АВТОНОМНАЯ НЕКОММЕРЧЕСКАЯ ОРГАНИЗАЦИЯ
ПРОФЕССИОНАЛЬНАЯ ОБРАЗОВАТЕЛЬНАЯ ОРГАНИЗАЦИЯ
МОСКОВСКИЙ МЕЖДУНАРОДНЫЙ КОЛЛЕДЖ ЦИФРОВЫХ ТЕХНОЛОГИЙ
«АКАДЕМИЯ ТОП»

ИНДИВИДУАЛЬНЫЙ ПРОЕКТ

Уровень профессионального образования:
Среднее профессиональное образование

Программа подготовки специалистов среднего звена
по специальности:
09.02.07 Информационные системы и программирование

Квалификация: Программист

Учебный предмет: Технология доступа к базам данных ADO.NET

Тема: Система аренды электросамокатов и велосипедов.

Преподаватель:

О. А. Рослова _____

Обучающиеся:

М.С. Селиванов_____

Санкт-Петербург, 24 октября 2025 год

Введение

Актуальность проекта

Развитие городской мобильности и популяризация экологичных видов транспорта обуславливают высокий спрос на услуги краткосрочной аренды электросамокатов и велосипедов. Эффективное управление таким бизнесом, включающее учет транспорта, клиентов, арендных сессий и финансовых операций, требует надежной и производительной системы автоматизации. Разработка специализированного программного обеспечения позволяет не только оптимизировать эти процессы, но и обеспечить масштабируемость бизнеса и высокое качество обслуживания клиентов.

Технологический контекст и обоснование выбора технологии

Для взаимодействия приложения с базой данных был выбран Dapper – высокопроизводительный микро-ORM (Object-Relational Mapper), разработанный командой Stack Overflow и расширяющий возможности IDbConnection.

Выбор Dapper в качестве основной технологии доступа к данным обусловлен следующими факторами:

Производительность: Dapper демонстрирует скорость работы, близкую к сырому ADO.NET, поскольку генерирует минимальный накладной код при маппинге результатов SQL-запросов в объекты. Это критически важно для системы, которая должна быстро обрабатывать множество операций аренды и возврата.

Простота и контроль: В отличие от полнофункциональных ORM, таких как Entity Framework, Dapper не скрывает SQL от разработчика. Это предоставляет полный контроль над формированием запросов, что позволяет писать максимально эффективные и оптимизированные команды для сложных выборок и отчетов.

Безопасность: Dapper полностью поддерживает параметризованные запросы, что является основной мерой защиты от SQL-инъекций, обеспечивая при этом удобный синтаксис работы с параметрами.

Легковесность: Dapper добавляет минимальное количество абстракций, что делает его простым в изучении и интеграции в проект, а также снижает потребление ресурсов приложения.

Цель и задачи проекта

Целью проекта является разработка высокопроизводительного и масштабируемого программного обеспечения для автоматизации деятельности сервиса аренды электросамокатов и велосипедов с использованием современного стека технологий .NET и микро-ORM Dapper.

Для достижения этой цели необходимо решить следующие задачи:

Провести анализ предметной области и сформулировать функциональные и нефункциональные требования к системе.

Спроектировать нормализованную реляционную модель базы данных, включающую таблицы для учета транспорта, пользователей, арендных

операций, платежей и справочной информации.

Реализовать слой доступа к данным (Data Access Layer, DAL) с использованием Dapper для выполнения CRUD-операций (Create, Read, Update, Delete) и сложных бизнес-запросов.

Разработать пользовательский интерфейс для операторов системы и клиентов, обеспечивающий интуитивное взаимодействие с основными функциями: регистрация, поиск транспорта, оформление и завершение аренды, формирование отчетов.

Провести интеграционное тестирование слоя доступа к данным для проверки корректности маппинга и работы всех запросов.

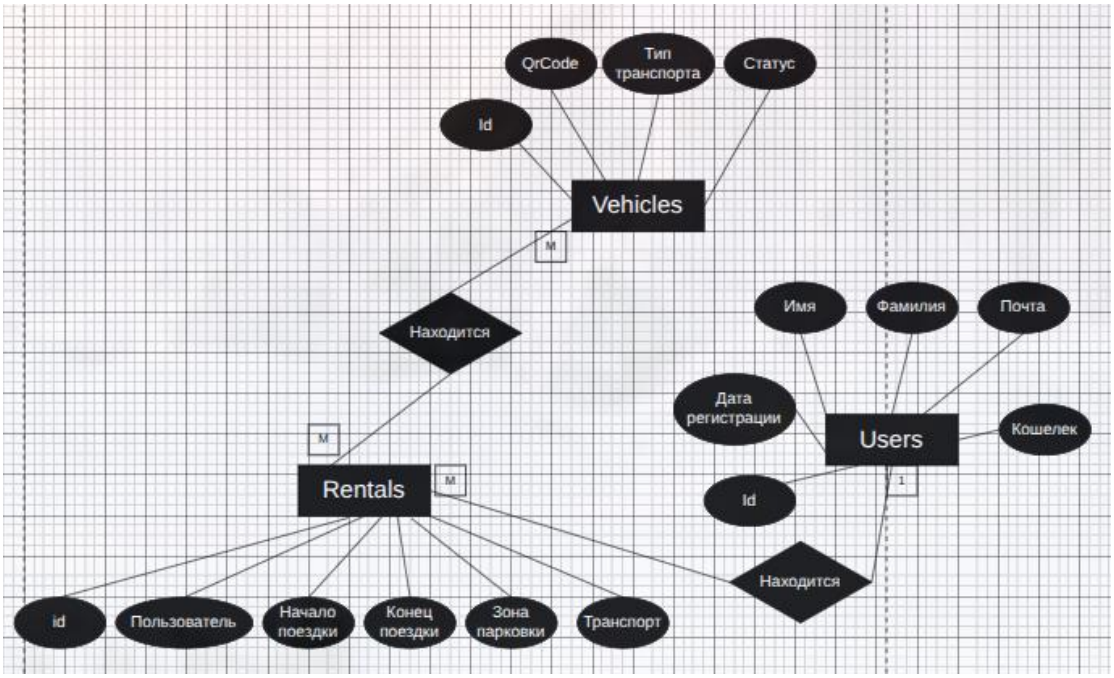
Проектирование базы данных

Для создание базы были выделены ключевые объекты (сущности):

1. Users (Пользователи)
   1.1. Атрибуты: id, Дата регистрации, Имя, Фамилия, Почта, Кошелек.
   1.2. Описание: содержит в себе полные данные о арендаторах, которые берут транспорт на прокат. Также содержит кошелек, в котором хранится номер карты пользователя, с которого списываются деньги за аренду.

2. Vehicles (Транспорт)
   2.1. Атрибуты: id, QrCode, Тип траспорта, Статус
   2.2. Описание: в этой таблице содержится вся информации об определенном транспорте, его статус работы, тип транспорта для учета цены в минуту и уникальный qrcode, для уникальности устройства от себя подобных.

3. Rentals (Аренды)
   3.1. Атрибуты: id, Пользователь, Начало поездки, Конец поездки, Зона парковки, Транспорт.
   3.2. Описание: Это основная таблица, в которой хранятся все данные об арендах, в ней показано дата/время – начала и конца поездки, зона парковки, чтобы узнать место, куда припаркован транспорт, чтобы выписать штраф или найти его, также указан пользователь, чтобы идентифицировать арендатора, а также сам транспорт.



Vehicles			
id	QrCode	Тип транспорта	Статус
1	hjkjl234lhl234lhh3k	Велосипед, Самокат	Не на ходу, Недостаточно заряда

Users					
id	Дата регистрации	Имя	Фамилия	Почта	Кошелек
1	19.09.2009	Матвей	Селиванов	matvejselivdsnjd@yandex.ru	8805553535 4345 09/19

Rentals					
id	Пользователь	Начало поездки	Конец поездки	Зона парковки	Транспорт
1	8805553535 4345 09/1	20:47-21.04.2024	20:56-21.04.2024	Невский проспект, 74-76Г	hjkjl234lhl234lhh3k

Нормализация в 1NF

Vehicles			
id	QrCode	Тип транспорта	Статус
1	hjkjl234lhl234lhh3k	Самокат	Недостаточно заряда

Users					
id	Дата регистрации	Имя	Фамилия	Почта	Кошелек
1	19.09.2009	Матвей	Селиванов	matvejselivdsnjd@yandex.ru	8805553535 4345 09/19

Rentals					
id	Пользователь_FK	Начало поездки	Конец поездки	Зона парковки	Транспорт_FK
1	1	20:47-21.04.2024	20:56-21.04.2024	Невский проспект, 74-76Г	1

Для нормализации в 1 форму, было выполнено:

1. Атомарные данные, нету несколько данных в одной клетке.
2. Каждая запись стала уникальная.
3. Появились ключи, по которым можно получить данные из другой таблицы без повторений.

Нормализация во 2NF

Vehicles			
id	QrCode	Тип транспорта	Статус_FK
1	hjkjl234lhl234lhh3k	Самокат	1

Users					
id	Дата регистрации	Имя	Фамилия	Почта	Кошелек_FK
1	19.09.2009	Матвей	Селиванов	matvejselivdsnjd@yandex.ru	1

Rentals					
id	Пользователь_FK	Начало поездки	Конец поездки	Зона парковки_FK	Транспорт_FK
1	1	20:47-21.04.2024	20:56-21.04.2024	1	1

Статус		
id	Тип статуса	Описание статуса
1	Критический	Повреждена батарея
2	Предупреждение	Повреждена батарея

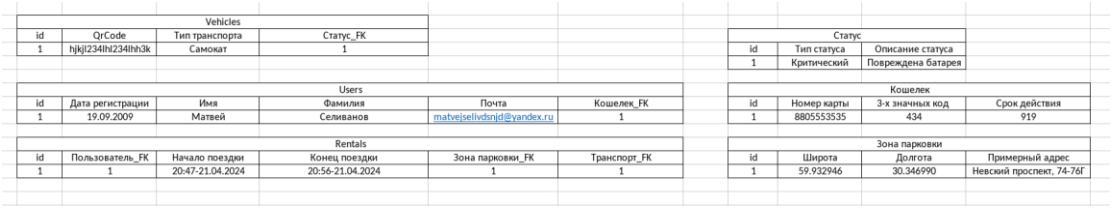
Кошелек			
id	Номер карты	3-х значных код	Срок действия
1	8805553535	434	919

Зона парковки			
id	Широта	Долгота	Примерный адрес
1	59.932946	30.346990	Невский проспект, 74-76Г

Для нормализации во 2 форму, было выполнено:

1. Статус в таблице “Vehicles” вынесен в отдельную таблицу “Статус”, где более четко расписан.
2. Кошелек в таблице “Users” был вынесен в отдельную таблицу “Кошелек”, где полные данные о карте были разнесены по разным колонкам для лучшего их чтения
3. Зона парковки в таблице “Rentals” была вынесена в таблицу “Зона парковки”, где достаточно расписаны точные координаты транспорта и его примерный адрес.

Нормализация в 3NF



Vehicles			
id	QrCode	Тип транспорта	Статус_FK
1	hjkjl234lhl234lhh3k	Самокат	1

Статус	
Тип статуса	Описание статуса
Критический	Повреждена батарея

Users				
id	Дата регистрации	Имя	Фамилия	Почта	Кошелек_FK
1	19.09.2009	Матвей	Селиванов	matvejselivdsnjd@yandex.ru	1

Кошелек		
Номер карты	3-х значных код	Срок действия
8805553535	434	919

Rentals					
id	Пользователь_FK	Начало поездки	Конец поездки	Зона парковки_FK	Транспорт_FK
1	1	20:47-21.04.2024	20:56-21.04.2024	1	1

Зона парковки		
Широта	Долгота	Примерный адрес
59.932946	30.346990	Невский проспект, 74-76Г

Для нормализации в 3 форму, было выполнено:

1. Убраны транзитивные зависимости и повторяющиеся данные.

Полная база данных в 3NF

Vehicles	
id	INT NOT NULL PRIMARY KEY IDENTITY
QrCode	NVARCHAR(19) NOT NULL
Тип транспорта	NVARCHAR(30) NOT NULL
Статус_FK	INT NOT NULL FOREIGN KEY REFERENCES Статус(id)

Users	
id	INT NOT NULL PRIMARY KEY IDENTITY
Дата регистрации	DATE NOT NULL
Имя	NVARCHAR(50) NOT NULL
Фамилия	NVARCHAR(50) NOT NULL
Почта	NVARCHAR(100) NOT NULL
Кошелек	INT NOT NULL FOREIGN KEY REFERENCES Кошелек(id)

Rentals	
id	INT NOT NULL PRIMARY KEY IDENTITY
Пользователь_FK	INT NOT NULL FOREIGN KEY REFERENCES Users(id)
Начало поездки	DATETIME NOT NULL
Конец поездки	DATETIME NOT NULL
Зона парковки_FK	INT NOT NULL FOREIGN KEY REFERENCES Зона парковки(id)
Транспорт_FK	INT NOT NULL FOREIGN KEY REFERENCES Vehicles(id)

Статус	
id	INT NOT NULL PRIMARY KEY IDENTITY
Тип статуса	NVARCHAR(30) NOT NULL
Описание статуса	NVARCHAR(255) NOT NULL

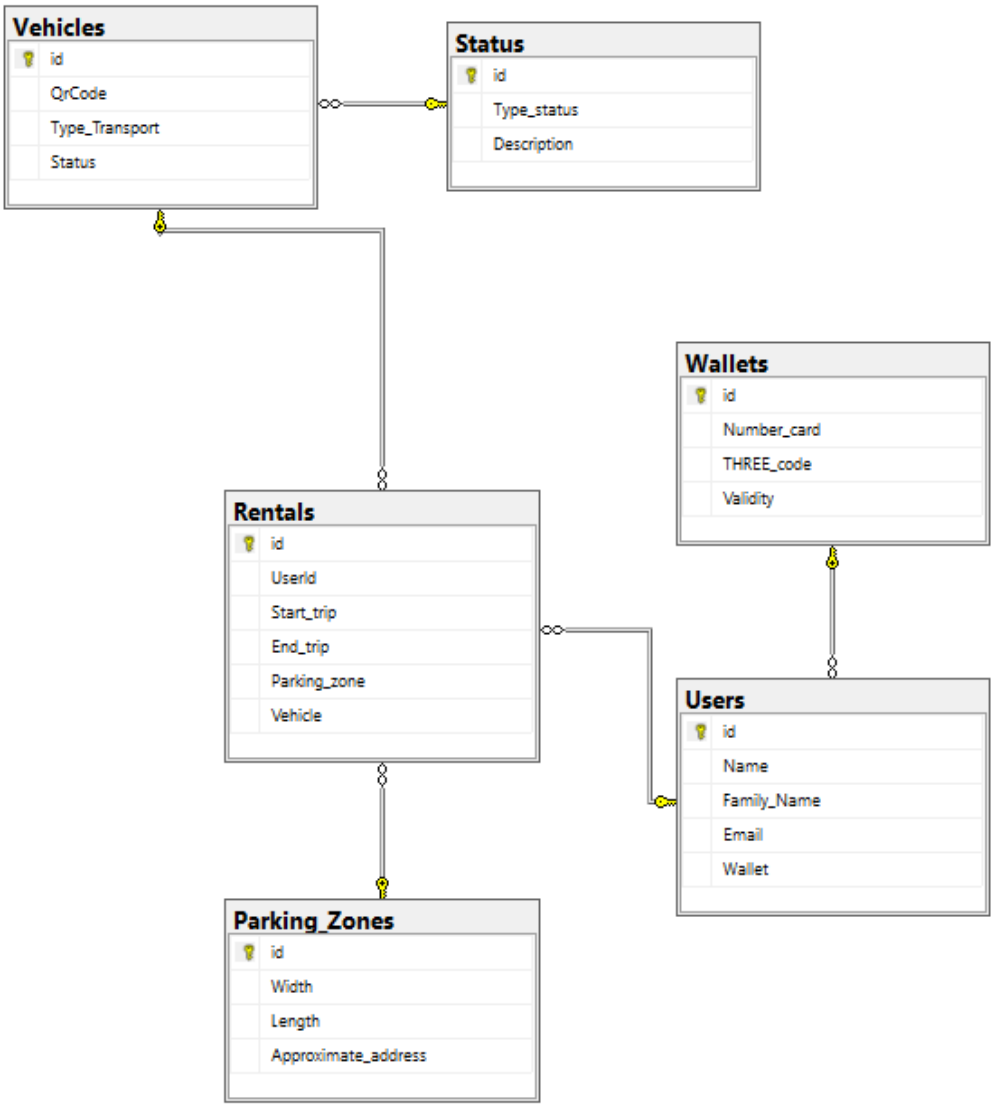
Кошелек	
id	INT NOT NULL PRIMARY KEY IDENTITY
Номер карты	INT(16) NOT NULL
3-х значных код	INT(3) NOT NULL
Срок действия	INT(4) NOT NULL

Зона парковки	
id	INT NOT NULL PRIMARY KEY IDENTITY
Широта	DECIMAL(8,6) NOT NULL
Долгота	DECIMAL(8,6) NOT NULL
Примерный адрес	NVARCHAR(255) NOT NULL

Реализация таблиц в СУБД

Файл Rental_system.sql

Диаграмма базы данных

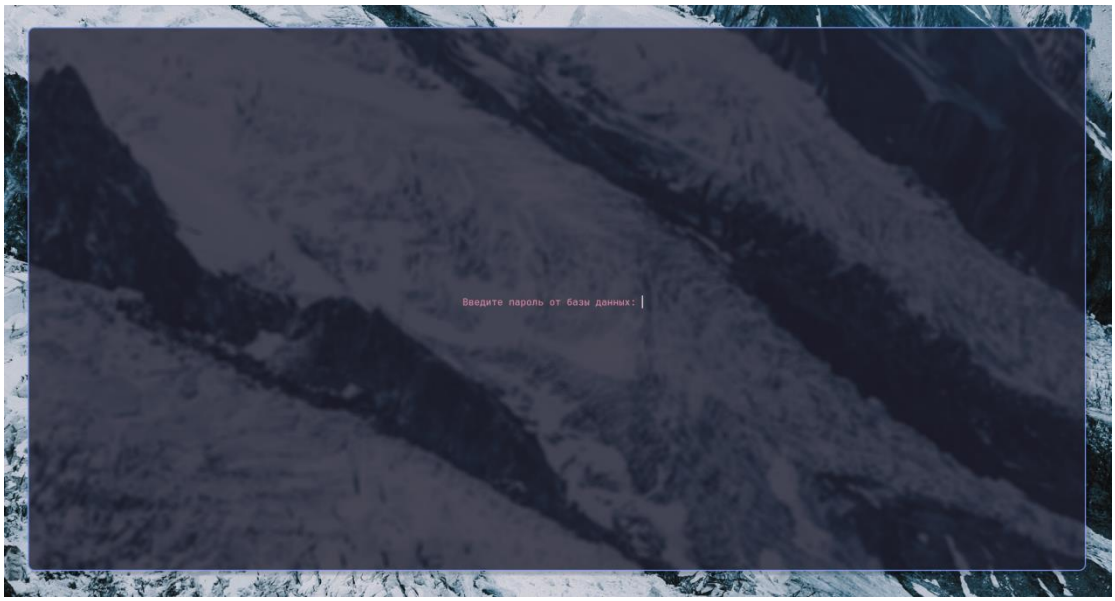


Между таблицами установлена связь через внешние ключи, они обеспечивают целостность данных.
Сама же диаграмма, отображает эти связи, где один пользователь может совершать несколько аренд, а одно транспортное средство может участвовать в различных арендах. Также у одного транспорта может быть несколько несколько статусов работы.

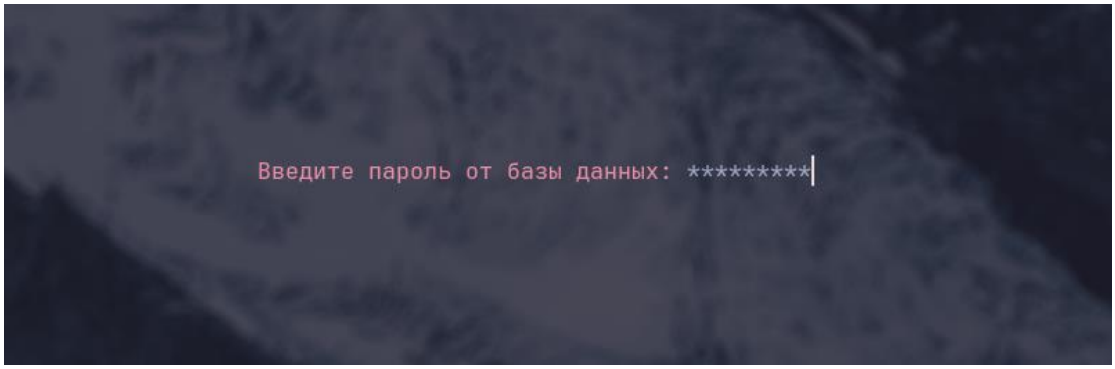
Документация

Вход в программу

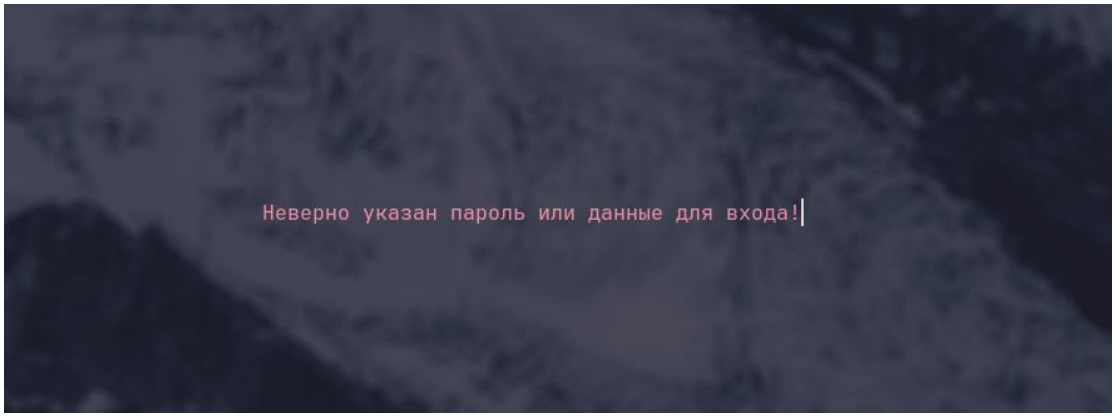
Как выглядит программа в терминале.



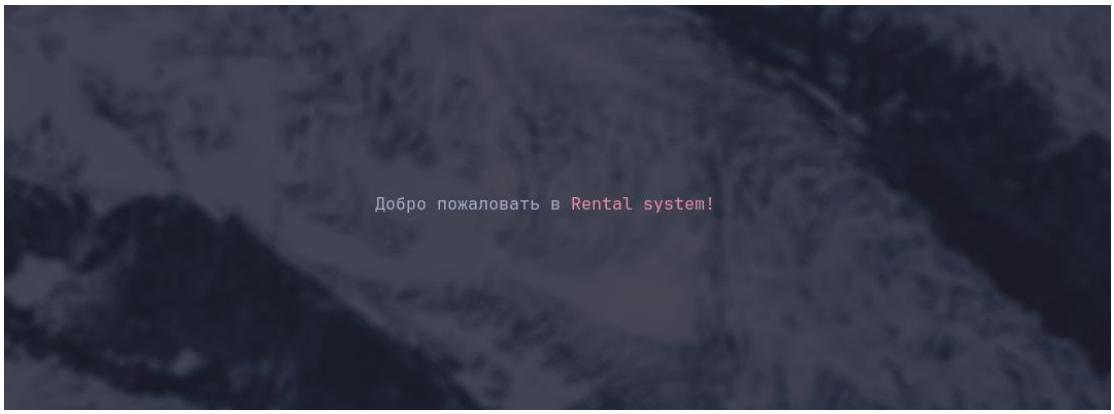
Вход в базу данных, для входа нужно ввести пароль, который показывается в звездочках.



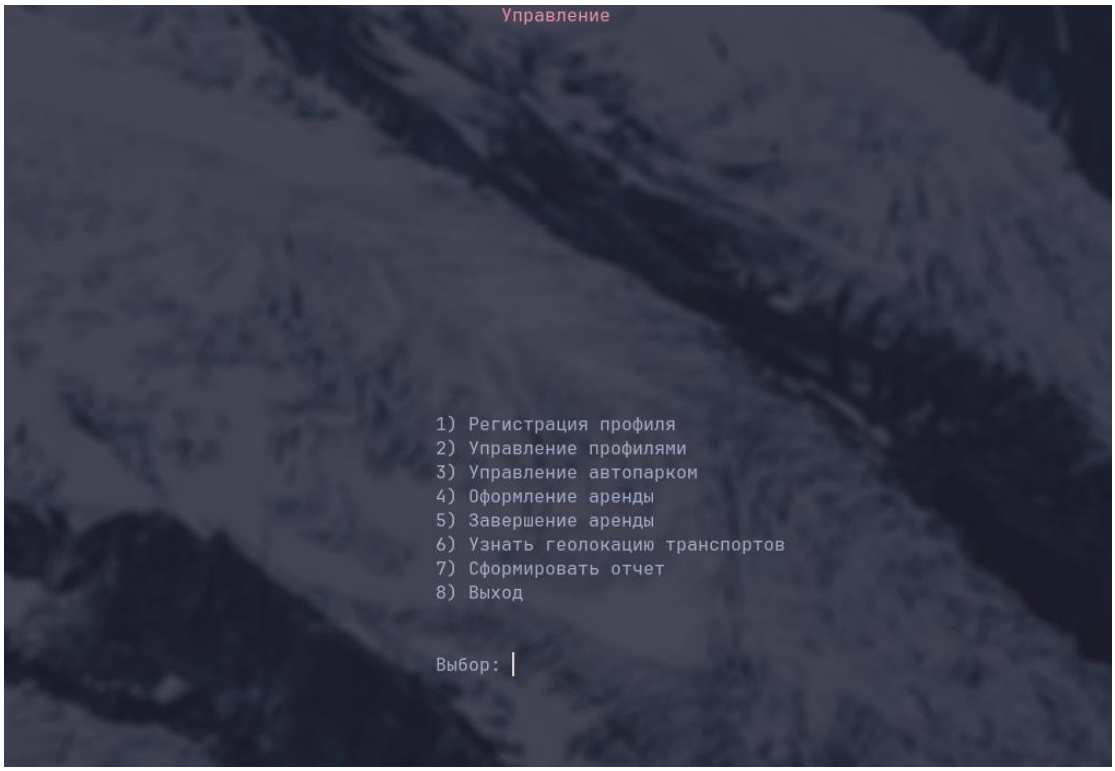
В случае неудачи входа выводит ошибку.



Если мы зашли, то показывает приветствие.



Главное меню управления

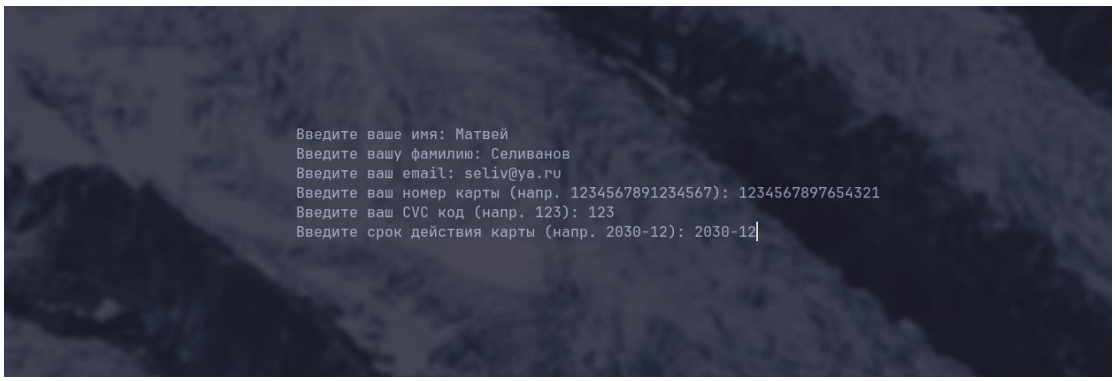


В этом меню осуществляется главное управление программой, в нем есть

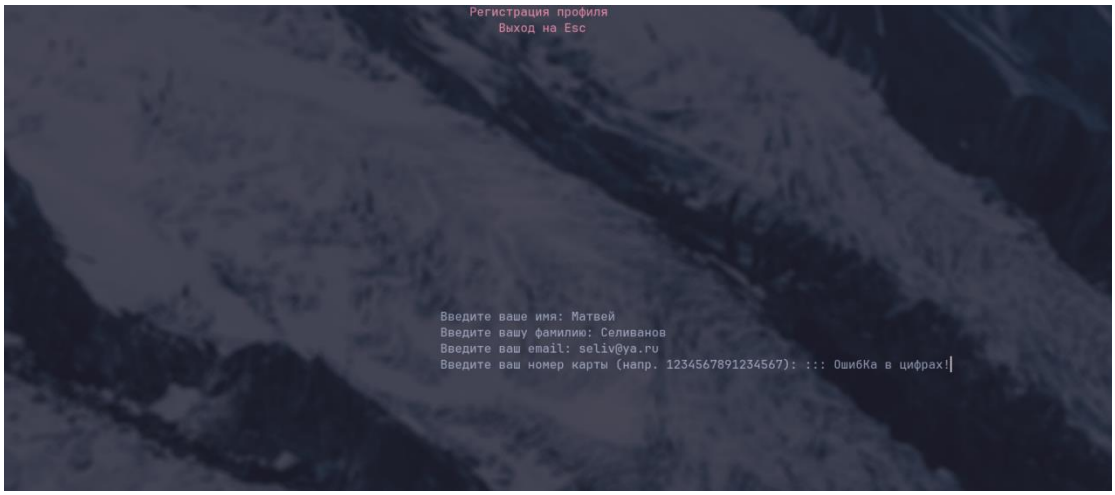
пункты:

1) Регистрация профиля
2) Управление профилями
3) Управление автопарком
4) Оформление аренды
5) Завершение аренды
6) Узнать геолокацию
7) Сформировать отчёт
8) Выход

Регистрация профиля

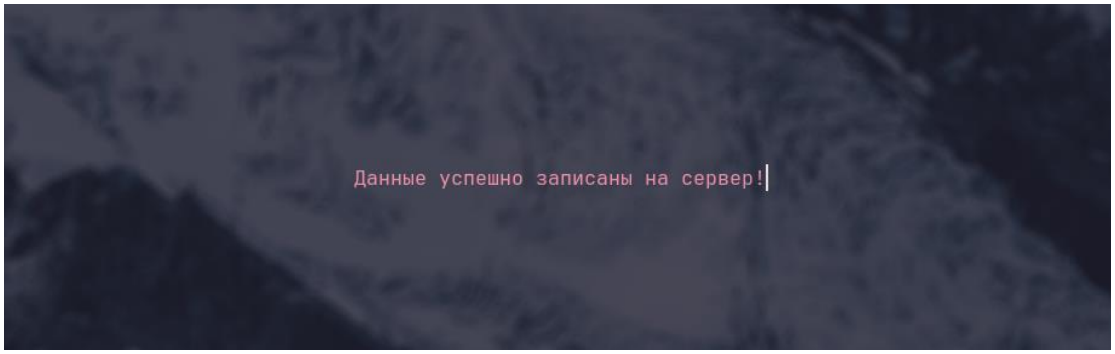


Открывается новое меню, где можно создать нового пользователя. В этом меню вводится его полная информация.
В случае неудачи ввода, программа выведет ошибку, и по новой запросит ввод.



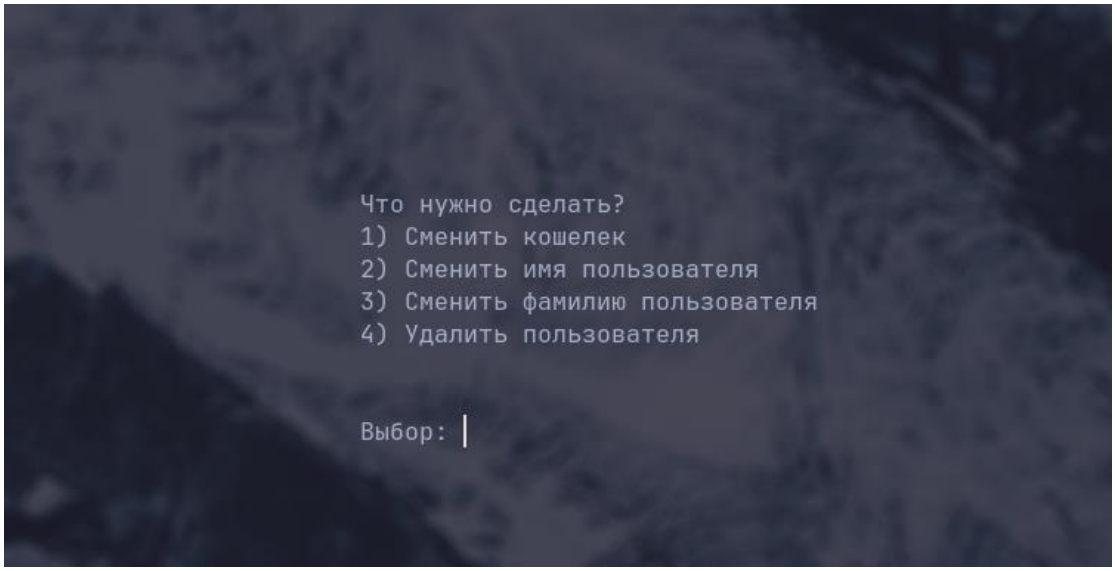
Также, в из каждого подменю можно выйти в самое главное меню, для этого нужно нажать “Esc”.

Если мы успешно ввели данные и они смогли загрузится на сервер, то программа оповестит об этом.

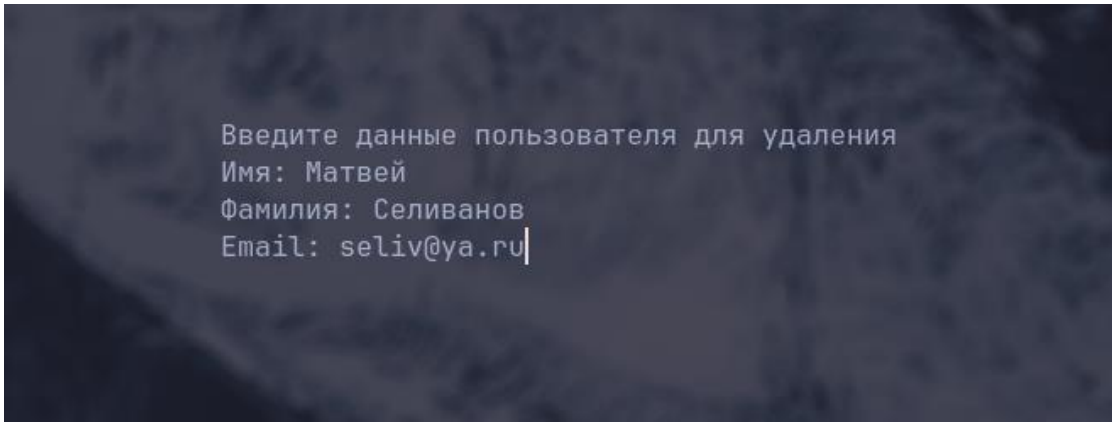


А дальше она перекинет в главное меню.

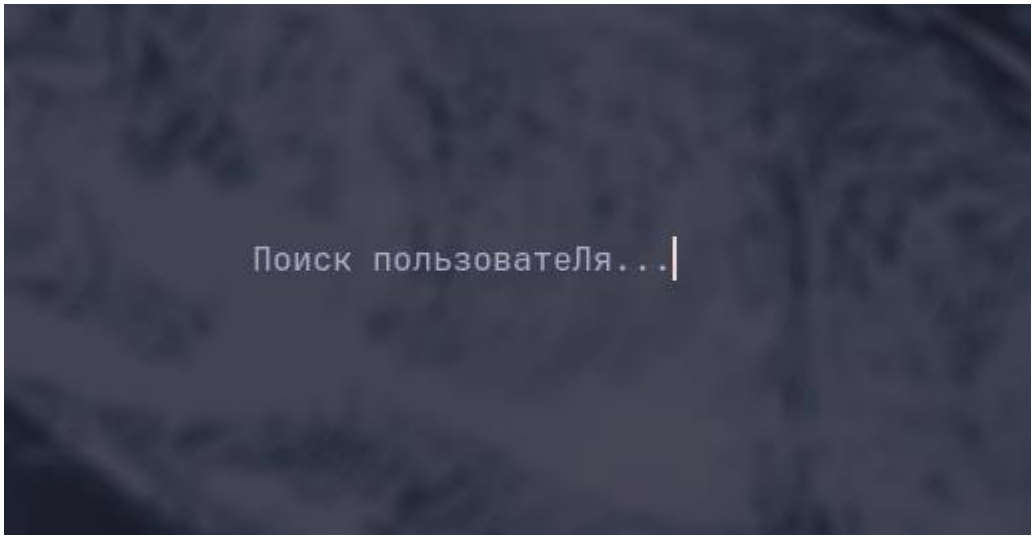
Управление профилем



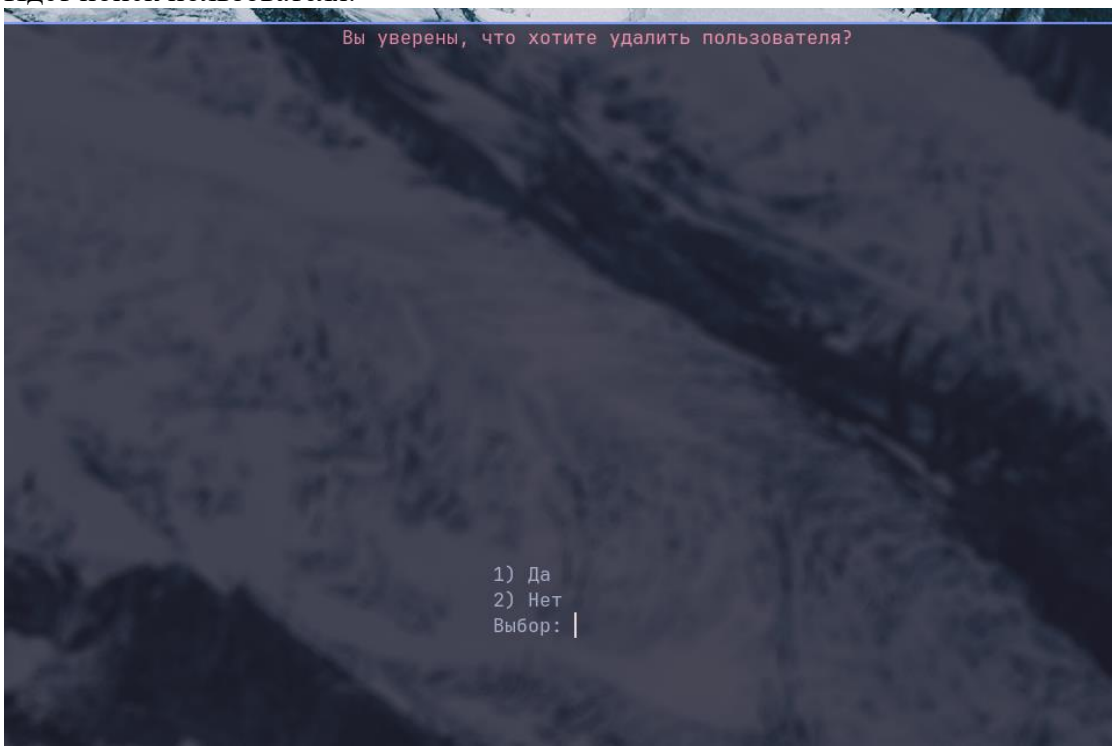
В этом подменю есть еще одно подменю, где есть выбор, что можно сделать. Возьмем к примеру 4 пункт.



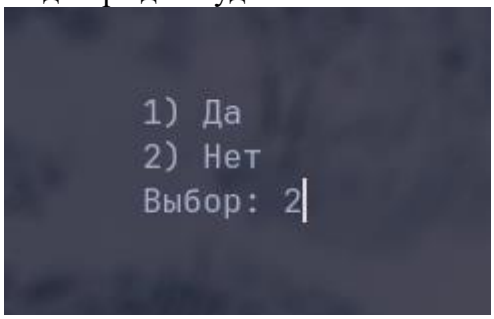
Вводим данные.

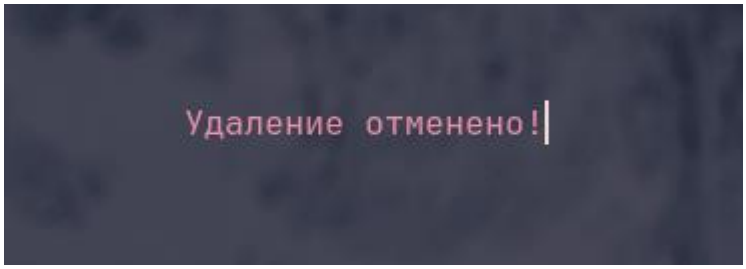


Идет поиск пользователя.

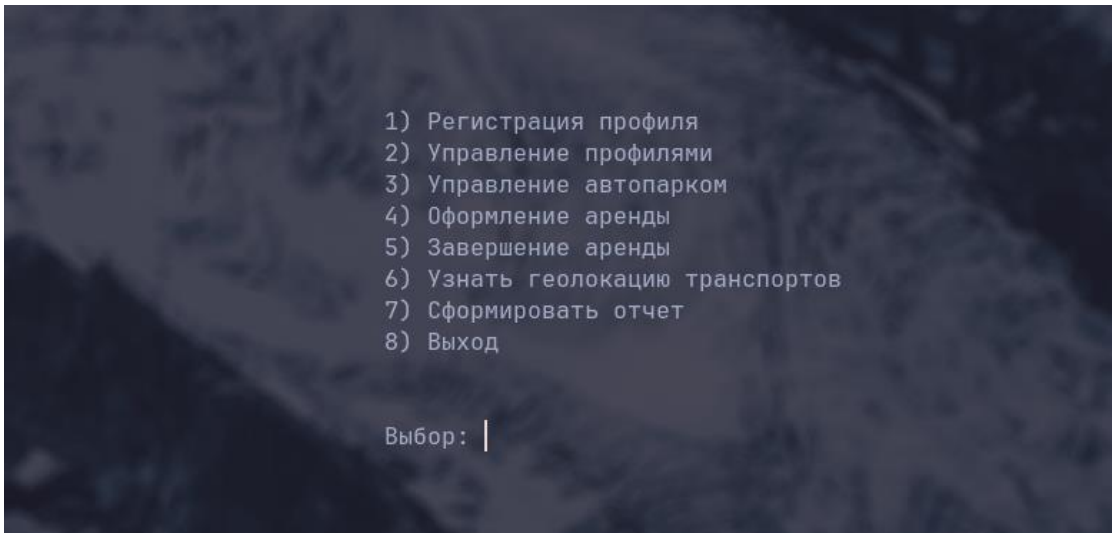


Подтверждаем удаление пользователя или нет.

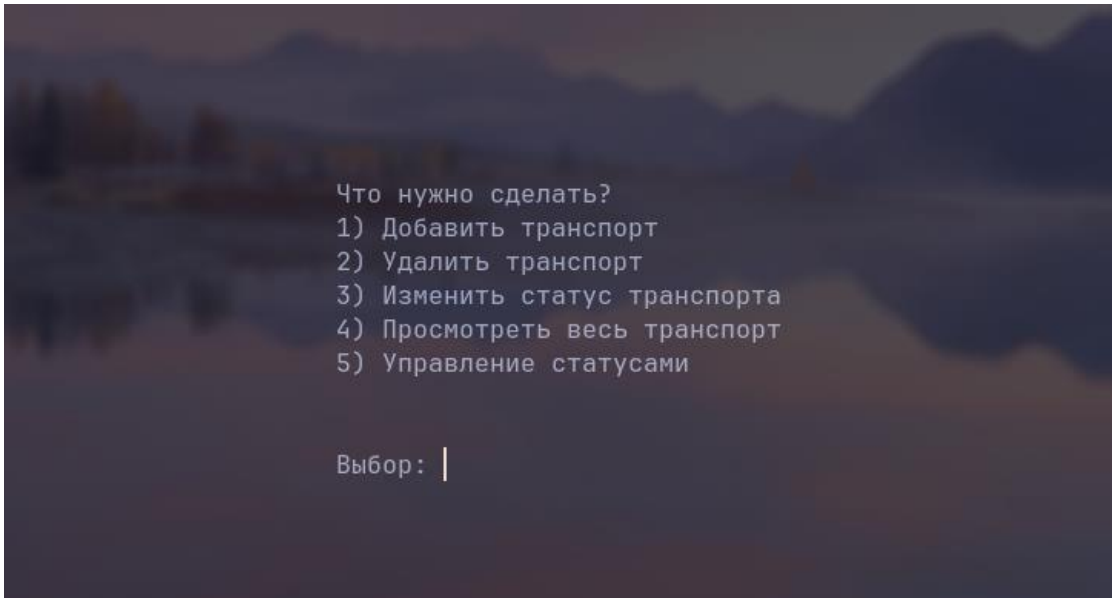




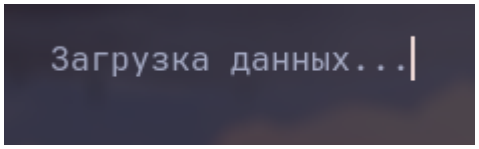
В случае отмены, подменю возвращает в главное меню программы.



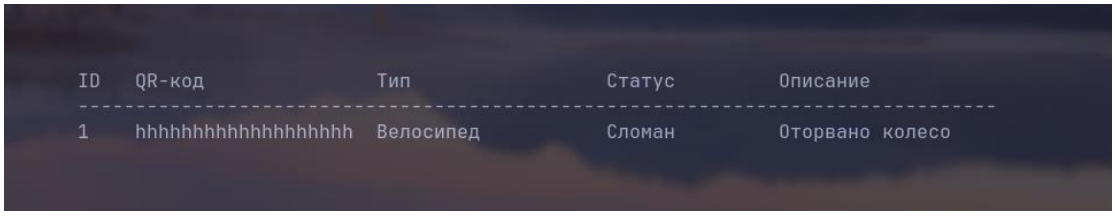
Управление автопарком



В этом подменю есть пункты:
1) Добавить транспорт
2) Удалить транспорт
3) Изменить статус транспорта
4) Посмотреть весь транспорт
5) Управление статусами
Выбираем 4 пункт.

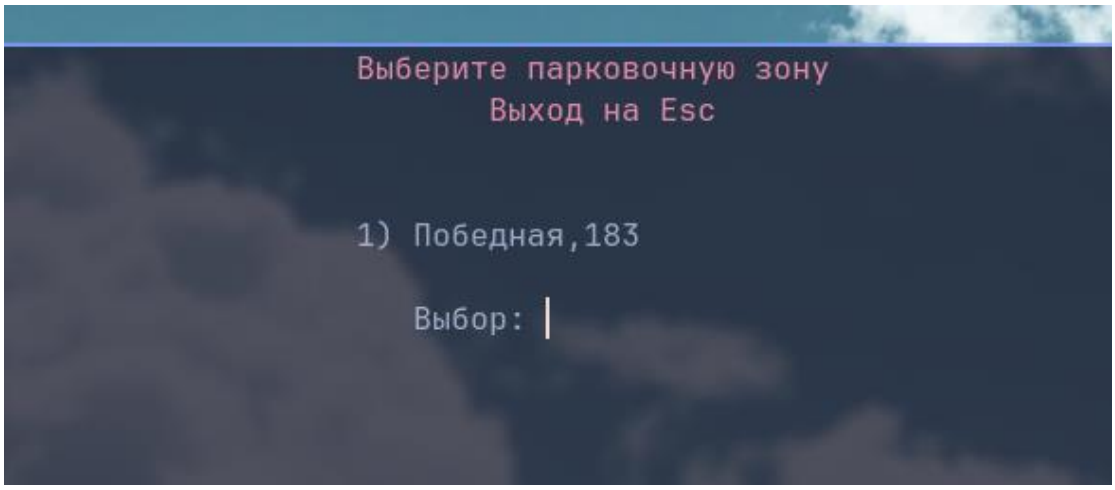


Загрузка.

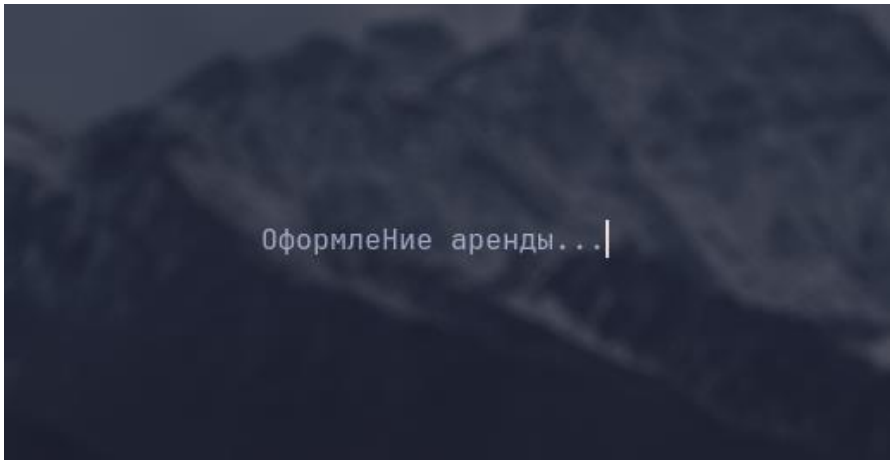


Показываются все транспорта, которые есть в базе данных.

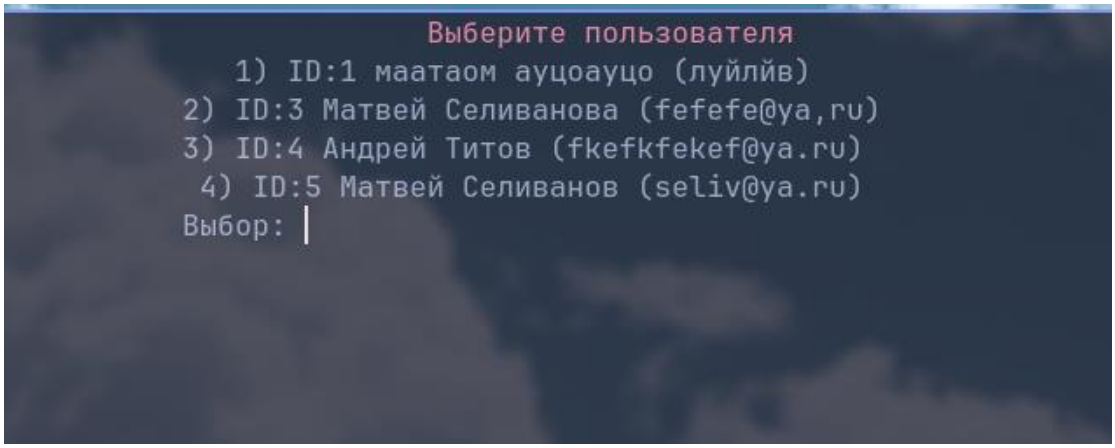
Оформление аренды



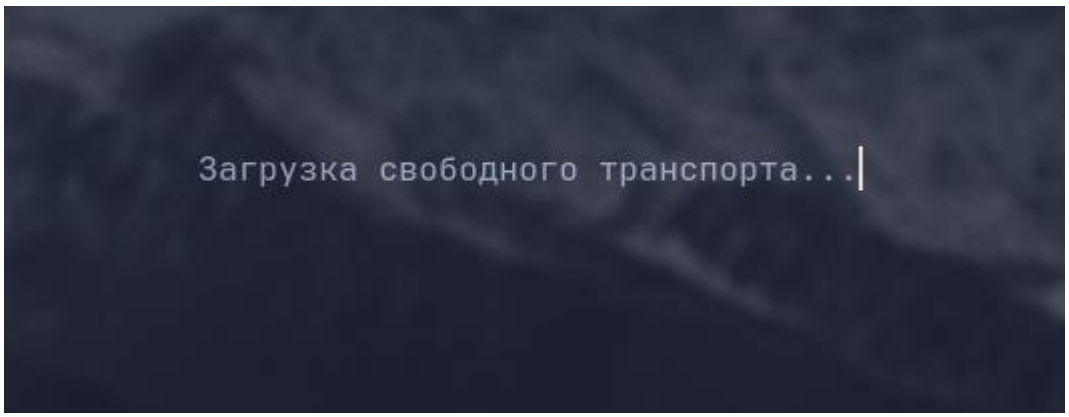
Нужно выбрать парковочную зону, где находится.



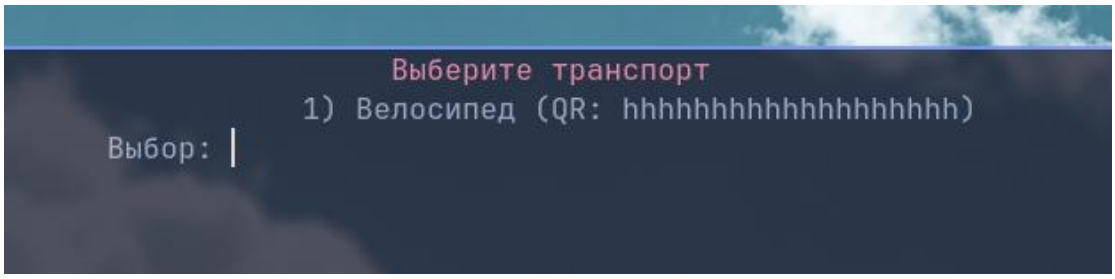
Загрузка следующего этапа.

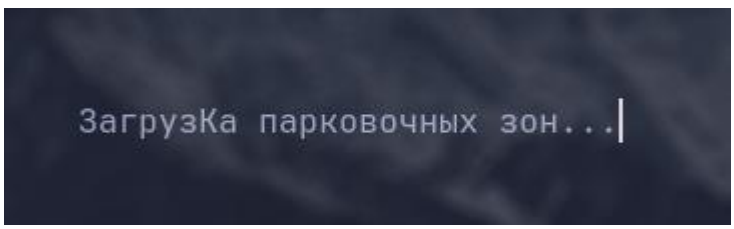


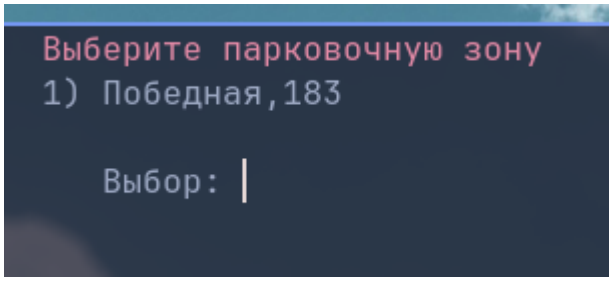
Выбираем пользователя.



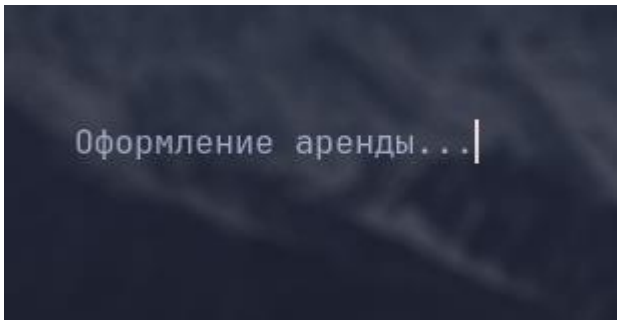
Выбираем свободный транспорт.



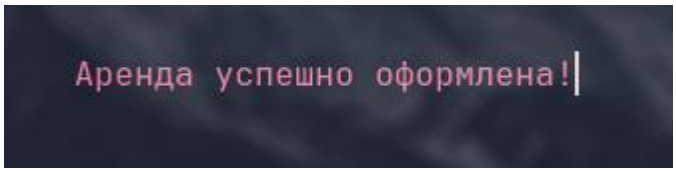




Дальше выбираем парковочную зону.

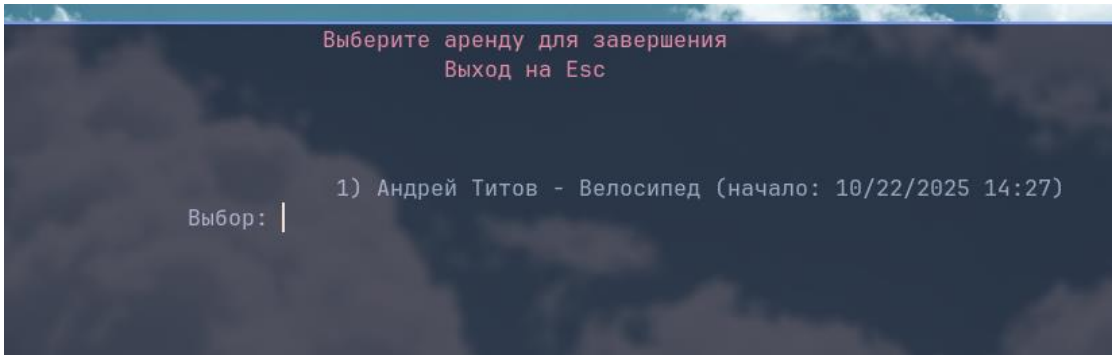


Идет оформление.

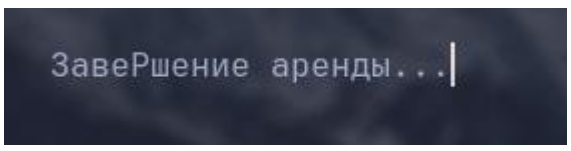


В случае успеха, программа выводит сообщение.

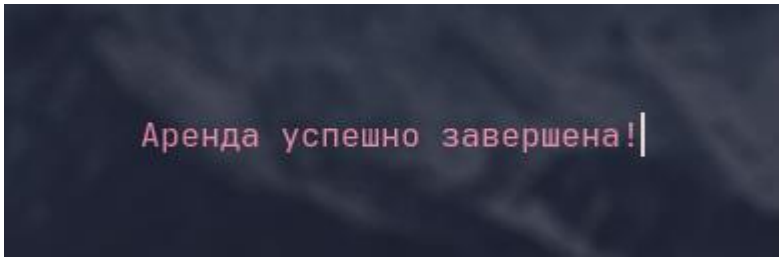
Завершение аренды



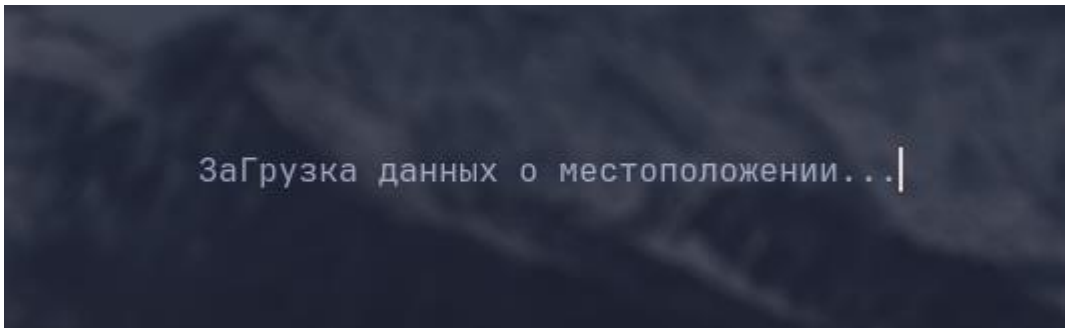
Выбираем пользователя, у которого активная аренда.

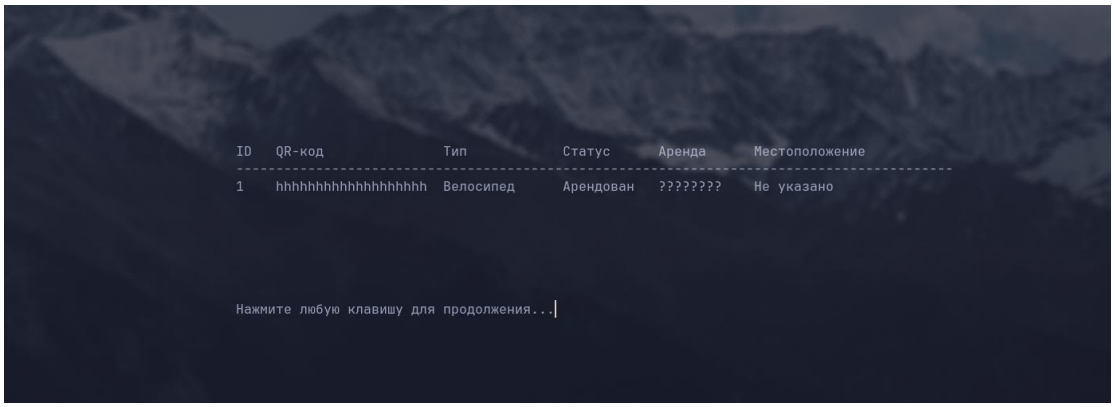


В случае успеха, программа выводит сообщение.



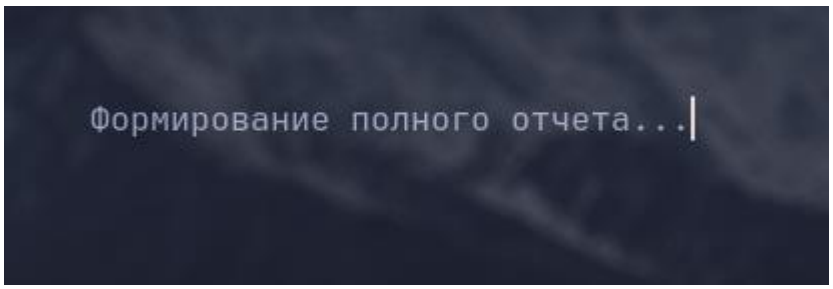
Геолокация транспорта





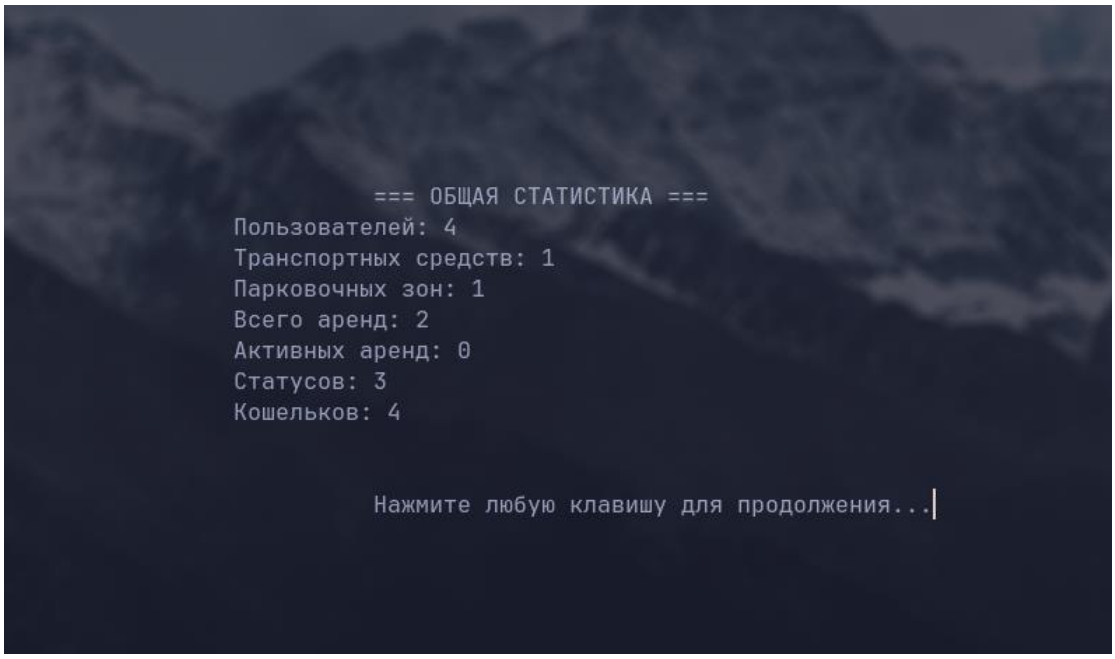
В данном выводе показывается вся таблица с геопозицией, где привязан весь транспорт.

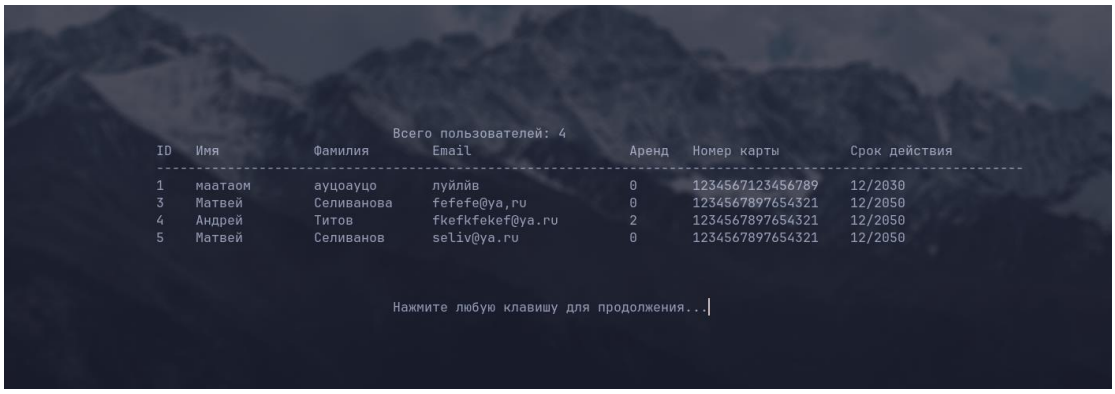
Формирование отчета

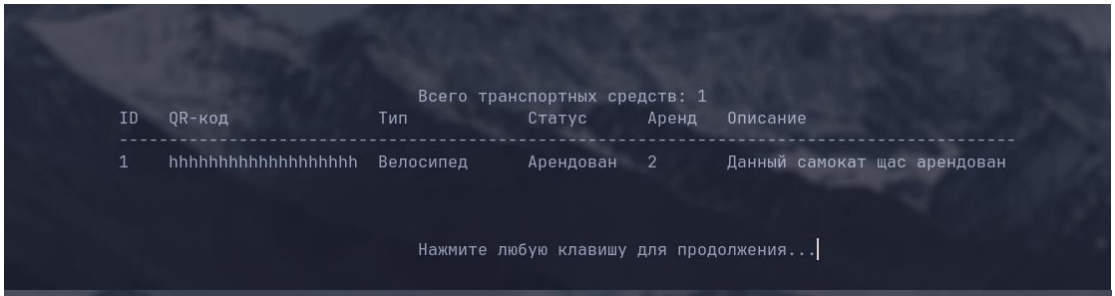


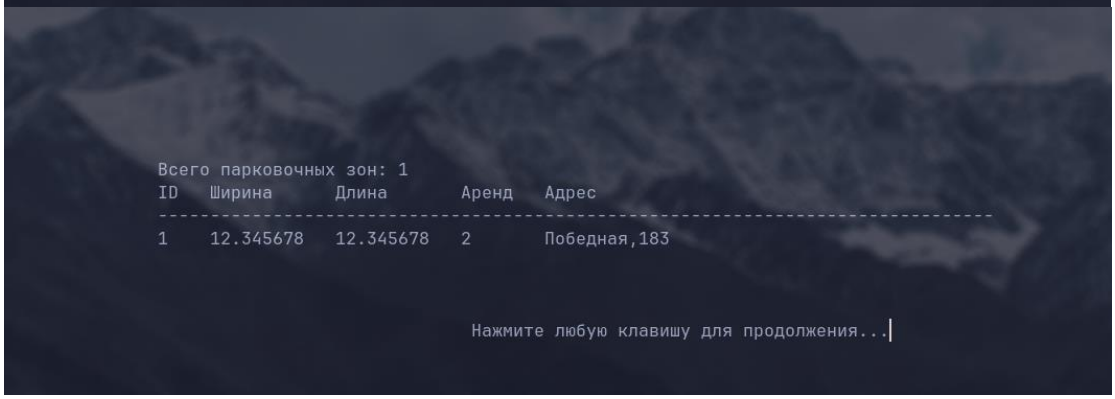
Загрузка.

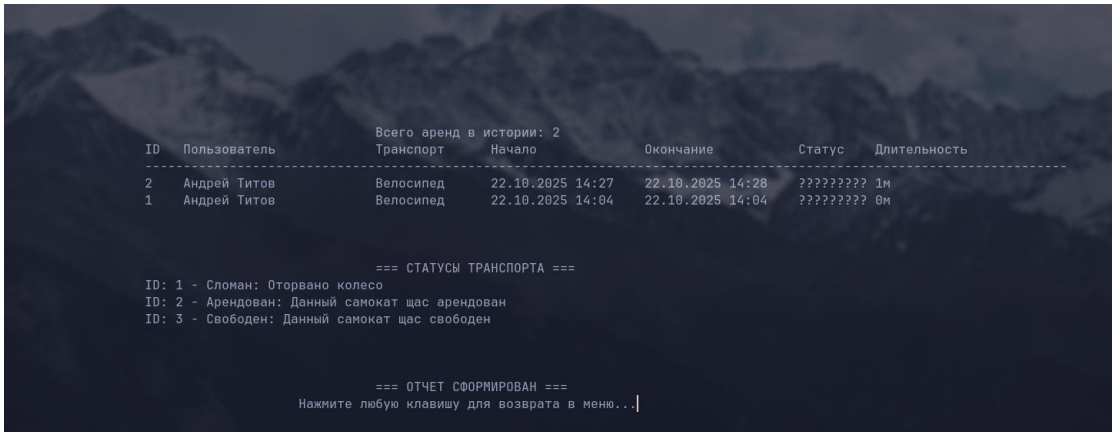
В данном подменю показывается вся информация по базе данных.





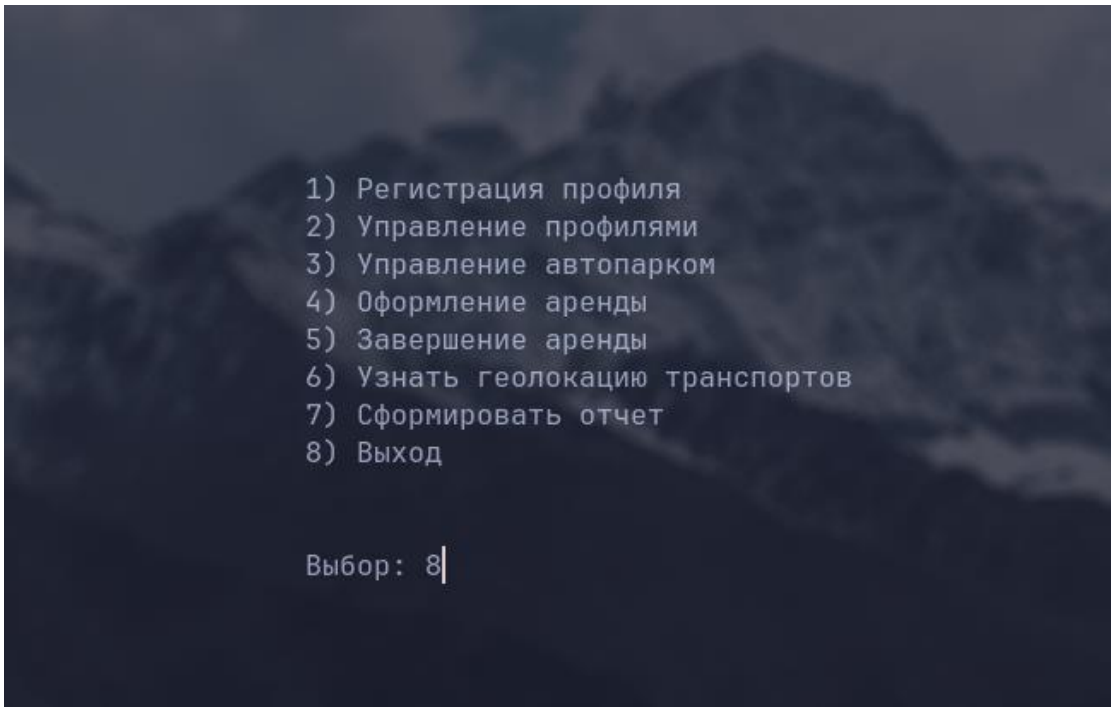






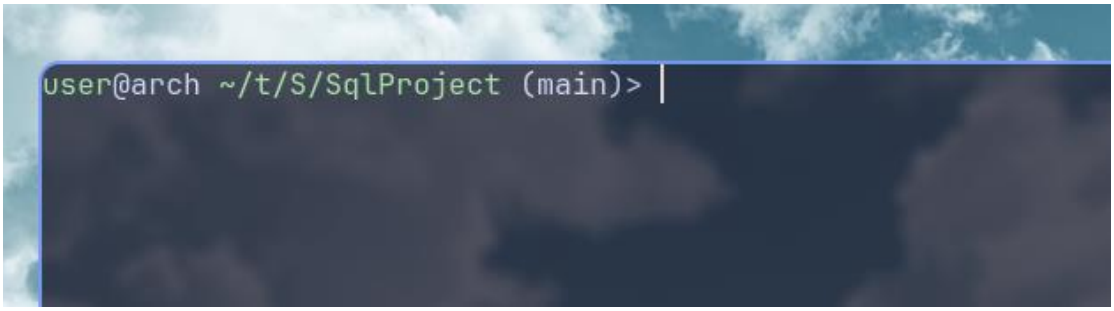
Завершение работы

Под конец работы нужно выбрать 8 пунк - выход.



База данных автоматически отсоединится и закроет все активные незавершенные процессы, а также попрощается.

После, произойдет очищение терминала, где запущена программа



Тестирование программы

Test Case 1: Успешная регистрация профиля

ID	Шаг	Действие	Ожидаемый результат	Фактический результат	Статус
1.1	Предусловия	База данных пуста	Система готова к регистрации	✅Система готова	✅PASS
1.2	Главное меню	Выбрать "1) Регистрация профиля"	Открывается форма регистрации	✅Форма отображена	PASS
1.3	Ввод данных	Ввести: Имя="Иван", Фамилия="Петров"	Данные принимаются	✅Данные приняты	PASS
1.4	Ввод email	Ввести: "ivan@mail.ru"	Email принимается	✅Email принят	PASS
1.5	Ввод номера карты	Ввести: "1234567890123456" (16 цифр)	Номер карты принимаетс	✅Карта принята	PASS

ID	Шаг	Действие	Ожидаемый результат	Фактический результат	Статус
			я		
1.6	Ввод CVC	Ввести: "123" (3 цифры)	CVC код принимается	✅CVC принят	PASS
1.7	Ввод срока действия	Ввести: "2025-12"	Дата принимается	✅Дата принята	PASS
1.8	Завершение регистрации	Подтвердить ввод	Сообщение об успешной регистрации	✅ "Загрузка данных на сервер" → "Успешно!"	PASS
1.9	Проверка БД: Users	Проверить таблицу Users	Новая запись с введенными данными	✅Запись создана	PASS
1.10	Проверка БД: Wallets	Проверить таблицу Wallets	Новая запись с данными карты	✅Запись создана	PASS
1.11	Проверка связи	Проверить связь Users-Wallet	Wallet ID корректно связан	✅Связь установлена	PASS

Результат тестирования: ✅УСПЕШНО

Test Case 2: Регистрация с некорректными данными

ID	Шаг	Действие	Ожидаемый результат	Фактический результат	Статус
2.1	Некорректный номер карты	Ввести: "123" (меньше 16 цифр)	Сообщение об ошибке	✓"Ошибка в цифрах!"	PASS
2.2	Некорректный CVC	Ввести: "12" (меньше 3 цифр)	Сообщение об ошибке	✓"Ошибка в цифрах!"	PASS
2.3	Некорректная дата	Ввести: "неправильная-дата"	Сообщение об ошибке	✓"Ошибка даты!"	PASS
2.4	Прерывание регистрации	Нажать Esc на любом этапе	Возврат в главное меню	✓Возврат выполнен	PASS

Результат тестирования: ✓УСПЕШНО

Сценарий 2: Управление профилями

Test Case 3: Смена кошелька пользователя

ID	Шаг	Действие	Ожидаемый результат	Фактический результат	Статус
3.1	Главное меню	Выбрать "2) Управление профилями"	Открывается подменю управления	✓Подменю отображено	✓PASS

ID	Шаг	Действие	Ожидаемый результат	Фактический результат	Статус
3.2	Выбор операции	Выбрать "1) Сменить кошелек"	Запрос данных пользователя	✅Форма смены кошелька открыта	✅PASS
3.3	Ввод данных	Ввести существующие: Имя="Иван", Email="ivan@mail.ru"	Поиск пользователя в БД	✅ Пользователь найден	✅PASS
3.4	Новые данные карты	Ввести: "9876543210987654"	Данные принимаются	✅Карта принята	✅PASS
3.5	Новый CVC	Ввести: "456"	CVC принимается	✅CVC принят	✅PASS
3.6	Новая дата	Ввести: "2026-06"	Дата принимается	✅Дата принята	✅PASS
3.7	Подтверждение	Завершить ввод	Сообщение об успешном обновлении	✅ "Успешно!"	✅PASS
3.8	Проверка БД: Wallets	Проверить обновленные данные	Данные кошелька изменены	✅Данные обновлены	✅PASS

Результат тестирования: ✅УСПЕШНО

Вывод

В данной программе было реализованы функции, такие как: регистрация профиля, управление профилем, управление автопарком, оформление аренды, завершение аренды, информация о геолокации транспорта и формирование отчета.
Эта программа получилась достаточно информативной, красивой и больших количеством проверки данных пользователя, перед выполнением запроса в базу данных.
Цель проекта была достигнуна, все функции по плану были добавлены.
Единственные недочеты, которые есть в программе, это иногда съежает текст, и не всегда автоматичски сменяется аренда с состояния “Занят” на “Свободный”, но в будущем, эта проблема будет решена.

Программный код

Файл: Program.cs

```
using System;
using System.Text;
using Dapper;
using Microsoft.Data.SqlClient;

namespace MyApp;


internal class Program
{
   private const string IP = "192.168.9.90";
   private const string DATABASE = "Rental_system";
   private const string USER = "sa";

   public class ConnectionString
   {
      private string Server;
      private string Database;
      private string User;
      private string Password;

      public ConnectionString(string server, string database, string user, string
password)
      {
         Server = server;
         Database = database;
         User = user;
         Password = password;
      }

      public override string ToString()
      {
         return
$"Server={Server};Database={Database};User={User};Password={Password};Trust
ServerCertificate=True;";
      }
   }


   static void Main(string[] args)
   {
      Console.Clear();

      foreach (var item in Animation.logo)
      {
         Animation.PrintRedText(item, true, false, 3, true, true);
```

```
        }
        Thread.Sleep(1000);
        Console.Clear();


        StringBuilder pass = new StringBuilder();
        Animation.PrintRedText("Введите пароль от базы данных: ", true, true, 50, true);

        int counter = 0;
        while (true)
        {
            var key = Console.ReadKey(intercept:true);
            if (key.Key == ConsoleKey.Enter)
                break;

            if(key.Key == ConsoleKey.Escape)
                continue;


            if (key.Key == ConsoleKey.Backspace && counter > 0)
            {
                pass.Remove(pass.Length - 1, 1);
                Console.Write("\b \b");
                counter--;

                continue;
            }

            if (key.Key != ConsoleKey.Backspace)
            {
                pass.Append(key.KeyChar.ToString());
                Console.Write("*");
                counter++;
            }

        }

        ConnectionString cs = new(IP, DATABASE, USER, pass.ToString());
        var con = new SqlConnection(cs.ToString());

        Animation.LoadingDatabase();


        try
        {
            con.Open();
        }
        catch (Exception)
        {
```

```
            Console.Clear();
            Animation.PrintRedText("Неверно указан пароль или данные для входа!",
true, true, 50);
            Console.Clear();
            return;
        }

        Animation.Welcome();

        Menu menu = new Menu(con);

        while (true)
        {
            Menu.StartMenu();
            if (!int.TryParse(Console.ReadLine(), out var input))
                input = Int32.MaxValue;


            switch ((CaseMenu)input)
            {
                case CaseMenu.RegistrationProfile:
                    var reg = menu.RegistrationProfile();
                    if (reg != null)
                    {
                        var result = reg.Upload(con);

                        if (result)
                        {
                            Animation.PrintRedText("Данные успешно записаны на сервер!",
true, true, 50);
                        }
                        else
                        {
                            Animation.PrintRedText("Данные не были записаны на сервер!",
true, true, 50);
                        }
                    }
                    break;
                case CaseMenu.ManagementProfiles:
                    menu.ManagementProfileMenu(con);
                    break;
                case CaseMenu.ManagementPark:
                    menu.ManagementFleetMenu(con);
                    break;
                case CaseMenu.StartRent:
                    menu.StartRent();
                    break;
                case CaseMenu.EndRent:
                    menu.EndRent();
                    break;
```

```
            case CaseMenu.FindGeo:
                menu.FindGeo();
                break;
            case CaseMenu.Report:
                menu.GenerateReport();
                break;
            case CaseMenu.Leave:
                con.Close();
                Animation.Exit();
                return;
            default:
                Console.Clear();
                Animation.PrintRedText("Неверно указано!",true, true, 100);
                Console.Clear();
                break;
        }
    }
  }

}
```

Файл: CaseMenu.cs

```
namespace MyApp;

public enum CaseMenu
{
  RegistrationProfile = 1,
  ManagementProfiles = 2,
  ManagementPark = 3,
  StartRent = 4,
  EndRent = 5,
  FindGeo = 6,
  Report = 7,
  Leave = 8
}

public enum CaseChangeProfile
{
  Wallet = 1,
  Name = 2,
  FamilyName = 3,
  Delete = 4
}

public enum CaseFleetManagement
{
  AddVehicle = 1,
  RemoveVehicle = 2,
  ChangeStatus = 3,
```

```
    ViewAll = 4,
    ManageStatuses = 5,
    AddParkingZone = 6
}

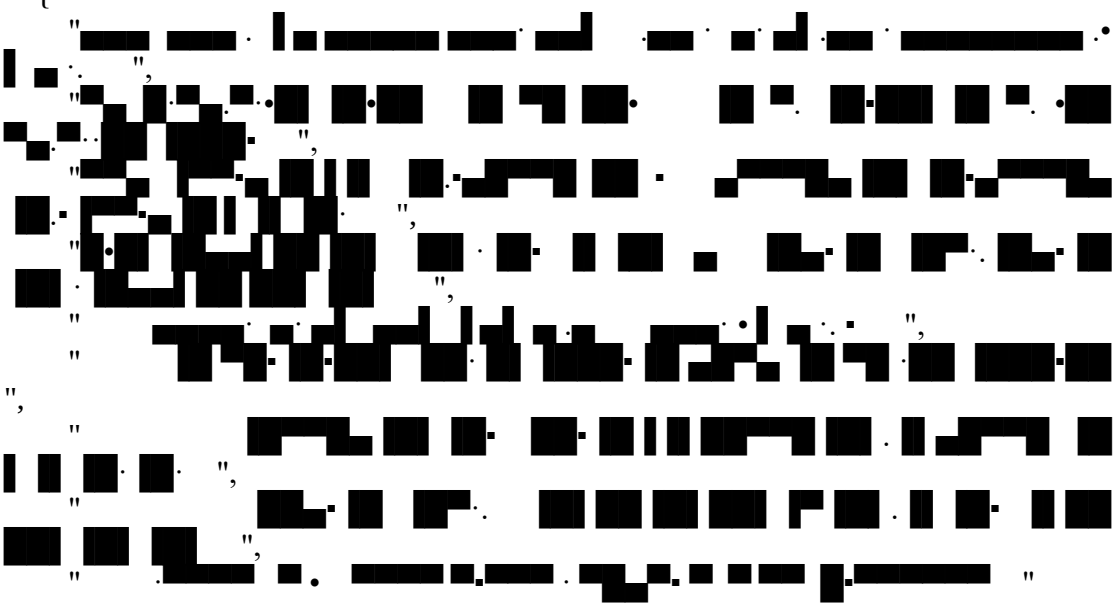
public enum CaseStatusManagement
{
    AddStatus = 1,
    ViewStatuses = 2,
    ViewParkingZones = 3,
    Back = 4,
}
```

Файл: Animation.cs

```
using System.Text;

namespace MyApp;

public class Animation
{

    public static List<string> logo = new List<string>()
    {
        "

    };
    public static void Welcome()
    {
        Console.Clear();
        Console.ForegroundColor = ConsoleColor.White;
        string welcome = "Добро пожаловать в Rental system!";

        for (int i = 0; i < Console.WindowHeight/2; i++)
            Console.WriteLine();
```

```
        Console.SetCursorPosition((Console.WindowWidth - welcome.Length) / 2,
Console.CursorTop);

        for (int i = 0; i < welcome.Length; i++)
        {
            if (welcome[i] == 'R')
                Console.ForegroundColor = ConsoleColor.Red;

            Console.Write(welcome[i]);
            Thread.Sleep(70);

            Console.Beep();
        }

        Console.ForegroundColor = ConsoleColor.White;
        Console.WriteLine();
        Thread.Sleep(2000);
        Console.Clear();
    }

    public static void PrintRedText(string text, bool center = false, bool centerTerminal
= false, int durability = 0, bool fastEnd = false, bool newLine = false)
    {
        if(centerTerminal)
        {
            for (int i = 0; i < Console.WindowHeight/2; i++)
                Console.WriteLine();
        }

        if (center)
        {
            int left = (Console.WindowWidth - text.Length) / 2;
            if (left < 0) left = 0;
            if (left >= Console.WindowWidth) left = Console.WindowWidth - 1;
            Console.SetCursorPosition(left, Console.CursorTop);
        }

        Console.ForegroundColor = ConsoleColor.Red;

        for (int i = 0; i < text.Length; i++)
        {
            Console.Write(text[i]);
            Thread.Sleep(durability);
        }

        if (!fastEnd)
            Thread.Sleep(1000);

        if(newLine)
```

```
            Console.WriteLine();
        Console.ForegroundColor = ConsoleColor.White;
    }

    public static void PrintCenterTerminal(string text, bool centerTerminal = false)
    {
        if (centerTerminal)
            for (int i = 0; i < Console.WindowHeight/2; i++)
                Console.WriteLine();


        Console.SetCursorPosition((Console.WindowWidth - text.Length) / 2,
Console.CursorTop);

        Console.WriteLine(text);
    }

    public static void PrintSetCursor(string text, int length, bool center = false, int
durabily = 0)
    {
        if(center)
        {
            for (int i = 0; i < Console.WindowHeight/2 - durabily/2; i++)
                Console.WriteLine();
        }

        int left = (Console.WindowWidth - length) / 2;
        if (left < 0) left = 0;
        if (left >= Console.WindowWidth) left = Console.WindowWidth - 1;
        Console.SetCursorPosition(left, Console.CursorTop);
        Console.Write(text);
    }

    public static void Exit()
    {
        Console.Clear();
        Thread.Sleep(500);
        PrintRedText("Прощай!", true, true, 100);
        Console.Clear();
    }

    public static void LoadingDatabase()
    {
        Console.Clear();
        StringBuilder str = new("Загрузка базы данных");

        PrintCenterTerminal(str.ToString(), true);

        for (int k = 0; k < 4; k++)
        {
```

```
            for (int i = 1; i < str.Length;i++)
            {
                str[i] = char.Parse(str[i].ToString().ToUpper());
                Thread.Sleep(50);
                Console.Clear();

                PrintCenterTerminal(str.ToString(), true);
                str[i] = char.Parse(str[i].ToString().ToLower());
            }
            str.Append('.');
        }

    }

    public static void AnimationText(string text, bool deleteWord = false, int cursorPosition = 0, int sizeOldText = 0, bool center = false)
    {
        StringBuilder str = new(text);

        int getCursor = cursorPosition;

        if (!center)
        {
            Console.SetCursorPosition(getCursor, Console.CursorTop - 1);
            for (int j = 0; j < str.Length + 4 + sizeOldText; j++)
            {
                Console.Write(' ');
            }
        }


        for (int k = 0; k < 6; k++)
        {
            for (int i = 1; i < str.Length;i++)
            {
                str[i] = char.Parse(str[i].ToString().ToUpper());

                if (!center)
                {
                    Console.SetCursorPosition(getCursor, Console.CursorTop);
                    Console.Write("::: " + str);
                    Thread.Sleep(50);
                }
                else
                {
                    for (int z = 0; z < Console.WindowHeight/2; z++)
                        Console.WriteLine();

                    Console.SetCursorPosition((Console.WindowWidth - text.Length) / 2, Console.CursorTop);
```

```
                Console.Write(str);
                Thread.Sleep(50);
                Console.Clear();
            }



            str[i] = char.Parse(str[i].ToString().ToLower());

        }
    }

    if (deleteWord && !center)
    {
        Console.SetCursorPosition(getCursor, Console.CursorTop);
        for (int j = 0; j < str.Length + 4 + sizeOldText; j++)
        {
            Console.Write(' ');
        }
        Console.SetCursorPosition(getCursor, Console.CursorTop);
    }

  }
}
```

Файл: Menu.cs

```
using System.ComponentModel.Design;
using System.Data;
using System.Data.SqlTypes;
using System.Text;
using Dapper;
using Microsoft.Data.SqlClient;

namespace MyApp;

public class Menu
{
  private bool exit = false;

  public SqlConnection conn {private get; set;}
  interface IUploadInServer
  {
    public bool Upload(SqlConnection connection);
  }

  public Menu(SqlConnection con)
    => conn = con;
```

```
private string ReadLineWithCancel()
{
    var input = new StringBuilder();
    while (true)
    {

        if (Console.KeyAvailable)
        {
            var key = Console.ReadKey(true);

            if (key.Key == ConsoleKey.Enter)
            {
                Console.WriteLine();
                return input.ToString();
            }
            else if (key.Key == ConsoleKey.Backspace && input.Length > 0)
            {
                input.Remove(input.Length - 1, 1);
                Console.Write("\b \b");
            }
            else if (!char.IsControl(key.KeyChar))
            {
                input.Append(key.KeyChar);
                Console.Write(key.KeyChar);
            }
            else if (key.Key == ConsoleKey.Escape)
            {
                exit = true;
                return string.Empty;
            }
        }
        else
        {
            Thread.Sleep(50);
        }
    }
}

public class AddParkingZone : IUploadInServer
{
    public decimal Width { private get; set; }
    public decimal Length { private get; set; }
    public string ApproximateAddress { private get; set; }

    public AddParkingZone() { }

    public bool Upload(SqlConnection connection)
    {
```

```
        try
        {
            string sql = "INSERT INTO Parking_Zones (Width, Length, Approximate_address) VALUES (@Width, @Length, @ApproximateAddress)";
            using (SqlCommand command = new SqlCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@Width", Width);
                command.Parameters.AddWithValue("@Length", Length);
                command.Parameters.AddWithValue("@ApproximateAddress", ApproximateAddress);

                int affectedRows = command.ExecuteNonQuery();
                return affectedRows > 0;
            }
        }
        catch (Exception ex)
        {
            Console.Clear();
            Animation.PrintRedText($"Ошибка добавления парковочной зоны: {ex.Message}", true, true, 50);
            return false;
        }
    }
}

public class UserRegistaryForm : IUploadInServer
{
    public string name { private get; set; }
    public string familyName { private get; set; }
    public string email { private get; set; }
    public Decimal numberCard { private get; set; }
    public int cvcCode { private get; set; }
    public DateTime Validity { private get; set; }

    public UserRegistaryForm(){}

    public override string ToString()
        => $"{name} {familyName} {email} {numberCard} {cvcCode} {Validity}";


    public bool Upload(SqlConnection connection)
    {
        try
        {
            string connectionString =
                "INSERT INTO Wallets (Number_card, THREE_code, Validity) VALUES (@numberCard, @cvcCode, @Validity)";

            using (SqlCommand command = new SqlCommand(connectionString, connection))
```

```
            {
                command.Parameters.AddWithValue("@cvcCode", cvcCode);
                command.Parameters.AddWithValue("@Validity", Validity);
                command.Parameters.AddWithValue("@numberCard", numberCard);

                command.ExecuteNonQuery();
            }

            int id = connection.Query("SELECT id FROM Wallets WHERE Number_card = @numberCards",
                new { @numberCards = numberCard }).First().id;

            connectionString =
                "INSERT INTO Users (Name, Family_Name, Email, Wallet) VALUES (@name, @familyName, @email, @Wallet)";

            using (SqlCommand command = new SqlCommand(connectionString, connection))
            {
                command.Parameters.AddWithValue("@name", name);
                command.Parameters.AddWithValue("@familyName", familyName);
                command.Parameters.AddWithValue("@email", email);
                command.Parameters.AddWithValue("@Wallet", id);

                command.ExecuteNonQuery();
            }
        }
        catch (Exception)
        {
            return false;
        }

        return true;
    }
}

public class ChangeWallet : IUploadInServer
{
    public string name;
    public string email;


    private int idWallet;

    public decimal numberCard { private get; set; }
    public int cvcCode {private get; set; }
    public DateTime Validity {private get; set;}

    public ChangeWallet() { }
```

```
        public bool SearchData(SqlConnection connection)
        {
            try
            {
                var result = connection.Query<string>("SELECT W.id FROM Wallets as
W, Users as U" +
                                        " WHERE U.Email = @email AND U.Wallet =
W.id AND U.Name = @name", new {@email = email, @name =
name}).First().ToString();

                idWallet = int.Parse(result);

            }
            catch (Exception)
            {
                return false;
            }

            return true;
        }
        public bool Upload(SqlConnection connection)
        {
            try
            {
                string connectStr = "UPDATE Wallets SET Number_card = @numberCard,
THREE_code = @three_code, Validity = @Validity WHERE Id = @id";
                using (SqlCommand command = new SqlCommand(connectStr,
connection))
                {
                    command.Parameters.AddWithValue("@numberCard", numberCard);
                    command.Parameters.AddWithValue("@three_code", cvcCode);
                    command.Parameters.AddWithValue("@Validity", Validity);
                    command.Parameters.AddWithValue("@id", idWallet);

                    command.ExecuteNonQuery();
                }

            }
            catch (Exception)
            {
                return false;
            }

            return true;
        }
    }

    public class ChangeNameOrFamilyName : IUploadInServer
    {
        public string name;
```

```
        public string email;
        public string familyName;

        private int idUser;
        private bool familyNameValid;
        public ChangeNameOrFamilyName() {}

        public bool SearchData(SqlConnection connection)
        {
            try
            {
                string result = connection
                    .Query<string>("SELECT * FROM Users WHERE Family_Name = @FamilyName AND Email = @Email", new {@FamilyName = this.familyName, @Email = this.email}).First()
                    .ToString();

                idUser = int.Parse(result);
            }
            catch (Exception)
            {
                return false;
            }

            return true;
        }

        public bool SearchDataFamily(SqlConnection connection)
        {
            try
            {
                string result = connection
                    .Query<string>("SELECT * FROM Users WHERE Name = @Name AND Email = @Email", new {@Name = this.name, @Email = this.email}).First()
                    .ToString();

                idUser = int.Parse(result);
            }
            catch (Exception e)
            {
                return false;
            }

            familyNameValid = true;
            return true;
        }

        public bool Upload(SqlConnection connection)
        {
            if (familyNameValid)
```

```
        {
            try
            {
                string connectStr = "UPDATE Users SET Family_Name = @familyName WHERE Id = @id";
                using (SqlCommand command = new SqlCommand(connectStr, connection))
                {
                    command.Parameters.AddWithValue("@familyName", familyName);
                    command.Parameters.AddWithValue("@id", idUser);
                    command.ExecuteNonQuery();
                }

            }
            catch (Exception)
            {
                return false;
            }

            return true;
        }


        try
        {
            string connectStr = "UPDATE Users SET Name = @name WHERE Id = @id";
            using (SqlCommand command = new SqlCommand(connectStr, connection))
            {
                command.Parameters.AddWithValue("@name", name);
                command.Parameters.AddWithValue("@id", idUser);

                command.ExecuteNonQuery();
            }

        }
        catch (Exception)
        {
            return false;
        }

        return true;
    }

}

public class RemoveUser : IUploadInServer
{
    public string name;
```

```
        public string email;
        public string familyName;

        private int idUser;
        private int idWallet;

        public RemoveUser() { }

        public bool SearchData(SqlConnection connection)
        {
            try
            {
                var result = connection.QueryFirstOrDefault<(int UserId, int WalletId)>(
                    @"SELECT U.id as UserId, W.id as WalletId
                      FROM Users as U
                      INNER JOIN Wallets as W ON U.Wallet = W.id
                      WHERE U.Name = @Name AND U.Family_Name = @FamilyName
AND U.Email = @Email",
                    new { Name = name, FamilyName = familyName, Email = email });

                if (result.UserId != 0)
                {
                    idUser = result.UserId;
                    idWallet = result.WalletId;
                    return true;
                }
            }
            catch (Exception ex)
            {
                Console.Clear();
                Console.WriteLine($"Ошибка поиска: {ex.Message}");
            }
            return false;
        }

        public bool CheckRentals(SqlConnection connection)
        {
            try
            {
                var result = connection.QueryFirstOrDefault<int?>(
                    "SELECT COUNT(*) FROM Rentals WHERE UserId = @UserId",
                    new { UserId = idUser });

                return result.HasValue && result.Value > 0;
            }
            catch (Exception ex)
            {
                return true;
            }
        }
```

```
    public bool Upload(SqlConnection connection)
    {
        using var transaction = connection.BeginTransaction();
        try
        {
            string deleteRentalsSql = "DELETE FROM Rentals WHERE UserId = @UserId";
            connection.Execute(deleteRentalsSql, new { UserId = idUser }, transaction);

            string deleteUserSql = "DELETE FROM Users WHERE id = @UserId";
            int userDeleted = connection.Execute(deleteUserSql, new { UserId = idUser }, transaction);

            string deleteWalletSql = "DELETE FROM Wallets WHERE id = @WalletId";
            int walletDeleted = connection.Execute(deleteWalletSql, new { WalletId = idWallet }, transaction);

            transaction.Commit();
            return userDeleted > 0 && walletDeleted > 0;
        }
        catch (Exception ex)
        {
            transaction.Rollback();
            Console.Clear();
            Console.WriteLine($"Ошибка удаления: {ex.Message}");
            return false;
        }
    }
}

public class AddVehicle : IUploadInServer
{
    public string QrCode { private get; set; }
    public string TypeTransport { private get; set; }
    public int StatusId { private get; set; }

    public AddVehicle() { }

    public bool Upload(SqlConnection connection)
    {
        try
        {
            string sql = "INSERT INTO Vehicles (QrCode, Type_Transport, Status) VALUES (@QrCode, @TypeTransport, @StatusId)";
            using (SqlCommand command = new SqlCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@QrCode", QrCode);
                command.Parameters.AddWithValue("@TypeTransport",
```

```
TypeTransport);
                command.Parameters.AddWithValue("@StatusId", StatusId);

                int affectedRows = command.ExecuteNonQuery();
                return affectedRows > 0;
            }
        }
        catch (Exception ex)
        {
            Console.Clear();
            Console.WriteLine($"Ошибка добавления: {ex.Message}");
            return false;
        }
    }
}

public class RemoveVehicle : IUploadInServer
{
    public string QrCode { private get; set; }
    private int vehicleId;

    public RemoveVehicle() { }

    public bool SearchData(SqlConnection connection)
    {
        try
        {
            var result = connection.QueryFirstOrDefault<int?>(
                "SELECT id FROM Vehicles WHERE QrCode = @QrCode",
                new { QrCode });

            if (result.HasValue)
            {
                vehicleId = result.Value;
                return true;
            }
            return false;
        }
        catch (Exception ex)
        {
            Console.Clear();
            Console.WriteLine($"Ошибка поиска: {ex.Message}");
            return false;
        }
    }

    public bool CheckRentals(SqlConnection connection)
    {
        try
        {
```

```
            var result = connection.QueryFirstOrDefault<int?>(
                "SELECT COUNT(*) FROM Rentals WHERE Vehicle = @VehicleId",
                new { VehicleId = vehicleId });

            return result.HasValue && result.Value > 0;
        }
        catch (Exception ex)
        {
            Console.Clear();
            Console.WriteLine($"Ошибка проверки аренд: {ex.Message}");
            return true;
        }
    }

    public bool Upload(SqlConnection connection)
    {
        try
        {
            string sql = "DELETE FROM Vehicles WHERE id = @VehicleId";
            using (SqlCommand command = new SqlCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@VehicleId", vehicleId);
                int affectedRows = command.ExecuteNonQuery();
                return affectedRows > 0;
            }
        }
        catch (Exception ex)
        {
            Console.Clear();
            Console.WriteLine($"Ошибка удаления: {ex.Message}");
            return false;
        }
    }
}

    public class ChangeVehicleStatus : IUploadInServer
    {
        public string QrCode { private get; set; }
        public int NewStatusId { private get; set; }
        private int vehicleId;

        public ChangeVehicleStatus() { }

        public bool SearchData(SqlConnection connection)
        {
            try
            {
                var result = connection.QueryFirstOrDefault<int?>(
                    "SELECT id FROM Vehicles WHERE QrCode = @QrCode",
                    new { QrCode });
```

```
            if (result.HasValue)
            {
                vehicleId = result.Value;
                return true;
            }
            return false;
        }
        catch (Exception ex)
        {
            Console.Clear();
            Console.WriteLine($"Ошибка поиска: {ex.Message}");
            return false;
        }
    }

    public bool Upload(SqlConnection connection)
    {
        try
        {
            string sql = "UPDATE Vehicles SET Status = @NewStatusId WHERE id =
@VehicleId";
            using (SqlCommand command = new SqlCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@NewStatusId", NewStatusId);
                command.Parameters.AddWithValue("@VehicleId", vehicleId);
                int affectedRows = command.ExecuteNonQuery();
                return affectedRows > 0;
            }
        }
        catch (Exception ex)
        {
            Console.Clear();
            Console.WriteLine($"Ошибка обновления: {ex.Message}");
            return false;
        }
    }
}

    public class AddStatus : IUploadInServer
    {
        public string TypeStatus { private get; set; }
        public string Description { private get; set; }

        public AddStatus() { }

        public bool Upload(SqlConnection connection)
        {
            try
            {
```

```
            string sql = "INSERT INTO Status (Type_status, Description) VALUES
(@TypeStatus, @Description)";
            using (SqlCommand command = new SqlCommand(sql, connection))
            {
                command.Parameters.AddWithValue("@TypeStatus", TypeStatus);
                command.Parameters.AddWithValue("@Description", Description);

                int affectedRows = command.ExecuteNonQuery();
                return affectedRows > 0;
            }
        }
        catch (Exception ex)
        {
            Console.Clear();
            Animation.PrintRedText($"Ошибка добавления статуса: {ex.Message}",
true, true, 50);
            return false;
        }
    }
}

public static void StartMenu()
{
    Console.Clear();
    Animation.PrintRedText("Управление", true, false, 70);

    Animation.PrintSetCursor("1) Регистрация профиля", 22, true, 9);
    Console.WriteLine();
    Animation.PrintSetCursor("2) Управление профилями", 22);
    Console.WriteLine();
    Animation.PrintSetCursor("3) Управление автопарком", 22);
    Console.WriteLine();
    Animation.PrintSetCursor("4) Оформление аренды", 22);
    Console.WriteLine();
    Animation.PrintSetCursor("5) Завершение аренды", 22);
    Console.WriteLine();
    Animation.PrintSetCursor("6) Узнать геолокацию транспортов", 22);
    Console.WriteLine();
    Animation.PrintSetCursor("7) Сформировать отчет", 22);
    Console.WriteLine();
    Animation.PrintSetCursor("8) Выход", 22);
    Console.WriteLine("\n\n");
    Animation.PrintSetCursor("Выбор: ", 22);
}

public UserRegistaryForm RegistrationProfile()
{
    exit = false;
    UserRegistaryForm user = new UserRegistaryForm();
```

```
        Console.Clear();

        Animation.PrintRedText("Регистрация профиля", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Введите ваше имя: ", 27, true, 6);
        user.name = ReadLineWithCancel();
        if (exit)
            return null;
        Animation.PrintSetCursor("Введите вашу фамилию: ", 27);
        user.familyName = ReadLineWithCancel();
        if (exit)
            return null;
        Animation.PrintSetCursor("Введите ваш email: ", 27);
        user.email = ReadLineWithCancel();
        if (exit)
            return null;
        Animation.PrintSetCursor("Введите ваш номер карты (напр. 1234567891234567): ", 27);

        Decimal numberCard;

        int cursorPosition = Console.GetCursorPosition().Left;

        while (true)
        {
            string temp = ReadLineWithCancel();;
            if (!Decimal.TryParse(temp, out numberCard))
            {
                if (exit)
                    return null;

                Animation.AnimationText("Ошибка ввода!", true, cursorPosition, temp.Length);
                continue;
            }

            if (numberCard.ToString().Length == 16)
                break;

            Animation.AnimationText("Ошибка в цифрах!", true, cursorPosition, temp.Length);
        }

        user.numberCard = numberCard;

        Animation.PrintSetCursor("Введите ваш CVC код (напр. 123): ", 27);

        cursorPosition = Console.GetCursorPosition().Left;
```

```
        int number;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (!int.TryParse(temp, out number))
            {
                if (exit)
                    return null;

                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }

            if (number.ToString().Length == 3)
                break;

            Animation.AnimationText("Ошибка в цифрах!", true, cursorPosition,
temp.Length);
        }

        user.cvcCode = number;

        Animation.PrintSetCursor("Введите срок действия карты (напр. 2030-12): ",
27);

        cursorPosition = Console.GetCursorPosition().Left;

        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit)
                return null;

            try
            {
                user.Validity = DateTime.Parse(temp);
                break;
            }
            catch (Exception)
            {
                Animation.AnimationText("Ошибка даты!", true, cursorPosition,
temp.Length);
            }

        }

        Console.Clear();
```

```
        Animation.AnimationText("Загрузка данных на сервер", false, 0, 0, true);

        return user;
    }


    public void ManagementProfileMenu(SqlConnection connection)
    {
        exit = false;

        Console.Clear();

        Animation.PrintRedText("Управление профилем", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Что нужно сделать?", 27, true, 6);
        Console.WriteLine();
        Animation.PrintSetCursor("1) Сменить кошелек", 27);
        Console.WriteLine();
        Animation.PrintSetCursor("2) Сменить имя пользователя", 27);
        Console.WriteLine();
        Animation.PrintSetCursor("3) Сменить фамилию пользователя", 27);
        Console.WriteLine();
        Animation.PrintSetCursor("4) Удалить пользователя", 27);
        Console.WriteLine("\n\n");
        Animation.PrintSetCursor("Выбор: ", 27);

        int cursorPosition = Console.GetCursorPosition().Left;
        int choice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if(exit)
                return;
            if (!int.TryParse(temp, out choice))
            {
                if (exit)
                    return;

                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }

            if (int.Parse(temp) >= 1 && int.Parse(temp) <= 4)
                break;


            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
```

```
temp.Length);
        }


        switch ((CaseChangeProfile)choice)
        {
            case CaseChangeProfile.Wallet:
                ManagementChangeWallet();
                break;
            case CaseChangeProfile.Name:
                ManagementChangeName();
                break;
            case CaseChangeProfile.FamilyName:
                ManagementChangeFamilyName();
                break;
            case CaseChangeProfile.Delete:
                ManagementRemoveUser();
                break;
            default:
                throw CheckoutException.Canceled;
        }
    }

    public bool ManagementRemoveUser()
    {
        exit = false;
        RemoveUser remover = new RemoveUser();

        Console.Clear();
        Animation.PrintRedText("Удаление пользователя", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Введите данные пользователя для удаления", 35,
true, 4);
        Console.WriteLine();
        Animation.PrintSetCursor("Имя: ", 35);
        remover.name = ReadLineWithCancel();
        if (exit)
            return false;
        Animation.PrintSetCursor("Фамилия: ", 35);
        remover.familyName = ReadLineWithCancel();
        if (exit)
            return false;
        Animation.PrintSetCursor("Email: ", 35);
        remover.email = ReadLineWithCancel();
        if (exit)
            return false;

        Console.Clear();
```

```
        Animation.AnimationText("Поиск пользователя...", false, 0, 0, true);

        if (!remover.SearchData(conn))
        {
            Animation.PrintRedText("Пользователь не найден!", true, true, 50);
            return false;
        }

        if (remover.CheckRentals(conn))
        {
            Animation.PrintRedText("У пользователя есть активные аренды! Удаление невозможно.", true, true, 50);
            return false;
        }

        Console.Clear();
        Animation.PrintRedText("Вы уверены, что хотите удалить пользователя?", true, false, 50);
        Console.WriteLine();
        Animation.PrintSetCursor("1) Да", 18, true, 3);
        Console.WriteLine();
        Animation.PrintSetCursor("2) Нет", 18);
        Console.WriteLine();
        Animation.PrintSetCursor("Выбор: ", 18);

        int cursorPosition = Console.GetCursorPosition().Left;
        int choice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit)
                return false;
            if (!int.TryParse(temp, out choice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition, temp.Length);
                continue;
            }
            if (choice == 1 || choice == 2)
                break;
            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition, temp.Length);
        }

        if (choice == 2)
        {
            Console.Clear();
            Animation.PrintRedText("Удаление отменено!", true, true, 50);
            return false;
        }
```

```
    Console.Clear();
    Animation.AnimationText("Удаление пользователя...", false, 0, 0, true);

    if (!remover.Upload(conn))
    {
        Console.Clear();
        Animation.PrintRedText("Ошибка удаления пользователя!", true, true, 50);
        return false;
    }
    else
    {
        Console.Clear();
        Animation.PrintRedText("Пользователь успешно удален!", true, true, 50);
        return true;
    }
}

public bool ManagementChangeFamilyName()
{
    exit = false;
    ChangeNameOrFamilyName changer = new ChangeNameOrFamilyName();

    Console.Clear();
    Animation.PrintRedText("Смена фамилии", true, false, 50);
    Console.WriteLine();
    Animation.PrintRedText("Выход на Esc", true, false, 50);

    Animation.PrintSetCursor("Введите данные", 18, true, 4);
    Console.WriteLine();
    Animation.PrintSetCursor("Имя пользователя: ", 18);
    changer.name = ReadLineWithCancel();
    if(exit)
        return false;
    Animation.PrintSetCursor("Почта пользователя: ", 18);
    changer.email = ReadLineWithCancel();
    if(exit)
        return false;

    Console.Clear();

    Animation.AnimationText("Запрос данных...", false, 0, 0, true);

    if (changer.SearchDataFamily(conn))
    {
        Console.Clear();
        Animation.PrintRedText("Ввод новых данных", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);
```

```
        Animation.PrintSetCursor("Введите ваше новую фамилию: ", 27, true, 1);
        changer.familyName = ReadLineWithCancel();


        Console.Clear();
        Animation.AnimationText("Загрузка данных на сервер...", false, 0, 0, true);

        if (!changer.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка загрузки данных!", true, true, 50);

            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Успешно!", true, true, 50);

            return true;
        }
    }
    else
    {
        Animation.PrintRedText("Такого пользователя нету!", true, true, 50);
        return false;
    }
}

public bool ManagementChangeName()
{
    exit = false;
    ChangeNameOrFamilyName changer = new ChangeNameOrFamilyName();

    Console.Clear();
    Animation.PrintRedText("Смена имени", true, false, 50);
    Console.WriteLine();
    Animation.PrintRedText("Выход на Esc", true, false, 50);

    Animation.PrintSetCursor("Введите данные", 18, true, 4);
    Console.WriteLine();
    Animation.PrintSetCursor("Фамилия пользователя: ", 18);
    changer.familyName = ReadLineWithCancel();
    if(exit)
        return false;
    Animation.PrintSetCursor("Почта пользователя: ", 18);
    changer.email = ReadLineWithCancel();
    if(exit)
        return false;
```

```
    Console.Clear();

    Animation.AnimationText("Запрос данных...", false, 0, 0, true);



    Console.Clear();
    if (changer.SearchData(conn))
    {
        Console.Clear();
        Animation.PrintRedText("Ввод новых данных", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Введите ваше новое имя: ", 27, true, 1);
        changer.name = ReadLineWithCancel();


        Console.Clear();
        Animation.AnimationText("Загрузка данных на сервер...", false, 0, 0, true);


        if (!changer.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка загрузки данных!", true, true, 50);

            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Успешно!", true, true, 50);

            return true;
        }
    }
    else
    {
        Animation.PrintRedText("Такого пользователя нету!", true, true, 50);
        return false;
    }
}
public bool ManagementChangeWallet()
{
    exit = false;
    ChangeWallet wallet = new ChangeWallet();
    Console.Clear();

    Animation.PrintRedText("Смена кошелька", true, false, 50);
```

```
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);


        Animation.PrintSetCursor("Введите данные", 18, true, 4);
        Console.WriteLine();
        Animation.PrintSetCursor("Имя пользователя: ", 18);
        wallet.name = ReadLineWithCancel();
        if(exit)
           return false;
        Animation.PrintSetCursor("Почта пользователя: ", 18);
        wallet.email = ReadLineWithCancel();
        if(exit)
           return false;

        Console.Clear();

        Animation.AnimationText("Запрос данных...", false, 0, 0, true);

        if (wallet.SearchData(conn))
        {
           Console.Clear();
           Animation.PrintRedText("Ввод данных кошелька", true, false, 50);
           Console.WriteLine();
           Animation.PrintRedText("Выход на Esc", true, false, 50);

           Animation.PrintSetCursor("Введите ваш номер карты (напр. 1234567891234567): ", 40, true, 3);

           Decimal numberCard;

           int cursorPosition = Console.GetCursorPosition().Left;

           while (true)
           {
              string temp = ReadLineWithCancel();;
              if (!Decimal.TryParse(temp, out numberCard))
              {
                 if (exit)
                    return false;

                 Animation.AnimationText("Ошибка ввода!", true, cursorPosition, temp.Length);
                 continue;
              }

              if (numberCard.ToString().Length == 16)
                 break;

              Animation.AnimationText("Ошибка в цифрах!", true, cursorPosition,
```

```
temp.Length);
        }

        wallet.numberCard = numberCard;

        Animation.PrintSetCursor("Введите ваш CVC код (напр. 123): ", 40);

        cursorPosition = Console.GetCursorPosition().Left;

        int number;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (!int.TryParse(temp, out number))
            {
                if (exit)
                    return false;

                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }

            if (number.ToString().Length == 3)
                break;

            Animation.AnimationText("Ошибка в цифрах!", true, cursorPosition,
temp.Length);
        }

        wallet.cvcCode = number;

        Animation.PrintSetCursor("Введите срок действия карты (напр. 2030-12): ",
40);

        cursorPosition = Console.GetCursorPosition().Left;

        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit)
                return false;

            try
            {
                wallet.Validity = DateTime.Parse(temp);
                break;
            }
            catch (Exception)
            {
```

```
                Animation.AnimationText("Ошибка даты!", true, cursorPosition,
temp.Length);
            }

        }

        Console.Clear();
        Animation.AnimationText("Загрузка данных на сервер...", false, 0, 0, true);


        if (!wallet.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка загрузки данных!", true, true, 50);

            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Успешно!", true, true, 50);
        }
        return true;
    }
    else
    {
        Animation.PrintRedText("Такого кошелька нету!", true, true, 50);
        return false;
    }
}

public void ManagementFleetMenu(SqlConnection connection)
{
    exit = false;

    Console.Clear();
    Animation.PrintRedText("Управление автопарком", true, false, 50);
    Console.WriteLine();
    Animation.PrintRedText("Выход на Esc", true, false, 50);

    Animation.PrintSetCursor("Что нужно сделать?", 27, true, 6);
    Console.WriteLine();
    Animation.PrintSetCursor("1) Добавить транспорт", 27);
    Console.WriteLine();
    Animation.PrintSetCursor("2) Удалить транспорт", 27);
    Console.WriteLine();
    Animation.PrintSetCursor("3) Изменить статус транспорта", 27);
    Console.WriteLine();
    Animation.PrintSetCursor("4) Просмотреть весь транспорт", 27);
    Console.WriteLine();
```

```
        Animation.PrintSetCursor("5) Управление статусами", 27);
        Console.WriteLine();
        Animation.PrintSetCursor("6) Добавить парковочную зону", 27);
        Console.WriteLine("\n\n");
        Animation.PrintSetCursor("Выбор: ", 27);

        int cursorPosition = Console.GetCursorPosition().Left;
        int choice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if(exit)
                return;
            if (!int.TryParse(temp, out choice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }

            if (choice >= 1 && choice <= 6)
                break;

            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
        }

        switch ((CaseFleetManagement)choice)
        {
            case CaseFleetManagement.AddVehicle:
                ManagementAddVehicle();
                break;
            case CaseFleetManagement.RemoveVehicle:
                ManagementRemoveVehicle();
                break;
            case CaseFleetManagement.ChangeStatus:
                ManagementChangeVehicleStatus();
                break;
            case CaseFleetManagement.ViewAll:
                ManagementViewAllVehicles();
                break;
            case CaseFleetManagement.ManageStatuses:
                ManagementStatusMenu(connection);
                break;
            case CaseFleetManagement.AddParkingZone:
                ManagementAddParkingZone();
                break;
            default:
                throw CheckoutException.Canceled;
        }
```

```
    }

    public bool ManagementAddVehicle()
    {
        exit = false;
        AddVehicle vehicle = new AddVehicle();

        Console.Clear();
        Animation.PrintRedText("Добавление транспорта", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Введите QR-код транспорта (19 символов): ", 27,
true, 1);

        int cursorPosition = Console.GetCursorPosition().Left;
        string qrCode;
        while (true)
        {
            qrCode = ReadLineWithCancel();
            if (exit) return false;

            if (qrCode.Length == 19)
                break;

            Animation.AnimationText("QR-код должен быть 19 символов!", true,
cursorPosition, qrCode.Length);
        }
        vehicle.QrCode = qrCode;

        Animation.PrintSetCursor("Введите тип транспорта: ", 27);
        vehicle.TypeTransport = ReadLineWithCancel();
        if (exit) return false;

        Console.Clear();
        Animation.AnimationText("Загрузка списка статусов...", false, 0, 0, true);

        var statuses = conn.Query("SELECT * FROM Status").ToList();

        Console.Clear();
        Animation.PrintRedText("Выберите статус транспорта", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);
        Console.WriteLine();
        Console.WriteLine();

        for (int i = 0; i < statuses.Count; i++)
        {
            Animation.PrintSetCursor($"{i + 1}) {statuses[i].Type_status} -
{statuses[i].Description}", 27);
```

```
            Console.WriteLine();
        }

        Animation.PrintSetCursor("Выбор: ", 5, true, 1);

        cursorPosition = Console.GetCursorPosition().Left;
        int choice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!int.TryParse(temp, out choice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }
            if (choice >= 1 && choice <= statuses.Count)
                break;
            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
        }

        vehicle.StatusId = statuses[choice - 1].id;

        Console.Clear();
        Animation.AnimationText("Добавление транспорта...", false, 0, 0, true);

        if (!vehicle.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка добавления транспорта!", true, true, 50);
            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Транспорт успешно добавлен!", true, true, 50);
            return true;
        }
    }

    public bool ManagementRemoveVehicle()
    {
        exit = false;
        RemoveVehicle remover = new RemoveVehicle();

        Console.Clear();
        Animation.PrintRedText("Удаление транспорта", true, false, 50);
        Console.WriteLine();
```

```
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Введите QR-код транспорта для удаления (19 символов): ", 27, true, 6);

        int cursorPosition = Console.GetCursorPosition().Left;
        string qrCode;
        while (true)
        {
            qrCode = ReadLineWithCancel();
            if (exit) return false;

            if (qrCode.Length == 19)
                break;

            Animation.AnimationText("QR-код должен быть 19 символов!", true, cursorPosition, qrCode.Length);
        }
        remover.QrCode = qrCode;

        Console.Clear();
        Animation.AnimationText("Поиск транспорта...", false, 0, 0, true);

        if (!remover.SearchData(conn))
        {
            Animation.PrintRedText("Транспорт не найден!", true, true, 50);
            return false;
        }

        if (remover.CheckRentals(conn))
        {
            Animation.PrintRedText("У транспорта есть активные аренды! Удаление невозможно.", true, true, 50);
            return false;
        }

        Console.Clear();
        Animation.PrintRedText("Вы уверены, что хотите удалить транспорт?", true, false, 50);
        Console.WriteLine();
        Animation.PrintSetCursor("1) Да", 18, true, 3);
        Console.WriteLine();
        Animation.PrintSetCursor("2) Нет", 18);
        Console.WriteLine();
        Animation.PrintSetCursor("Выбор: ", 18);

        cursorPosition = Console.GetCursorPosition().Left;
        int choice;
        while (true)
        {
```

```
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!int.TryParse(temp, out choice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }
            if (choice == 1 || choice == 2)
                break;
            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
        }

        if (choice == 2)
        {
            Console.Clear();
            Animation.PrintRedText("Удаление отменено!", true, true, 50);
            return false;
        }

        Console.Clear();
        Animation.AnimationText("Удаление транспорта...", false, 0, 0, true);

        if (!remover.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка удаления транспорта!", true, true, 50);
            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Транспорт успешно удален!", true, true, 50);
            return true;
        }
    }

    public bool ManagementChangeVehicleStatus()
    {
        exit = false;
        ChangeVehicleStatus changer = new ChangeVehicleStatus();

        Console.Clear();
        Animation.PrintRedText("Изменение статуса транспорта", true, false, 50);
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);

        Animation.PrintSetCursor("Введите QR-код транспорта (19 символов): ", 27,
true, 1);
```

```
        int cursorPosition = Console.GetCursorPosition().Left;
        string qrCode;
        while (true)
        {
            qrCode = ReadLineWithCancel();
            if (exit) return false;

            if (qrCode.Length == 19)
                break;

            Animation.AnimationText("QR-код должен быть 19 символов!", true, cursorPosition, qrCode.Length);
        }
        changer.QrCode = qrCode;

        Console.Clear();
        Animation.AnimationText("Поиск транспорта...", false, 0, 0, true);

        if (!changer.SearchData(conn))
        {
            Animation.PrintRedText("Транспорт не найден!", true, true, 50);
            return false;
        }

        var statuses = conn.Query("SELECT * FROM Status").ToList();

        Console.Clear();
        Animation.PrintRedText("Выберите новый статус транспорта", true, false, 50);
        Console.WriteLine();

        for (int i = 0; i < statuses.Count; i++)
        {
            Animation.PrintSetCursor($"{i + 1}) {statuses[i].Type_status} - {statuses[i].Description}", 27);
            Console.WriteLine();
        }

        Animation.PrintSetCursor("Выбор: ", 27);

        cursorPosition = Console.GetCursorPosition().Left;
        int choice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!int.TryParse(temp, out choice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition, temp.Length);
```

```
            continue;
        }
        if (choice >= 1 && choice <= statuses.Count)
            break;
        Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
    }

    changer.NewStatusId = statuses[choice - 1].id;

    Console.Clear();
    Animation.AnimationText("Изменение статуса...", false, 0, 0, true);

    if (!changer.Upload(conn))
    {
        Console.Clear();
        Animation.PrintRedText("Ошибка изменения статуса!", true, true, 50);
        return false;
    }
    else
    {
        Console.Clear();
        Animation.PrintRedText("Статус успешно изменен!", true, true, 50);
        return true;
    }
}

public void ManagementViewAllVehicles()
{
    Console.Clear();
    Animation.AnimationText("Загрузка данных...", false, 0, 0, true);

    try
    {
        var vehicles = conn.Query(@"
            SELECT V.id, V.QrCode, V.Type_Transport, S.Type_status, S.Description
            FROM Vehicles V
            INNER JOIN Status S ON V.Status = S.id").ToList();

        Console.Clear();
        Animation.PrintRedText("Список всего транспорта", true, false, 50);
        Console.WriteLine();

        if (vehicles.Count == 0)
        {
            Animation.PrintSetCursor("Транспорт не найден", 20, true, 2);
        }
        else
        {
            Animation.PrintSetCursor("ID".PadRight(5) + "QR-код".PadRight(21) +
```

```
"Тип".PadRight(20) + "Статус".PadRight(15) + "Описание", 80, true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor(new string('-', 80), 80);
            Console.WriteLine();

            foreach (var vehicle in vehicles)
            {
               Animation.PrintSetCursor(
                  vehicle.id.ToString().PadRight(5) +
                  vehicle.QrCode.PadRight(21) +
                  vehicle.Type_Transport.PadRight(20) +
                  vehicle.Type_status.PadRight(15) +
                  vehicle.Description, 80);
               Console.WriteLine();
            }
         }

         Console.WriteLine("\n\n");
         Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
80);
         Console.ReadKey();
      }
      catch (Exception ex)
      {
         Console.Clear();
         Animation.PrintRedText($"Ошибка загрузки данных: {ex.Message}", true,
true, 50);
      }
   }

   public void ManagementStatusMenu(SqlConnection connection)
   {
      exit = false;

      while (!exit)
      {
         Console.Clear();
         Animation.PrintRedText("Управление статусами", true, false, 50);
         Console.WriteLine();
         Animation.PrintRedText("Выход на Esc", true, false, 50);

         Animation.PrintSetCursor("Что нужно сделать?", 27, true, 6);
         Console.WriteLine();
         Animation.PrintSetCursor("1) Добавить новый статус", 27);
         Console.WriteLine();
         Animation.PrintSetCursor("2) Просмотреть все статусы", 27);
         Console.WriteLine();
         Animation.PrintSetCursor("3) Просмотреть парковочные зоны", 27);
         Console.WriteLine();
         Animation.PrintSetCursor("4) Назад в меню автопарка", 27);
```

```
            Console.WriteLine("\n\n");
            Animation.PrintSetCursor("Выбор: ", 27);

            int cursorPosition = Console.GetCursorPosition().Left;
            int choice;
            while (true)
            {
                string temp = ReadLineWithCancel();
                if(exit)
                    return;
                if (!int.TryParse(temp, out choice))
                {
                    Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                    continue;
                }

                if (choice >= 1 && choice <= 4)
                    break;

                Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
            }

            switch ((CaseStatusManagement)choice)
            {
                case CaseStatusManagement.AddStatus:
                    ManagementAddStatus();
                    break;
                case CaseStatusManagement.ViewStatuses:
                    ManagementViewAllStatuses();
                    break;
                case CaseStatusManagement.ViewParkingZones:
                    ManagementViewAllParkingZones();
                    break;
                case CaseStatusManagement.Back:
                    return;
                default:
                    throw new Exception("Неизвестный выбор");
            }
        }
    }

    public bool ManagementAddStatus()
    {
        exit = false;
        AddStatus status = new AddStatus();

        Console.Clear();
        Animation.PrintRedText("Добавление нового статуса", true, false, 50);
```

```
    Console.WriteLine();
    Animation.PrintRedText("Выход на Esc", true, false, 50);

    Animation.PrintSetCursor("Введите название статуса: ", 27, true, 6);
    status.TypeStatus = ReadLineWithCancel();
    if (exit)
        return false;

    Animation.PrintSetCursor("Введите описание статуса: ", 27);
    status.Description = ReadLineWithCancel();
    if (exit)
        return false;

    Console.Clear();
    Animation.AnimationText("Добавление статуса...", false, 0, 0, true);

    if (!status.Upload(conn))
    {
        Console.Clear();
        Animation.PrintRedText("Ошибка добавления статуса!", true, true, 50);
        return false;
    }
    else
    {
        Console.Clear();
        Animation.PrintRedText("Статус успешно добавлен!", true, true, 50);
        return true;
    }
}

public void ManagementViewAllStatuses()
{
    Console.Clear();
    Animation.AnimationText("Загрузка данных...", false, 0, 0, true);

    try
    {
        var statuses = conn.Query("SELECT * FROM Status").ToList();

        Console.Clear();
        Animation.PrintRedText("Список всех статусов", true, false, 50);
        Console.WriteLine();

        if (statuses.Count == 0)
        {
            Animation.PrintSetCursor("Статусы не найдены", 20, true, 2);
        }
        else
        {
            Animation.PrintSetCursor("ID".PadRight(5) + "Название".PadRight(20) +
```

```
"Описание", 20, true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor(new string('-', 60), 20);
            Console.WriteLine();

            foreach (var status in statuses)
            {
               Animation.PrintSetCursor(
                  status.id.ToString().PadRight(5) +
                  status.Type_status.PadRight(20) +
                  status.Description, 20);
               Console.WriteLine();
            }
         }

         Console.WriteLine("\n\n");
         Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
20);
         Console.ReadKey();
      }
      catch (Exception ex)
      {
         Console.Clear();
         Animation.PrintRedText($"Ошибка загрузки данных: {ex.Message}", true,
true, 50);
      }
   }

   public class StartRental : IUploadInServer
   {
      public int UserId { private get; set; }
      public int VehicleId { private get; set; }
      public int ParkingZoneId { private get; set; }
      public DateTime StartTrip { private get; set; }

      public StartRental() { }

      public bool CheckUserExists(SqlConnection connection)
      {
         try
         {
            var result = connection.QueryFirstOrDefault<int?>(
               "SELECT id FROM Users WHERE id = @UserId",
               new { UserId });

            if (!result.HasValue)
            {
               Console.Clear();
               Animation.PrintRedText($"Ошибка: пользователь с ID {UserId} не
найден!", true, true, 50);
```

```
            return false;
        }
        return true;
    }
    catch (Exception ex)
    {
        Console.Clear();
        Animation.PrintRedText($"Ошибка проверки пользователя: {ex.Message}", true, true, 50);
        return false;
    }
}

public bool CheckVehicleAvailable(SqlConnection connection)
{
    try
    {
        var result = connection.QueryFirstOrDefault<int?>(
            @"SELECT V.id FROM Vehicles V
              WHERE V.id = @VehicleId AND V.Status = 1",
            new { VehicleId });

        if (!result.HasValue)
        {
            Console.Clear();
            Animation.PrintRedText($"Ошибка: транспорт с ID {VehicleId} не найден или недоступен!", true, true, 50);
            return false;
        }
        return true;
    }
    catch (Exception ex)
    {
        Console.Clear();
        Animation.PrintRedText($"Ошибка проверки транспорта: {ex.Message}", true, true, 50);
        return false;
    }
}

public bool CheckParkingZoneExists(SqlConnection connection)
{
    try
    {
        var result = connection.QueryFirstOrDefault<int?>(
            "SELECT id FROM Parking_Zones WHERE id = @ParkingZoneId",
            new { ParkingZoneId });

        if (!result.HasValue)
        {
```

```
                Console.Clear();
                Animation.PrintRedText($"Ошибка: парковочная зона с ID {ParkingZoneId} не найдена!", true, true, 50);
                return false;
            }
            return true;
        }
        catch (Exception ex)
        {
            Console.Clear();
            Animation.PrintRedText($"Ошибка проверки парковочной зоны: {ex.Message}", true, true, 50);
            return false;
        }
    }

    public bool Upload(SqlConnection connection)
    {
        if (!CheckUserExists(connection))
            return false;

        if (!CheckVehicleAvailable(connection))
            return false;

        if (!CheckParkingZoneExists(connection))
            return false;

        using var transaction = connection.BeginTransaction();
        try
        {
            string rentalSql = @"
                INSERT INTO Rentals (UserId, Start_trip, Parking_zone, Vehicle)
                VALUES (@UserId, @StartTrip, @ParkingZoneId, @VehicleId)";

            int affectedRows = connection.Execute(rentalSql, new
            {
                UserId,
                StartTrip,
                ParkingZoneId,
                VehicleId
            }, transaction);

            string vehicleSql = "UPDATE Vehicles SET Status = 2 WHERE id = @VehicleId";
            connection.Execute(vehicleSql, new { VehicleId }, transaction);

            transaction.Commit();
            return affectedRows > 0;
        }
        catch (Exception ex)
```

```
        {
            transaction.Rollback();
            Console.Clear();
            Animation.PrintRedText($"Ошибка оформления аренды: {ex.Message}",
true, true, 50);

            if (ex.Message.Contains("FOREIGN KEY"))
            {
                Animation.PrintRedText("Проблема с ссылочной целостностью
данных.", true, true, 50);
                Animation.PrintRedText("Проверьте существование UserId, VehicleId
и ParkingZoneId.", true, true, 50);
            }

            return false;
        }
    }
}

public bool StartRent()
{
    exit = false;
    StartRental rental = new StartRental();

    Console.Clear();
    Animation.PrintRedText("Оформление аренды", true, false, 50);
    Console.WriteLine();
    Animation.PrintRedText("Выход на Esc", true, false, 50);

    Console.Clear();
    Animation.AnimationText("Проверка данных...", false, 0, 0, true);

    try
    {
        var userCount = conn.QueryFirstOrDefault<int>("SELECT COUNT(*)
FROM Users");
        var vehicleCount = conn.QueryFirstOrDefault<int>("SELECT COUNT(*)
FROM Vehicles WHERE Status = 1");
        var zoneCount = conn.QueryFirstOrDefault<int>("SELECT COUNT(*)
FROM Parking_Zones");

        Console.Clear();
        Animation.PrintRedText("Статус системы:", true, false, 50);
        Console.WriteLine();
        Console.WriteLine();
        Console.WriteLine();
        Animation.PrintSetCursor($"Пользователей: {userCount}", 20);
        Console.WriteLine();
        Animation.PrintSetCursor($"Доступного транспорта: {vehicleCount}", 20);
        Console.WriteLine();
```

```
            Animation.PrintSetCursor($"Парковочных зон: {zoneCount}", 20);
            Console.WriteLine("\n\n");
            Console.WriteLine("\n\n");
            Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
20);
            Console.ReadKey();

            if (userCount == 0)
            {
                Animation.PrintRedText("В системе нет зарегистрированных
пользователей!", true, true, 50);
                Animation.PrintRedText("Сначала зарегистрируйте пользователя.", true,
true, 50);
                return false;
            }

            if (vehicleCount == 0)
            {
                Animation.PrintRedText("Нет доступного транспорта для аренды!", true,
true, 50);
                return false;
            }

            if (zoneCount == 0)
            {
                Animation.PrintRedText("В системе нет парковочных зон!", true, true,
50);
                Animation.PrintRedText("Сначала добавьте парковочные зоны.", true,
true, 50);
                return false;
            }
        }
        catch (Exception ex)
        {
            Console.Clear();
            Animation.PrintRedText($"Ошибка проверки данных: {ex.Message}", true,
true, 50);
            return false;
        }

        Console.Clear();
        Animation.AnimationText("Загрузка списка пользователей...", false, 0, 0, true);

        var users = conn.Query("SELECT id, Name, Family_Name, Email FROM
Users").ToList();

        Console.Clear();
        Animation.PrintRedText("Выберите пользователя", true, false, 50);
        Console.WriteLine();
```

```
        for (int i = 0; i < users.Count; i++)
        {
            string displayText = $"{i + 1}) ID:{users[i].id} {users[i].Name} {users[i].Family_Name} ({users[i].Email})";
            if (displayText.Length > Console.WindowWidth - 10)
            {
                Console.WriteLine(displayText);
            }
            else
            {
                Animation.PrintSetCursor(displayText, Math.Min(displayText.Length + 10, Console.WindowWidth - 5));
                Console.WriteLine();
            }
        }

        Animation.PrintSetCursor("Выбор: ", 50);

        int cursorPosition = Console.GetCursorPosition().Left;
        int userChoice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!int.TryParse(temp, out userChoice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition, temp.Length);
                continue;
            }
            if (userChoice >= 1 && userChoice <= users.Count)
                break;
            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition, temp.Length);
        }

        rental.UserId = users[userChoice - 1].id;

        Console.Clear();
        Animation.AnimationText("Загрузка свободного транспорта...", false, 0, 0, true);

        var vehicles = conn.Query(@"SELECT V.id, V.QrCode, V.Type_Transport
                                    FROM Vehicles V
                                    WHERE V.Status = 1").ToList();

        if (vehicles.Count == 0)
        {
            Animation.PrintRedText("Нет свободного транспорта!", true, true, 50);
            return false;
```

```
        }

        Console.Clear();
        Animation.PrintRedText("Выберите транспорт", true, false, 50);
        Console.WriteLine();

        for (int i = 0; i < vehicles.Count; i++)
        {
            Animation.PrintSetCursor($"{i + 1}) {vehicles[i].Type_Transport} (QR: {vehicles[i].QrCode})", 27);
            Console.WriteLine();
        }

        Animation.PrintSetCursor("Выбор: ", 50);

        cursorPosition = Console.GetCursorPosition().Left;
        int vehicleChoice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!int.TryParse(temp, out vehicleChoice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition, temp.Length);
                continue;
            }
            if (vehicleChoice >= 1 && vehicleChoice <= vehicles.Count)
                break;
            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition, temp.Length);
        }

        rental.VehicleId = vehicles[vehicleChoice - 1].id;

        Console.Clear();
        Animation.AnimationText("Загрузка парковочных зон...", false, 0, 0, true);

        var parkingZones = conn.Query("SELECT id, Approximate_address FROM Parking_Zones").ToList();

        Console.Clear();
        Animation.PrintRedText("Выберите парковочную зону", true, false, 50);
        Console.WriteLine();

        for (int i = 0; i < parkingZones.Count; i++)
        {
            string displayText = $"{i + 1}) {parkingZones[i].Approximate_address}";
            if (displayText.Length > Console.WindowWidth - 10)
            {
```

```
            Console.WriteLine(displayText);
        }
        else
        {
            Animation.PrintSetCursor(displayText, Math.Min(displayText.Length + 10,
Console.WindowWidth - 5));
            Console.WriteLine();
        }
    }

    Console.WriteLine();
    Animation.PrintSetCursor("Выбор: ", 20);

    cursorPosition = Console.GetCursorPosition().Left;
    int zoneChoice;
    while (true)
    {
        string temp = ReadLineWithCancel();
        if (exit) return false;
        if (!int.TryParse(temp, out zoneChoice))
        {
            Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
            continue;
        }
        if (zoneChoice >= 1 && zoneChoice <= parkingZones.Count)
            break;
        Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
    }

    rental.ParkingZoneId = parkingZones[zoneChoice - 1].id;

    rental.StartTrip = DateTime.Now;

    Console.Clear();
    Animation.AnimationText("Оформление аренды...", false, 0, 0, true);

    if (!rental.Upload(conn))
    {
        Console.Clear();
        Animation.PrintRedText("Ошибка оформления аренды!", true, true, 50);
        return false;
    }
    else
    {
        Console.Clear();
        Animation.PrintRedText("Аренда успешно оформлена!", true, true, 50);
        return true;
    }
```

```
    }

    public class EndRental : IUploadInServer
    {
    public int RentalId { private get; set; }
    public DateTime ActualEndTime { private get; set; }

    public EndRental() { }

    public bool CheckRentalExists(SqlConnection connection)
    {
        try
        {
            var result = connection.QueryFirstOrDefault<int?>(
                "SELECT id FROM Rentals WHERE id = @RentalId AND End_trip > GETDATE()",
                new { RentalId });
            return result.HasValue;
        }
        catch (Exception)
        {
            return false;
        }
    }

    public int GetVehicleId(SqlConnection connection)
    {
        try
        {
            var result = connection.QueryFirstOrDefault<int?>(
                "SELECT Vehicle FROM Rentals WHERE id = @RentalId",
                new { RentalId });
            return result ?? 0;
        }
        catch (Exception)
        {
            return 0;
        }
    }

    public bool Upload(SqlConnection connection)
    {
        using var transaction = connection.BeginTransaction();
        try
        {
            string rentalSql = "UPDATE Rentals SET End_trip = @ActualEndTime WHERE id = @RentalId";
            int affectedRows = connection.Execute(rentalSql, new
            {
                ActualEndTime,
```

```
            RentalId
        }, transaction);

        int vehicleId = GetVehicleId(connection);

        string vehicleSql = "UPDATE Vehicles SET Status = 1 WHERE id = @VehicleId";
        connection.Execute(vehicleSql, new { VehicleId = vehicleId }, transaction);

        transaction.Commit();
        return affectedRows > 0;
      }
      catch (Exception ex)
      {
        transaction.Rollback();
        Console.Clear();
        Animation.PrintRedText($"Ошибка завершения аренды!", true, true, 50);
        return false;
      }
    }
}

    public bool EndRent()
    {
      exit = false;
      EndRental endRental = new EndRental();

      Console.Clear();
      Animation.PrintRedText("Завершение аренды", true, false, 50);
      Console.WriteLine();
      Animation.PrintRedText("Выход на Esc", true, false, 50);

      Console.Clear();
      Animation.AnimationText("Загрузка активных аренд...", false, 0, 0, true);

      var activeRentals = conn.Query(@"
        SELECT R.id, U.Name, U.Family_Name, V.Type_Transport, R.Start_trip
        FROM Rentals R
        INNER JOIN Users U ON R.UserId = U.id
        INNER JOIN Vehicles V ON R.Vehicle = V.id
        WHERE R.End_trip IS NULL").ToList();

      if (activeRentals.Count == 0)
      {
        Animation.PrintRedText("Нет активных аренд!", true, true, 50);
        return false;
      }

      Console.Clear();
      Animation.PrintRedText("Выберите аренду для завершения", true, false, 50);
```

```
        Console.WriteLine();
        Animation.PrintRedText("Выход на Esc", true, false, 50);
        Console.WriteLine("\n\n\n");

        for (int i = 0; i < activeRentals.Count; i++)
        {
            Animation.PrintSetCursor($"{i + 1}) {activeRentals[i].Name}
{activeRentals[i].Family_Name} - " +
                $"{activeRentals[i].Type_Transport} (начало:
{activeRentals[i].Start_trip:g})", 27);
            Console.WriteLine();
        }

        Animation.PrintSetCursor("Выбор: ", 50);

        int cursorPosition = Console.GetCursorPosition().Left;
        int rentalChoice;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!int.TryParse(temp, out rentalChoice))
            {
                Animation.AnimationText("Ошибка ввода!", true, cursorPosition,
temp.Length);
                continue;
            }
            if (rentalChoice >= 1 && rentalChoice <= activeRentals.Count)
                break;
            Animation.AnimationText("Ошибка в выборе!", true, cursorPosition,
temp.Length);
        }

        endRental.RentalId = activeRentals[rentalChoice - 1].id;
        endRental.ActualEndTime = DateTime.Now;

        Console.Clear();
        Animation.AnimationText("Завершение аренды...", false, 0, 0, true);

        if (!endRental.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка завершения аренды!", true, true, 50);
            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Аренда успешно завершена!", true, true, 50);
            return true;
```

```
        }
    }

    public void FindGeo()
    {
        Console.Clear();
        Animation.AnimationText("Загрузка данных о местоположении...", false, 0, 0,
true);

        try
        {
            var vehicleLocations = conn.Query(@"
                SELECT
                    V.id,
                    V.QrCode,
                    V.Type_Transport,
                    S.Type_status as Status,
                    PZ.Approximate_address as Location,
                    CASE
                        WHEN R.id IS NOT NULL AND R.End_trip IS NULL THEN 'В
аренде'
                        ELSE 'Свободен'
                    END as RentalStatus,
                    U.Name + ' ' + U.Family_Name as RentedBy
                FROM Vehicles V
                INNER JOIN Status S ON V.Status = S.id
                LEFT JOIN Rentals R ON R.Vehicle = V.id AND R.End_trip IS NULL
                LEFT JOIN Parking_Zones PZ ON R.Parking_zone = PZ.id
                LEFT JOIN Users U ON R.UserId = U.id
                ORDER BY V.id").ToList();

            Console.Clear();
            Animation.PrintRedText("Геолокация транспортов", true, false, 50);
            Console.WriteLine("\n\n\n");

            if (vehicleLocations.Count == 0)
            {
                Animation.PrintSetCursor("Транспорт не найден", 20, true, 2);
            }
            else
            {
                Animation.PrintSetCursor("ID".PadRight(5) + "QR-код".PadRight(21) +
"Тип".PadRight(15) +
                                         "Статус".PadRight(12) + "Аренда".PadRight(12) +
"Местоположение", 50, true, 2);
                Console.WriteLine();
                Animation.PrintSetCursor(new string('-', 90), 50);
                Console.WriteLine();

                foreach (var vehicle in vehicleLocations)
```

```
            {
                string location = vehicle.Location ?? "Не указано";
                string rentedBy = vehicle.RentedBy ?? "-";

                Animation.PrintSetCursor(
                    vehicle.id.ToString().PadRight(5) +
                    vehicle.QrCode.PadRight(21) +
                    vehicle.Type_Transport.PadRight(15) +
                    vehicle.Status.PadRight(12) +
                    vehicle.RentalStatus.PadRight(12) +
                    location, 50);
                Console.WriteLine();

                if (vehicle.RentalStatus == "В аренде")
                {
                    Animation.PrintSetCursor("     Арендован: " + rentedBy, 50);
                    Console.WriteLine();
                }
            }
        }

        Console.WriteLine("\n\n");
        Console.WriteLine("\n\n");
        Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
50);
        Console.ReadKey();
    }
    catch (Exception ex)
    {
        Console.Clear();
        Animation.PrintRedText($"Ошибка загрузки данных: {ex.Message}", true,
true, 50);
    }
}

public bool ManagementAddParkingZone()
{
    exit = false;
    AddParkingZone parkingZone = new AddParkingZone();

    Console.Clear();
    Animation.PrintRedText("Добавление парковочной зоны", true, false, 50);
    Console.WriteLine();
    Animation.PrintRedText("Выход на Esc", true, false, 50);

    Animation.PrintSetCursor("Введите ширину зоны (например: 12.345678): ",
27, true, 6);

    int cursorPosition = Console.GetCursorPosition().Left;
    decimal width;
```

```
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!decimal.TryParse(temp, out width))
            {
                Animation.AnimationText("Ошибка ввода! Введите число.", true,
cursorPosition, temp.Length);
                continue;
            }
            break;
        }
        parkingZone.Width = width;

        Animation.PrintSetCursor("Введите длину зоны (например: 12.345678): ", 27);

        cursorPosition = Console.GetCursorPosition().Left;
        decimal length;
        while (true)
        {
            string temp = ReadLineWithCancel();
            if (exit) return false;
            if (!decimal.TryParse(temp, out length))
            {
                Animation.AnimationText("Ошибка ввода! Введите число.", true,
cursorPosition, temp.Length);
                continue;
            }
            break;
        }
        parkingZone.Length = length;

        Animation.PrintSetCursor("Введите приблизительный адрес: ", 27);
        parkingZone.ApproximateAddress = ReadLineWithCancel();
        if (exit) return false;

        Console.Clear();
        Animation.AnimationText("Добавление парковочной зоны...", false, 0, 0, true);

        if (!parkingZone.Upload(conn))
        {
            Console.Clear();
            Animation.PrintRedText("Ошибка добавления парковочной зоны!", true,
true, 50);
            return false;
        }
        else
        {
            Console.Clear();
            Animation.PrintRedText("Парковочная зона успешно добавлена!", true,
```

```
true, 50);
            return true;
        }
    }

    public void ManagementViewAllParkingZones()
    {
        Console.Clear();
        Animation.AnimationText("Загрузка данных...", false, 0, 0, true);

        try
        {
            var parkingZones = conn.Query("SELECT * FROM
Parking_Zones").ToList();

            Console.Clear();
            Animation.PrintRedText("Список всех парковочных зон", true, false, 50);
            Console.WriteLine();

            if (parkingZones.Count == 0)
            {
                Animation.PrintSetCursor("Парковочные зоны не найдены", 20, true, 2);
            }
            else
            {
                Animation.PrintSetCursor("ID".PadRight(5) + "Ширина".PadRight(12) +
"Длина".PadRight(12) + "Адрес", 20, true, 2);
                Console.WriteLine();
                Animation.PrintSetCursor(new string('-', 80), 20);
                Console.WriteLine();

                foreach (var zone in parkingZones)
                {
                    Animation.PrintSetCursor(
                        zone.id.ToString().PadRight(5) +
                        zone.Width.ToString().PadRight(12) +
                        zone.Length.ToString().PadRight(12) +
                        zone.Approximate_address, 20);
                    Console.WriteLine();
                }
            }

            Console.WriteLine("\n\n");
            Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
20);
            Console.ReadKey();
        }
        catch (Exception ex)
        {
            Console.Clear();
```

```
            Animation.PrintRedText($"Ошибка загрузки данных: {ex.Message}", true,
true, 50);
        }
    }

    public void GenerateReport()
    {
        Console.Clear();
        Animation.AnimationText("Формирование полного отчета...", false, 0, 0, true);

        try
        {
            var generalStats = conn.Query(@"
                SELECT
                    (SELECT COUNT(*) FROM Users) as TotalUsers,
                    (SELECT COUNT(*) FROM Vehicles) as TotalVehicles,
                    (SELECT COUNT(*) FROM Parking_Zones) as TotalParkingZones,
                    (SELECT COUNT(*) FROM Rentals) as TotalRentals,
                    (SELECT COUNT(*) FROM Rentals WHERE End_trip IS NULL) as
ActiveRentals,
                    (SELECT COUNT(*) FROM Status) as TotalStatuses,
                    (SELECT COUNT(*) FROM Wallets) as TotalWallets
            ").First();

            Console.Clear();
            Animation.PrintRedText("ПОЛНЫЙ ОТЧЕТ СИСТЕМЫ АРЕНДЫ", true,
false, 50);
            Console.WriteLine();

            Animation.PrintSetCursor("=== ОБЩАЯ СТАТИСТИКА ===", 20, true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor($"Пользователей: {generalStats.TotalUsers}", 40);
            Console.WriteLine();
            Animation.PrintSetCursor($"Транспортных                    средств:
{generalStats.TotalVehicles}", 40);
            Console.WriteLine();
            Animation.PrintSetCursor($"Парковочных                    зон:
{generalStats.TotalParkingZones}", 40);
            Console.WriteLine();
            Animation.PrintSetCursor($"Всего аренд: {generalStats.TotalRentals}", 40);
            Console.WriteLine();
            Animation.PrintSetCursor($"Активных аренд: {generalStats.ActiveRentals}",
40);
            Console.WriteLine();
            Animation.PrintSetCursor($"Статусов: {generalStats.TotalStatuses}", 40);
            Console.WriteLine();
            Animation.PrintSetCursor($"Кошельков: {generalStats.TotalWallets}", 40);
            Console.WriteLine("\n\n");

            Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
```

```
20);
        Console.ReadKey();

        var users = conn.Query(@"
            SELECT U.id, U.Name, U.Family_Name, U.Email, W.Number_card,
W.Validity,
                (SELECT COUNT(*) FROM Rentals R WHERE R.UserId = U.id) as
RentalCount
            FROM Users U
            INNER JOIN Wallets W ON U.Wallet = W.id
            ORDER BY U.id").ToList();

        Console.Clear();
        Animation.PrintRedText("ОТЧЕТ: ПОЛЬЗОВАТЕЛИ", true, false, 50);
        Console.WriteLine();

        if (users.Count == 0)
        {
            Animation.PrintSetCursor("Пользователи не найдены", 20, true, 2);
        }
        else
        {
            Animation.PrintSetCursor($"Всего пользователей: {users.Count}", 20,
true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor("ID".PadRight(5) + "Имя".PadRight(15) +
"Фамилия".PadRight(15) +
                                     "Email".PadRight(25) + "Аренд".PadRight(8) + "Номер
карты".PadRight(20) + "Срок действия", 80);
            Console.WriteLine();
            Animation.PrintSetCursor(new string('-', 110), 80);
            Console.WriteLine();

            foreach (var user in users)
            {
                Animation.PrintSetCursor(
                    user.id.ToString().PadRight(5) +
                    user.Name.PadRight(15) +
                    user.Family_Name.PadRight(15) +
                    user.Email.PadRight(25) +
                    user.RentalCount.ToString().PadRight(8) +
                    user.Number_card.ToString().PadRight(20) +
                    user.Validity.ToString("MM/yyyy"), 80);
                Console.WriteLine();
            }
        }

        Console.WriteLine("\n\n");
        Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
20);
```

```
        Console.ReadKey();

        var vehicles = conn.Query(@"
            SELECT V.id, V.QrCode, V.Type_Transport, S.Type_status as Status,
S.Description,
                (SELECT COUNT(*) FROM Rentals R WHERE R.Vehicle = V.id) as
RentalCount
            FROM Vehicles V
            INNER JOIN Status S ON V.Status = S.id
            ORDER BY V.id").ToList();

        Console.Clear();
        Animation.PrintRedText("ОТЧЕТ: ТРАНСПОРТ", true, false, 80);
        Console.WriteLine();

        if (vehicles.Count == 0)
        {
            Animation.PrintSetCursor("Транспорт не найден", 20, true, 2);
        }
        else
        {
            Animation.PrintSetCursor($"Всего транспортных средств:
{vehicles.Count}", 20, true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor("ID".PadRight(5) + "QR-код".PadRight(21) +
"Тип".PadRight(15) +
                                     "Статус".PadRight(12) + "Аренд".PadRight(8) +
"Описание", 80);
            Console.WriteLine();
            Animation.PrintSetCursor(new string('-', 90), 80);
            Console.WriteLine();

            foreach (var vehicle in vehicles)
            {
                Animation.PrintSetCursor(
                    vehicle.id.ToString().PadRight(5) +
                    vehicle.QrCode.PadRight(21) +
                    vehicle.Type_Transport.PadRight(15) +
                    vehicle.Status.PadRight(12) +
                    vehicle.RentalCount.ToString().PadRight(8) +
                    vehicle.Description, 80);
                Console.WriteLine();
            }
        }

        Console.WriteLine("\n\n");
        Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
20);
        Console.ReadKey();
```

```
        var parkingZones = conn.Query(@"
            SELECT PZ.id, PZ.Width, PZ.Length, PZ.Approximate_address,
                (SELECT COUNT(*) FROM Rentals R WHERE R.Parking_zone =
PZ.id) as RentalCount
            FROM Parking_Zones PZ
            ORDER BY PZ.id").ToList();

        Console.Clear();
        Animation.PrintRedText("ОТЧЕТ: ПАРКОВОЧНЫЕ ЗОНЫ", true, false,
50);
        Console.WriteLine();

        if (parkingZones.Count == 0)
        {
            Animation.PrintSetCursor("Парковочные зоны не найдены", 80, true, 2);
        }
        else
        {
            Animation.PrintSetCursor($"Всего парковочных зон:
{parkingZones.Count}", 80, true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor("ID".PadRight(5) + "Ширина".PadRight(12) +
"Длина".PadRight(12) +
                                     "Аренд".PadRight(8) + "Адрес", 80);
            Console.WriteLine();
            Animation.PrintSetCursor(new string('-', 80), 80);
            Console.WriteLine();

            foreach (var zone in parkingZones)
            {
                Animation.PrintSetCursor(
                    zone.id.ToString().PadRight(5) +
                    zone.Width.ToString().PadRight(12) +
                    zone.Length.ToString().PadRight(12) +
                    zone.RentalCount.ToString().PadRight(8) +
                    zone.Approximate_address, 80);
                Console.WriteLine();
            }
        }

        Console.WriteLine("\n\n");
        Animation.PrintSetCursor("Нажмите любую клавишу для продолжения...",
20);
        Console.ReadKey();

        var rentals = conn.Query(@"
            SELECT
                R.id,
                U.Name + ' ' + U.Family_Name as UserName,
                V.Type_Transport as VehicleType,
```

```
                V.QrCode,
                PZ.Approximate_address as ParkingLocation,
                R.Start_trip,
                R.End_trip,
                CASE
                    WHEN R.End_trip IS NULL THEN 'Активна'
                    ELSE 'Завершена'
                END as Status,
                DATEDIFF(MINUTE, R.Start_trip, ISNULL(R.End_trip, GETDATE()))
as DurationMinutes
            FROM Rentals R
            INNER JOIN Users U ON R.UserId = U.id
            INNER JOIN Vehicles V ON R.Vehicle = V.id
            INNER JOIN Parking_Zones PZ ON R.Parking_zone = PZ.id
            ORDER BY R.Start_trip DESC").ToList();

        Console.Clear();
        Animation.PrintRedText("ОТЧЕТ: ИСТОРИЯ АРЕНД", true, false, 50);
        Console.WriteLine();

        if (rentals.Count == 0)
        {
            Animation.PrintSetCursor("Аренды не найдены", 80, true, 2);
        }
        else
        {
            Animation.PrintSetCursor($"Всего аренд в истории: {rentals.Count}", 20,
true, 2);
            Console.WriteLine();
            Animation.PrintSetCursor("ID".PadRight(5)                                    +
"Пользователь".PadRight(25) + "Транспорт".PadRight(15) +
                                "Начало".PadRight(20)    +    "Окончание".PadRight(20)    +
"Статус".PadRight(10) + "Длительность", 80);
            Console.WriteLine();
            Animation.PrintSetCursor(new string('-', 120), 80);
            Console.WriteLine();

            foreach (var rental in rentals)
            {
                string endTime = rental.End_trip?.ToString("dd.MM.yyyy HH:mm") ??
"---";
                string duration = rental.DurationMinutes >= 60
                    ? $"{rental.DurationMinutes / 60}ч {rental.DurationMinutes % 60}м"
                    : $"{rental.DurationMinutes}м";

                Animation.PrintSetCursor(
                    rental.id.ToString().PadRight(5) +
                    rental.UserName.PadRight(25) +
                    rental.VehicleType.PadRight(15) +
                    rental.Start_trip.ToString("dd.MM.yyyy HH:mm").PadRight(20) +
```

```
                endTime.PadRight(20) +
                rental.Status.PadRight(10) +
                duration, 80);
            Console.WriteLine();
        }
    }

    var statuses = conn.Query("SELECT * FROM Status ORDER BY id").ToList();

    Console.WriteLine("\n\n");
    Animation.PrintSetCursor("=== СТАТУСЫ ТРАНСПОРТА ===", 20);
    Console.WriteLine();

    if (statuses.Count == 0)
    {
        Animation.PrintSetCursor("Статусы не найдены", 20);
    }
    else
    {
        foreach (var status in statuses)
        {
            Animation.PrintSetCursor($"ID: {status.id} - {status.Type_status}: {status.Description}", 80);
            Console.WriteLine();
        }
    }

    Console.WriteLine("\n\n");
    Animation.PrintSetCursor("=== ОТЧЕТ СФОРМИРОВАН ===", 20);
    Console.WriteLine();
    Animation.PrintSetCursor("Нажмите любую клавишу для возврата в меню...", 40);
    Console.ReadKey();
  }
  catch (Exception ex)
  {
    Console.Clear();
    Animation.PrintRedText($"Ошибка формирования отчета: {ex.Message}", true, true, 50);
    Animation.PrintRedText("Нажмите любую клавишу для продолжения...", true, true, 50);
    Console.ReadKey();
  }
 }

}
```