**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ**

**УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

Специальность 1-40 04 01 «Информатика и технологии программирования»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по дисциплине «Объектно-ориентированное программирование»

на тему: **«Моделирование услуг автопарка, предоставляющего аренду автомобилей населению»**

Исполнитель: студент гр. ИП-21 Лютиков М. С.

Руководитель: ст. преподаватель

Шибеко В. Н.

Дата проверки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата допуска к защите: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подписи членов комиссии

По защите курсового проекта: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Гомель 2023

**СОДЕРЖАНИЕ**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | Введение5 | | | 1 Анализ предметной области6 | | |  | 1.1 Обзор объекта автоматизации и существующих аналогов6  1.2 Обзор используемых технологий7  1.3 Требования к проектируемому программному обеспечению8  1.4 Обзор существующих методов решения9 | | 2 Разработка архитектуры и моделирование предметной области11 | | |  | 2.1 Описание предметной области11  2.2 Структура базы данных и СУБД13 | |  | 2.3 Модели классов доменов17  2.4 Защита информации18 | | 3 Структура программного обеспечения21 | | |  | 3.1 Анализ программного кода21 | |  | 3.2 Описание интерфейса23 | | 4 Тестирование приложения44 | | |  | 4.1 Общее описание тестирования проекта44 | |  | 4.2 Функции и их реализация в формах и алгоритмах48 | | Заключение50 | | | Список использованных источников51 | | | Приложение А – Листинг программного кода52 | | | |
|  | |
|  | |
|  |  |
|  | |
|  |  |
|  |  |
|  | |
|  |  |
|  |  |
|  | |
|  | |
|  | |
|  | |

**ВВЕДЕНИЕ**

Современная социально-экономическая ситуация в Республике Беларусь, как и во всем мире, характеризуется стремительным развитием технологий и услуг, включая сферу автотранспортных услуг. Актуальной проблемой, которая воздействует на уровень жизни населения и экономику страны в целом, становится необходимость эффективного использования транспортных средств, особенно в условиях ограниченных ресурсов и повышенного спроса на транспортные услуги.

В последнее время наблюдается рост популярности услуг аренды автомобилей, что связано с изменением транспортных предпочтений населения. В свою очередь, возникает потребность в развитии эффективных моделей организации и управления автопарками, предоставляющими такие услуги.

Аренда автомобилей становится все более востребованной как на короткий, так и на длительный срок. Это требует внедрения новых технологий и методов управления, которые могут повысить эффективность работы автопарка, улучшить качество обслуживания клиентов и повысить уровень удовлетворенности потребителей.

В 2020-е годы стала заметна тенденция к использованию информационных технологий в управлении автопарками. При этом, одним из основных направлений становится моделирование услуг автопарка, которое позволяет оптимизировать процессы управления и планирования, учесть предпочтения и потребности клиентов, а также повысить экономическую эффективность предприятия.

В соответствии с концепцией развития автопарка, важнейшим звеном становится создание системы моделирования услуг, которая обеспечивает анализ и прогнозирование различных сценариев работы автопарка. Она должна включать в себя комплекс подходов и методов, позволяющих управлять ресурсами автопарка и предоставлять услуги аренды автомобилей наиболее эффективным образом.

Создание системы моделирования услуг автопарка способно помочь решить множество проблем, с которыми сталкиваются такие предприятия в своей работе. Это включает, но не ограничивается, оптимизацией использования автомобилей, прогнозированием спроса на услуги, управлением рисками, связанными с простоем транспортных средств, и улучшением качества услуг.

В целом, разработка и внедрение системы моделирования услуг автопарка является важным шагом в повышении эффективности обслуживания населения и обеспечения его мобильности. Она также может способствовать снижению экологического воздействия транспорта, оптимизируя использование автомобилей и снижая их простой.

**1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

* 1. **Обзор объекта автоматизации и существующих аналогов**

В современном быстро развивающемся мире потребители и предприятия все больше полагаются на услуги аренды автомобилей для удовлетворения своих транспортных потребностей. Это особенно актуально в условиях глобализации и мобильности, когда необходимость в перемещениях на автомобиле возникает все чаще, а стоимость владения автомобилем может быть высокой. Предприятия, предоставляющие услуги по аренде автомобилей, сегодня находятся во многих местах: в аэропортах, городах, туристических курортах, в деловых и торговых центрах, и даже в медицинских и образовательных учреждениях. Поэтому, безусловно, требуется автоматизированная система для управления этим сложным процессом.

Система моделирования услуг автопарка, которая будет представлена здесь, нацелена на автоматизацию процесса управления автопарком и организации аренды автомобилей. Эта система обеспечивает эффективное управление ресурсами автопарка, упрощает процесс бронирования и аренды автомобилей, облегчает взаимодействие между клиентами и заказами, и в целом улучшает качество услуг.

Можно сравнить эту систему с веб-сайтами и мобильными приложениями, которые также предлагают ряд услуг, однако система моделирования предлагает ряд ключевых преимуществ. Одно из главных - более быстрая и надежная обработка данных благодаря минимальной нагрузке на источники хранения данных. Это особенно важно, учитывая объем информации, которым необходимо управлять в рамках автопарка, включая данные об автомобилях, их состоянии, информацию о клиентах и детали аренды.

Многие государственные и частные автопарки, работающие с подобными большими и разнообразными наборами данных и имеющие различные требования к уровню доступа к этим данным, нуждаются в удобной и эффектив- ной системе для просмотра, добавления, редактирования, удаления, поиска и сохранения записей. Так, моделирующая система, разработанная в рамках этого курсового проекта, будет служить таким инструментом, позволяя автопаркам гибко управлять своими ресурсами и обеспечивать высокий уровень обслуживания клиентов.

Важной частью разработки этой системы является создание простого, но функционального и удовлетворяющего всем требованиям интерфейса. Это поможет обеспечить эффективную работу сотрудников автопарка и улучшить обслуживание клиентов. Система будет обладать интуитивно понятными и удобными средствами для работы с данными, которые максимально облегчат процесс работы с ними.

Среди обязательных информационно-справочных услуг, которые будут предоставляться клиентам, пользователям и сотрудникам автопарка с помощью моделирующей системы, можно выделить следующие:

– информация о доступных для аренды автомобилях, их характеристиках и стоимости аренды, а так же местоположении после заказа;

– функция бронирования автомобилей на определенный период;

– информация о состоянии бронирования и истории бронирований пользователя.

Таким образом, разработка моделирующей системы для услуг автопарка, предоставляющего аренду автомобилей населению, представляет собой актуальную и важную задачу, решение которой позволит повысить эффективность работы автопарков и уровень удовлетворенности клиентов.

* 1. **Обзор используемых технологий**

Для реализации моделирующей системы автопарка в форме *Windows*-приложения в этом проекте будет использоваться язык программирования *C#*. Это высокоуровневый компилируемый язык общего назначения со статической типизацией, который применим для создания широкого спектра приложений. *C#* в настоящее время считается одним из самых популярных и широко используемых языков программирования.

*C#* обладает богатыми возможностями для работы с памятью, унаследованными от Си и *C++*. В связи с этим *C#* часто применяется в объектно-ориентированном программировании, включая создание операционных систем, драйверов, разнообразных утилит, антивирусов, веб-сервисов и т.д. Кроме того, он широко используется для создания графических приложений, различного рода прикладных программ и даже игр с насыщенной визуализацией.

В рамках данного проекта выбрана технология *Windows Forms* в среде разработки *Microsoft Visual Studio 2022*, основанная на платформе *C# .NET*. Эта платформа предоставляет полный набор функционала для создания приложений, работающих на операционной системе *Windows*.

*C#* является компилируемым языком, а это означает, что компилятор переводит исходный код на *C#* в исполняемый файл, содержащий набор машинных инструкций. Однако, поскольку разные платформы имеют свои особенности, скомпилированные программы нельзя просто перенести с одной платформы на другую без дополнительных действий. Несмотря на это, исходный код на *C#* в большинстве случаев обладает высокой степенью переносимости, если не используются функции, специфичные для определенной операционной системы. К тому же, наличие компиляторов, библиотек и инструментов разработки для большинства распространенных платформ позволяет компилировать один и тот же исходный код на *C#* в приложения под эти платформы.

* 1. **Требования к проектируемому программному обеспечению**

После анализа предметной области, к проектируемому приложению можно выдвинуть несколько требований.

Любой пользователь, установивший приложение на свое устройство, должен иметь возможность ознакомиться с основной информацией службы (информацией о заказах, профиле, ограничениях и т. д.). Пользователям должна быть предоставлена возможность пройти стандартный процесс авторизации (если пользователь уже является клиентом службы) или регистрации (если пользователь хочет стать клиентом службы).

Клиент службы должен обладать как минимум тем же функционалом, что и новый пользователь. Кроме того, клиент может отслеживать состояние своего заказа и просматривать историю своих заказов через личный кабинет.

Служба должна обладать возможностью добавления, редактирования, изменения и удаления различной информации о предоставляемых услугах и о клиентах. Для организации работы с клиентами должна быть предусмотрена автоматическая система, обладающая необходимым функционалом для обработки заказов и безошибочного взаимодействия с клиентами. Для обеспечения корректности работы, должна быть предусмотрена роль администратора, которая включает в себя возможность корректировки данных о персонале и обеспечения безопасности информации.

В контексте программного обеспечения для сервиса каршеринга, интерфейс играет ключевую роль. Для большинства пользователей, не знакомых с деталями программирования, интерфейс - это весь софт. Независимо от того, сколько времени и сил было вложено в разработку бэкенда и функциональности, удобство использования приложения во многом определяется его интерфейсом.

Удобство взаимодействия с приложением каршеринга важно для обеспечения комфортного пользовательского опыта. В процессе создания данного приложения учтены следующие рекомендации экспертов в области *UI/UX*:

– интерфейс должен быть интуитивно понятным, чтобы пользователю не требовалось дополнительных объяснений о том, как пользоваться приложением;

– не должно быть лишней информации, которая может отвлекать пользователя от выполнения задач. Не нужны лишние баннеры, разделы и категории;

– яркость и контрастность элементов интерфейса должны быть выбраны таким образом, чтобы обеспечить комфортное восприятие информации;

– размер символов и элементов управления должен быть оптимизирован для обеспечения быстрого и точного восприятия информации.

Приложение должно иметь возможность взаимодействовать с базой данных и предоставлять к ней доступ в удобной и подходящей форме. Кроме того, должны быть предусмотрены различные уровни доступа к данным и возможность взаимодействия с ними для различных ролей пользователей: неподтверждённого клиента, клиента, сотрудника и администратора.

В итоге, с учетом всех вышеуказанных требований, приложение каршеринга должно представлять собой интуитивно понятный и удобный в использовании интерфейс, что снизит порог входа для пользователей всех возрастных групп. Это, в свою очередь, уменьшит нагрузку на поддержку со стороны сервиса каршеринга, обеспечит эффективное выполнение всех задач и позволит быстро обновлять информацию.

**1.4 Обзор существующих методов решения**

Сегодня в области предоставления услуг по аренде автомобилей (каршеринг) функционирует большое количество современных платформ, оснащенных передовыми технологиями, включая веб-сайты, мобильные приложения и информационные системы. Таким образом, на рынке существует множество возможных подходов к созданию подходящего программно-аппаратного комплекса для каршеринга. Стоит отметить, что эти решения обладают более широким функционалом и возможностями по сравнению с приложением, которое планируется разработать в рамках данного проекта.

Одним из примеров уже существующих решений в данной области может служить приложение «*Zipcar*», разработанное для автоматизации процесса аренды автомобилей. Этот веб-сайт и мобильное приложение обладают понятным интерфейсом, что позволяет клиентам быстро и без лишних сложностей осуществить бронирование автомобиля. Приложение обеспечивает высокий уровень безопасности благодаря встроенным алгоритмам шифрования данных пользователей и защищенным каналам связи с базой данных сервиса. Благодаря встроенному программному обеспечению, любой клиент может ознакомиться со списком доступных автомобилей и совершить необходимую операцию.

Приложение для каршеринга не требует специальных навыков программирования для его эксплуатации. Владельцы автопарка могут самостоятельно наполнить его необходимыми данными: описаниями автомобилей, фотографиями, и прочей информацией. Приложение обладает следующими функциями:

– информирование зарегистрированных пользователей об изменениях;

– просмотр доступных для аренды автомобилей;

– возможность прохождения процесса бронирования автомобиля;

– предоставление информации о техническом состоянии автомобилей;

– предоставление контактной информации службы поддержки;

– доступ к информации о партнёрах и специальных предложениях;

– процесс оплаты аренды;

– раздел с ответами на часто задаваемые вопросы;

– возможность связаться со службой поддержки;

– возможность получения электронного чека после совершения оплаты.

Другими примерами служб с подобным функционалом являются сайты таких каршеринговых компаний как «*Car2Go*», «*DriveNow*» и «*Getaround*», которые используют продвинутые технологии, такие как: сканирование пластиковых карт, системы *GPS*-слежения, датчики движения, аудиосистемы, сканеры штрих-кодов, системы распознавания лиц и т. д.

Основываясь на анализе существующих программных решений, можно сделать вывод, что реализация всего существующего функционала может быть затруднительной в рамках данного проекта (согласно [3]) из-за ограниченности программных и технических ресурсов, а также необходимости большого количества времени на разработку. Однако основные задачи, которые должно решать приложение, включают в себя: наличие понятного пользовательского интерфейса, базы данных с информацией об автомобилях, возможность её редактирования и просмотра, а также различный уровень доступа для разных пользователей.

**2 РАЗРАБОТКА АРХИТЕКТУРЫ И МОДЕЛИРОВАНИЕ**

**ПРЕДМЕТНОЙ ОБЛАСТИ**

**2.1 Описание предметной области**

Модель предметной области в контексте каршеринга должна иллюстрировать сущности, включая автомобили, пользователей, бронирования, а также их взаимоотношения между собой. Сущности описывают объекты, которые являются предметом деятельности каршеринга, и субъекты, осуществляющие деятельность в рамках этой предметной области. Свойства объектов и субъектов реального мира описываются с помощью атрибутов.

Взаимоотношения между сущностями иллюстрируются с помощью связей. Правила и ограничения взаимоотношений описываются с помощью свойств связей. Обычно связи определяют либо зависимости между сущностями, либо влияние одной сущности на другую, согласно [4].

Для формирования представления о предметной области, а также для моделирования функциональных требований системы каршеринга и ее взаимодействия с внешними актерами используют *UML* диаграммы. Среди *UML* различают также диаграммы прецедентов, которые описывают функциональность системы в терминах ее использования. Они состоят из прецедентов (*use cases*) - действий, которые пользователи (актеры) могут выполнять в системе, и связей между ними.

Для каждого прецедента на диаграмме прецедентов указывается его название, описание и список актеров, которые могут выполнять этот прецедент. Также на диаграмме могут быть показаны связи между прецедентами, например, какой прецедент вызывает другой. Примеры таких *UML*-диаграмм в контексте каршеринга могут включать процессы бронирования автомобиля, оплаты за аренду, отмены бронирования и т.д.



Рисунок 2.1 – Диаграмма прецедентов и актёров (Клиент)

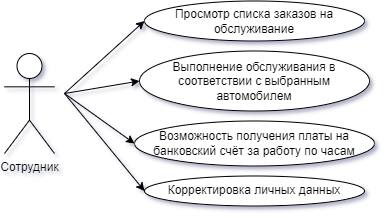


Рисунок 2.2 – Диаграмма прецедентов и актёров (Сотрудник)

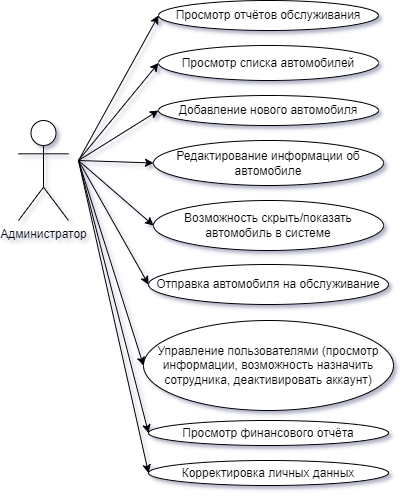


Рисунок 2.3 – Диаграмма прецедентов и актёров (Администратор)

В соответствии с моделью предметной области и *UML* диаграммами можно сделать анализ задачи. Необходимо реализовать три роли: клиент, сотрудник и администратор, которые будут взаимодействовать через услуги, оказываемые автопарком.

Клиент будет связан с услугами аренды автомобилей, сотрудник отвечает за обслуживание автомобилей, а администратор отвечает за управление данными автопарка, включая добавление, редактирование и удаление информации об автомобилях, а также, при необходимости, отключение системы.

В соответствии с анализом задачи, можно описать прецеденты и актеров:

– актер «Клиент» – это актер с возможностью забронировать автомобиль для аренды. Он имеет возможность ознакомиться с услугами автопарка;

– актер «Сотрудник» – это актер, обладающий возможностями добавления информации об обслуживании автомобилей;

– актер «Администратор» – это актер, который имеет возможность просматривать, редактировать, удалять, добавлять и обновлять всю информацию, необходимую для работы автопарка.

Описание основных прецедентов:

– прецедент «Просмотр информации об автомобилях» – прецедент для просмотра списка доступных автомобилей, их характеристик и стоимости аренды;

– прецедент «Авторизация и регистрация» – прецедент для создания аккаунта и доступа к полной версии сервиса;

– прецедент «Редактирование, удаление и добавление данных» – прецедент для просмотра и, в случае необходимости, выполнения различных манипуляций с данными, в результате которых они будут изменены.

Структура базы данных и *СУБД* в контексте моделирования услуг автопарка, предоставляющего аренду автомобилей населению (каршеринг), отличается простотой и эффективностью благодаря использованию *JSON*-формата для хранения данных и *LINQ* для их обработки.

*JSON*, что означает *JavaScript Object Notation*, это легковесный формат обмена данными, который основан на подмножестве *JavaScript*. Он обладает открытым стандартом и широко используется для представления структурированных данных в виде человеко-читаемого текста. *JSON* позволяет представлять данные в виде пар ключ-значение, и поддерживает базовые типы данных, такие как строки, числа, логические значения, массивы и вложенные объекты.

Дополнительное преимущество использования *JSON* заключается в высокой скорости чтения и записи данных. Благодаря простому и лаконичному формату, *JSON* обеспечивает быстрый парсинг и генерацию данных, что критически важно для высокопроизводительных приложений, таких как система каршеринга.

**2.2 Структура базы данных и СУБД**

База данных приложения представлена в виде списка объектов различных классов, каждый из которых отвечает за определенный аспект функционирования системы, согласно [2]. Ниже представлены основные компоненты базы данных:

– *User* - простой список объектов, содержащий информацию о пользователях системы;

– *Car* - более сложный список объектов, отражающий информацию об автомобилях, доступных для аренды;

– *Order* - сложный список объектов, в котором хранится информация обо всех заказах;

– *Payment* - сложный список объектов, содержащий данные об оплатах, произведенных пользователями;

– *BankTransaction* - простой список объектов, отслеживающий банковские транзакции, связанные с операциями аренды;

– *ServiceReport* - сложный список объектов, в котором сохраняются отчеты о техническом обслуживании автомобилей.

С использованием *JSON*-формата все данные загружаются в соответствующие массивы объектов при запуске программы, и сохраняются во время работы программы, что обеспечивает актуальность и целостность данных. Чтобы избежать проблему дублирования в *JSON*, после загрузки базы данных программа заменяет все копии объектов в сложных обектах на их ссылки из массивов.

База данных приложения представлена в виде списка объектов различных классов, каждый из которых отвечает за определенный аспект функционирования системы. С использованием *JSON*-формата все данные загружаются в соответствующие массивы объектов при запуске программы, и сохраняются во время работы программы, что обеспечивает актуальность и целостность данных.

При работе с данными активно используется технология *LINQ*, что позволяет эффективно осуществлять поиск, сортировку, группировку и другие операции над данными, а также упрощает код и делает его более читаемым. Это обеспечивает гибкость и масштабируемость приложения, позволяет быстро и эффективно реализовывать новые функции и улучшать существующие.

Пример *JSON* на рисунке 2.4:

{

"Users": [

{

"id": 1,

"name": "Иван Иванов",

"email": "ivanov@example.com",

"phone": "+375291234567",

"driving\_license": "AB1234567"

},

...

]

}

Рисунок 2.4 – Пример *JSON*

В ходе работы с *JSON* и связи данных с выбранным языком программирования была разработана структура данных, соответствующая требованиям курсового проекта и активно применяемая в приложении службы социальной помощи. На рисунке 2.5 представлена схема данных, использующиеся в ней отношения, основные атрибуты и связи.

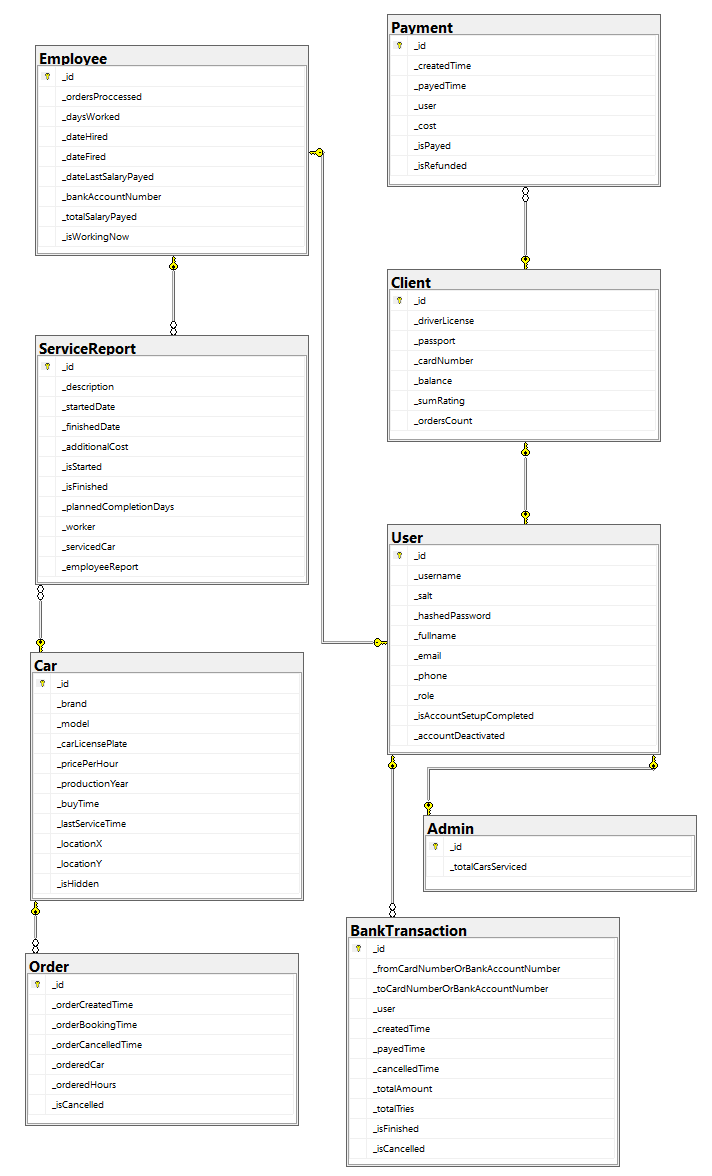


Рисунок 2.5 – Схема базы данных

Класс «*User*» (пользователь) является базовым классом для классов «*Client*», «*Employee*» и «*Admin*» в этом проекте. Он содержит общие атрибуты, такие как уникальный идентификатор (\_*id*), имя пользователя (\_*username*), хэш и соль пароля (\_*hashedPassword* и \_*salt*), *ФИО* (\_*fullname*), адрес электронной почты (\_*email*), номер телефона (\_*phone*) и роль (\_*role*). Класс также включает атрибуты, указывающие на завершение настройки учетной записи (\_*isAccountSetupCompleted*) и деактивацию учетной записи (\_*accountDeactivated*).

Класс «*Client*» (клиент), наследующий от класса «*User*» в этом проекте, содержит дополнительные атрибуты, такие как номер водительского удостоверения (\_*driverLicense*), паспортные данные (\_*passport*), номер банковской карты (\_*cardNumber*), баланс (\_*balance*), общий рейтинг (\_*sumRating*) и количество заказов (\_*ordersCount*).

В этом проекте класс «*Employee*» (сотрудник) также наследует от класса «*User*» и содержит дополнительные атрибуты, такие как количество обработанных заказов (\_*ordersProccessed*), количество отработанных дней (\_*daysWorked*), даты принятия на работу, увольнения и последней выплаты зарплаты (\_*dateHired*, \_*dateFired*, \_*dateLastSalaryPayed*), номер банковского счета (\_*bankAccountNumber*), общая выплаченная зарплата (\_*totalSalaryPayed*) и статус работы (\_*isWorkingNow*). Класс также содержит статический атрибут «*SalaryPerDay*» (зарплата в день).

Класс «*Admin*» (администратор), наследующий от класса «*User*» в этом проекте, содержит атрибут, указывающий на общее количество обслуживаемых автомобилей (\_*totalCarsServiced*).

Класс «*Car*» (автомобиль) в этом проекте содержит атрибуты, такие как уникальный идентификатор (\_*id*), марка (\_*brand*), модель (\_*model*), государственный номер (\_*carLicensePlate*), стоимость аренды в час (\_*pricePerHour*), год выпуска (\_*productionYear*), время покупки (\_*buyTime*), последняя дата обслуживания (\_*lastServiceTime*), координаты местоположения (\_*locationX* и \_*locationY*) и статус скрытия автомобиля (\_*isHidden*).

Класс «*Order*» (заказ) содержит информацию об оплате (\_*orderPayment*), список дополнительных платежей (\_*orderExtendPayments*), и статус отмены заказа (\_*isCancelled*).

Класс «*Payment*» (оплата) содержит атрибуты, такие как уникальный идентификатор (\_*id*), время создания и оплаты (\_*createdTime*, \_*payedTime*), клиент (\_*user*), стоимость (\_*cost*), и статусы оплаты и возврата средств (\_*isPayed*, \_*isRefunded*).

Класс «*BankTransaction*» (банковская транзакция) в проекте содержит атрибуты, такие как уникальный идентификатор (\_*id*), номера счетов отправителя и получателя (\_*fromCardNumberOrBankAccountNumber*, \_*toCardNumberOrBankAccountNumber*), пользователь (\_*user*), время создания, оплаты и отмены транзакции (\_*createdTime*, \_*payedTime*, \_*cancelledTime*), общая сумма (\_*totalAmount*), общее количество попыток (\_*totalTries*), и статусы завершения и отмены транзакции (\_*isFinished*, \_*isCancelled*). Класс также содержит статические атрибуты, такие как номера банковских счетов центра обслуживания и организации, секретный код организации, количество попыток транзакции и время для выполнения транзакции.

Класс «*ServiceReport*» (отчет об обслуживании) в проекте содержит атрибуты, такие как уникальный идентификатор (\_*id*), описание (\_*description*), даты начала и окончания обслуживания (\_*startedDate*, \_*finishedDate*), дополнительная стоимость (\_*additionalCost*), статусы начала и завершения обслуживания (\_*isStarted*, \_*isFinished*), планируемое количество дней до завершения (\_*plannedCompletionDays*), сотрудник, выполняющий обслуживание (\_*worker*), обслуживаемый автомобиль (\_*servicedCar*), и отчет сотрудника (\_*employeeReport*).

В завершение, «*RolesContainer*» (контейнер ролей) в проекте является перечислением для определения ролей пользователя: клиент (*Client* = 1), сотрудник (*Employee* = 2), администратор (*Admin* = 3).

**2.3 Модели классов доменов**

После определения прецедентов и сценариев, следующим шагом является анализ основных сущностей для работы с базой данных в контексте моделирования услуг автопарка, предоставляющего аренду автомобилей населению (каршеринг).

Следовательно, мы можем выделить следующие доменные классы, соответствующие сущностям базы данных:

– Класс *User* – предназначен для хранения информации о пользователе системы. Он включает в себя все необходимые поля, такие как идентификатор пользователя, имя, контактные данные и т.д., а также конструктор для создания объекта данного класса;

– Класс *Car* – содержит информацию об автомобилях, доступных для аренды, включая идентификатор автомобиля, марку, модель, год выпуска и другие специфические характеристики;

– Класс *Order* – отражает информацию о заказах, включая идентификатор заказа, идентификатор пользователя, идентификатор автомобиля, дату и время начала и окончания аренды и так далее;

– Класс *Payment* – хранит информацию о платежах пользователей, включая идентификатор платежа, идентификатор пользователя, сумму платежа и дату платежа;

– Класс *BankTransaction* – содержит информацию о банковских транзакциях, связанных с платежами пользователей;

– Класс *ServiceReport* – предназначен для хранения отчетов о техническом обслуживании автомобилей;

– Класс *Admin*, *Employee* и *Client* представляют различные роли в системе и содержат специфическую для роли информацию и функциональность. Наследуются от класса *User*;

– Класс *JSON\_Export* используется для экспорта данных в формате *JSON*;

– Перечисление (*enum*) *RolesContainer* служит для управления и хранения информации о ролях в системе.

Для управления этими моделями и выполнения операций над данными используются контроллеры, представленные классами *AesGсm256*, *CryptographyControl*, *SaveLoadControl* и *UtilsControl*.

Вместо компонента *DataGridView*, используемого в Windows Forms для отображения и редактирования табличных данных, в данной системе используется *ListView*. *ListView* предлагает больше гибкости при отображении данных и позволяет представить данные в виде списка, таблицы или даже в виде маленьких и больших иконок.

Для взаимодействия с базой данных используется *LINQ*. Существует класс «*SaveLoadControl*», который содержит статические методы для выполнения различных операций с базой данных. Эти операции включают, но не ограничиваются, добавлением, обновлением, удалением и извлечением и сохранением данных из базы данных.

Важно отметить, что в нашей системе аренды автомобилей каждый из этих классов играет важную роль в обеспечении функциональности системы. Они вместе обеспечивают надежное и эффективное управление данными, что обеспечивает плавность и эффективность работы всей системы.

В заключение, структура классов домена и контроллеров, вместе с использованием *ListView* для визуализации данных и *JSON* для работы с базой данных, обеспечивает гибкость, масштабируемость и производительность, которые являются критически важными для успешной реализации системы каршеринга.

**2.4 Защита информации**

Защита информации представляет собой критически важную область деятельности любого предприятия, особенно в отраслях, где требуется обработка и хранение чувствительной информации, такой как персональные данные и финансовые операции. Важные принципы, которые следует соблюдать, включают обеспечение конфиденциальности, целостности и доступности данных.

В эпоху цифровизации, когда обработка, передача и хранение информации происходят в виртуальном пространстве, появляется ряд новых угроз. Для борьбы с ними в данной системе используются современные методы шифрования и защиты данных.

В конкретной системе каршеринга для шифрования файла базы данных «*db.json*» используется алгоритм *AesGcm256*. Это современный алгоритм шифрования, который обеспечивает высокий уровень безопасности данных. Он преобразует исходные данные с использованием секретного ключа, таким образом, что их можно восстановить только при использовании этого же ключа. Ключ хранится в закодированном алгоритмом *Base64* виде в файле «*enc.key*».

Для повышения безопасности учетных данных пользователей применяется процесс хеширования с использованием соли, реализованный в классе *CryptographyControl*. Хеширование с использованием соли - это метод, при котором к паролю пользователя добавляется случайная последовательность символов, известная как «соль», перед процессом хеширования. Это делает хеш-функцию более устойчивой к атакам по словарю и атакам поиском с применением «радужных таблиц», так как даже небольшие изменения в исходных данных приводят к значительному изменению результата хеширования (так называемый «лавинный эффект» хеш-функции).

В дополнение к этим мерам, для защиты исполняемых файлов приложения от обратного инжиниринга используется инструмент обфускации *ConfuserEx*. Обфускация кода – это процесс преобразования кода приложения в форму, которая сохраняет его функциональность, но делает его практически невозможным для понимания и анализа. Это дополнительная защита от попыток взлома и несанкционированного использования программного обеспечения. *ConfuserEx* обеспечивает высокий уровень обфускации *.NET* приложений, затрудняя процесс декомпиляции и понимания и исходного кода. Этот инструмент предлагает множество функций обфускации, включая переименование классов, методов и переменных, управление потоком и сокрытие строк. Благодаря этому код приложения становится более сложным для анализа и взлома, что повышает общий уровень защиты программного обеспечения.

Совокупность этих методов обеспечивает комплексную защиту информационной системы. Шифрование с помощью *AesGcm256* обеспечивает безопасное хранение и передачу данных (согласно [5]), хеширование паролей с использованием соли увеличивает уровень защиты от атак на учетные данные пользователей, а применение обфускации с использованием *ConfuserEx* защищает исходный код приложения от несанкционированного использования и взлома. Все эти меры помогают создать надежную и безопасную информационную среду.

Что касается примеров хеширования и шифрования, предположим, что у нас есть следующая соль и пароль:

Соль (в виде *Base64* строки): *MTIzNDU2Nzg5MGFiY2RlZg==*

Пароль: *MyS3cur3P@ssw0rd*

При применении указанной выше функции *HashPasswordWithSalt*, соль и пароль объединяются и проходят через алгоритм хеширования. После этого пароль может выглядеть примерно так:

Хешированный пароль: *h1F1cG2bI2p4v5E+6k3Lr1E+0bR...*

Обратим внимание, что в реальных условиях хешированный пароль будет значительно длиннее и сложнее.

Теперь рассмотрим шифрование строки с помощью *AesGcm256*. Допустим, у нас есть следующая исходная строка:

Исходная строка: *Sensitive Information*

При использовании функции *encrypt* класса *AesGcm256* с определенным ключом и вектором инициализации (*IV*), исходная строка может быть преобразована в следующий зашифрованный текст:

Зашифрованный текст:

*QklFMUJDMTItN0U3MC00NjZBLUIwQzEtMjM5NTZGQjY5NTgx*

Безусловно, это всего лишь примеры того, как можно использовать технологии хеширования и шифрования. В реальном мире результаты и объемы хеширования и шифрования могут варьироваться в широких пределах. Это происходит потому, что они напрямую зависят от ряда ключевых факторов, таких как выбранные алгоритмы, ключи шифрования и другие специфические параметры. Иногда разница в результатах может быть значительной и может варьироваться в зависимости от контекста и конкретных условий применения. Также следует учесть, что эффективность и безопасность хеширования и шифрования во многом зависят от правильного выбора и применения соответствующих методов. Поэтому всегда важно принимать во внимание эти факторы при реализации систем безопасности данных.

**3 СТРУКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ**

**3.1 Анализ программного кода**

Основной целью данного проекта является разработка интуитивно понятного приложения для аренды автомобилей, которое поможет упростить взаимодействие между компанией и клиентами, а также повысит эффективность и качество предоставляемых услуг. Приложение должно быть разработано с учетом потребностей и особенностей пользователей, включая сотрудников компании и клиентов, и предоставлять широкий набор функций для обеспечения удобства и эффективности взаимодействия.

В рамках данного проекта был использован язык программирования *C#*, применяющий объектно-ориентированный подход к программированию. *C#* был выбран из-за его широкой популярности и возможностей для создания масштабируемых и надежных приложений. Объектно-ориентированный подход позволяет разделить приложение на отдельные модули или классы, которые могут быть использованы повторно в различных частях программы. Это способствует повышению эффективности разработки и облегчает поддержку кода в долгосрочной перспективе.

Для создания пользовательского интерфейса в проекте была использована технология *Windows Forms*, позволяющая создавать элементарные абстрактные единицы с интерфейсом, такие как формы. Разработка пользовательского интерфейса производилась путем перетаскивания элементов управления на форму и настройки их свойств. *WinForms* предлагает большое количество предопределенных элементов управления, которые покрывают большую часть потребностей в создании пользовательского интерфейса. Это позволяет быстро и легко создавать функциональные и эстетически приятные окна приложений. Каждый элемент управления в *Windows Forms* имеет широкий набор настраиваемых свойств и событий, которые позволяют полностью настроить их внешний вид и поведение.

В ходе разработки приложения также были созданы специализированные классы для защиты данных, обработки связи с базами данных и манипуляции данными из *БД*.

Для обеспечения безопасности и защиты конфиденциальной информации, такой как личные данные клиентов, были разработаны классы, отвечающие за шифрование и дешифрование данных, управление доступом и аутентификацию пользователей. Эти классы включают методы для шифрования паролей и контроля доступа к конфиденциальной информации.

В данном проекте для взаимодействия с данными и выполнения операций чтения, записи и обновления был использован формат *JSON*. Формат *JSON* выбран ввиду его простоты, гибкости и универсальности, что делает его идеальным выбором для хранения и передачи данных в данном контексте.

Были разработаны специальные классы, которые осуществляют чтение и загрузку данных из *JSON* файлов в программу. Эти классы отвечают за открытие файлов, чтение данных, их парсинг в соответствующие объекты и структуры данных внутри программы. При этом обеспечивается возможность гибкого манипулирования данными, их обновления и сохранения обратно в *JSON* файл.

Важным аспектом работы с *JSON* файлами является корректное и безопасное чтение и запись данных. Поэтому были применены методы, предотвращающие повреждение или потерю данных в случае неожиданных ошибок или исключений. В частности, использованы механизмы, обеспечивающие сохранность исходных данных в случае неудачной записи, а так же механизмы, предотвращающие дублирование объектов.

Стоит отметить, что использование *JSON* файлов для хранения данных позволяет легко и быстро обрабатывать данные в различных языках программирования, что может быть особенно полезно при дальнейшем развитии и масштабировании проекта.

Таким образом, в рамках проекта был создан надежный и гибкий механизм работы с данными, обеспечивающий возможность эффективного управления данными автопарка и предоставления пользовательских услуг на высоком уровне.

Основная структура классов и форм проекта представлена на рисунке 3.1.

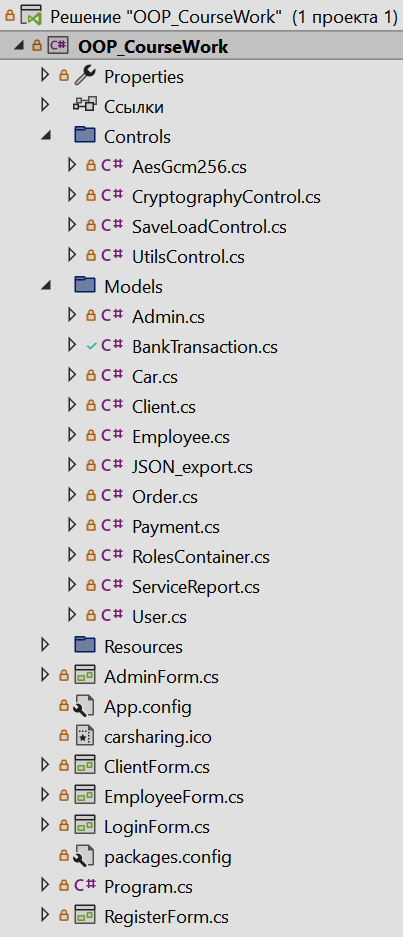


Рисунок 3.1 – Структура программы

Подобная структура была достигнута в результате написания программного кода и проектирования предметной области по заданию курсового проекта.

**3.2 Описание интерфейса**

Одним из основных требований к приложению автопарка, предоставляющего услуги аренды автомобилей, является наличие интуитивно понятного и простого в использовании интерфейса.

В данном контексте, главная страница, представленная на рисунке 3.2, разработана таким образом, чтобы быть удобной и информативной для всех пользователей.

Активация кнопки «Войти» позволяет пользователю пройти стандартный процесс авторизации в системе, как это показано на рисунке 3.2.

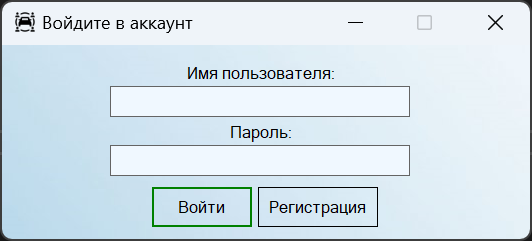


Рисунок 3.2 – Форма авторизации

При нажатии на кнопку «Регистрация», пользователю предоставляется возможность заполнить анкету, демонстрируемую на рисунке 3.3.

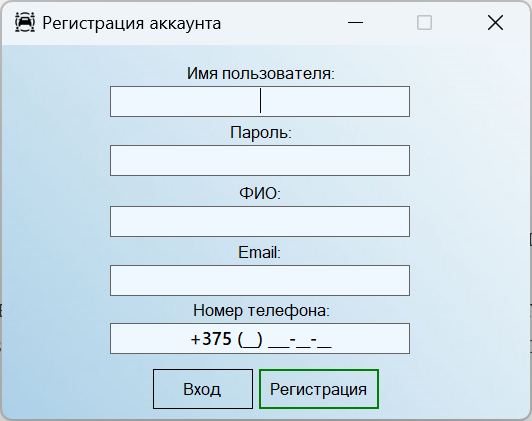


Рисунок 3.3 – Форма регистрации

Таким образом, в этом проекте был реализован подход, направленный на максимальное удобство использования приложения для конечного пользователя.

После авторизации под учётной записью с ролью клиента, пользователя встречает следующее окно, показанное на рисунке 3.4, с активной вкладкой «Список заказов».

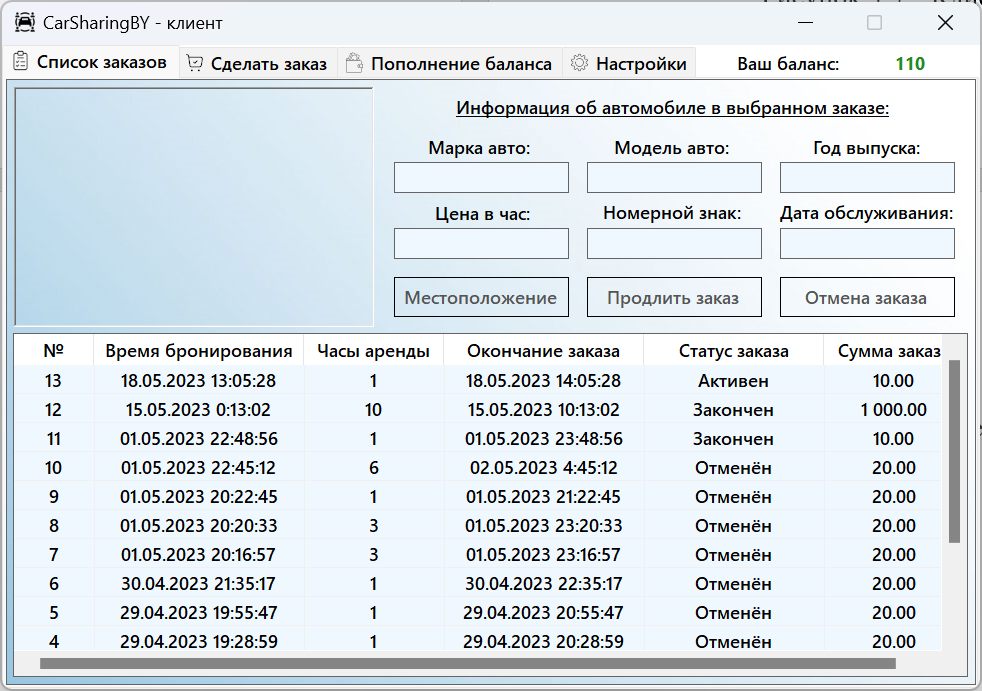


Рисунок 3.4 – Список заказов клиента

Пользователь может просмотреть историю своих заказов, нажав на любой из них в списке, а так же просмотреть информацию об автомобиле, как показано на рисунке 3.5.

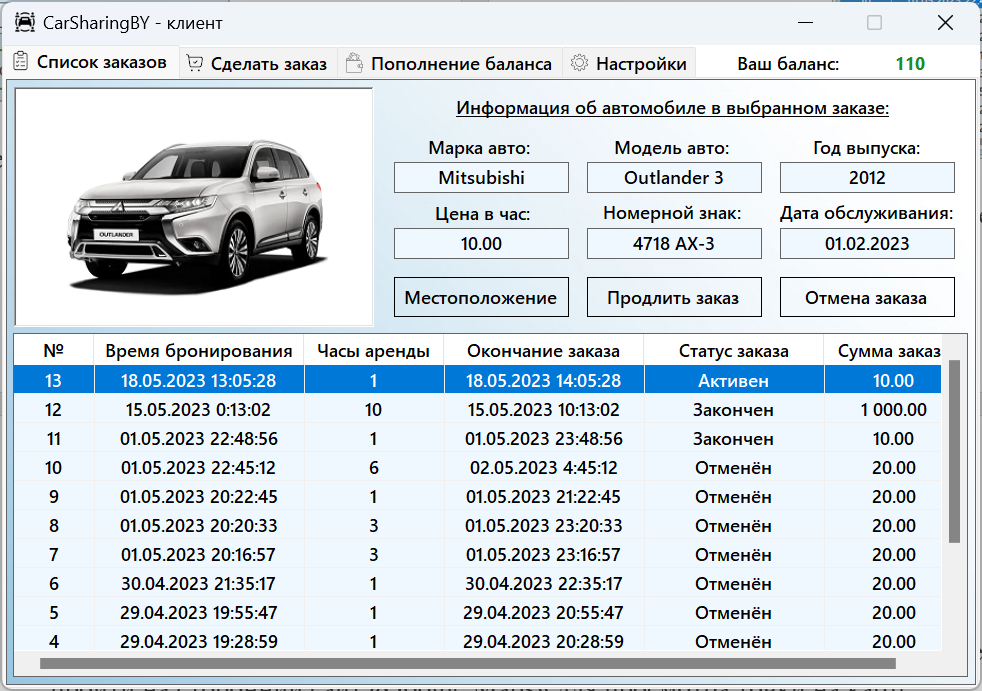


Рисунок 3.5 – Клиент выбрал заказ для просмотра информации

В зависимости от статуса заказа, меняют своё состояние с активного на неактивное (и наоборот) кнопки «Местоположение», «Продлить заказ», «Отмена заказа».

При нажатии кнопки «Местоположение» пользователю будет предложено пройти на сторонний сайт «*Google Maps*» для просмотра точки на карте местности, в которой находится автомобиль. У него будет возможность построить маршрут до точки, осмотреть местность, оценить расстояние.

При нажатии кнопки «Продлить заказ» пользователя встречает следующее окно, показанное на рисунке 3.6. В нём пользователю будет предложено указать количество часов, на которое он хочет продлить заказ. Максимальное суммарное время заказа автомобиля – не более 192 часов, что составляет 8 суток. Пользователю показывается возможное число часов для продления его заказа.

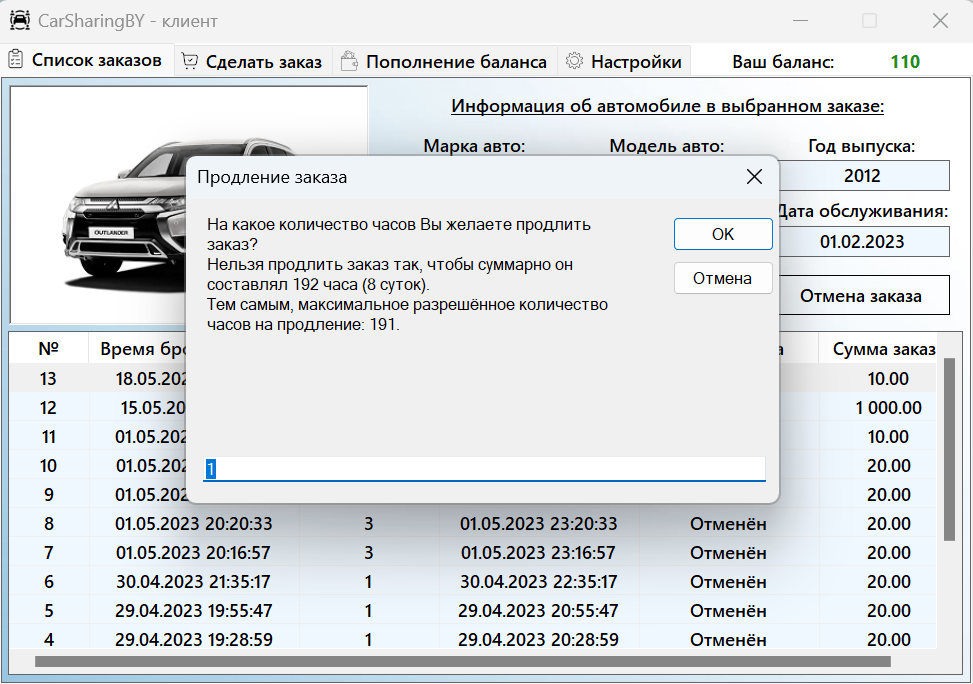


Рисунок 3.6 – Продление заказа клиентом

Пользователь может отказаться, однако, если он принимает решение продлить заказ, то он встречается со следующим интерфейсом. Пользователю представляется окно, иллюстрированное на рисунке 3.7. Это окно подтверждает, что продление заказа было успешно выполнено. Здесь стоит отметить, что продление заказа возможно только при наличии необходимых ресурсов. Ключевыми факторами являются: достаточное количество средств на балансе пользователя и правильность указанного количества часов для продления.

Предположим, пользователь желает продлить аренду, но у него недостаточно средств на балансе или количество часов, указанных для продления, находится за пределами разрешенного. В таких случаях, система реагирует соответствующим образом. Пользователю отображается еще одно окно, которое представлено на рисунке 3.8. Оно информирует пользователя о невозможности выполнить его запрос на продление аренды, указывая причину этого.

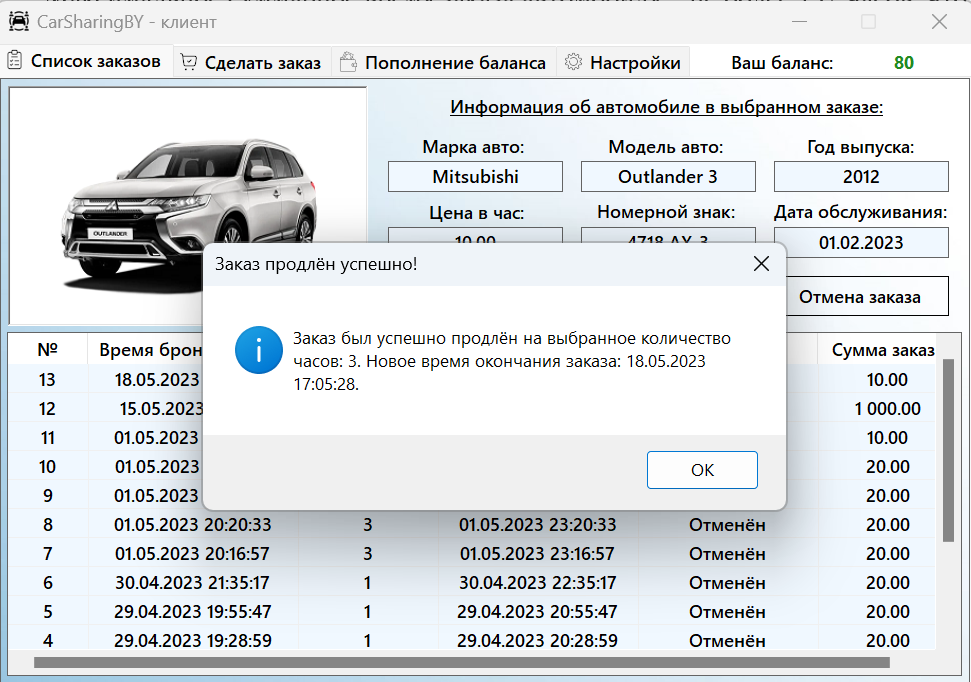


Рисунок 3.7 – Успешное продление заказа клиентом

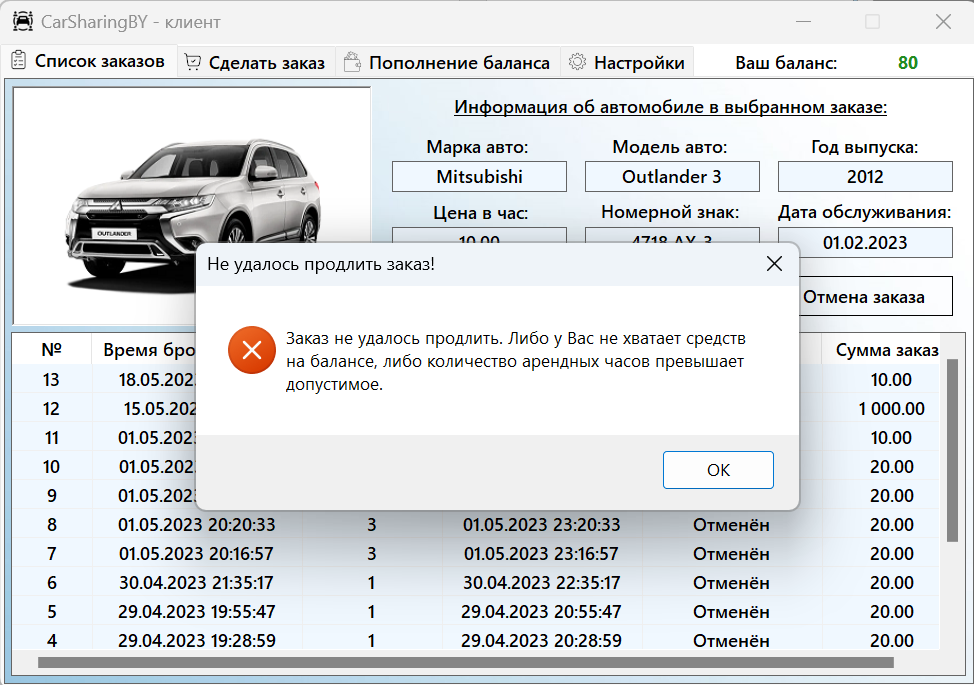


Рисунок 3.8 – Неуспешное продление заказа клиентом

При взаимодействии с приложением каршеринга пользователь имеет право отменить существующий заказ. Этот процесс начинается с нажатия кнопки «Отмена заказа». Это действие активно в приложении, при условии, что временные рамки бронирования позволяют это сделать.

После нажатия на кнопку «Отмена заказа», программа не сразу выполняет данное действие. Вместо этого она уточняет у пользователя, насколько он уверен в своем решении. Это делается для того, чтобы избежать случайной или преждевременной отмены заказа, которая может привести к неудобствам или ошибкам.

В случае, если пользователь подтверждает свое желание отменить заказ, система переходит к следующему этапу. Пользователю представляется новое окно, иллюстрация которого отображена на рисунке 3.9. В этом окне пользователю подтверждается, что отмена заказа произведена успешно.

Стоит отметить, что возможность отмены заказа ограничена определенными условиями. Наиболее важным из них является то, что заказ может быть отменен, только если время, на которое автомобиль был забронирован, еще не наступило. Это означает, что пользователь не может отменить заказ после начала времени его бронирования.

Если все условия для отмены заказа соблюдены и отмена успешно осуществлена, система автоматически возвращает средства на баланс пользователя. Это обеспечивает беспрепятственное возврат средств и облегчает процесс взаимодействия пользователя с системой каршеринга. Таким образом, программа стремится обеспечить максимальное удобство и гибкость для своих пользователей при управлении их заказами.

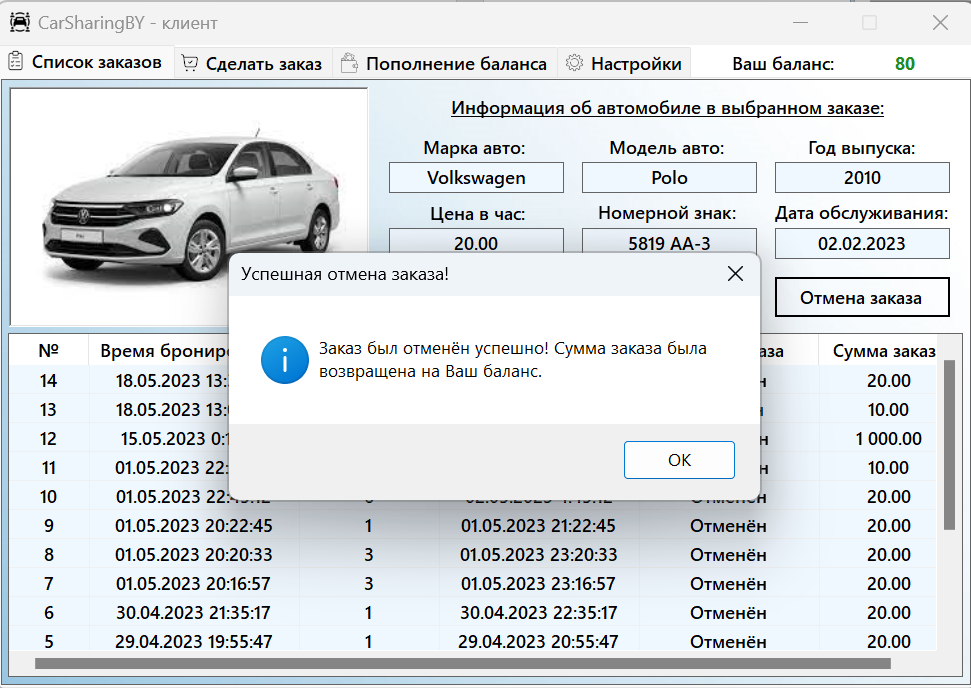


Рисунок 3.9 – Успешная отмена заказа клиентом

При переходе на вкладку «Сделать заказ» пользователя встречает окно, представленное на рисунке 3.10. На нём он может видеть список автомобилей, как доступных, так и недоступных для заказа, а так же информацию об автомобилях, такую как название и модель, год выпуска, цена аренды в час и фотографию. Есть специальные элементы для ввода количества часов аренды, времени бронирования, а так же элемент только для чтения, который покажет время окончания заказа с учётом указанных данных.

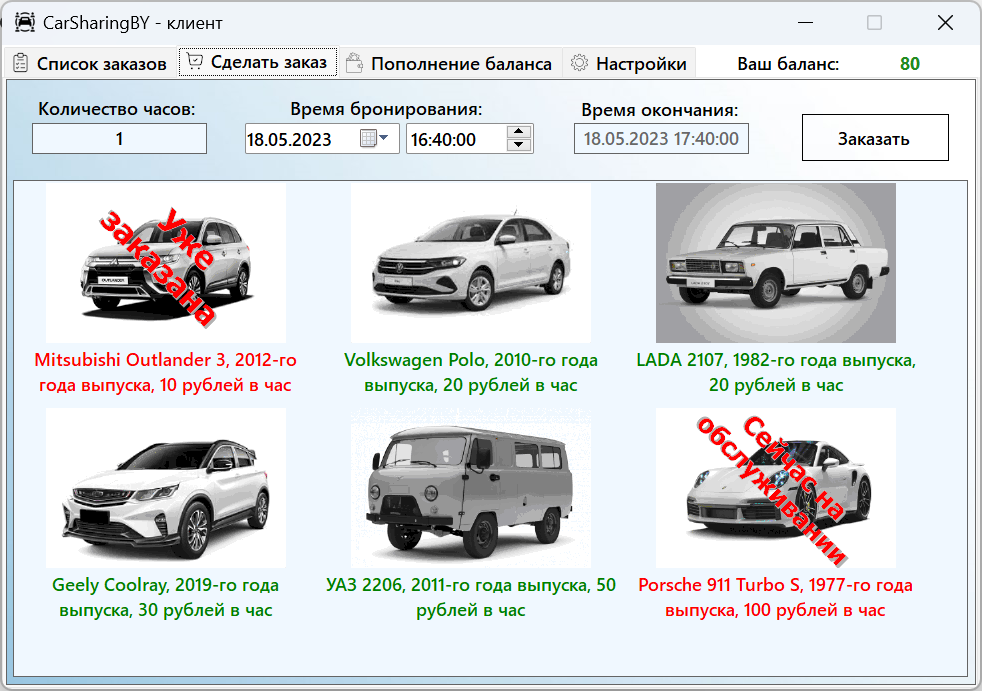


Рисунок 3.10 – Вкладка «Сделать заказ»

Кнопка «Заказать» проверяет введённые данные и уточняет у пользователя, уверен ли он в своём выборе. Если ответ утвердительный, то пользователь видит окно, представленное на рисунке 3.11. Машина становится недоступной для заказа на время использования, на ней появляется надпись «Уже заказана» (автомобиль невозможно заказать более чем на 192 часа, что равно 8 суткам). Аналогично, если на фотографии автомобиля присутствует надпись «Сейчас на обслуживании», то автомобиль так же недоступен для заказа. Если у пользователя нет ресурсов для того, чтобы сделать заказ, или же введённые данные некорректны, то система сообщит об этом. После успешного создания заказа он появляется во вкладке «Список заказов». Заказ нельзя сделать или продлить более трёх раз за последний час. Сделано это в противодействие возможности сабботажа системы злоумышленниками.

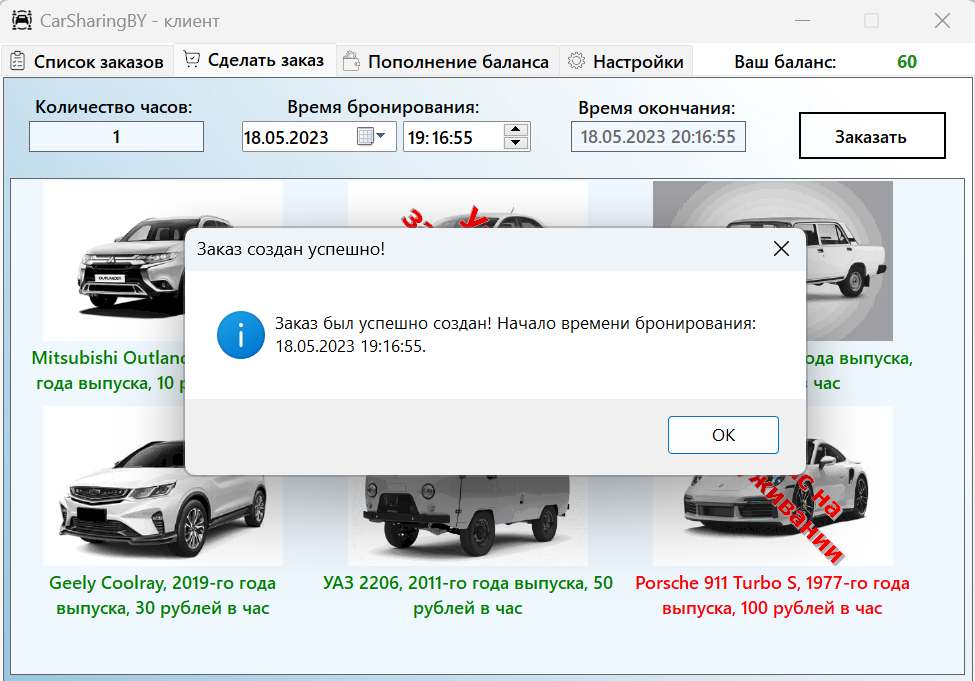


Рисунок 3.11 – Заказ успешно сделан пользователем

При переходу на вкладку «Пополнение баланса» пользователя встречает окно, указанное на рисунке 3.12. Здесь присутствуют следующие элементы: элемент для отображения номера карты (только для чтения), элемент для ввода секретного кода и суммы пополнения баланса.

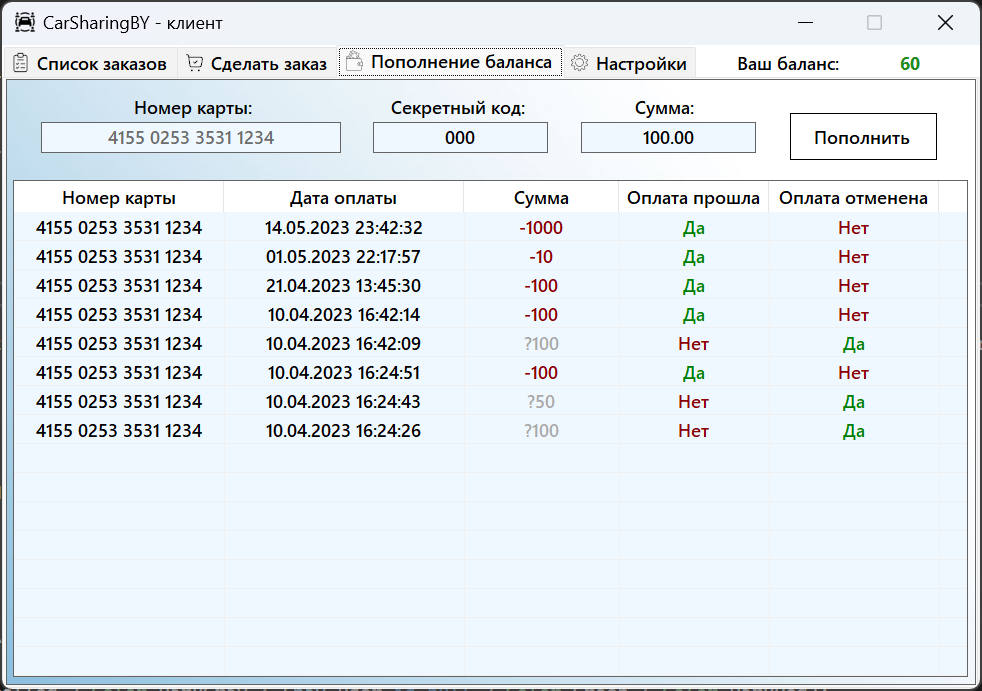


Рисунок 3.12 – Заказ успешно сделан пользователем

При нажатии кнопки «Пополнить», программа уточняет у пользователя, готов ли он пополнить баланс на выбранную сумму. В случае утвердительного ответа, программа проверяет данные для платежа, и если они не являются верными, то выдаётся сообщение, представленное на рисунке 3.13. Если же данные верны, то пользователю предоставляется сообщение, указанное на рисунке 3.14. После платежа секретный код, введённый пользователем, заменяется на значение по умолчанию из соображений безопасности, а так же нигде и никогда не хранится. Баланс обновляется до актуального значения.

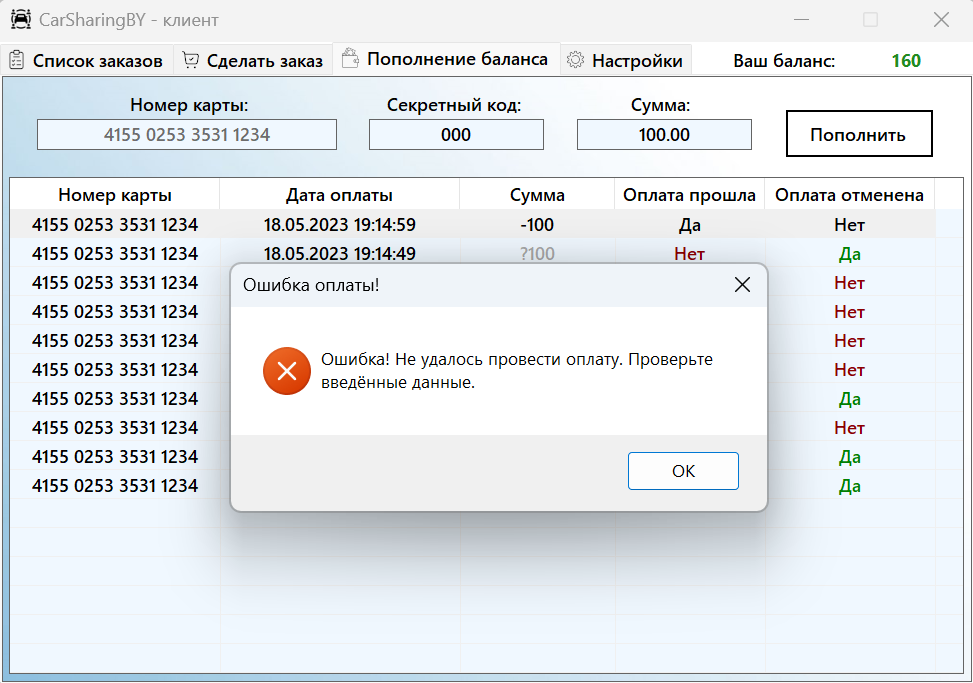


Рисунок 3.13 – Данные для оплаты неверны

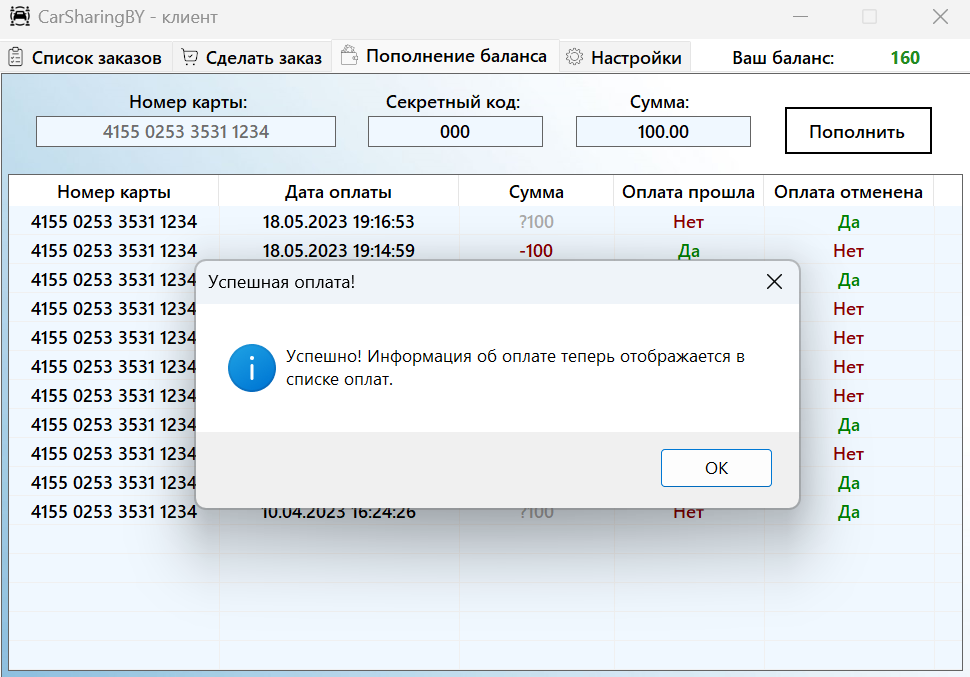


Рисунок 3.14 – Данные для оплаты верны

На рисунке 3.15 можно увидеть, что баланс обновился до актуального значения (прибавилось число сто).

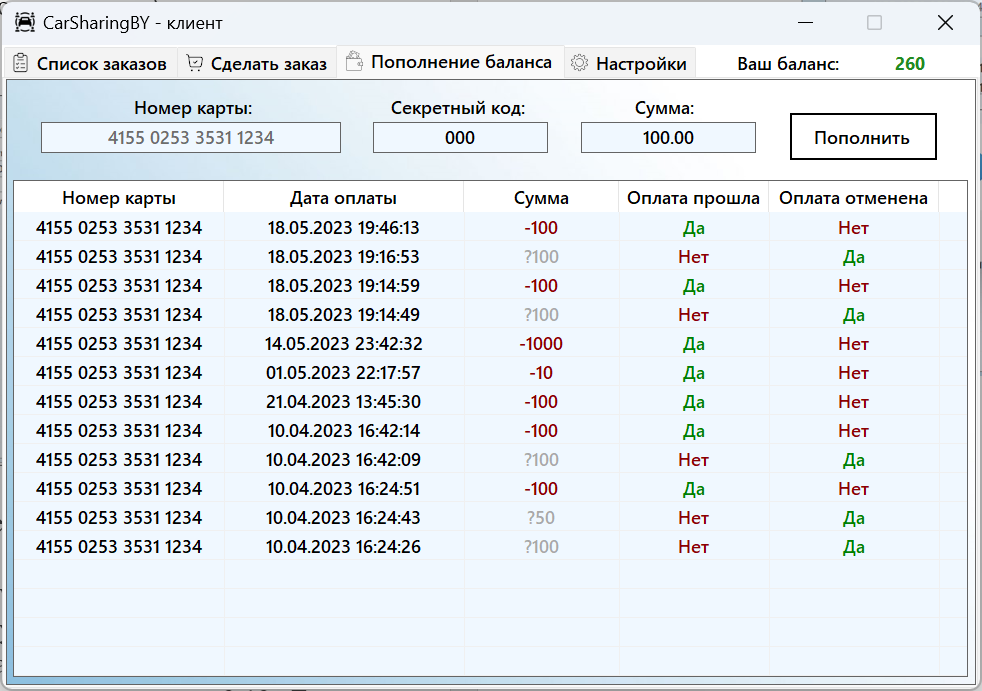


Рисунок 3.15 – Данные для оплаты верны

При переходе на влкадку «Настройки» пользователь видит следующее окно, иллюстрированное на рисунке 3.16. В этой вкладке пользователь может изменить необходимые для работы системы настройки, сменить пароль или деактивировать аккаунт, а так же посмотреть статус своего аккаунта: активирован он или нет.

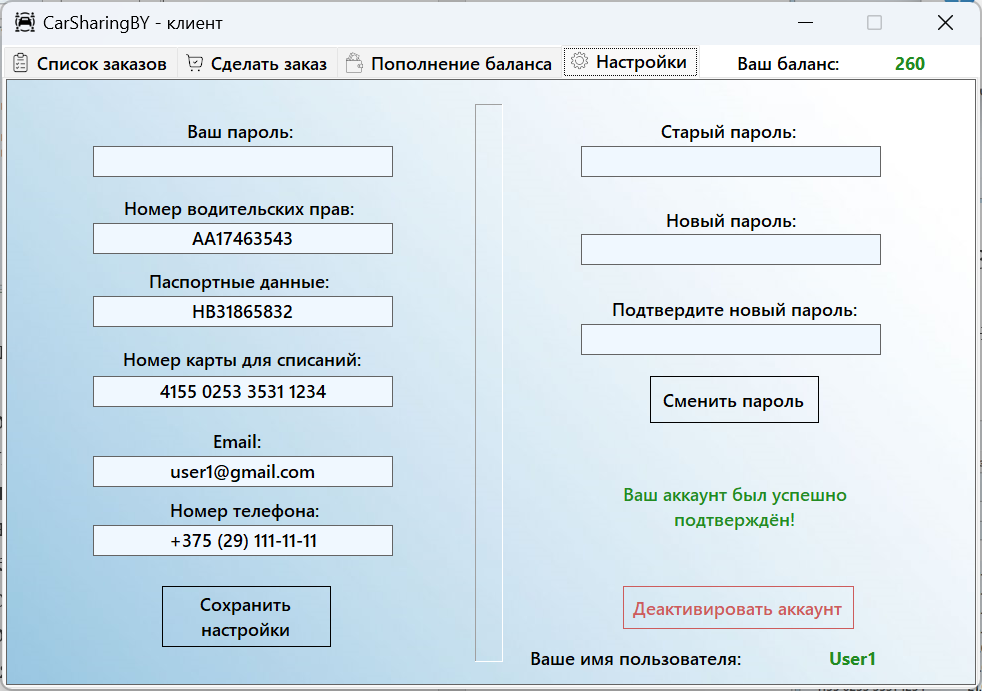


Рисунок 3.16 – Вкладка «Настройки»

Если аккаунт не активирован (что происходит сразу после регистрации), то пользователь не может получить полноценный доступ к системе. Ему будет доступна только вкладка «Настройки». При попытке перехода на другую вкладку, пользователь видит сообщение, изображённое на рисунке 3.17. После подтверждения всех настроек статус пользователя меняется на «Аккаунт подтверждён», что видно по зелёной надписи во вкладке «Настройки», и он может воспользоваться всеми остальными доступными функциями системы. Для изменения настроек пользователю требуется ввести свой текущий пароль из соображений безопасности.

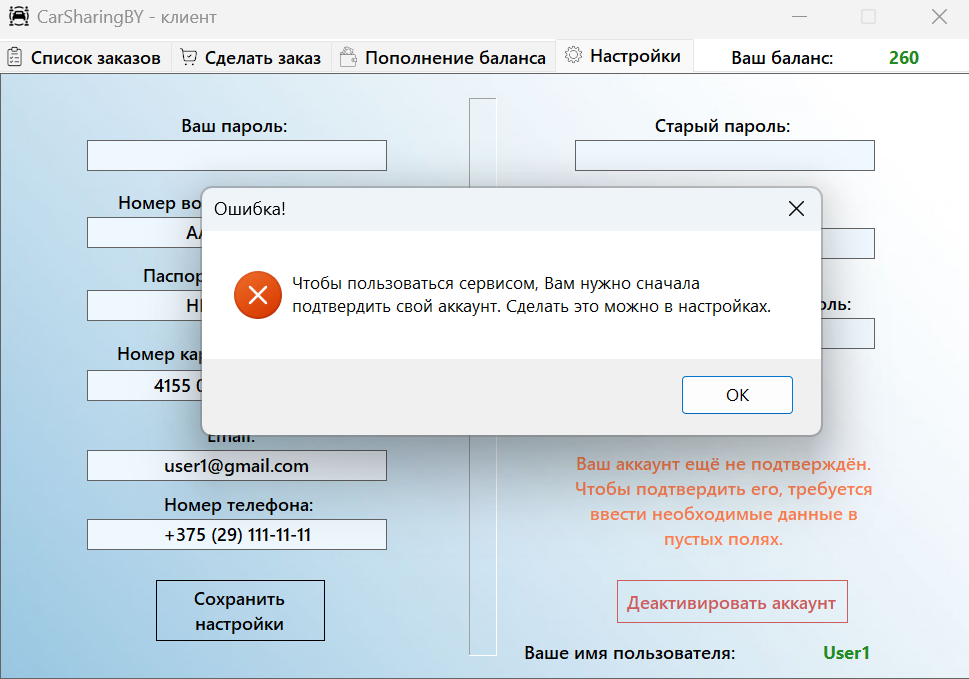


Рисунок 3.17 – Сообщение при неактивированном аккаунте

При успешной активации аккаунта пользователь увидит соответствующее сообщение, показанное на рисунке 3.18.

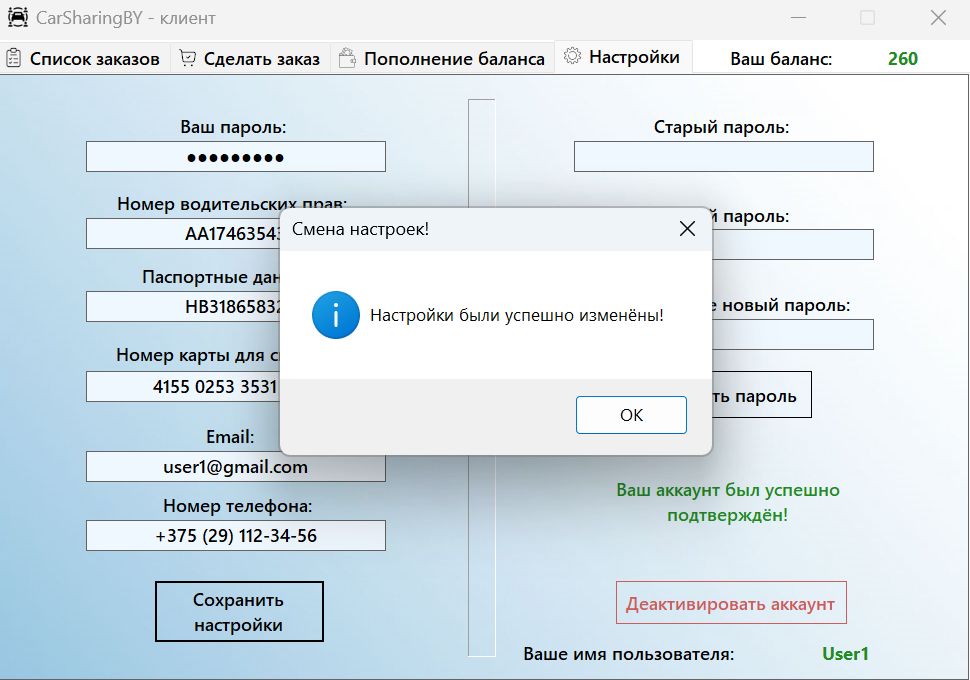


Рисунок 3.18 – Сообщение при успешной активации аккаунта

Кнопки «Сохранить настройки» и «Сменить пароль» проверяют данную пользователем информацию и исполняют свои функции в соответствии с названием. Об успешном или неуспешном выполнении программа сообщает пользователю в виде сообщений. Для их работы пользователю требуется ввести свой текущий пароль в рамках безопасности.

Кнопка «Деактивировать аккаунт» позволяет пользователю полностью отключить аккаунт, что сравнимо с его полным удалением из системы. После деактивации аккаунта пользователь больше не сможет самостоятельно вернуться в систему, поэтому эта кнопка запрашивает подтверждение у пользователя, уверен ли он в своих действиях, а так же запрашивает текущий пароль от аккаунта.

Если пользователь действительно захотел деактивировать свой аккаунт, то система просит ввести пароль. В случае, если пароль был введён и он верный, система автоматически отменяет все заказы пользователя, которые можно сейчас отменить, деактивирует аккаунт, выводит соответствующее об этом сообщение на экран, после чего приложение перезапускается. Этот случай можно увидеть на рисунке 3.19.

В случае, если пользователь вводит не те данные, программа информирует его об этом и не даёт деактивировать аккаунт.

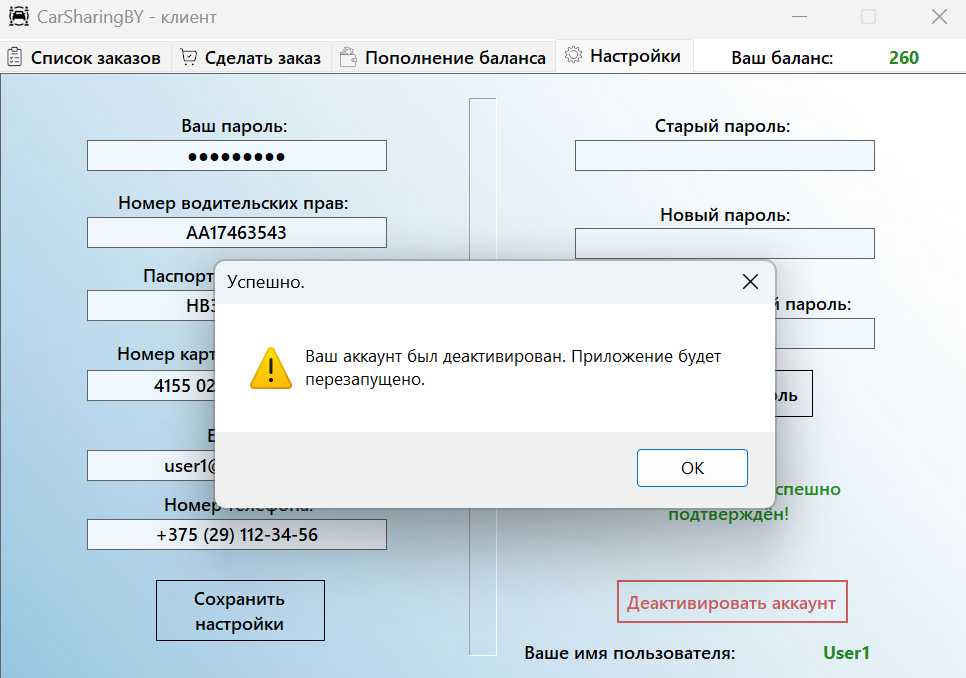


Рисунок 3.19 – Успешная деактивация аккаунта пользователем

Это были все функции пользователя с ролью «клиент». Рассмотрим пользователя с ролью «Сотрудник».

После авторизации под учётной записью с ролью сотрудника, пользователя встречает следующее окно, показанное на рисунке 3.20, с активной вкладкой «Сервисные отчёты».

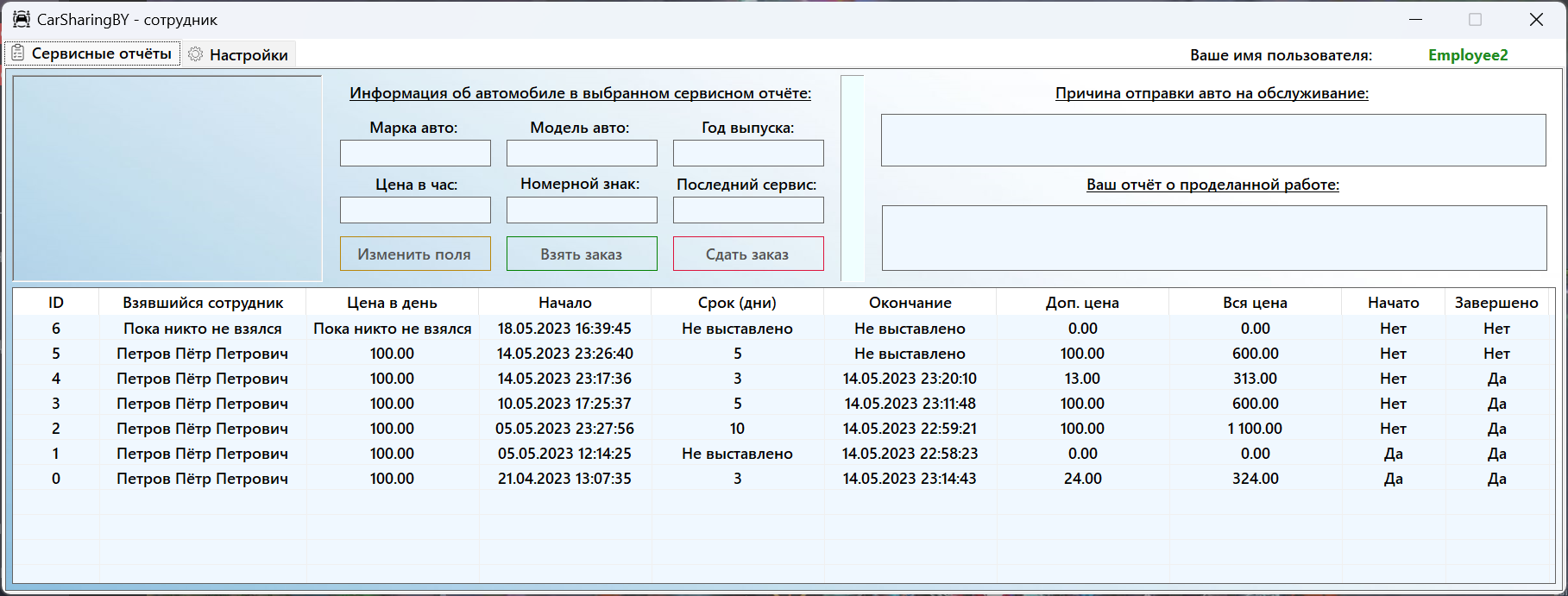


Рисунок 3.20 – Вкладка «Сервисные отчёты» сотрудника

Сотрудник может просмотреть историю своих отчётов обслуживания, а так же список свободных отчётов, нажав на любой из них в списке, как и просмотреть информацию об автомобиле, как показано на рисунке 3.21, а так же причину отправки автомобиля на обслуживание, и свой отчёт о проделанной работе, если таковой имеется.

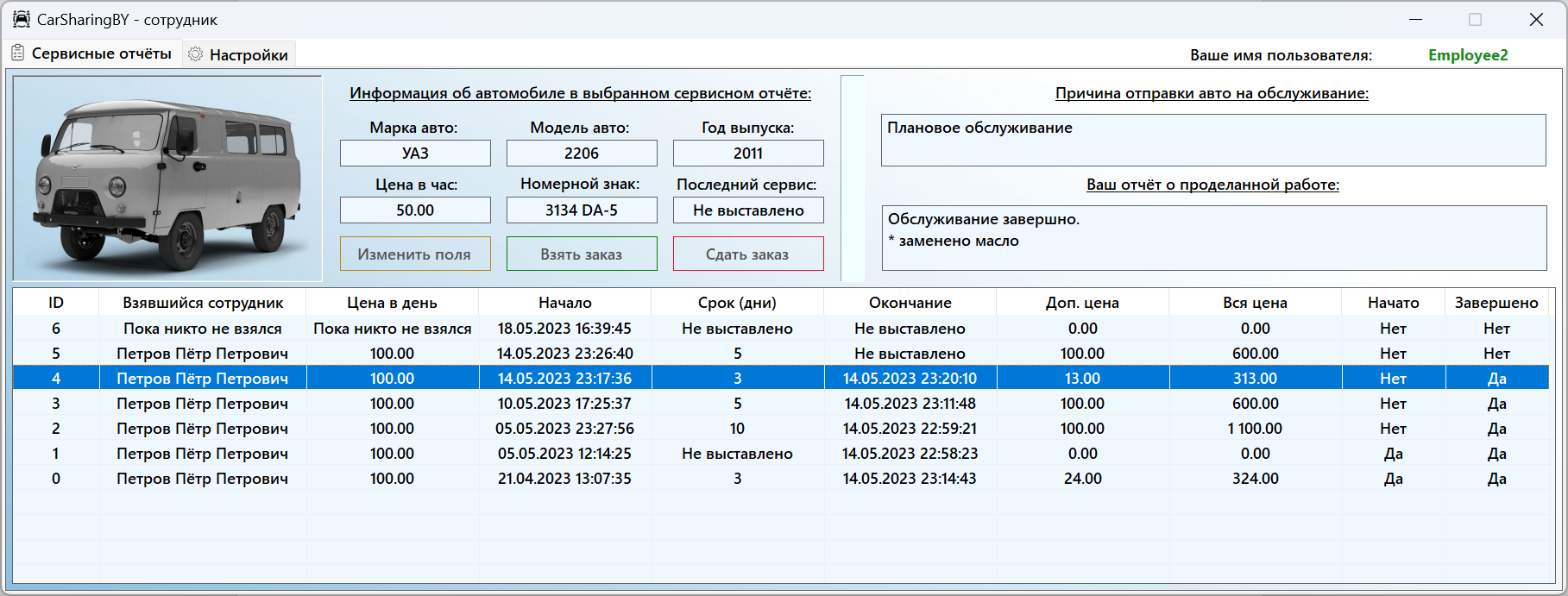


Рисунок 3.21 – Сотрудник выбрал сервисный отчёт

При нажатии кнопки «Изменить поля», если она доступна, сотруднику предлагается изменить два значения – запланированный срок выполнения задачи (в днях) и дополнительную цену обслуживания (например, на дополнительные запчасти) – с помощью окон, представленных на рисунках 3.22 и 3.23.

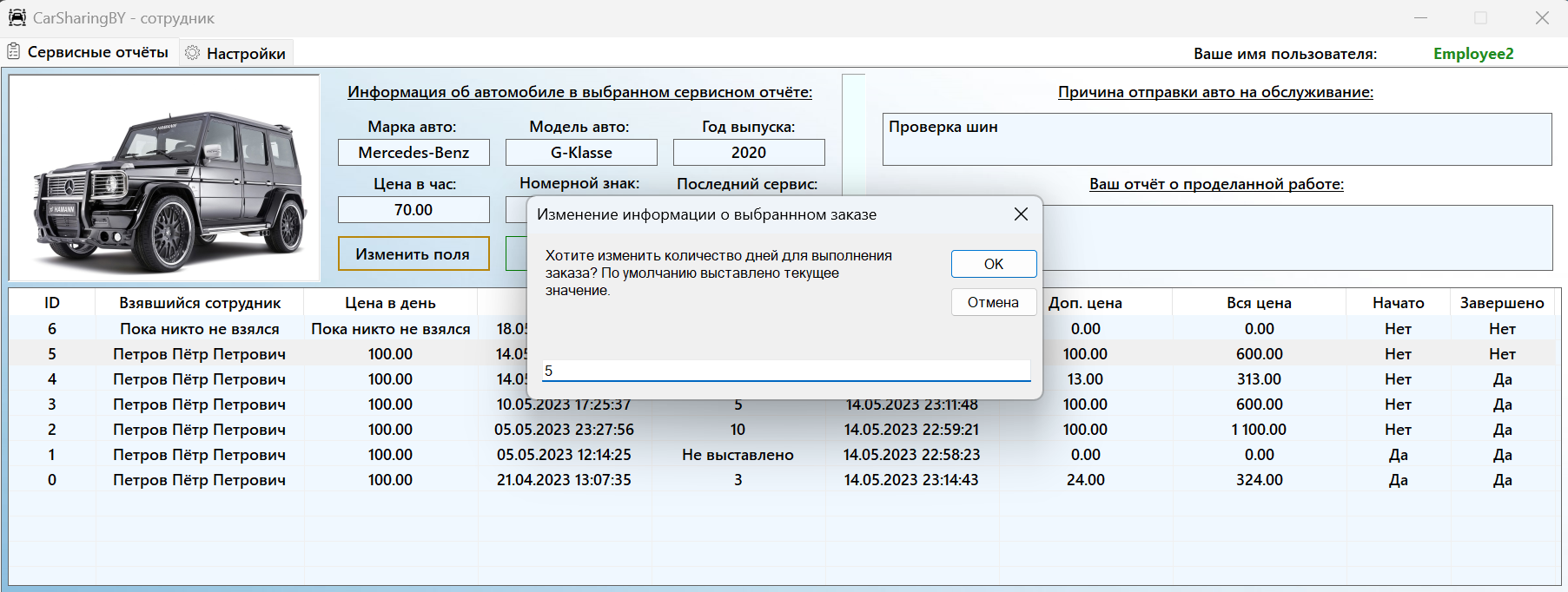


Рисунок 3.22 – Сотруднику предложено изменить количество дней

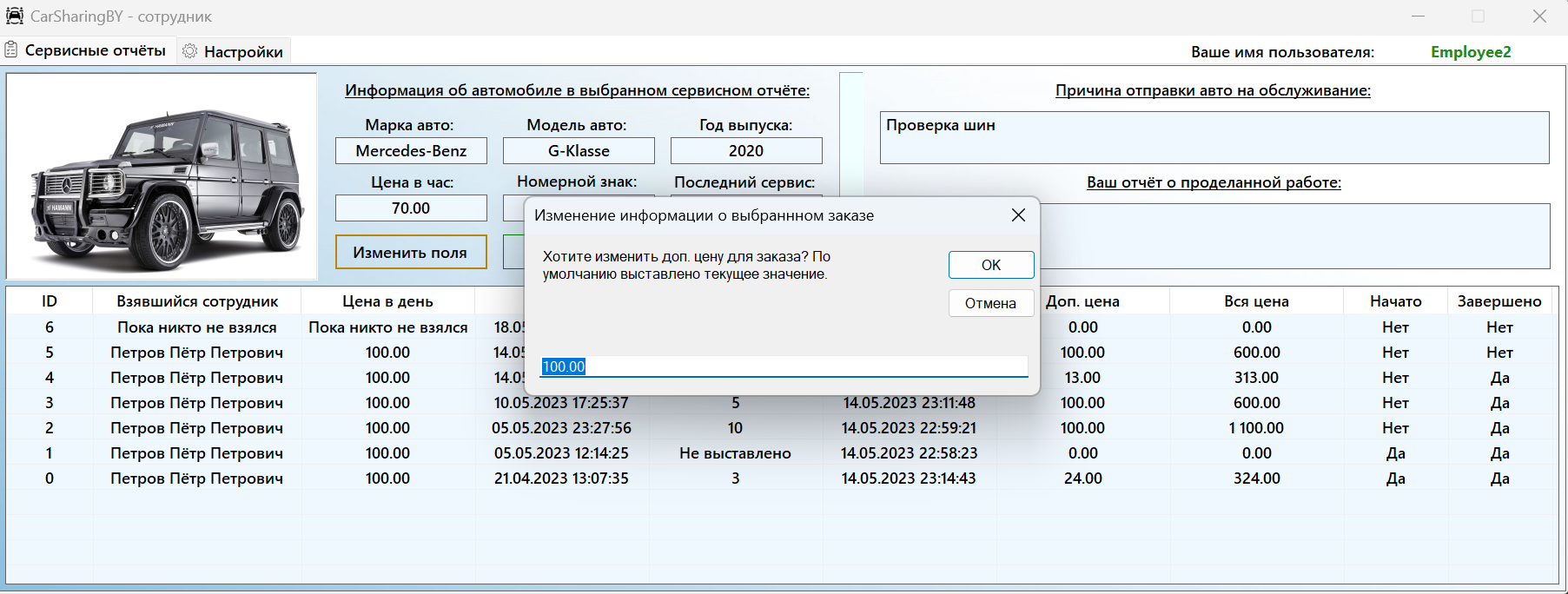


Рисунок 3.23 – Сотруднику предложено изменить дополнительную цену

При нажатии кнопки «Взять заказ» программа уточняет у сотрудника, уверен ли он в своём выборе. Если ответ утвердительный, то программа обновляет информацию о сервисном отчёте, указывая, какой именно сотрудник взялся за обслуживание. Статус отчёта обслуживания меняется, теперь другим сотрудникам он не показывается, а показывается только тому сотруднику, который его взял в работу. Сотруднику показывается соответствующее уведомление, представленное на рисунке 3.24. После этого список сервисных отчётов обновляется до актуального значения.

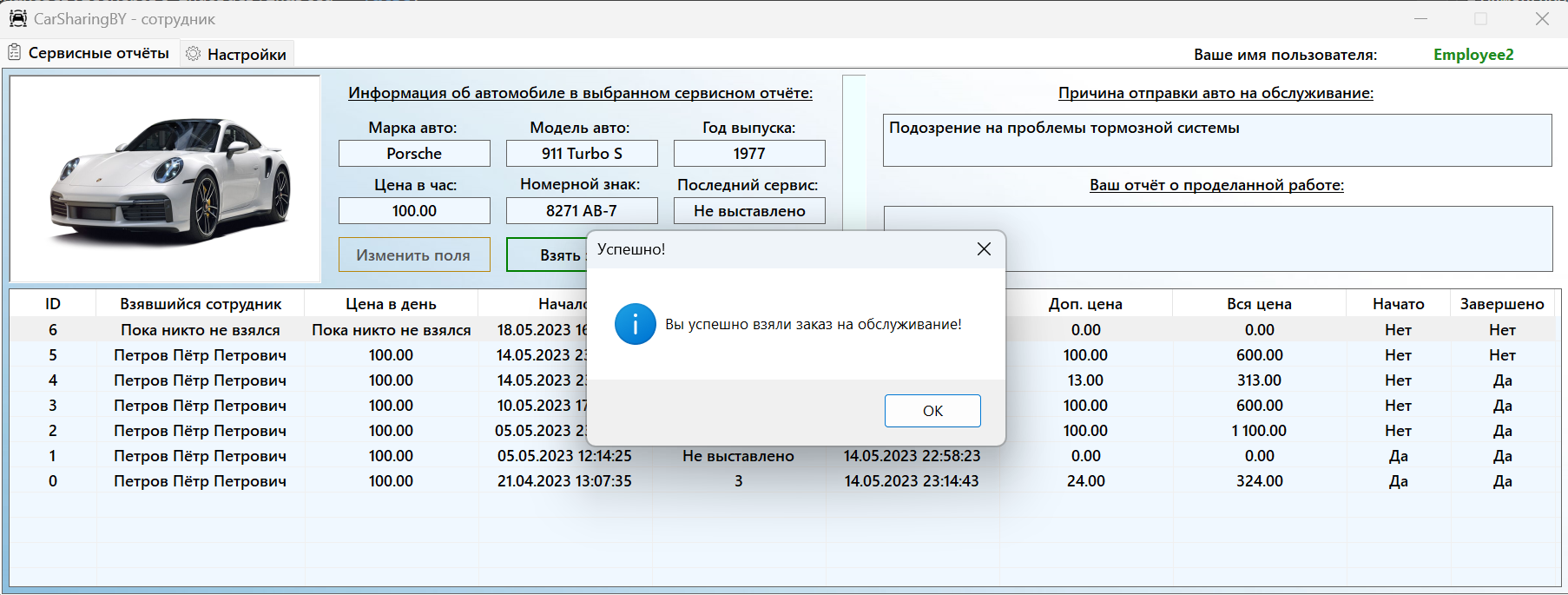


Рисунок 3.24 – Сотруднику успешно взял заказ в работу

При нажатии кнопки «Сдать заказ» программа уточняет у сотрудника, уверен ли он в своём выборе. Если ответ утвердительный, то программа проверяет, заполнен ли отчёт сотрудника о проделанной работе. Если отчёт не был заполнен, то программа выводит об этом соответствующее сообщение, указанное на рисунке 3.25.

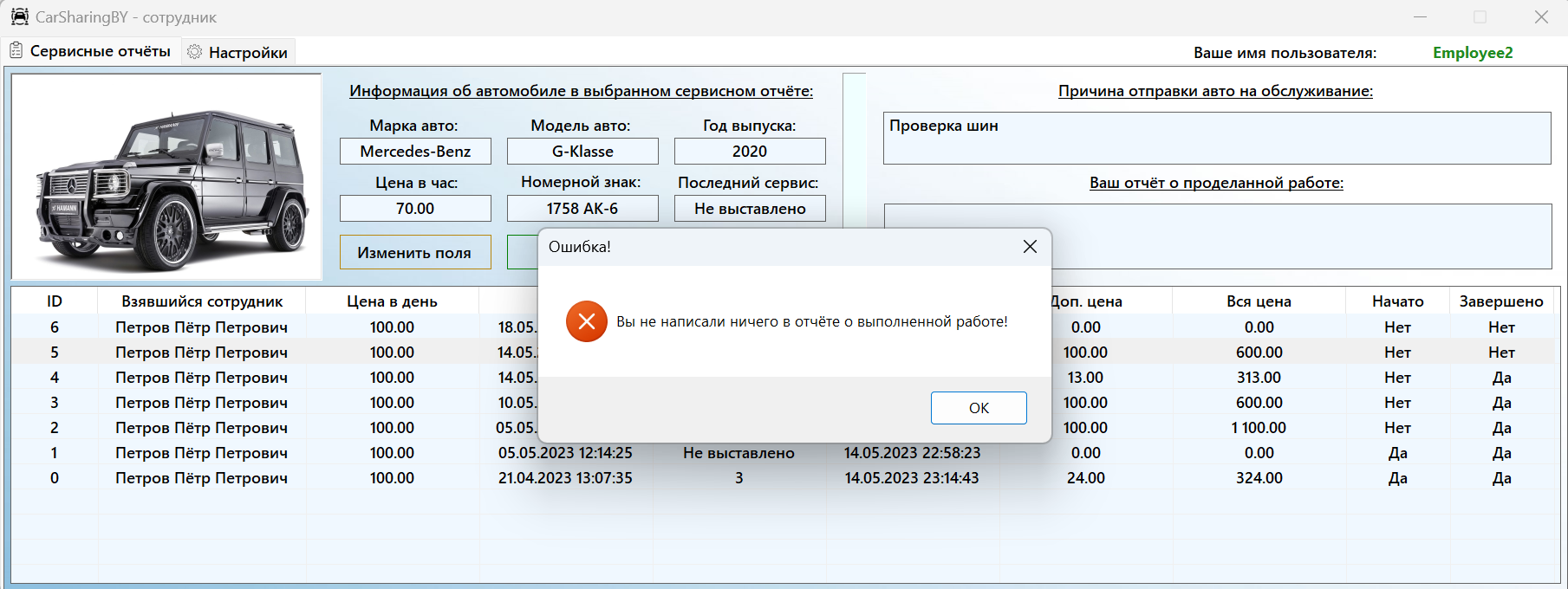


Рисунок 3.25 – Сотрудник ничего не написал в отчёте о работе

После написания отчёта о проделанной работе, заказ успешно засчитывается как сданный. Сотрудник получает сообщение, указанное на рисунке 3.26. После этого программа автоматически отправляет банковскую транзакцию от лица компании на счёт партнёра-компании *СТО* на сумму, указанную в заказе как «Дополнительная сумма». А так же от лица компании рассчитывает и выплачивает зарплату сотруднику в соответствии с количеством отработанных дней, и обновляет число всей выплаченной зарплаты сотруднику за всё время.



Рисунок 3.26 – Сотрудник успешно сдал заказ на обслуживание

При переходе на вкладку «Настройки» сотруднику предоставляется окно, иллюстрированное на рисунке 3.27. Здесь пользователю представлена возможность смены пароля, смены номера карты или банковского счёта для выплат зарплаты, а так же там указано число всей выплаченной зарплаты сотруднику за всё время его работы.



Рисунок 3.27 – Сотрудник успешно сдал заказ на обслуживание

Кнопки «Сохранить настройки» и «Сменить пароль» проверяют данную пользователем информацию и исполняют свои функции в соответствии с названием. Об успешном или неуспешном выполнении программа сообщает пользователю в виде сообщений. Для их работы пользователю требуется ввести свой текущий пароль в рамках безопасности.

Так же в любой вкладке сотрудник может видеть своё имя пользователя, расположенное в виде надписи в верхнем правом углу.

Это были все функции пользователя с ролью «сотрудник». Рассмотрим пользователя с ролью «Администратор».

После авторизации под учётной записью с ролью администратора, пользователя встречает следующее окно, показанное на рисунке 3.28, с активной вкладкой «Сервисные отчёты».

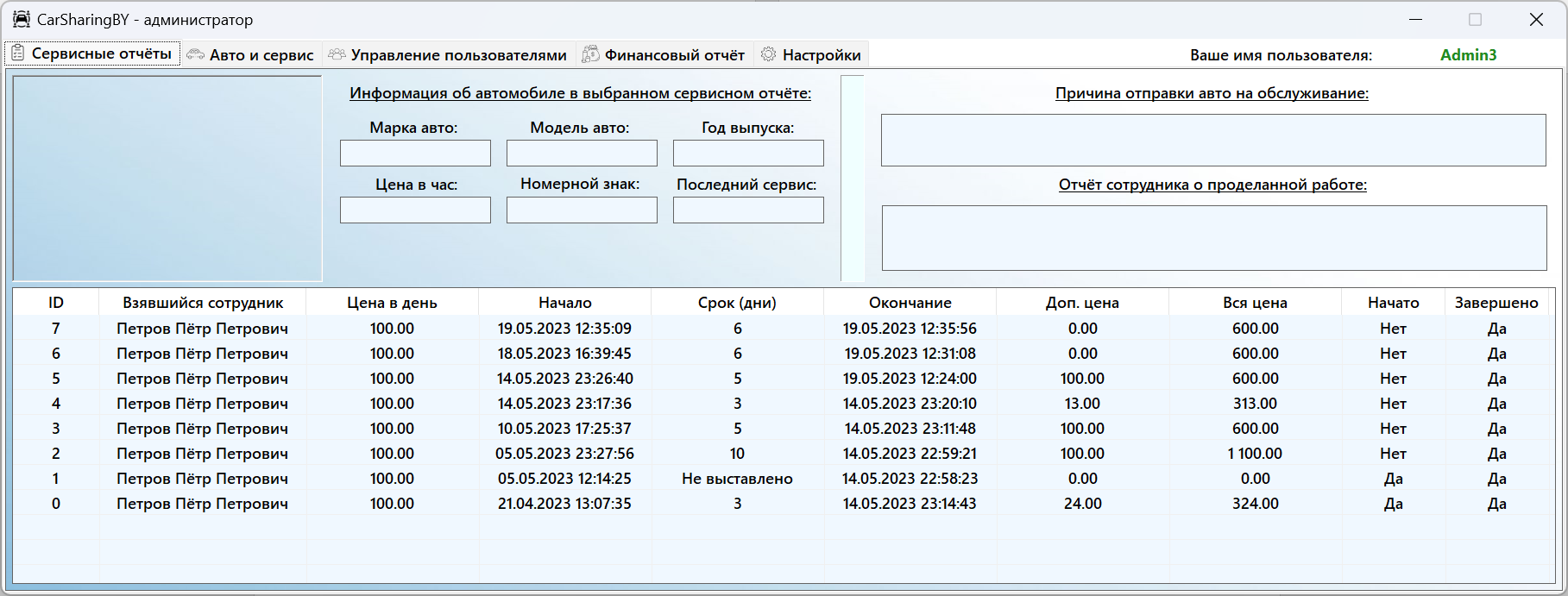


Рисунок 3.28 – Вкадка «Сервисные отчёты» администратора

Администратор может просмотреть историю всех отчётов обслуживания в системе, нажав на любой из них в списке, как и просмотреть информацию об автомобиле, как показано на рисунке 3.29, а так же причину отправки автомобиля на обслуживание, и отчёт сотрудника о проделанной работе, если таковой имеется.

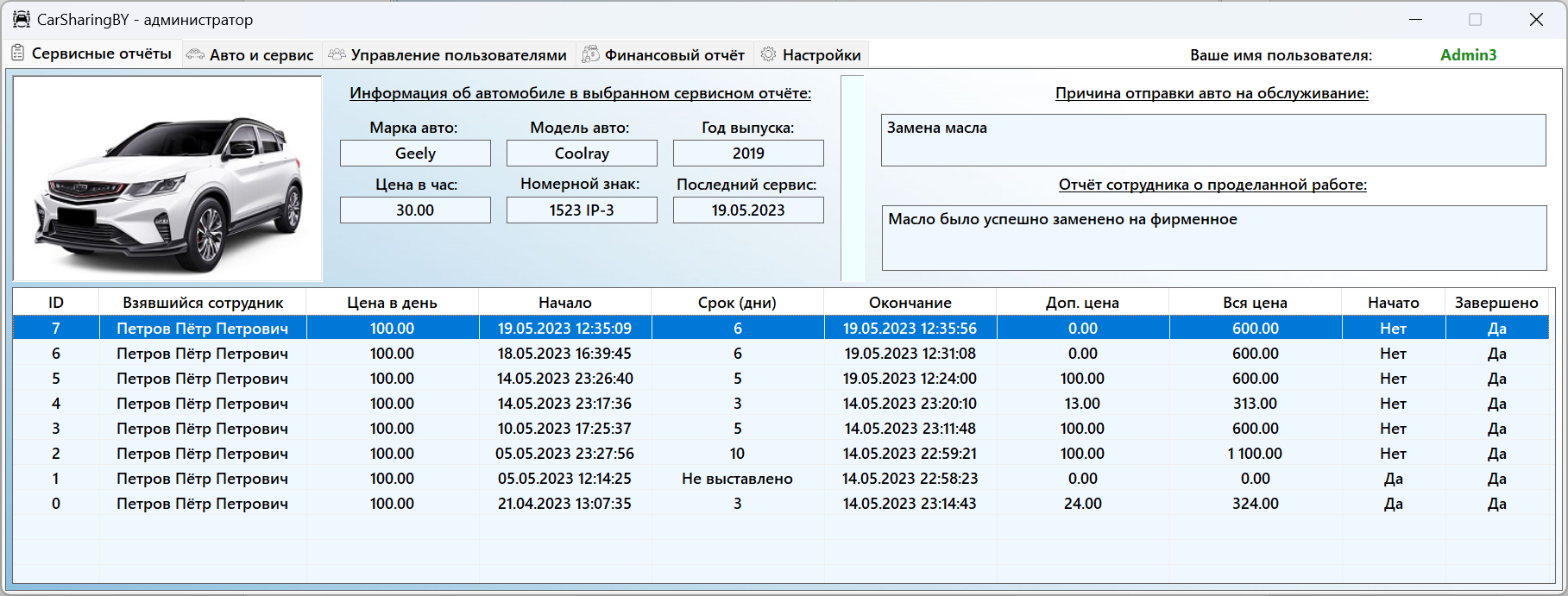


Рисунок 3.29 – Администратор выбрал сервисный отчёт

При переходе на вкладку «Авто и сервис» администратора встречает окно, представленно на рисунке 3.30. Здесь администратор может отправить автомобиль на обслуживание своим сотрудникам, указав причину обслуживания, а так же просмотреть информацию об автомобиле (что показано на рисунке 3.31), изменить её при желании (если администратор выберет режим работы «Изменить авто»), или добавить новый автомобиль в систему. Администратор может скрыть автомобиль из системы, чтобы пользователи не могли им пользоваться.

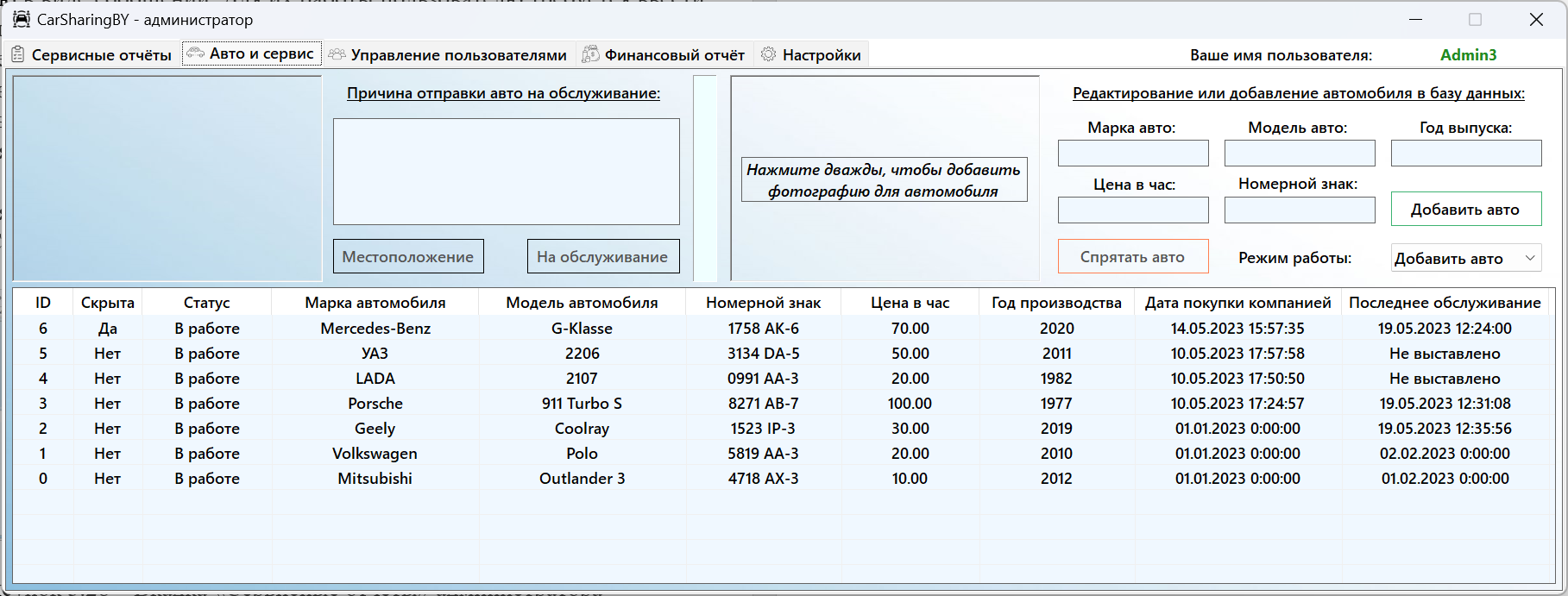


Рисунок 3.30 – Вкадка «Авто и сервис» администратора

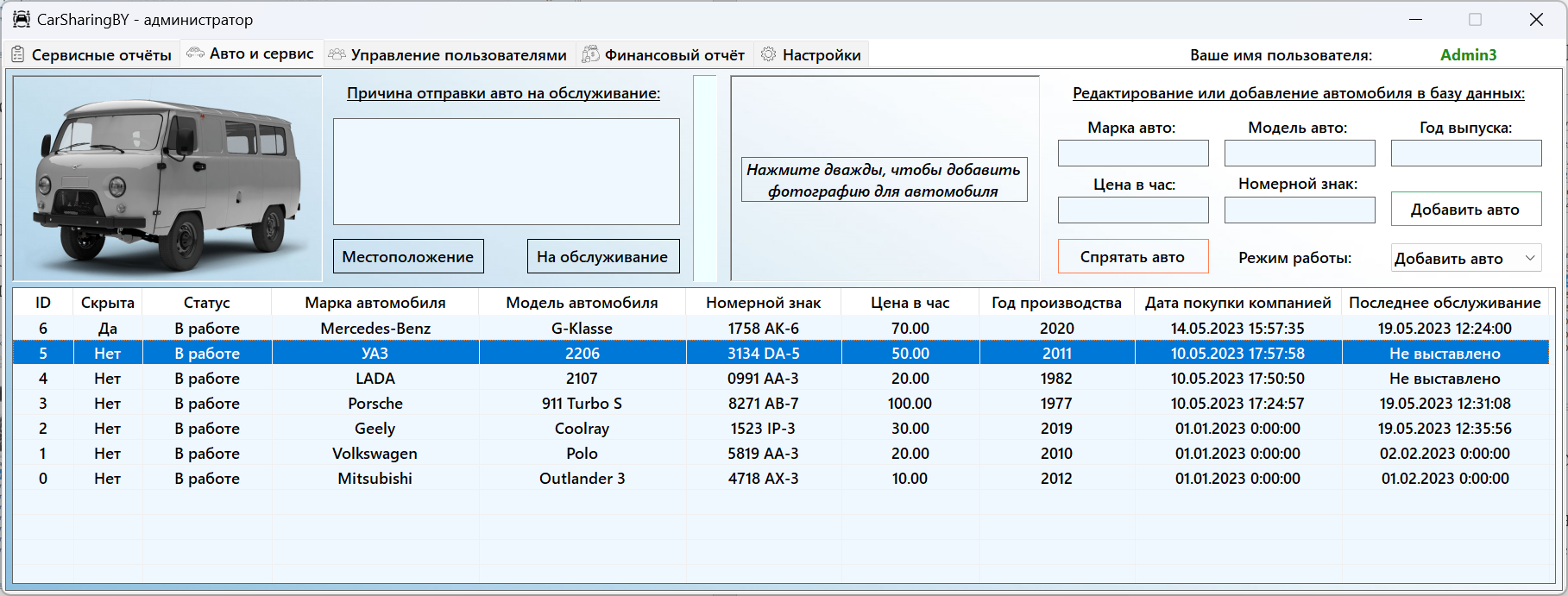


Рисунок 3.31 – Администратор выбрал автомобиль в режиме «Добавить авто»

При нажатии кнопки «Местоположение» администратор может перейти на сторонний сайт «*Google Maps*» для просмотра точки местоположения автомобиля в данный момент, если даст утвердительный ответ на уточнение программы после нажатия кнопки.

При нажатии кнопки «На обслуживание» программа проверяет, указал ли администратор причину отправки автомобиля на обслуживание, и не находится ли автомобиль сейчас в работе. Если всё верно, то программа уточняет, уверен ли администратор в своём выборе. Если администратор уверен, то автомобиль отправляется на обслуживание. Теперь сотрудники видят его в своих аккаунтах, а так же автомобиль становится недоступным для заказа клиентам, получая соответствующую отметку на своей фотографии.

При изменении режима работы на «Добавление авто», элементы в правой верхней части экрана становятся пустыми, позволяя администратору указать информацию для добавления автомобиля в систему. Администратор так же может нажать дважды на пустое место для фотографии, чтобы выбрать её для автомобиля и загрузить в систему. После нажатия кнопки «Добавить авто» программа проверяет все введённые данные, и если всё верно и администратор уверен в своём выборе, то автомобиль добавляется в систему, но с флагом «Скрыт», тем самым, он не появляется сразу в системе у клиентов, и даёт право администратору на исправление какой-либо ошибки, например, опечатки в номере автомобиля.

При изменении режима работы на «Изменение авто», элементы правой верхней части экрана автоматически будут заполняться информацией о выбранном в данный момент автомобиле (рисунок 3.32). Эту информацию пользователь может изменить так же, как и в случае с добавлением автомобиля. После нажатия кнопки «Изменить авто» программа проверяет все введённые данные, и если всё верно и администратор уверен в своём выборе, то система изменяет данные автомобиля и информирует администратора об этом.

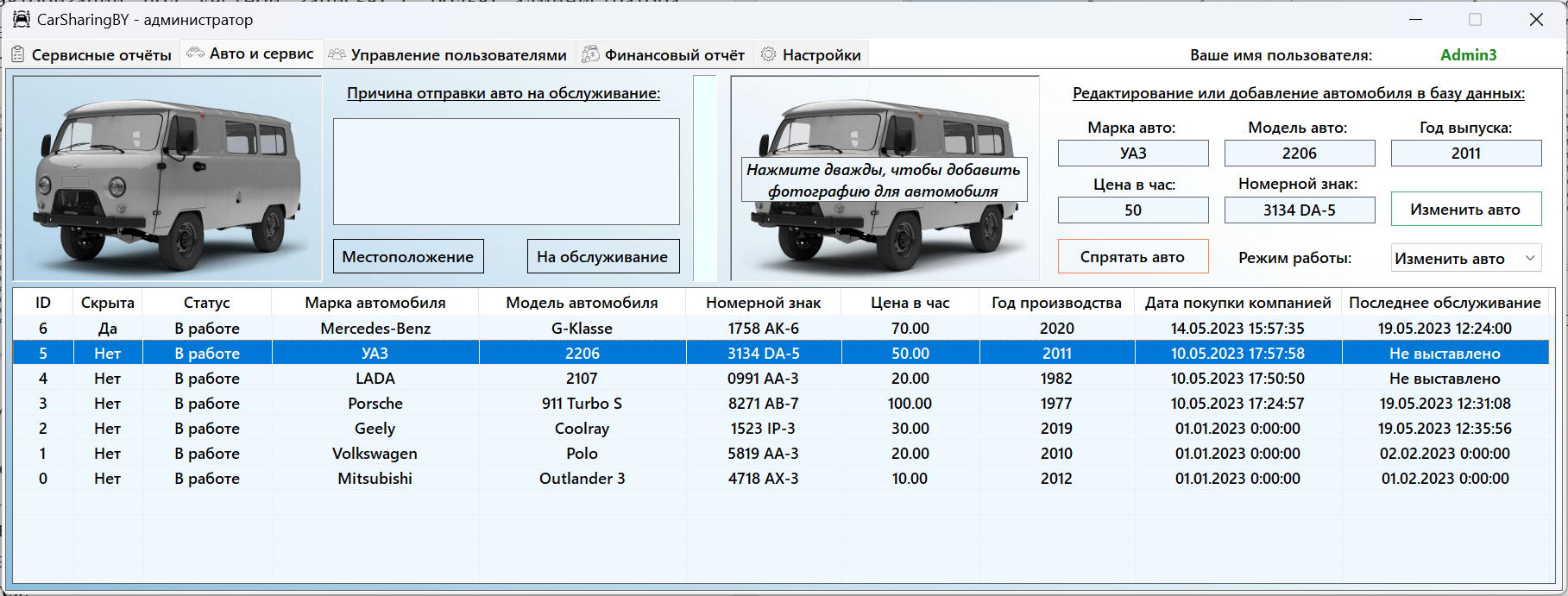


Рисунок 3.32 – Администратор выбрал автомобиль в режиме «Изменить авто»

При нажатии кнопки «Спрятать авто» программа уточняет у пользователя, уверен ли он в своём выборе. Если ответ утвердительный, то автомобиль помечается специальным флагом «Скрыт», после чего система перестаёт показывать этот автомобиль клиентам как возможного для заказа. Данную кнопку можно нажать только тогда, когда автомобиль в данный момент не имеет флага «Скрыт», что так же можно увидеть на рисунке 3.32. После нажатия кнопки программа обновляет информацию на экране и показывает пользователю соответствующие сообщения.

При нажатии кнопки «Показать авто» программа уточняет у пользователя, уверен ли он в своём выборе. Если ответ утвердительный, то система удаляет у автомобиля специальный флаг «Скрыт», после чего система снова будет показывать этот автомобиль клиентам как возможный для заказа. Данную кнопку можно нажать только тогда, когда автомобиль в данный момент имеет флаг «Скрыт», что можно видеть на рисунке 3.33. После нажатия кнопки программа обновляет информацию на экране и показывает пользователю соответствующие сообщения.

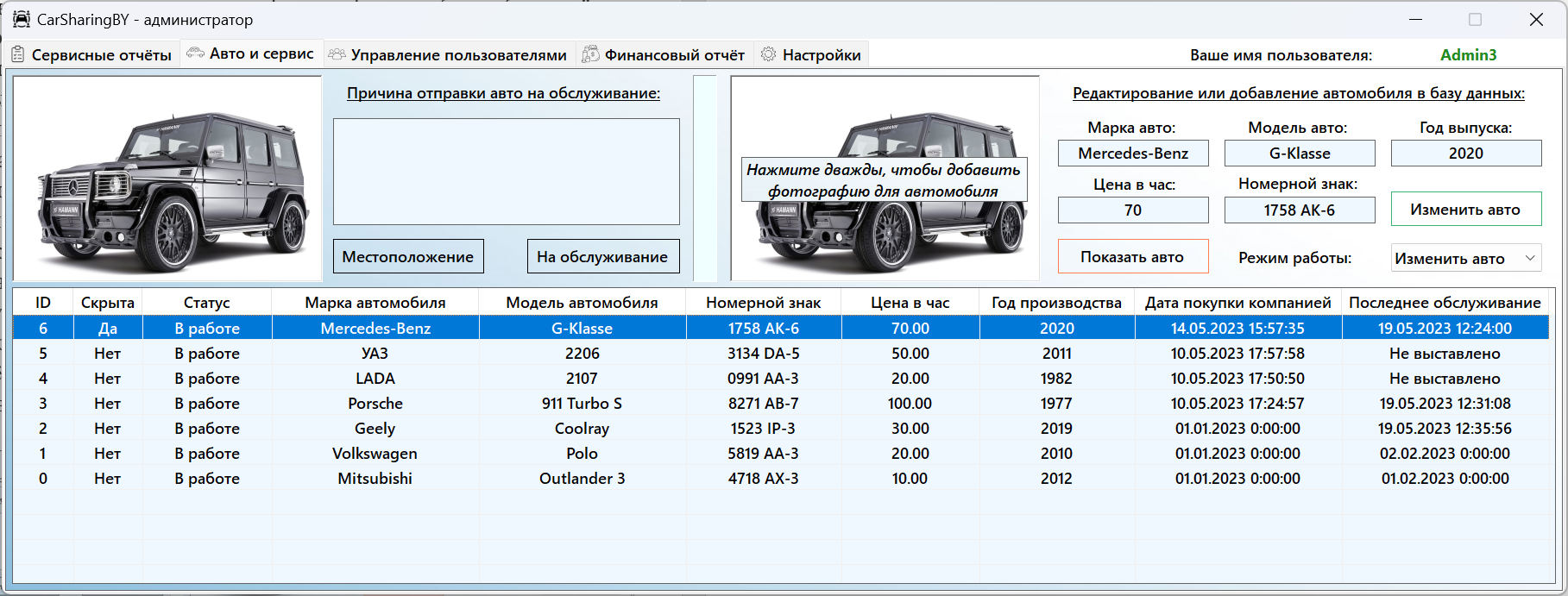


Рисунок 3.33 – Администратор выбрал уже скрытый в системе автомобиль

При переходе на вкладку «Управление пользователями», администратор видит окно, представленное на рисунке 3.34. В этом окне администратор может присвоить клиенту роль сотрудника (то есть взять его на работу), и наоборот (если прошлый шаг был сделан по ошибке). Так же администратор может деактивировать или активировать аккаунт клиенту или сотруднику.

На вкладке возможен поиск по таблице. Программа проверяет каждое поле строки таблицы, и если найдено хотя бы одно совпадение, то весь ряд остаётся в таблице, в противном случае он скрывается (рисунок 3.37).

При нажатии на пользователя, в зависимости от его роли, администратору становятся доступны или недоступны различные кнопки, например, рисунки 3.34, 3.35 и 3.36 (клиент, сотрудник и администратор были выбраны соответственно).

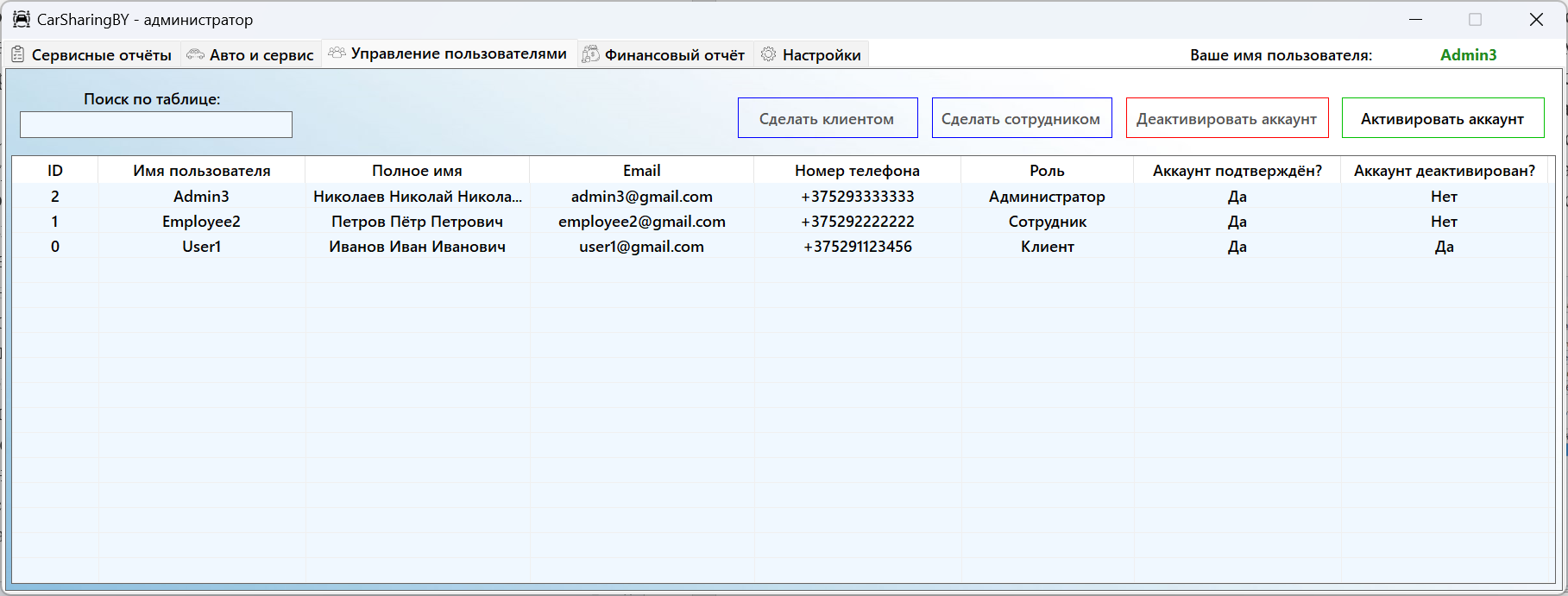


Рисунок 3.33 – Вкладка «Управление пользователями» администратора

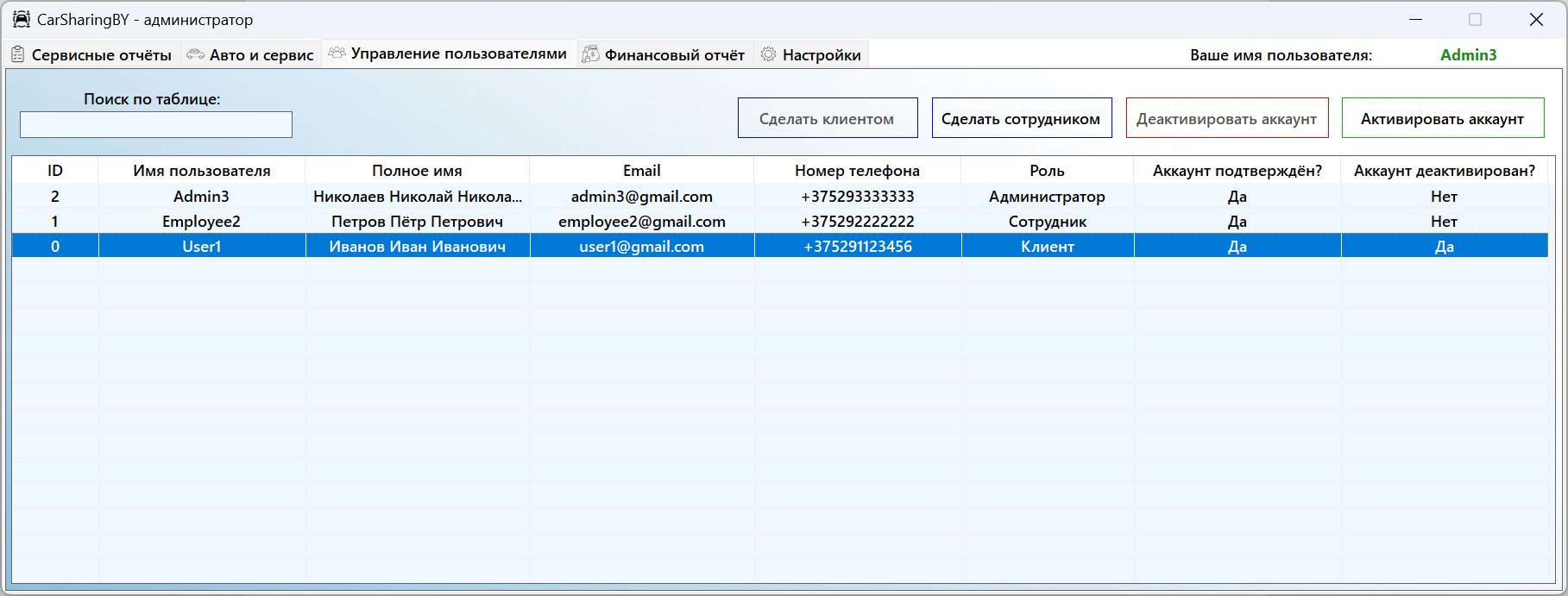


Рисунок 3.34 – Администратор выбрал клиента

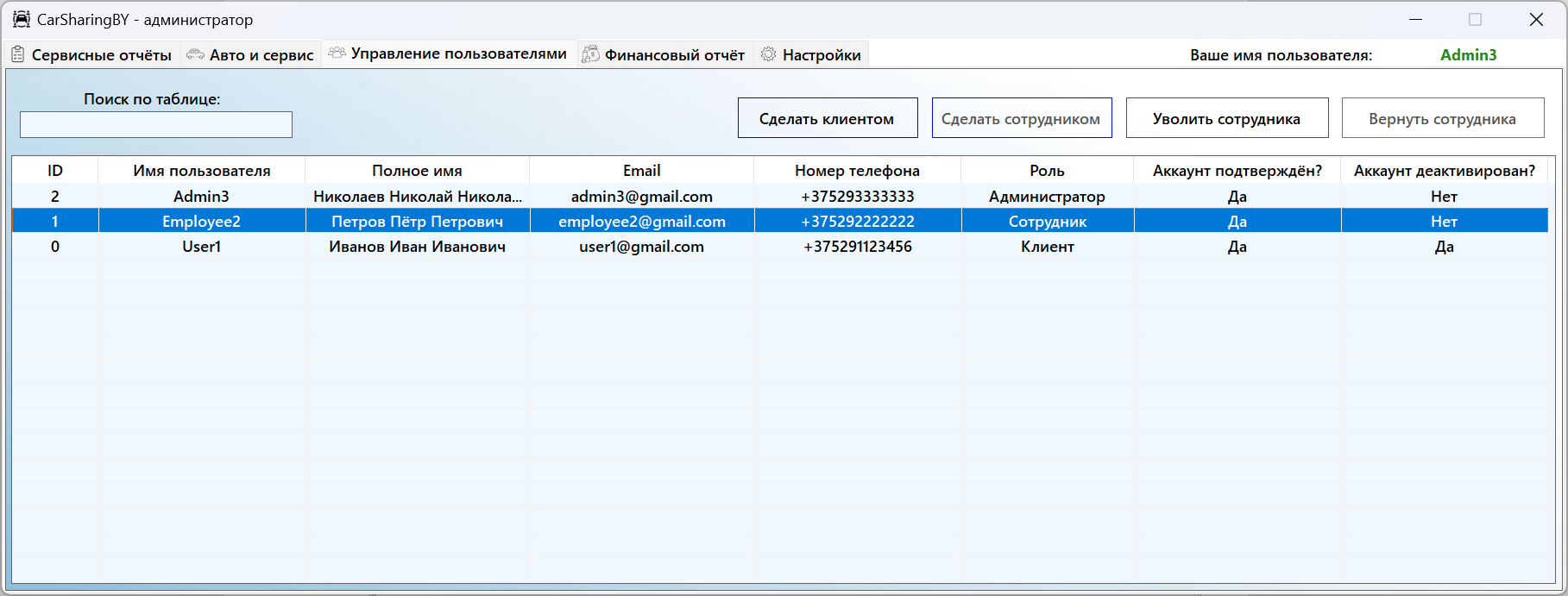


Рисунок 3.35 – Администратор выбрал сотрудника

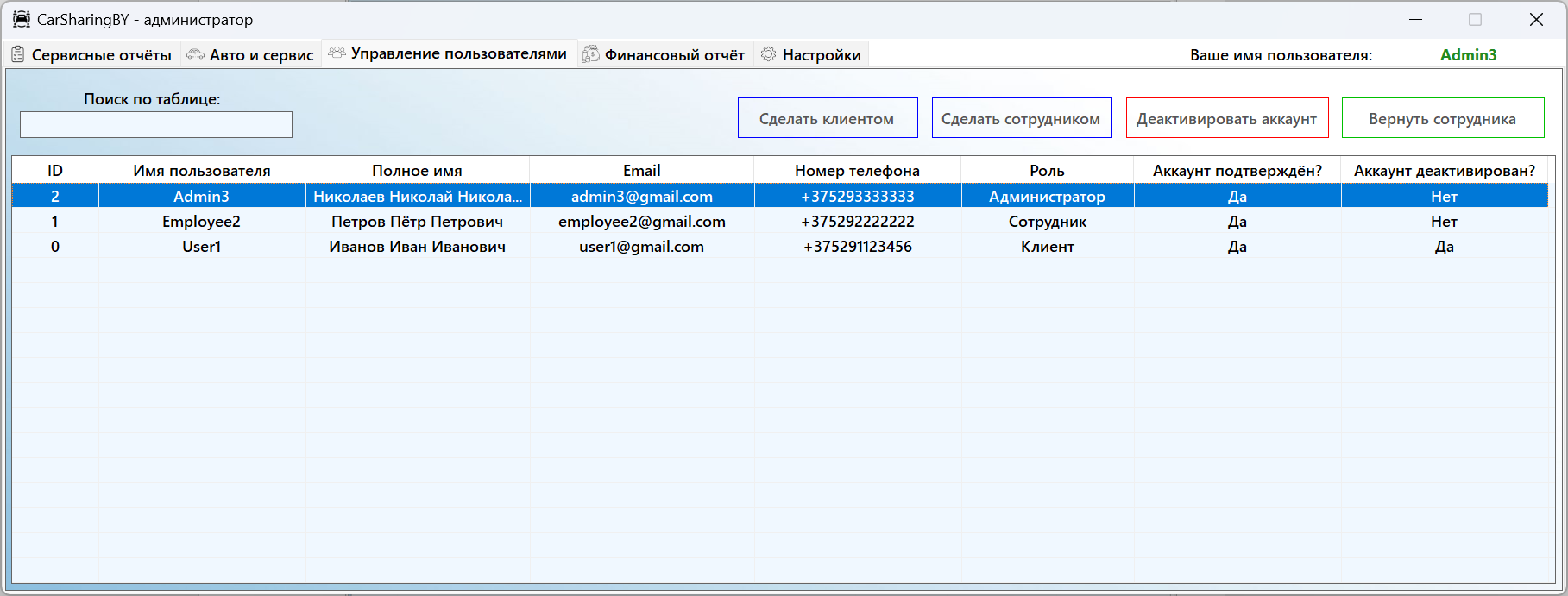


Рисунок 3.36 – Администратор выбрал администратор

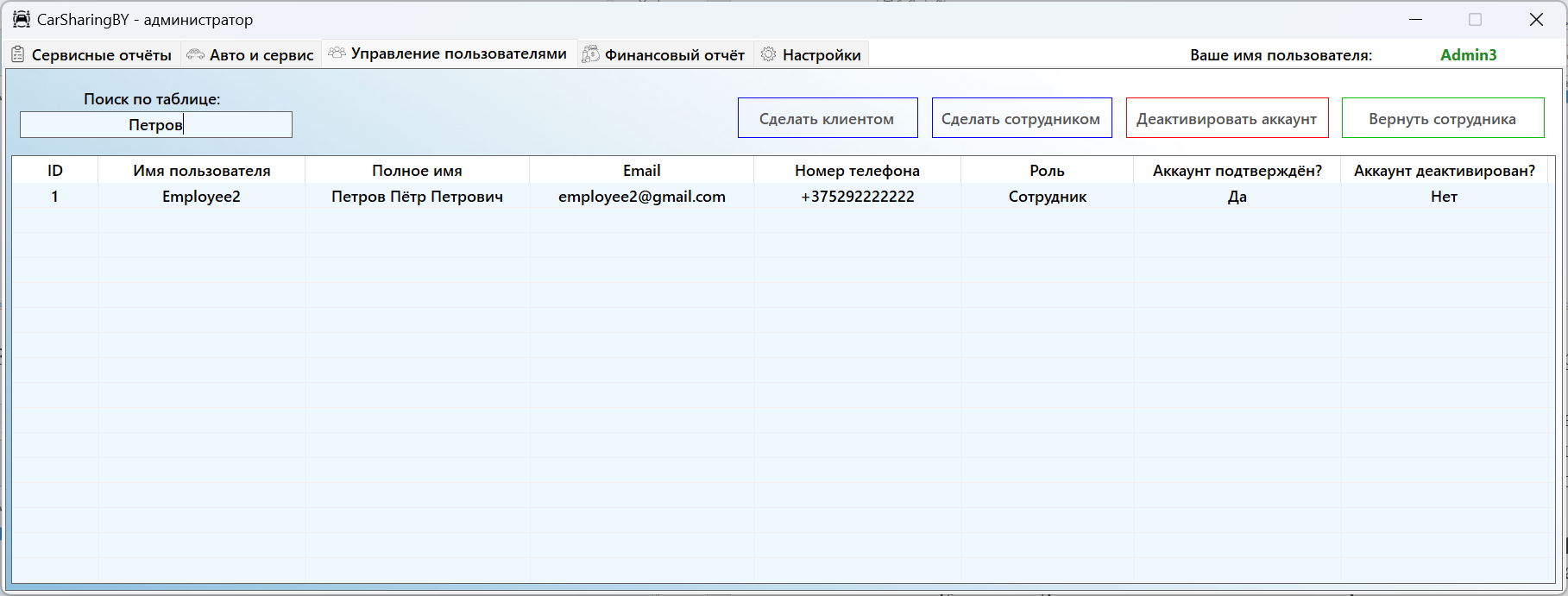


Рисунок 3.37 – Поиск по таблице пользователей

Как можно заметить, при выборе пользователя с ролью «Клиент» становится активной кнопка «Сделать сотрудником», в то время как кнопка «Сделать клиентом» недоступна. При выборе пользователя с ролью «Сотрудник» кнопка «Сделать клиентом», наоборот в случае от клиента, становится активной. При выборе пользователя с ролью администратор ни одна кнопка недоступна, потому что администраторы имеют равные права в системе.

Кнопки «Деактивировать аккаунт» и «Активировать аккаунт» становятся доступными и недоступными в зависимости от статуса выбранного аккаунта – активирован он или нет. Если пользователь является сотрудником, то надписи меняются на «Уволить сотрудника» и «Вернуть сотрудника» соответственно.

При переходе на вкладку «Финансовый отчёт» пользователя встречает окно, изображённое на рисунке 3.38. На вкладке администратор может с помощью двух элементов выбрать начальную и конечную дату периода, за который он хочет получить финансовый отчёт.

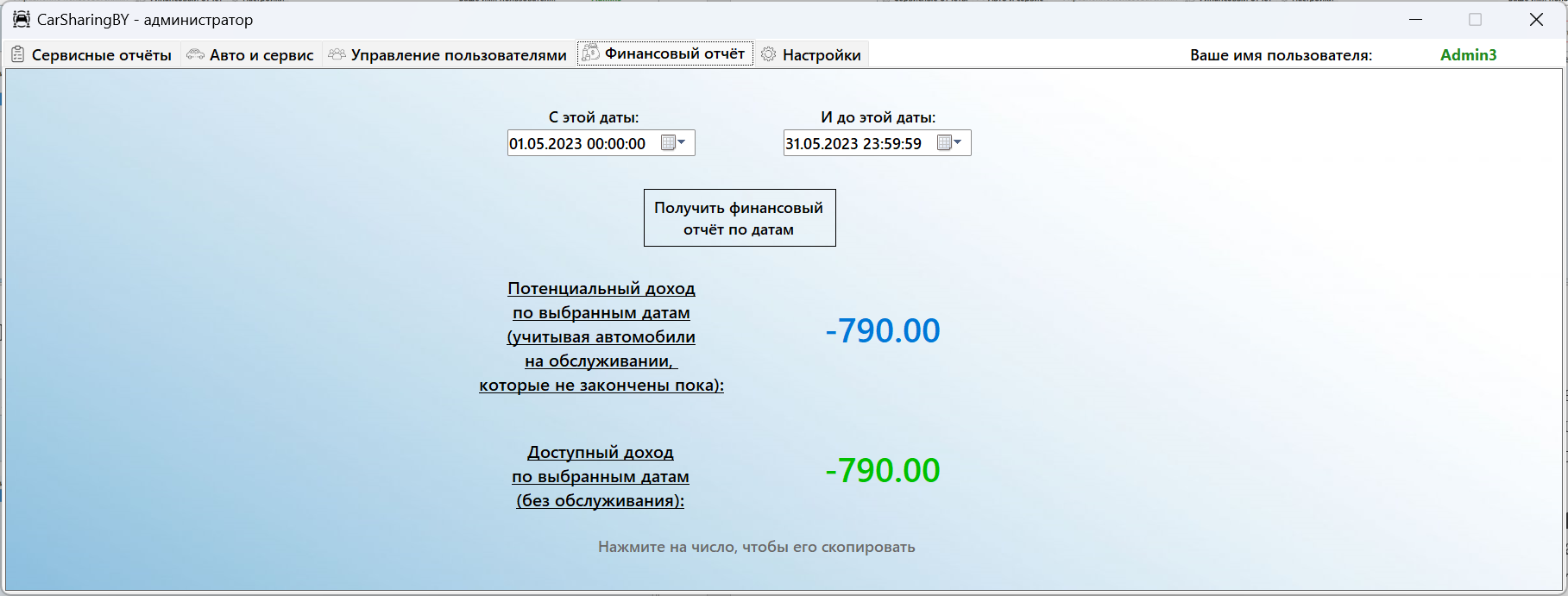


Рисунок 3.38 – Вкладка «Финансовый отчёт» администратора

При нажатии кнопки «Получить финансовый отчёт по датам» программа считает потенциальный доход и доступный доход проекта в рамках указанных дат, и отображает их в соответстувующих надписях в окне. При нажатии на число оно будет скопировано в буфер обмена.

По умолчанию в полях для дат стоят даты начала и окончания текущего месяца соответственно.

Доступный доход – это разность суммы дохода и суммы трат компанией за выбранный период. Потенциальный доход – это разность доступного дохода и всех цен на обслуживание автомобилей на данный момент.

При переходе на вкладку «Настройки» администратор видит окно, представленное на рисунке 3.39. Администратор может сменить свой пароль и просмотреть, сколько всего автомобилей он отправил на обслуживание за всё время своей работы. Кнопка «Сменить пароль» проверяет данную пользователем информацию и исполняет свою функцию в соответствии с названием. Об успешном или неуспешном выполнении программа сообщает пользователю в виде сообщений. Для работы этой функции пользователю требуется ввести свой текущий пароль в рамках безопасности.

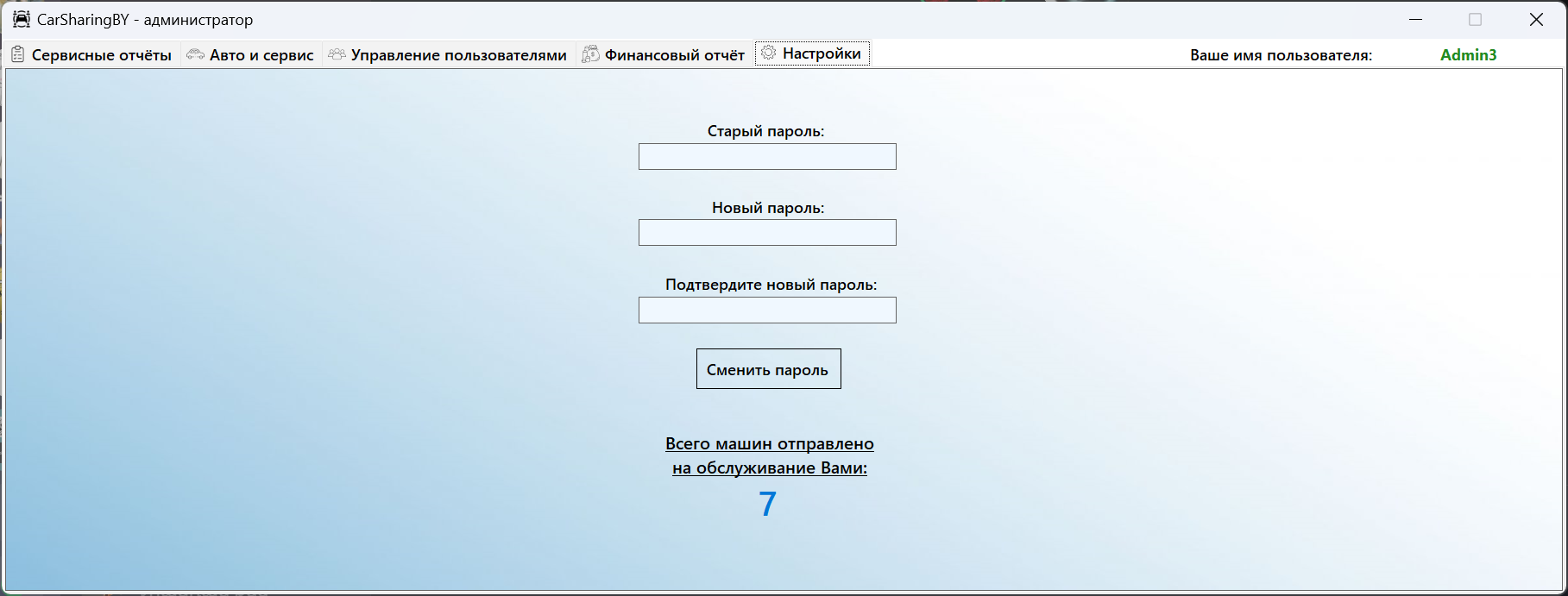


Рисунок 3.39 – Вкладка «Настройки» администратора

**4 ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ**

**4.1 Общее описание тестирования проекта**

Обеспечение стабильной и эффективной работы приложения каршеринга требует внимательного отношения к обработке исключительных ситуаций. Это включает в себя сценарии, когда пользователь вводит недопустимые данные, а также возможные ошибки в процессе выполнения операций. В ходе этого проекта было осуществлено рассмотрение и обработка ряда таких исключений. Подробности об этих обработанных исключительных ситуациях представлены в таблице 4.1.

Таблица 4.1 – Тестирование приложения

|  |  |
| --- | --- |
| **Исключительная ситуация** | **Результат** |
| Отправка пустого поля в случае, когда поле должно быть заполнено | Сообщение о том, что поле не может быть пустым |
| Попытка ввода значений не подходящего формата | Сообщение о неправильном формате ввода |
| Ошибка ввода в поле, т.е. ввод неподходящего параметра | Сообщение о недопустимости такого ввода |
| Попытка регистрации в системе с уже существующем паролем | Сообщение о том, что данный пароль уже существует |
| Вход с несуществующим логином или паролем | Вывод сообщения о том, что следует проверить корректность вводимых данных |

В рамках тестирования приложения были произведены проверочные вводы неверных значений. Например, был проведен тест на обработку пустых или неправильных значений в форме авторизации. При подобном вводе программа генерирует и выводит соответствующее сообщение об ошибке. Результат этого теста можно увидеть на рисунке 4.1.

Также была проведена проверка на обработку ситуации, когда при регистрации нового пользователя вводится уже существующий логин или некорректный пароль. В этом случае, система корректно отклоняет попытку регистрации и выводит информативное сообщение, показывающее ошибку. Результат этого теста демонстрируется на рисунке 4.2.

Можно также провести проверку на ввод некорректных значений при бронировании автомобиля, например, ввод невозможных дат и времени. В этом случае программа должна отклонять недопустимый ввод и выводить соответствующее сообщение об ошибке.

Еще одним примером тестового сценария может быть ситуация, когда пользователь пытается продлить бронирование автомобиля, не имея достаточного баланса. В этом случае приложение должно корректно обрабатывать эту ситуацию и информировать пользователя о недостатке средств для выполнения операции (рисунок 4.3).

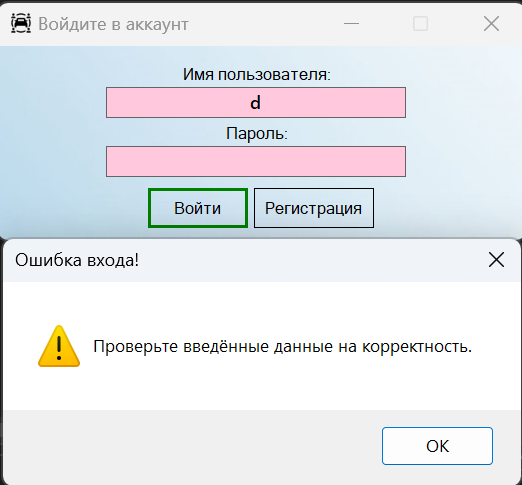


Рисунок 4.1 – Проверка на ошибки в окне авторизации

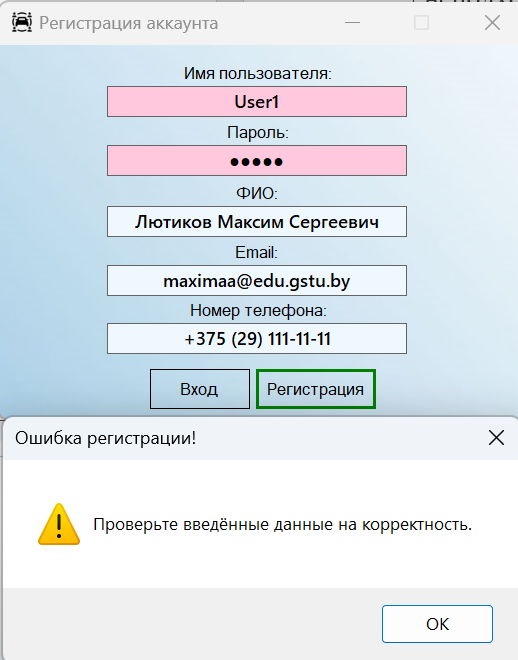


Рисунок 4.2 – Проверка на ошибки в окне регистрации

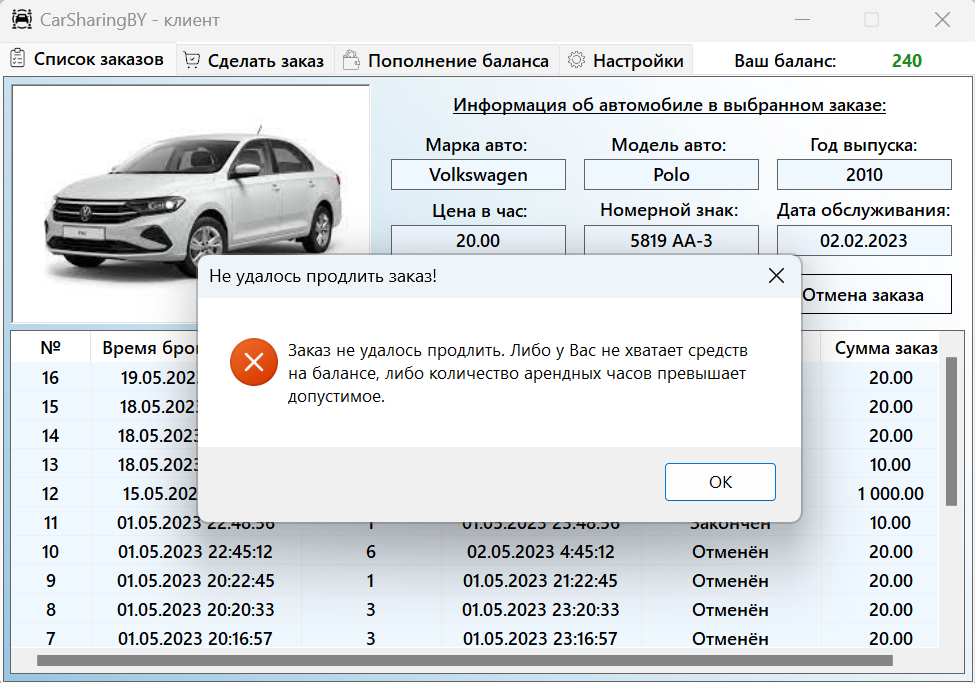


Рисунок 4.3 – Проверка на ошибки в окне авторизации

Другим примером тестового сценария является ситуация, когда пользователь пытается пополнить баланс, вводя некорректные данные. В таком случае приложение должно корректно обрабатывать эту ситуацию и информировать пользователя об ошибке введенных данных при попытке выполнить операцию пополнения баланса.



Рисунок 4.4 – Проверка на ошибки во вкладке «Пополнение баланса» клиента

Еще одним примером тестового сценария является ситуация, когда пользователь пытается изменить свои личные данные во вкладке «Настройки» клиента, вводя некорректные данные в поля «Ваш пароль», «Номер карты для списаний», «Email». Приложение в этом случае должно корректно обрабатывать такую ситуацию и информировать пользователя о неверно введенных данных при попытке сохранения изменений в настройках (рисунок 4.5).

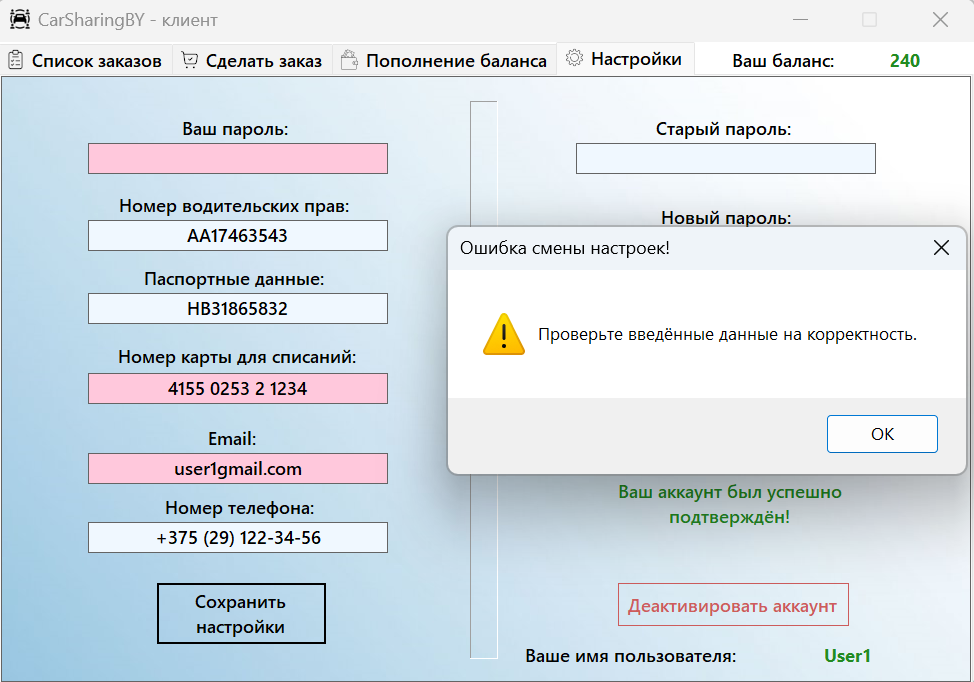


Рисунок 4.5 – Проверка на ошибки во вкладке «Настройки» клиента

Еще одним примером тестового сценария может быть ситуация, когда администратор пытается добавить новую информацию об автомобиле или обновить существующую, вводя некорректные данные в поля «Фотография», «Марка авто», «Модель авто», «Год выпуска», «Цена в час», «Номерной знак». В этом случае приложение должно корректно обрабатывать такую ситуацию и информировать администратора о неверно введенных данных при попытке сохранения изменений (рисунок 4.6).

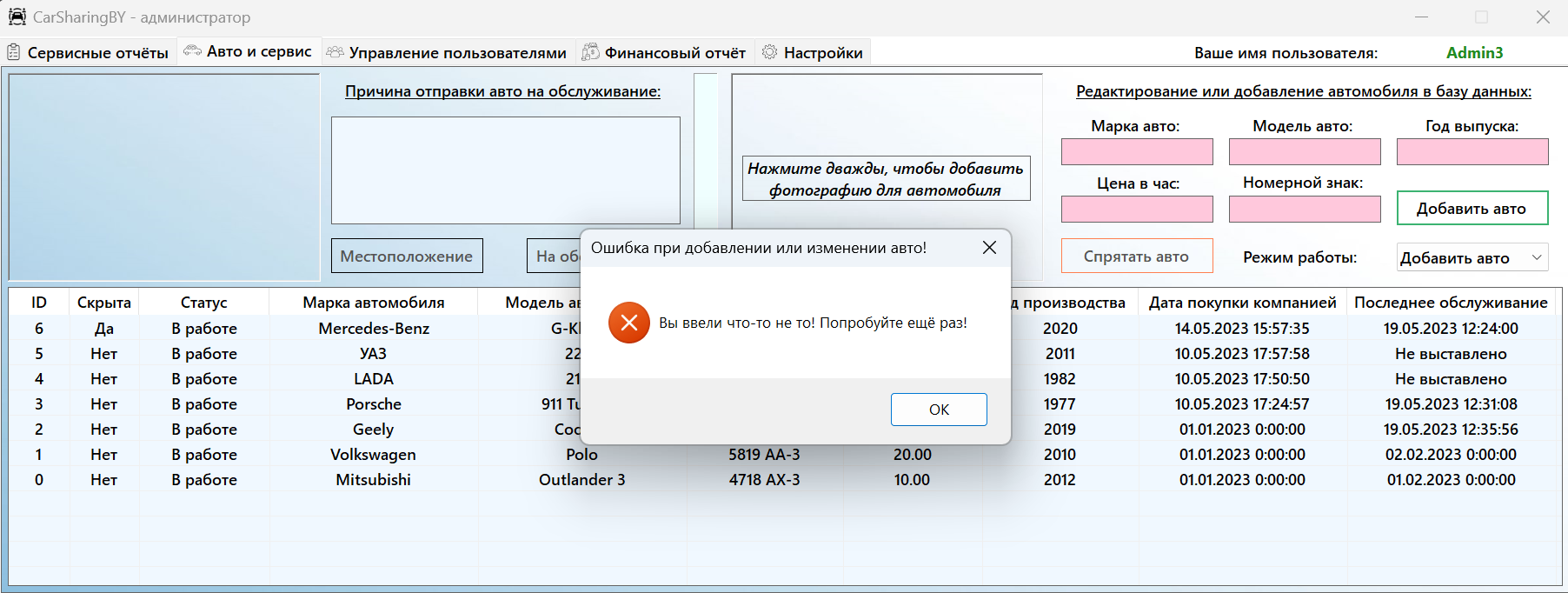


Рисунок 4.6 – Проверка на ошибки во вкладке «Авто и сервис» администратора

**4.2 Функции и их реализация в формах и алгоритмах**

Следующая таблица обозначает взаимосвязь между ключевыми пользовательскими функциями в нашем каршеринг-сервисе и их соответствующими формами и алгоритмами. Каждая функция здесь связана с определённой формой, предоставляющей необходимый пользовательский интерфейс для взаимодействия с системой, и специфическим алгоритмом, который описывает последовательность действий, выполняемых в ходе реализации функции.

Функции включают такие действия, как регистрация нового пользователя, авторизация пользователя, просмотр списка автомобилей, бронирование автомобиля и многие другие. Формы представляют собой пользовательский интерфейс, через который пользователь может взаимодействовать с данными функциями, а алгоритмы - это последовательность действий, которые система выполняет, когда пользователь активирует функцию.

Важно отметить, что это упрощенное представление реальных алгоритмов, используемых в программе. Полные алгоритмы будут гораздо более сложными и включать дополнительные детали и шаги, учитывающие различные случаи и исключения.

Таблица 4.2 – функции и их реализация в формах и алгоритмах

|  |  |  |
| --- | --- | --- |
| **Функция** | **Форма** | **Алгоритм** |
| Регистрация нового пользователя | Форма регистрации | Алгоритм проверки уникальности имени |
| Авторизация пользователя | Форма авторизации | Алгоритм проверки пароля |
| Просмотр списка автомобилей | Форма списка автомобилей | Алгоритм получения данных из базы |
| Бронирование автомобиля | Форма бронирования | Алгоритм бронирования |
| Просмотр и изменение настроек | Форма настроек | Алгоритм обновления настроек |
| Взятие автомобиля на обслуживание | Форма обслуживания | Алгоритм обслуживания |
| Закрытие отчёта обслуживания | Форма отчётов | Алгоритм закрытия отчёта |
| Просмотр отчётов обслуживания | Форма просмотра отчётов | Алгоритм просмотра данных |
| Добавление нового автомобиля | Форма добавления автомобиля | Алгоритм добавления |
| Изменение информации об автомобиле | Форма изменения информации | Алгоритм обновления данных |
| Управление пользователями | Форма управления пользователями | Алгоритм управления |

В рамках данного проекта особое внимание уделялось тестированию ролей Клиента, Сотрудника и Администратора и связанных с ними функций в системе каршеринга. Это было осуществлено с целью обеспечения бесперебойной работы приложения и повышения его эффективности. Безусловно, такое детальное исследование каждой из этих ролей в контексте системы позволяет выявить потенциальные проблемы и несоответствия, которые могут возникнуть в процессе их взаимодействия с системой.

Основной задачей тестирования в данном проекте являлось обеспечение того, что каждая функция, связанная с каждой ролью, работает как предполагалось и не вызывает нежелательных последствий. Это требует особого внимания к деталям, включая тщательное изучение и тестирование каждой пользовательской функции. Каждая роль в системе вносит свой уникальный вклад в общую эффективность и производительность приложения, поэтому их правильное функционирование является жизненно важным для обеспечения высокого качества сервиса.

Тестирование приложения в этом проекте охватывало разнообразные аспекты, включая функциональное тестирование, тестирование производительности, а также тестирование на соответствие требованиям безопасности. Наличие разнообразных пользовательских потребностей и требований подчеркивает важность глубокого понимания и проверки функций каждой роли. Необходимость обеспечения гибкого, надежного и безопасного сервиса делает тестирование ключевым этапом в процессе разработки приложения.

В результате проведенного тестирования в проекте было возможно идентифицировать и устранить различные проблемы и ошибки, что в конечном итоге привело к повышению стабильности и надежности работы приложения, улучшению его производительности и удобства использования. Это подтверждает важность тестирования в рамках процесса разработки приложений, подчеркивая его роль в обеспечении высокого уровня удовлетворенности пользователей и достижении стратегических целей проекта.

**ЗАКЛЮЧЕНИЕ**

В результате выполнения курсового проекта была проведена тщательная работа по анализу и изучению предметной области моделирования услуг автопарка, предоставляющего аренду автомобилей населению (каршеринг). Это позволило выявить ключевые проблемы и недостатки в существующих системах и разработать продукт, который соответствует нормам и требованиям, представленным в главе 1. В рамках проекта была освоена работа с *JSON*-файлами и получен опыт разработки интерфейса с помощью технологии *Windows Forms*. Весь проект был реализован с использованием объектно-ориентированного программирования на языке *C#*.

Анализ и исследование позволили определить ключевые потребности и требования пользователей к сервису каршеринга, а также выработать рекомендации по дальнейшему улучшению и развитию сервиса. Использовались различные методы и инструменты, включая изучение технической документации, анализ существующих подобных систем и обработку статистических данных. Полученные результаты стали основой для разработки более эффективных стратегий и решений, направленных на удовлетворение потребностей пользователей и обеспечение качественного предоставления услуг аренды автомобилей.

В рамках разработанного проекта были учтены разнообразные аспекты использования приложения, включая обработку исключительных ситуаций, которые могут возникнуть в процессе его использования, обеспечивая комфортную работу с сервисом.

Поставленные задачи были успешно выполнены, и приложение демонстрирует готовность к дальнейшему расширению и модернизации. Система каршеринга может быть масштабирована до размеров приложений, описанных в главе 1, с дополнительной проработкой предметной области и наличием соответствующего оборудования и инфраструктуры.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Романькова, Т. Л. Конструирование программ и языки программирования: пособие по одноим. курсу для студентов техн. специальностей дневной формы обучения / Т. Л. Романькова, Е. В. Коробейникова. – Гомель.: ГГТУ им. П.О.Сухого, 2010 – 43 с.
2. Рихтер, Дж. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#. 3-e изд. – СПб.: Питер, 2016 – 896 с.
3. Албахари, Дж. C# 7.0. Карманный справочник. – СПб.: ООО “Альфа-книга”, 2017. – 224 с.
4. Мартин, Роберт Чистый код. Создание, анализ и рефакторинг / Роберт Мартин.-М.: «Питер», 2019.–464 с.
5. Троелсен, Э. Язык программирования C# 5.0 и платформа .NET 4.5 / Э. Троелсен – М.: Вильямс, 2015. – 126 c.

**ПРИЛОЖЕНИЕ А**

**ЛИСТИНГ ПРОГРАММНОГО КОДА**

Форма «LoginForm.cs»:

using OOP\_CourseWork.Controls;

using System;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace OOP\_CourseWork

{

public partial class LoginForm : Form

{

public static readonly Color AllowedColor = Color.AliceBlue;

public static readonly Color DeniedColor = Color.FromArgb(255, 200, 220);

private string \_wasLogedIn = "no";

private RegisterForm \_registerForm = new RegisterForm();

public LoginForm()

{

InitializeComponent();

this.textBoxUsername.KeyDown += Enter\_KeyDown;

this.textBoxPassword.KeyDown += Enter\_KeyDown;

toolTipUsername.SetToolTip(textBoxUsername, "Имя пользователя может состоять из символов латинского алфавита,\n" +

"символа нижнего подчёркивания и цифр.\n" +

"Вы не можете использовать имя пользователя, если оно уже занято.");

toolTipPassword.SetToolTip(textBoxPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

this.FormClosing += LoginForm\_FormClosing;

}

private void textBoxUsername\_TextChanged(object sender, EventArgs e)

{

if (textBoxUsername.TextLength > 0)

{

char c = textBoxUsername.Text[textBoxUsername.TextLength - 1];

if (!char.IsLetter(c) && !char.IsDigit(c) && c != '\_')

{

textBoxUsername.Text = textBoxUsername.Text.Remove(textBoxUsername.TextLength - 1, 1);

return;

}

}

if (textBoxUsername.TextLength != 0 && SaveLoadControl.Users.FirstOrDefault(x => x.UserName == textBoxUsername.Text && !x.AccountDeactivated) is null)

{

textBoxUsername.BackColor = DeniedColor;

}

else

{

textBoxUsername.BackColor = AllowedColor;

}

}

private void textBoxPassword\_TextChanged(object sender, EventArgs e)

{

if (UtilsControl.CheckPasswordStrength(textBoxPassword.Text) < UtilsControl.PasswordScore.Medium)

{

textBoxPassword.BackColor = DeniedColor;

}

else

{

textBoxPassword.BackColor = AllowedColor;

}

}

private void Enter\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

buttonLogin\_Click(this, new EventArgs());

e.SuppressKeyPress = true;

}

}

private void LoginForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_wasLogedIn != "yes")

{

SaveLoadControl.SaveJSON();

Environment.Exit(0);

}

}

private void buttonLogin\_Click(object sender, EventArgs e)

{

textBoxUsername\_TextChanged(this, new EventArgs());

textBoxPassword\_TextChanged(this, new EventArgs());

if (textBoxUsername.BackColor == DeniedColor || textBoxPassword.BackColor == DeniedColor)

{

MessageBox.Show("Проверьте введённые данные на корректность.", "Ошибка входа!", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

var user = SaveLoadControl.Users.FirstOrDefault(x => x.UserName == textBoxUsername.Text.Trim() && !x.AccountDeactivated);

if (user != null)

{

if (user.IsPasswordCorrect(textBoxPassword.Text))

{

\_wasLogedIn = "yes";

SaveLoadControl.CurrentUser = user;

this.Close();

return;

}

}

MessageBox.Show("Неверное имя пользователя или пароль!", "Ошибка входа!", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

private void buttonRegister\_Click(object sender, EventArgs e)

{

this.Hide();

\_registerForm.ShowDialog();

this.Show();

}

}

}

Форма «RegisterForm.cs»:

using OOP\_CourseWork.Controls;

using OOP\_CourseWork.Models;

using System;

using System.Drawing;

using System.Linq;

using System.Net.Mail;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace OOP\_CourseWork

{

public partial class RegisterForm : Form

{

public static readonly Color AllowedColor = Color.AliceBlue;

public static readonly Color DeniedColor = Color.FromArgb(255, 200, 220);

private string \_wasRegistered = "no";

public RegisterForm()

{

InitializeComponent();

this.textBoxUsername.KeyDown += Enter\_KeyDown;

this.textBoxPassword.KeyDown += Enter\_KeyDown;

this.textBoxFullname.KeyDown += Enter\_KeyDown;

this.textBoxEmail.KeyDown += Enter\_KeyDown;

this.maskedTextBoxPhoneNumber.KeyDown += Enter\_KeyDown;

toolTipUsername.SetToolTip(textBoxUsername, "Имя пользователя может состоять из символов латинского алфавита,\n" +

"символа нижнего подчёркивания и цифр.\n" +

"Вы не можете использовать имя пользователя, если оно уже занято.");

toolTipPassword.SetToolTip(textBoxPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

toolTipFullname.SetToolTip(textBoxFullname, "Введите 3 слова: Вашу фамилию, имя, отчество.");

toolTipEmail.SetToolTip(textBoxEmail, "Введите Ваш действительный адрес электронной почты.\n" +

"Вы не можете использовать адрес, если он уже был использован.");

toolTipPhoneNumber.SetToolTip(maskedTextBoxPhoneNumber, "Введите номер телефона в указанном формате.\n" +

"Вы не можете использовать номер, если он уже был использован.");

this.FormClosing += RegisterForm\_FormClosing;

}

private void textBoxUsername\_TextChanged(object sender, EventArgs e)

{

if (textBoxUsername.TextLength > 0)

{

char c = textBoxUsername.Text[textBoxUsername.TextLength - 1];

if (!char.IsLetter(c) && !char.IsDigit(c) && c != '\_')

{

textBoxUsername.Text = textBoxUsername.Text.Remove(textBoxUsername.TextLength - 1, 1);

return;

}

}

if (textBoxUsername.TextLength != 0 && SaveLoadControl.Users.FirstOrDefault(x => x.UserName == textBoxUsername.Text) is null)

{

textBoxUsername.BackColor = AllowedColor;

} else

{

textBoxUsername.BackColor = DeniedColor;

}

}

private void textBoxPassword\_TextChanged(object sender, EventArgs e)

{

if (UtilsControl.CheckPasswordStrength(textBoxPassword.Text) < UtilsControl.PasswordScore.Medium)

{

textBoxPassword.BackColor = DeniedColor;

} else

{

textBoxPassword.BackColor = AllowedColor;

}

}

private void textBoxFullname\_TextChanged(object sender, EventArgs e)

{

var r = new Regex("^\\S+\\s\\S+\\s\\S+$");

if (r.IsMatch(textBoxFullname.Text))

{

textBoxFullname.BackColor = AllowedColor;

}

else

{

textBoxFullname.BackColor = DeniedColor;

}

}

private void textBoxEmail\_TextChanged(object sender, EventArgs e)

{

try

{

if (!(SaveLoadControl.Users.FirstOrDefault(x => x.Email == textBoxEmail.Text) is null)) throw new Exception("This email is already taken.");

MailAddress m = new MailAddress(textBoxEmail.TextLength == 0 ? " " : textBoxEmail.Text);

textBoxEmail.BackColor = AllowedColor;

}

catch

{

textBoxEmail.BackColor = DeniedColor;

}

}

private void maskedTextBoxPhoneNumber\_TextChanged(object sender, EventArgs e)

{

var r = new Regex("^\\+375\\s\\(\\d{2}\\)\\s\\d{3}\\-\\d{2}\\-\\d{2}$");

if (r.IsMatch(maskedTextBoxPhoneNumber.Text) && SaveLoadControl.Users.FirstOrDefault(x => x.Phone == maskedTextBoxPhoneNumber.Text) is null)

{

maskedTextBoxPhoneNumber.BackColor = AllowedColor;

}

else

{

maskedTextBoxPhoneNumber.BackColor = DeniedColor;

}

}

private void Enter\_KeyDown(object sender, KeyEventArgs e)

{

if (e.KeyCode == Keys.Enter)

{

buttonRegister\_Click(this, new EventArgs());

e.SuppressKeyPress = true;

}

}

private void RegisterForm\_FormClosing(object sender, FormClosingEventArgs e)

{

if (\_wasRegistered != "yes")

{

e.Cancel = false;

this.Hide();

} else

{

SaveLoadControl.SaveJSON();

Application.Restart();

}

}

private void buttonLogin\_Click(object sender, EventArgs e)

{

this.Close();

}

private void buttonRegister\_Click(object sender, EventArgs e)

{

textBoxUsername\_TextChanged(this, new EventArgs());

textBoxPassword\_TextChanged(this, new EventArgs());

textBoxFullname\_TextChanged(this, new EventArgs());

textBoxEmail\_TextChanged(this, new EventArgs());

maskedTextBoxPhoneNumber\_TextChanged(this, new EventArgs());

if (textBoxUsername.BackColor == DeniedColor ||

textBoxPassword.BackColor == DeniedColor ||

textBoxFullname.BackColor == DeniedColor ||

textBoxEmail.BackColor == DeniedColor ||

maskedTextBoxPhoneNumber.BackColor == DeniedColor)

{

MessageBox.Show("Проверьте введённые данные на корректность.", "Ошибка регистрации!", MessageBoxButtons.OK, MessageBoxIcon.Warning);

return;

}

try

{

Client client = new Client(SaveLoadControl.Users.Count, textBoxUsername.Text, textBoxPassword.Text,

textBoxFullname.Text, textBoxEmail.Text, maskedTextBoxPhoneNumber.Text);

SaveLoadControl.Users.Add(client);

MessageBox.Show("Успешная регистрация! Войдите в аккаунт для его использования.", "Успешно!", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

catch

{

MessageBox.Show("Не удалось зарегистрироваться по неизвестной причине!", "Ошибка регистрации!", MessageBoxButtons.OK, MessageBoxIcon.Warning);

}

}

}

}

Форма «ClientForm.cs»:

using Microsoft.VisualBasic;

using OOP\_CourseWork.Controls;

using OOP\_CourseWork.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Drawing.Drawing2D;

using System.Drawing.Text;

using System.Linq;

using System.Net.Mail;

using System.Text.RegularExpressions;

using System.Windows.Forms;

namespace OOP\_CourseWork

{

    public partial class ClientForm : Form

    {

        public static readonly Color AllowedColor = Color.AliceBlue;

        public static readonly Color DeniedColor = Color.FromArgb(255, 200, 220);

        public Size ImageSize = new Size(240, 160);

        public static List<Image> CarsOrderImages = new List<Image>();

        public ClientForm()

        {

            InitializeComponent();

            toolTipPassword.SetToolTip(textBoxSettings\_Password, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipPassword.SetToolTip(textBoxSettings\_OldPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipCardNumber.SetToolTip(textBoxSettings\_CardNumber, "Введите 16 цифр - номер Вашей карты для\n" +

                                                            "последующего пополнения баланса сервиса.");

            toolTipCardNumber.SetToolTip(textBoxPayment\_CardNumber, "Введите 16 цифр - номер Вашей карты для\n" +

                                                            "последующего пополнения баланса сервиса.");

            toolTipEmail.SetToolTip(textBoxSettings\_Email, "Введите Ваш действительный адрес электронной почты.\n" +

                                                           "Вы не можете использовать адрес, если он уже был использован.");

            toolTipPhoneNumber.SetToolTip(maskedTextBoxSettings\_PhoneNumber, "Введите номер телефона в указанном формате.\n" +

                                                                             "Вы не можете использовать номер, если он уже был использован.");

            toolTipSecretCode.SetToolTip(textBoxPayments\_SecretCode, "Введите секретный номер карты.\nОн состоит из трёх цифр.");

            toolTipCost.SetToolTip(textBoxPayments\_Cost, "Введите сумму пополнения Вашего баланса на нашем сервисе.\n" +

                                                         "Сумма округляется автоматически до двух знаков после запятой.");

            toolTipOrderHours.SetToolTip(textBoxMakeAnOrder\_BookingHours, "Введите количество часов, на которые хотите арендовать автомобиль.\n" +

                                                                          "Не может превышать 192-х часов (8 суток).");

            toolTipOrderBookingDateTime.SetToolTip(dateTimePickerMakeAnOrder\_BookingDate, "Укажите время, на которое бронируете автомобиль.\n" +

                                                                                          "Это будет время начала Вашего заказа.");

            toolTipOrderBookingDateTime.SetToolTip(dateTimePickerMakeAnOrder\_BookingTime, "Укажите время, на которое бронируете автомобиль.\n" +

                                                                                          "Это будет время начала Вашего заказа.");

            toolTipOrderBookingEndDateTime.SetToolTip(textBoxMakeAnOrder\_BookingEndTime,  "Время окончания заказа, рассчитанное с помощью указанных Вами параметров.");

            toolTipOrderListCarPicture.SetToolTip(pictureBoxCarPicture, "Фотография автомобиля выбранного заказа.");

            tabControlClient.SelectedIndexChanged += TabControlClient\_SelectedIndexChanged;

            listViewPayments.ColumnWidthChanging += ListViewPayments\_ColumnWidthChanging;

            listViewOrderList.ColumnWidthChanging += ListViewOrderList\_ColumnWidthChanging;

            listViewOrderList.ItemSelectionChanged += ListViewOrderList\_ItemSelectionChanged;

            this.FormClosing += ClientForm\_FormClosing;

            labelUsername.Text = SaveLoadControl.CurrentUser.UserName;

            CarsOrderImages = UtilsControl.LoadCarsOrderImages();

        }

        private void ClientForm\_FormClosing(object sender, FormClosingEventArgs e)

        {

            var result = MessageBox.Show("Вы уверены, что хотите выйти из программы?", "Точно выйти?", MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            if (result == DialogResult.Yes)

            {

                SaveLoadControl.SaveJSON();

                Environment.Exit(0);

            } else

            {

                e.Cancel = true;

            }

        }

    }

}

Форма «EmployeeForm.cs»:

using Microsoft.VisualBasic;

using OOP\_CourseWork.Controls;

using OOP\_CourseWork.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Windows.Forms;

namespace OOP\_CourseWork

{

    public partial class EmployeeForm : Form

    {

        public static readonly Color AllowedColor = Color.AliceBlue;

        public static readonly Color DeniedColor = Color.FromArgb(255, 200, 220);

        public static List<Image> CarsOrderImages = new List<Image>();

        public static string AddNewCarImagePath = "";

        public EmployeeForm()

        {

            InitializeComponent();

            toolTipOrderListCarPicture.SetToolTip(pictureBoxServiceReportList\_CarPicture, "Фотография выбранного автомобиля.");

            toolTipSettings\_Password.SetToolTip(textBoxSettings\_OldPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipSettings\_Password.SetToolTip(textBoxSettings\_NewPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipSettings\_Password.SetToolTip(textBoxSettings\_NewPasswordConfirmation, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            tabControlEmployee.SelectedIndexChanged += TabControlEmployee\_SelectedIndexChanged;

            listViewServiceReportList.ColumnWidthChanging += ListViewServiceReportList\_ColumnWidthChanging;

            listViewServiceReportList.ItemSelectionChanged += listViewServiceReportList\_ItemSelectionChanged;

            this.FormClosing += EmployeeForm\_FormClosing;

            labelUsername.Text = SaveLoadControl.CurrentUser.UserName;

            CarsOrderImages = UtilsControl.LoadCarsOrderImages();

        }

    }

}

Форма «AdminForm.cs»:

using OOP\_CourseWork.Controls;

using OOP\_CourseWork.Models;

using System;

using System.Collections.Generic;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.IO;

using System.Linq;

using System.Windows.Forms;

namespace OOP\_CourseWork

{

    public partial class AdminForm : Form

    {

        public static readonly Color AllowedColor = Color.AliceBlue;

        public static readonly Color DeniedColor = Color.FromArgb(255, 200, 220);

        public static List<Image> CarsOrderImages = new List<Image>();

        public static string AddNewCarImagePath = "";

        public AdminForm()

        {

            InitializeComponent();

            toolTipOrderListCarPicture.SetToolTip(pictureBoxServiceReportList\_CarPicture, "Фотография выбранного автомобиля.");

            toolTipOrderListCarPicture.SetToolTip(pictureBoxMakeServiceReport\_CarPicture, "Фотография выбранного автомобиля.");

            toolTipUsersManagement\_FindInTheTable.SetToolTip(textBoxUsersManagement\_FindInTheTable,

                                                             "Введите что-нибудь в это поле, чтобы запустить поиск по таблице. " +

                                                             "Если в каком-то поле будет совпадение, то вся строка останется в таблице. " +

                                                             "Иначе она будет скрыта.");

            toolTipMakeServiceReport\_PricePerHour.SetToolTip(textBoxMakeMakeServiceReport\_NewCarPricePerHour,

                                                             "Введите цену автомобиля в час. " +

                                                             "Сумма округляется автоматически до двух знаков после запятой.");

            toolTipMakeServiceReport\_ProductionYear.SetToolTip(textBoxMakeMakeServiceReport\_NewCarProductionYear, "Введите год производства автомобиля.");

            toolTipSettings\_Password.SetToolTip(textBoxSettings\_OldPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipSettings\_Password.SetToolTip(textBoxSettings\_NewPassword, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipSettings\_Password.SetToolTip(textBoxSettings\_NewPasswordConfirmation, "Введите пароль, состоящий из любых символов.\nПароль должен быть достаточно сильным.");

            toolTipMakeServiceReport\_HideOrShowCar.SetToolTip(buttonMakeServiceReport\_HideShowCar, "Если скрыть автомобиль, то клиенты не будут иметь возможности его заказать. " +

                                                                                             "Они не будут видеть этот автомобиль в списке для возможного заказа.");

            tabControlAdmin.SelectedIndexChanged += TabControlAdmin\_SelectedIndexChanged;

            listViewServiceReportList.ColumnWidthChanging += ListViewServiceReportList\_ColumnWidthChanging;

            listViewUsersManagement.ColumnWidthChanging += ListViewUsersManagement\_ColumnWidthChanging;

            listViewMakeServiceReport.ColumnWidthChanging += ListViewMakeServiceReport\_ColumnWidthChanging;

            listViewUsersManagement.ItemSelectionChanged += ListViewUsersManagement\_ItemSelectionChanged;

            listViewServiceReportList.ItemSelectionChanged += listViewServiceReportList\_ItemSelectionChanged;

            listViewMakeServiceReport.ItemSelectionChanged += ListViewMakeServiceReport\_ItemSelectionChanged;

            this.FormClosing += AdminForm\_FormClosing;

            dateTimePickerFinancialReport\_FromDate.Value = DateTime.Today.AddDays(-DateTime.Today.Day + 1);

            dateTimePickerFinancialReport\_ToDate.Value = DateTime.Today.AddDays(-DateTime.Today.Day + 1 + DateTime.DaysInMonth(DateTime.Now.Year, DateTime.Now.Month)).AddSeconds(-1);

            textBoxUsersManagement\_FindInTheTable.GotFocus += TextBoxUsersManagement\_FindInTheTable\_GotFocus;

            comboBoxMakeServiceReport\_AddOrEditCarMode.SelectedIndex = 0;

            labelUsername.Text = SaveLoadControl.CurrentUser.UserName;

            CarsOrderImages = UtilsControl.LoadCarsOrderImages();

        }

}

}

Файл «User.cs»:

using OOP\_CourseWork.Controls;

using System;

namespace OOP\_CourseWork.Models

{

internal class User

{

private int \_id;

private string \_username;

private string \_salt;

private string \_hashedPassword;

private string \_fullname;

private string \_email;

private string \_phone;

private RolesContainer \_role;

private bool \_isAccountSetupCompleted;

private bool \_accountDeactivated;

public User()

{

\_id = 0;

\_username = \_salt = \_hashedPassword = \_fullname = \_email = \_phone = string.Empty;

\_accountDeactivated = false;

\_isAccountSetupCompleted = false;

\_role = new RolesContainer();

}

public User(int id, string username, string salt, string hashedPassword, string fullname, string email, string phone, RolesContainer role, bool isAccountSetupCompleted, bool accountDeactivated)

{

\_id = id;

\_username = username;

\_salt = salt;

\_hashedPassword = hashedPassword;

\_fullname = fullname;

\_email = email;

\_phone = phone;

\_role = role;

\_isAccountSetupCompleted = isAccountSetupCompleted;

\_accountDeactivated = accountDeactivated;

}

public User(int id, string username, string password, string fullname, string email, string phone, RolesContainer role)

{

\_id = id;

\_username = username;

\_salt = Convert.ToBase64String(CryptographyControl.GenerateSalt());

\_hashedPassword = CryptographyControl.HashPasswordWithSalt(password, \_salt);

\_fullname = fullname;

\_email = email;

\_phone = phone;

\_role = role;

\_isAccountSetupCompleted = false;

\_accountDeactivated = false;

}

public int Id

{

get

{

return \_id;

}

set

{

\_id = value;

}

}

public string UserName

{

get

{

return \_username;

}

set

{

\_username = value;

}

}

public string Salt

{

get

{

return \_salt;

}

set

{

\_salt = value;

}

}

public string HashedPassword

{

get

{

return \_hashedPassword;

}

set

{

\_hashedPassword = value;

}

}

public string FullName

{

get

{

return \_fullname;

}

set

{

\_fullname = value;

}

}

public string Email

{

get

{

return \_email;

}

set

{

\_email = value;

}

}

public string Phone

{

get

{

return \_phone;

}

set

{

\_phone = value;

}

}

public RolesContainer Role

{

get

{

return \_role;

}

set

{

\_role = value;

}

}

public bool IsAccountSetupCompleted

{

get

{

return \_isAccountSetupCompleted;

}

set

{

\_isAccountSetupCompleted = value;

}

}

public bool AccountDeactivated

{

get

{

return \_accountDeactivated;

}

set

{

\_accountDeactivated = value;

}

}

public override string ToString()

{

return \_role.ToString() + ";" + \_id + ";" + \_username + ";" + \_salt + ";" + \_hashedPassword + ";" + \_fullname + ";" + \_email + ";" + \_phone + ";" + \_accountDeactivated;

}

public void CompleteAccountSetup()

{

\_isAccountSetupCompleted = true;

}

public void ActivateAccount()

{

\_accountDeactivated = false;

}

public void DeactivateAccount()

{

\_accountDeactivated = true;

}

public bool IsPasswordCorrect(string password)

{

return CryptographyControl.HashPasswordWithSalt(password, \_salt) == \_hashedPassword;

}

public bool ChangePassword(string oldPassword, string newPassword)

{

if (IsPasswordCorrect(oldPassword))

{

\_salt = Convert.ToBase64String(CryptographyControl.GenerateSalt());

\_hashedPassword = CryptographyControl.HashPasswordWithSalt(newPassword, \_salt);

return true;

}

return false;

}

public bool ChangeEmail(string oldPassword, string newEmail)

{

if (IsPasswordCorrect(oldPassword))

{

\_email = newEmail;

return true;

}

return false;

}

public bool ChangePhoneNumber(string oldPassword, string newPhoneNumber)

{

if (IsPasswordCorrect(oldPassword))

{

\_phone = newPhoneNumber;

return true;

}

return false;

}

}

}

Файл «Order.cs»:

using OOP\_CourseWork.Controls;

using System;

using System.Collections.Generic;

namespace OOP\_CourseWork.Models

{

internal class Order

{

private int \_id;

private DateTime \_orderCreatedTime;

private DateTime \_orderBookingTime;

private DateTime \_orderCancelledTime;

private Car \_orderedCar;

private int \_orderedHours;

private Payment \_orderPayment;

private List<Payment> \_orderExtendPayments;

private bool \_isCancelled;

public Order()

{

\_id = 0;

\_orderCreatedTime = DateTime.Now;

\_orderBookingTime = DateTime.MaxValue;

\_orderCancelledTime = DateTime.MaxValue;

\_orderedHours = 0;

\_orderedCar = new Car();

\_orderPayment = new Payment();

\_orderExtendPayments = new List<Payment>();

\_isCancelled = false;

}

public Order(int id, Payment payment, Car orderedCar, Client user, DateTime orderBookingTime, int orderedHours)

{

\_id = id;

\_orderCreatedTime = DateTime.Now;

\_orderBookingTime = orderBookingTime;

\_orderCancelledTime = DateTime.MaxValue;

\_orderedCar = orderedCar;

\_orderedHours = orderedHours;

\_orderPayment = payment;

SaveLoadControl.Payments.Add(\_orderPayment);

\_orderExtendPayments = new List<Payment>();

\_isCancelled = false;

}

public Order(int id, Car orderedCar, Client user, DateTime orderBookingTime, int orderedHours)

{

\_id = id;

\_orderCreatedTime = DateTime.Now;

\_orderBookingTime = orderBookingTime;

\_orderCancelledTime = DateTime.MaxValue;

\_orderedCar = orderedCar;

\_orderedHours = orderedHours;

\_orderPayment = new Payment(SaveLoadControl.Payments.Count, user, orderedHours \* orderedCar.PricePerHour);

SaveLoadControl.Payments.Add(\_orderPayment);

\_orderExtendPayments = new List<Payment>();

\_isCancelled = false;

}

public Order(int id, Car orderedCar, DateTime orderCreatedTime, DateTime orderBookingTime,

DateTime orderCancelledTime, Payment orderPayment, int orderedHours, List<Payment> orderExtendPayments, bool isCancelled)

{

\_id = id;

\_orderCreatedTime = orderCreatedTime;

\_orderBookingTime = orderBookingTime;

\_orderCancelledTime = orderCancelledTime;

\_orderedCar = orderedCar;

\_orderPayment = orderPayment;

\_orderedHours = orderedHours;

\_orderExtendPayments = orderExtendPayments;

\_isCancelled = isCancelled;

}

public int Id

{

get

{

return \_id;

}

set

{

\_id = value;

}

}

public DateTime OrderCreatedTime

{

get

{

return \_orderCreatedTime;

}

set

{

\_orderCreatedTime = value;

}

}

public DateTime OrderBookingTime

{

get

{

return \_orderBookingTime;

}

set

{

\_orderBookingTime = value;

}

}

public DateTime OrderCancelledTime

{

get

{

return \_orderCancelledTime;

}

set

{

\_orderCancelledTime = value;

}

}

public Car OrderedCar

{

get

{

return \_orderedCar;

}

set

{

\_orderedCar = value;

}

}

public int OrderHours

{

get

{

return \_orderedHours;

}

set

{

\_orderedHours = value;

}

}

public Payment OrderPayment

{

get

{

return \_orderPayment;

}

set

{

\_orderPayment = value;

}

}

public List<Payment> OrderExtendPayments

{

get

{

return \_orderExtendPayments;

}

set

{

\_orderExtendPayments = value;

}

}

public bool IsCancelled

{

get

{

return \_isCancelled;

}

set

{

\_isCancelled = value;

}

}

public bool Cancel()

{

if (DateTime.Now >= \_orderBookingTime) return false;

if (\_orderPayment.IsPayed) \_orderPayment.Refund();

foreach (var payment in \_orderExtendPayments)

{

if (payment.IsPayed) payment.Refund();

}

\_orderCancelledTime = DateTime.Now;

\_isCancelled = true;

return true;

}

public bool ExtendOrder(int hours)

{

if (hours <= 0 || hours > 192) return false;

if (\_orderedHours + hours > 192) return false;

var payment = new Payment(SaveLoadControl.Payments.Count, \_orderPayment.User, hours \* \_orderedCar.PricePerHour);

if (!payment.Pay()) return false;

SaveLoadControl.Payments.Add(payment);

\_orderExtendPayments.Add(payment);

\_orderedHours += hours;

return true;

}

}

}

Файл «Employee.cs»:

using Newtonsoft.Json;

using OOP\_CourseWork.Controls;

using System;

namespace OOP\_CourseWork.Models

{

internal class Employee : User

{

public static double SalaryPerDay = 100;

private int \_ordersProccessed;

private int \_daysWorked;

private DateTime \_dateHired;

private DateTime \_dateFired;

private DateTime \_dateLastSalaryPayed;

private string \_bankAccountNumber;

private double \_totalSalaryPayed;

private bool \_isWorkingNow;

public Employee() : base()

{

\_ordersProccessed = \_daysWorked = 0;

\_dateHired = DateTime.Now;

\_dateFired = DateTime.MinValue;

\_bankAccountNumber = string.Empty;

\_totalSalaryPayed = 0;

\_isWorkingNow = true;

}

public Employee(User user) : base(user.Id, user.UserName, user.Salt, user.HashedPassword, user.FullName, user.Email,

user.Phone, RolesContainer.Employee, user.IsAccountSetupCompleted, user.AccountDeactivated)

{

\_ordersProccessed = \_daysWorked = 0;

\_dateHired = DateTime.Now;

\_dateFired = DateTime.MinValue;

\_bankAccountNumber = string.Empty;

\_totalSalaryPayed = 0;

\_isWorkingNow = true;

}

public Employee(int id, string username, string password, string fullname, string email, string phone)

: base(id, username, password, fullname, email, phone, RolesContainer.Employee)

{

\_ordersProccessed = \_daysWorked = 0;

\_dateHired = DateTime.Now;

\_dateFired = DateTime.MinValue;

\_bankAccountNumber = string.Empty;

\_totalSalaryPayed = 0;

\_isWorkingNow = true;

}

public Employee(int id, string username, string salt, string hashedPassword, string fullname, string email, string phone, bool isAccountSetupCompleted, bool accountDeactivated,

int ordersProccessed, int daysWorked, DateTime dateHired, DateTime dateFired, string bankAccountNumber, double totalSalaryPayed, bool isWorkingNow)

: base(id, username, salt, hashedPassword, fullname, email, phone, RolesContainer.Employee, isAccountSetupCompleted, accountDeactivated)

{

\_ordersProccessed = ordersProccessed;

\_daysWorked = daysWorked;

\_dateHired = dateHired;

\_dateFired = dateFired;

\_bankAccountNumber = bankAccountNumber;

\_totalSalaryPayed = totalSalaryPayed;

\_isWorkingNow = isWorkingNow;

}

public int OrderProccessed

{

get

{

return \_ordersProccessed;

}

set

{

\_ordersProccessed = value;

}

}

public int DaysWorked

{

get

{

return \_daysWorked;

}

set

{

\_daysWorked = value;

}

}

public DateTime DateHired

{

get

{

return \_dateHired;

}

set

{

\_dateHired = value;

}

}

public DateTime DateFired

{

get

{

return \_dateFired;

}

set

{

\_dateFired = value;

}

}

public DateTime DateLastSalaryPayed

{

get

{

return \_dateLastSalaryPayed;

}

set

{

\_dateLastSalaryPayed = value;

}

}

public string BankAccountNumber

{

get

{

return \_bankAccountNumber;

}

set

{

\_bankAccountNumber = value;

}

}

public double TotalSalaryPayed

{

get

{

return \_totalSalaryPayed;

}

set

{

\_totalSalaryPayed = value;

}

}

public bool IsWorkingNow

{

get

{

return \_isWorkingNow;

}

set

{

\_isWorkingNow = value;

}

}

[JsonIgnore]

public double Salary

{

get

{

return SalaryPerDay \* \_daysWorked;

}

}

public override string ToString()

{

return base.ToString() + ";" + \_ordersProccessed + ";" + \_daysWorked + ";" + \_dateHired + ";" + \_dateFired + ";" + \_isWorkingNow;

}

public bool FireEmployee()

{

if (\_isWorkingNow)

{

if (!PaySalary()) return false;

\_dateFired = DateTime.Now;

\_isWorkingNow = false;

base.DeactivateAccount();

return true;

}

return false;

}

public bool HireEmployee()

{

if (!\_isWorkingNow)

{

\_dateHired = DateTime.Now;

\_isWorkingNow = true;

base.ActivateAccount();

return true;

}

return false;

}

public bool PaySalary()

{

BankTransaction bankTransaction = new BankTransaction(SaveLoadControl.BankTransactions.Count, BankTransaction.OurOrganizationBankAccountNumber,

\_bankAccountNumber, Salary, null);

SaveLoadControl.BankTransactions.Add(bankTransaction);

if (!bankTransaction.Debit(BankTransaction.OurOrganizationSecretCode)) return false;

\_totalSalaryPayed += Salary;

\_daysWorked = 0;

\_ordersProccessed = 0;

\_dateLastSalaryPayed = DateTime.Now;

return true;

}

}

}

Файл «Client.cs»:

using Newtonsoft.Json;

using OOP\_CourseWork.Controls;

namespace OOP\_CourseWork.Models

{

internal class Client : User

{

private string \_driverLicense;

private string \_passport;

private string \_cardNumber;

private double \_balance;

private double \_sumRating;

private int \_ordersCount;

public Client() : base()

{

\_driverLicense = \_passport = \_cardNumber = string.Empty;

\_balance = \_sumRating = 0;

\_ordersCount = 0;

}

public Client(int id, string username, string password, string fullname, string email, string phone)

: base(id, username, password, fullname, email, phone, RolesContainer.Client)

{

\_driverLicense = \_passport = \_cardNumber = string.Empty;

\_balance = 0;

\_sumRating = 0;

\_ordersCount = 0;

}

public Client(User user) : base(user.Id, user.UserName, user.Salt, user.HashedPassword, user.FullName, user.Email,

user.Phone, RolesContainer.Client, user.IsAccountSetupCompleted, user.AccountDeactivated)

{

\_driverLicense = \_passport = \_cardNumber = string.Empty;

\_balance = 0;

\_sumRating = 0;

\_ordersCount = 0;

}

public Client(int id, string username, string password, string fullname, string email, string phone,

string driverLicense, string passport, string cardNumber)

: base(id, username, password, fullname, email, phone, RolesContainer.Client)

{

\_driverLicense = driverLicense;

\_passport = passport;

\_cardNumber = cardNumber;

\_balance = 0;

\_sumRating = 0;

\_ordersCount = 0;

}

public Client(int id, string username, string salt, string hashedPassword, string fullname, string email, string phone, bool isAccountSetupCompleted, bool accountDeactivated,

string driverLicense, string passport, string cardNumber, double balance, double sumRating, int ordersCount)

: base(id, username, salt, hashedPassword, fullname, email, phone, RolesContainer.Client, isAccountSetupCompleted, accountDeactivated)

{

\_driverLicense = driverLicense;

\_passport = passport;

\_cardNumber = cardNumber;

\_balance = balance;

\_sumRating = sumRating;

\_ordersCount = ordersCount;

}

public string DriverLicense

{

get

{

return \_driverLicense;

}

set

{

\_driverLicense = value;

}

}

public string Passport

{

get

{

return \_passport;

}

set

{

\_passport = value;

}

}

public string CardNumber

{

get

{

return \_cardNumber;

}

set

{

\_cardNumber = value;

}

}

public double Balance

{

get

{

return \_balance;

}

set

{

\_balance = value;

}

}

public double SumRating

{

get

{

return \_sumRating;

}

set

{

\_sumRating = value;

}

}

public int OrderCount

{

get

{

return \_ordersCount;

}

set

{

\_ordersCount = value;

}

}

[JsonIgnore]

public double Rating

{

get

{

if (\_ordersCount == 0) return 5.0;

return \_sumRating / \_ordersCount;

}

}

public override string ToString()

{

return base.ToString() + ";" + \_driverLicense + ";" + \_passport + ";" + \_balance + ";" + \_sumRating + ";" + \_ordersCount;

}

public bool BalanceDeposit(double totalAmount, string CVV\_CVC\_code)

{

BankTransaction transaction = new BankTransaction(SaveLoadControl.BankTransactions.Count, \_cardNumber,

BankTransaction.OurOrganizationBankAccountNumber, totalAmount, this);

SaveLoadControl.BankTransactions.Add(transaction);

bool result = BalanceDeposit(transaction, CVV\_CVC\_code);

if (!result) transaction.Cancel();

return result;

}

public bool BalanceDeposit(BankTransaction transaction, string CVV\_CVC\_code)

{

if (transaction.Debit(CVV\_CVC\_code) && BalanceIncrease(transaction.TotalAmount)) return true;

return false;

}

public bool BalanceIncrease(double amount)

{

if ((long)\_balance + amount > int.MaxValue) return false;

\_balance += amount;

return true;

}

public bool BalanceDecrease(double amount)

{

if (\_balance < amount) return false;

\_balance -= amount;

return true;

}

public bool ChangeSettings(string password, string driverLicense, string passport, string cardNumber)

{

if (IsPasswordCorrect(password))

{

\_driverLicense = driverLicense;

\_passport = passport;

\_cardNumber = cardNumber;

return true;

}

return false;

}

}

}

Файл «Car.cs»:

using OOP\_CourseWork.Controls;

using System;

using System.Linq;

namespace OOP\_CourseWork.Models

{

internal class Car

{

private int \_id;

private string \_brand;

private string \_model;

private string \_carLicensePlate;

private double \_pricePerHour;

private DateTime \_productionYear;

private DateTime \_buyTime;

private DateTime \_lastServiceTime;

private double \_locationX;

private double \_locationY;

private bool \_isHidden;

public Car()

{

\_id = 0;

\_brand = string.Empty;

\_model = \_carLicensePlate = string.Empty;

\_productionYear = DateTime.MinValue;

\_buyTime = DateTime.MinValue;

\_lastServiceTime = DateTime.MinValue;

\_pricePerHour = 0;

\_locationX = 0;

\_locationY = 0;

\_isHidden = true;

}

public Car(int id, string brand, string model, string carLicensePlate, double pricePerHour,

DateTime productionYear, DateTime buyTime, DateTime lastServiceTime, double locationX, double locationY, bool isHidden)

{

\_id = id;

\_brand = brand;

\_model = model;

\_carLicensePlate = carLicensePlate;

\_productionYear = productionYear;

\_buyTime = buyTime;

\_lastServiceTime = lastServiceTime;

\_pricePerHour = pricePerHour;

\_locationX = locationX;

\_locationY = locationY;

\_isHidden = isHidden;

}

public int Id

{

get

{

return \_id;

}

set

{

\_id = value;

}

}

public string Brand

{

get

{

return \_brand;

}

set

{

\_brand = value;

}

}

public string Model

{

get

{

return \_model;

}

set

{

\_model = value;

}

}

public string CarLicensePlate

{

get

{

return \_carLicensePlate;

}

set

{

\_carLicensePlate = value;

}

}

public double PricePerHour

{

get

{

return \_pricePerHour;

}

set

{

\_pricePerHour = value;

}

}

public DateTime ProductionYear

{

get

{

return \_productionYear;

}

set

{

\_productionYear = value;

}

}

public DateTime BuyTime

{

get

{

return \_buyTime;

}

set

{

\_buyTime = value;

}

}

public DateTime LastServiceTime

{

get

{

return \_lastServiceTime;

}

set

{

\_lastServiceTime = value;

}

}

public double LocationX

{

get

{

return \_locationX;

}

set

{

\_locationX = value;

}

}

public double LocationY

{

get

{

return \_locationY;

}

set

{

\_locationY = value;

}

}

public bool IsHidden

{

get

{

return \_isHidden;

}

set

{

\_isHidden = value;

}

}

public bool IsOnServiceNow

{

get

{

return SaveLoadControl.ServiceReports.FirstOrDefault(x => !x.IsFinished && x.ServicedCar.Id == \_id) != null;

}

}

public bool IsOrderedNow

{

get

{

if (SaveLoadControl.Orders.FirstOrDefault(x => x.OrderedCar == this) is null) return false;

return SaveLoadControl.Orders.FirstOrDefault(x => x.OrderedCar == this && !x.IsCancelled && x.OrderBookingTime.AddHours(x.OrderHours) > DateTime.Now) != null;

}

}

public void HideOrShow()

{

\_isHidden = !\_isHidden;

}

public void CheckCarLocation()

{

UtilsControl.GetRandomCoords(out \_locationX, out \_locationY);

}

}

}

Файл «BankTransaction.cs»:

using System;

using System.Windows.Forms;

namespace OOP\_CourseWork.Models

{

internal class BankTransaction

{

public static readonly string ServiceCentreBankAccountNumber = "BY20 OLMP 3135 0000 0010 0000 0933";

public static readonly string OurOrganizationBankAccountNumber = "BY20 OLMP 3136 0000 0020 0010 9045";

public static readonly string OurOrganizationSecretCode = "045";

public static readonly int TransactionTries = 3;

public static readonly int MinutesForTransaction = 15;

private int \_id;

private string \_fromCardNumberOrBankAccountNumber;

private string \_toCardNumberOrBankAccountNumber;

private User \_user;

private DateTime \_createdTime;

private DateTime \_payedTime;

private DateTime \_cancelledTime;

private double \_totalAmount;

private int \_totalTries;

private bool \_isFinished;

private bool \_isCancelled;

public BankTransaction()

{

\_id = 0;

\_fromCardNumberOrBankAccountNumber = \_toCardNumberOrBankAccountNumber = string.Empty;

\_createdTime = DateTime.Now;

\_payedTime = DateTime.MinValue;

\_cancelledTime = DateTime.MinValue;

\_totalAmount = 0;

\_totalTries = 0;

\_isFinished = false;

\_isCancelled = false;

}

public BankTransaction(int id, string fromCardNumberOrBankAccountNumber, string toCardNumberOrBankAccountNumber, double totalAmount, User user)

{

\_id = id;

\_fromCardNumberOrBankAccountNumber = fromCardNumberOrBankAccountNumber;

\_toCardNumberOrBankAccountNumber = toCardNumberOrBankAccountNumber;

\_user = user;

\_createdTime = DateTime.Now;

\_payedTime = DateTime.MinValue;

\_cancelledTime = DateTime.MinValue;

\_totalAmount = totalAmount;

\_totalTries = 0;

\_isFinished = false;

\_isCancelled = false;

}

public BankTransaction(int id, string fromCardNumberOrBankAccountNumber, string toCardNumberOrBankAccountNumber, User user, DateTime createdTime, DateTime payedTime, DateTime cancelledTime, double totalAmount, int totalTries, bool isPayed, bool isCancelled)

{

\_id = id;

\_fromCardNumberOrBankAccountNumber = fromCardNumberOrBankAccountNumber;

\_toCardNumberOrBankAccountNumber = toCardNumberOrBankAccountNumber;

\_user = user;

\_createdTime = createdTime;

\_payedTime = payedTime;

\_cancelledTime = cancelledTime;

\_totalAmount = totalAmount;

\_totalTries = totalTries;

\_isFinished = isPayed;

\_isCancelled = isCancelled;

}

public int Id

{

get

{

return \_id;

}

set

{

\_id = value;

}

}

public string FromCardNumberOrBankAccountNumber

{

get

{

return \_fromCardNumberOrBankAccountNumber;

}

set

{

\_fromCardNumberOrBankAccountNumber = value;

}

}

public string ToCardNumberOrBankAccountNumber

{

get

{

return \_toCardNumberOrBankAccountNumber;

}

set

{

\_toCardNumberOrBankAccountNumber = value;

}

}

public User User

{

get

{

return \_user;

}

set

{

\_user = value;

}

}

public DateTime CreatedTime

{

get

{

return \_createdTime;

}

set

{

\_createdTime = value;

}

}

public DateTime PayedTime

{

get

{

return \_payedTime;

}

set

{

\_payedTime = value;

}

}

public DateTime CancelledTime

{

get

{

return \_cancelledTime;

}

set

{

\_cancelledTime = value;

}

}

public double TotalAmount

{

get

{

return \_totalAmount;

}

set

{

\_totalAmount = value;

}

}

public int TotalTries

{

get

{

return \_totalTries;

}

set

{

\_totalTries = value;

}

}

public bool IsFinished

{

get

{

return \_isFinished;

}

set

{

\_isFinished = value;

}

}

public bool IsCancelled

{

get

{

return \_isCancelled;

}

set

{

\_isCancelled = value;

}

}

// Симуляция запроса в банк. В зависимости от адреса выходящего и входящего будем понимать, пришли или ушли деньги относительно нашего счёта

public bool Debit(string secretCode)

{

if (DateTime.Now >= \_createdTime.AddMinutes(MinutesForTransaction))

{

Cancel();

return false;

}

if (++\_totalTries > TransactionTries)

{

Cancel();

return false;

}

if (\_isFinished) return false;

if (\_fromCardNumberOrBankAccountNumber.Substring(\_fromCardNumberOrBankAccountNumber.Length - 3) != secretCode) return false;

\_payedTime = DateTime.Now;

\_isFinished = true;

return true;

}

public bool Cancel()

{

if (\_isCancelled) return false;

\_cancelledTime = DateTime.Now;

\_isCancelled = true;

return true;

}

}

}

Файл «Admin.cs»:

using OOP\_CourseWork.Controls;

using System;

using System.Linq;

namespace OOP\_CourseWork.Models

{

internal class Admin : User

{

private int \_totalCarsServiced;

public Admin() : base()

{

\_totalCarsServiced = 0;

base.CompleteAccountSetup();

}

public Admin(User user) : base(user.Id, user.UserName, user.Salt, user.HashedPassword, user.FullName, user.Email,

user.Phone, RolesContainer.Employee, user.IsAccountSetupCompleted, user.AccountDeactivated)

{

\_totalCarsServiced = 0;

base.CompleteAccountSetup();

}

public Admin(int id, string username, string password, string fullname, string email, string phone)

: base(id, username, password, fullname, email, phone, RolesContainer.Admin)

{

\_totalCarsServiced = 0;

base.CompleteAccountSetup();

}

public Admin(int id, string username, string salt, string hashedPassword, string fullname, string email, string phone, bool isAccountSetupCompleted, bool accountDeactivated,

int totalCarServiced)

: base(id, username, salt, hashedPassword, fullname, email, phone, RolesContainer.Admin, isAccountSetupCompleted, accountDeactivated)

{

\_totalCarsServiced = totalCarServiced;

base.CompleteAccountSetup();

}

public int TotalCarsServiced

{

get

{

return \_totalCarsServiced;

}

set

{

\_totalCarsServiced = value;

}

}

public override string ToString()

{

return base.ToString();

}

public double GetPotentionalFinancialReport(DateTime date1, DateTime date2)

{

var serviceTransactions = SaveLoadControl.ServiceReports.Where(x => x.StartedDate >= date1 &&

x.FinishedDate == DateTime.MinValue &&

x.StartedDate.AddDays(x.PlannedCompletionDays) <= date2);

return GetFinancialReport(date1, date2) - serviceTransactions.Sum(x => x.Cost);

}

public double GetFinancialReport(DateTime date1, DateTime date2)

{

var plusTransactions = SaveLoadControl.BankTransactions.Where(x => x.IsFinished &&

x.ToCardNumberOrBankAccountNumber == BankTransaction.OurOrganizationBankAccountNumber &&

x.CreatedTime >= date1 &&

x.CreatedTime <= date2);

var minusTransactions = SaveLoadControl.BankTransactions.Where(x => x.IsFinished &&

x.FromCardNumberOrBankAccountNumber == BankTransaction.OurOrganizationBankAccountNumber &&

x.CreatedTime >= date1 &&

x.CreatedTime <= date2);

return plusTransactions.Sum(x => x.TotalAmount) - minusTransactions.Sum(x => x.TotalAmount);

}

public bool PutCarOnService(Car car, string description)

{

ServiceReport serviceReport = new ServiceReport(SaveLoadControl.ServiceReports.Count, description, car);

SaveLoadControl.ServiceReports.Add(serviceReport);

++\_totalCarsServiced;

return true;

}

}

}

Файл «UtilsControl.cs»:

using OOP\_CourseWork.Models;

using System;

using System.Collections.Generic;

using System.Drawing;

using System.Linq;

using System.Text.RegularExpressions;

using System.Threading;

using System.Windows.Forms;

namespace OOP\_CourseWork.Controls

{

internal class UtilsControl

{

public enum PasswordScore

{

Blank = 0,

VeryWeak = 1,

Weak = 2,

Medium = 3,

Strong = 4,

VeryStrong = 5

}

public static PasswordScore CheckPasswordStrength(string password)

{

int score = 0;

if (password.Length < 1) return PasswordScore.Blank;

if (password.Length < 4) return PasswordScore.Weak;

if (password.Length >= 8) ++score;

if (password.Length >= 12) ++score;

if (Regex.Match(password, "\\d+", RegexOptions.ECMAScript).Success) ++score;

if (Regex.Match(password, "[a-zа-я]", RegexOptions.ECMAScript).Success &&

Regex.Match(password, "[A-ZА-Я]", RegexOptions.ECMAScript).Success) ++score;

if (Regex.Match(password, "[!@#$%^&\*?\_~.,\\-£+()]", RegexOptions.ECMAScript).Success) ++score;

return (PasswordScore)score;

}

public static void StartNeccesaryForm()

{

SaveLoadControl.LoadJSON();

new Thread(() => SaveLoadControl.SaveWithCheck()).Start();

Thread.Sleep(250);

Application.Run(new LoginForm());

if (SaveLoadControl.CurrentUser is Client)

{

Application.Run(new ClientForm());

}

else

if (SaveLoadControl.CurrentUser is Employee)

{

Application.Run(new EmployeeForm());

}

else

if (SaveLoadControl.CurrentUser is Admin)

{

Application.Run(new AdminForm());

}

else

{

SaveLoadControl.SaveJSON();

Environment.Exit(0);

}

}

public static void GetRandomCoords(out double locationX, out double locationY)

{

const int arr\_length = 5;

double[] x\_s = new double[arr\_length] { 52.405456, 52.406755, 52.407157, 52.405168, 52.406949 };

double[] y\_s = new double[arr\_length] { 30.937795, 30.936868, 30.939400, 30.940515, 30.935177 };

Random random = new Random(Guid.NewGuid().GetHashCode());

int index = random.Next(arr\_length);

locationX = x\_s[index];

locationY = y\_s[index];

}

public static List<Image> LoadCarsOrderImages()

{

List<Image> list = new List<Image>();

for (int i = 0; i < SaveLoadControl.Cars.Count; i++)

{

Image image;

try

{

image = LoadImageFromFileSafely($"images\\car\_{i}.png");

}

catch

{

image = (Image)Properties.Resources.TheImageHaveDisappeared.Clone();

}

list.Add(image);

}

return list;

}

public static Image LoadImageFromFileSafely(string fileName)

{

Image image;

using (var bmpTemp = new Bitmap(fileName))

{

image = new Bitmap(bmpTemp);

}

return image;

}

Файл «SaveLoadControl.cs»:

using Newtonsoft.Json;

using OOP\_CourseWork.Models;

using System;

using System.Collections.Generic;

using System.IO;

using System.Threading;

namespace OOP\_CourseWork.Controls

{

internal class SaveLoadControl

{

public static readonly int SaveRefreshRateMilliseconds = 1000;

private static readonly string DBPath = "db.json";

private static readonly string KeyPath = "enc.key";

private static string oldDBString = "";

private static JsonSerializerSettings settingsJSON = new JsonSerializerSettings { TypeNameHandling = TypeNameHandling.All };

private static byte[] EncryptionKey;

private static byte[] EncryptionIV;

public static List<User> Users = new List<User>(); //simple

public static List<Car> Cars = new List<Car>(); //complicated

public static List<Order> Orders = new List<Order>(); //complicated

public static List<Payment> Payments = new List<Payment>(); //complicated

public static List<BankTransaction> BankTransactions = new List<BankTransaction>(); //simple

public static List<ServiceReport> ServiceReports = new List<ServiceReport>(); //complicated

public static User CurrentUser = null;

public static void SaveWithCheck()

{

while (true)

{

string newDBString = GenerateDataForSaving();

if (newDBString != oldDBString)

{

SaveJSON();

oldDBString = newDBString;

}

Thread.Sleep(SaveRefreshRateMilliseconds);

}

}

public static bool SaveJSON()

{

try

{

var json = GenerateDataForSaving();

if (json == "-1") return false;

File.WriteAllText(DBPath, AesGcm256.encrypt(json, EncryptionKey, EncryptionIV));

return true;

} catch

{

return false;

}

}

public static string GenerateDataForSaving()

{

try

{

JSON\_export.Root export = new JSON\_export.Root();

export.Users = Users;

export.Cars = Cars;

export.Orders = Orders;

export.Payments = Payments;

export.BankTransactions = BankTransactions;

export.ServiceReports = ServiceReports;

var json = JsonConvert.SerializeObject(export, settingsJSON);

return json;

}

catch

{

return "-1";

}

}

public static bool LoadJSON()

{

try

{

GetEncryptionKey();

string data = File.ReadAllText(DBPath);

oldDBString = AesGcm256.decrypt(data, EncryptionKey, EncryptionIV);

JSON\_export.Root deserialized = JsonConvert.DeserializeObject<JSON\_export.Root>(oldDBString, settingsJSON);

Users = deserialized.Users;

Cars = deserialized.Cars;

Orders = deserialized.Orders;

Payments = deserialized.Payments;

BankTransactions = deserialized.BankTransactions;

ServiceReports = deserialized.ServiceReports;

// Исправляем проблему дублирования. Присваиваем объекты из массива, новосозданные (дублированные) объекты удаляются.

foreach (var a in Orders)

{

a.OrderedCar = Cars[a.OrderedCar.Id];

a.OrderPayment = Payments[a.OrderPayment.Id];

}

foreach (var a in Payments)

{

a.User = (Client)Users[a.User.Id];

}

foreach (var a in ServiceReports)

{

a.ServicedCar = Cars[a.ServicedCar.Id];

}

return true;

} catch

{

return false;

}

}

public static bool GetEncryptionKey()

{

if (!File.Exists(DBPath)) File.WriteAllText(DBPath, "");

if (!File.Exists(KeyPath) || !File.Exists(DBPath)) return false;

var lines = File.ReadAllLines(KeyPath);

if (lines.Length != 2) return false;

var key = Convert.FromBase64String(lines[0]);

var iv = Convert.FromBase64String(lines[1]);

if (key.Length != AesGcm256.KeyBitSize / 8 && iv.Length != AesGcm256.NonceBitSize / 8) return false;

EncryptionKey = key;

EncryptionIV = iv;

return true;

}

public static void CreateKeys()

{

File.WriteAllText(KeyPath, Convert.ToBase64String(AesGcm256.NewKey()) + Environment.NewLine);

File.AppendAllText(KeyPath, Convert.ToBase64String(AesGcm256.NewIv()));

}

}

}

Файл «CryptographyControl.cs»:

using System;

using System.Security.Cryptography;

using System.Text;

namespace OOP\_CourseWork.Controls

{

internal class CryptographyControl

{

public readonly static int bytesSize = 96;

public readonly static int iterationsCount = 10000;

public static string HashPasswordWithSalt(string password, string salt)

{

var saltBytes = Convert.FromBase64String(salt);

var passBytes = Encoding.Unicode.GetBytes(password);

using (var rfc2898DeriveBytes = new Rfc2898DeriveBytes(passBytes, saltBytes, iterationsCount))

{

return Convert.ToBase64String(rfc2898DeriveBytes.GetBytes(bytesSize));

}

}

public static byte[] GenerateSalt()

{

var salt = new byte[bytesSize];

using (var random = new RNGCryptoServiceProvider())

{

random.GetNonZeroBytes(salt);

}

return salt;

}

}

}

Файл «AesGcm256.cs»:

using Org.BouncyCastle.Crypto.Engines;

using Org.BouncyCastle.Crypto.Modes;

using Org.BouncyCastle.Crypto.Parameters;

using Org.BouncyCastle.Security;

using System;

using System.Text;

namespace OOP\_CourseWork.Controls

{

internal class AesGcm256

{

private static readonly SecureRandom Random = new SecureRandom();

// Pre-configured Encryption Parameters

public static readonly int NonceBitSize = 128;

public static readonly int MacBitSize = 128;

public static readonly int KeyBitSize = 256;

private AesGcm256() { }

public static byte[] NewKey()

{

var key = new byte[KeyBitSize / 8];

Random.NextBytes(key);

return key;

}

public static byte[] NewIv()

{

var iv = new byte[NonceBitSize / 8];

Random.NextBytes(iv);

return iv;

}

public static Byte[] HexToByte(string hexStr)

{

byte[] bArray = new byte[hexStr.Length / 2];

for (int i = 0; i < (hexStr.Length / 2); i++)

{

byte firstNibble = Byte.Parse(hexStr.Substring((2 \* i), 1),

System.Globalization.NumberStyles.HexNumber); // [x,y)

byte secondNibble = Byte.Parse(hexStr.Substring((2 \* i) + 1, 1),

System.Globalization.NumberStyles.HexNumber);

int finalByte = (secondNibble) | (firstNibble << 4); // bit-operations

// only with numbers, not bytes.

bArray[i] = (byte)finalByte;

}

return bArray;

}

public static string toHex(byte[] data)

{

string hex = string.Empty;

foreach (byte c in data)

{

hex += c.ToString("X2");

}

return hex;

}

public static string toHex(string asciiString)

{

string hex = string.Empty;

foreach (char c in asciiString)

{

int tmp = c;

hex += string.Format("{0:x2}", System.Convert.ToUInt32(tmp.ToString()));

}

return hex;

}

public static string encrypt(string PlainText, byte[] key, byte[] iv)

{

string sR = string.Empty;

try

{

byte[] plainBytes = Encoding.UTF8.GetBytes(PlainText);

GcmBlockCipher cipher = new GcmBlockCipher(new AesEngine());

AeadParameters parameters =

new AeadParameters(new KeyParameter(key), 128, iv, null);

cipher.Init(true, parameters);

byte[] encryptedBytes = new byte[cipher.GetOutputSize(plainBytes.Length)];

Int32 retLen = cipher.ProcessBytes

(plainBytes, 0, plainBytes.Length, encryptedBytes, 0);

cipher.DoFinal(encryptedBytes, retLen);

sR = Convert.ToBase64String(encryptedBytes, Base64FormattingOptions.None);

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

return sR;

}

public static string decrypt(string EncryptedText, byte[] key, byte[] iv)

{

string sR = string.Empty;

try

{

byte[] encryptedBytes = Convert.FromBase64String(EncryptedText);

GcmBlockCipher cipher = new GcmBlockCipher(new AesEngine());

AeadParameters parameters =

new AeadParameters(new KeyParameter(key), 128, iv, null);

//ParametersWithIV parameters = new ParametersWithIV(new KeyParameter(key), iv);

cipher.Init(false, parameters);

byte[] plainBytes = new byte[cipher.GetOutputSize(encryptedBytes.Length)];

Int32 retLen = cipher.ProcessBytes

(encryptedBytes, 0, encryptedBytes.Length, plainBytes, 0);

cipher.DoFinal(plainBytes, retLen);

sR = Encoding.UTF8.GetString(plainBytes).TrimEnd("\r\n\0".ToCharArray());

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

Console.WriteLine(ex.StackTrace);

}

return sR;

}

}

}

Файл «Program.cs»:

using OOP\_CourseWork.Controls;

using System;

using System.Windows.Forms;

namespace OOP\_CourseWork

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

UtilsControl.StartNeccesaryForm();

}

}

}