Министерство образования Республики Беларусь
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Инженерно-экономический факультет
Кафедра экономической информатики

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе по курсу:
«Программирование сетевых приложений»
на тему:

«Система учёта продаж автомобилей в автосалонах»

Выполнил: студент группы 073601
Каминская Елена Викторовна

Руководитель: Ассистент кафедры ЭИ
Лыщик А.П.

Минск 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многие в настоящее время пользуются личным автомобилем. У некоторых их даже несколько. Большим спросом пользуются компьютерные программы, способные упростить и автоматизировать работу, ранее требующую больших временных и трудовых затрат.

Таким образом, актуальность данного курсового проекта вызвана необходимостью разработки программы, которая будет наглядно показывать наличие автомобиля, его марку, модель, его стоимость, цвет, год производства.

Целью данного курсового проекта является разработка системы учёта продаж автомобилей в автосалонах.

Для достижения цели требуется реализовать следующие задачи:

– проанализировать логическую и физическую модель представления данных;

– описать принцип работы автосалона;

– разработать схемы алгоритмов работы приложения;

– разработать функциональную тему задачи;

– детально протестировать разработанную программу;

– описать работу программы;

– создать базу данных.

Объектом исследования является процесс учёта продаж автомобилей в автосалонах.

1 АНАЛИЗ И МОДЕЛИРОВАНИЕ ПРЕДМЕТНОЙ ОБЛАСТИ ПРОГРАММНОГО СРЕДСТВА

Автосалон является неотъемлемо важным предприятием для жизни людей. Он пользуется спросом и у людей с личным авто, и без. А при наличии приложения, в котором данные автосалона упорядочены, клиентам становится легко и просто приобрести автомобиль. А администраторам легко упорядочить продажи, а также с помощью приложения может значительно расширить каналы сбыта продукции за счёт потоков новых клиентов через Интернет.

1.1 Описание предметной области

Одним из преимуществ электронного автосалона является его круглосуточная работа. Покупатель сможет всегда вне зависимости от времени суток, выходных и праздников ознакомиться с автомобилями, находящимися в салоне и оформить заявку на понравившийся автомобиль. Таким образом владелец получает дополнительный конкурентный шанс среди подобных салонов, так как может постоянно, информативно и быстро обеспечить дополнительный удобный для клиента сервис.

Для продуктивной работы автосалона желательно иметь минимальный штат для первого старта. На начальном этапе со всем справляется администратор. Он общается с дилерами, организовывает доставку, оформляет машины, обрабатывает заказ. В таком случае система работы автосалона для продажи автомобиля следующая: дилер занимается добавлением автомобилей на сайт, а пользователь оформляет заявку, которая затем обрабатывается продавцом и дилером. Покупатель заходит в электронное приложение, выбирает понравившийся ему автомобиль и оформляет заявку. С точки зрения клиента такая система оформления заявки на автомобиль, несомненно, имеет ряд преимуществ перед традиционными автосалонами:

- отсутствие очередей;
- возможность бронирования выбранного автомобиля;
- круглосуточный прием заявок.

Во многих житейских ситуациях покупка автомобиля – это разумный и экономный выход. Свой ухоженный автомобиль выручает в непредвиденных обстоятельствах, упрощает передвижение по городу, дает возможность продемонстрировать более высокий статус.

Единственным минусом такой системы брони можно назвать лишь отсутствие возможности предварительно посмотреть машину «вживую».

Также можно сказать, что любая компания заранее обречена на провал в случае отсутствия эффективной рекламы и продвижения в интернете.

Каждая машина в автосалоне должна содержать:

– уникальный номер;

– торговую марку автомобиля;

– цену за автомобиль;

– модель автомобиля;

– цвет;

– год выпуска.

1.2 Разработка функциональной модели предметной области.

Для создания функциональной модели работы нашего магазина техники был выбран стандарт IDEF0 – методология функционального моделирования и графическая нотация, предназначенная для формализации описания бизнес–процессов. В данной методологии важна не временная последовательность работ, а отношение между ними.

На рисунке 1.1 контекстная диаграмма верхнего уровня отображает функциональную модель «Система учёта продаж автомобилей в автосалонах».

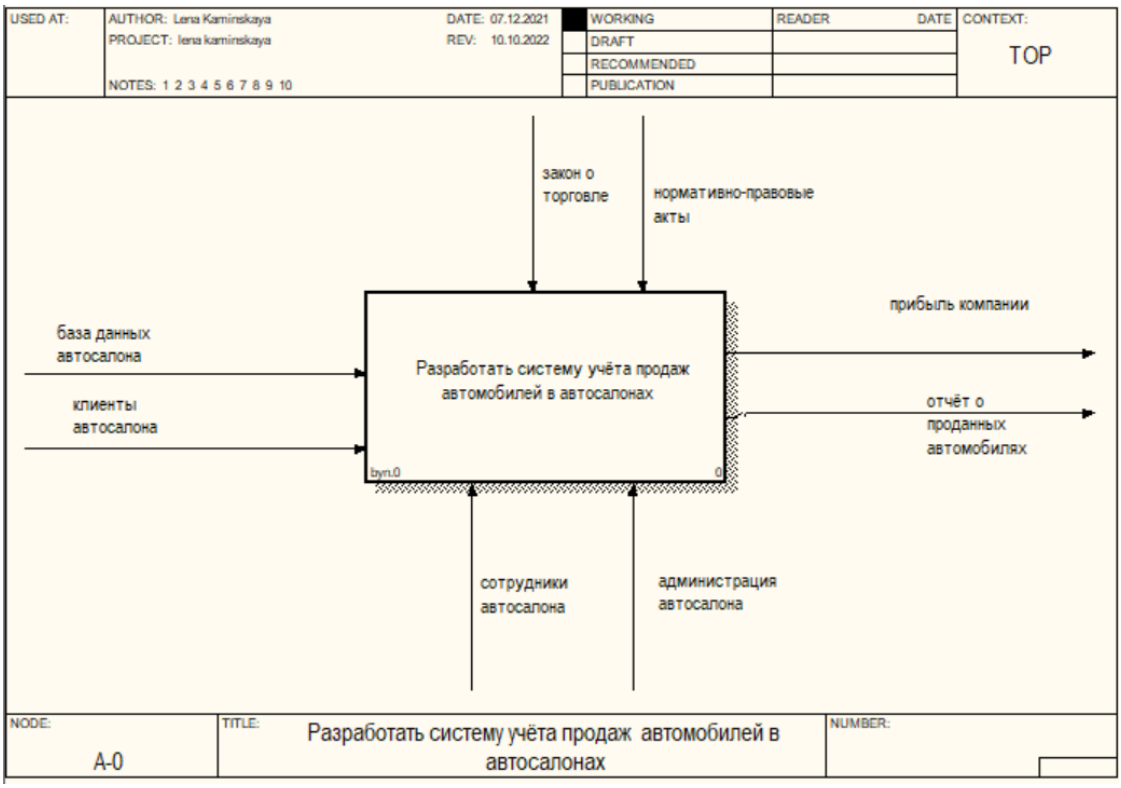

Рисунок 1.1 – Контекстная диаграмма верхнего уровня

Входной поток включает в себя базу данных автосалона, в которой находится информация об автомобилях, и клиентов автосалона. В выводном потоке находятся данные о продаже автомобилей и прибыль компании. Закон о торговле необходим для легального функционирования системы. В качестве механизмов выступают сотрудники автосалона и его администрация.

На рисунке 1.2 – декомпозиция контекстной диаграммы, которая состоит из четырёх блоков: «Закупить автомобили», «Выставить автомобили на продажу», «Продать автомобили», «Составить отчёт».

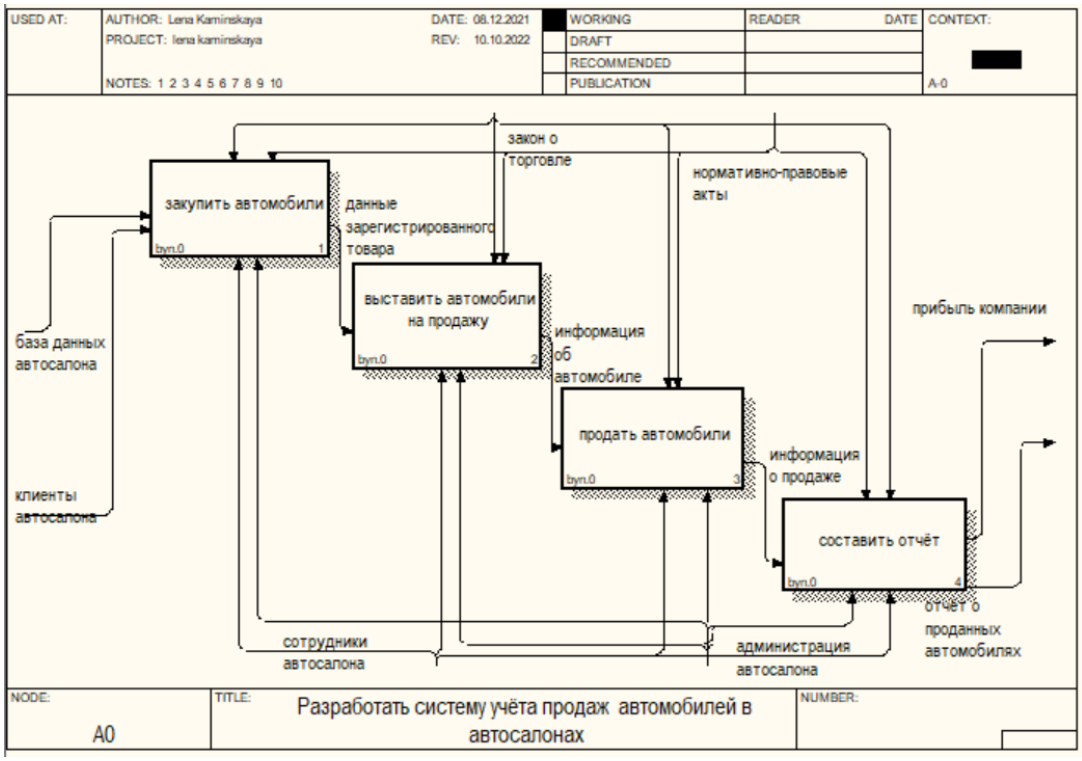

Рисунок 1.2 – Декомпозиция контекстной диаграммы верхнего уровня

В первом компоненте данной декомпозиции («Закупить автомобили») мы получаем информацию из базы данных магазина для закупки автомобилей на продажу. Во втором компоненте содержится информация о выставлении автомобиля на продажу. В третьем блоке происходит процесс продажи автомобиля. На выходе имеем информацию о проданных автомобилях для блока №4 («Составить отчёт»). В итоге получаем отчет о проданных автомобилях.

Декомпозиция блока «Закупить автомобили» показана на рисунке 1.3. Она состоит из четырёх компонентов: «Сделать запрос об автомобиле»,

«Оплатить автомобиль», «Доставить автомобиль в автосалон», «Занести в базу данных».

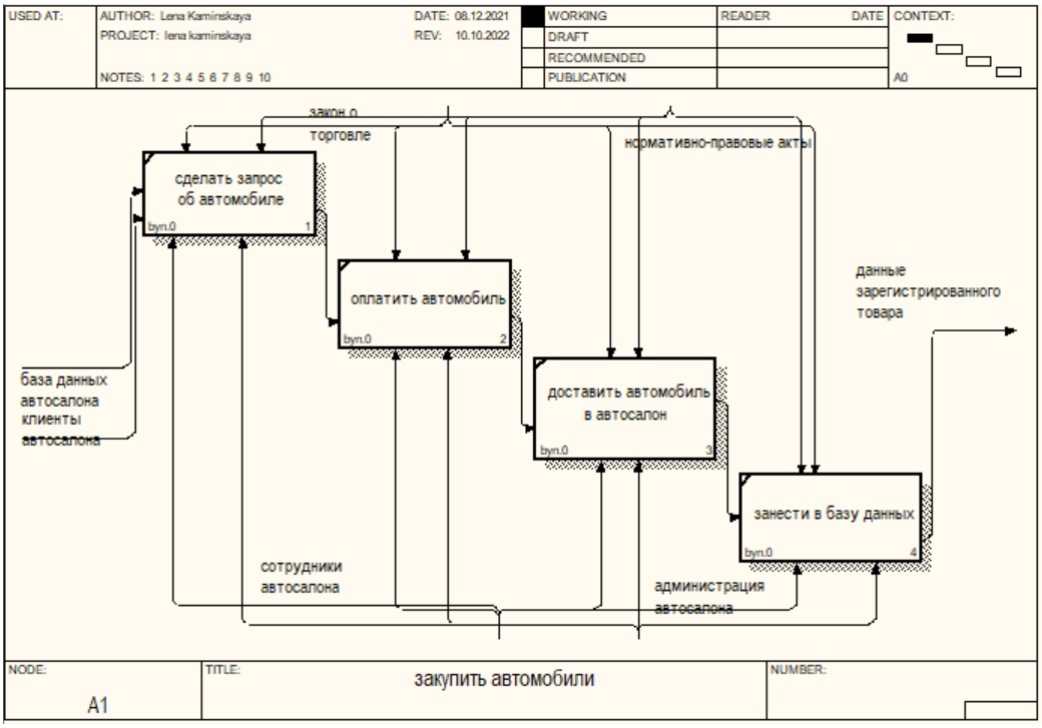

Рисунок 1.3 – Декомпозиция блока «Закупить автомобиль»

Декомпозиция блока «Выставить автомобиль на продажу» показана на рисунке 1.4. Она состоит из трёх компонентов: «Выбрать автомобиль», «Проверить на подлинность», «Выставить на продажу».

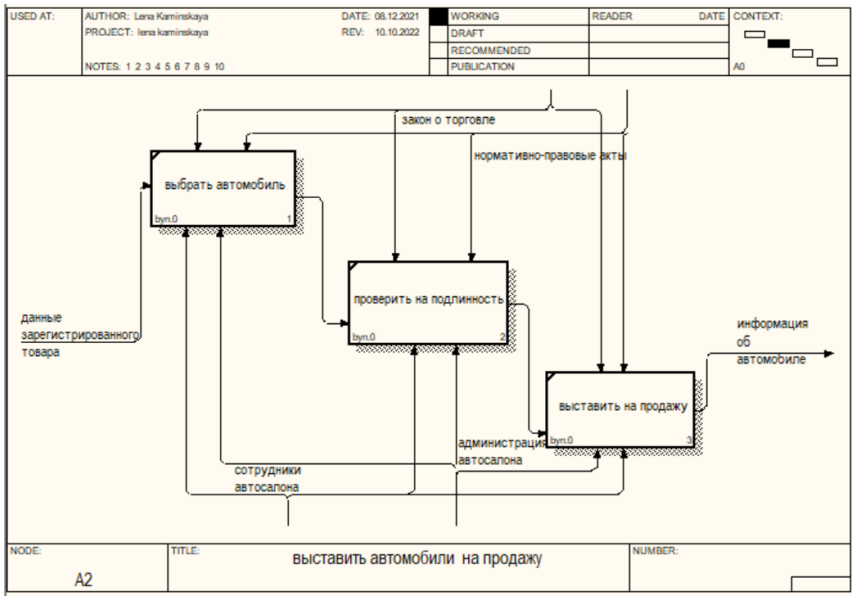

Рисунок 1.4 – Декомпозиция блока «Выставить автомобиль на продажу»

Декомпозиция блока «Продать автомобиль» показана на рисунке 1.5. Она состоит из трёх компонентов: «Оформить заказ», «Произвести транзакцию», «Распечатать чек».

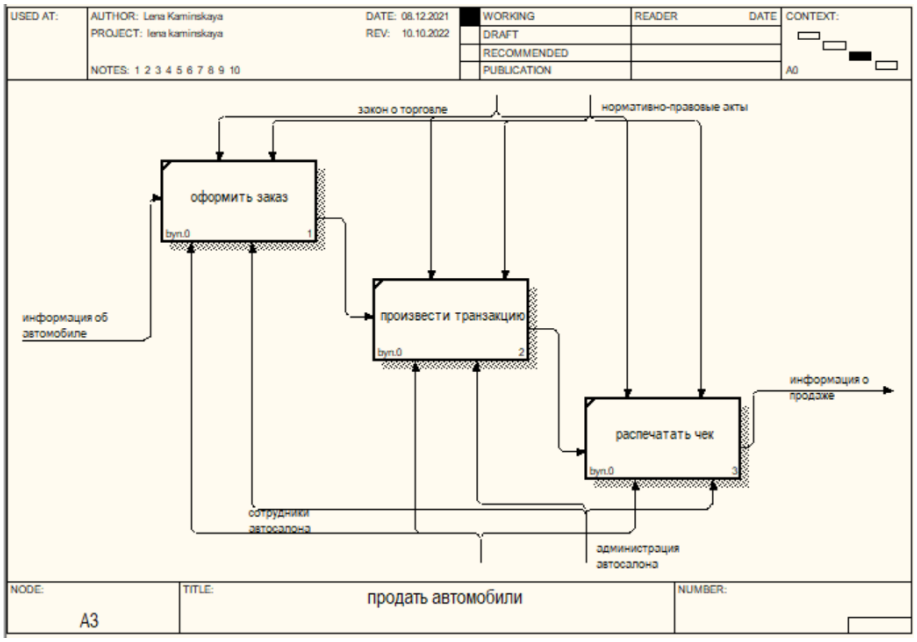

Рисунок 1.5 – Декомпозиция блока «Продать товар»

Декомпозиция блока «Составить отчёт» изображена на рисунке 1.6. Она состоит из трех компонентов: «Собрать информацию о проданном товаре», «Занести информацию в базу данных» и «Составить отчет».

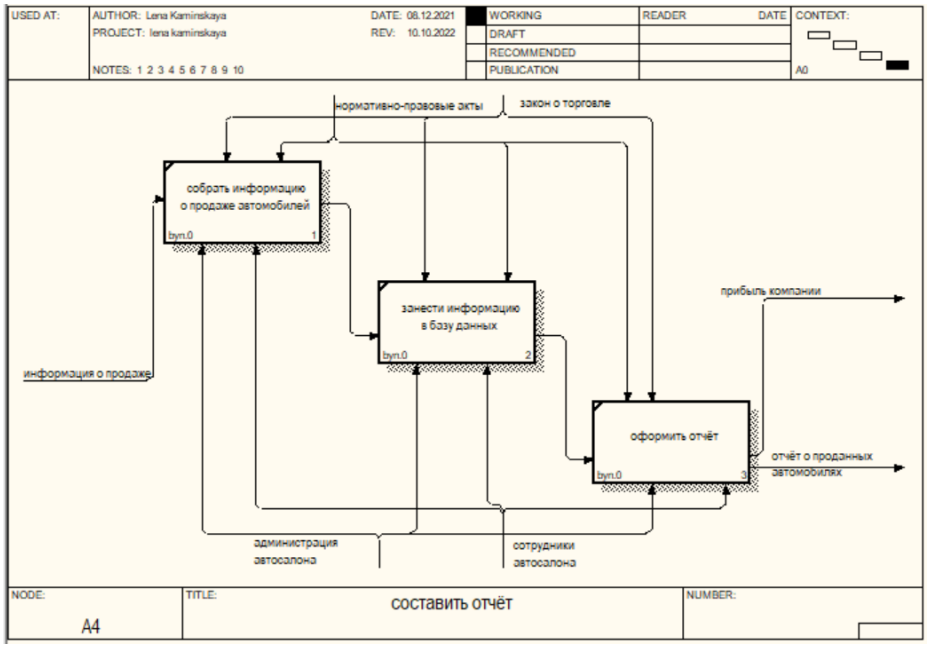

Рисунок 1.6 – Декомпозиция блока «Составить отчёт»

1.3 Анализ требований к разрабатываемому программному средству. Спецификация функциональных требований.

Для разработки программного средства необходимо понять принцип работы автосалона.

Система учёта продаж автомобилей в автосалонах – система, в которой представлены автомобили из автосалона.

Пользователь может зайти в программу и найти автомобиль по подходящим ему критериям и оформить заявку. Каждое объявление о продаже содержит достаточное количество критериев продаваемого автомобиля, чтобы потенциальный покупатель мог оценить автомобиль.

Так как у каждого автосалона имеется список автомобилей, которые в нём находятся, необходимо спроектировать базу данных, которая осуществляла бы их хранение в электронном виде. Это позволит облегчить поиск автомобилей для администратора автосалона, а также для пользователей. Для корректной работы необходимо предусмотреть возможность добавления, удаления и редактирования информации в базе данных. Для хранения информации будет использована СУБД MySQL. Подключение к ней будет осуществляться при авторизации и в зависимости от полученных прав пользователь, будет получать определённый набор возможностей для выполнения своей работы.

Для ведения отчетности, администратор сможет составить отчет о проданных автомобилях в салоне. Это позволит не только скорректировать работу автосалона, но и выявить какие автомобили лучше продаются, что в дальнейшем может быть полезно для оптимизации работы автосалона, выявить какой марки или в каком ценовом диапазоне закупать автомобили в большем количестве.

Use case диаграмма описывает с точки зрения действующего лица группу действий в системе, которые приводят к конкретному результату. Данная диаграмма является описанием типичных взаимодействий между пользователями системы и самой системой. Они отображают внешний интерфейс системы и указывают форму того, что система должна сделать.

На рисунке Б.1 приложения Б представлена диаграмма вариантов использования данной системы.

Актёрами на данной диаграмме являются администратор автосалона, пользователь, дилер и продавец. Теперь рассмотрим процессы, доступные для каждого из актёров.

Пользователь может просмотреть данные об автомобилях, оформить заявку, отсортировать данные об автомобилях и просмотреть свои заявки.

У администратора автосалона больший функционал работы с данными. В этот функционал входят как действия, доступные пользователю, а также администратору доступно составление отчёта, просмотр аккаунтов пользователей, их блокировка и разблокировка.

Дилеру доступно изменение, добавление, удаление данных об автомобилях, а также одобрение или отклонение заявки от пользователя.

А продавец может просматривать заявки от пользователей, направлять их дилеру или отклонять.

1.4 Разработка информационной модели предметной области.

Для описания структуры хранимых данных используется язык моделирования IDEF1X.

IDEF1X используется для создания графической информационной модели, которая представляет структуру и семантику информации в среде или системе. Использование этого стандарта позволяет создавать семантические модели данных, которые могут служить для поддержки управления данными как ресурсами, интеграции информационных систем и построения компьютерных баз данных.

В процессе проектирования информационной модели учёта автомобилей в автосалоне были выделены следующие сущности:

- пользователь;
- роль;
- автомобиль;
- заказ;
- статус заказа.

Рассмотрим более подробно представленные сущности.

Сущность «Пользователь» (user) хранит информацию о пользователях данного программного продукта: id, имя, фамилия, электронная почта. Поля логин и пароль обеспечивают сохранность выполняемых в последствии действий.

Сущность «Автомобиль» (Car) содержит в себе информацию о ключевом уникальном поле id автомобиля, его марке, модели, цене, цвете, годе выпуска.

Сущность «Роль» (Role) содержит прежде всего id роли и её наименование.

Сущность «Заказ» (Order) позволяет хранить информацию об id автомобиля, id пользователя, id статуса заказа.

Сущность «Статус заказа» содержит прежде всего id статуса и его наименование.

На рисунке 1.7 представлена структура хранимых данных разработанного программного приложения.

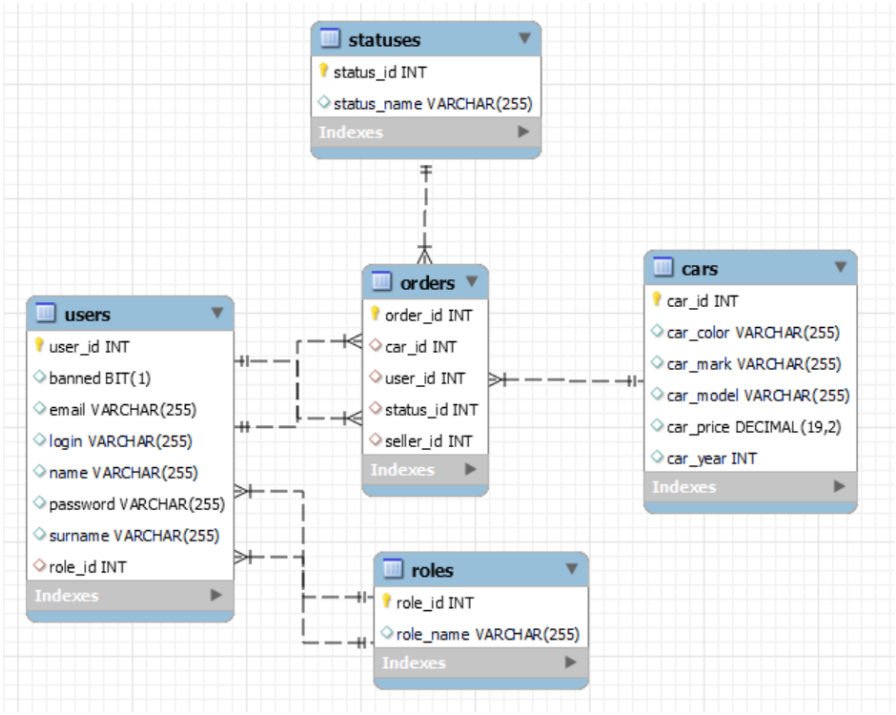

Рисунок 1.7 - Диаграмма IDEF1X

1.5 UML-модели представления программного средства и их описание.

Унифицированный язык моделирования (Unified Modeling Language, UML) – это графический язык для визуализации, специфицирования, конструирования и документирования систем, в которых главная роль принадлежит программному обеспечению.

Диаграмма – это графическое представление набора элементов, чаще всего изображенного в виде связного графа вершин (сущностей) и путей (связей).

На рисунке 1.8 представлена диаграмма последовательности, на которой отображен процесс взаимодействия пользователя и сервера.

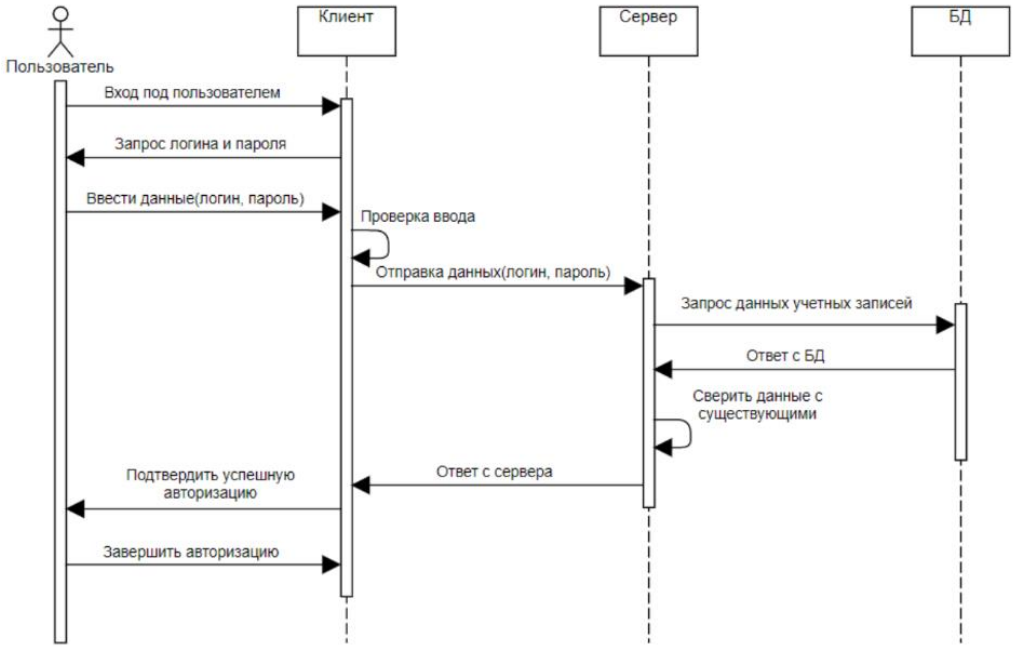

Рисунок 1.8 – Диаграмма последовательности

Сначала пользователь проверяет соединение с сервером, далее на экран сервера выводится информация о подключении. После этого клиент вводит логин и пароль. Эти данные отправляются на сервер, там они обрабатываются и выполняются различные операции над ними, затем информация записываются в файл. Когда данные записаны или не записаны в файл выводится соответствующее сообщение на экран. Клиент получает необходимую информацию.

Диаграмма последовательностей описывает поведенческие моменты системы, взаимодействие объектов в течение определённого отрезка времени. На диаграмме последовательностей могут присутствовать участники процесса, компоненты системы, и сообщения, которые они могут вернуть. После осуществления запроса на авторизацию и ввода логина и пароля, данные передаются вначале в приложение клиент, затем на сервер, где данные проверяются. Потом сервер запрашивает у базы данных данные имеющихся учётных записей и сравнивает полученные данные пользователя с имеющимися, после этого отправляет ответ клиенту, который передаёт ответ и передаёт управление в окно авторизации.

Диаграмма состояний описывает все возможные состояния процесса регистрации пользователя и последовательность его переходов из одного состояния в другое. Если пользователь ввел все данные корректно, то новый пользователь в конечном итоге перейдет в состояние «зарегистрированный пользователь». Если пользователь введет неверные данные или данные пользователя, который уже есть, то пользователь перейдет в состояния, связанные

с ошибкой ввода, после чего необходимо будет заново вводить данные. Данная диаграмма отображена на рисунке 1.9.

Рисунок 1.9 – Диаграмма состояний

Таким образом при помощи различных UML диаграмм удалось представить разработанное программное средство с разных ракурсов. На описанных диаграммах продемонстрированы основные составляющие программного приложения, представлены некоторые алгоритмы действий программы и клиентов, описаны возможные варианты действий.

2 ПРОЕКТИРОВАНИЕ И КОНСТРУИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

В данном курсовом проекте необходимо разработать программное средство, которое позволит оптимизировать работу автосалона.

Каждый проект – это индивидуальный подход и разработка уникальности от наполнения до продвижения с постоянной поддержкой. Поэтому при разработке программного средства необходимо знать факторы, определяющие заранее успех его функционирования, удобства работы.

Важным фактором является удовлетворение клиента полученным результатом работы разработчиков. Чем довольнее клиент, тем выше репутация у автосалона и тем большее количество клиентов автосалон может привлечь.

2.1 Постановка задачи

Согласно техническому заданию, задачей курсового проекта является создание программы для автоматизации системы учёта продаж автомобилей в автосалоне. Приложение должно быть выполнено в архитектуре клиент-сервер с многопоточным сервером с организацией взаимодействия с базой данных на объектно-ориентированном языке Java.

Для корректной работы необходимо предусмотреть возможность добавления, удаления и редактирования информации в базе данных. Для хранения информации будет использован MySQL Server. Подключение к нему будет осуществляться при авторизации и в зависимости от полученных прав, пользователь будет получать определённый набор возможностей для выполнения своей работы.

В автосалоне пользователь может отправить заявку на автомобиль, которую затем обработает дилер. Дилер будет добавлять автомобили в базу и обрабатывать заявки. Далее после принятого решения по заявкам клиенту на почту будет отправлено письмо со статусом заказа.

Для ведения отчетности, каждый администратор будет составлять отчет о продажах продавцов. Это позволит не только скорректировать работу продавцов, так и выявить тенденции посещаемости автосалона, что в дальнейшем может быть полезно для оптимизации его работы.

2.2 Архитектурные решения

Диаграмма классов (class diagram) показывает набор классов, интерфейсов и кооперaций, а также их связи. Диаграммы этого вида чаще всего

используются для моделирования объектно-ориентированных систем. Предназначены для статического представления системы. Диаграммы классов, включающие активные классы, представляют статическое представление процессов системы.

В данном проекте описаны следующие классы: Role (Роль), User (Пользователь), Car (Автомобиль), Order (Заказ), Status (статус заказа).

В классе Order содержатся такие поля, как carId (идентификационный номер машины), userId (идентификационный номер пользователя), statusId (идентификационный номер статус).

В классе User содержатся такие поля, как roleId (идентификационный номер), login (логин), password (пароль), name (имя пользователя), surname (фамилия пользователя) и email (электронная почта).

Класс Role имеет roleId (идентификационный номер), name (название роли).

Класс Car имеет поля mark (марка), model (модель), color (цвет), year (год выпуска), price (цена).

Класс Status имеет statusId (идентификационный номер), name (название статуса).

На рисунке А.1 приложения А представлена иерархия классов, используемых в данной программе.

Паттерн проектирования – часто встречающееся решение определённой проблемы при проектировании архитектуры программ.

В отличие от готовых функций или библиотек, паттерн нельзя просто взять и скопировать в программу. Паттерн представляет собой не какой-то конкретный код, а общую концепцию решения той или иной проблемы, которую нужно будет ещё подстроить под нужды вашей программы [4]

Можно выделить следующие основные группы паттернов:

- порождающие паттерны;
- структурные паттерны;
- поведенческие паттерны.

Паттерны, использованные в данном курсовом проекте, рассмотрены ниже.

2.2.1 Паттерн проектирования Command

Command используется в случае, когда необходимо послать объекту запрос, не зная о том, выполнение какой операции запрошено.

Шаблон Команда абстрагирует вызов и выполнение операции в программном коде системы. Это реализуется с помощью абстрактного класса

Command, который реализует команды, используемые в приложении. Например, метод execute(), который наследуется во всех классах-командах.

2.2.2 Паттерн проектирования Singleton

Одиночка – это порождающий паттерн проектирования, который гарантирует, что у класса есть только один экземпляр, и предоставляет к нему глобальную точку доступа.

Паттерн создает класс с одним экземпляром объекта, который его создает, инициализирует и дает непосредственный доступ к нему. Экземпляр является приватным и статическим. Статичность позволяет вызывать элемент программного кода без создания экземпляра повсеместно. Публичным в данном случае является метод, который инкапсулирует инициализацию и доступ к экземпляру. Диаграмма данного паттерна представлена на рисунке 2.1.

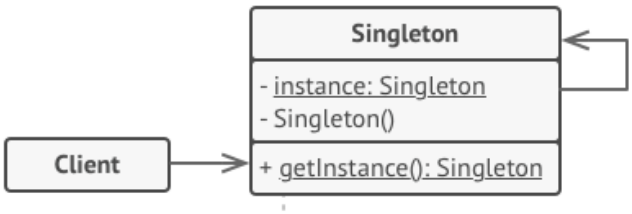

Рисунок 2.1 – Диаграмма паттерна проектирования Одиночка

2.2.3 Паттерн проектирования MVC

Данная концепция призвана упростить разработку, поддержку и изменение программы.

Разработанная система расчета заработной платы использует библиотеку пользовательского интерфейса JavaFX. При создании клиентского приложения сразу формируется некоторый каркас, включающий файлы fxml, контроллеры для каждого из них, а также некоторый класс для запуска программы. То есть структура Model-View-Controller заключена в разделении модели, представления и контроллера программы. Модель управляет данными, используемыми в программе; представление – это описание и реализация пользовательского интерфейса программы; контроллер – часть программы, регулирующая пользовательские запросы к модели через пользовательских интерфейс (рисунок 2.2).

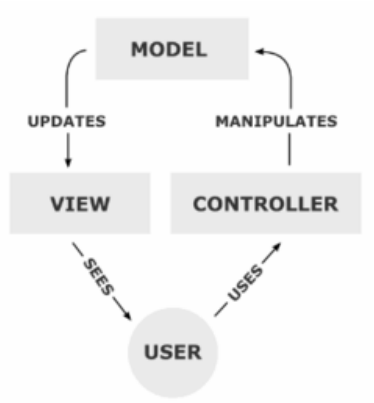

Рисунок 2.2 – Паттерн MVC

Рассмотренные паттерны упрощают и структурируют разработку программного средства, увеличивается эффективность работы, обеспечивая своевременную модернизацию различных частей кода, без оказания влияния на участки программы, не требующие изменений.

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

Основные алгоритмы, осуществляющие работу программы, – алгоритм смены статуса заявки и алгоритм добавления автомобиля в базу.

Схема алгоритма смены статуса заявки отображает, как будет происходить процесс подтверждения или отклонения заявки. После прохождения заявки у продавца, заявка направляется к дилеру.

Схема, представленная на рисунке В.1 приложения В, отображает работу этого алгоритма.

Добавление нового автомобиля в систему доступно только дилеру. Для этого ему нужно авторизоваться, перейти на нужную вкладку. Далее следует ввести все нужные данные. После того, как все поля будут заполнены, необходимо нажать кнопку «Сохранить». Если все было заполнено верно, то данные отправятся на сервер и данные будут сохранены в БД. Если дилер допустит ошибки при вводе значений, на экране появится сообщение об этом. Схема данного алгоритма представлена на рисунке В.2 приложения В.

2.4 Проектирование пользовательского интерфейса

Пользовательский интерфейс – это система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонен-

тов экрана (окон, значков, меню, кнопок, списков и т.п.). При этом, в отличие от интерфейса командной строки, пользователь имеет произвольный доступ (с помощью клавиатуры или указательного устройства ввода) ко всем видимым экранным объектам, а на экране реализуется модель мира в соответствии с некоторой метафорой и осуществляется прямое манипулирование.

Приложение имеет оконный интерфейс, разработанный с использованием JavaFX. JavaFX – платформа на основе Java для создания приложений с графическим интерфейсом.

При создании клиентского приложения сразу формируется некоторый каркас, включающий файлы fxml, контроллеры для каждого из них, а также некоторый класс для запуска программы.

2.5 Обоснование выбора компонентов и технологий для реализации программного средства

Клиент-серверная архитектура представляет собой иерархическую сеть, которая состоит из узлов-клиентов и центрального сервера, через который выполняется хранение и обработка данных, а также передача их в обоих направлениях. На текущий момент большая часть Интернета и локальных сетей используют именно архитектуру Клиент-Сервер для приема и передачи данных.

Клиент – это пользователь сервиса, который обращается к серверу для получения какой-то информации.

Сервер – место, где располагается веб-приложение или его серверная часть. Он владеет необходимой информацией о пользователях или может ее запрашивать. Также при обращении клиента сервер возвращает ему запрашиваемую информацию.

Клиент реализует пользовательский интерфейс. Для этого используется стандартные библиотеки пользовательского интерфейса JavaFX. Бизнес-логика курсового проекта выполняется на серверной части. Клиент отправляет запросы на сервер, где с помощью SQL-запросов будет добавляться, обновляться, удаляться и выбираться вся необходимая информация из БД.

Основной средой разработки был выбран IDE IntelliJ IDEA. MySQL Server используется в качестве СУБД. Enterprise Architect используется для разработки и построения UML-диаграмм. CASE-средство CA AllFusion Process Modeler r7 (BPwin) используется для проектирование IDEF0 модели. CA AllFusion ERwin Data Modeler r7 (ERwin) используется для проектирования информационной модели.

3 ТЕСТИРОВАНИЕ И ПРОВЕРКА РАБОТОСПОСОБНОСТИ ПРОГРАММНОГО СРЕДСТВА

В данной главе рассмотрены случаи поведения системы при создании очевидно ошибочных ситуаций. Ожидается, что система сообщит о некорректных действиях, а также предоставит понятное сообщение, чтобы можно было продолжить работу правильно.

Первая тестируемая ситуация – авторизация. Пользователь может ввести в целом неверные данные, либо неверное сочетание логина и пароля. Разработанная система отслеживает эти действия и выводит на экран соответствующие сообщения (рисунок 3.1 и 3.2):

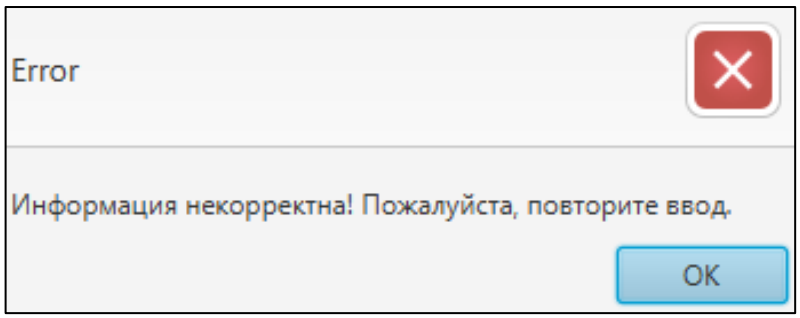

Рисунок 3.1 – Ввод неверных данных

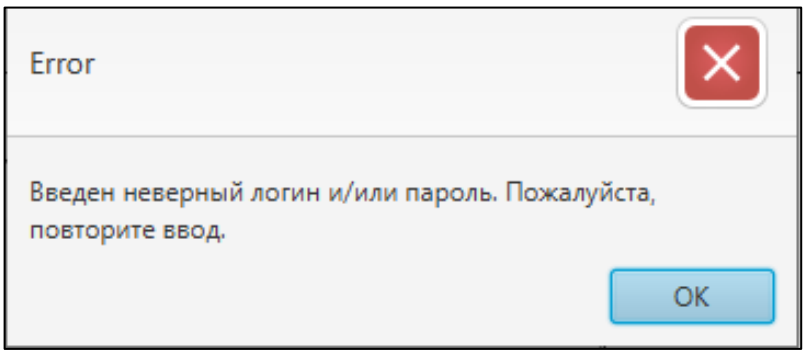

Рисунок 3.2 – Неправильный ввод логина и/или пароля

Следующая рассматриваемая ситуация – изменение пароля в личном кабинете пользователя. Если пользователь совершит ошибку, пароль изменен не будет, а на экране появится сообщение (рисунок 3.3):

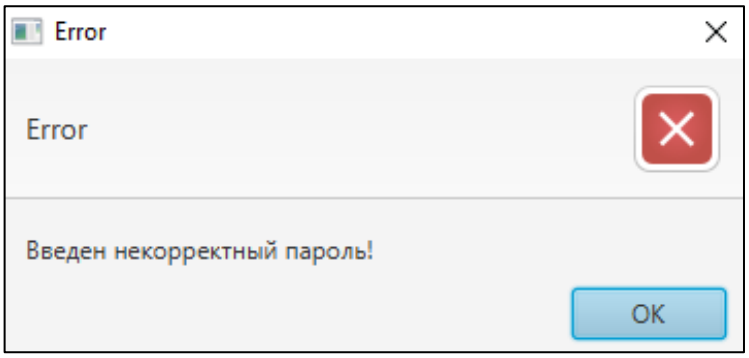

Рисунок 3.3 – Ошибка при изменении пароля

Если при изменении электронной почты ввести не валидное значение, будет выведено следующее сообщение об ошибке (рисунок 3.4):

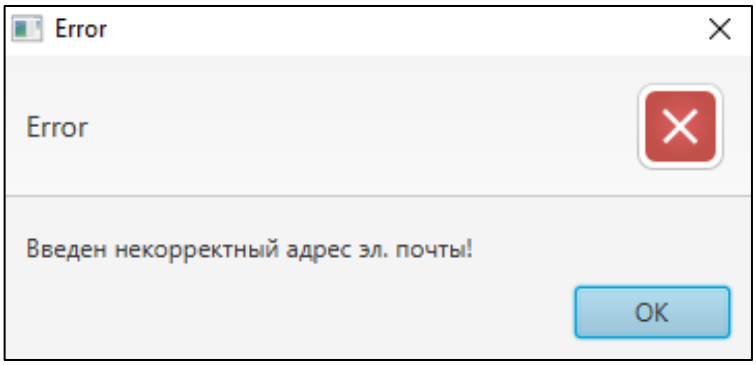

Рисунок 3.4 – Неверный ввод электронной почты

При добавлении нового автомобиля необходимо заполнить множество полей. На них предусмотрены проверки (рисунок 3.5):

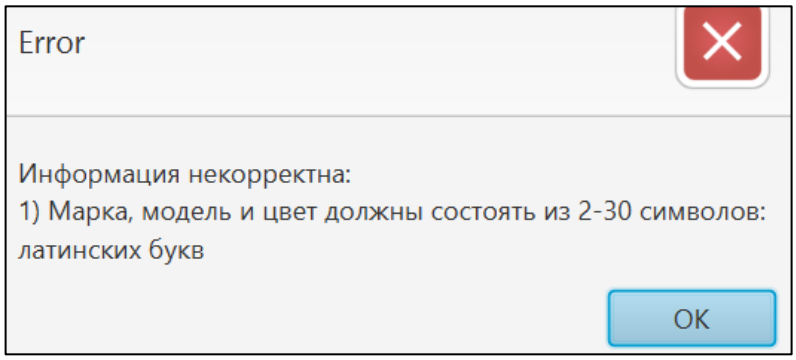

Рисунок 3.5 – Неверный ввод данных

При добавлении нового автомобиля также предусмотрена проверка на год выпуска, он должен быть в пределах от 1950 до 2023 (рисунок 3.5):

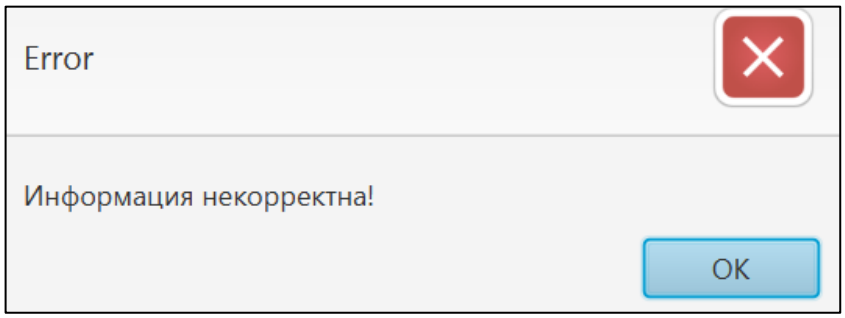

Рисунок 3.5 – Неверный ввод данных

Валидация данных при регистрации на рисунке 3.6:

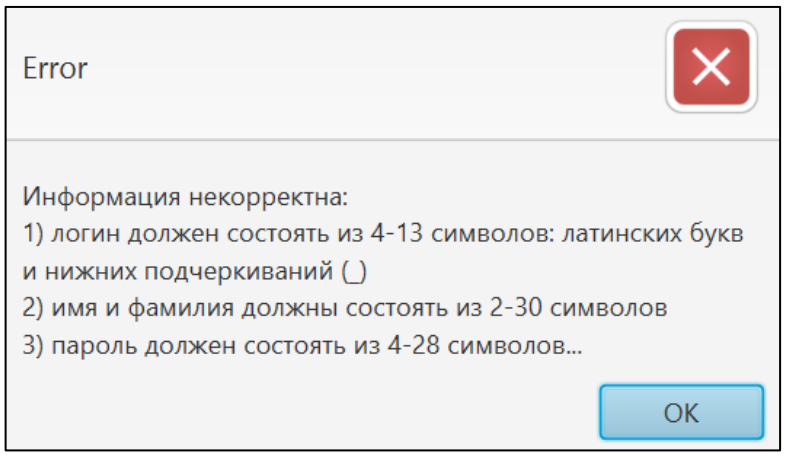

Рисунок 3.6 – Валидация при регистрации

Тестирование разработанной системы показало, что любые неверные действия пользователя сопровождаются понятным комментарием, что позволяет исправить введенные данные и продолжить работу. Выявленные в процессе тестирования неточности были успешно устранены и протестированы еще раз.

4 РУКОВОДСТВО ПО РАЗВЕРТЫВАНИЮ И ИСПОЛЬЗОВАНИЮ ПРОГРАММНОГО СРЕДСТВА

Развертывание программного обеспечения - один из наиболее важных аспектов процесса разработки программного обеспечения.

Развертывание программного обеспечения включает в себя все шаги, процессы и действия, необходимые для того, чтобы сделать систему программного обеспечения или обновление доступными для предполагаемых пользователей.

Для запуска программного средства необходимо предварительно установить на компьютере Intellij IDEA.

Для того, чтобы установить проект необходимо совершить следующие действия:

- скачать и установить базу данных MySQL;
- присоединить базу данных к приложению;
- запустить скрипт для установления некоторых данных в базу.

Для запуска проекта необходимо:

- открыть приложение клиента и сервера;
- запустить сначала серверную, затем клиентскую часть.

После запуска сервера появляется окно с информацией о работе сервера (см. рисунок 4.1):

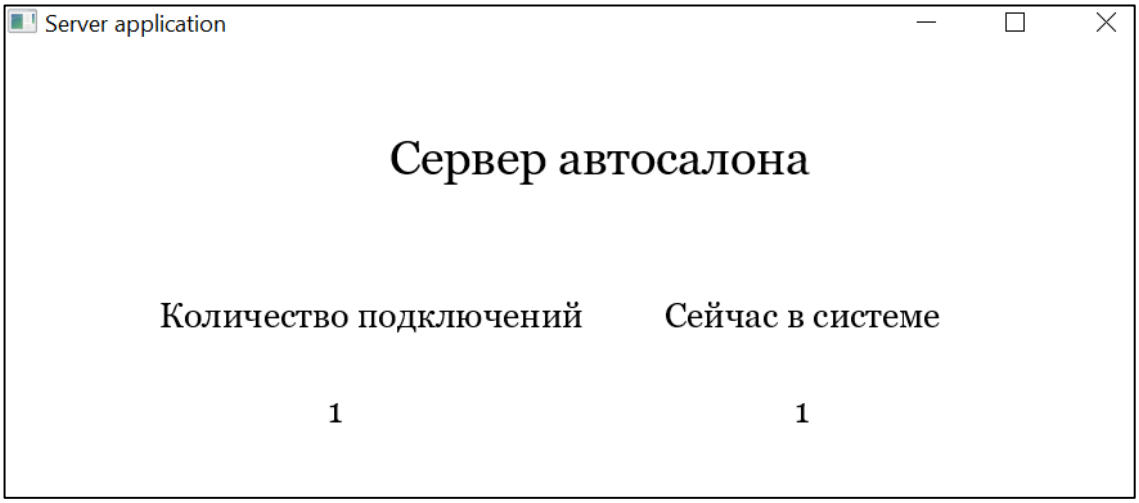

Рисунок 4.1 – Информация сервера

Появляется главное окно, в котором можно авторизоваться либо зарегистрироваться (см. рисунок 4.2):

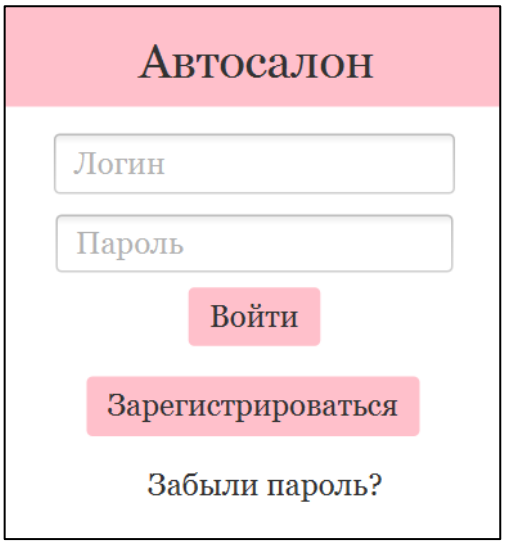

Рисунок 4.2 – Главное окно

После того, как мы выберем кнопку зарегистрироваться, появляется окно регистрации (см. рисунок 4.3):

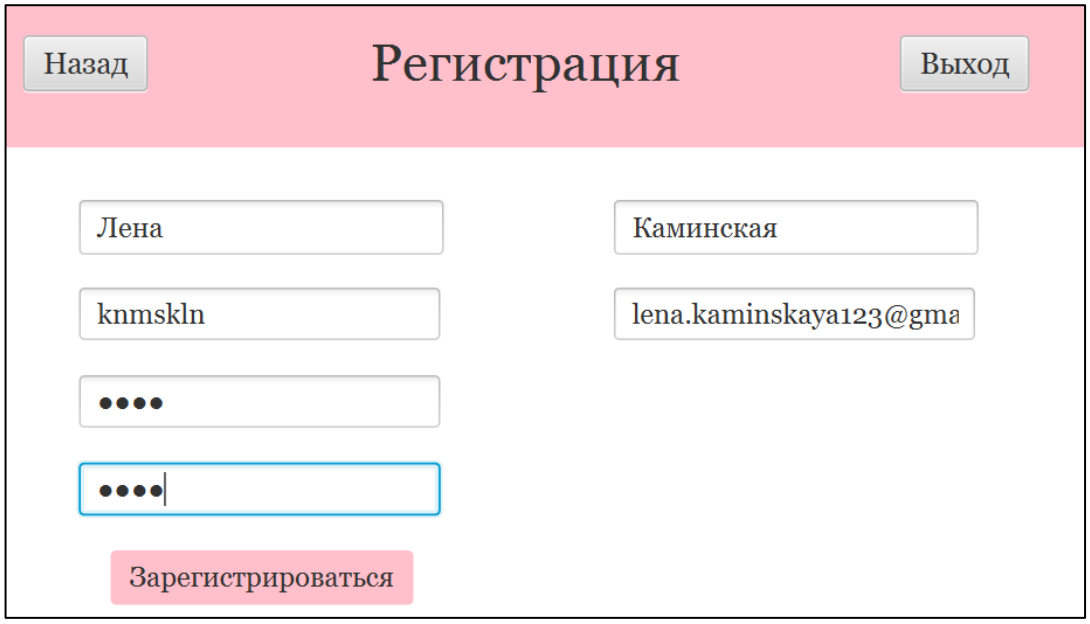

Рисунок 4.3 – Регистрация

Войдём под логином и паролем пользователя, открывается меню, где мы можем просмотреть автомобили, просмотреть свой профиль и просмотреть свои заявки (см. рисунок 4.4):

Рисунок 4.4 – Меню пользователя

Откроем базу автомобилей, появляются автомобили, на которые можно оставить заявку, а также сохранить их в файл (см. рисунок 4.5):

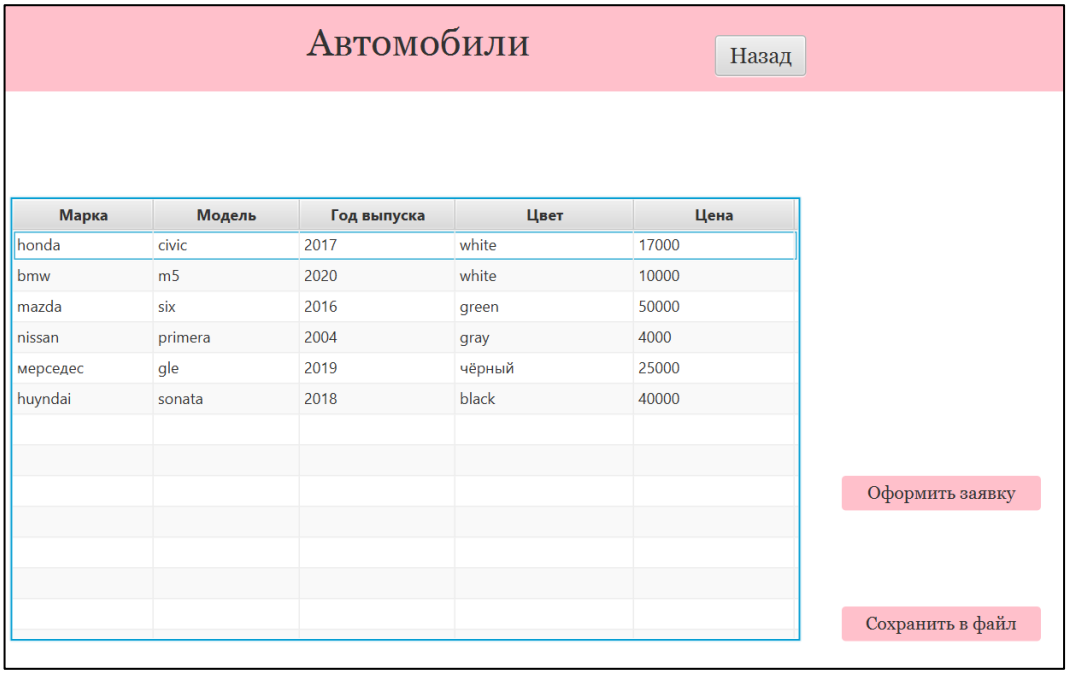

Рисунок 4.5 – База автомобилей

После оформления заявки можно просмотреть свои заявки и их статусы (см. рисунок 4.6):

Рисунок 4.6 – Просмотр заявок

Зайдём с помощью логина и пароля продавца, продавец может также смотреть профиль, просматривать базу автомобилей и смотреть заявки, которые были направлены продавцу на рассмотрение (см. рисунок 4.7):

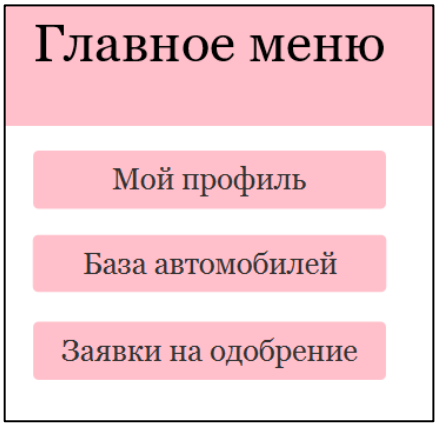

Рисунок 4.7 – Меню продавца

Кликнем на заявки, которые пришли на одобрение, выбирая заявку, появляются кнопки с действиями, если направляем заявку дилеру или отклоняем, то её статус меняется (см. рисунок 4.8):

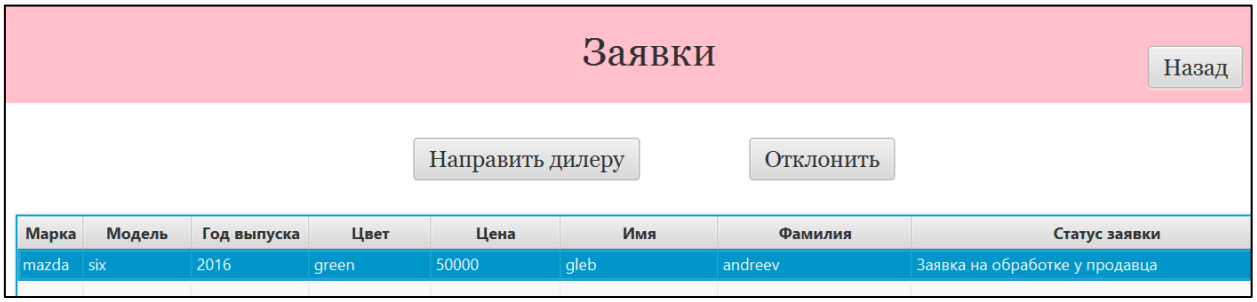

Рисунок 4.8 – Заявки на одобрение

Зайдём с помощью логина и пароля дилера, меню идентичное с продавцом, но когда мы зайдём в базу автомобилей, у нас появляется возможность добавить, отредактировать и удалить автомобиль (см. рисунок 4.9):

Рисунок 4.9 – База автомобилей дилера

Кликнем на добавить новый автомобиль, появляется дополнительное окно, вводим в ячейки марку, модель, год выпуска, цвет и цену (см. рисунок 4.10):

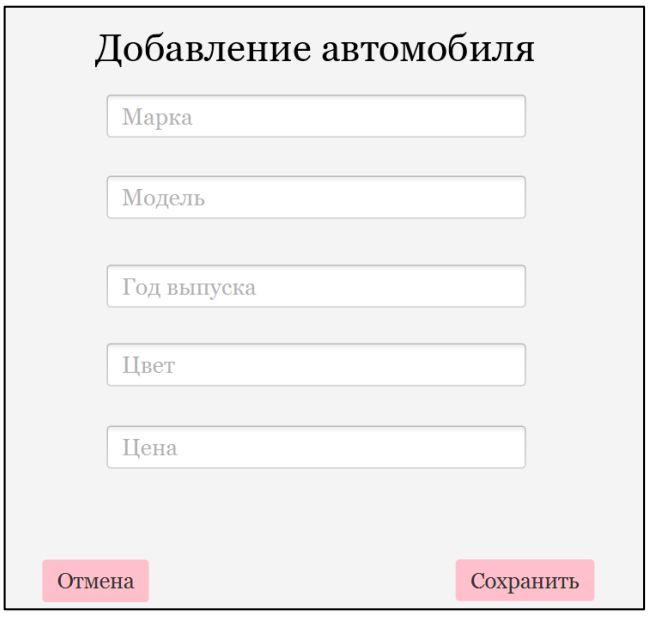

Рисунок 4.10 – Добавление автомобиля

Кликнем два раза по строчке с автомобилем, который хотим отредактировать и появляется дополнительное окно (см. рисунок 4.11):

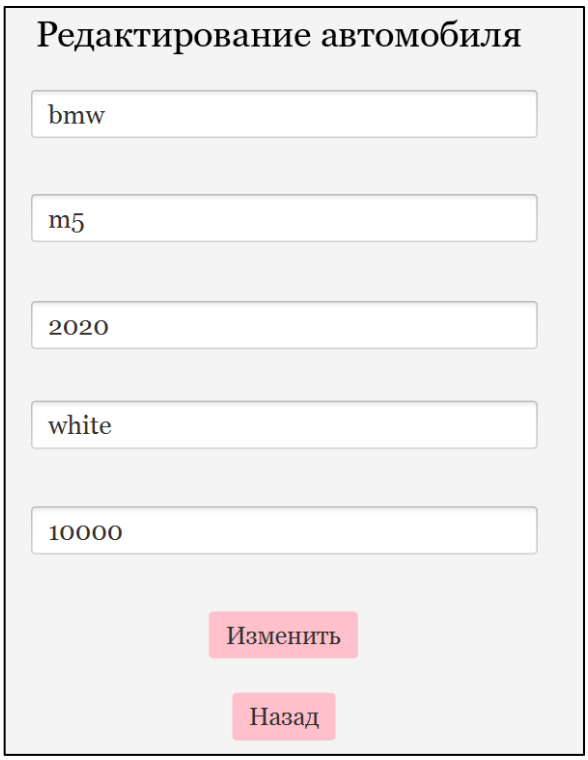

Рисунок 4.11 – Редактирование автомобиля

Просмотрим заявки на одобрение, при нажатии на строчку с заявкой появляются кнопки подтвердить или отклонить (см. рисунок 4.12):

Рисунок 4.12 – Просмотр заявок дилера

После одобрения или отклонения заявки пользователю на почту приходит письмо (см. рисунок 4.13):

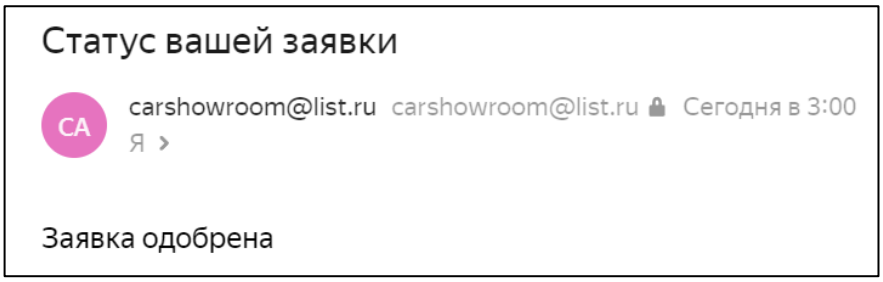

Рисунок 4.13 – Сообщение о статусе заявки

Зайдём с помощью логина и пароля администратора, появляется меню, в котором мы можем просмотреть личные данные, базу автомобилей, просмотреть все заявки и их статусы, просмотреть всех пользователей и просмотреть статистику (см. рисунок 4.14):

Рисунок 4.14 – Меню администратора

Выберем просмотр всех заявок (см. рисунок 4.15):

Рисунок 4.15 – Просмотр всех заявок

Кликнем кнопку все пользователи, здесь мы увидим всех пользователей, которые зарегистрированы в системе, а также мы можем заблокировать пользователя (см. рисунок 4.16):

Рисунок 4.16 – Действия со всеми пользователями

При клике на кнопку Статистика появляется окно, где мы видим продажи, т.е. только те заявки, которые одобрены дилером (см. рисунок 4.17):

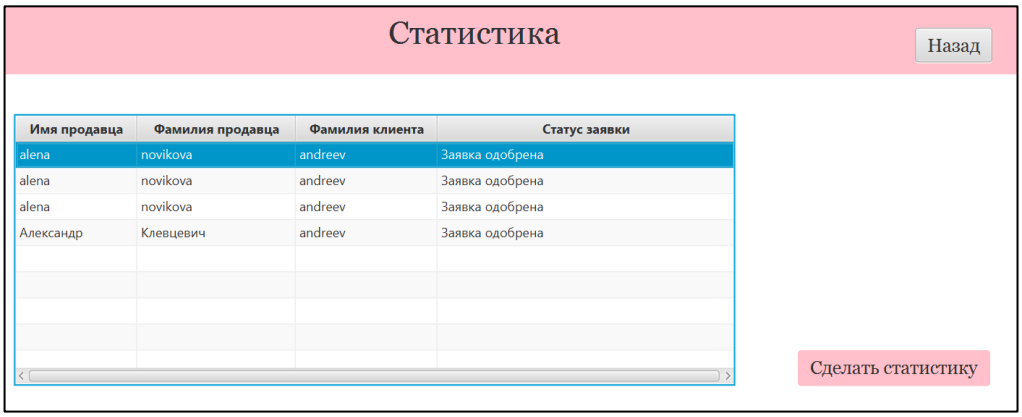

Рисунок 4.17 – Продажи

При клике на продавца и кнопку Сделать статистику у нас выводится количество продаж и KPI продавца (см. рисунок 4.17):

Рисунок 4.17 – Продажи

ЗАКЛЮЧЕНИЕ

При разработке данного приложения была исследована предметная область, которая показала, что при хранении огромного количества информации высока вероятность ошибки и все это можно устранить при помощи разработки специализированного приложения.

Итогом данной работы является функционирующее приложение. При разработке и написании программы использовались принципы объектно-ориентированного проектирования и программирования, основные средства языка Java. Оно предоставляет возможность осуществления различных процедур с данными.

Для правильной разработки проекта необходима тщательная проработка предметной области.

Приложение было разработано на основе клиент-серверной архитектуры, что позволяет увеличить производительность работы приложения.

Так же были решены следующие задачи:

– были реализованы классы и методы для работы с данными;

– были разработаны схемы алгоритмов работы приложения;

– была разработана функциональная тема задачи;

– была описана программа.

При разработке данного программного продукта была учтена логика пользователя и проанализированы вероятные сценарии работы программы (возможные ошибки ввода). Важной особенностью программы является лёгкая модифицируемость, в дальнейшем это поможет вносить как незначительные, так и весомые изменения и модификации. Это является огромным плюсом, ведь любой программный продукт без совершенствования устаревает и приходит в ненадобность. Поэтому созданное программное приложение нацелено на продолжительное использование и открыто для модификаций и успешного развития.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Репин В.В. Разработка архитектуры бизнес-процессов компании в Business Studio / В.В. Репин. – М.: Манн, Иванов и Фербер, 2019.–142с.

[2] Скотт К., Фауер М. UML. Основы. 3-е издание. / К. Скотт, М. Фаулер. – М.: Символ, 2016. – 192 с.

[3] Кретов И.И. Организация маркетинга на преприятии: практич. пособие. / И.И. Кретов. – М.: Юристъ, 2001. – 96 с.

[4] Паттерны проектирования [Электронный ресурс]. – Режим доступа: https://refactoring.guru/ru/design-patterns/what-is-pattern]

[5] Черемных, С. Моделирование и анализ систем. IDEF – технологии. /С. Черемных— Москва: Финансы и статистика,2006. — 188 с.

[6] Дейт, К. Введение в системы баз данных. Восьмое издание / К. Дейт- Москва, Санкт-Петербург, Киев.: Издательский дом «Вильяме», 2005. - 1328 с.

[7] Клиент-сервер [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.mstu.edu.ru/study/materials/zelenkov/ch

ПРИЛОЖЕНИЕ А
(обязательное)
Диаграмма классов

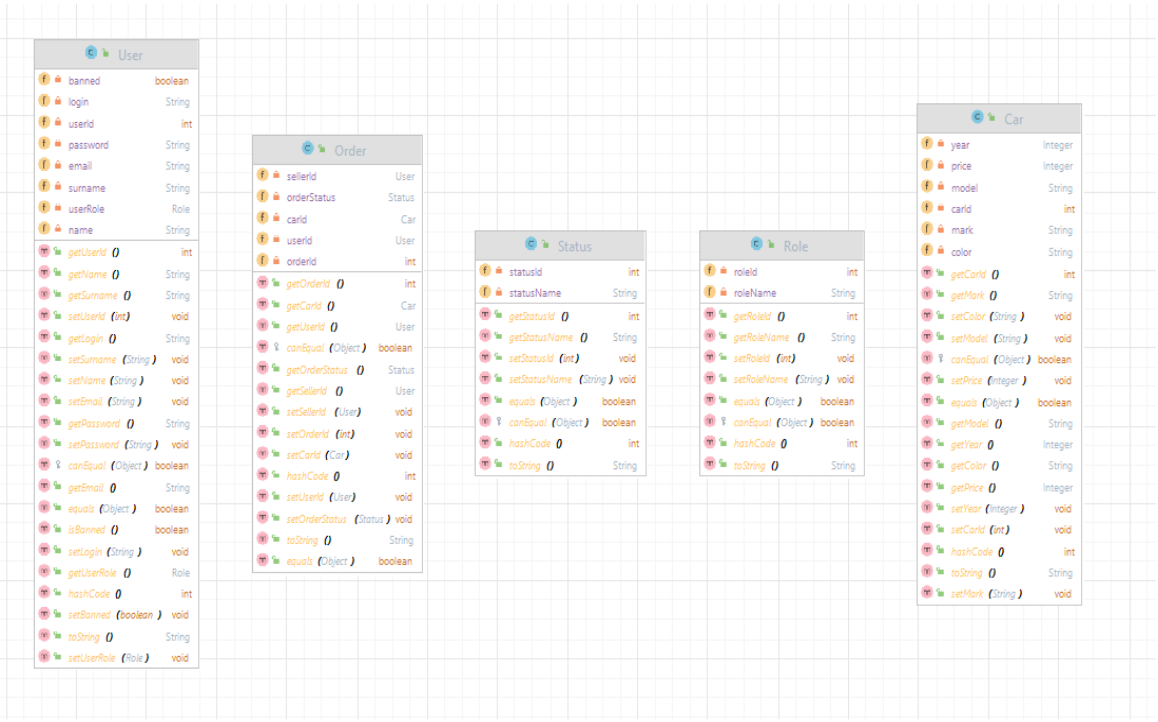

Рисунок А.1 – Диаграмма классов

ПРИЛОЖЕНИЕ Б
(обязательное)
Диаграмма вариантов использования

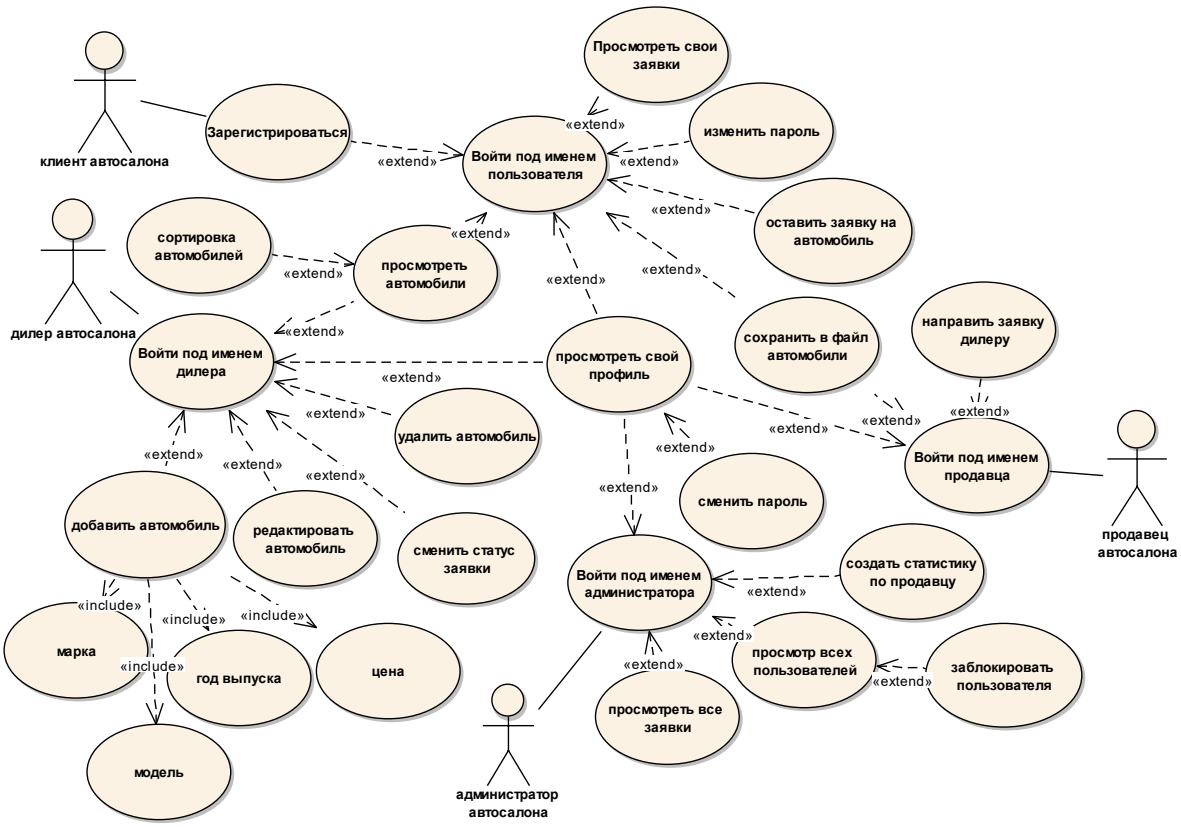

Рисунок Б.1 – Диаграмма вариантов использования

ПРИЛОЖЕНИЕ В
(обязательное)
Алгоритмы работы программы

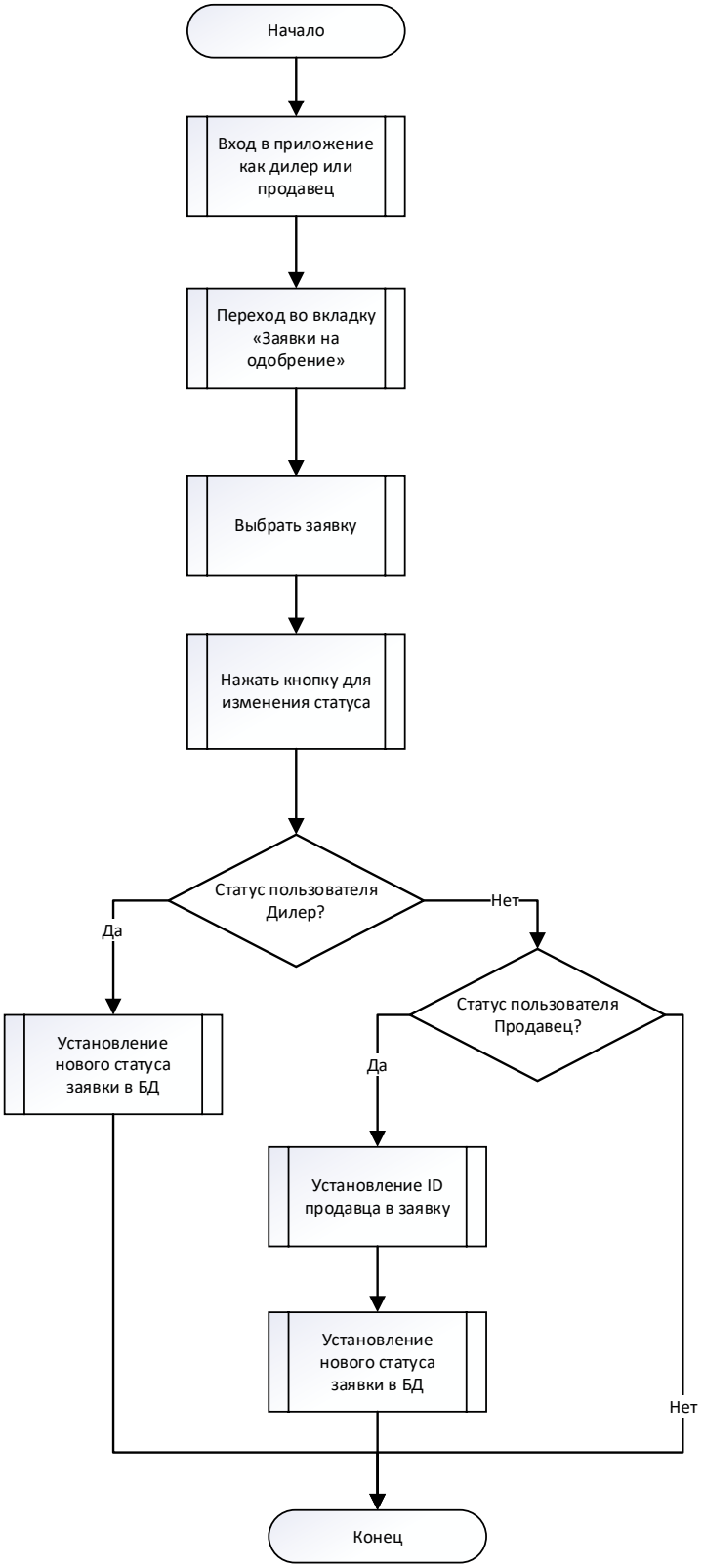

Рисунок В.1 – Схема алгоритма смены статуса заявки

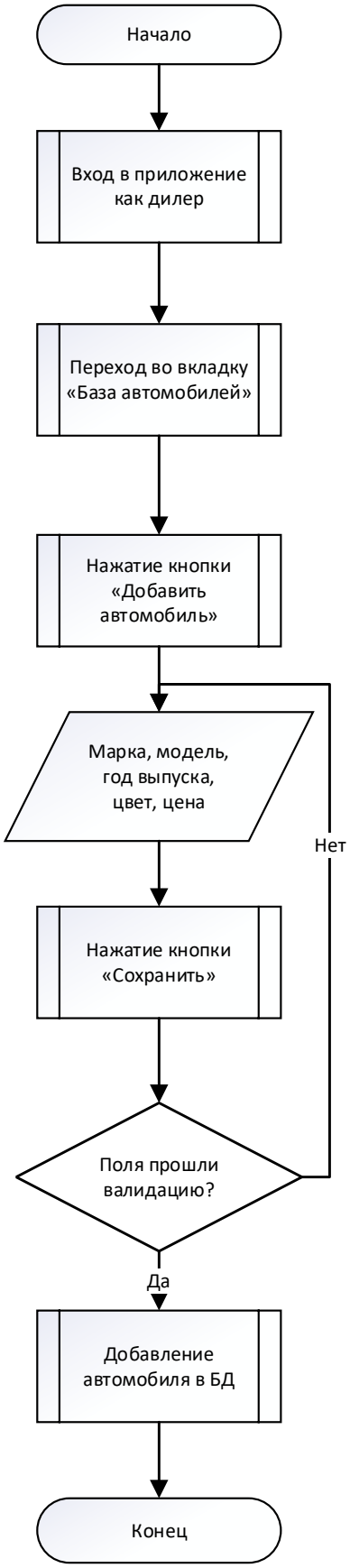

Рисунок В.2 – Схема алгоритма добавления автомобиля

ПРИЛОЖЕНИЕ Г
(обязательное)
Листинг кода

Сервер:

```
public class CommandFactory {
   private static final CommandFactory INSTANCE = new CommandFacto-
ry();
   public static CommandFactory getInstance() {
      return INSTANCE;
   }
   private CommandFactory() {
   }
   public Command createCommand(String name, ClientRequest request,
ServerResponse response) {
      System.out.println("Выполняется команда: " + name);
      switch (name) {
         case "signIn":
            return new SignInCommand(request, response);
         case "signUp":
            return new SignUpCommand(request, response);
         case "restorePassword":
            return new RestorePasswordCommand(request, response);
         case "getAllCars":
            return new GetAllCarsCommand(request, response);
         case "addCar":
            return new AddCarCommand(request, response);
         case "deleteCar":
            return new DeleteCarCommand(request, response);
         case "getUser":
            return new GetUserCommand(request, response);
         case "getOrderByUserId":
            return new GetOrderByUserIdCommand(request, response);
         case "getOrderByStatus":
            return new GetOrderByStatusIdCommand(request, response);
         case "changeEmail":
            return new ChangeEmailCommand(request, response);
         case "changePassword":
            return new ChangePasswordCommand(request, response);
```

```
                case "editCar":
                    return new EditCarCommand(request, response);
                case "getCarById":
                    return new GetCarByIdCommand(request, response);
                case "getAllUsers":
                    return new GetAllUsersCommand(request, response);
                case "changeBanStatus":
                    return new ChangeBanStatusCommand(request, response);
                case "changeOrderStatus":
                    return new ChangeOrderStatusCommand(request, response);
                case "orderCar":
                    return new OrderCarCommand(request, response);
                case "getAllOrders":
                    return new GetAllOrders(request, response);
                case "getOrdersForStatistics":
                    return new GetOrdersForStatisticsCommand(request,response);
                case "getCountApproved":
                    return new GetCountApprovedCommand(request,response);
            }
            throw new RuntimeException("Команда не найдена");
        }
    }

    public class AddCarCommand implements Command {
        private DealerService service;
        private ClientRequest request;
        private ServerResponse response;

        public AddCarCommand(ClientRequest request, ServerResponse re-
sponse) {
            this.service = DealerServiceImpl.getInstance();
            this.request = request;
            this.response = response;
        }

        @Override
        public ServerResponse execute() throws CommandException {
            Map<String, Object> data = request.getData();
```

```
            String mark = (String) data.get("mark");
            String model = (String) data.get("model");
            Integer year = (Integer) data.get("year");
            String color = (String) data.get("color");
            Integer price = (Integer) data.get("price");

            try {
                service.addCar(mark, model, year, color, price);
            } catch (ServiceException e) {
                throw new CommandException(e);
            }
            return response;
        }
    }

    public class DealerServiceImpl implements DealerService {
        private static final DealerServiceImpl INSTANCE = new DealerService-
Impl();

        public static DealerServiceImpl getInstance() {
            return INSTANCE;
        }

        private DealerServiceImpl() {
        }
        private final CarInformationValidator carValidator = CarInformationVali-
dator.getInstance();
        private final CarRepository carRepository = CarJpaReposito-
ry.getInstance();
        @Override
        public void deleteCar(int carId) throws ServiceException {
            try {
                carRepository.delete(carId);
            } catch (RepositoryException e) {
                throw new ServiceException(e);
            }
        }
        @Override
```

```
        public void addCar(String mark, String model, Integer year, String color,
Integer price) throws ServiceException {
            if (!carValidator.validate(mark, model, color,year, price)) {
                throw new ServiceException("Информация некорректна!
Пожалуйста, повторите ввод.");
            }
            try {//
                Car car = new Car();
                car.setMark(mark);
                car.setModel(model);
                car.setYear(year);
                car.setColor(color);
                car.setPrice(price);
                carRepository.add(car);
            } catch (RepositoryException e) {
                throw new ServiceException(e);
            } }}
```

Клиент:

```
public class AddCarController {
    @FXML
    private Button addCar;
    @FXML
    private Button cancel;
    @FXML
    private TextField mark;
    @FXML
    private TextField model;
    @FXML
    private TextField year;
    @FXML
    private TextField color;
    @FXML
    private TextField price;
    private MapParser parser = MapParser.getInstance();
    private CarInformationValidator validator =
CarInformationValidator.getInstance();
    @FXML
    private void initialize() {
```

```
            if (!"dealer".equals(Runner.getStatus().getRoleName())) {
                addCar.setVisible(false);
            }
            addCar.setOnAction(event -> addCar());
            cancel.setOnAction(event -> cancel.getScene().getWindow().hide());
        }
        private void addCar() {
            String markText = this.mark.getText();
            String modelText = this.model.getText();
            String yearText = this.year.getText();
            String colorText = this.color.getText();
            String priceText = this.price.getText();
            if (validator.validate(markText, modelText, colorText)){
                Map<String, Object> data = new HashMap<>();
                data.put("mark", markText);
                data.put("model", modelText);
                data.put("year", Integer.parseInt(yearText));
                data.put("color", colorText);
                data.put("price", Integer.parseInt(priceText));
                Runner.sendData(new ClientRequest("addCar", data));
                ServerResponse response = Runner.getData();
                if (!response.isError()) {
                    addCar.getScene().getWindow().hide();
                    Alert alert = new Alert(INFORMATION, "Автомобиль успешно
добавлен!");
                    alert.show();
                } else {
                    Alert alert = new Alert(ERROR, "Информация некорректна!");
                    alert.show();
                }
            }
            else {
                Alert alert = new Alert(ERROR, "Информация некорректна:\n" +
                        "1) Марка, модель и цвет должны состоять из 2-30 симво-
лов: латинских букв\n");
                alert.show();
            }
        }
    }
```