|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Государственное бюджетное профессиональное образовательное учреждение Новосибирской области | | | | | | | | |
| **«Новосибирский химико-технологический колледж им. Д.И. Менделеева»** | | | | | | | | |
| (ГБПОУ НСО «Новосибирский химико-технологический колледж им. Д.И.Менделеева») | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
| **КУРСОВАЯ РАБОТА** | | | | | | | | |
| (Курсовой проект) | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
|  | | | | | | | | |
| на тему: Разработка программного обеспечения для аренды автомобилей | | | | | | | | |
|  | |  | | | | | | |
| Специальность: | | *Информационные системы и программирование* | | | | | | |
|  | |  | | | | | | |
| Выполнил | | *Верёвченко Максим Владимирович* | | | | | | |
| студент | | | | | | | | |
|  | группы | 09.07.10 |  | 3 | курса | очной | | формы обучения |
|  |  |  |  |  |  | *очной, заочной* | |  |
| Руководитель | | |  | | |  |  | |
|  | | | *подпись* | | |  | фамилия, имя, отчество | |
| Председатель ПЦК | | |  | | |  |  | |
|  | | |  | | |  |  | |
| Зав. отделением | | | *подпись* | | |  | *фамилия, инициалы* | |
|  | | |  | | |  |  | |
|  | | | *подпись* | | |  | *фамилия, инициалы* | |
| Новосибирск 2023 | | | | | | | | |

ВВЕДЕНИЕ

Аренда автомобиля избавляет человека от множества хлопот. Благодаря данной услуге можно существенно сэкономить. Нет необходимости покупать личный автомобиль или просить друга об одолжении – предоставить машину на определенное время.

Нет нужды пользоваться услугами такси, постоянно ездить на маршрутках. Об аренде автомобиля во время командировки лучше побеспокоиться заранее, чтобы не терять время. За несколько дней до поездки можно найти в Интернете подходящую точку проката в городе, в который Вы отправляетесь. Договориться с представителями компании можно удаленно, представив все необходимые документы через электронную почту или специальную форму на сайте за любое время до командировки.

Для некоторых людей аренда автомобиля – прекрасный шанс обкатать машину, которую они собираются купить. Например, понравилась новая модель BMW X5. Можно арендовать конкретный автомобиль на день. Таким образом, есть возможность оценить управляемость транспортного средства, уровень комфортности, скоростные характеристики и другие важнейшие параметры. Такой тест-драйв позволяет создать полноценное впечатление о покупаемом автомобиле. То есть клиент на практике убеждается, подходит ли ему конкретная модель или нет.

В соответствии с поставленной целью в работе определены следующие задачи:

* Разработать техническое задание.
* Изучить предметную область.
* Провести анализ возможностей разработки в секторе инструментальных прикладных приложений (в том числе анализ уже существующих разработок по теме).
* Описать основные этапы проектирования и разработки приложения.
* Учитывая полученные на этапе анализа результаты, разработать приложение для ПК в соответствии с техническим заданием.

1 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ.

1.1 Техническое задание.

1.1.1 Введение.

1.1.1.1 Наименование программного продукта или программы: «RentalCar».

1.1.1.2 Краткая характеристика области применения: аренда автомобилей.

1.1.2 Основания для разработки.

1.1.2.1 Основания для проведения разработки: задание курсового проекта от преподавателя ГБПОУ НСО «Новосибирский химико-технологический колледж им. д. и. Менделеева».

1.1.2.2 Наименование и условное обозначение разработки: аренда автомобилей.

1.1.3.Назначение разработки.

1.1.3.1 Функциональное назначение: аренда автомобилей.

1.1.3.2 Эксплуатационное назначение: для использования приложения пользователями.

1.1.4 Требования к продукту.

1.1.4.1 Программа должна обеспечивать возможность выполнения перечисленных ниже функций:

* Предоставление интерфейса администрирования.
  + Добавление/удаление автомобилей для аренды.
* Предоставление возможности аренды автомобиля.
* Отправка заявки на аренду автомобиля.
* Сохранение/загрузка состояния базы данных.
* Синхронизация данных на разных устройствах.

1.1.4.2 Организация входных данных: название приложения, иконки, картинки, формы QT, файл ресурсов QT, логотип, макет приложения, краткое описание функций приложения.

1.1.4.3 Организация выходных данных: исполняемый файл EXE, пакет файлов извлеченных из исполняемого файла при использовании, пакет файлов генерируемых приложением.

1.1.4.4 Временные характеристики: время работы пользователя в программе.

1.1.4.5 Требования к функциональным характеристикам.

1.1.4.5.1 Поддерживаемая операционная система: Windows 10, некоторые дистрибутивы Linux.

1.1.4.5.2 Минимальные системные требования.

* Процессор 2 ядра, с тактовой частотой: 1600 Mhz.
* Объем жёсткого диска: не менее 1 Gb свободной памяти.
* Видеоадаптер с объемом памяти: 256 Mb.
* Объём оперативной памяти: 2 Gb.

1.1.5 Требования к программной документации.

1.1.5.1 Состав программной документации.

* Инструкция по инсталляции.
* Руководство по эксплуатации пользователем программного продукта.

1.1.7 Календарный план работ (сроки разработки).

Таблица 1 – «Календарный план»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| № этапа | Название этапа | Исполнители | Сроки выполнения | Чем заключается этап | Отметка о выполнении этапа |
| 1 | Сбор информации, анализ требований к приложению | Верёвченко М. В. | 26.11.22 г. | Электронный вариант информации | ✓ |
| 2 | Оформление технического задания | Верёвченко М. В. | 28.12.22 г. | Техническое задание | ✓ |
| 3 | Проектирование | Верёвченко М. В. | 04.01.23 г. | Написание приложения | ✓ |
| 4 | Тестирование программного продукта | Верёвченко М. В. | 20.02.23 г. | Тестирование приложения  на ошибки | ✓ |
| 5 | Сопровождение (редактирование кода) | Верёвченко М. В. | 15.03.23 г. | Готовый программный продукт | ✓ |

1.1.8 Приложения.

* Инструкцию по инсталляции.
* Инструкции по эксплуатации:
* Для пользователя.
* Для администратора.

1.2 Описание выбранного инструментария.

1.2.2 Выбор языка программирования.

Язык программирования Python 3 — это мощный инструмент для создания программ самого разнообразного назначения, гибок, полностью удовлетворяет требованием, не требует компиляции, кроссплатформенный.

1.2.3 Выбор среды разработки.

PyCharm — это среда программирования для языка Python, или IDE от компании JetBrains. Средами называют программы, в которых можно писать, запускать и отлаживать код, устанавливать новые расширения и дополнительные модули. Это мощный многофункциональный инструмент для разработчиков.

1.2.4 Выбор фреймворков.

Для выбора технологий разработки GUI необходимо учесть множество факторов.

1.2.4.1 Qt.

Qt — фреймворк для разработки кроссплатформенного программного обеспечения на языке программирования C++. Для многих языков программирования существуют наборы библиотеки, позволяющие использовать преимущества Qt: Python — PyQt, PySide; Ruby — QtRuby; Java — Qt Jambi; PHP — PHP-Qt и другие.

Также будет использоваться прикладная программа QtDesigner для создания Qt форм.

1.2.4.2 unittest.

Фреймворк модульного тестирования. Среда unittest модульного тестирования изначально была вдохновлена ​​JUnit и имеет тот же вкус, что и основные среды модульного тестирования на других языках. Он поддерживает автоматизацию тестирования, совместное использование кода установки и завершения тестов, объединение тестов в коллекции и независимость тестов от структуры отчетности.

1.2.5 Выбор библиотек.

Библиотека — это набор готовых функций, классов и объектов для решения каких-то задач.

1.2.5.1 QT-PyQt-PySide-Custom-Widgets.

Удивительные пользовательские виджеты, созданные для настольных приложений QT. Упростите процесс разработки пользовательского интерфейса. Эти виджеты можно использовать в QT Designer, а затем импортировать в код PySide.

1.2.5.2 PySide2.

PySide2 — официальный модуль Python из проекта Qt for Python , предоставляющий доступ ко всему фреймворку Qt 5.12+.

1.3 Анализ существующих решений в предметной области.

1.3.1 Делимобиль.

Приложение для совместного использования поездок, которое позволяет пользователям арендовать автомобиль или заказать такси через свои мобильные устройства.

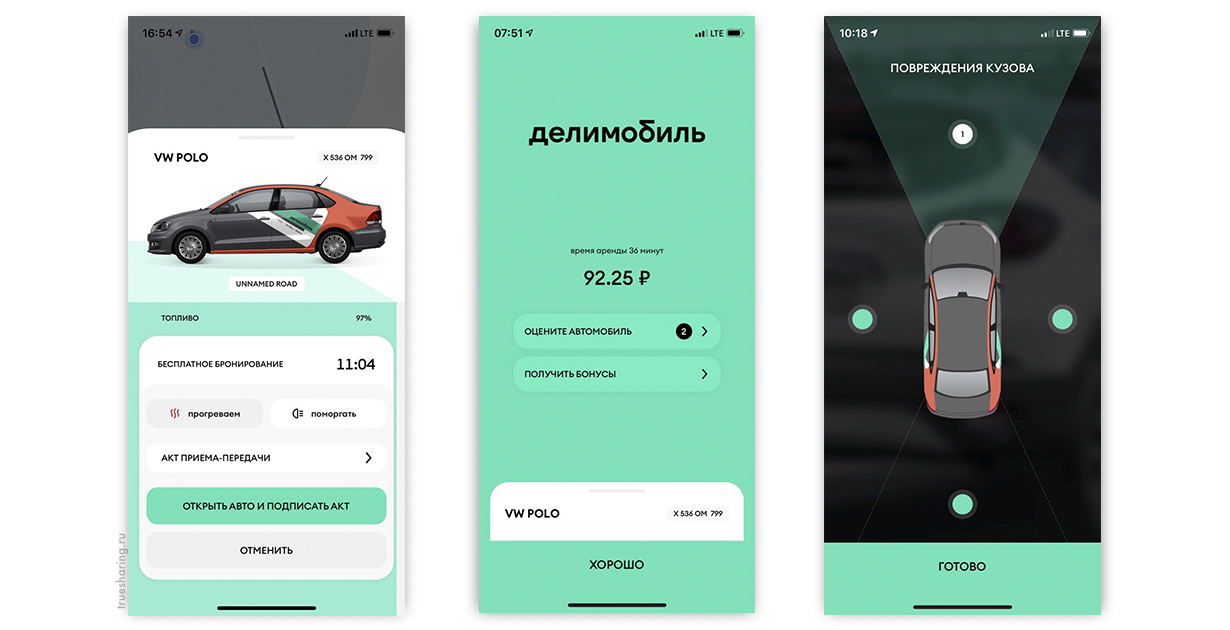


Рисунок 1 – «Интерфейс Делимобиль.»

Преимущества:

* Широкий выбор автомобилей.
* Недорогая поминутная аренда.
* Простой дизайн.
* Карта с автомобилями данного сервиса в реальном времени.

Недостатки:

* Приложение зависает в самый неподходящий момент. Поддержка клиентов не какая.

1.3.2 Яндекс Драйв.

Лучший сервис каршеринга, функционирующий в Москве. Занимает лидирующие позиции по количеству автомобилей в России, второе место в Европе. Автопарк оператора насчитывает 4300 транспортных средств.

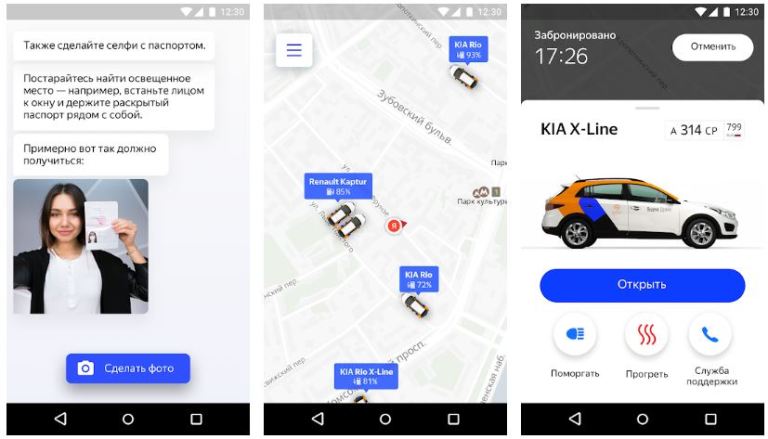


Рисунок 2 – Яндекс Драйв

Преимущества:

* Большой выбор авто.
* Простой и удобный дизайн.
* Карта с автомобилями данного сервиса в реальном времени.

Недостатки:

* Долгий процесс подтверждения документов.

2 ПРОЕКТИРОВАНИЕ

Выполнение поставленных задач подразумевает разработку внутренней части приложения с предусмотрением интерфейсов для подключения графических окон управления разработанными программными модулями. Была разработана архитектура приложения и в дальнейшем это было программно реализовано.

2.1 Архитектура приложения

Архитектура программного обеспечения (англ. software architecture) — совокупность важнейших решений об организации программной системы. Архитектура включает:

* выбор структурных элементов и их интерфейсов, с помощью которых составлена система, а также их поведения в рамках сотрудничества структурных элементов.
* соединение выбранных элементов структуры и поведения во всё более крупные системы.
* архитектурный стиль, который направляет всю организацию — все элементы, их интерфейсы, их сотрудничество и их соединение.

Документирование архитектуры программного обеспечения (ПО) упрощает процесс коммуникации между разработчиками, позволяет зафиксировать принятые проектные решения и предоставить информацию о них эксплуатационному персоналу системы, повторно использовать компоненты и шаблоны проекта в других.

2.1.1 Типы архитектур.

2.1.1.1 Многоуровневая архитектура.

Система делится на уровни, каждый из которых взаимодействует лишь с двумя соседними. Поэтому запросы к БД, которая обычно располагается в самом конце цепочки взаимодействия, проходят последовательно сквозь каждый «слой».

Архитектура не подразумевает какое-то обязательное количество уровней — их может быть три, четыре, пять и больше. Чаще всего используют трехзвенные системы: с уровнем представления (клиентом), уровнем логики и уровнем данных.

Преимущества:

* Каждый уровень этой архитектуры выполняет строго ограниченный набор функций (которые не повторяются от слоя к слою) и не знает о том, как устроены остальные уровни. Поэтому «содержимое» уровней можно изменять без риска глобальных конфликтов между слоями.

Недостатки:

* Низкая скорость работы. Очень много информации начинает бесполезно проходить от слоя к слою, не используя бизнес-логику. Иногда эту проблему называют sinkhole anti-pattern — шаблон проектирования, когда количество бесполезных операций начинает преобладать над полезными.
* Поиск багов в многоуровневых системах также может быть затруднен.

Хорошо подходит для:

* создания новых приложений, которые нужно развернуть по-быстрому. Это своеобразный «шаблон общего назначения».

2.1.1.2 Событийно-ориентированная архитектура.

Разработчик прописывает для программы поведение (реакции) при возникновении каких-либо событий. Событием в системе считается существенное изменение её состояния.

Система, управляемая событиями, обычно содержит два компонента: источники событий (агенты) и их потребители (стоки). Типов событий обычно тоже два: инициализирующее событие и событие, на которое реагируют потребители.

Преимущества:

* Так как приложения состоят из большого количества асинхронных модулей (у которых нет информации о реализации друг друга), их легко масштабировать. Такие системы собираются как конструктор — прописывать зависимости не нужно, достаточно реализовать новый модуль.
* Дополнительно асинхронная модель позволяет добиться высокой производительности приложений.

Недостатки:

* Асинхронная натура таких приложений усложняет отладку. Одно событие может запускать сразу несколько цепочек действий. Если таких цепочек будет много, то понять, что именно вызвало сбой, может быть затруднительно. Для решения проблемы приходится прорабатывать сложные условия обработки ошибок.
* Проблема с журналированием — логи трудно структурировать.

Подходит для:

* Создания асинхронных систем. Это очевидно, поскольку сама архитектура состоит из большого количества асинхронных модулей.
* Можно применить для создания UI. Веб-страница выступает в роли контейнера, в котором каждый её компонент изолирован и реагирует на определённые действия пользователя.
* Для организации обмена сообщениями между различными информационными системами.

2.1.1.3 Микроядерная архитектура.

Этот тип архитектуры состоит из двух компонентов: ядра системы и плагинов. Плагины отвечают за бизнес-логику, а ядро руководит их загрузкой и выгрузкой.

Преимущества:

* Легко портировать приложение из одной среды в другую, поскольку модифицировать нужно только микроядро.
* Разделение высокоуровневых политик и низкоуровневых механизмов упрощает поддержку системы и обеспечивает её расширяемость.

Недостатки:

* Производительность приложения снижается, если подключать слишком много модулей. Однако бывает проблематично найти баланс между количеством плагинов и числом задач микроядра (обычно оно содержит лишь часто используемой код).
* Также сложно определить заранее (до начала разработки приложения) оптимальную степень дробления кода микроядра. А поменять подход позднее практически невозможно.

Хорошо подходит для:

* Создания расширяемых приложений, которыми пользуется большое количество людей. Например, ОС для iPhone имеет «микроядерные» корни — её разработчики черпали вдохновение в Mach (это один из самых первых примеров микроядра).
* Создания приложений с четким разделением базовых методов и высокоуровневых правил.
* Разработки систем с динамически меняющимся набором правил, которые приходится часто обновлять.

2.1.1.4 Микросервисы.

Похожи на архитектуру, управляемую событиями, и микроядро. Но используются тогда, когда отдельные задачи приложения можно легко разделить на небольшие функции — независимые сервисы. Эти сервисы могут быть написаны на разных языках программирования, поскольку общаются друг с другом при помощи REST API (например, с использованием JSON или Thrift).

В каких пропорциях делить код, решает разработчик, но Сэм Ньюмен (Sam Newman), автор книги «Создание микросервисов», рекомендует выделять на микросервис столько строк кода, сколько команда сможет воспроизвести за две недели. По его словам, это позволит избежать излишнего «раздувания» архитектуры.

Чаще всего микросервисы запускаются в так называемых контейнерах. Эти контейнеры доступны по сети другим микросервисами и приложениям, а управляет ими всеми система оркестровки: примерами могут быть Kubernetes, Docker Swarm и др.

Преимущества:

* Микросервисная архитектура упрощает масштабирование приложений. Чтобы внедрить новую функцию достаточно написать новый сервис. Если функция стала не нужна, микросервис можно отключить.
* Каждый микросервис — это отдельный проект, потому работу над ними легко распределить между командами разработчиков.

Недостатки:

* Сложно искать ошибки. В отличие от монолитных систем (когда все функции находятся в одном ядре), бывает сложно определить, почему «упал» запрос. За деталями приходится идти в логи «виновного» процесса (если их несколько, то проблема усугубляется).
* Появляются дополнительные накладные расходы на передачу сообщений между микросервисами. По нашим оценкам, рост сетевых издержек может достигать 25%.
* Необходимость мириться с концепцией eventual consistency (согласованность в конечном счёте). У микросервисов есть собственные хранилища данных, к которым обращаются другие микросервисы. Информация об изменении этих данных распространяется по системе не мгновенно. Потому возникают ситуации, когда у некоторых микросервисов (пусть и на крайне короткий промежуток времени) оказываются устаревшие данные.

Где использовать:

* В крупных проектах с высокой нагрузкой. Например, микросервисы используются стриминговыми платформами. Системы доставки контента и иные вспомогательные сервисы можно масштабировать независимо друг от друга, подстраиваясь под изменения нагрузки.
* В системах, использующих «разномастные» ресурсы. Если одной части приложения нужно больше процессорного времени, а второй — памяти, то имеет смысл разделить их на микросервисы. После чего их можно захостить на разных машинах — с мощным CPU или большим объемом памяти соответственно.
* Когда нужна безопасность. Так как микросервисы изолированы и общаются по API, можно гарантировать, что передаваться будет только та информация, которая нужна тому или иному сервису. Это важно при работе с паролями или данными платёжных карт.

2.1.2 Выбранная архитектура.

Выбор пал на событийно-ориентированную архитектуру. Данный выбор поможет решить поставленные задачи:

Система управления доступом на основе ролей:

Для снижения стоимости разработки, стоимости сопровождения и сжатия сроков сдачи программного продукта, было решено ввести ввести такое понятие как, управление доступом на основе ролей (англ. Role Based Access Control, RBAC) — развитие политики избирательного управления доступом, при этом права доступа субъектов системы на объекты группируются с учётом специфики их применения, образуя роли.

Выделены следующие роли:

* Пользователь.
* Администратор.

Администрирование:

* Возможность вносить и удалять автомобили.

Подача заявки на аренду конкретного автомобиля.

Система уведомлений об изменении статуса заявок.

Предусмотреть интерфейсы для дальнейшего подключения графических окон управления представленными программными модулями.

Предусмотреть интерфейсы для дальнейшего подключения интеграции FTP сервера для синхронизации данных на разных устройствах.

2.1.3 Паттерны проектирования.

Паттерн проектирования — это 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.

Паттерн — это высокоуровневое описание решения, реализация которого может отличаться в двух разных программах.

Если привести аналогии, то алгоритм — это кулинарный рецепт с чёткими шагами, а паттерн — инженерный чертёж, на котором нарисовано решение, но не конкретные шаги его реализации.

Описания паттернов обычно очень формальны и чаще всего состоят из таких пунктов:

* проблема, которую решает паттерн.
* мотивации к решению проблемы способом, который предлагает паттерн.
* структуры классов, составляющих решение.
* примера на одном из языков программирования.
* особенностей реализации в различных контекстах.
* связей с другими паттернами.

Такой формализм в описании позволил создать обширный каталог паттернов, проверив каждый из них на состоятельность.

2.1.3.1 Порождающие паттерны проектирования.

Порождающие шаблоны — шаблоны проектирования, которые имеют дело с процессом создания объектов. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов.

2.1.3.1.1 Singleton (Одиночка).

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

Одиночка решает сразу две проблемы, нарушая принцип единственной ответственности класс.

Гарантирует наличие единственного экземпляра класса. Чаще всего это полезно для доступа к какому-то общему ресурсу, например, базе данных.

Представьте, что создали объект, а через некоторое время пробуете создать ещё один. В этом случае хотелось бы получить старый объект, вместо создания нового.

Такое поведение невозможно реализовать с помощью обычного конструктора, так как конструктор класса всегда возвращает новый объект.

Предоставляет глобальную точку доступа. Это не просто глобальная переменная, через которую можно достучаться к определённому объекту. Глобальные переменные не защищены от записи, поэтому любой код может подменять их значения без ведома разработчика.

Но есть и другой нюанс. Неплохо бы хранить в одном месте и код, который решает проблему №1, а также иметь к нему простой и доступный интерфейс.

Интересно, что сейчас паттерн стал настолько известен, что теперь люди называют «одиночками» даже те классы, которые решают лишь одну из проблем, перечисленных выше.

Все реализации одиночки сводятся к тому, чтобы скрыть конструктор по умолчанию и создать публичный статический метод, который и будет контролировать жизненный цикл объекта-одиночки.

Если есть доступ к классу одиночки, значит, будет доступ и к этому статическому методу. Из какой точки кода бы его ни вызвали, он всегда будет отдавать один и тот же объект.

Применимость:

* Когда в программе должен быть единственный экземпляр какого-то класса, доступный всем клиентам (например, общий доступ к базе данных из разных частей программы).
* Одиночка скрывает от клиентов все способы создания нового объекта, кроме специального метода. Этот метод либо создаёт объект, либо отдаёт существующий объект, если он уже был создан.
* Когда хочется иметь больше контроля над глобальными переменными.
* В отличие от глобальных переменных, Одиночка гарантирует, что никакой другой код не заменит созданный экземпляр класса, поэтому можно всегда быть увереным в наличии лишь одного объекта-одиночки.
* Тем не менее, в любой момент можно расширить это ограничение и позволить любое количество объектов-одиночек, поменяв код в одном месте (метод getInstance).

Шаги реализации:

* Добавление в класс приватного статического поля которое будет содержать одиночный объект.
* Объявление статического создающего метода, который будет использоваться для получения одиночки.
* Добавление «ленивой инициализации» (создание объекта при первом вызове метода) в создающий метод одиночки.
* Конструктор класса приватный.
* В клиентском коде заменить вызовы конструктора одиночка вызовами его создающего метода.

Преимущества:

* Гарантирует наличие единственного экземпляра класса.
* Предоставляет к нему глобальную точку доступа.
* Реализует отложенную инициализацию объекта-одиночки.

Недостатки:

* Нарушает принцип единственной ответственности класса.
* Маскирует плохой дизайн.
* Проблемы мультипоточности.
* Требует постоянного создания Mock-объектов при юнит-тестировании.

Отношения с другими паттернами:

* Фасад можно сделать Одиночкой, так как обычно нужен только один объект-фасад.
* Паттерн Легковес может напоминать Одиночку, если для конкретной задачи получилось свести количество объектов к одному. Но, между паттернами есть два кардинальных отличия:
  + В отличие от Одиночки, можно иметь множество объектов-легковесов.
  + Объекты-легковесы должны быть неизменяемыми, тогда как объект-одиночка допускает изменение своего состояния.
* Абстрактная фабрика, Строитель и Прототип могут быть реализованы при помощи Одиночки.

2.2 Хранение данных.

Все данные касаемо автомобилей хранятся в таблице cars в базе данных MySQL размещенной на удаленном сервере. База размещенная на удаленном сервере позволит обращаться к ней с разных устройств, что способствует синхронизации данных между разными устройствами.

2.2.1 Структура хранения данных в базе данных.

2.2.2.1 Таблица cars.

* Поле id: первичный ключ с автоинкрементом, числовой тип. Автогенерацию первичного ключа часто называют автоинкрементом.
* Поле model: название марки/модели автомобиля, символьное поле с максимальной длиной в 45 символов.
* Поле modelYear: год производства автомобиля, числовой тип.
* Поле image: относительный путь к изображению на FTP сервере, символьное поле с максимальной длиной в 255 символов.
* Поле specifications: технические характеристики автомобиля такие как: объем двигателя, мощность двигателя, расхож топлива, тип привода, количество пассажиров, коробка передач, автозапуск авто, в нашем парке с (дата), цвет; символьное поле с максимальной длиной в 255 символов.
* Поле cost: стоимость аренды в сутки, числовой тип.

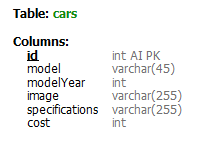


Рисунок 3 – «Структура таблицы cars в базе данных.»

2.2.2 Структура хранения файлов на FTP сервере.

На удаленном FTP сервере в корневом каталоге создан каталог image.  
В данный каталог будут загружаться изображения автомобиля при его добавлении в базу данных. При удалении автомобиля из базы данных, изображение привязанное к этому автомобилю будет удалено с сервера. А так же эти изображения будут загружаться на компьютер клиента в каталог imageCars.

Файл состояния доступа FTP сервера. Данный подход обеспечивает одиночную, по файловую загрузку файлов, во время загрузки или отправки файлов, сервер принимает состояние блокировки. Это позволяет избежать перегрузки соединения с сервером, из-за чего могут возникнуть проблемы отправки и загрузки файлов (смотреть рисунок 4). Есть всего 2 состояния:

* locked – сервер заблокирован, выполнение других операций обращения к серверу невозможно.
* unlocked – сервер доступен.

Рисунок 4 - «Структура хранения файлов на FTP сервере.»



2.2.3 Структура локального хранения изображений автомобилей.

При запуске приложения будет создан каталог imageCars, располагающийся в корневом каталоге в котором установленно приложение, при условии что данный каталог отсутствует. После авторизации пользователя изображения с удаленного FTP сервера будут скачаны и помещены в каталог imageCars, при условии что этих изображений не было в данном каталоге (смотреть рисунок 5).

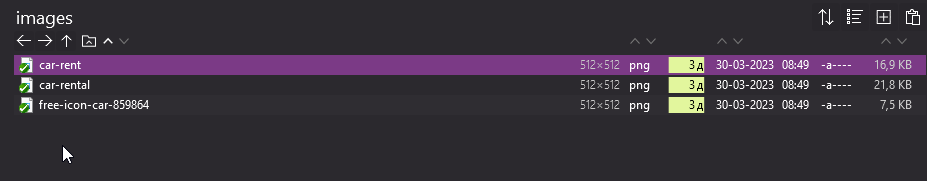


Рисунок 5 - «Структура локального хранения изображений автомобилей.»

2.2 Прототип интерфейса.

Прототипирование — это процесс в работе дизайнера, когда создается «иллюзию» рабочего интерфейса на базе нарисованных макетов или вайрфреймов в короткие временные сроки.

Были разработаны следующие прототипы интерфейса:

* Страница авторизации (смотреть Приложение 1. Рисунок 6).
* Главная страница (интерфейс пользователя) (смотреть Приложение 2. Рисунок 7).
* Страница терминала. Активация роли администратора (интерфейс администратора) (смотреть Приложение 3. Рисунок 8).
* Главная страница (интерфейс администратора) (смотреть Приложение 4. Рисунок 9).
* Страница добавления автомобиля (интерфейс администратора) (смотреть Приложение 5. Рисунок 10).
* Форма отправки заявки на аренду (смотреть Приложение 6. Рисунок 11)

3 ПРАКТИЧЕСКАЯ ЧАСТЬ

Для начала следует ввести следующие понятия:

* Интерфейс командной строки CLI (Command-line interface) — способ взаимодействия между человеком и компьютером путём отправки компьютеру команд, представляющих собой последовательность символов.
* Система контроля версий VCS (Version Control System) — это система, записывающая изменения в файл или набор файлов в течение времени и позволяющая вернуться позже к определённой версии.
* Графический интерфейс пользователя, графический пользовательский интерфейс GUI (Graphical User Interface) — система средств для взаимодействия пользователя с компьютером, основанная на представлении всех доступных пользователю системных объектов и функций в виде графических компонентов экрана.

При разработке обязательно должна использоваться VCS. Выбран Git.

3.1 Версионирование.

Версионирование — разработка и управление несколькими выпусками продукта, которые имеют тот же общий функционал, но усовершенствованы, модернизированы либо индивидуализированы. Коротко, версия говорит об изменении продукта.

Все версии условно обозначаются: V0.0.0

Начальная версия: V0.0.1

3.2 Программная реализация.

main.py

Импорт библиотек и модулей (смотреть рисунок 12).

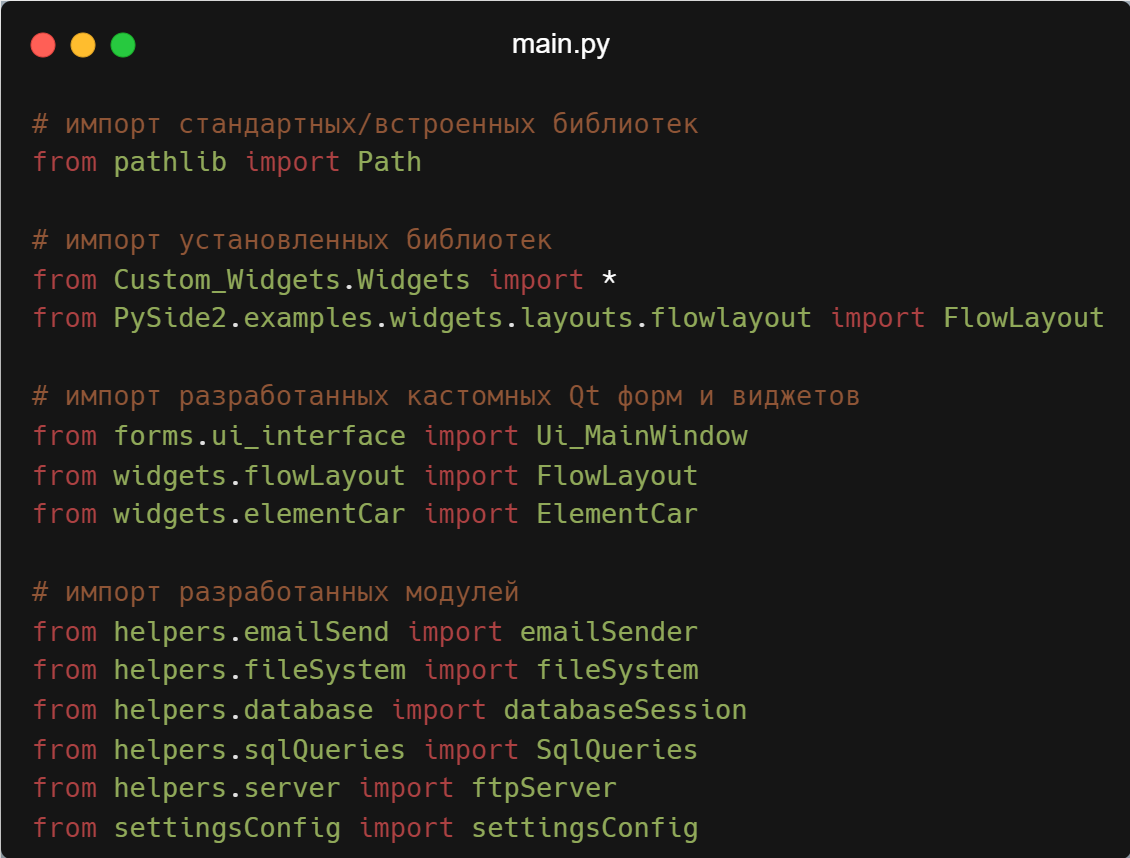


Рисунок 12 – «main.py импорт библиотек и модулей.»

Определение необходимых констант. Создать ссылку на путь к директории imageCars. Получить токен для доступа к администрированию (смотреть рисунок 13).

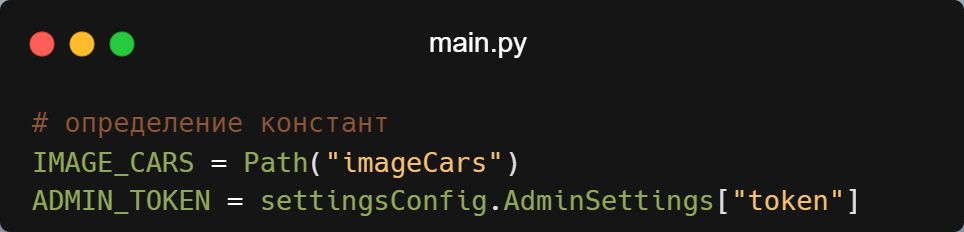


Рисунок 13 – «main.py инициализация необходимых констант.»

Конструктор главного окна. Конструктор — уникальный метод класса, который называется \_\_init\_\_ . Первый параметр конструктора во всех случаях self (ключевое слово, которое ссылается на сам класс). Конструктор нужен для создания объекта. Конструктор передает значения аргументов свойствам создаваемого объекта. (смотреть рисунок 14).



Рисунок 14 – «main.py конструктор 1.»

Скрытие этикеток, которые будут выводить сообщения об ошибках (смотреть рисунок 15).

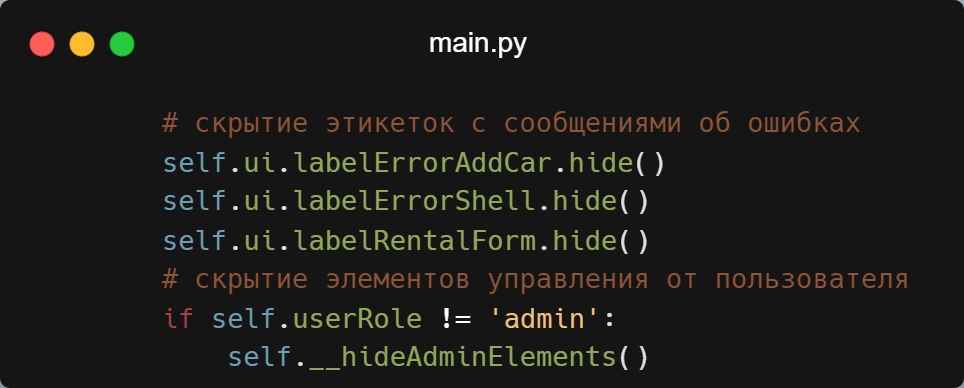


Рисунок 15 – «main.py конструктор 2.»

Инициализация кастомного Layout для более эфективного размещения виджетов. Подключение методов к кнопкам, к событию нажатия кнопки (смотреть рисунок 16).

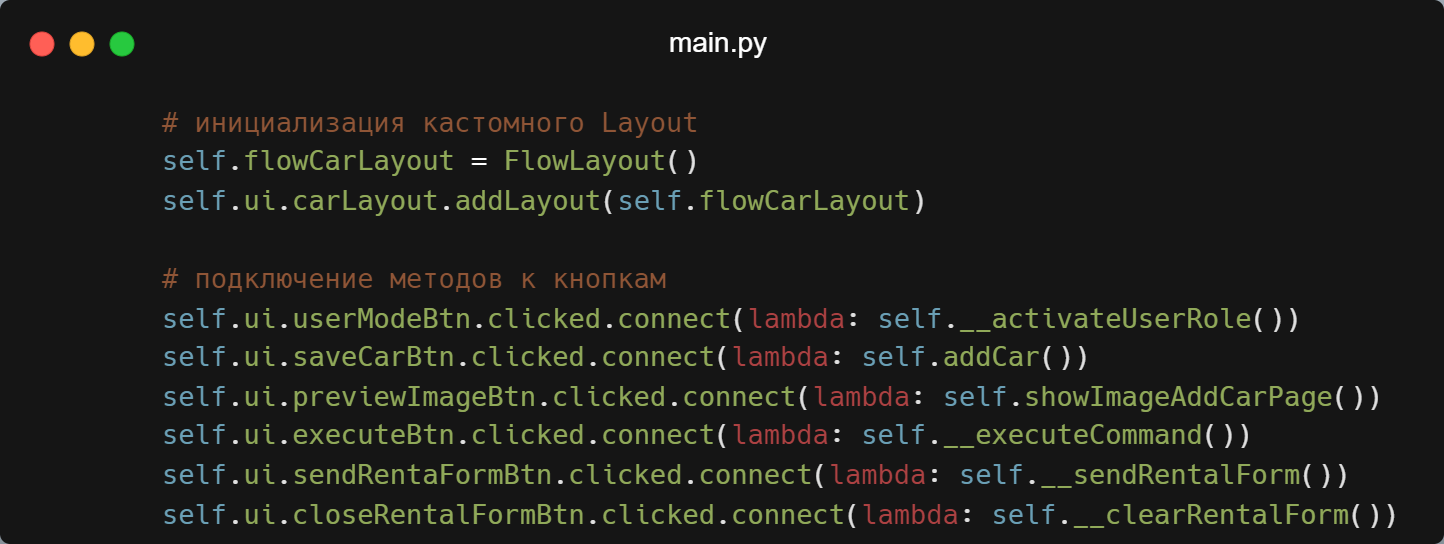


Рисунок 16 – «main.py конструктор 3.»

Создание необходимого каталога. Скачивания всех изображений с FTP сервера. Получение данных о каждом автомобиле из базы данных. Вставка данных в форму добавления автомобиля (смотреть рисунок 17).



Рисунок 17 – «main.py метод \_\_loadData.»

Методы для активации роли (смотреть рисунок 18).



Рисунок 18 – «main.py методы для активации роли.»

Методы отображения этикеток с текстом об ошибках (смотреть рисунок 19).



Рисунок 19 – «main.py методы отображения этикеток с текстом об ошибках.»

Реализация выполнения команд в терминале для администратора, событие при нажатии кнопки Выполнить команду на странице терминала (смотреть рисунок 20).

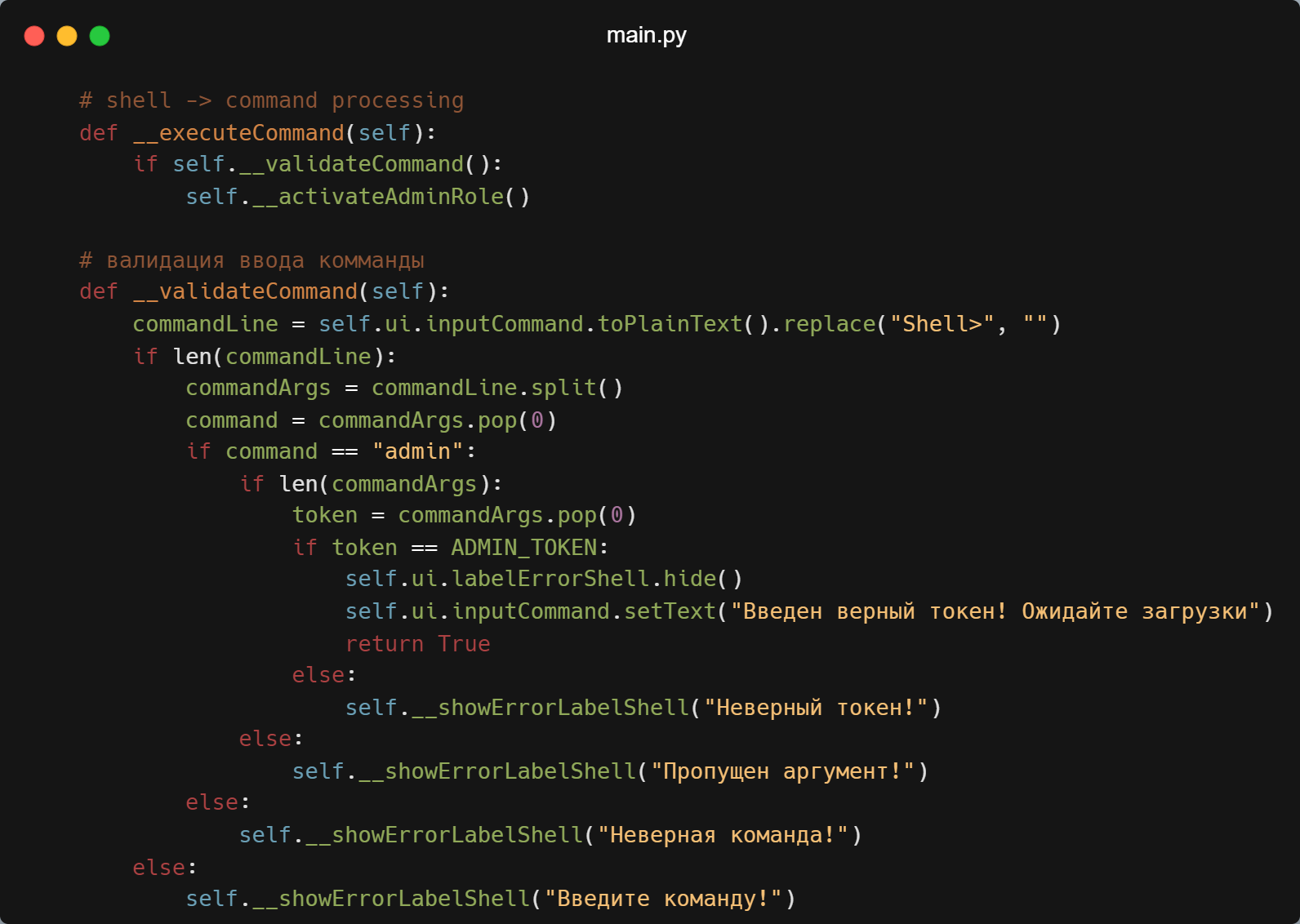


Рисунок 20 – «main.py ввод команд в терминале для администратора.»

Событие, вызываемое при нажатии кнопки Сохранить на странице добавления автомобиля (смотреть рисунок 21).

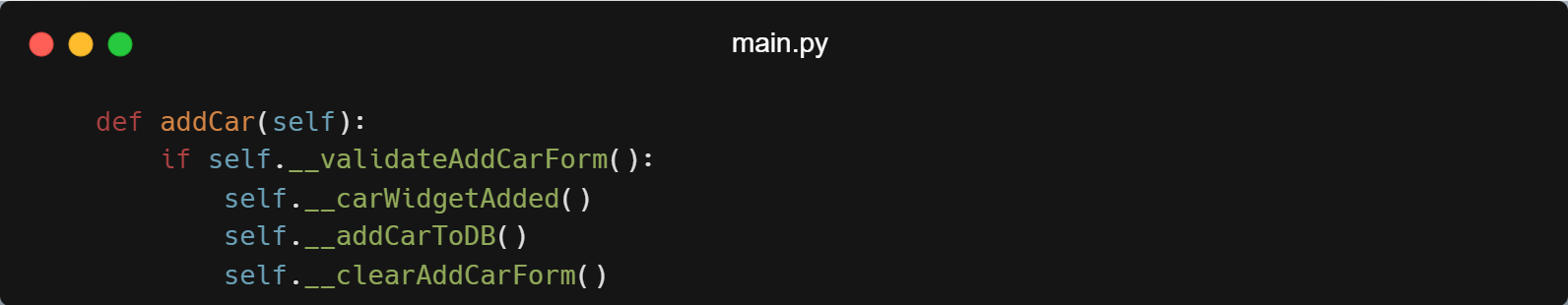


Рисунок 21 – «main.py событие добавления автомобиля.»

Валидация введенных данных в форме добавления автомобиля (смотреть рисунок 22).



Рисунок 22 – «main.py валидация введенных данных в форме добавления автомобиля .»

Создание виджета/карточки автомобиля (смотреть рисунок 23).



Рисунок 23 – «main.py создание виджета/карточки автомобиля .»

Добавление автомобиля в базу данных (смотреть рисунок 24).

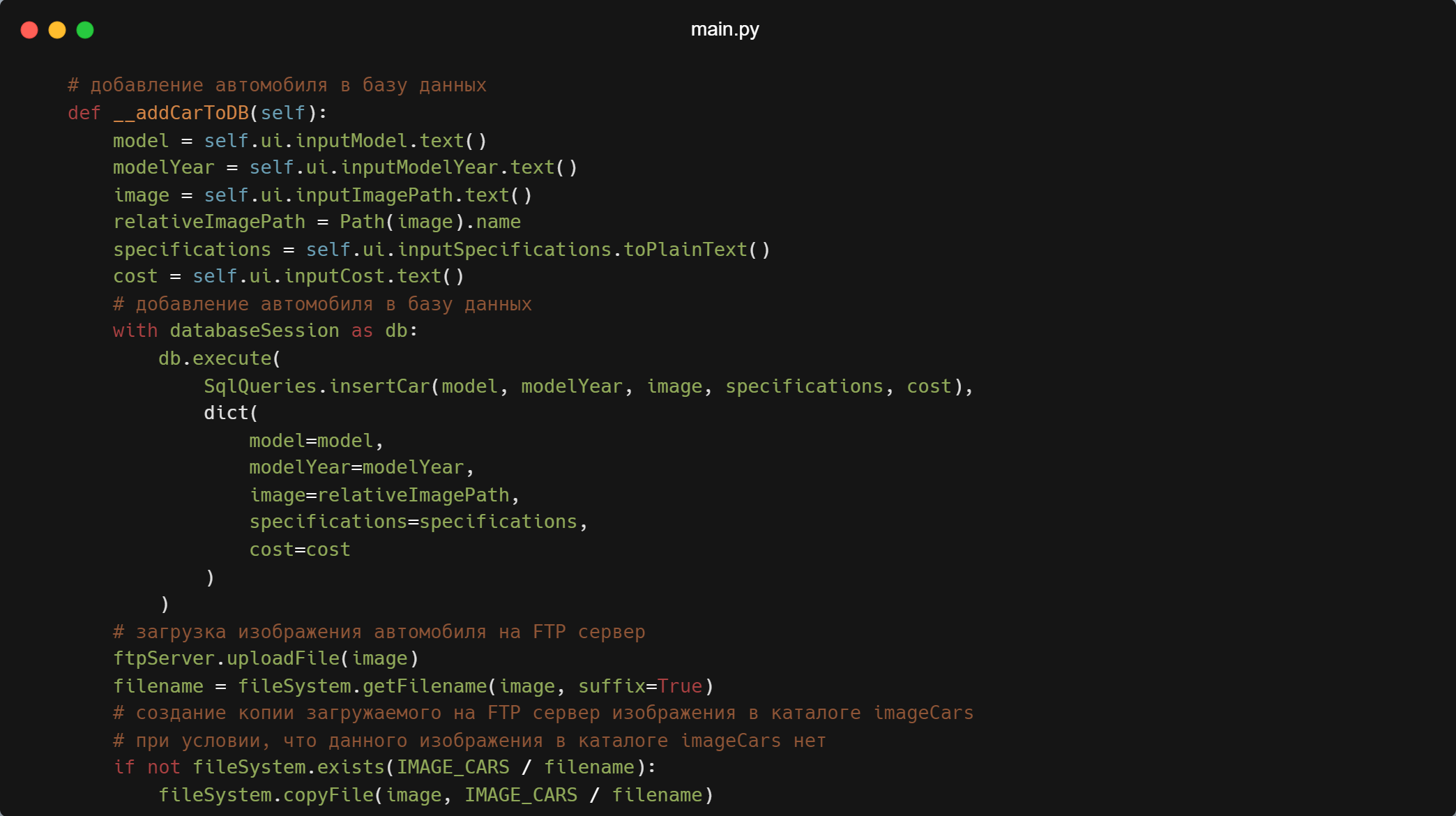


Рисунок 24 – «main.py добавление автомобиля в базу данных .»

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Кратко о типах архитектур программного обеспечения, и какую из них выбрали мы для IaaS-провайдера. - [Электронный ресурс]: страница Habr.com. – Режим доступа: <https://habr.com/ru/company/1cloud/blog/424911/>

Сэм Ньюмен (Sam Newman), «Создание микросервисов» (Building Microservices), 2015. - [Книга]

Майкл Т. Фишер, Мартин Л. Абботт, «Art of Scalability, The: Scalable Web Architecture, Processes, and Organizations for the Modern Enterprise», 2009. - [Книга]

Шаблон проектирования. - [Электронный ресурс]: страница Wikipedia. - Режим доступа: https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD\_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F

Паттерны проектирования. - [Электронный ресурс]: страница refactoring.guru. – Режим доступа: https://refactoring.guru/ru/design-patterns

Паттерн проектирования Singleton (Одиночка). - [Электронный ресурс]: страница refactoring.guru. – Режим доступа: https://refactoring.guru/ru/design-patterns/singleton

UML-диаграммы классов. - [Электронный ресурс]: страница prog-cpp.ru. – Режим доступа: https://prog-cpp.ru/uml-classes/

ПРИЛОЖЕНИЕ 1

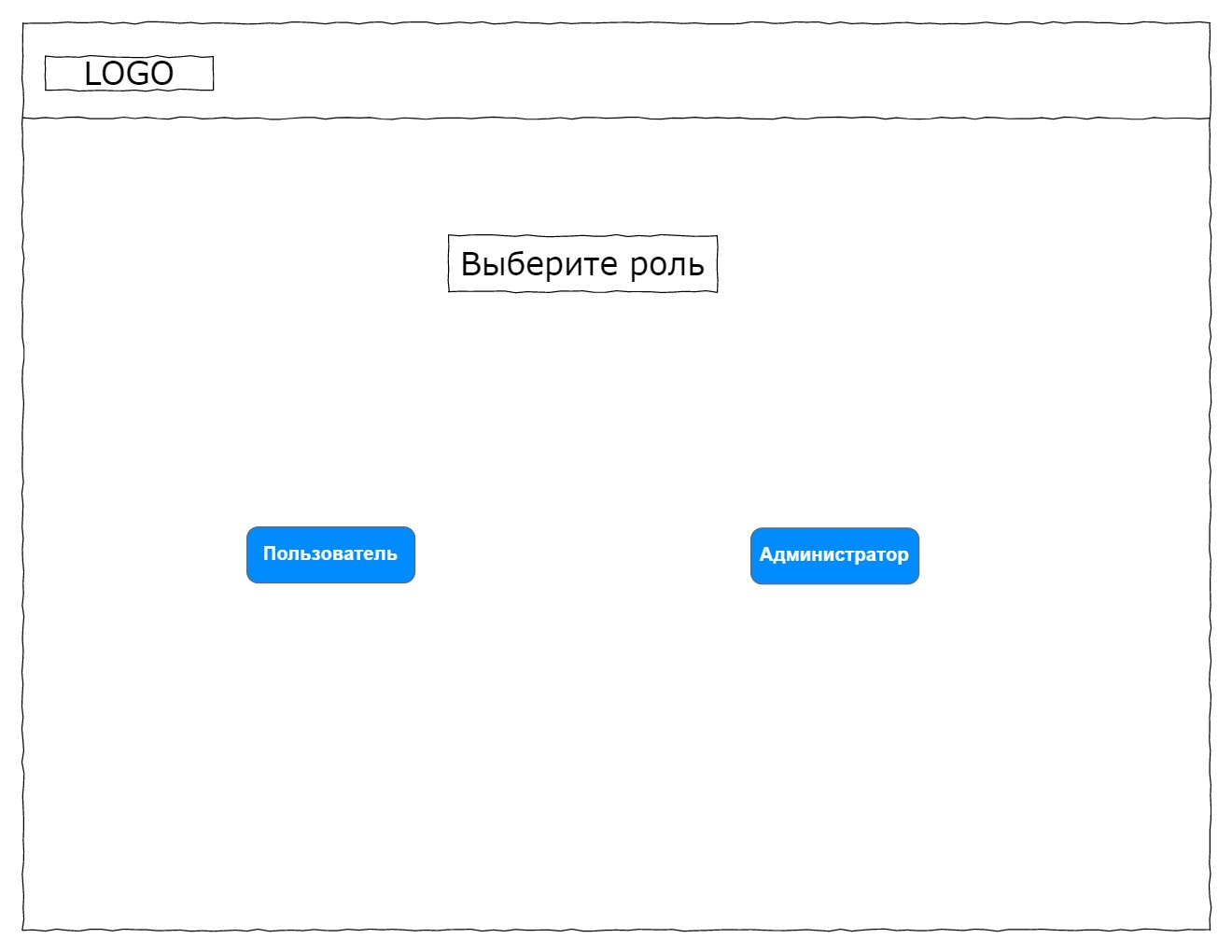


Рисунок 6 «Прототип интерфейса. Страница авторизации.»

ПРИЛОЖЕНИЕ 2

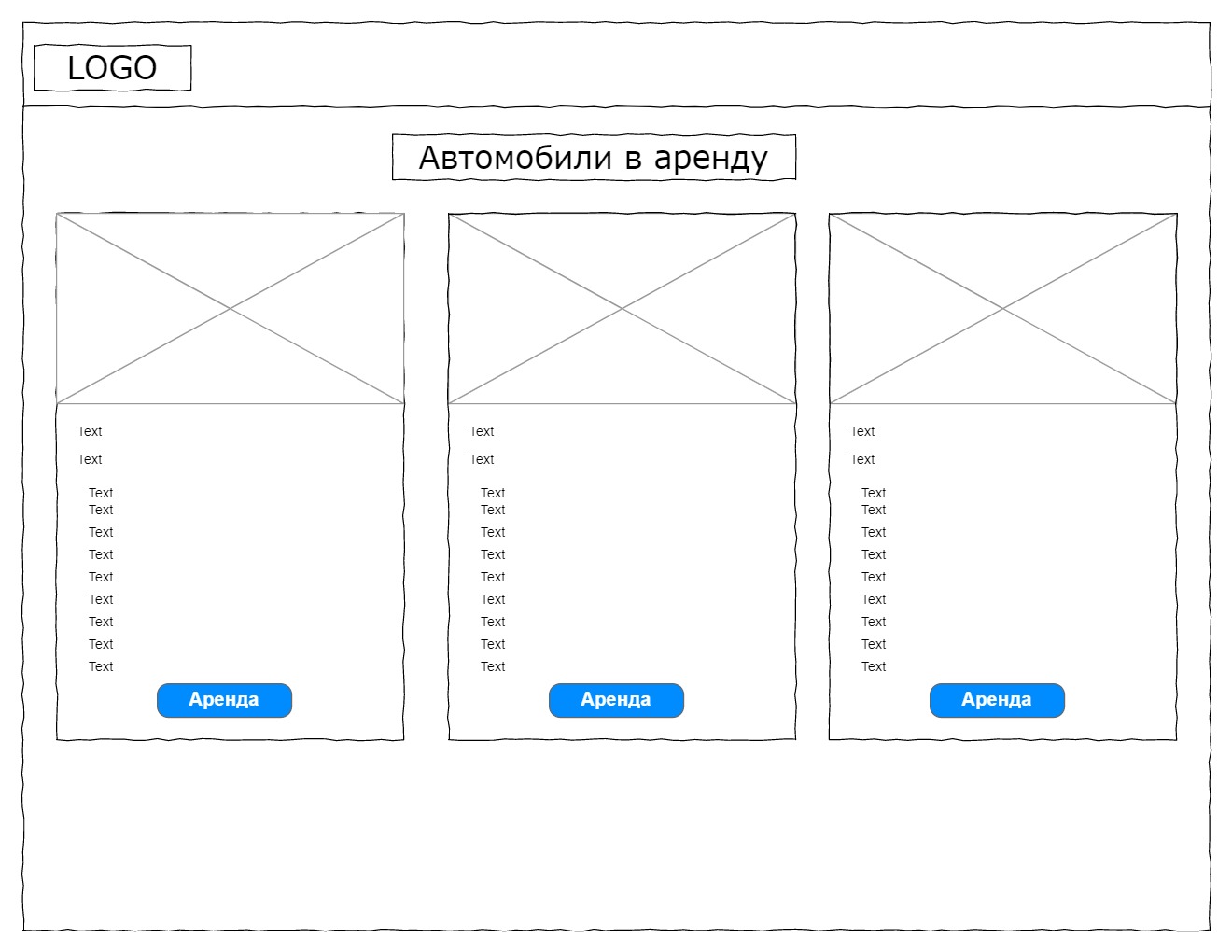


Рисунок 7 - «Прототип интерфейса. Главная страница (интерфейс пользователя).»

ПРИЛОЖЕНИЕ 3



Рисунок 8 - «Прототип интерфейса. Страница терминала. Активация роли администратора (интерфейс администратора).»

ПРИЛОЖЕНИЕ 4

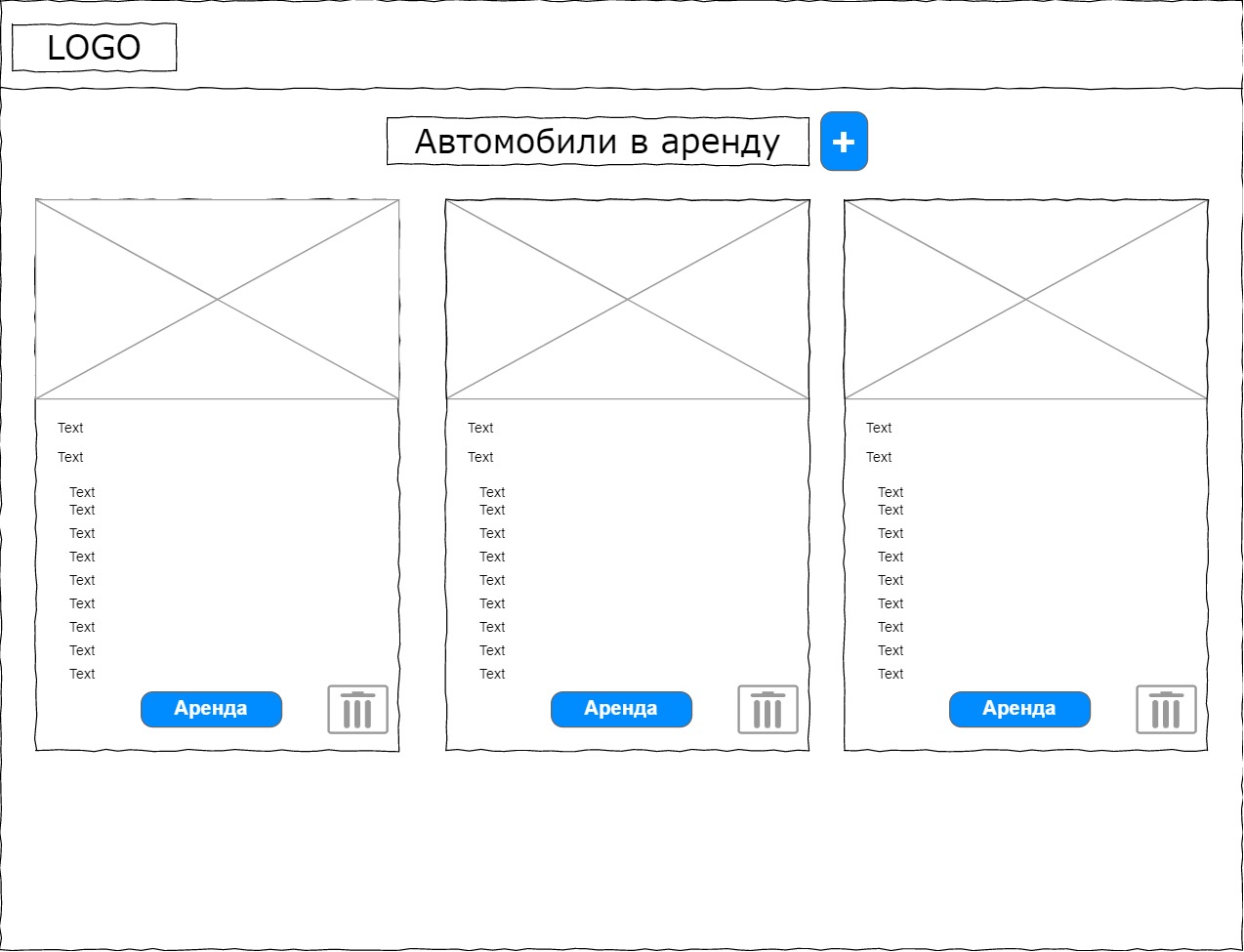


Рисунок 9 - «Прототип интерфейса. Главная страница (интерфейс администратора).»

ПРИЛОЖЕНИЕ 5

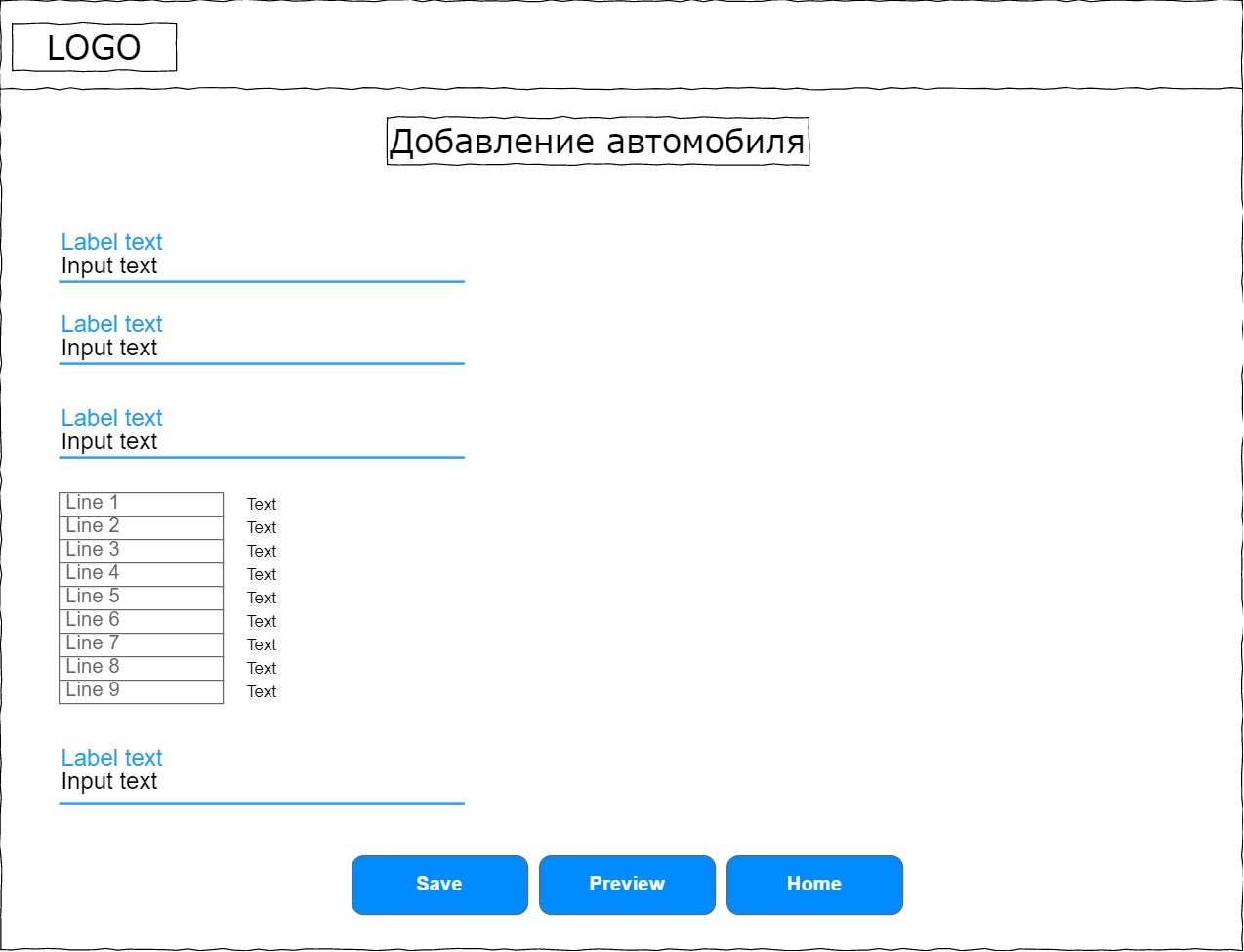


Рисунок 10 - «Прототип интерфейса. Страница добавления автомобиля (интерфейс администратора).»

ПРИЛОЖЕНИЕ 6

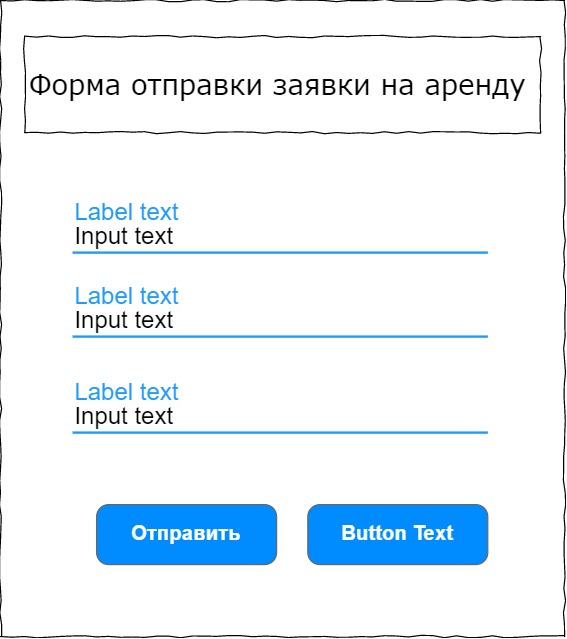


Рисунок 11 - «Прототип интерфейса. Форма отправки заявки на аренду.»

ПРИЛОЖЕНИЕ 7

Инструкция по инсталляции.

* Перейти в Git репозиторий, в раздел [Releases](https://github.com/LunexCoding/CarRental/releases) (смотреть рисунок 12).
* Загрузить последнюю версию программного продукта.
* Запустить инсталлятор setupCarRental.exe. Может потребоваться разрешить вносить приложению от неизвестного издателя изменения на вашем устройстве (смотреть рисунок 13).

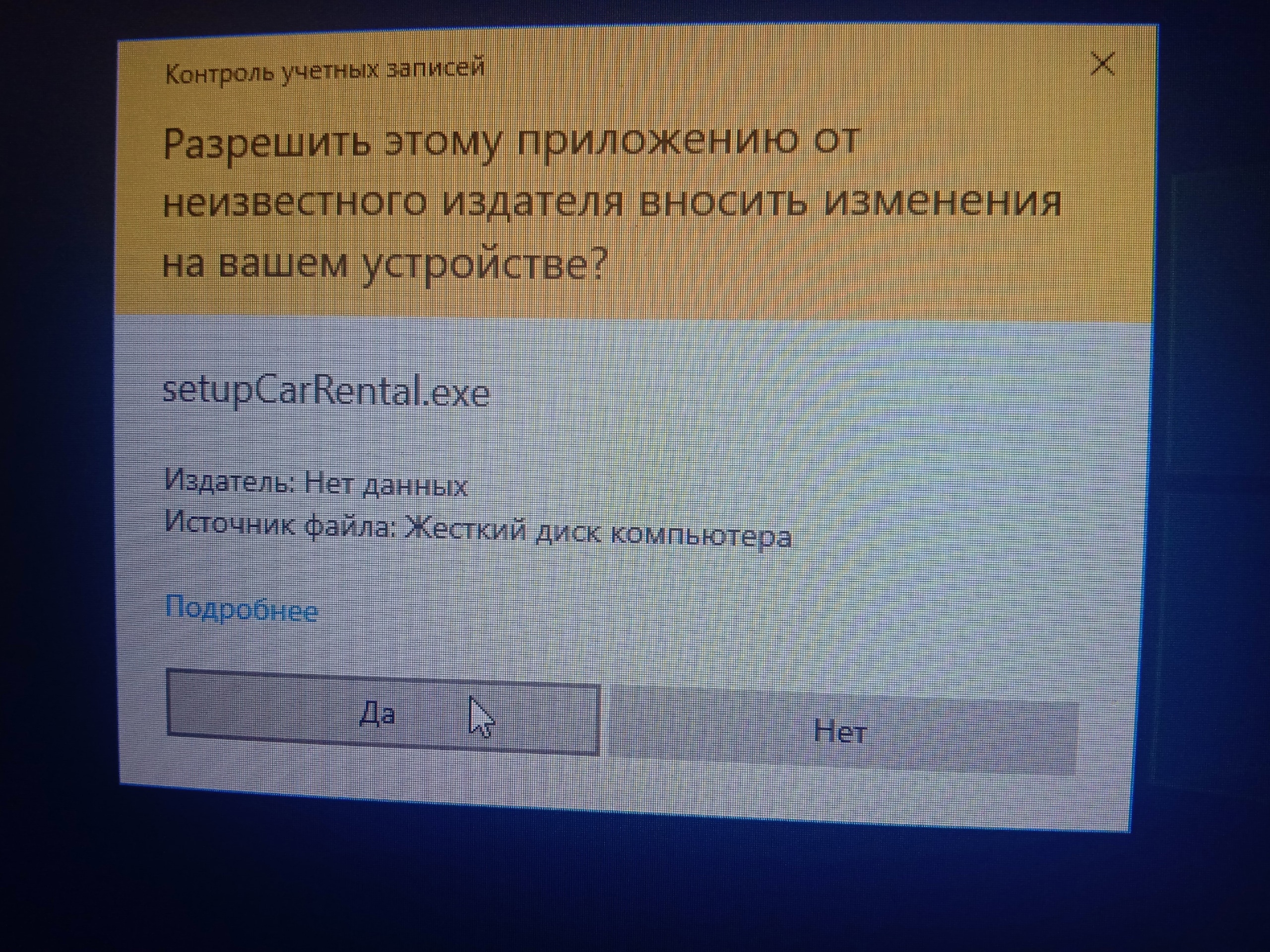


Рисунок 13 - «Запуск инсталлятора. Разрешить вносить приложению от неизвестного издателя изменения на вашем устройстве.»

* Завершить инсталляцию (смотреть рисунки 14, 15, 16, 17, 18).

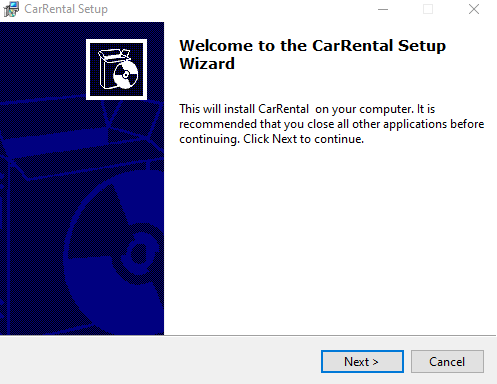


Рисунок 14 - «Инсталляция приложения 1.»

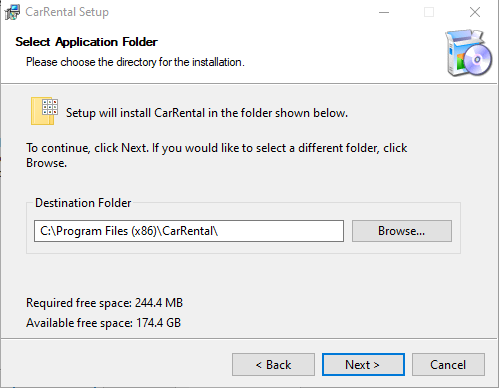


Рисунок 15 - «Инсталляция приложения 2.»

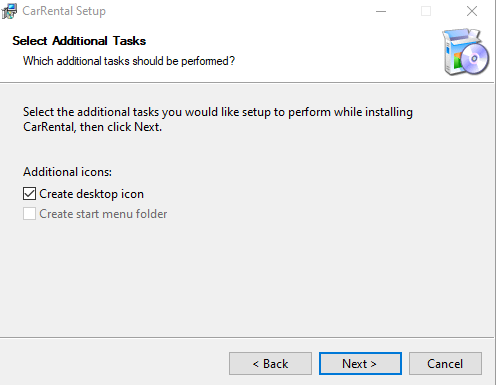


Рисунок 16 - «Инсталляция приложения 3.»

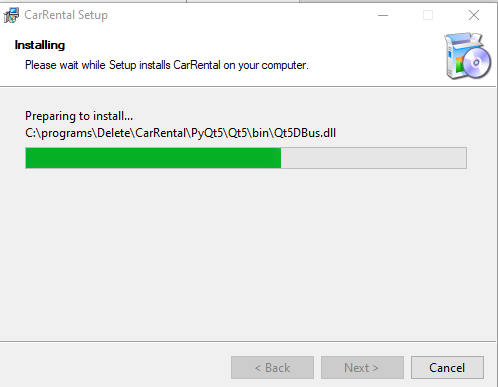


Рисунок 17 - «Инсталляция приложения 4.»

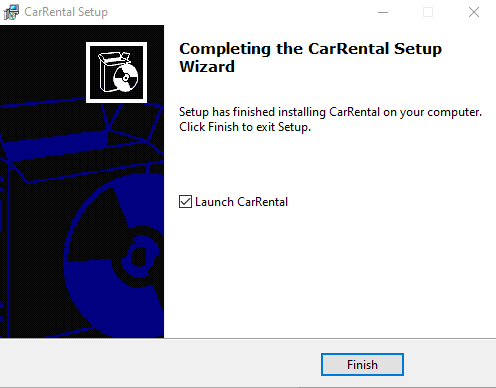


Рисунок 18 - «Инсталляция приложения 5.»

ПРИЛОЖЕНИЕ 8

Инструкция по эксплуатации для пользователя.

Для запуска приложения требуется стабильное интернет соединение. После открытия исполнительного файла, откроется страница выбора роли (смотреть рисунок 19). Выбираете Пользователь.

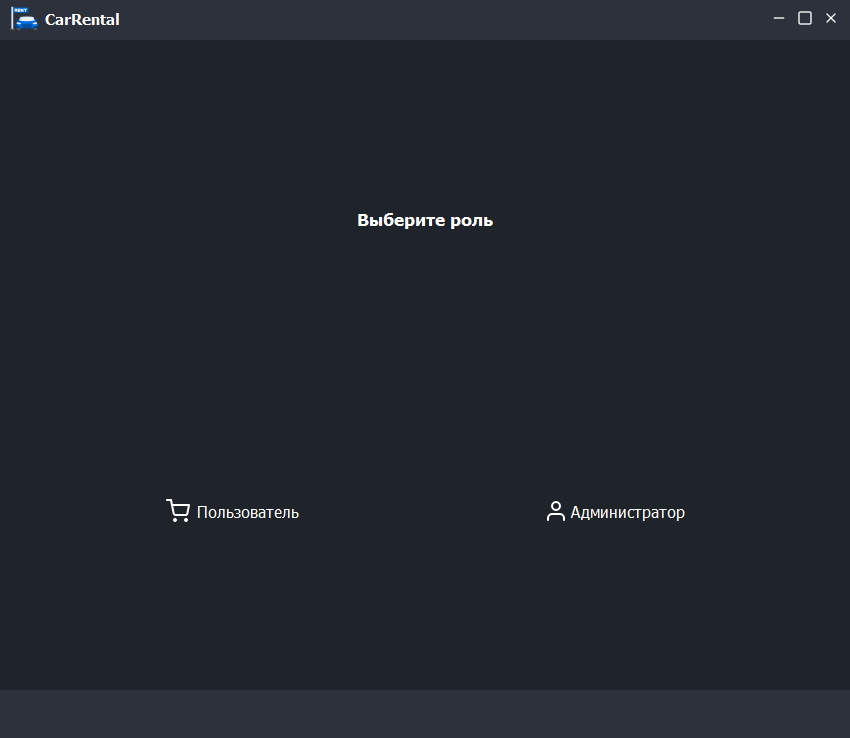


Рисунок 19 - «Страница выбора роли.»

Откроется главная страница приложения. На рисунке 20 отображена ситуация, когда нет актуальных предложений об аренде автомобилей. (смотреть Рисунок 20).

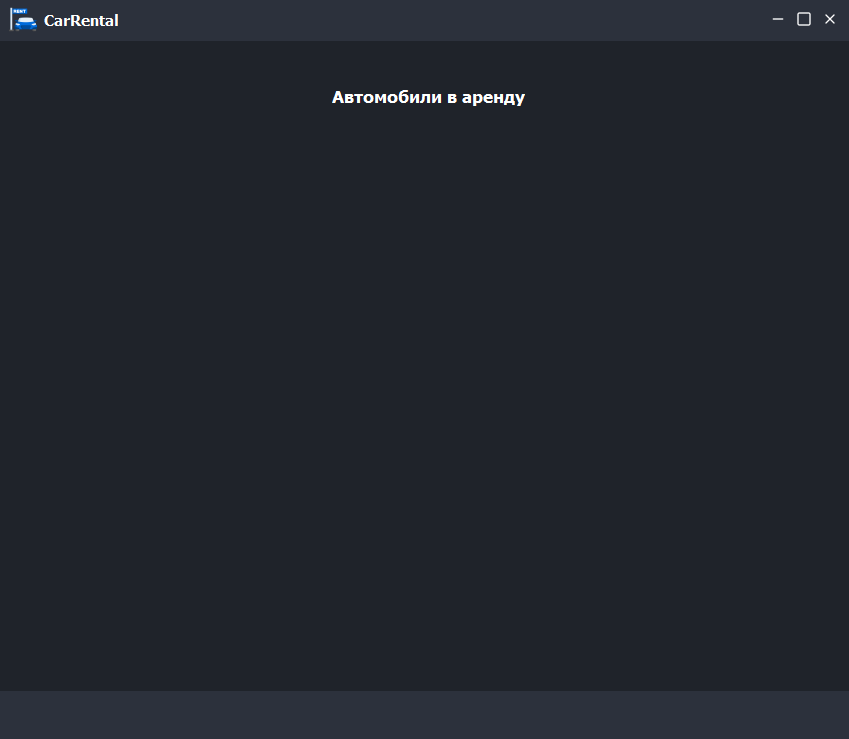


Рисунок 20 - «Главная страница пользователя 1.»

На рисунке 21 отображена ситуации, когда есть актуальные предложения об аренде автомобилей. Можно ознакомиться с автомобилем, а именно с его техническими характеристики, а так же с ценой аренды (смотреть рисунок 21).

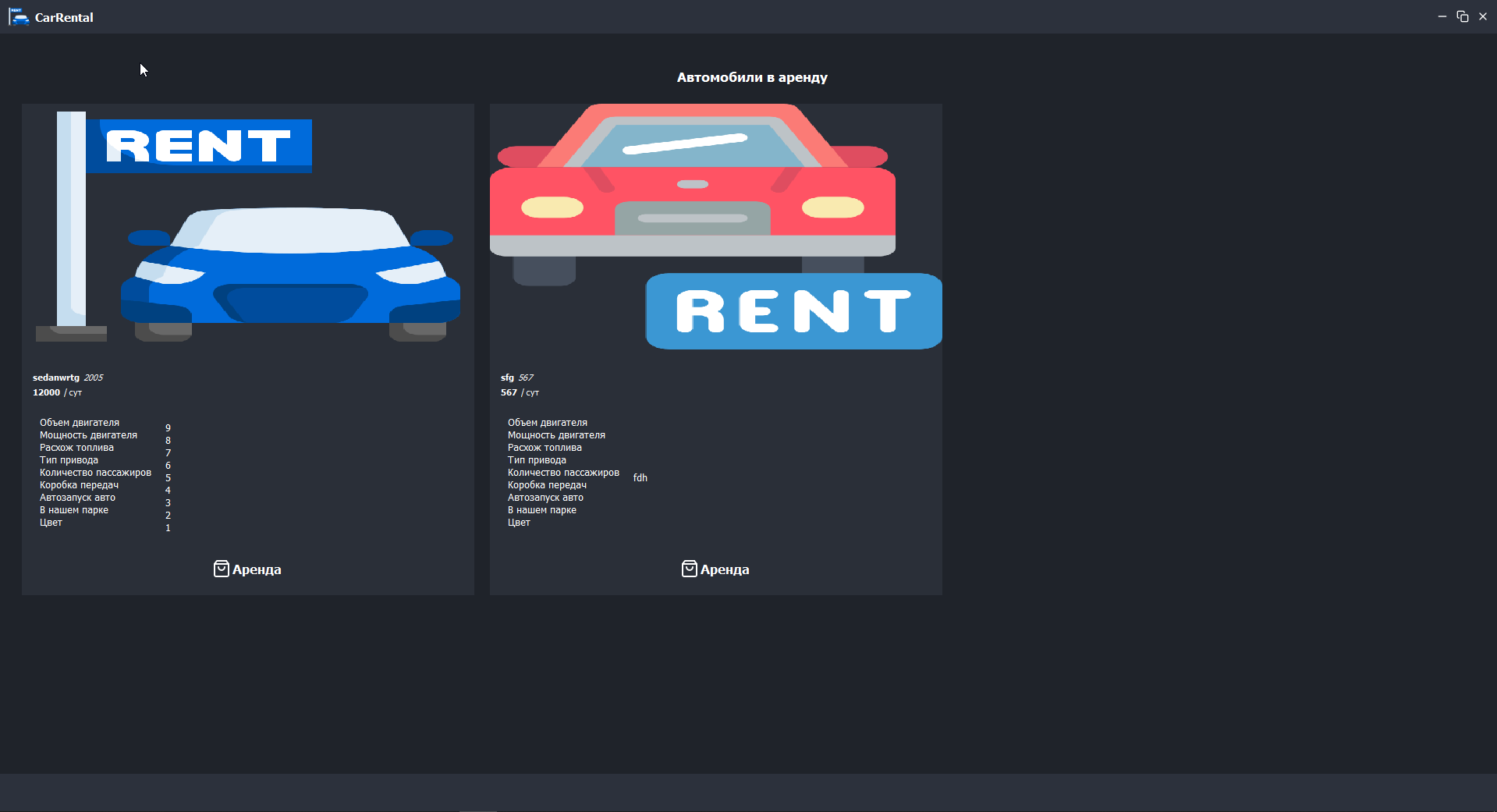


Рисунок 21- «Главная страница пользователя 2.»

У каждой карточки автомобиля есть кнопка Аренда (смотреть рисунок 22), нажав на которую откроется форму отправки заявки на аренду (смотреть рисунок 23).

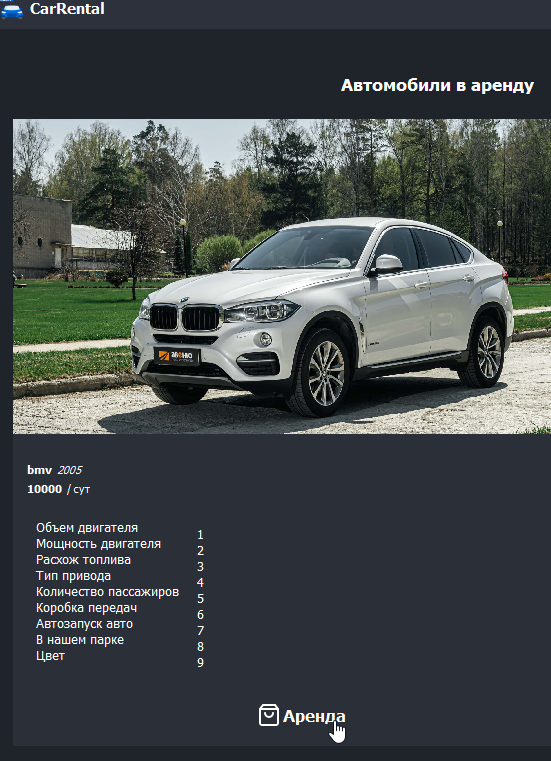


Рисунок 22- «Кнопка открытия формы заявки на аренду.»

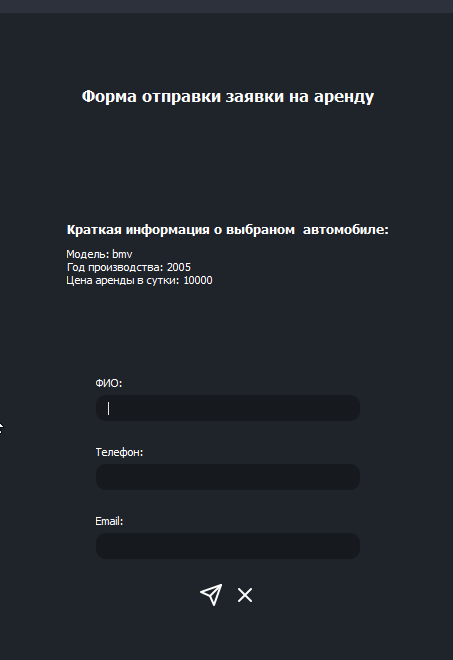


Рисунок 23- «Форма отправки заявки на аренду.»

При нажатии на крестик, откроется главная страница.

ПРИЛОЖЕНИЕ 9

Инструкция по эксплуатации для администратора.

Интерфейс администратора схож с интерфейсом пользователя в основных моментах, но так же он обладает уникальными элементами управления.

Для запуска приложения требуется стабильное интернет соединение. После открытия исполнительного файла, откроется страница выбора роли (смотреть рисунок 19). Выбираете Администратор.

Откроется главная страница приложения. На рисунке 24 отображена ситуация, когда нет актуальных предложений об аренде автомобилей. (смотреть Рисунок 24).

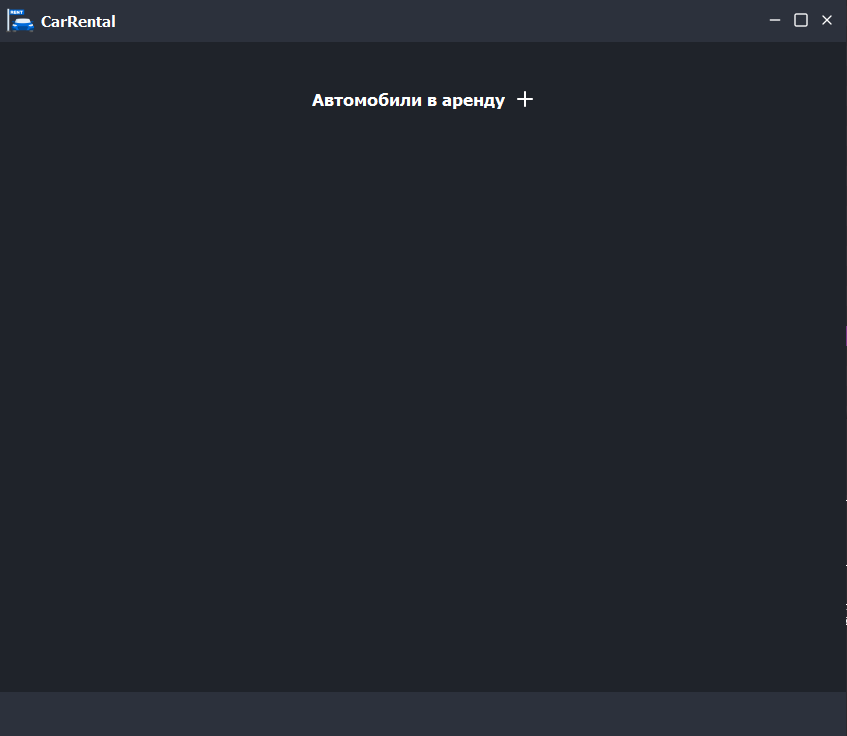


Рисунок 24 - «Главная страница администратора 1.»

На рисунке 25 отображена ситуации, когда есть актуальные предложения об аренде автомобилей. Можно ознакомиться с автомобилем, а именно с его техническими характеристики, а так же с ценой аренды (смотреть рисунок 25).

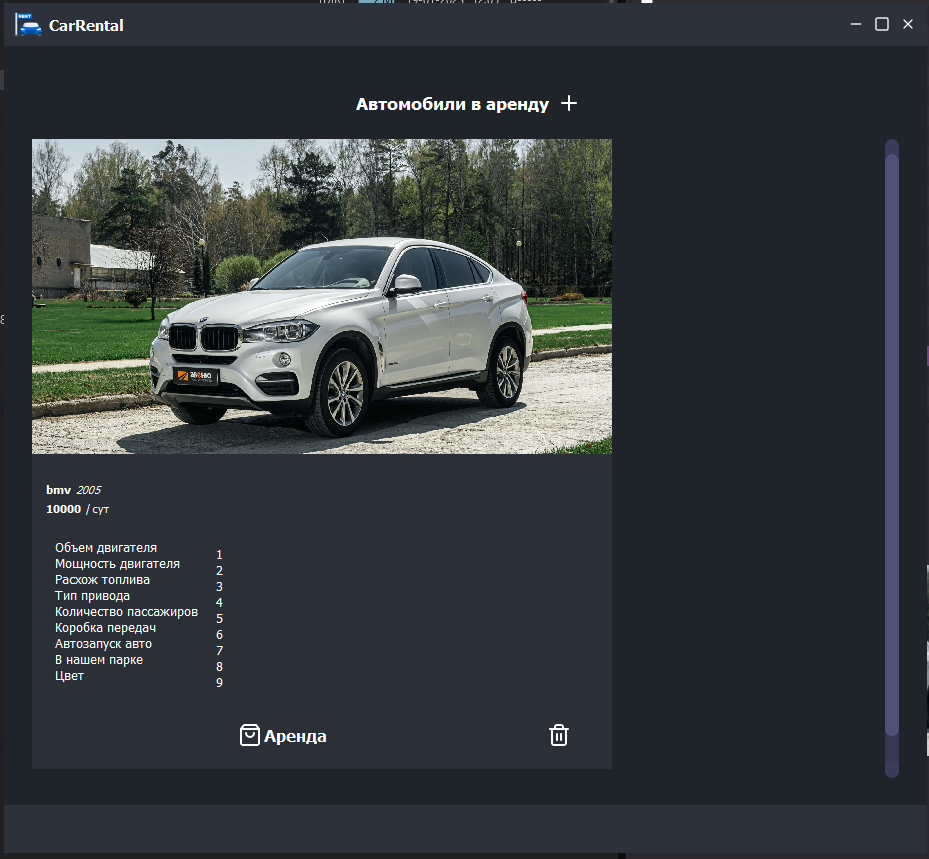


Рисунок 25 - «Главная страница администратора 2.»

Нажав на кнопку + (смотреть рисунок 26), откроется страница с формой заполнения для добавления автомобиля (смотреть рисунок 27).

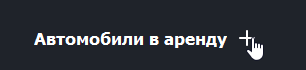


Рисунок 26 - «Главная страница администратора. Кнопка добавить автомобиль.»

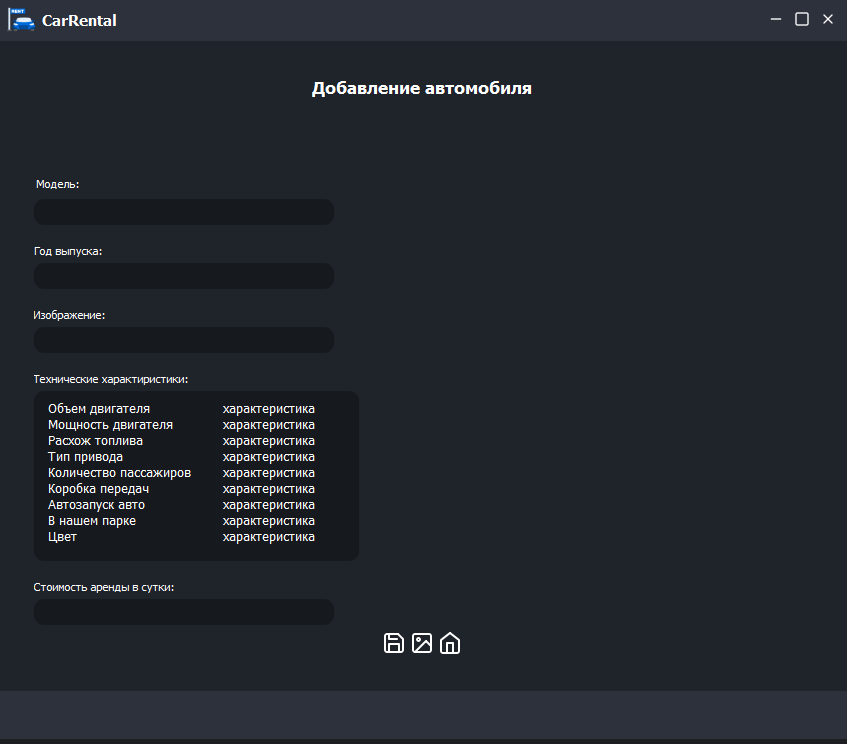


Рисунок 27 - «Форма добавления автомобиля.»

Все поля должны быть заполнены. Поля:

* Модель: модель/марка автомобиля, строковой тип данных.
* Год выпуска: год выпуска авто, числовой тип данных.
* Изображение: абсолютный путь к изображению, строковый тип данных.
* Технические характеристики: набор свойств/характеристик автомобиля, строковый тип данных. Не должно остаться ни одной незаполненной характеристики.
* Стоимость аренды в сутки: стоимость аренды в сутки, числовой тип данных.

Нажав кнопку Preview (смотреть рисунок 28) выведется на экран изображение, на которое указывает путь в поле Изображение (смотреть рисунок 29).

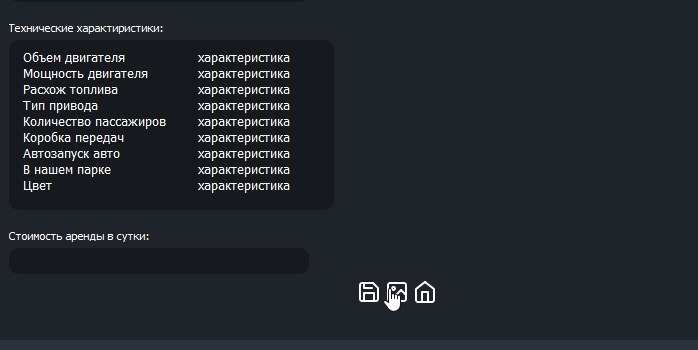


Рисунок 28 - «Кнопка Preview выбранного изображения.»

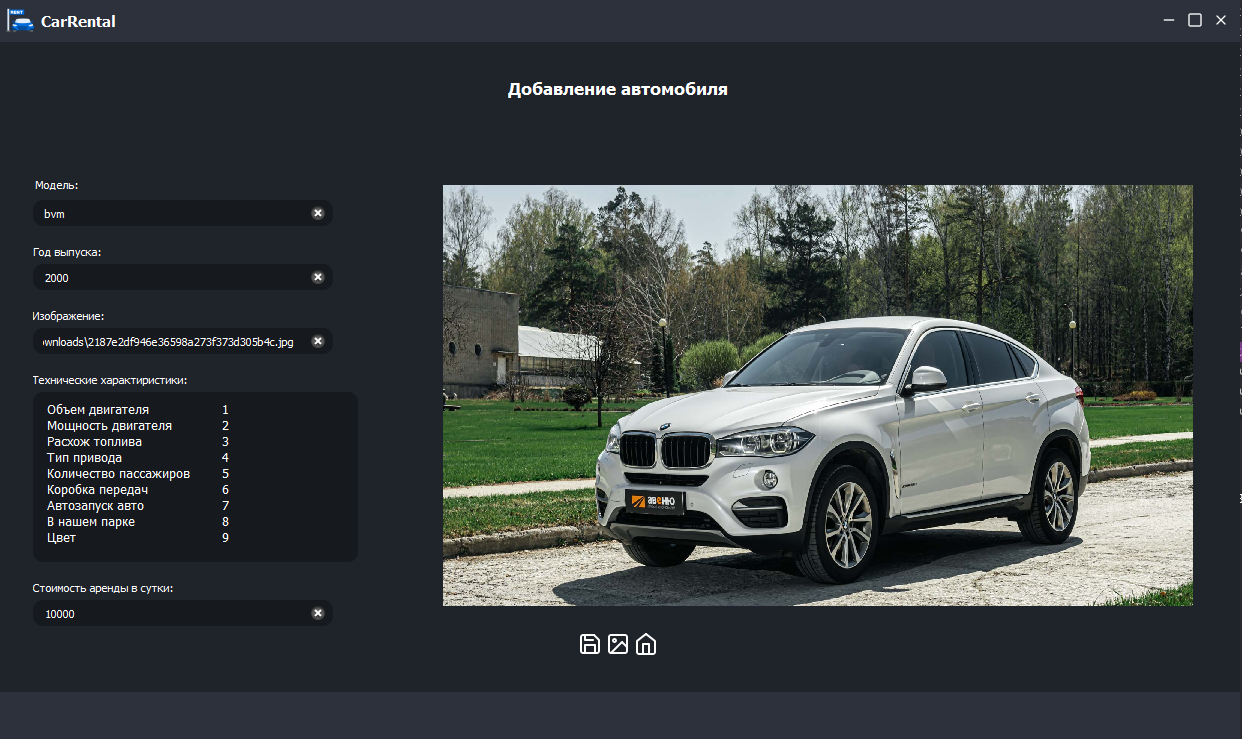


Рисунок 29 - «Preview выбранного изображения.»

Нажав кнопку сохранить (смотреть рисунок 30), произойдет добавления автомобиля в базу данных данных и в список автомобилей на главной странице, очистятся поля формы добавления автомобиля.

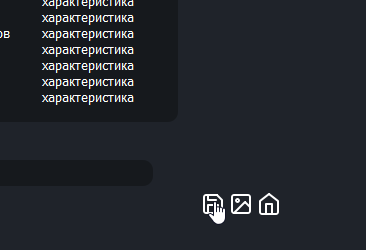


Рисунок 30 - «Кнопка сохранения автомобиля.»

При нажатии на кнопку домой (смотреть рисунок 31), произойдет переход на главную страницу. Данные внесенные в форму добавления автомобиля сохранятся в форме. Можно будет дописать их впоследствии. При выходе из приложения, эти данные будут утеряны.

На главной странице на каждой карточке автомобиля есть кнопка удаления выбранного автомобиля (смотреть рисунок 31). Нажав эту кнопку, произойдет удаление автомобиля из базы данных и из списка автомобилей на главной странице.

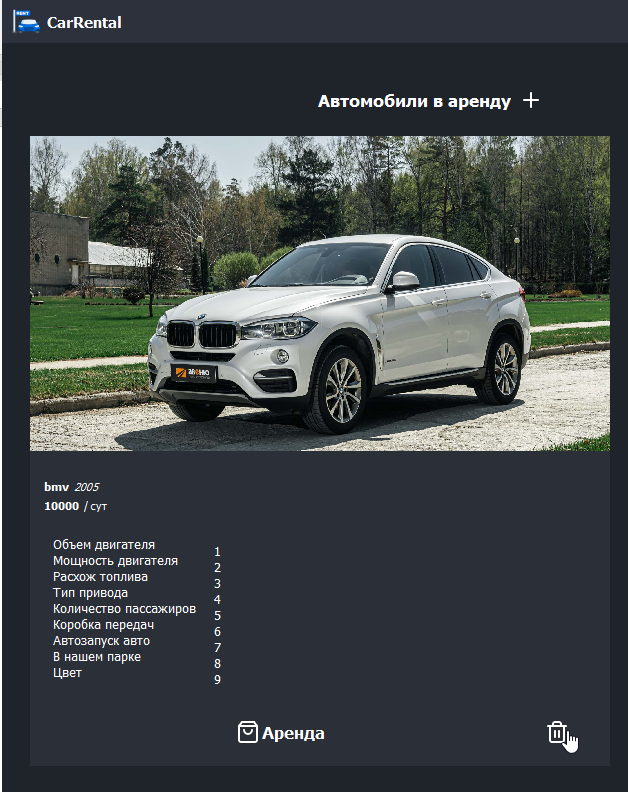


Рисунок 31 «Кнопка удаления автомобиля.»

ПРИЛОЖЕНИЕ 10