфейса – это его способность учитывать уровень подготовки и производительность труда пользователя. Свойство гибкости предполагает возможность изменения структуры диалога и/или входных данных.

Эстетическая привлекательность. Корректное визуальное представление используемых объектов обеспечивает передачу весьма важной дополнительной информации о поведении и взаимодействии различных объектов. В то же время следует помнить, что каждый визуальный элемент, который появляется на экране, потенциально требует внимания пользователя, которое не безгранично [6].

В данном приложении для разработки пользовательского интерфейса использовались элементы библиотеки JavaFX.

2.5 Обоснование выбора компонентов и технологий для реализации программного средства

Использование клиент-серверного подхода имеет ряд преимуществ. Главное из которых – снижение сетевого трафика при выполнении запросов Например, при необходимости выбора пяти записей из таблицы, содержащей миллион, клиентское приложение посылает серверу запрос, который сервером компилируется и выполняется, после чего результат запроса (те самые пять записей, а вовсе не вся таблица) передается обратно на рабочую станцию.

Вторым преимуществом архитектуры клиент-сервер является возможность хранения бизнес-правил на сервере, что позволяет избежать дублирования кода в различных приложениях, использующих общую базу данных. Кроме того, в этом случае любое редактирование данных, в том числе и редактирование нештатными средствами, может быть произведено только в рамках этих правил.

Итак, клиент-серверная информационная система состоит в простейшем случае из трех основных компонентов:

– сервер баз данных, управляющий хранением данных, доступом и защитой, резервным копированием, отслеживающий целостность данных в соответствии с бизнес-правилами и, самое главное, выполняющий запросы клиента;

– клиент, предоставляющий интерфейс пользователя, выполняющий логику приложения, проверяющий допустимость данных, посылающий запросы к серверу и получающий ответы от него;

– сеть и коммуникационное программное обеспечение, осуществляющее взаимодействие между клиентом и сервером посредством сетевых протоколов.

По заданию проекта должен использоваться объектно-ориентированный язык *Java*.

Приложения *Java* обычно транслируются в специальный байт-код, поэтому они могут работать на любой виртуальной *Java*-машине вне зависимости от компьютерной архитектуры [6].

Достоинством подобного способа выполнения программ является полная независимость байт-кода от операционной системы и оборудования, что позволяет выполнять *Java*-приложения на любом устройстве, для которого существует соответствующая виртуальная машина. Другой важной особенностью технологии *Java* является гибкая система безопасности, в рамках которой исполнение программы полностью контролируется виртуальной машиной. Любые операции, которые превышают установленные полномочия программы (например, попытка несанкционированного доступа к данным или соединения с другим компьютером), вызывают немедленное прерывание [7].

Часто к недостаткам концепции виртуальной машины относят снижение производительности. Ряд усовершенствований несколько увеличил скорость выполнения программ на *Java*:

– применение технологии трансляции байт-кода в машинный код непосредственно во время работы программы (JIT-технология) с возможностью сохранения версий класса в машинном коде;

– широкое использование платформенно-ориентированного кода (native-код) в стандартных библиотеках;

– аппаратные средства, обеспечивающие ускоренную обработку байт-кода (например, технология Jazelle, поддерживаемая некоторыми процессорами фирмы ARM) [5].

Необходимо предусмотреть разделение ролей: наличие администратора, который руководит деятельностью пользователей, и наличие пользователей(клиентов), которые приобретают товары. Информация о пользователях программы хранится в базе данных.

В качестве СУБД мною была выбрана *MySQL* 8.0 – свободная реляционная система управления базами данных. Разработку и поддержку *MySQL* осуществляет корпорация *Oracle*. *MySQL* является решением для малых и средних приложений. Гибкость СУБД *MySQL* обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа *MyISAM*, поддерживающие полнотекстовый поиск, так и таблицы *InnoDB*, поддерживающие транзакции на уровне отдельных записей. Более того, СУБД *MySQL* поставляется со специальным типом таблиц *EXAMPLE*, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой архитектуре и *GPL*-лицензированию, в СУБД *MySQL* постоянно появляются новые типы таблиц [8].

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

Рассмотрим различные исключительные ситуации, которые могут возникнуть при использовании разработанного программного обеспечения. Все исключительные ситуации обработаны.

Рассмотрим, например, ситуацию, когда клиент попытается зарегистрироваться в системе, а его возраст меньше 18 (рисунок 3.1).

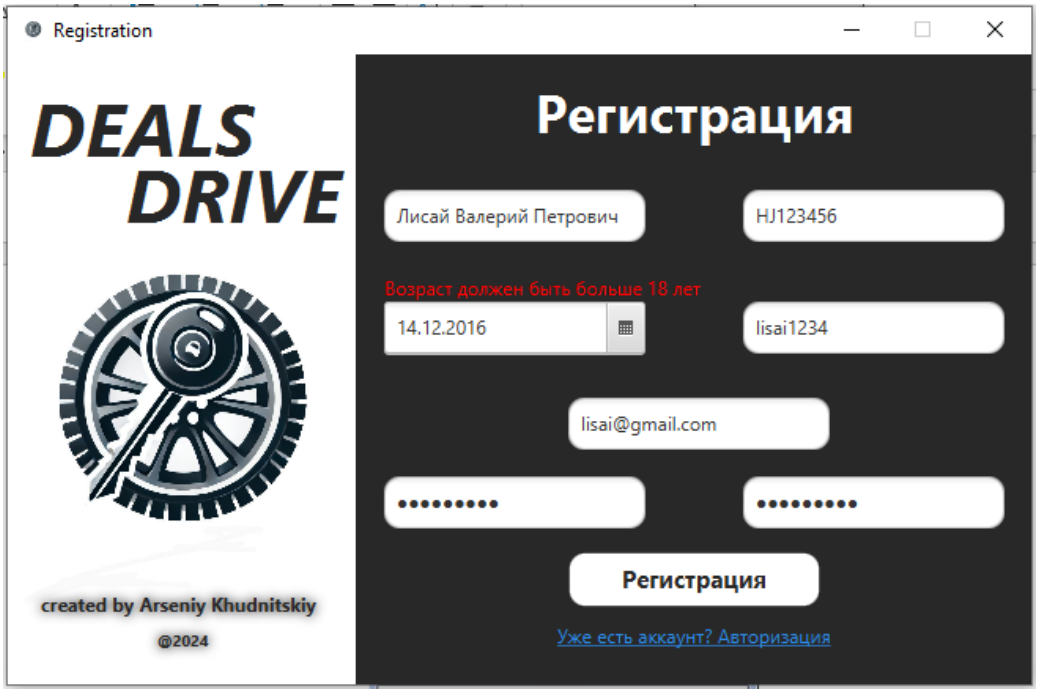

Рисунок 3.1 – Окно регистрации в случае регистрации пользователя, не достигшего 18 лет

Также обработаны ситуации в случае пустых полей (рисунок 3.2), а также несоответствия введенных данных форматам (рисунок 3.3).

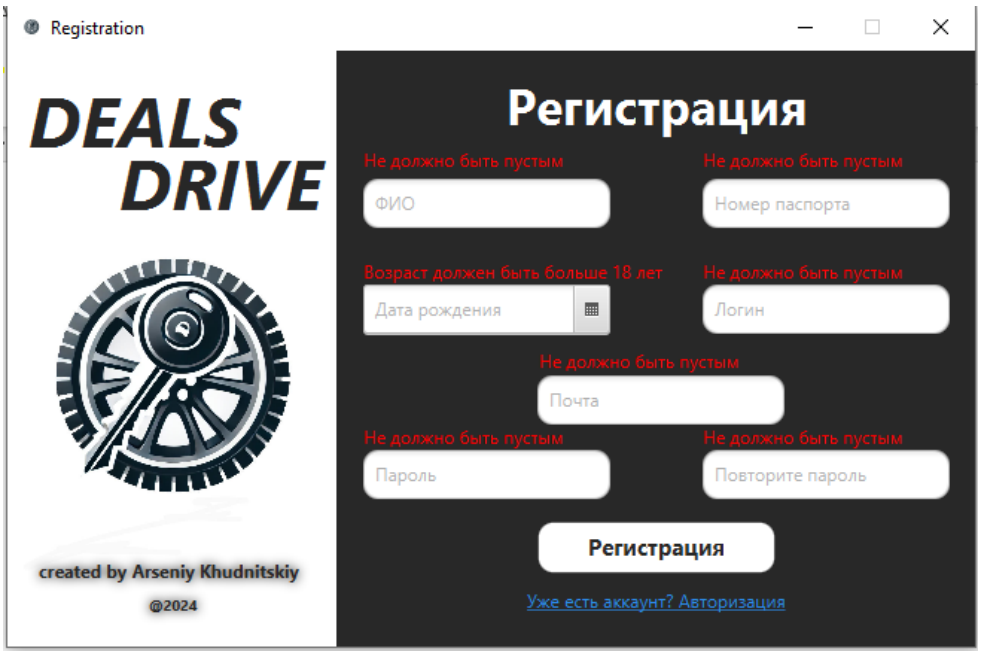

Рисунок 3.2 – Окно регистрации в случае пустых полей

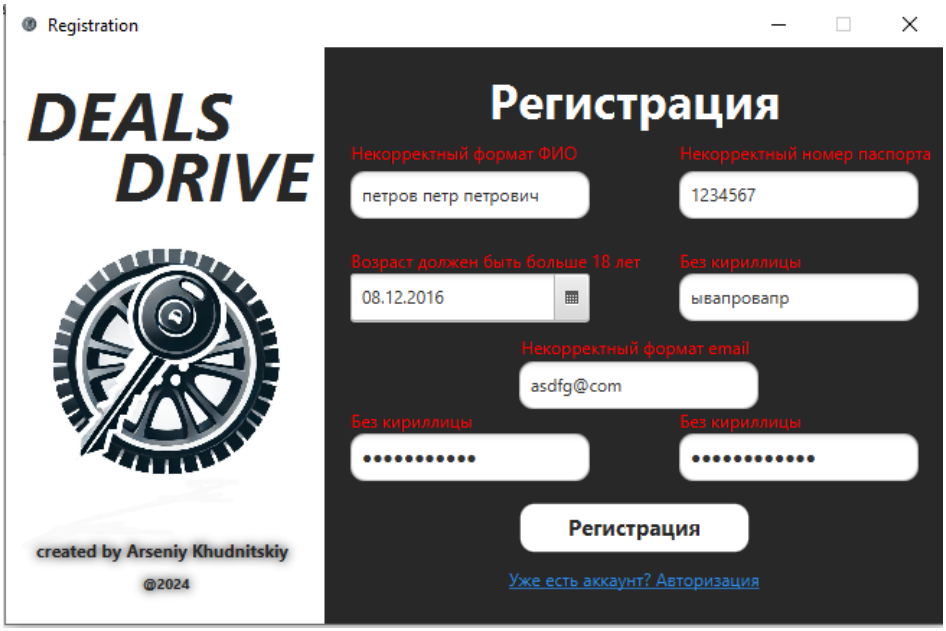

Рисунок 3.3 – Окно регистрации в случае некорректных форматов данных

Такие же проверки существуют и в окне авторизации. Также в окне авторизации, случае если в бд не будет найден пользователь с такими данными, будет выдано соответствующее оповещение (рисунок 3.4).

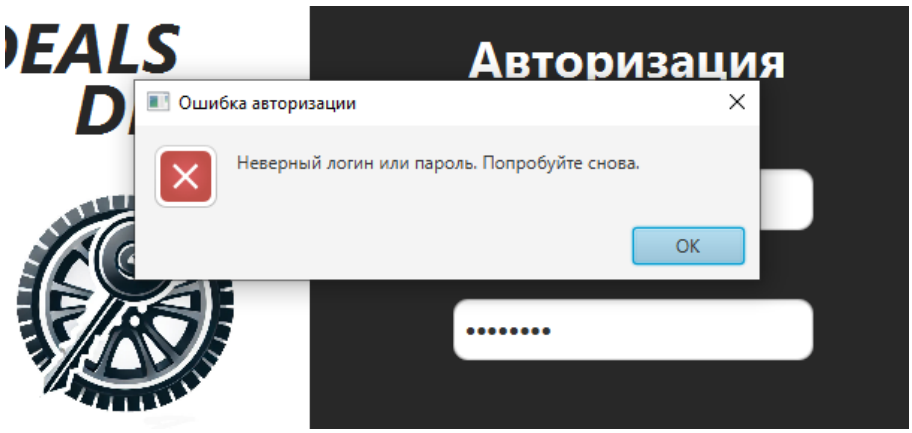

Рисунок 3.4 – Диалоговое окно, при неудачной авторизации пользователя

Также может быть исключительная ситуация в случае добавления авто администратором. Могут быть некорректно введенные данные (рисунок 3.5) или пустые поля (рисунок 3.6).

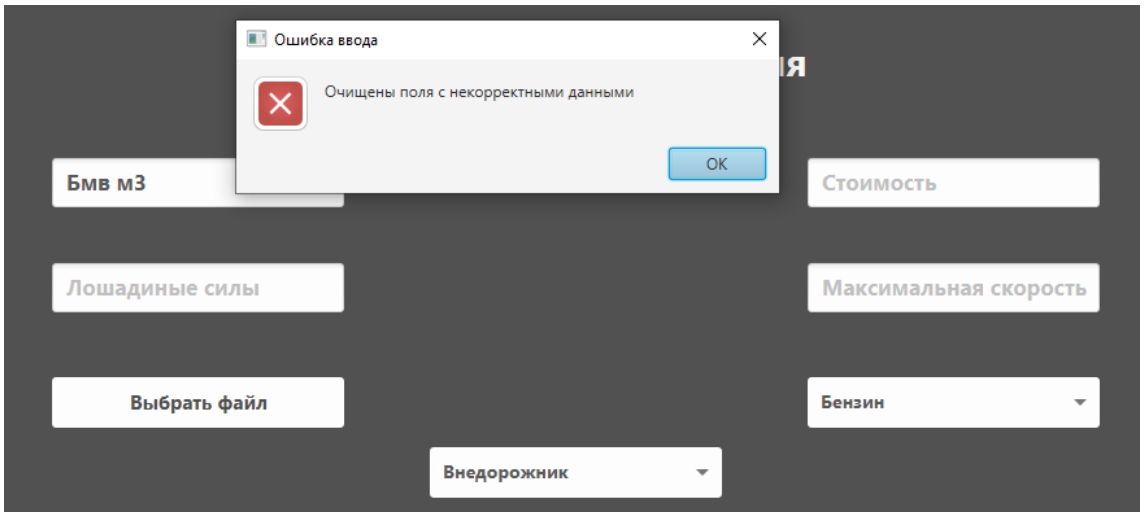

Рисунок 3.5 – Диалоговое окно, в случае некорректно введенных данных для автомобиля

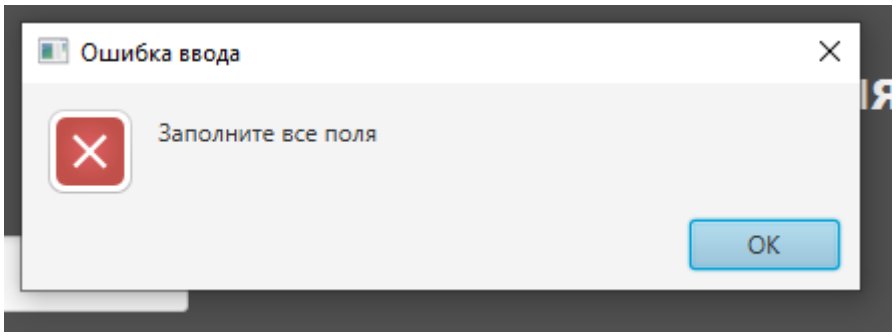

Рисунок 3.5 – Диалоговое окно, в случае пустых полей при введении данных об автомобиле

Также может быть ситуация, что администратор решил выдать роли работников каким- пользователям, однако никого не выбрал. В этом случае будет выдано оповещение (рисунок 3.6). Аналогично будет выглядеть оповещение, если администратор не выбрал автомобили для удаления или сотрудников для увольнения.

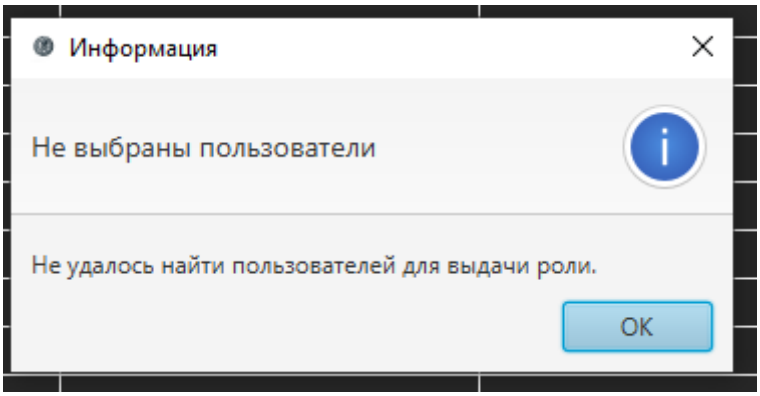

Рисунок 3.6 – Диалоговое окно при отсутствии выбранных пользователей для выдачи роли

Так же обрабатываются исключительные ситуации для пользователей. В случае если пользователь заходит просмотреть свои избранные автомобили, изучать свои заявки или отследить статус свои тест-драйвов, однако у него нет избранных автомобилей, нет заявок, а также он не записывался на тест-драйв. В этом случае пользователь также увидит, что в этих панелях пусто (рисунок 3.7, рисунок 3.8).

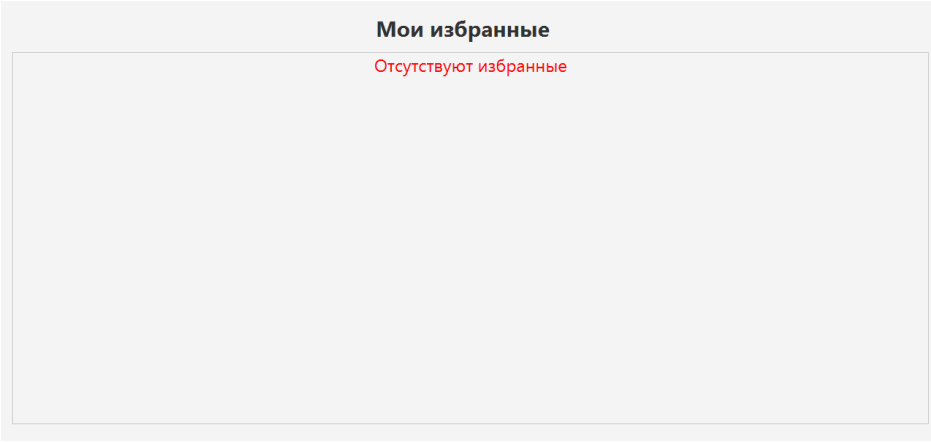

Рисунок 3.7 – Оповещение пользователя об отсутствии избранных автомобилей

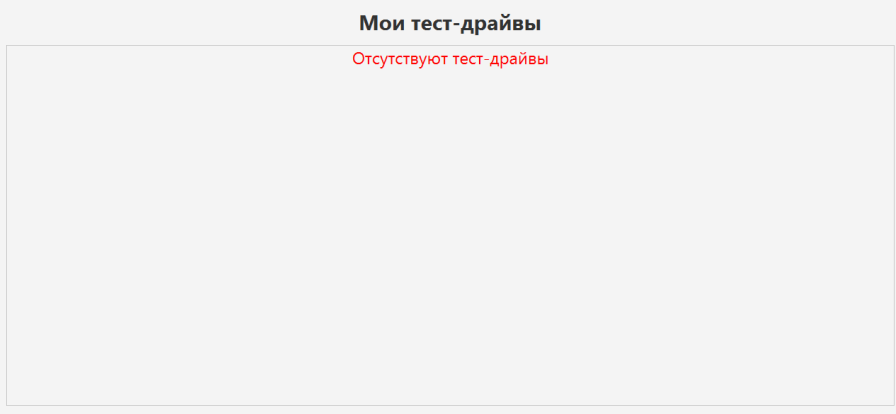

Рисунок 3.8 – Оповещение пользователя об отсутствии записей на тест-драйвы

Аналогично будет выглядеть оповещение в случае, если у пользователя отсутствуют заявки на автомобили.

Также используем модульное тестирование для проверки функциональности класса UserService с помощью фреймворка JUnit. Модульное тестирование позволяет изолировать отдельные компоненты приложения и проверять их поведение в различных условиях.

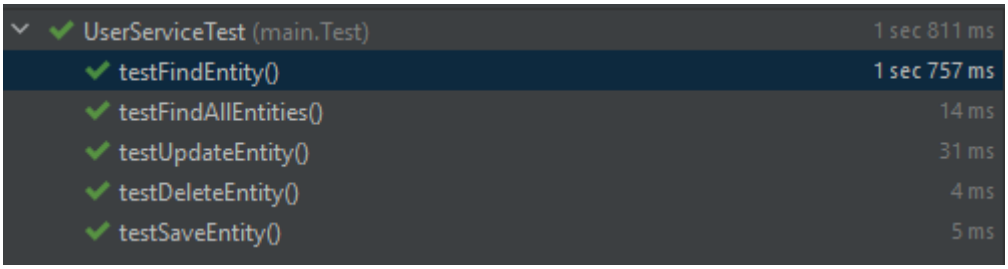

Рисунок 3.9 – Успешное модульное тестирование

В заключение можно сказать, что в программном средстве предусмотрена обработка любых ошибок. Все пользователи системы получают оповещения в случае некорретных действий и введении неверных данных. Также обработаны все исключительные ситуации и проведены тесты отдельных модулей.

4 ИНСТРУКЦИЯ ПО РАЗВЕРТЫВАНИЮ ПРИЛОЖЕНИЯ И СКВОЗНОЙ ТЕСТОВЫЙ ПРИМЕР, НАЧИНАЯ ОТ АВТОРИЗАЦИИ, ДЕМОНСТРИРУЯ РЕАЛИЗАЦИЮ ВСЕХ ВАРИАНТОВ ИСПОЛЬЗОВАНИЯ

Стартовым окном нашего приложения является окно авторизации. На рисунке 4.1 показан скриншот этого окна. Здесь пользователь может авторизоваться или перейдя по ссылке зарегистрироваться (рисунок 4.2).

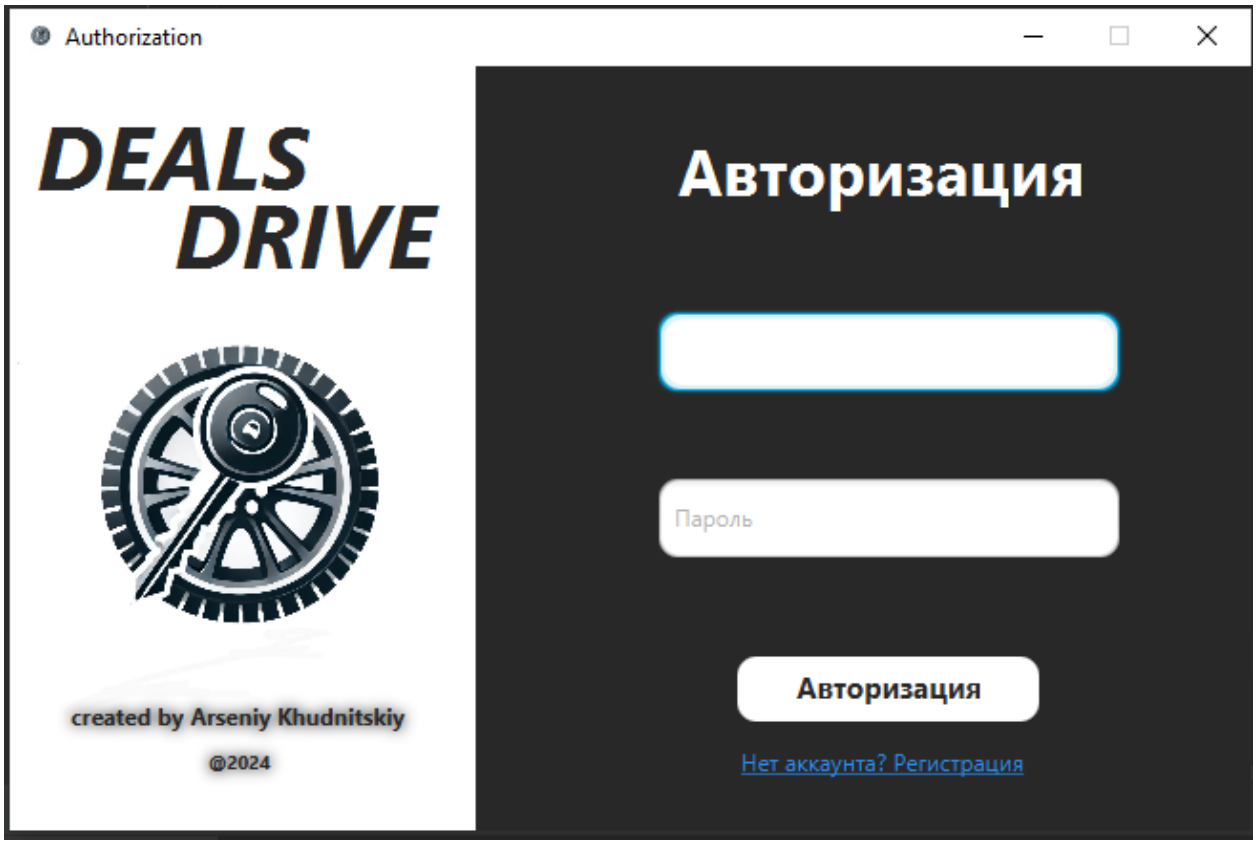

Рисунок 4.1 – Окно авторизации

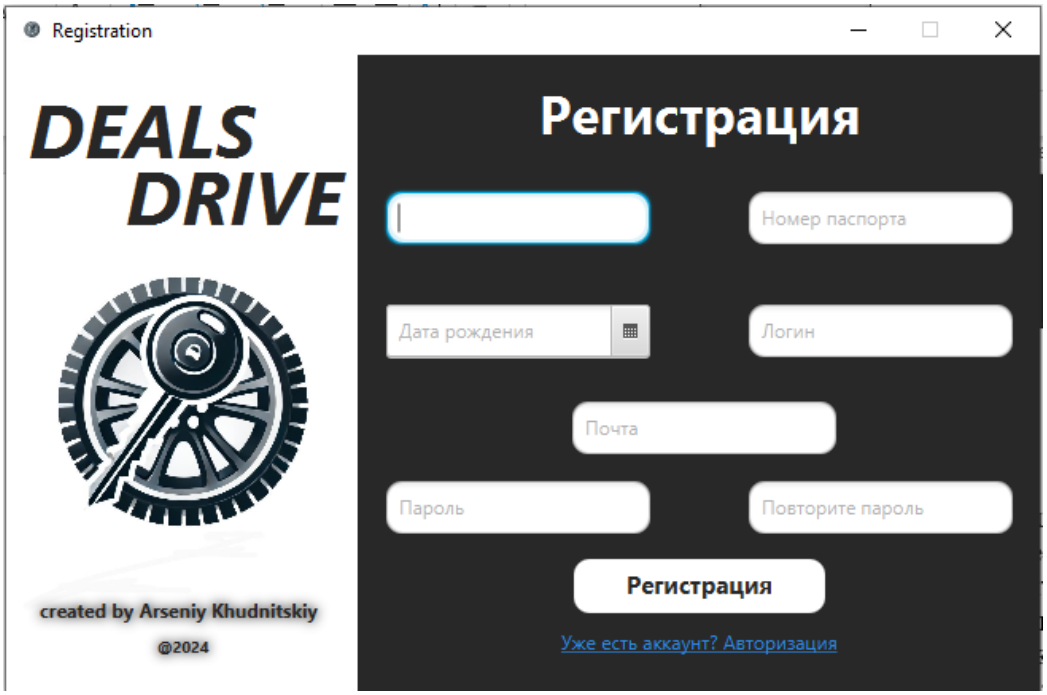

Рисунок 4.2 – Окно регистрации

Рассмотрим панель администратора. Скриншот данного окна показан на рисунке 4.3. Здесь администратор может добавлять новое авто в систему, удалять авто из системы, выдавать пользователю роль сотрудника, а также увольнять сотрудника. Сверху своей панели ему, как и любому пользователю доступен просмотр информации о компании, просмотр автомобилей системы с возможностью сортировки, а также выход из аккаунта и просмотр своего профиля.

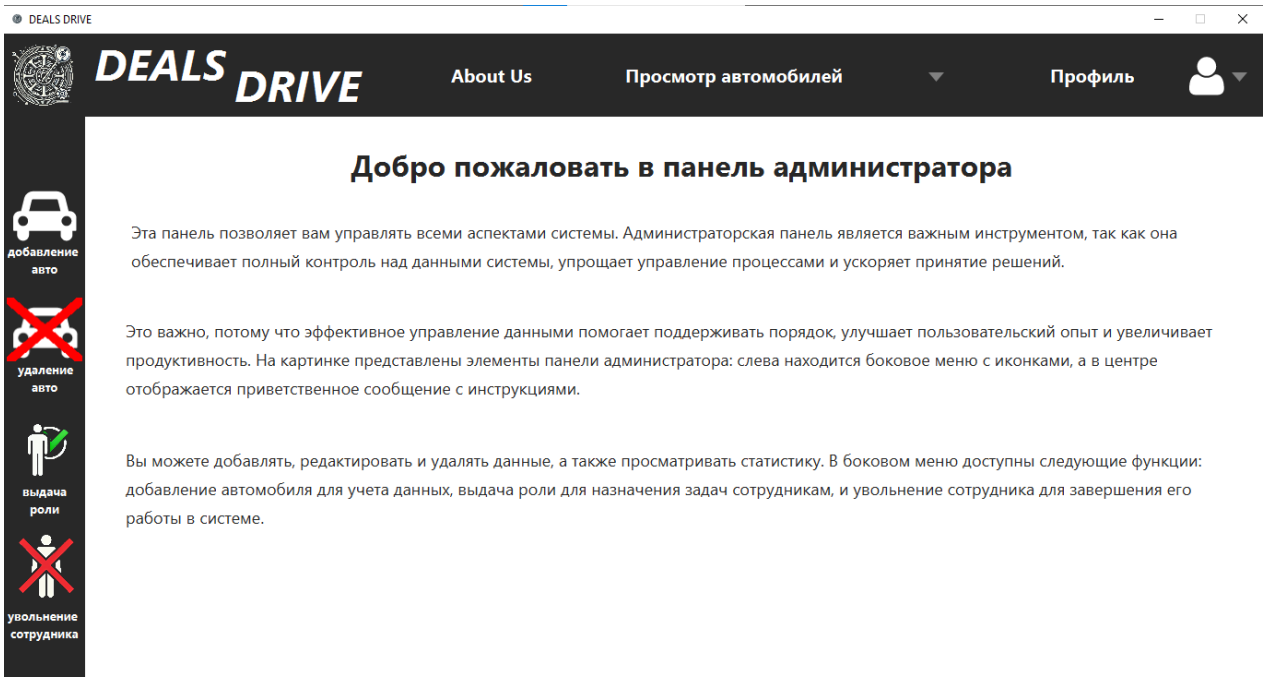

Рисунок 4.3 – Стартовая панель администратора

Далее конкретные функции, которые доступны админу. Мы рассмотрим функцию добавления автомобиля в систему. Скриншот рассматриваемого окна показан на рисунке 4.4. Администратор вносит корректные данные об автомобиле указывая все характеристики и выбирает нужное фото. После этого нажимает кнопку добавить и происходит добавление автомобиля в систему (рисунок 4.5).

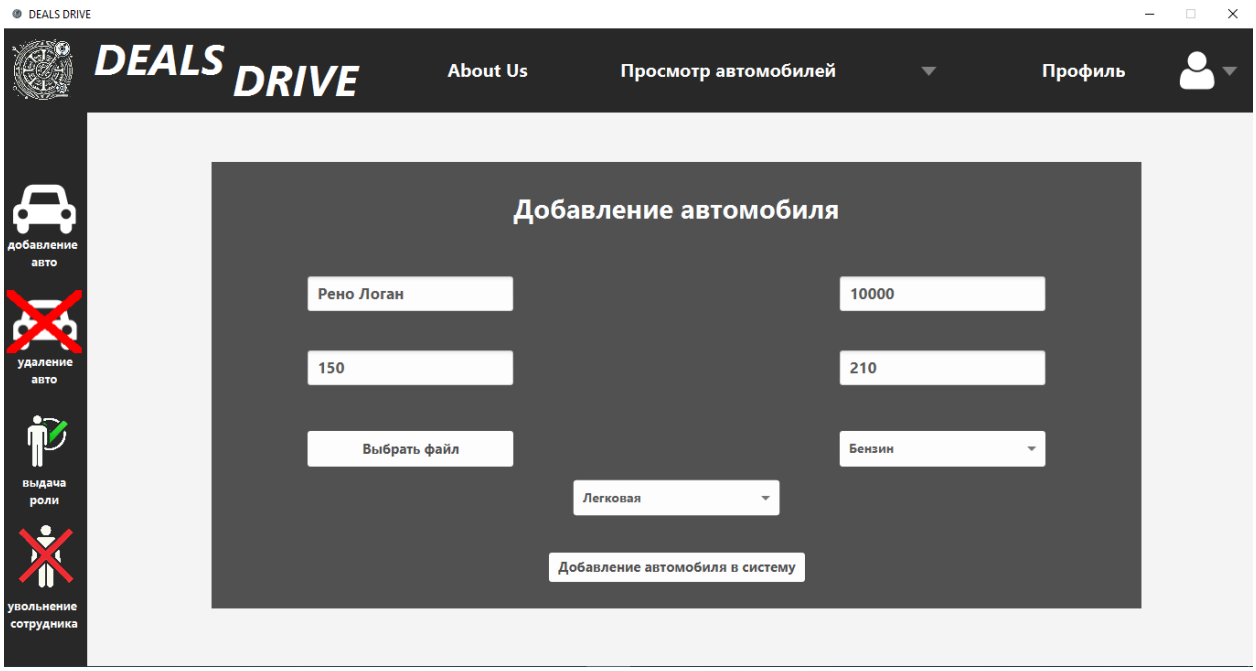

Рисунок 4.4 – Панель добавления

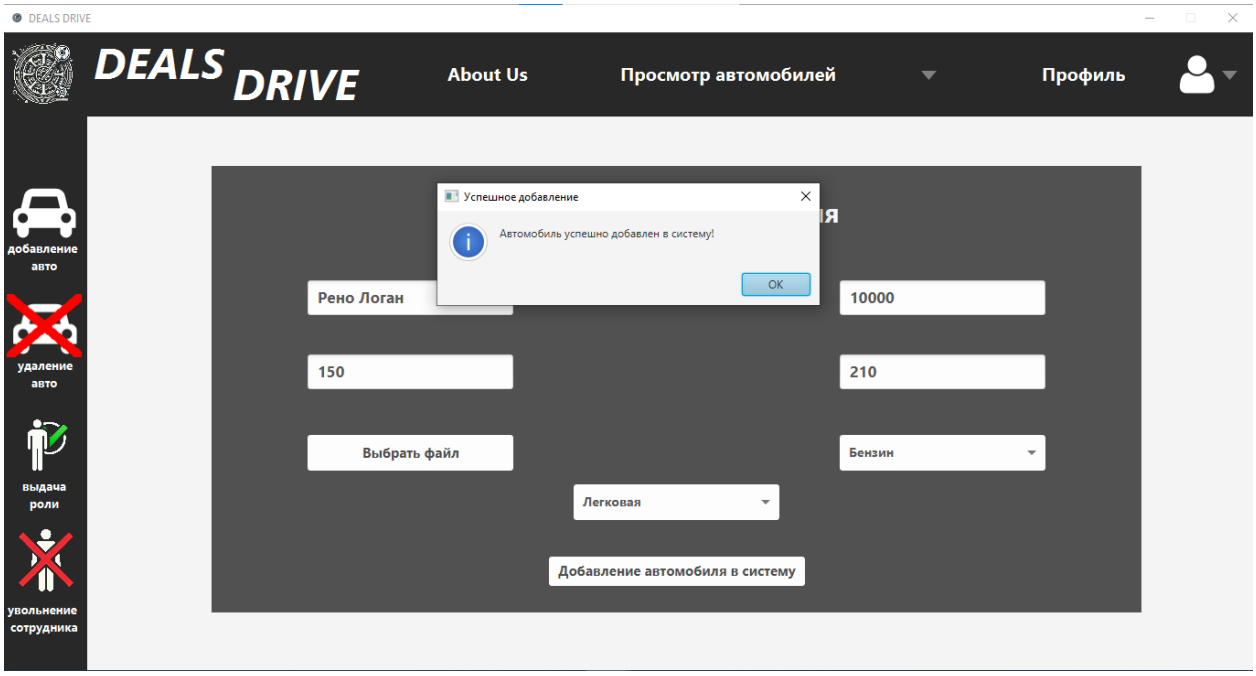

Рисунок 4.5 – Оповещение об успешности добавления в систему

Скриншот окна удаления автомобиля из системы на рисунке 4.6. Администратору для этого нужна на выбранном автомобиле нажать кнопку удалить, расположенную в одной строке с конкретным авто. После этого получит уведомление об успешности удаления.

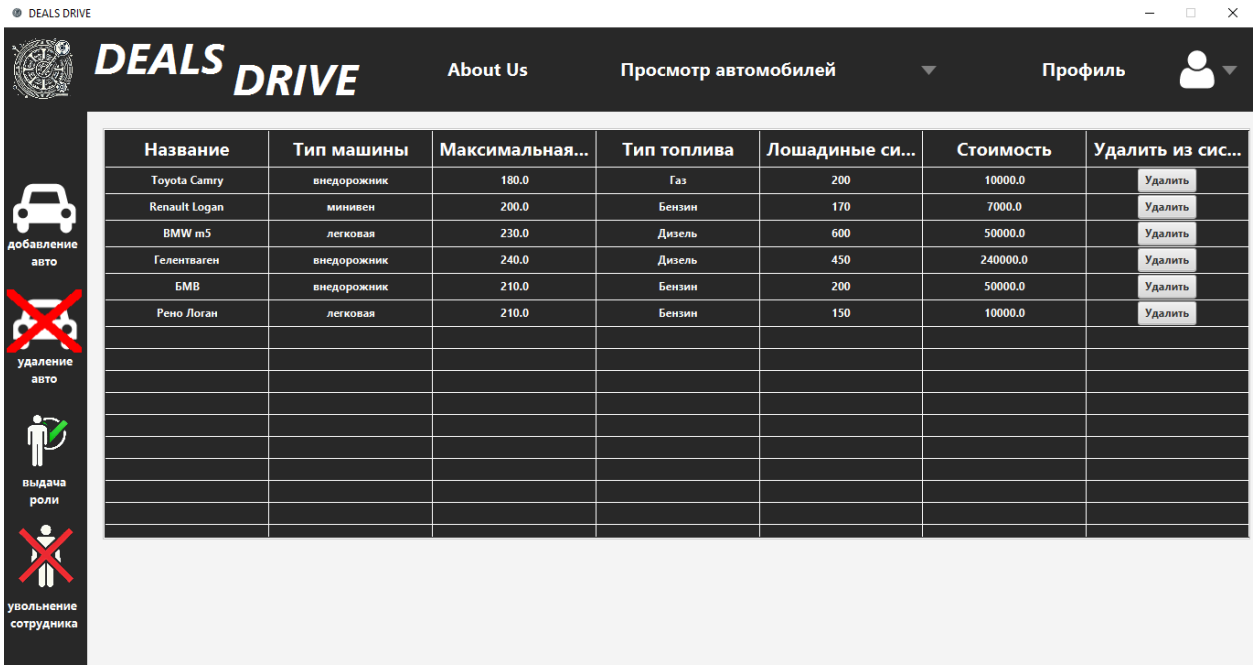

Рисунок 4.6 – Окно удаления авто из системы

Функции выдачи роли и удаления пользователей из системы выглядят примерно одинаково. Администратор получает полный список всех пользователей в виде таблицы, далее выбором чекбоксов определяет каким пользователям выдать роль работника или удалить. После нажатия соответствующих кнопок “Выдать роль” и “Уволить сотрудников” произойдут выбранные действия. Скриншоты этих окон представлены на рисунке 4.7 и рисунке 4.8.

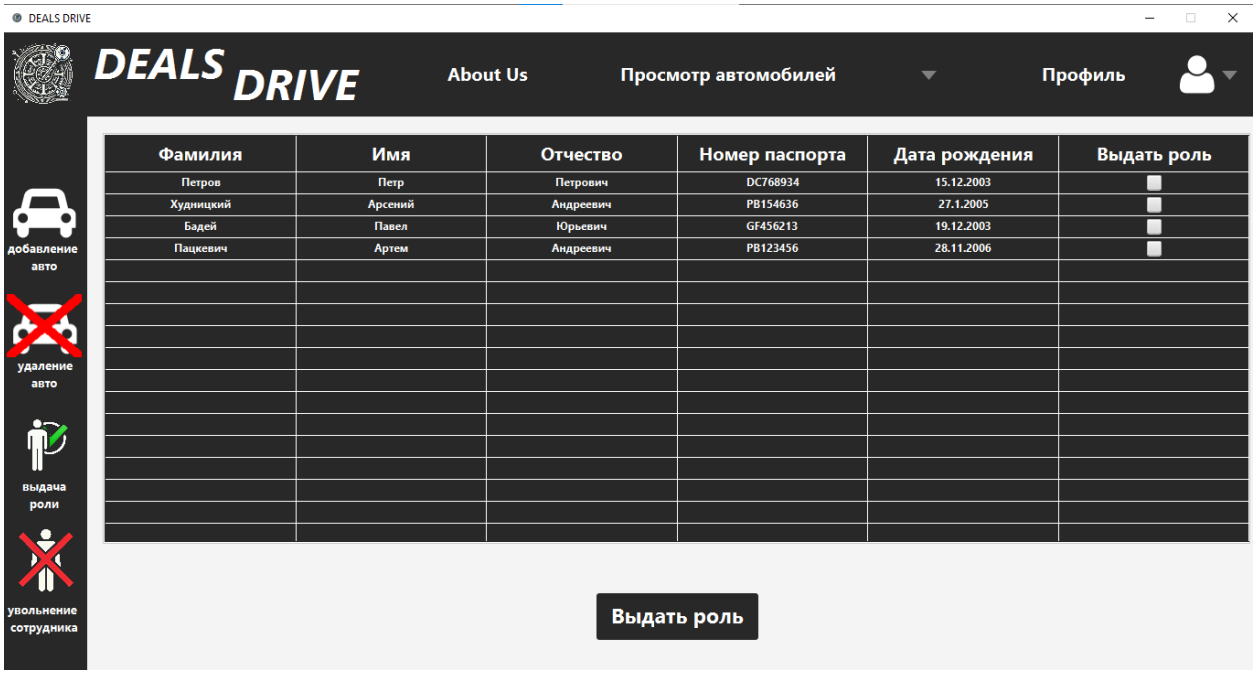

Рисунок 4.7 – Панель выдачи роли

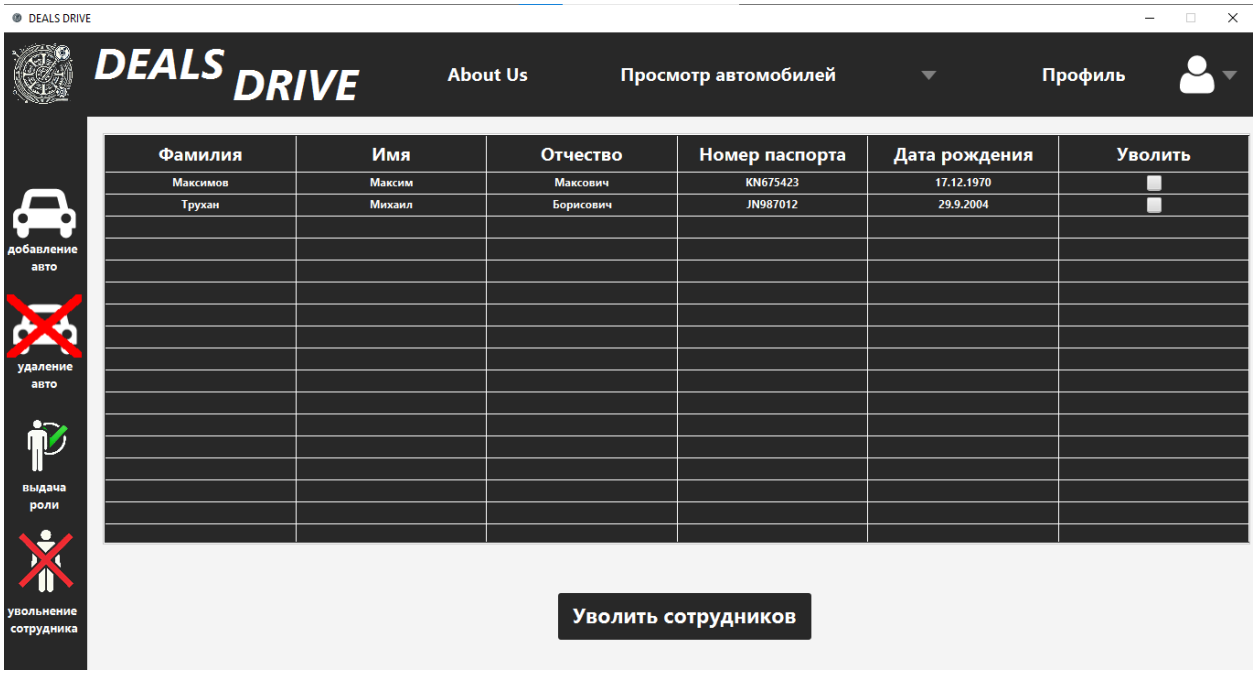

Рисунок 4.8 – Панель увольнения сотрудника

Также существует возможность выхода из аккаунта. Эта кнопка находится в выпадающем списке профиля. После выхода из аккаунта пользователь возвращается в окно авторизации.

Сейчас рассмотрим панель работника. Работник также имеет широкий спектр возможностей для взаимодействия с системой. Работник может составлять отчет по всем одобренным заявкам на автомобили, просматривать все заявки в системе, одобрять или отклонять заявки, работать с тест-

драйвами, а также блокировать клиентов. Скриншот окна просмотра заявок представлен на рисунке 4.9.

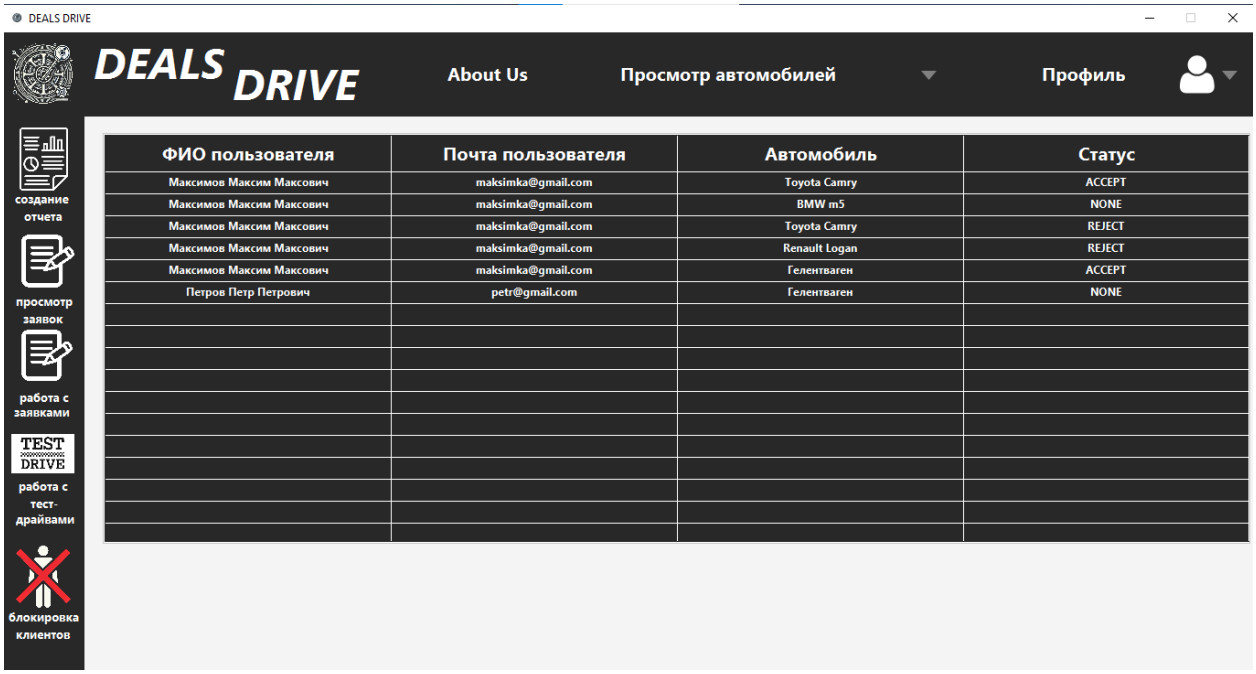

Рисунок 4.8 – Панель просмотра заявок

Здесь работник может лишь ознакомиться со всеми заявками в системе и их статусами. На рисунке 4.9 уже представлена панель для выдачи статуса заявкам, заявки могут быть статуса: NONE, WAIT, REJECT, ACCEPT. В данной панели ему выдаются только те заявки, у которых статус WAIT или NONE.

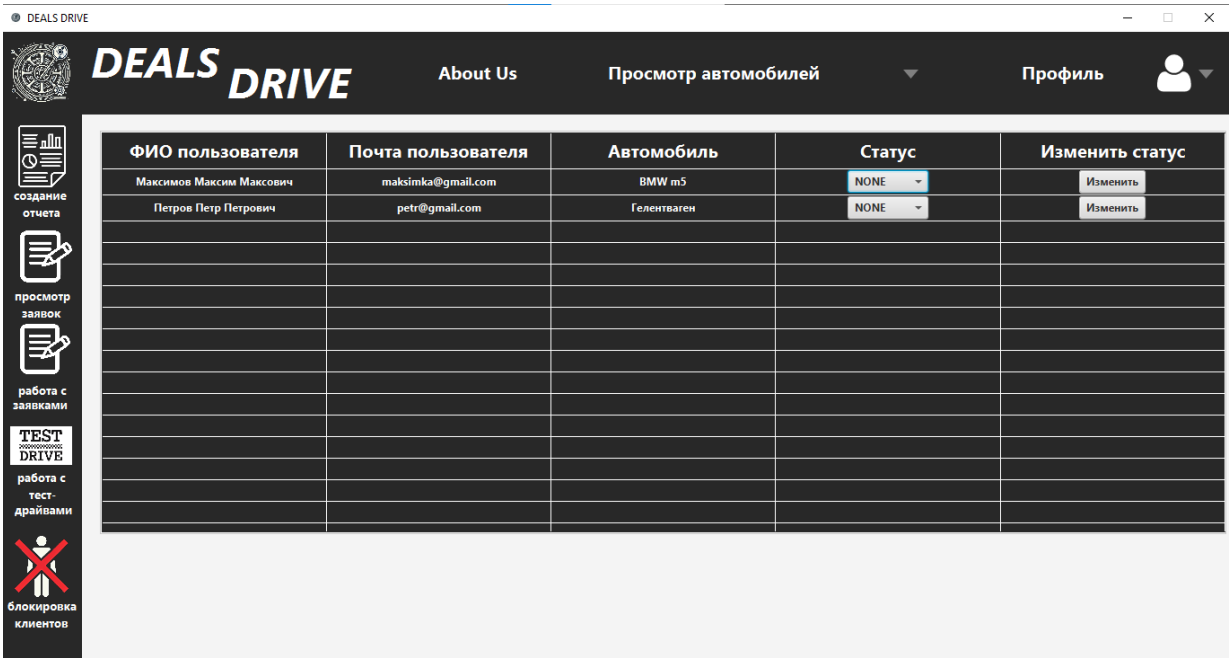

Рисунок 4.9 – Панель работы с заявками

После выдачи статуса таблица динамически обновляется и если статус изменен на ACCEPT или REJECT, то заявка пропадает из таблицы.

Следующая панель для работы с тест-драйвами. В ней выводятся все тест-драйвы в системе, однако статус может быть изменен только у тех, которые имеют статус NONE. Панель представлена на рисунке 4.10.

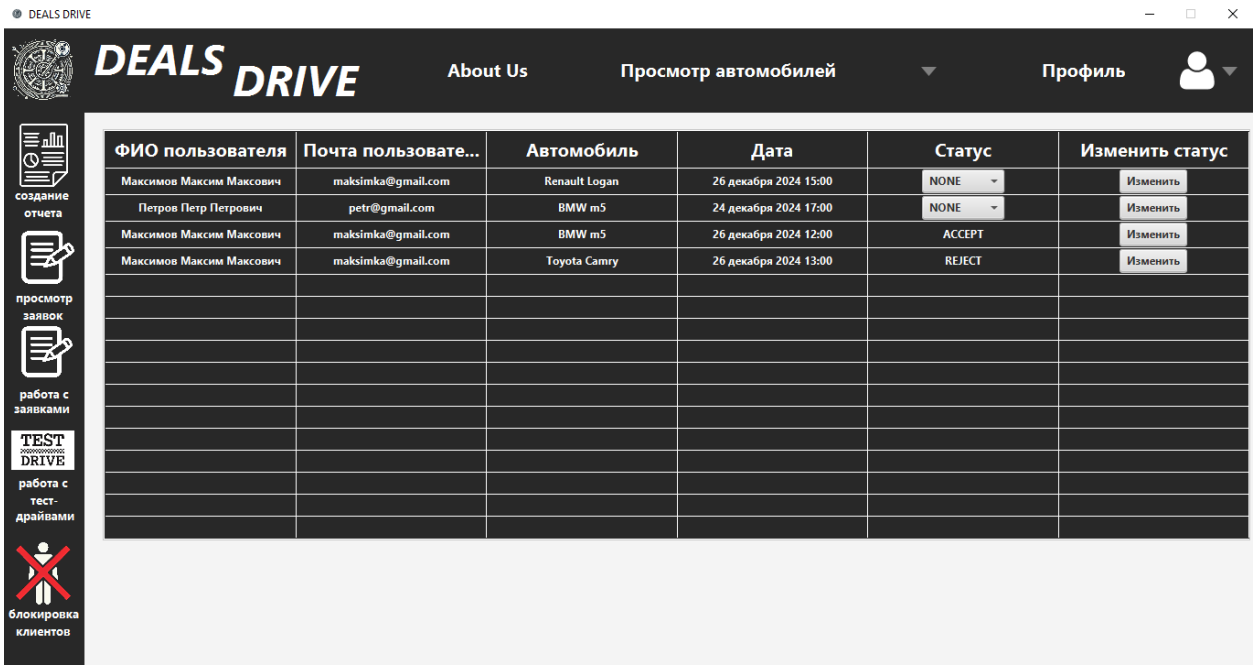

Рисунок 4.10 – Окно работы с тест-драйвами

Следующая панель для удаления пользователей из системы. Выбрав пользователей для удаления с помощью чекбоксов и нажав кнопку удалить пользователей, произойдет удаление всех пользователей и соответствующих им элементов системы. Скриншот панели представлен на рисунке 4.11.

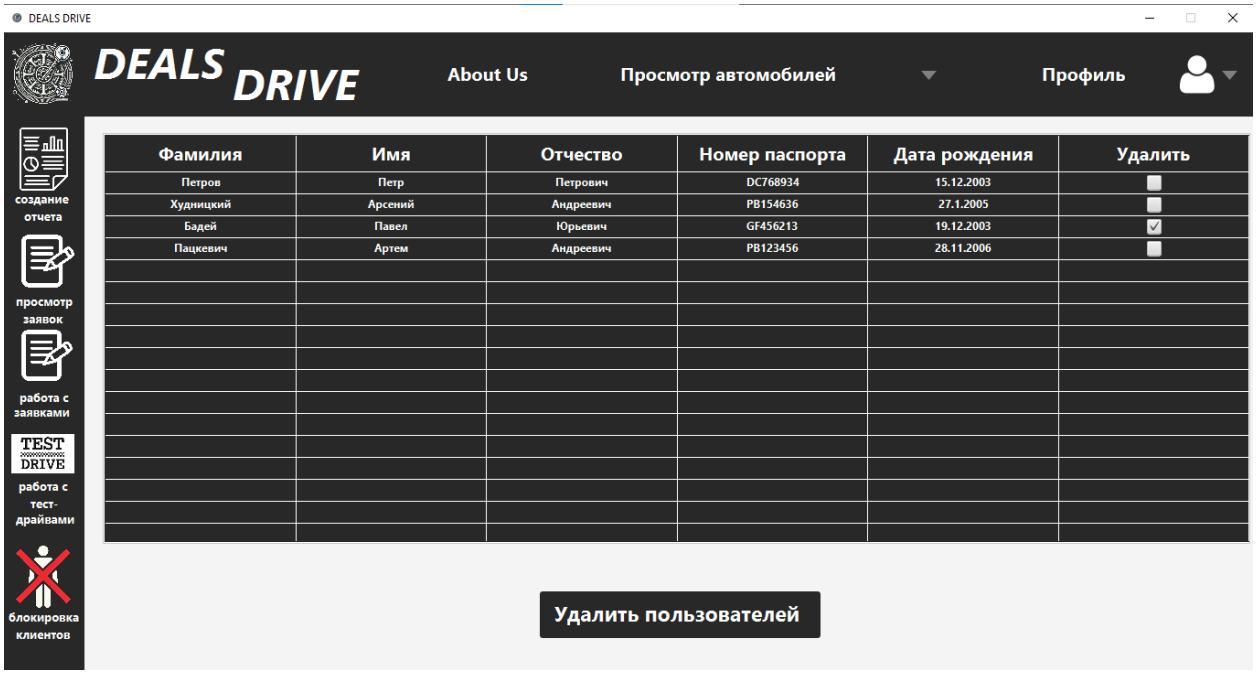

Рисунок 4.11 – Панель блокировки клиентов

При создании работником отчета он появляется в текстовом формате в таком виде (рисунок 4.12).

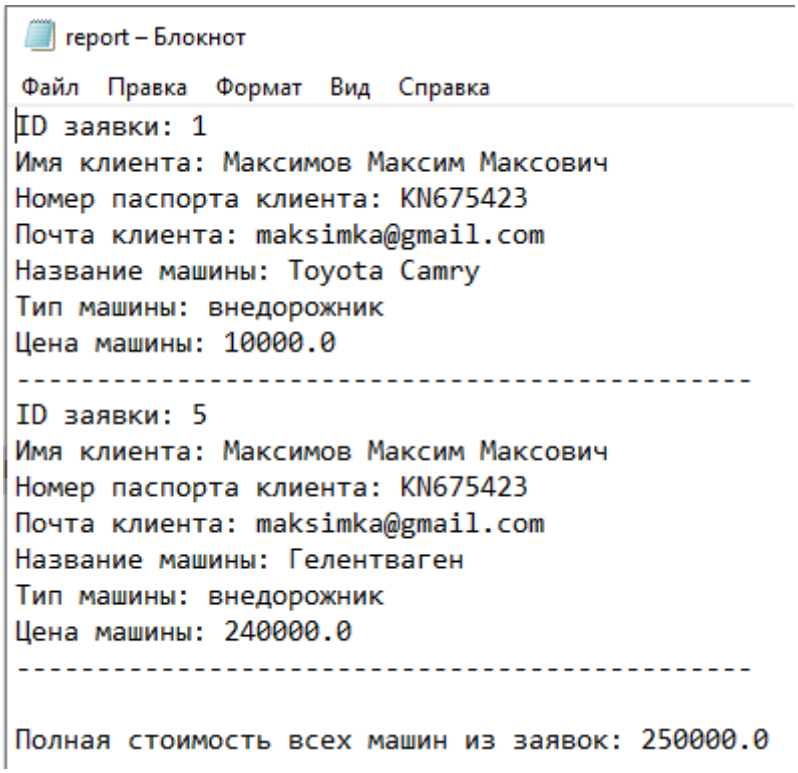

Рисунок 4.12 – Формат отчета

Также работник может, как и администратор, может просматривать информацию о компании (рисунок 4.13), просматривать все автомобили и

сортировать их (рисунок 4.14), а также просматривать информацию о своем профиле (рисунок 4.15).

Рисунок 4.13 – Панель данных о компании

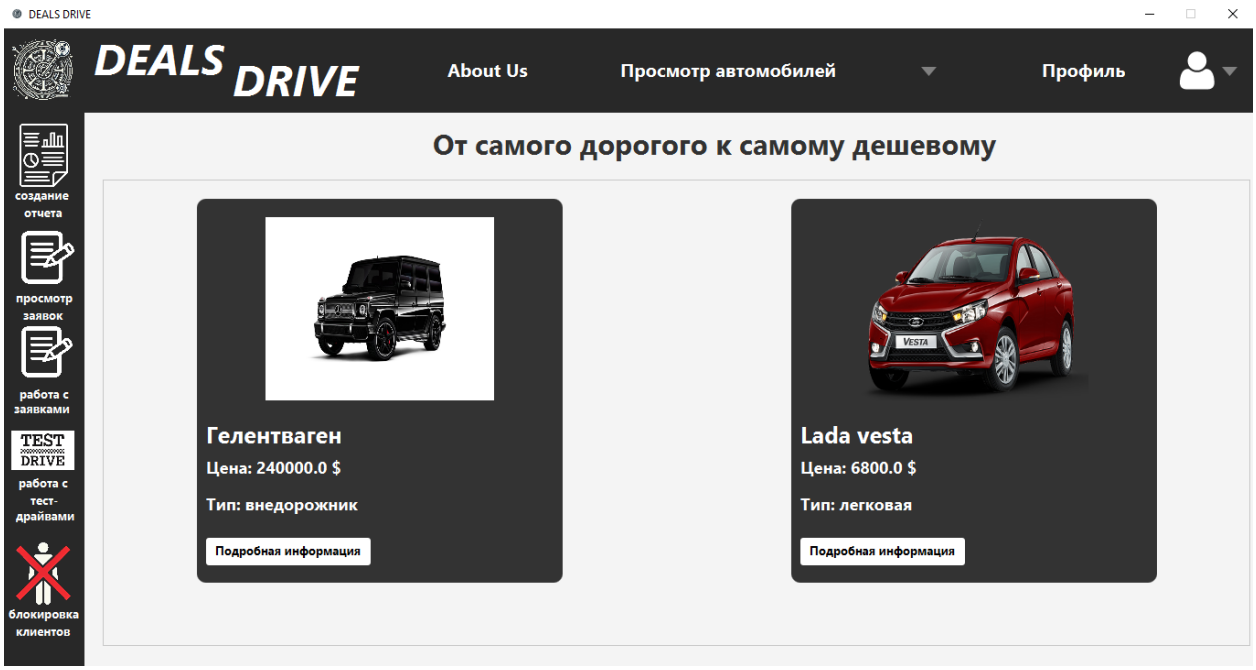

Рисунок 4.14 – Панель просмотра автомобилей с сортировками

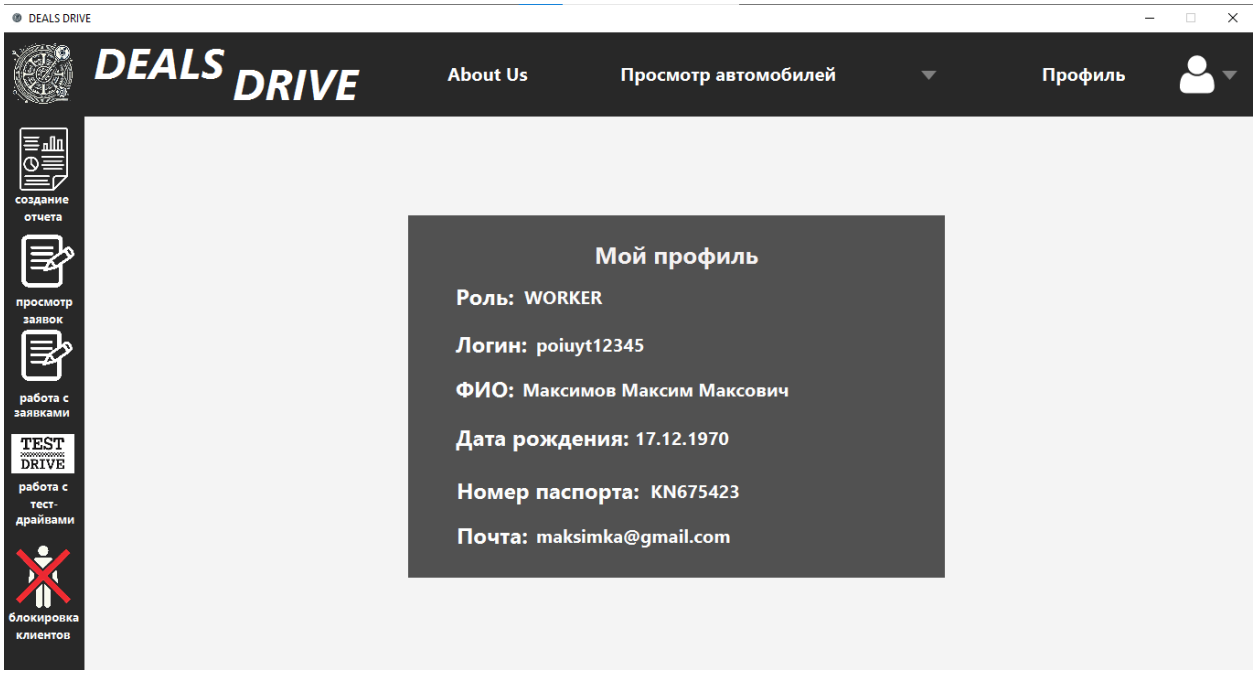

Рисунок 4.15 – Панель с данными профиля работника

Последней ролью в нашей системе остается роль клиента. У него также широкий спектр возможностей в нашей программе. Клиент может: добавлять автомобили в избранное, удалять из избранного, записываться на тест-драйв, оставлять заявку на автомобиль, просматривать свои тест-драйвы и заявки. Скриншот окна представлен на рисунке 4.16.

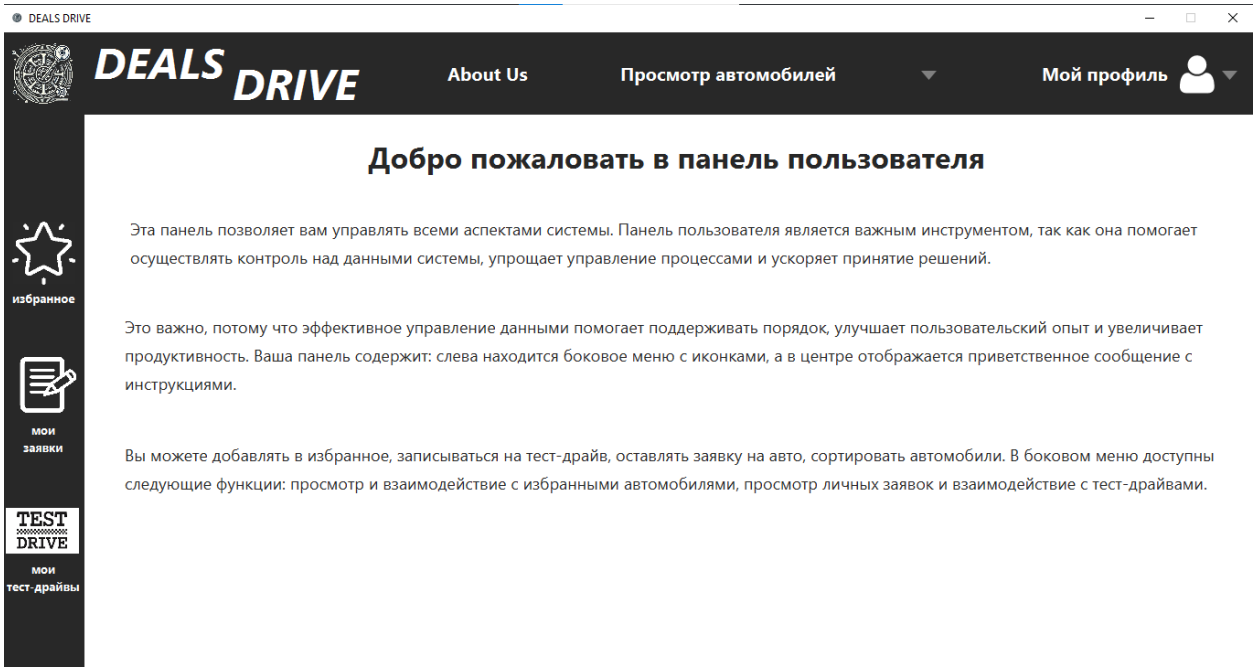

Рисунок 4.16 – Панель пользователя

В этой панели пользователя как раз и взаимодействует с системой. В панели избранное ему выдаются все карточки, которые он добавлял в избранное в этой же панели он может удалить автомобиль из раздела (рисунок 4.17).

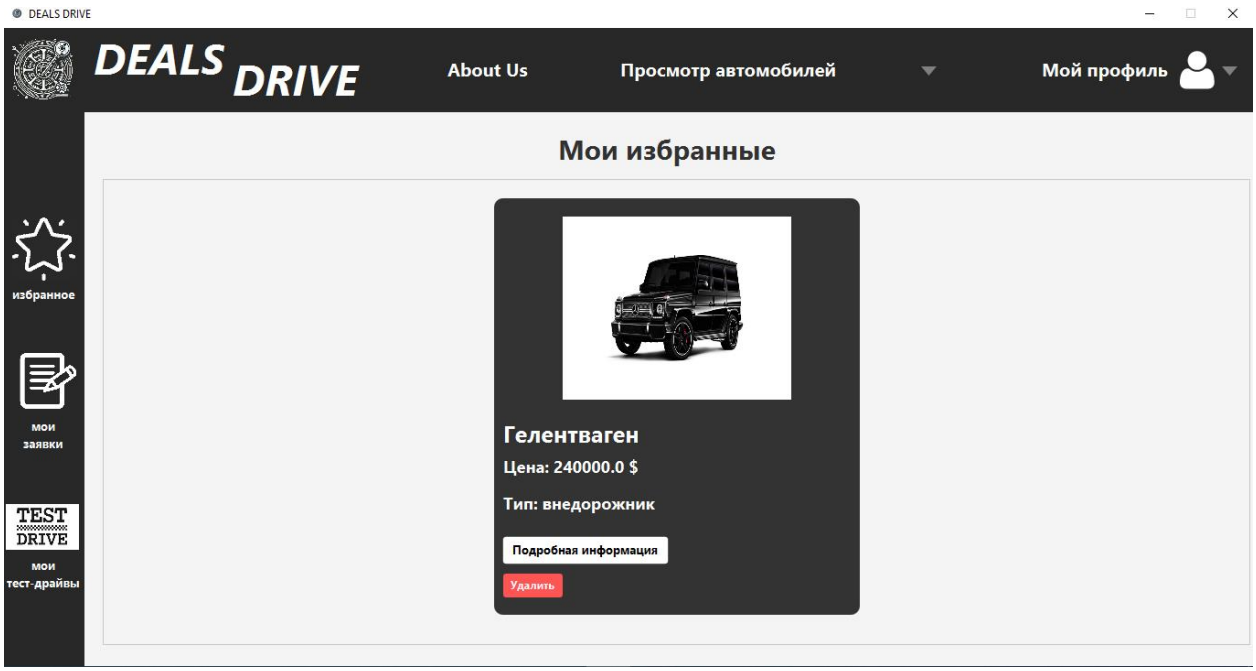

Рисунок 4.17 – Панель избранного

Также имеется кнопка удалить, нажав которую появится требование подтверждения удаления авто из избранного, и в случае подтверждения будет выдан алерт об успешности удаления (рисунок 4.18).

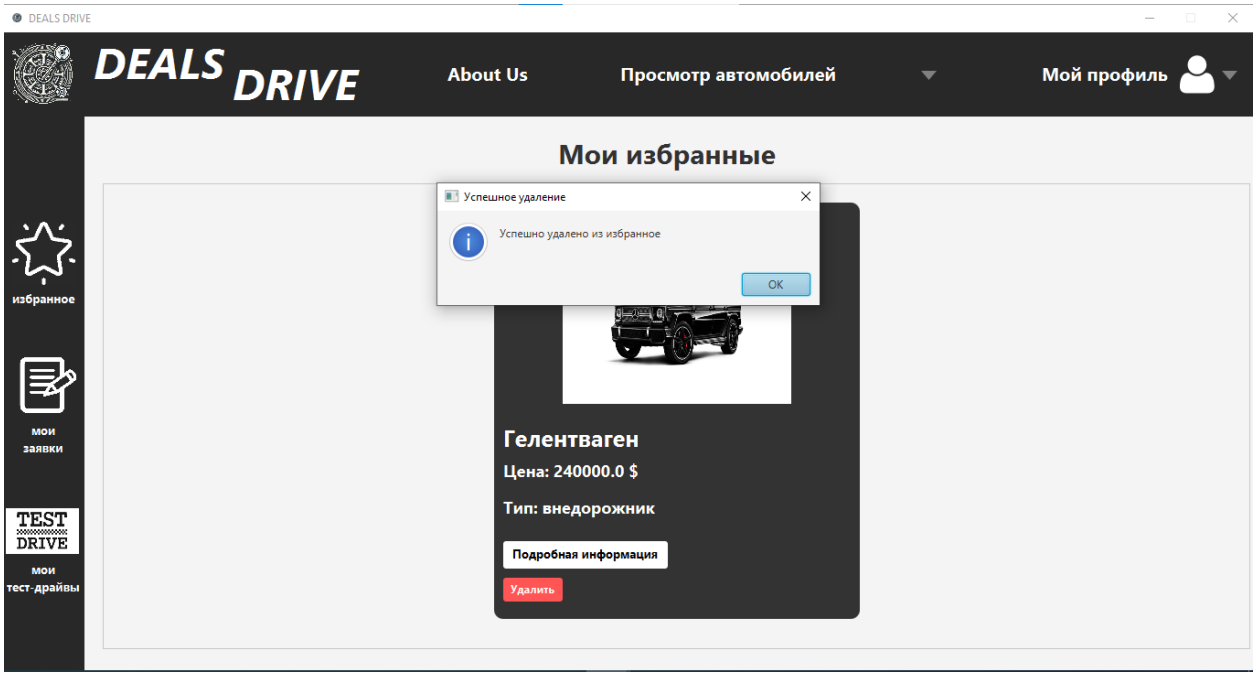

Рисунок 4.18 – Удаление элемента из избранного

Панель просмотра заявок представлена на рисунке 4.19. Пользователь не может с ними взаимодействовать, лишь отслеживать статус заявки.

Рисунок 4.19 – Панель заявок пользователя

Панель тест-драйвов пользователя представлена на рисунке 4.20. Он может просматривать свои тест-драйвы, отслеживать их статус и также удалять.

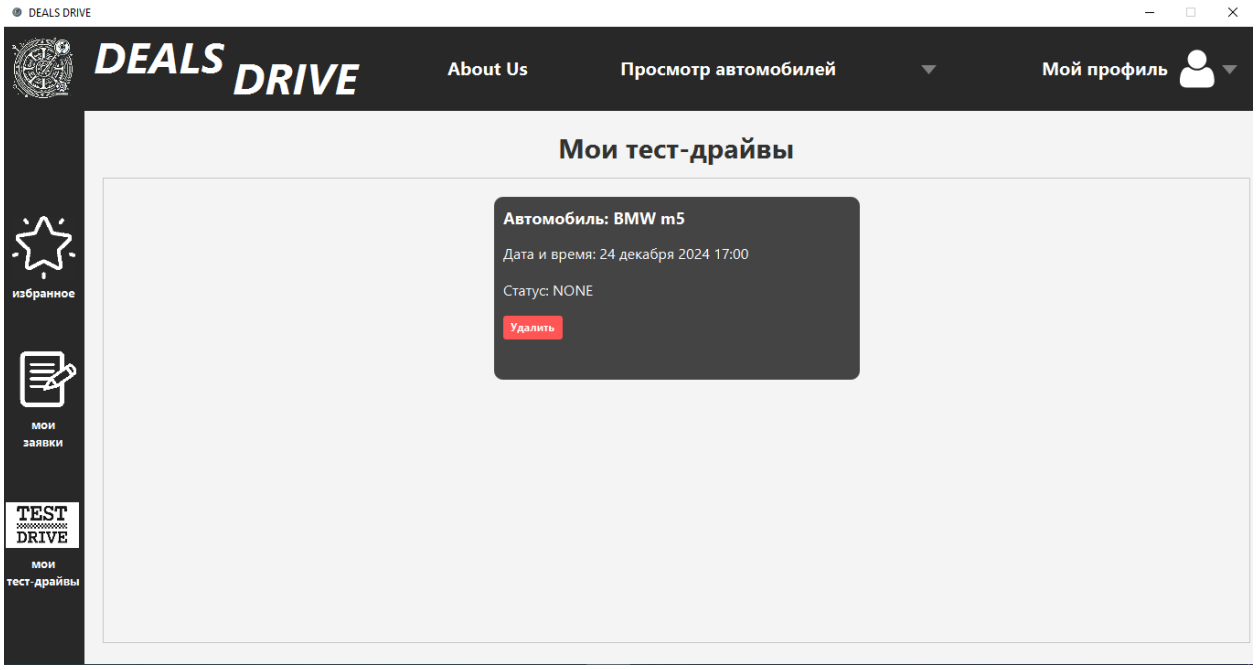

Рисунок 4.20 – Панель тест-драйвов пользователя

Также у пользователя отличаются карточки автомобилей. Он не только просматривает подробную информацию, а сразу может оставить заявку, добавить в избранное и записаться на тест-драйв (рисунок 4.21).

Рисунок 4.23 – Карточка автомобиля при просмотре пользователем

При записи на тест-драйв вылетает модальное окно с выбором даты и времени (рисунок 4.24). Минимальная дата, которая доступна для записи это через неделю от текущей даты, после введения корректных данных пользователь получит уведомление об успешной записи на тест-драйв (рисунок 4.25).

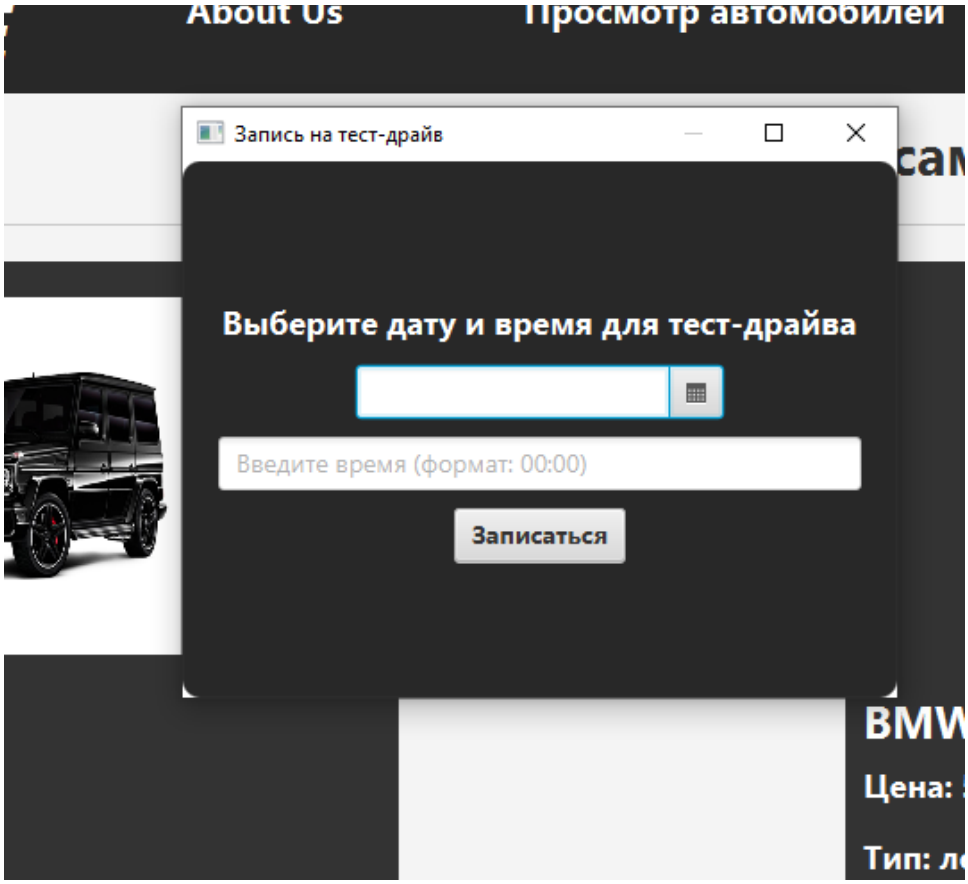

Рисунок 4.24 – Модальное окно для выбора даты и времени

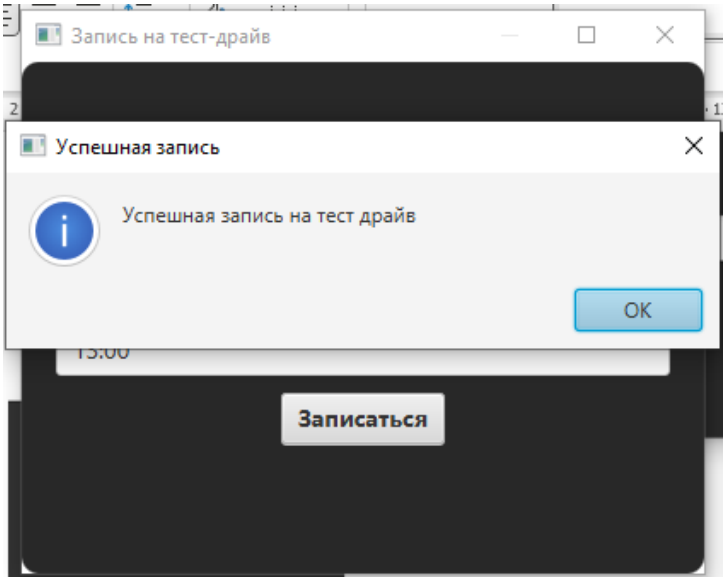

Рисунок 4.25 – Оповещение об успешной записи

Просмотр данных профиля (рисунок 4.26) и возможности доступные при нажатии на профиль (рисунок 4.27) совпадают с пользователями остальных ролей.

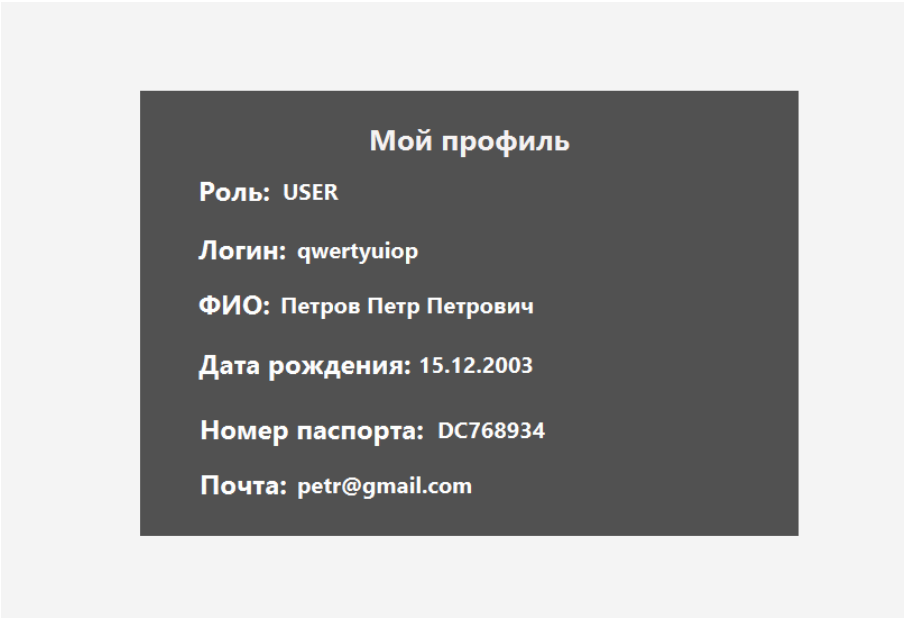

Рисунок 4.26 – Профиль пользователя

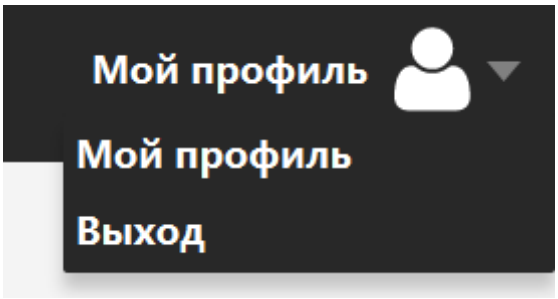

Рисунок 4.27 – Возможности при взаимодействии с профилем

В заключение можно сказать, что приложение имеет широкий функционал для каждой из ролей. Каждая роль использует интуитивно понятный интерфейс для взаимодействия с системой, каждая функция работает корректно и автоматизирует свои процессы.

ЗАКЛЮЧЕНИЕ

В результате выполнения курсового проекта была достигнута цель повышения эффективности работы автосалона за счет внедрения автоматизированной системы управления заявками и обработки данных. Разработанное программное обеспечение предоставило сотрудникам автосалона удобный инструмент для оптимизации взаимодействия с клиентами, ускорения обработки заявок и повышения точности учета информации. Клиенты получили возможность оперативного доступа к актуальной информации о наличии автомобилей и записи на тест-драйвы через интуитивно понятный интерфейс.

Программное обеспечение не только упростило текущие процессы, но и создало основу для дальнейшего развития автосалона в условиях конкурентной среды. Особое внимание было уделено разработке интерфейса, который ориентирован на удобство использования и интуитивную понятность для всех категорий пользователей. Автоматизация позволила улучшить координацию между отделами, минимизировать ошибки, связанные с ручным вводом данных, и обеспечить высокую точность обработки запросов.

Проектирование системы было выполнено с учетом возможности её масштабирования и усовершенствования. В будущем можно расширить функционал программы, добавив дополнительные модули, такие как интеграция с внешними сервисами, предоставляющими кредитование или страхование автомобилей, а также создание аналитических инструментов для оценки клиентских предпочтений.

Таким образом, реализованная система автоматизации не только повышает качество обслуживания клиентов, но и способствует увеличению общей производительности работы автосалона. Она создает основу для долгосрочного успеха и развития бизнеса, учитывая растущие требования современного рынка и предпочтения пользователей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] [Электронный ресурс]. – Как работает автосалон. – Режим доступа: https://drive-63ru.livejournal.com/8448.html

[2] [Электронный ресурс]. – Назанчение и принцип действия DAO. – Режим доступа: http://javatutor.net/articles/j2ee-pattern-data-access-object

[3] [Электронный ресурс]. – Паттерн проектирования “Стратегия”. – Режим доступа: https://javarush.com/groups/posts/2271-pattern-proektirovanija-strategija

[4] [Электронный ресурс]. – Паттерн проектирования Factory. – Режим доступа: https://javarush.com/groups/posts/2370-pattern-proektirovanija-factory

[5] И.Н. Блинов, В.С. Романчик, «Java. Методы программирования» Минск: издательство «Четыре четверти», 2013. — 896 с.

[6] Брюс Эккель, «Философия Java», 2009, 4-е издание. – 638 с.

[7] Герберт Шилдт, «Java 8. Руководство для начинающих», 6-е издание, 2015. – 712 с.

[8] Бен Форта-Освой самостоятельно SQL. 10 минут на урок, 3-е издание.

[9] Проблемы планирования и управления. Опыт системных исследований. / Под ред. Голубкова Е.П. – М., Экономика, 1987. – 311с.

[10] UML. Классика CS. 2-у изд./Пер. с англ.; Под общей редакцией проф. С. Орлова - СПб.: Питер, 2006. - 736 с.: ил.

ПРИЛОЖЕНИЕ А
(обязательное)

Отчёт о проверке на заимствование в системе «Антиплагиат»

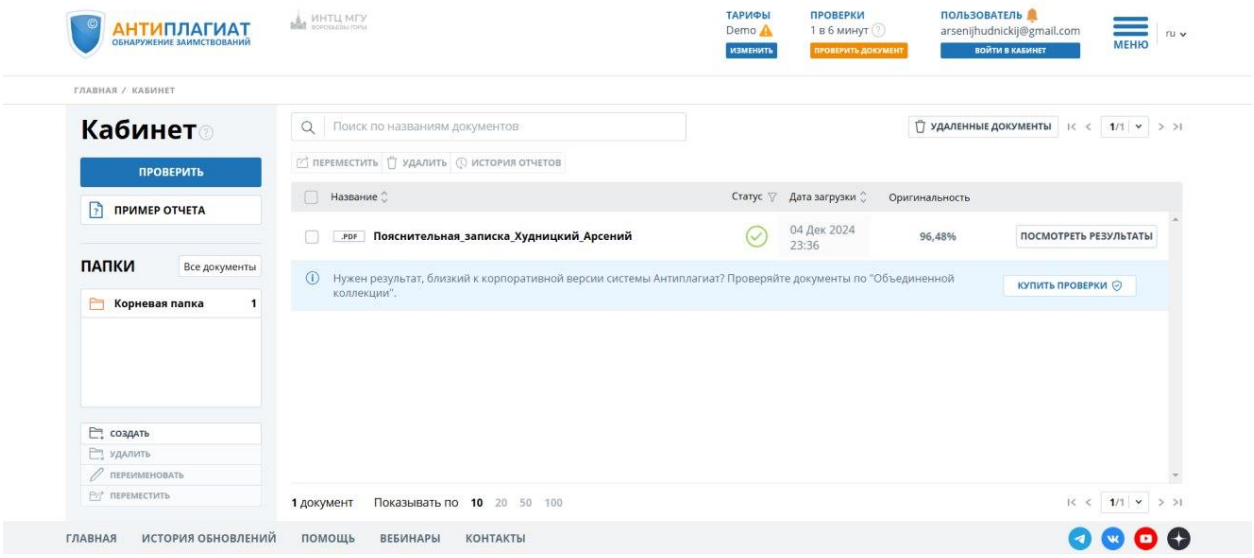

Рисунок А.1 – Результат проверки в системе «Антиплагиат»

ПРИЛОЖЕНИЕ Б
(обязательное)

Листинг SQL-скрипта

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
START TRANSACTION;
SET time_zone = "+00:00";

CREATE DATABASE IF NOT EXISTS kursavayapsp DEFAULT CHARACTER SET utf8 COLLATE
utf8_general_ci;
USE kursavayapsp;

-- Структура таблицы users
CREATE TABLE users (
user_id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100),
passport_number VARCHAR(100),
person_id INT,
birth_date INT,
birth_month INT,
birth_year INT,
gmail VARCHAR(100),
FOREIGN KEY (person_id) REFERENCES person(person_id)
) ENGINE=InnoDB; DEFAULT CHARSET=utf8;

-- Структура таблицы person
CREATE TABLE person (
person_id INT AUTO_INCREMENT PRIMARY KEY,
login VARCHAR(50),
password VARCHAR(256),
role_id INT,
FOREIGN KEY (role_id) REFERENCES role(role_id)
)
ENGINE=InnoDB; DEFAULT CHARSET=utf8;

-- Структура таблицы orders
CREATE TABLE test_drive (
drive_id INT AUTO_INCREMENT PRIMARY KEY,
car_id INT, -- Идентификатор автомобиля
user_id INT, -- Внешний ключ к таблице users
drive_date DATETIME, -- Дата тест-драйва
status ENUM('ACCEPT', 'REJECT', 'NONE') DEFAULT 'NONE',
FOREIGN KEY (user_id) REFERENCES users(user_id),
FOREIGN KEY (car_id) REFERENCES cars(car_id)
) ENGINE=InnoDB;DEFAULT CHARSET=utf8;
```

ПРИЛОЖЕНИЕ В
(обязательное)

Схема алгоритмов работы программы

Рисунок В.1 – Схема алгоритма работы сервера

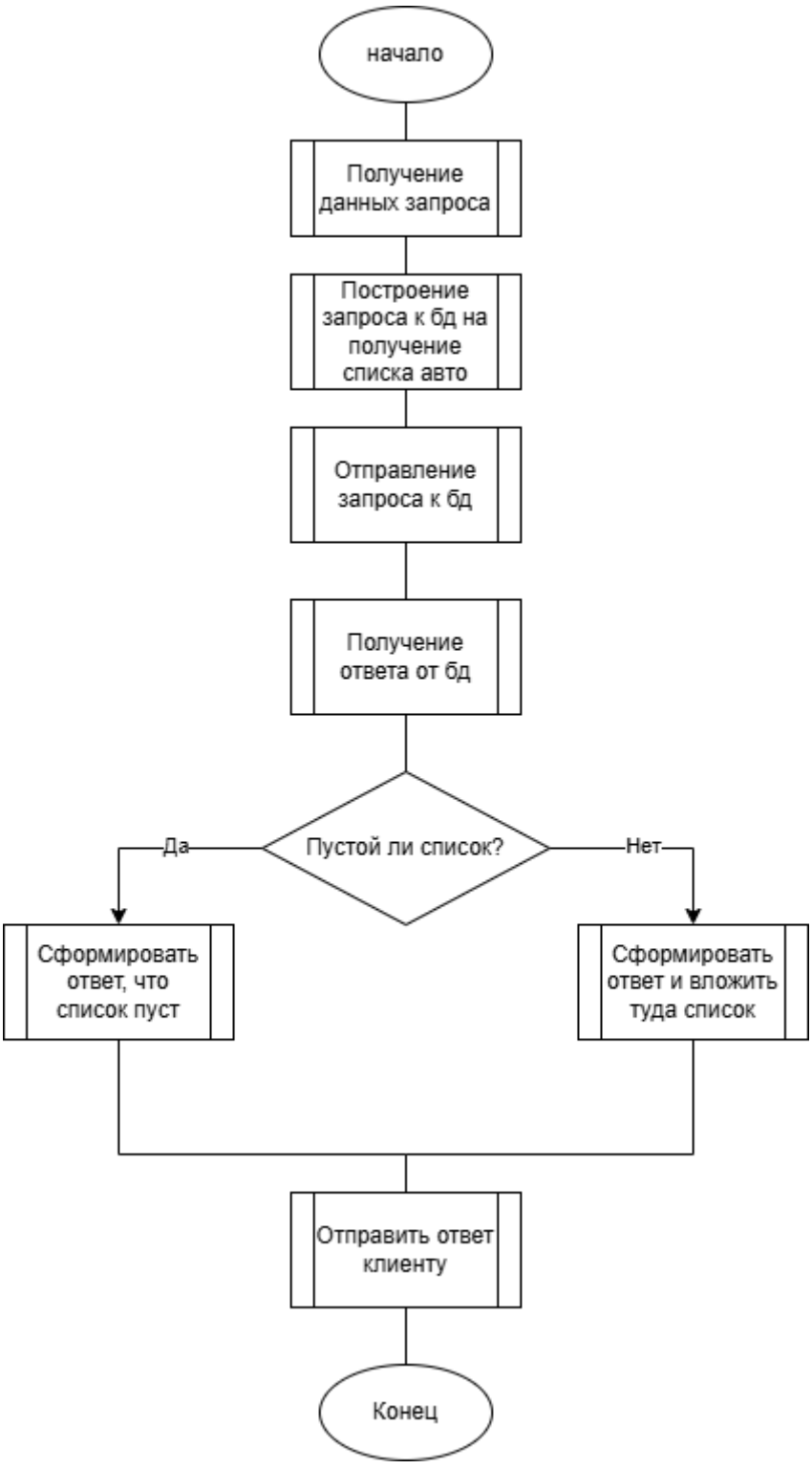

Рисунок В.2 – Схема алгоритма обработки запроса на получение списка автомобилей

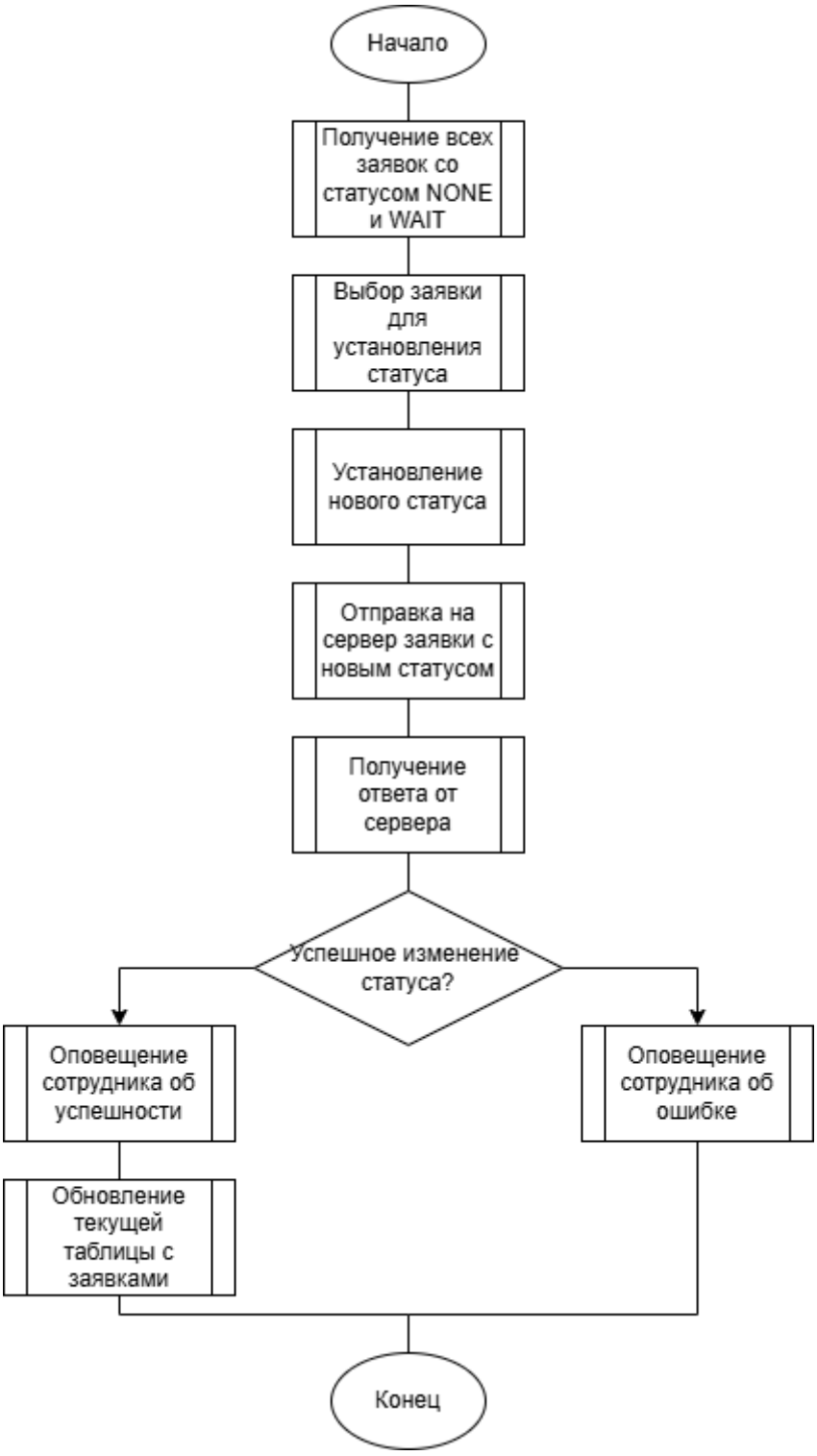

Рисунок В.3 – Схема алгоритма выдачи статуса заявке работником

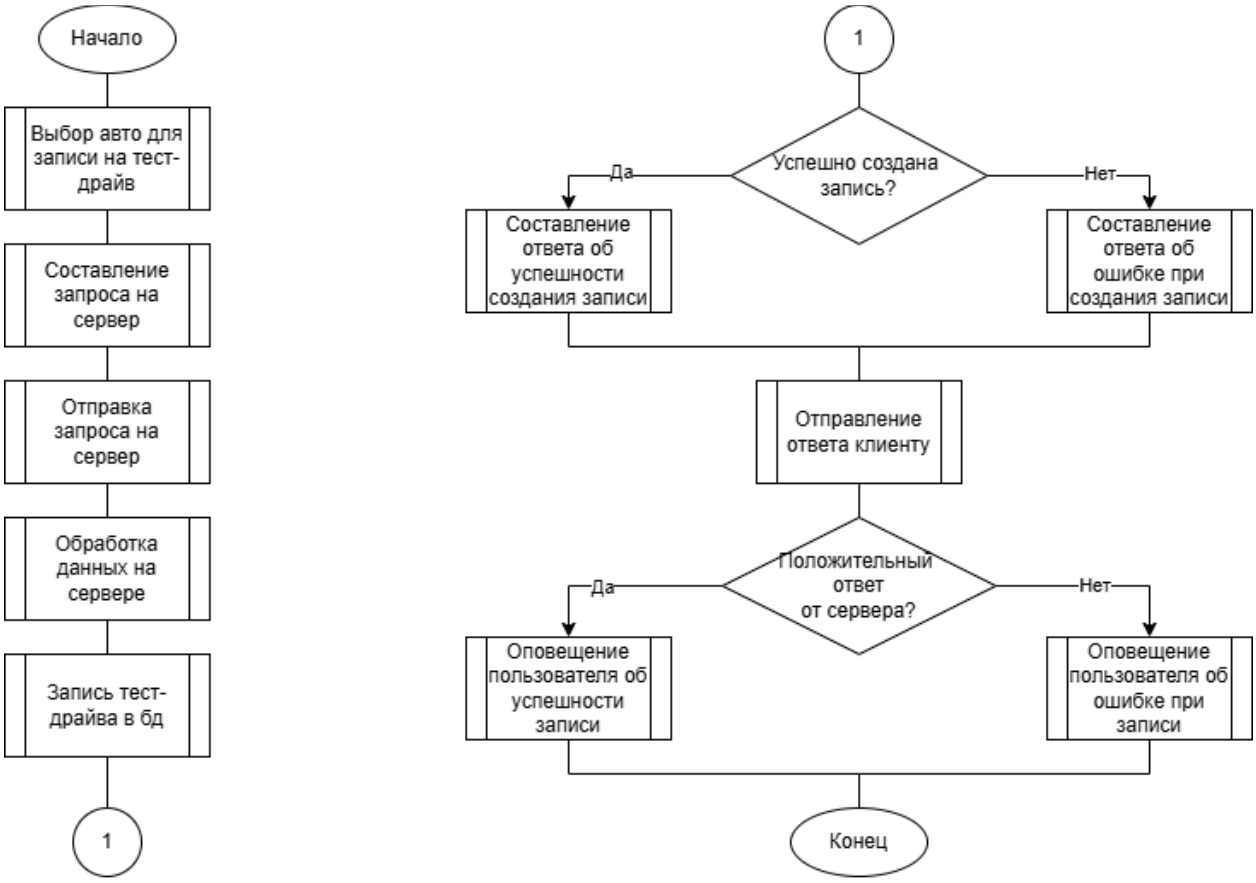

Рисунок В.4 – Схема алгоритма взаимодействия клиента и сервера при записи на тест-драйв

ПРИЛОЖЕНИЕ Г
(обязательное)

Листинг кода

Файл Main.java:

```
package main;

import main.Models.Entities.User;
import main.Services.UserService;
import main.Utility.ClientThread;
import main.Utility.HibernateSessionFactory;
import org.hibernate.Session;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class Main {
    public static final int PORT_NUMBER = 3010;
    private static ServerSocket serverSocket;
    private static ClientThread clientHandler;
    private static Thread thread;
    private static List<Socket> currentSockets = new ArrayList<Socket>();

    public static void main(String[] args) throws IOException {
        serverSocket = new ServerSocket(PORT_NUMBER);
        while (true) {

            Iterator<Socket> iterator = currentSockets.iterator();
            while (iterator.hasNext()) {
                Socket socket = iterator.next();
                if (socket.isClosed()) {
                    continue;
                }
                String socketInfo = "Client: " + socket.getInetAddress() +
":" + socket.getPort();
                System.out.println(socketInfo);
            }

            Socket socket = serverSocket.accept();
            currentSockets.add(socket);
            clientHandler = new ClientThread(socket);
            thread = new Thread(clientHandler);
            thread.start();
            System.out.flush();
        }
    }
}
```

Файл LocalDateTineAdapter.java:

```
package main.Models.Adapter;
```

Продолжение приложения Г

```
import com.google.gson.TypeAdapter;

import com.google.gson.stream.JsonReader;
import com.google.gson.stream.JsonWriter;

import java.io.IOException;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class LocalDateTimeAdapter extends TypeAdapter<LocalDateTime> {
    private final DateTimeFormatter formatter =
DateTimeFormatter.ISO_LOCAL_DATE_TIME;

    @Override
    public void write(JsonWriter out, LocalDateTime value) throws IOException
{
        if (value != null) {
            out.value(value.format(formatter));
        } else {
            out.nullValue();
        }
    }

    @Override
    public LocalDateTime read(JsonReader in) throws IOException {
        if (in.peek() == com.google.gson.stream.JsonToken.NULL) {
            in.nextNull();
            return null;
        }
        return LocalDateTime.parse(in.nextString(), formatter);
    }
}
```

Функция getCarRequestsFromServer ():

```
public List<CarRequest> getCarRequestsFromServer()
{
    List<CarRequest> carRequests = new ArrayList<>();

    Request request = new Request();
    request.setRequestMessage("");
    request.setRequestType(RequestType.GET_CAR_REQUESTS);
    try {
        ClientSocket.getInstance().getOut().println(new
Gson().toJson(request));
        ClientSocket.getInstance().getOut().flush();

        String responseJson = ClientSocket.getInstance().getIn().readLine();
        if (responseJson != null) {
            System.out.println("Response from server: " + responseJson);
            Response response = new Gson().fromJson(responseJson,
Response.class);

            if (response.getResponseStatus() == ResponseStatus.OK) {
                System.out.println("Successfully retrieved car requests.");
                Type carRequestListType = new TypeToken<List<CarRequest>>()
{}.getType();
                carRequests = new Gson().fromJson(new
Gson().toJson(response.getData()), carRequestListType);
```

```
            } else {

 System.out.println("Failed to retrieve car requests: " +
response.getMessage());
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    }

    return carRequests;
}
```

Функция handleTestDriveBooking(Car car):

```
private void handleTestDriveBooking(Car car)
{
    Stage modalStage = new Stage();
    modalStage.initModality(Modality.APPLICATION_MODAL);
    modalStage.setTitle("Запись на тест-драйв");

    VBox modalContainer = new VBox(10);
    modalContainer.setPadding(new Insets(20));
    modalContainer.setStyle("-fx-background-color: #282828; -fx-border-
radius: 10; -fx-background-radius: 10;");
    modalContainer.setAlignment(Pos.CENTER);

    Label headerLabel = new Label("Выберите дату и время для тест-драйва");
    headerLabel.setTextFill(Color.WHITE);
    headerLabel.setStyle("-fx-font-size: 18px; -fx-font-weight: bold;");

    DatePicker datePicker = new DatePicker();
    datePicker.setStyle("-fx-font-size: 14px;");
    datePicker.setDayCellFactory(param -> new DateCell() {
        @Override
        public void updateItem(LocalDate item, boolean empty) {
            super.updateItem(item, empty);
            if (item.isBefore(LocalDate.now().plusDays(7))) {
                setDisable(true);
                setStyle("-fx-background-color: #cccccc;");
            }
        }
    });
    TextField timeField = new TextField();
    timeField.setPromptText("Введите время (формат: 00:00)");
    timeField.setStyle("-fx-font-size: 14px;");

    Button bookButton = new Button("Записаться");
    bookButton.setStyle("-fx-font-size: 14px; -fx-font-weight: bold;");
    bookButton.setOnAction(e -> {
        if (datePicker.getValue() == null ||
timeField.getText().trim().isEmpty()) {
            Alert errorAlert = new Alert(Alert.AlertType.ERROR);
            errorAlert.setTitle("Ошибка");
            errorAlert.setHeaderText(null);
            errorAlert.setContentText("Пожалуйста, заполните все поля!");
            errorAlert.showAndWait();
            return;}
```

Продолжение приложения Г

```
String timePattern = "^([01]\\d|2[0-3]):[0-5]\\d$";
        if (!timeField.getText().matches(timePattern)) {
            Alert errorAlert = new Alert(Alert.AlertType.ERROR);
            errorAlert.setTitle("Ошибка");
            errorAlert.setHeaderText(null);
            errorAlert.setContentText("Неправильный формат времени! Введите
время в формате 00:00.");
            errorAlert.showAndWait();
            timeField.clear();
            return;
        }
        LocalDate selectedDate = datePicker.getValue();
        String time = timeField.getText();


        LocalDateTime driveDate = combineDateAndTime(selectedDate,time);
        TestDrive testDrive = new
TestDrive(car,Session.getUser(),driveDate,TestDriveStatus.NONE);

        boolean checker = sendTestDriveToServer(testDrive, SAVE_TEST_DRIVE);
        if (checker) {
            showSuccessAlert("Успешная запись", "Успешная запись на тест
драйв");
            modalStage.close();
        } else {
            showAlert("Ошибка", "Произошла ошибка при записи.");
        }
    });

    modalContainer.getChildren().addAll(headerLabel, datePicker, timeField,
bookButton);

    Scene modalScene = new Scene(modalContainer, 400, 300);
    modalStage.setScene(modalScene);
    modalStage.showAndWait();
}
```