**Министерство образования Российской Федерации**

**Пензенский государственный университет**

Кафедра «Математическое обеспечение и применение ЭВМ»

Пояснительная записка к курсовому проекту   
по дисциплине

«Объектно-ориентированное программирование».

Разработка программы с использованием объектно-ориентированного подхода.   
ИС «Автобусный парк»

Автор работы:

студенты группы 23ВП2

Петрунин А. Д.

Принял:

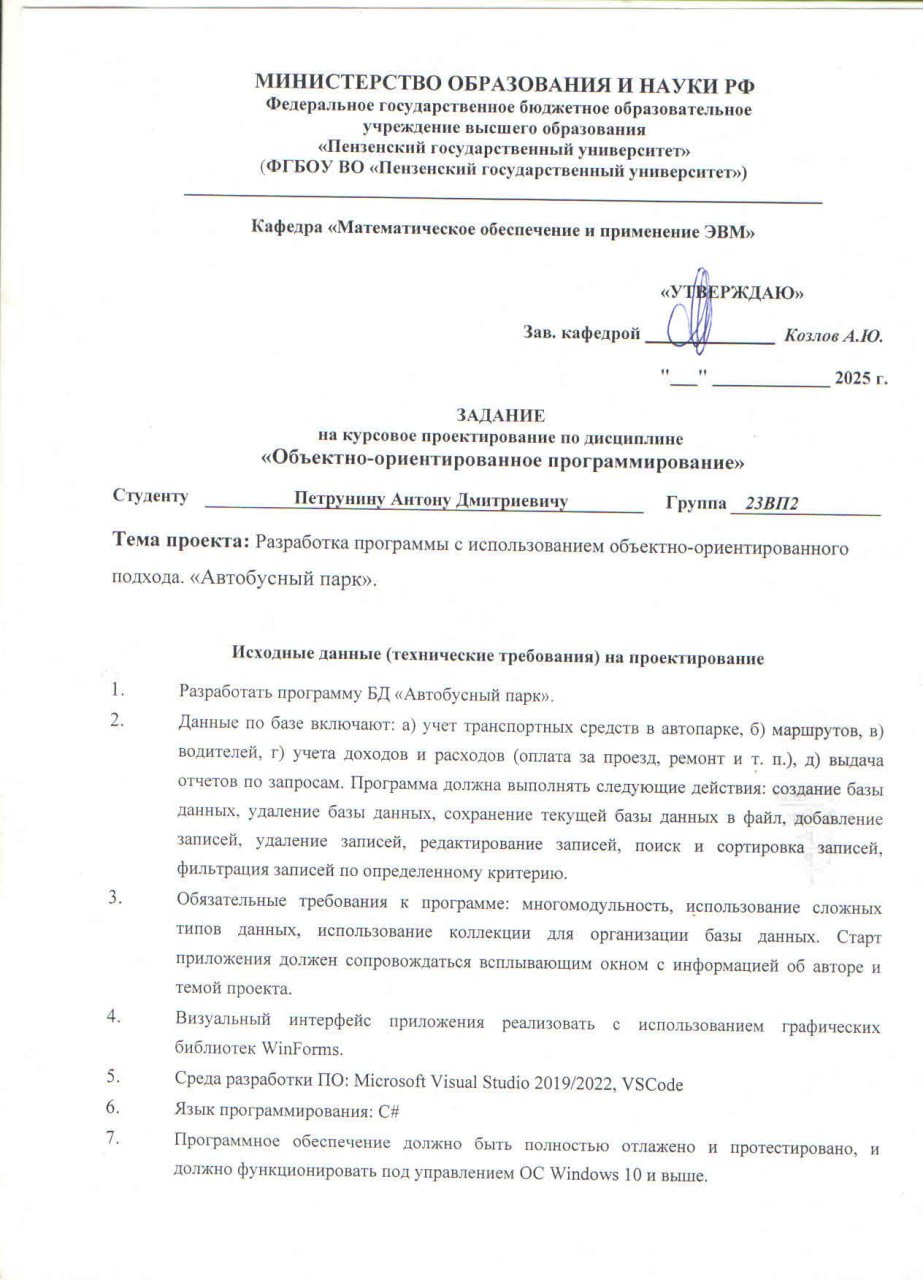
к.т.н. Афонин А.Ю.

Работа сдана \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Пенза 2025

Листы с заданием на курсовое проектирование



**Содержание**

[Введение 5](#_Toc199838341)

[1 Постановка задачи 6](#_Toc199838342)

[2 Выбор решения 8](#_Toc199838343)

[2.1 Определение необходимых модулей программы 8](#_Toc199838344)

[2.2 Определение структуры файла базы данных 10](#_Toc199838345)

[3 Описание разработки программы 11](#_Toc199838346)

[3.1 Кодирование 11](#_Toc199838347)

[3.2 Диаграмма компонентов 12](#_Toc199838348)

[4 Отладка и тестирование 13](#_Toc199838349)

[4.1. Функциональное тестирование 13](#_Toc199838350)

[4.2. Тестирование приложения 13](#_Toc199838351)

[5 Описание программы 19](#_Toc199838352)

[5.1 Разработка приложения Petrunin\_23VP2\_BusPark.exe 19](#_Toc199838353)

[Выводы 21](#_Toc199838354)

[Список использованных источников 22](#_Toc199838355)

[Приложение A – Код программы 23](#_Toc199838356)

[Приложение B – Руководство пользователя 40](#_Toc199838357)

# Введение

База данных — это упорядоченная совокупность данных, хранящихся в цифровом виде на компьютере. Она обеспечивает эффективное хранение, обработку и поиск информации, что делает её незаменимым инструментом во многих областях. Благодаря базам данных можно систематизировать значительные объёмы информации и упростить её анализ [1].

Современные базы данных чаще всего организованы в виде таблиц, состоящих из строк и столбцов. Такой формат упрощает управление данными, их обновление, модификацию и контроль доступа.

Разработка базы данных включает проектирование её структуры, определение связей между данными и настройку способов доступа к ним. Главная задача — обеспечить удобное, безопасное и быстрое хранение информации в соответствии с потребностями организации.

При проектировании важно учитывать оптимизацию запросов, масштабируемость и производительность системы. Также необходимо анализировать требования программного обеспечения, чтобы база данных поддерживала стабильную работу всей инфраструктуры [2].

В частности, разработка базы данных для автобусного парка играет ключевую роль в управлении транспортными операциями, обеспечивая быстрый доступ к информации и способствуя повышению качества обслуживания пассажиров, а также оптимизации логистики и технического обслуживания автобусов.

# 1 Постановка задачи

Создание программы «Автобусный парк». Необходимые данные включают:

* госномер,
* водитель,
* маршрут,
* вместимость,
* год выпуска,

Для реализации необходимо определить структуру базы данных предусмотреть **надежную валидацию данных**, вводимых пользователем, чтобы исключить ошибки при добавлении или редактировании записей (например, проверка формата госномера, допустимых значений маршрута и года выпуска). Это обеспечит корректную работу всей базы данных и повысит устойчивость приложения к некорректным действиям пользователя.

Приложение должно поддерживать добавление, удаление и редактирование записей, сохранения, загрузки базы данных, а также обеспечивать возможность сортировки, поиска и фильтрации списков по заданным критериям. Помимо этого, должна быть реализована функция удаления списков.

Для организации базы данных необходимо использовать многомодульную структуру, сложные типы данных и коллекции. При запуске приложения должно отображаться окно с информацией об авторе и теме проекта.

Важно создать интуитивно понятный интерфейс с чётко обозначенными его пунктами меню, определить тип и распределить зоны для отображения меню и таблицы с данными в консоли. Также следует изучить механизмы работы с файлами, чтобы полноценно реализовать функциональность программы [3].

Основная цель разработки — создание удобного и функционального интерфейса, обеспечивающего простое и эффективное выполнение ключевых задач программы.

В рамках постановки задачи была разработана диаграмма вариантов использования, представленная на рисунке 1. В ней можно увидеть основные функции, которые необходимо реализовать.

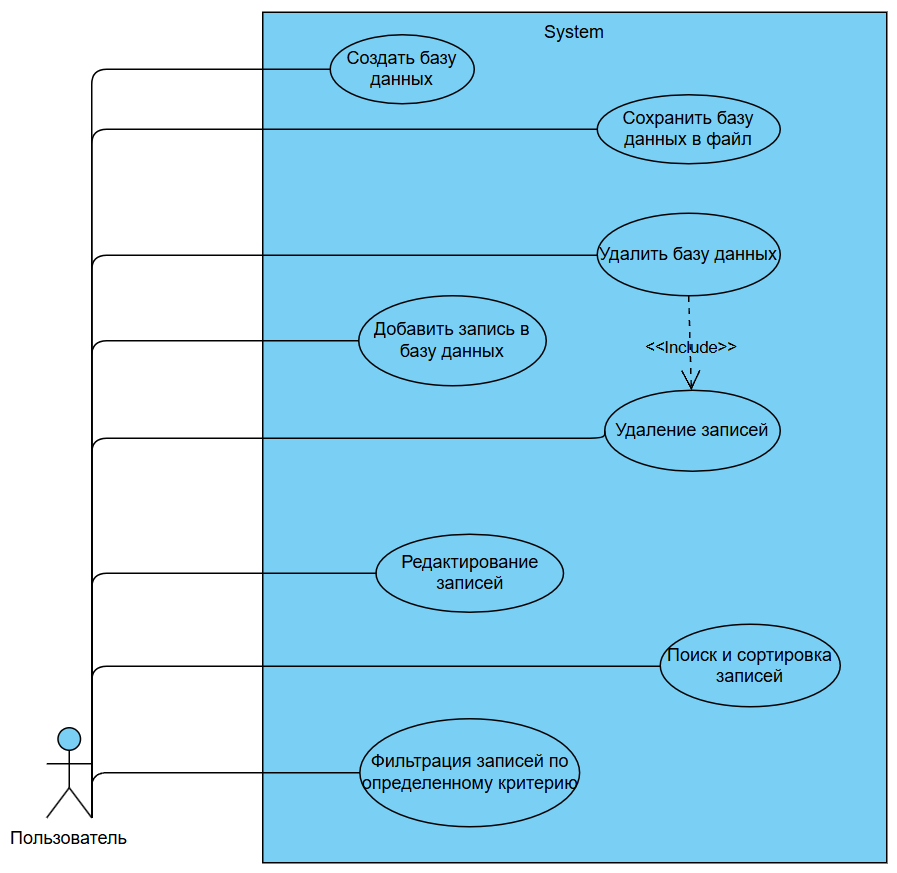


Рисунок 1 – Диаграмма вариантов использования

# 2 Выбор решения

Архитектура решения включает два основных компонента: интерфейсный модуль и модуль работы с базой данных. Схема их организации и взаимодействия демонстрируется на рисунке 2, где представлена диаграмма классов, отражающая ключевые связи между элементами программы.

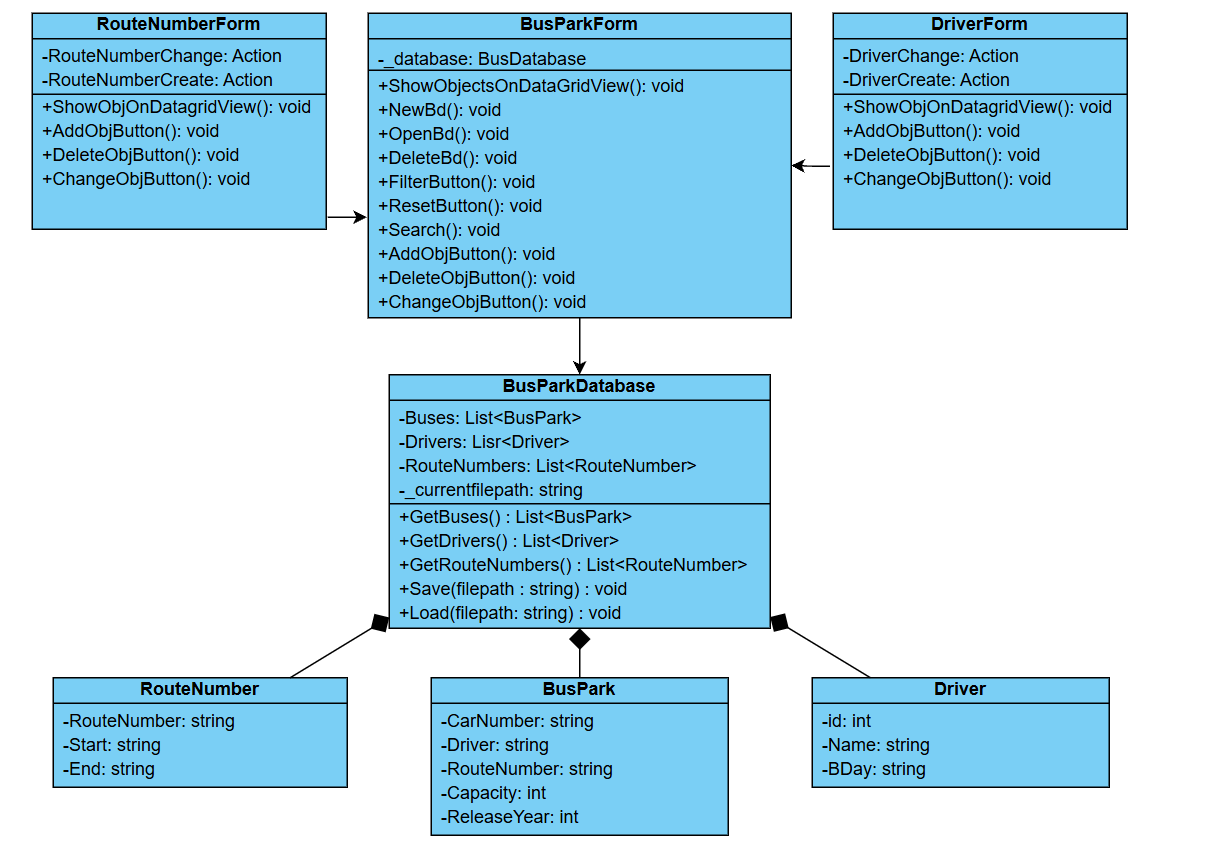


Рисунок 2 – Диаграмма классов

## 2.1 Определение необходимых модулей программы

Модуль для работы с коллекциями данных предоставляет методы для сохранения коллекций объектов в файл и загрузки информации из сохраненного файла. Включает методы Load для загрузки данных из файла в формате JSON и Save для записи текущих данных в файл. Кроме того, можно добавить, что данный класс использует сериализацию и десериализацию JSON через JsonSerializer для обработки данных. Код данного модуля приведен ниже [4].

public class BusDatabase

{

public List<BusPark> Buses { get; set; }

public List<Driver> Drivers { get; set; }

public List<RouteNumber> RouteNumbers { get; set; }

public string \_currentFilePath;

public BusDatabase()

public BusDatabase(List<BusPark> buses)

public List<BusPark> GetBuses()

public void Save(string filePath)

public static BusDatabase Load(string filePath)

}

В программе присутствует модуль для реализации взаимодействия пользователя и коллекции. Для этого был реализован удобный пользовательский интерфейс, в котором реализованы основные операции. Пример реализации данного модуля приведен ниже.

public partial class Form1 : Form

{

private BusDatabase \_database;

public Form1()

private void UpdateMenuItems()

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

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

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

private void UpdateAllUI()

private void UpdateDriversComboBox()

private void UpdateRoutesComboBox()

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

private void AddCell(PdfPTable table, string text, iTextSharp.text.Font font

private void button2\_Click(object sender, EventArgs e)

private void button3\_Click(object sender, EventArgs e)

private void button6\_Click(object sender, EventArgs e)

private void ShowObjectsOnDataGridView()

private void textBox4\_TextChanged(object sender, EventArgs e)

private void PerformSearch()

public void Form2\_DriversChanged(string changeDriver)

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

private void Form3\_RouteCreate()

private void Form3\_RouteChanged(string changeRoute)

Модель интерфейса программы представлена на рисунке 3.

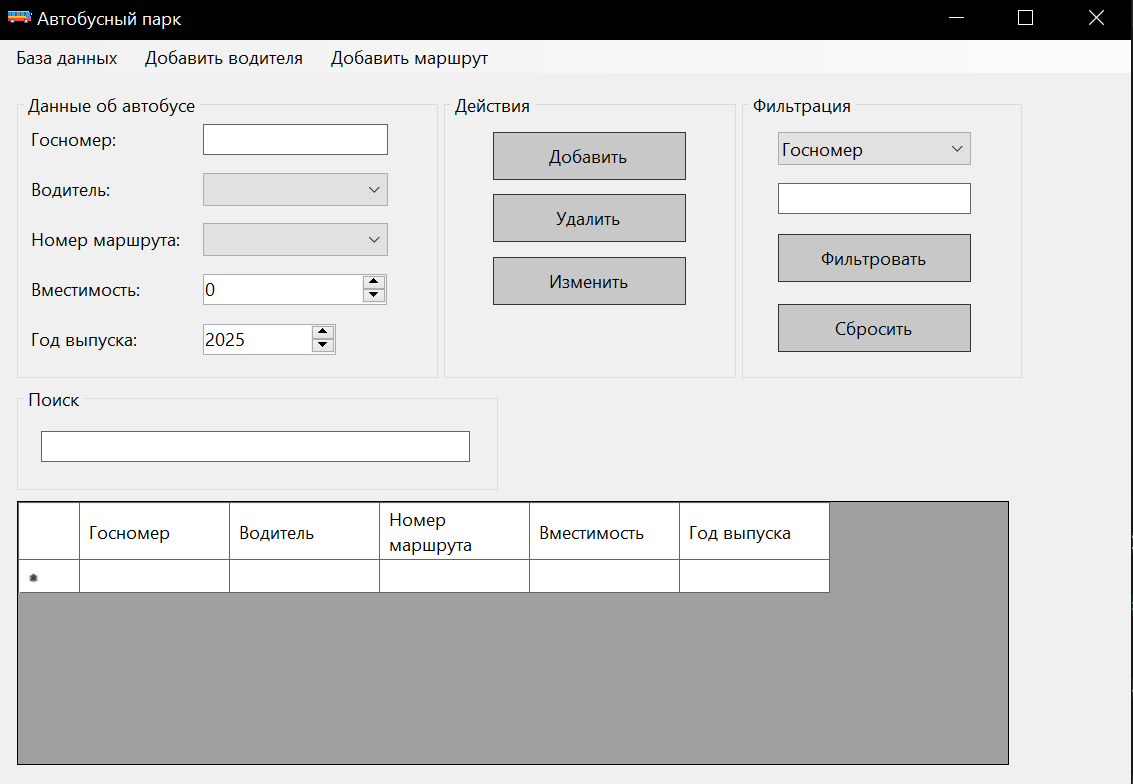


Рисунок 3 – Модель пользовательского интерфейса

Пользователь имеет возможность создать новую коллекцию и удалить уже существующую. Есть возможность изменить существующее поле и вывести информацию всей коллекции после данного действия.

## 2.2 Определение структуры файла базы данных

В качестве средства для хранения данных выбрана JSON сериализация в файл. Структура файла базы данных основывается на трех основных коллекциях: коллекции объектов BusPark (автобусный парк) и коллекции объектов Driver (водитель) и RouteNumber (номер маршрута). Каждый объект BusPark содержит информацию о автобусе, включая госномер, водителя, номер маршрута, вместимость, год выпуска. В свою очередь, объект Driver хранит информацию о водителе, включая его id, ФИО и дату рождения. Объект RouteNumber хранит информацию о номере маршрута, включая сам номер маршрута, начальную и конечную остановку.

Ниже, на рисунке 4, приведена модель базы данных, использующаяся в курсовой работе.

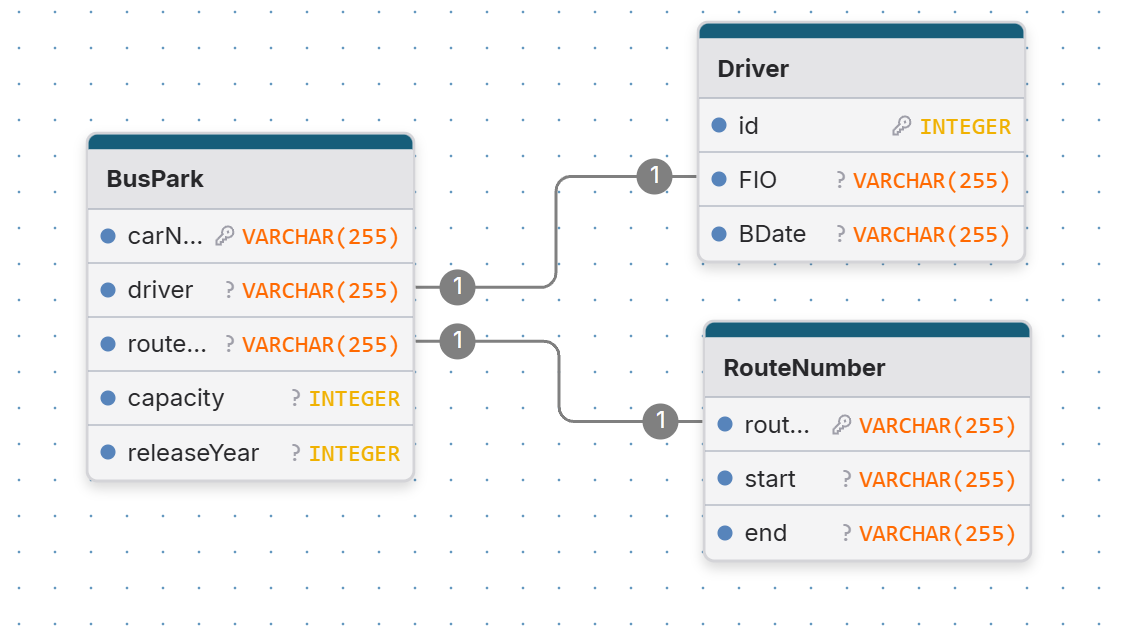


Рисунок 4 – Модель базы данных

# 3 Описание разработки программы

## 3.1 Кодирование

В ходе выполнения курсовой работы было разработана программа «Автобусный парк», код которого приведен в приложении А.

Для алгоритма работы открытия БД была разработана диаграмма деятельности (рисунок 5) [5].

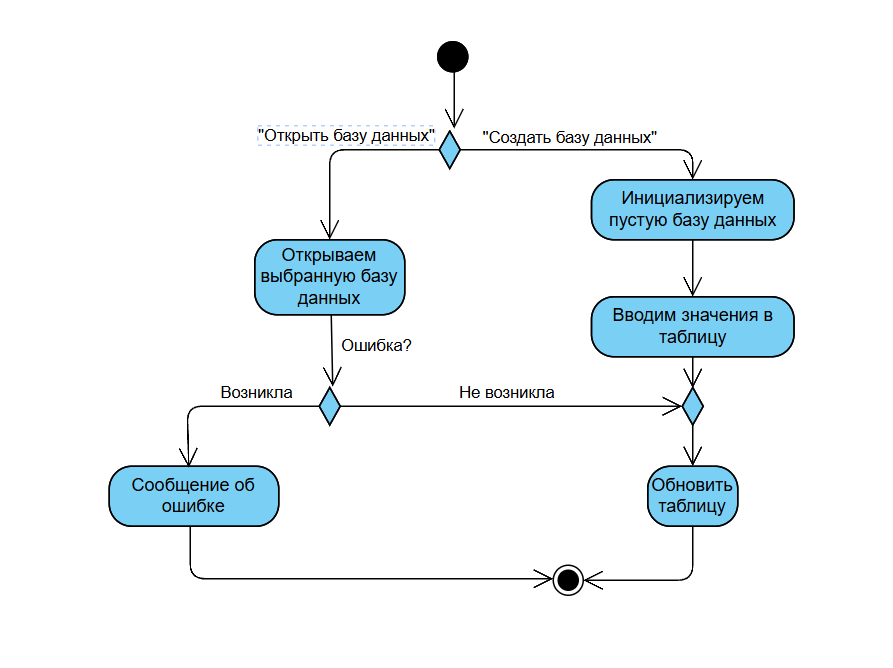


Рисунок 5 – Диаграмма деятельности

На диаграмме можно увидеть, что в начале программа смотрит что выбрал пользователь: “Открыть базу данных” – для выбора существующей базы данных или “Создать базу данных” – для создания новой базы данных. В первом случае программа проверяет на корректность файл, если все хорошо, то обновляем таблицу, в противном случаем выводим сообщение об ошибке. Во втором случае программа инициализирует пустую базу данных, в которую пользователь должен ввести данные для последующей работы с ней.

**3.2** Диаграмма компонентов

В процессе выполнения курсового проекта была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты и связи между ними (рисунок 6)

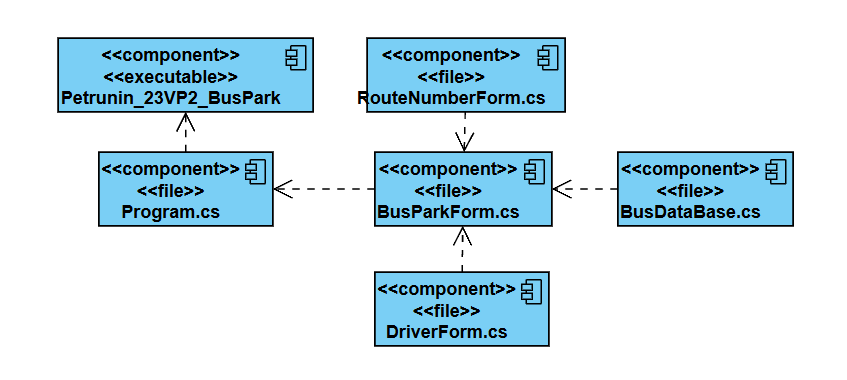


Рисунок 6 – Диаграмма компонентов

Описание компонентов приведено в таблице 1.

Таблица 1 – Компоненты

|  |  |
| --- | --- |
| Компоненты | Назначение |
| Program.cs | Главная программа |
| BusParkForm.cs | Файл с формой для добавления и редактирования автобусов |
| BusDataBase.cs | Файл с классом базы данных |
| RouteNumberForm.cs | Файл с формой для добавления и редактирования маршрутов |
| DriverForm.cs | Файл с формой для добавления и редактирования водителей |

# 4 Отладка и тестирование

## 4.1. Функциональное тестирование

Функциональное тестирование – это процесс проверки программного обеспечения на соответствие требованиям и спецификации продукта. Оно сосредоточено на внешнем поведении системы, не принимая во внимание её внутреннюю структуру. В ходе функционального тестирования анализируется правильность и полнота реализации определённых функций, а также соответствие продукта запросам клиента или пользователей. [6]

Каждая функция системы тестируется путём предоставления соответствующих входных данных, анализа выходных данных и сопоставления реальных результатов с ожидаемыми. Тестирование может осуществляться как вручную, так и с использованием автоматизированных инструментов.

## 4.2. Тестирование приложения

В курсовом проекте было выполнено функциональное тестирование разработанного программного обеспечения. Результаты тестирования приведены в таблице 2.

Таблица 2 – Функциональное тестирование

|  |  |  |
| --- | --- | --- |
| **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| Добавить автобус с полями, имеющую пустое поле | Появление окна с ошибкой и информацией о ней | Появилось окно с ошибкой и информацией о ней. (рисунок 7) |
| Добавить автобус с корректными данными | Появление в БД созданного автобуса | Появился в БД созданного автобуса (рисунок 8) |

Продолжение таблицы 2

|  |  |  |
| --- | --- | --- |
| Изменить вместимость автобуса | Изменение информации об автобусе | Изменилась информация в таблице(рисунок 9) |
| Сохранить БД в файл | Появление файла сохранения | Появился файл сохранения  (рисунок 10) |
| Удалить текущую БД | Стирается таблица | Таблица стёрлась  (рисунок 11) |
| Открыть БД | Появление сохранённых ранее данных | Появились данные (рисунок 12) |
| Поиск в БД | Стираются все данные не совпадающие со строкой поиска | Стерлись неподходящие данные (рисунок 13) |
| Фильтрация по критерию в БД | Стираются все неподходящие данные, значения критерия которых не совпадает со строкой поиска по критерию | Стерлись неподходящие данные (рисунок 14) |

Ниже, на рисунках 7-14, приведены скриншоты, отражающие результаты работы программы в процессе тестирования.

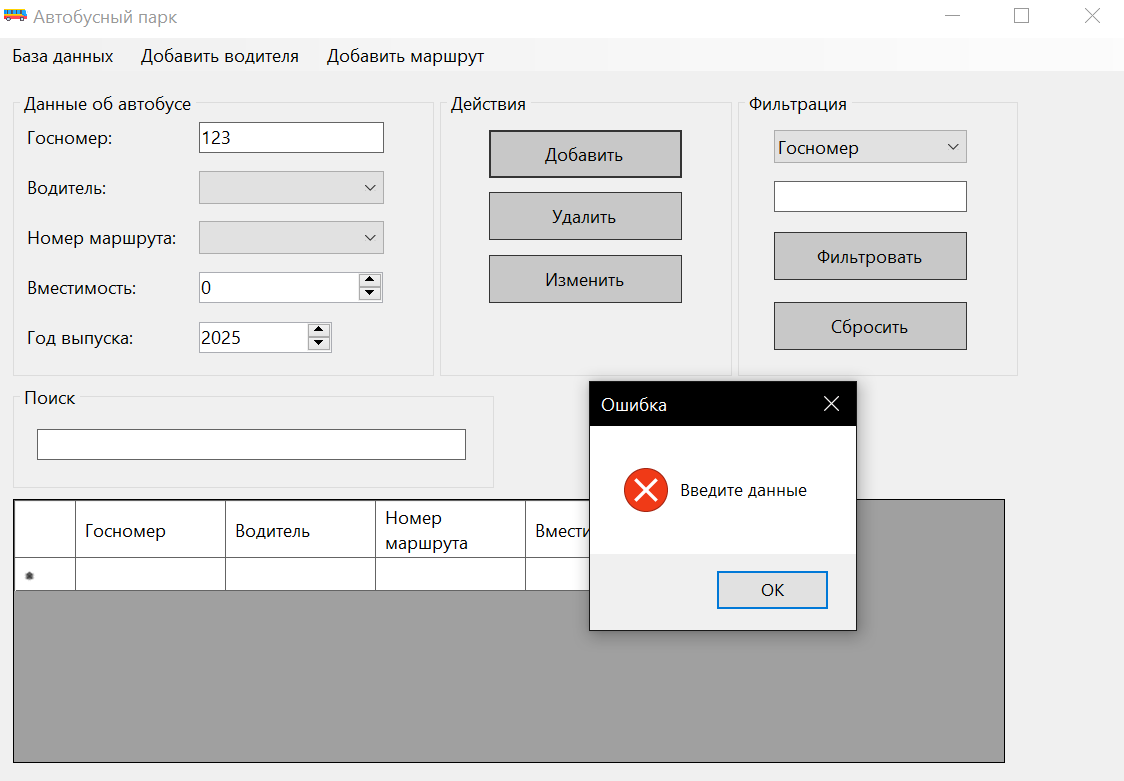


Рисунок 7 – Тест “Добавить автобус с полями, имеющую пустое поле”

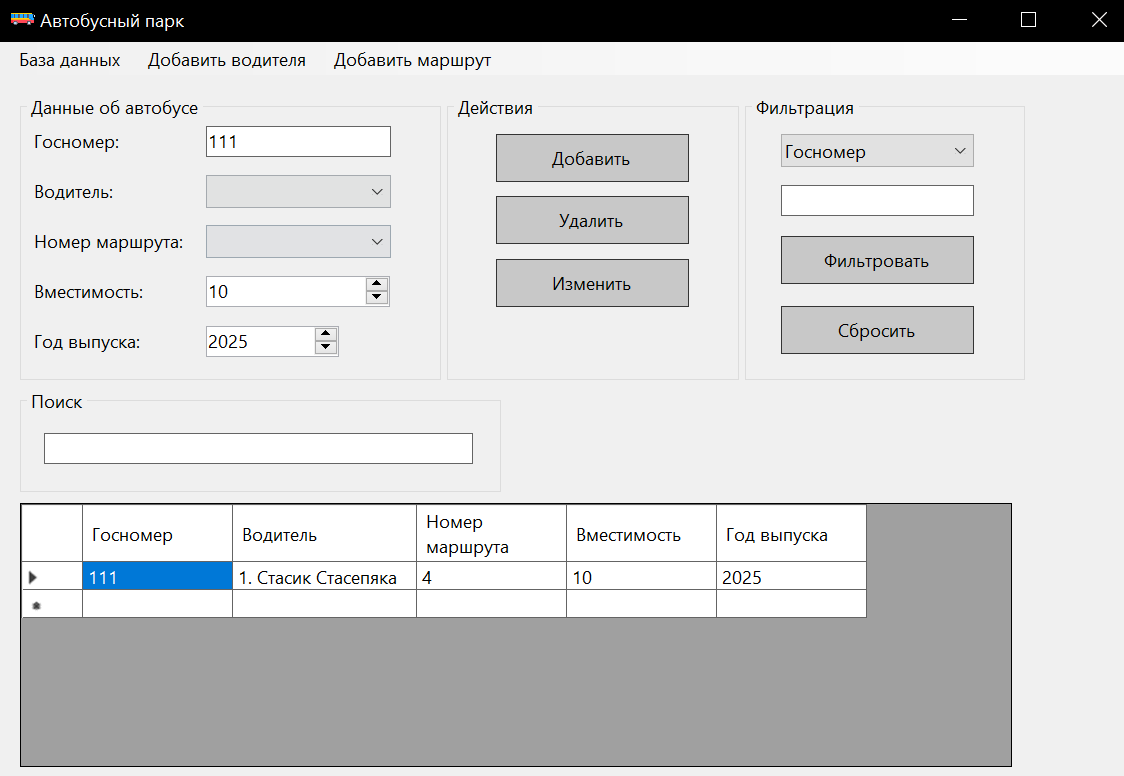


Рисунок 8 – Тест “Добавить автобус с корректными данными”

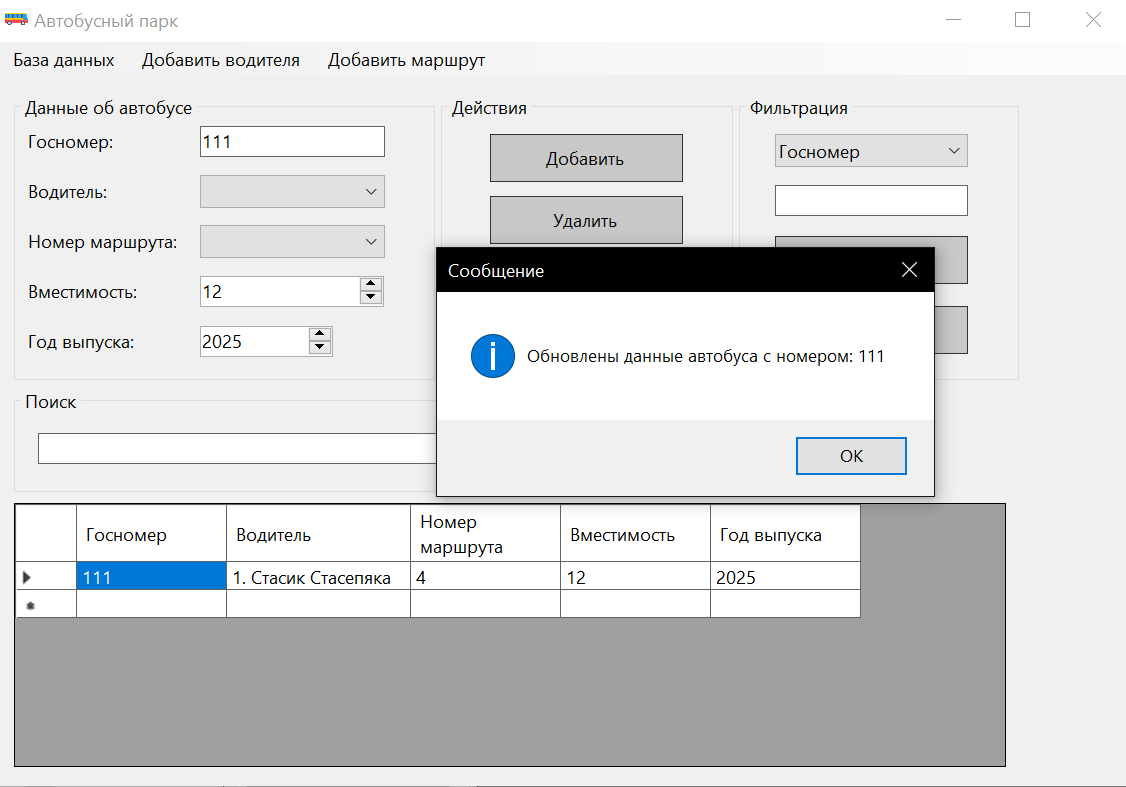


Рисунок 9 – Тест “ Изменить вместимость автобуса”

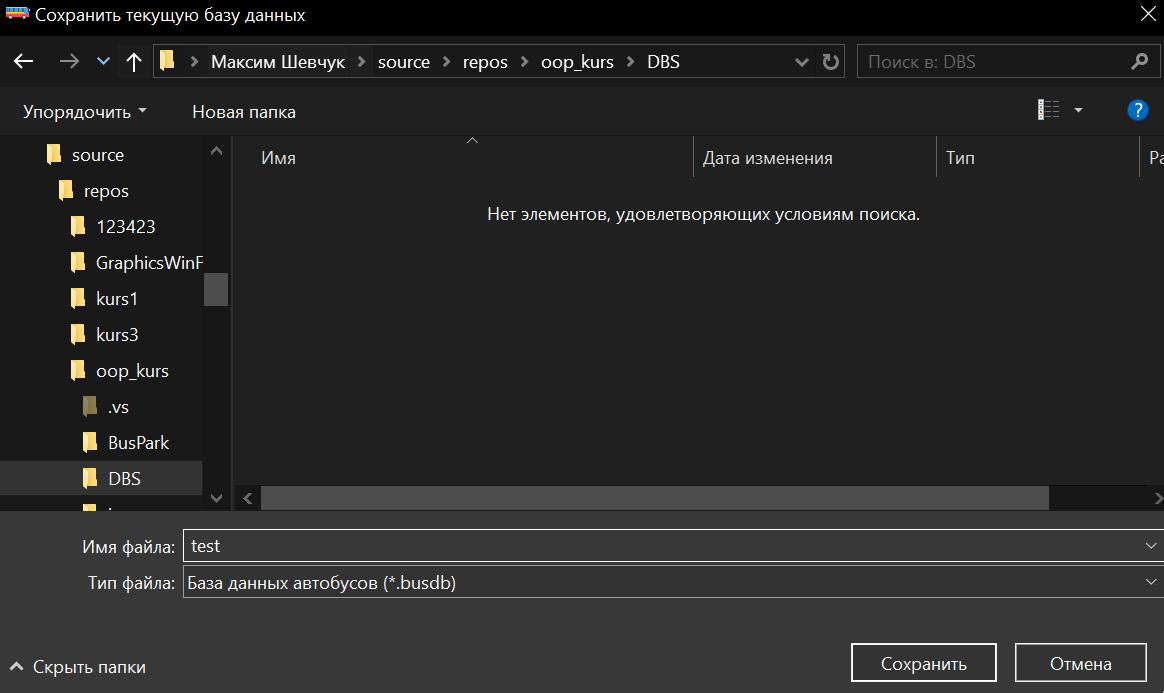


Рисунок 10 – Тест “Сохранить БД в файл”

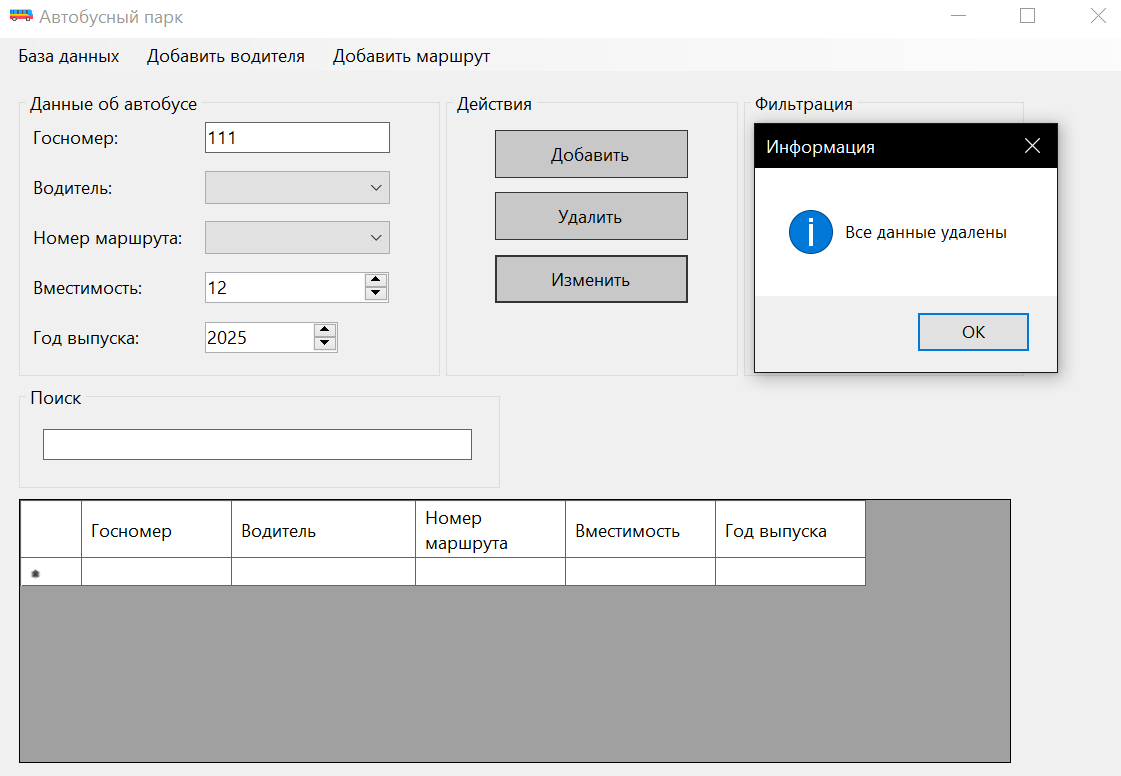


Рисунок 11 – Тест “Удалить текущую БД”

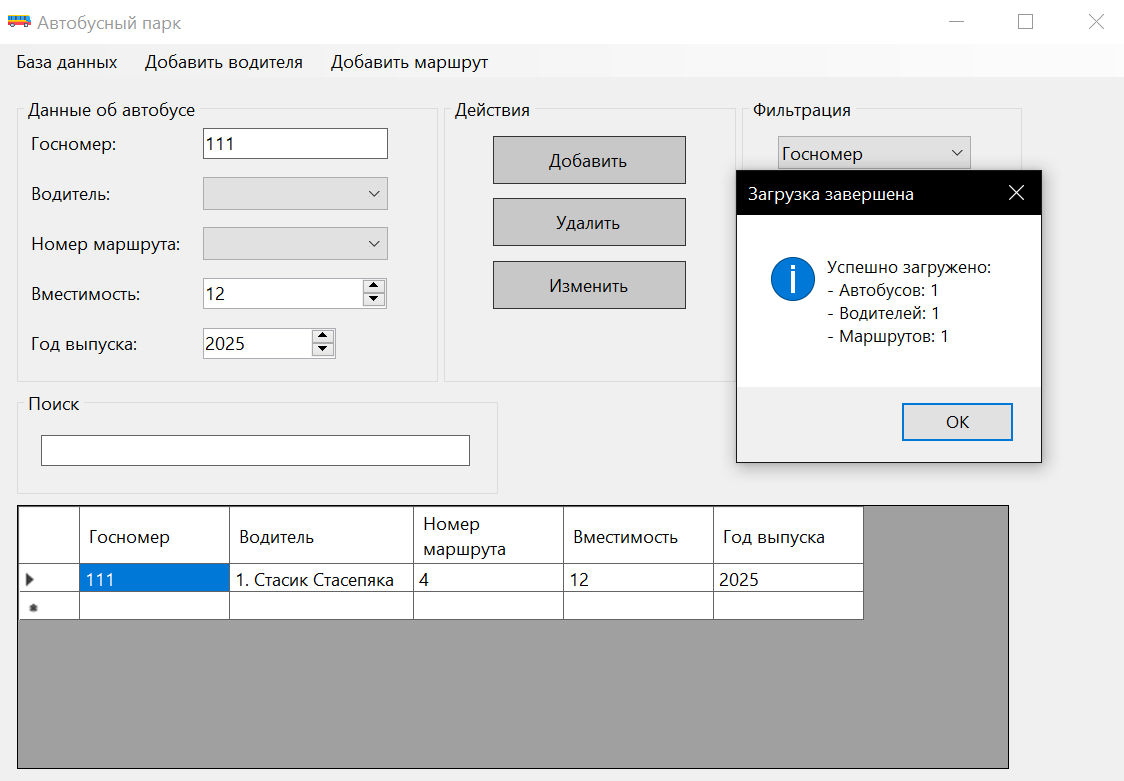


Рисунок 12 – Тест “Открыть БД”

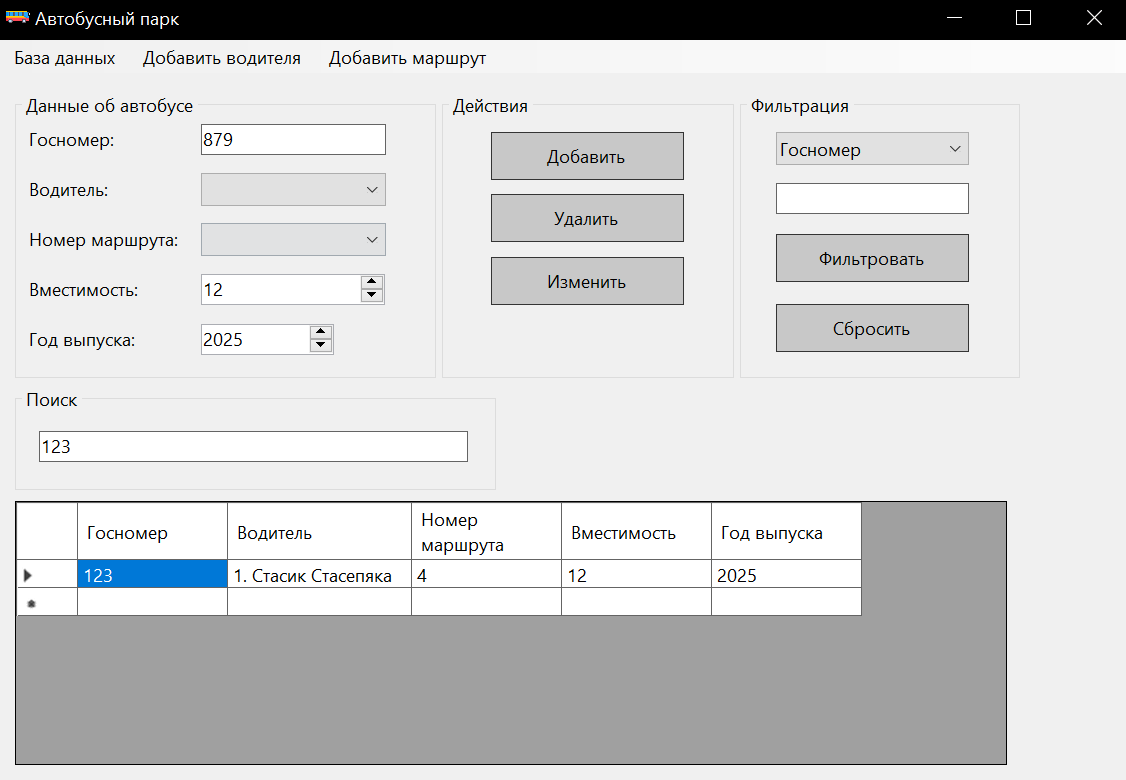


Рисунок 13 – Тест “Поиск по критерию в БД”

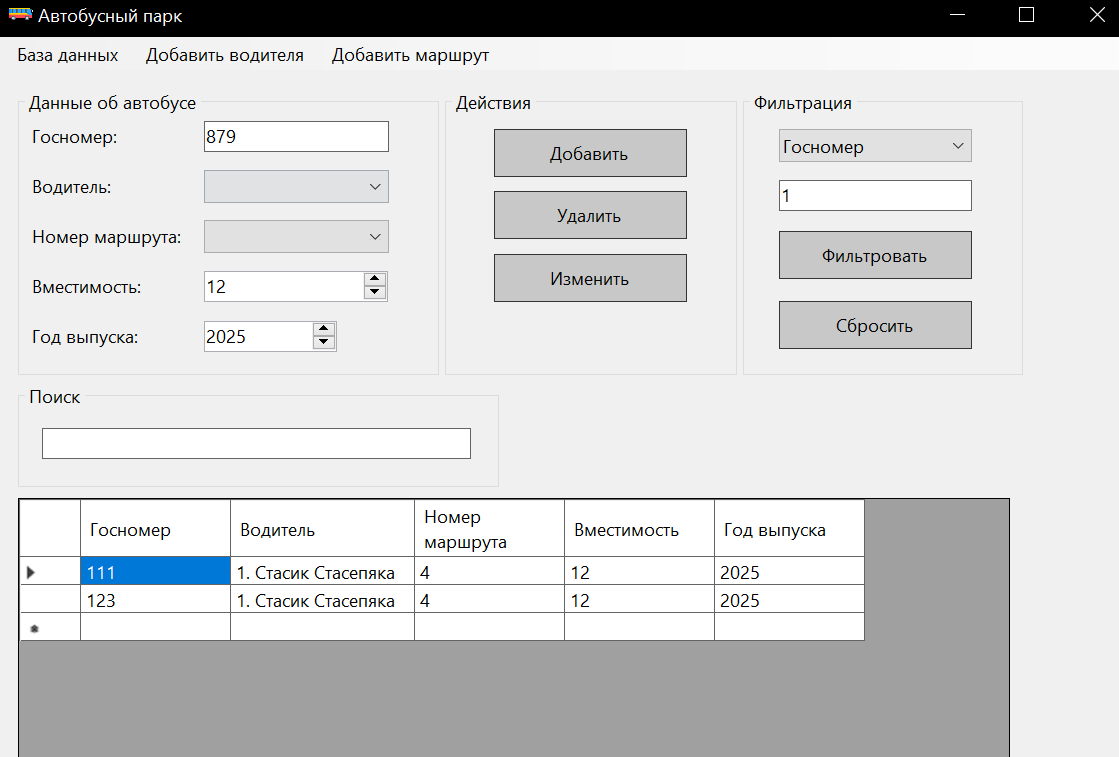


Рисунок 14 – Тест “ Фильтрация по критерию в БД ”

В ходе выполнения тестирования несовпадения ожидаемого и наблюдаемого результата не выявлены. Следовательно, можно сделать вывод, что программа работает корректно и готова к использованию.

# 5 Описание программы

## 5.1 Разработка приложения Petrunin\_23VP2\_BusPark.exe

Программа Petrunin\_23VP2\_BusPark.exe является основным модулем для работы с базой данных, предназначенной для хранения информации о автобусах, водителях и маршрутах. Программа разработана с учетом удобства пользователя и интуитивно понятного интерфейса, что позволяет легко выполнять операции по добавлению, редактированию, удалению записей, а также осуществлять поиск, фильтрацию и сортировку данных. Также предусмотрена возможность сохранения и загрузки базы данных, что обеспечивает долговременное хранение и актуальность информации [7].

При запуске программы, пользователю демонстрируется заставка с приветствием, после чего появляется главное окно с меню, из которого можно выбрать требуемую операцию. Программа ожидает действия пользователя, и при выборе одного из пунктов меню, управление передается в соответствующие функции, реализованные в файлах BusParkForm.cs.

Программа предоставляет пользователю возможность создавать новую базу данных, для чего достаточно выбрать пункт меню «Создать базу данных» в вкладке «Файл». Для работы с уже существующими данными, предусмотрена функция открытия базы данных. В этом случае пользователь выбрать файл базы данных из выбирает пункт «Открыть базу данных», после чего ему предложат директории. Это дает возможность загрузить ранее сохраненную информацию и продолжить работу с ней без потери данных.

Кроме того, программа поддерживает функцию сохранения базы данных. Пользователи могут выбрать пункт меню «Сохранить базу данных», чтобы сохранить текущие изменения в выбранной директории. При этом появится окно с выбором места и имени файла для сохранения, что обеспечивает удобство работы и сохранность данных.

Программа предоставляет возможность добавлять новые записи в базу данных. Для этого нужно выбрать пункт «Добавить», заполнить форму необходимыми данными и сохранить запись. Для редактирования существующих записей достаточно ввести госномер нужного автобуса в таблице, нажать «Редактировать» и внести необходимые изменения. Удаление записи выполняется просто — нужно ввести госномер нужного автобуса и нажать «Удалить».

Для эффективной работы с большим количеством записей программа также поддерживает поиск, фильтрацию и сортировку данных. Для поиска необходимо выбрать критерий, ввести значение и программа автоматически найдет все подходящие записи, подсвечивая их для удобства. Фильтрация данных позволяет отобразить только те записи, которые соответствуют заданному критерию. Сортировка позволяет упорядочить записи по различным параметрам, что значительно облегчает работу с большими объемами информации.

Все эти функции обеспечивают гибкость и удобство работы с базой данных, что делает программу мощным инструментом для хранения и обработки информации о автобусах, водителях и маршрутах [8].

# Выводы

В рамках курсового проекта была создана программа для управления базой данных, позволяющая выполнять операции с автобусами, маршрутами и водителями: добавление, редактирование, фильтрацию и удаление записей. Ключевая цель проекта — эффективное управление данными, их сохранение и последующее применение. Информация хранится в файловом формате, что обеспечивает актуальность базы данных и возможность её дальнейшего использования.

Программа реализована с использованием объектно-ориентированного подхода и поддерживает работу с коллекциями данных. Благодаря интуитивно понятному интерфейсу на Windows Forms пользователь может:

* создавать и редактировать базу данных,
* добавлять и удалять записи,
* фильтровать, искать и сортировать данные.

Для хранения информации выбран формат JSON, обеспечивающий удобство обработки и анализа данных.

Разработка проекта позволила закрепить навыки программирования на C# в Microsoft Visual Studio, углубить понимание принципов ООП и получить практический опыт в проектировании ПО. В ходе работы мы научились:

* проектировать функционал для взаимодействия с базой данных,
* разрабатывать графический интерфейс,
* обеспечивать сохранность данных.

На этапе тестирования были выявлены и исправлены ошибки, в результате чего программа полностью соответствует техническим требованиям и готова к эксплуатации.

Процесс разработки включал несколько этапов: анализ требований, проектирование, реализация и тестирование. Особое внимание уделялось структурированию данных и созданию удобного пользовательского интерфейса.

# Список использованных источников

1. Работа с базами данных в C# и .NET: учебное пособие / А.А. Лебедев – М.: ДМК Пресс, 2019. – 368 с.

2. Разработка баз данных в C#: учебное пособие / Е.Н. Сидоров – М.: Изд-во МГТУ, 2020. – 352 с.

3. Visual C# на примерах / М.Э. Абрамян – М.: ДМК Пресс, 2008. – 560 с.

4. Программирование на C#: учебное пособие / В.В. Поляков – М.: Юрайт, 2021. – 448 с.

5. Моделирование информационных систем. Unified Modeling Language. Учебное пособие для вузов: учебное пособие / И.Ю. Матюшичев, А.В. Флегонтов, Лань, - 2020 – 352 с.

6. Тестирование программного обеспечения: Учебное пособие: учебное пособие / Ю.В. Морозова, ТУСУР, 2019. — 120 с.

7. C# для начинающих / Б.И. Пахомов – СПб. : БХВ-Петербург, 2020. – 416 с.

8. Разработка на языке C# приложений с графическим интерфейсом (использование Windows Forms): учеб. пособие / С.Э. Мурадханов – Москва : Изд-во «МИСиС», 2019. – 396 с.

# Приложение A – Код программы

Program.cs

namespace oop\_kurs

{

internal static class Program

{

/// <summary>

/// The main entry point for the application.

/// </summary>

[STAThread]

static void Main()

{

// To customize application configuration such as set high DPI settings or default font,

// see https://aka.ms/applicationconfiguration.

System.Text.Encoding.RegisterProvider(System.Text.CodePagesEncodingProvider.Instance);

ApplicationConfiguration.Initialize();

Application.Run(new Form1());

}

}

}

BusParkForm.cs

using iTextSharp.text;

using iTextSharp.text.pdf;

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Drawing.Printing;

using System.IO;

using System.Linq;

using System.Reflection.Metadata;

using System.Text.Json;

using System.Text.Json.Serialization;

using System.Windows.Forms;

using System.Xml.Linq;

using static System.Windows.Forms.DataFormats;

namespace oop\_kurs

{

public partial class Form1 : Form

{

private BusDatabase \_database;

public Form1()

{

InitializeComponent();

comboBox1.Items.AddRange(new string[] { "Госномер", "Водитель", "Номер маршрута", "Вместимость", "Год выпуска" });

comboBox1.SelectedIndex = 0;

\_database = new BusDatabase();

UpdateMenuItems();

}

private void UpdateMenuItems()

{

saveToolStripMenuItem.Enabled = \_database != null && !string.IsNullOrEmpty(\_database.\_currentFilePath) && !(BusPark.busList.Count == 0);

newToolStripMenuItem.Enabled = !(BusPark.busList.Count == 0);

deleteAllToolStripMenuItem.Enabled = !(BusPark.busList.Count == 0);

}

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

{

// Проверяем, есть ли данные для сохранения

if (BusPark.busList.Count > 0 || Driver.Drivers.Count > 0 || RouteNumber.RouteNumbers.Count > 0)

{

using (SaveFileDialog saveDialog = new SaveFileDialog())

{

saveDialog.Filter = "База данных автобусов (\*.busdb)|\*.busdb";

saveDialog.DefaultExt = "busdb";

saveDialog.Title = "Сохранить текущую базу данных";

if (saveDialog.ShowDialog() == DialogResult.OK)

{

try

{

// Создаем временную базу со всеми текущими данными

var tempDb = new BusDatabase

{

Buses = new List<BusPark>(BusPark.busList),

Drivers = new List<Driver>(Driver.Drivers),

RouteNumbers = new List<RouteNumber>(RouteNumber.RouteNumbers)

};

// Сохраняем все данные

tempDb.Save(saveDialog.FileName);

MessageBox.Show("Все данные успешно сохранены:\n" +

$"- Автобусов: {BusPark.busList.Count}\n" +

$"- Водителей: {Driver.Drivers.Count}\n" +

$"- Маршрутов: {RouteNumber.RouteNumbers.Count}",

"Сохранение завершено",

MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при сохранении: {ex.Message}",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

return;

}

UpdateAllUI();

}

else

{

return; // Пользователь отменил сохранение

}

}

}

else

{

MessageBox.Show($"Добавьте элементы!",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

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

{

var result = MessageBox.Show("Вы уверены, что хотите удалить все данные?",

"Подтверждение удаления",

MessageBoxButtons.YesNo,

MessageBoxIcon.Warning);

if (result == DialogResult.Yes)

{

// Очищаем все данные

BusPark.busList.Clear();

BusPark.originalBusList.Clear();

Driver.Drivers.Clear();

RouteNumber.RouteNumbers.Clear();

// Обновляем интерфейс

comboBox2.Items.Clear();

comboBox3.Items.Clear();

UpdateAllUI();

// Сбрасываем базу данных

\_database = new BusDatabase();

\_database.\_currentFilePath = null;

MessageBox.Show("Все данные удалены", "Информация",

MessageBoxButtons.OK, MessageBoxIcon.Information);

}

}

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

{

using (OpenFileDialog openDialog = new OpenFileDialog())

{

openDialog.Filter = "База данных автобусов (\*.busdb)|\*.busdb";

openDialog.Title = "Открыть базу данных";

if (openDialog.ShowDialog() == DialogResult.OK)

{

try

{

// Загружаем все данные из файла

var loadedDb = BusDatabase.Load(openDialog.FileName);

// Обновляем статические списки

BusPark.busList = new List<BusPark>(loadedDb.Buses);

Driver.Drivers = new List<Driver>(loadedDb.Drivers);

RouteNumber.RouteNumbers = new List<RouteNumber>(loadedDb.RouteNumbers);

BusPark.originalBusList = new List<BusPark>(BusPark.busList);

\_database.\_currentFilePath = openDialog.FileName;

// Обновляем интерфейс

UpdateAllUI();

MessageBox.Show($"Успешно загружено:\n" +

$"- Автобусов: {BusPark.busList.Count}\n" +

$"- Водителей: {Driver.Drivers.Count}\n" +

$"- Маршрутов: {RouteNumber.RouteNumbers.Count}",

"Загрузка завершена",

MessageBoxButtons.OK,

MessageBoxIcon.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка загрузки: {ex.Message}",

"Ошибка",

MessageBoxButtons.OK,

MessageBoxIcon.Error);

}

}

}

}

private void UpdateAllUI()

{

// Обновляем основную таблицу

ShowObjectsOnDataGridView();

// Обновляем комбобоксы

UpdateDriversComboBox();

UpdateRoutesComboBox();

// Обновляем меню

UpdateMenuItems();

}

private void UpdateDriversComboBox()

{

comboBox2.Items.Clear();

foreach (Driver driver in Driver.Drivers)

{

comboBox2.Items.Add(driver.ID + ". " + driver.FIO);

}

}

private void UpdateRoutesComboBox()

{

comboBox3.Items.Clear();

foreach (RouteNumber route in RouteNumber.RouteNumbers)

{

comboBox3.Items.Add(route.routeNumber);

}

}

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

{

string fontPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Fonts), "arial.ttf");

// 2. Проверка существования файла шрифта

if (!File.Exists(fontPath))

{

MessageBox.Show("Шрифт Arial не найден! Используется стандартный шрифт.");

fontPath = Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.Fonts), "times.ttf");

}

// 3. Создаем базовый шрифт с поддержкой кириллицы

BaseFont baseFont;

try

{

baseFont = BaseFont.CreateFont(fontPath, BaseFont.IDENTITY\_H, BaseFont.EMBEDDED);

}

catch

{

// Если не удалось загрузить шрифт, используем встроенный

baseFont = BaseFont.CreateFont(BaseFont.HELVETICA, BaseFont.CP1252, BaseFont.NOT\_EMBEDDED);

}

// 4. Создаем основной шрифт для документа

iTextSharp.text.Font mainFont = new iTextSharp.text.Font(baseFont, 10);

iTextSharp.text.Font headerFont = new iTextSharp.text.Font(baseFont, 12, iTextSharp.text.Font.BOLD);

using (SaveFileDialog saveDialog = new SaveFileDialog())

{

saveDialog.Filter = "PDF файлы (\*.pdf)|\*.pdf";

if (saveDialog.ShowDialog() == DialogResult.OK)

{

try

{

// Создаем документ

iTextSharp.text.Document document = new iTextSharp.text.Document();

PdfWriter.GetInstance(document, new FileStream(saveDialog.FileName, FileMode.Create));

document.Open();

// Добавляем заголовок

Paragraph title = new Paragraph("Отчет по автобусам", headerFont);

title.Alignment = Element.ALIGN\_CENTER;

document.Add(title);

// Создаем таблицу

PdfPTable table = new PdfPTable(5);

table.WidthPercentage = 100;

// Заголовки таблицы

AddCell(table, "Госномер", mainFont);

AddCell(table, "Водитель", mainFont);

AddCell(table, "Маршрут", mainFont);

AddCell(table, "Вместимость", mainFont);

AddCell(table, "Год выпуска", mainFont);

// Данные

foreach (var bus in BusPark.busList)

{

AddCell(table, bus.CarNumber, mainFont);

AddCell(table, bus.Driver, mainFont);

AddCell(table, bus.RouteNumber, mainFont);

AddCell(table, bus.Capacity.ToString(), mainFont);

AddCell(table, bus.ReleaseYear.ToString(), mainFont);

}

document.Add(table);

document.Close();

MessageBox.Show("PDF успешно создан!");

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка: {ex.Message}");

}

}

}

}

private void AddCell(PdfPTable table, string text, iTextSharp.text.Font font)

{

PdfPCell cell = new PdfPCell(new Phrase(text, font));

cell.Padding = 5;

cell.BorderWidth = 0.5f;

table.AddCell(cell);

}

private void button1\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBox1.Text) || comboBox2.SelectedIndex == -1 || comboBox3.SelectedIndex == -1)

{

MessageBox.Show("Введите данные", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

string carnumber = textBox1.Text;

string driver = comboBox2.SelectedItem.ToString();

string routenumber = comboBox3.SelectedItem.ToString();

int capacity = Convert.ToInt32(numericUpDown1.Value);

int realeseyear = Convert.ToInt32(numericUpDown2.Value);

foreach (BusPark exitingBus in BusPark.busList)

{

if (exitingBus.CarNumber == carnumber)

{

throw new Exception("Автобус с номером " + carnumber + " уже есть!");

}

}

BusPark newBusPark = new BusPark(carnumber, driver, routenumber, capacity, realeseyear);

BusPark.busList.Add(newBusPark);

BusPark.originalBusList = new List<BusPark>(BusPark.busList);

UpdateAllUI();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при добавлении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button2\_Click(object sender, EventArgs e)

{

try

{

string carnumber = textBox1.Text;

string driver = comboBox2.SelectedItem.ToString();

string routenumber = comboBox3.SelectedItem.ToString();

int capacity = Convert.ToInt32(numericUpDown1.Value);

int realeseyear = Convert.ToInt32(numericUpDown2.Value);

BusPark busToUpdate = null;

foreach (BusPark bus in BusPark.busList)

{

if (bus.CarNumber == carnumber)

{

busToUpdate = bus;

break;

}

}

if (busToUpdate != null)

{

busToUpdate.Driver = driver;

busToUpdate.RouteNumber = routenumber;

busToUpdate.Capacity = capacity;

busToUpdate.ReleaseYear = realeseyear;

UpdateAllUI();

BusPark.originalBusList = new List<BusPark>(BusPark.busList);

MessageBox.Show($"Обновлены данные автобуса с номером: {carnumber}", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show($"Не найден автобус с номером: {carnumber}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при изменении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button3\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBox1.Text))

{

MessageBox.Show("Введите госномер автобуса для удаления", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

string carNumberToDelete = textBox1.Text;

bool isDeleted = false;

for (int i = 0; i < BusPark.busList.Count; i++)

{

if (BusPark.busList[i].CarNumber == carNumberToDelete)

{

BusPark.busList.RemoveAt(i);

isDeleted = true;

break;

}

}

if (isDeleted)

{

BusPark.originalBusList = new List<BusPark>(BusPark.busList);

UpdateAllUI();

MessageBox.Show($"Автобус с номером {carNumberToDelete} удален", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

textBox1.Text = "";

comboBox2.SelectedItem = -1;

comboBox2.Text = "";

comboBox3.SelectedItem = -1;

comboBox3.Text = "";

numericUpDown1.Value = numericUpDown1.Minimum;

numericUpDown2.Value = numericUpDown2.Maximum;

}

else

{

MessageBox.Show($"Автобус с номером {carNumberToDelete} не найден", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при удалении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button6\_Click(object sender, EventArgs e)

{

if (string.IsNullOrEmpty(textBox5.Text))

{

MessageBox.Show("Введите значение для фильтрации", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

if (comboBox1.SelectedItem == null)

{

MessageBox.Show("Выберите категорию для фильтрации", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

string filterCategory = comboBox1.SelectedItem.ToString();

string filterValue = textBox5.Text.ToLower();

var filteredList = BusPark.originalBusList.Where(bus =>

{

switch (filterCategory)

{

case "Госномер":

return bus.CarNumber.ToLower().Contains(filterValue);

case "Водитель":

return bus.Driver.ToLower().Contains(filterValue);

case "Номер маршрута":

return bus.RouteNumber.ToLower().Contains(filterValue);

case "Вместимость":

return bus.Capacity.ToString().Contains(filterValue);

case "Год выпуска":

return bus.ReleaseYear.ToString().Contains(filterValue);

default:

return false;

}

}).ToList();

BusPark.busList = filteredList;

UpdateAllUI();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при фильтрации: " + ex.Message,

"Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void button5\_Click(object sender, EventArgs e)

{

BusPark.busList = new List<BusPark>(BusPark.originalBusList);

UpdateAllUI();

textBox5.Text = "";

}

private void ShowObjectsOnDataGridView()

{

dataGridView1.Rows.Clear();

try

{

foreach (BusPark bus in BusPark.busList)

{

dataGridView1.Rows.Add(

bus.CarNumber,

bus.Driver,

bus.RouteNumber,

bus.Capacity,

bus.ReleaseYear

);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка:" + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void textBox4\_TextChanged(object sender, EventArgs e)

{

Task.Delay(300).ContinueWith(\_ =>

{

this.Invoke((MethodInvoker)delegate

{

PerformSearch();

});

});

}

private void PerformSearch()

{

string searchText = textBox4.Text.Trim().ToLower();

if (string.IsNullOrEmpty(searchText))

{

UpdateAllUI();

return;

}

try

{

var searchResults = BusPark.busList.Where(bus =>

bus.CarNumber.ToLower().Contains(searchText) ||

bus.Driver.ToLower().Contains(searchText) ||

bus.RouteNumber.ToLower().Contains(searchText) ||

bus.Capacity.ToString().Contains(searchText) ||

bus.ReleaseYear.ToString().Contains(searchText))

.ToList();

dataGridView1.Rows.Clear();

foreach (var bus in searchResults)

{

dataGridView1.Rows.Add(

bus.CarNumber,

bus.Driver,

bus.RouteNumber,

bus.Capacity,

bus.ReleaseYear

);

}

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при поиске: {ex.Message}", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

public void Form2\_DriverCreate()

{

comboBox2.Items.Clear();

foreach (Driver driver in Driver.Drivers)

{

comboBox2.Items.Add(driver.ID + ". " + driver.FIO);

}

}

public void Form2\_DriversChanged(string changeDriver)

{

comboBox2.Items.Clear();

foreach (Driver driver in Driver.Drivers)

{

comboBox2.Items.Add(driver.ID + ". " + driver.FIO);

}

foreach (BusPark bus in BusPark.busList)

{

bool check = true;

foreach (Driver driver in Driver.Drivers)

{

if (bus.Driver == (driver.ID + ". " + driver.FIO))

{

check = false;

break;

}

}

if (check)

{

bus.Driver = changeDriver;

}

}

UpdateAllUI();

}

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

{

Form2 form2 = new Form2(this);

form2.OnDriversChanged += Form2\_DriversChanged;

form2.OnDriverCreate += Form2\_DriverCreate;

form2.Show();

form2.FormClosed += (s, args) =>

{

добавитьВодителяToolStripMenuItem.Enabled = true;

};

добавитьВодителяToolStripMenuItem.Enabled = false;

}

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

{

Form3 form3 = new Form3(this);

form3.OnRouteChanged += Form3\_RouteChanged;

form3.OnRouteCreate += Form3\_RouteCreate;

form3.Show();

form3.FormClosed += (s, args) =>

{

добавитьМаршрутToolStripMenuItem.Enabled = true;

};

добавитьМаршрутToolStripMenuItem.Enabled = false;

}

private void Form3\_RouteCreate()

{

comboBox3.Items.Clear();

foreach (RouteNumber routeNumber in RouteNumber.RouteNumbers)

{

comboBox3.Items.Add(routeNumber.routeNumber);

}

}

private void Form3\_RouteChanged(string changeRoute)

{

comboBox3.Items.Clear();

foreach (RouteNumber routeNumber in RouteNumber.RouteNumbers)

{

comboBox3.Items.Add(routeNumber.routeNumber);

}

foreach (BusPark bus in BusPark.busList)

{

bool check = true;

foreach (RouteNumber routeNumber in RouteNumber.RouteNumbers)

{

if (bus.RouteNumber == routeNumber.routeNumber)

{

check = false;

break;

}

}

if (check)

{

bus.RouteNumber = changeRoute;

}

}

UpdateAllUI();

}

}

}

DriverForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using static System.Windows.Forms.VisualStyles.VisualStyleElement;

namespace oop\_kurs

{

public partial class Form2 : Form

{

private Form1 mainForm;

public event Action OnDriverCreate;

public event Action<string> OnDriversChanged;

public Form2(Form1 form1)

{

InitializeComponent();

mainForm = form1;

ShowObjectsOnDataGridView();

}

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

{

if (numericUpDown1.Value == 0 || string.IsNullOrEmpty(textBox1.Text))

{

MessageBox.Show("Введите данные", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

int id = (int) numericUpDown1.Value;

string fio = textBox1.Text;

string bDate = dateTimePicker1.Value.ToShortDateString();

foreach (Driver exitingDriver in Driver.Drivers)

{

if (exitingDriver.ID == id)

{

throw new Exception("Водитель с индексом " + id + " уже есть!");

}

}

Driver newDriver = new Driver(id, fio, bDate);

Driver.Drivers.Add(newDriver);

OnDriverCreate?.Invoke();

ShowObjectsOnDataGridView();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при добавлении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

if (string.IsNullOrEmpty(textBox1.Text))

{

MessageBox.Show("Введите id водителя для удаления", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

int deleteId = (int) numericUpDown1.Value;

bool isDeleted = false;

for (int i = 0; i < Driver.Drivers.Count; i++)

{

if (Driver.Drivers[i].ID == deleteId)

{

Driver.Drivers.RemoveAt(i);

isDeleted = true;

break;

}

}

if (isDeleted)

{

OnDriversChanged?.Invoke("Отсутствует");

ShowObjectsOnDataGridView();

MessageBox.Show($"Водитель с id {deleteId} удален", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

numericUpDown1.Value = numericUpDown1.Minimum;

textBox1.Text = "";

dateTimePicker1.Value = dateTimePicker1.MinDate;

}

else

{

MessageBox.Show($"Водитель с id {deleteId} не найден", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при удалении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

try

{

int id = (int)numericUpDown1.Value;

string fio = textBox1.Text;

string bDate = dateTimePicker1.Value.ToShortDateString();

Driver newDriver = new Driver(id, fio, bDate);

Driver driverToUpdate = null;

foreach (Driver driver in Driver.Drivers)

{

if (driver.ID == id)

{

driverToUpdate = newDriver;

break;

}

}

if (driverToUpdate != null)

{

driverToUpdate.FIO = fio;

driverToUpdate.BDate = bDate;

ShowObjectsOnDataGridView();

OnDriversChanged?.Invoke(newDriver.ID + ". " + newDriver.FIO);

MessageBox.Show($"Обновлены данные водителя с id: {id}", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show($"Не найден водитель с id: {id}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при изменении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ShowObjectsOnDataGridView()

{

dataGridView1.Rows.Clear();

try

{

foreach (Driver driver in Driver.Drivers)

{

dataGridView1.Rows.Add(

driver.ID,

driver.FIO,

driver.BDate

);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка:" + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

}

}

RouteNumberForm.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace oop\_kurs

{

public partial class Form3 : Form

{

private Form1 mainForm;

public event Action OnRouteCreate;

public event Action<string> OnRouteChanged;

public Form3(Form1 form1)

{

InitializeComponent();

mainForm = form1;

ShowObjectsOnDataGridView();

}

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

{

if (string.IsNullOrEmpty(textBox1.Text) || string.IsNullOrEmpty(textBox2.Text)

|| string.IsNullOrEmpty(textBox3.Text))

{

MessageBox.Show("Введите данные", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

string routenumber = textBox1.Text;

string start = textBox2.Text;

string end = textBox3.Text;

foreach (RouteNumber exitingRoute in RouteNumber.RouteNumbers)

{

if (exitingRoute.routeNumber == routenumber)

{

throw new Exception("Маршрут с номером " + routenumber + " уже есть!");

}

}

RouteNumber newRoute = new RouteNumber(routenumber, start, end);

RouteNumber.RouteNumbers.Add(newRoute);

OnRouteCreate?.Invoke();

ShowObjectsOnDataGridView();

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при добавлении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

if (string.IsNullOrEmpty(textBox1.Text))

{

MessageBox.Show("Введите номер маршрута для удаления", "Ошибка",

MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

try

{

string deleteRouteNumber = textBox1.Text;

bool isDeleted = false;

for (int i = 0; i < RouteNumber.RouteNumbers.Count; i++)

{

if (RouteNumber.RouteNumbers[i].routeNumber == deleteRouteNumber)

{

RouteNumber.RouteNumbers.RemoveAt(i);

isDeleted = true;

break;

}

}

if (isDeleted)

{

OnRouteChanged?.Invoke("Отсутствует");

ShowObjectsOnDataGridView();

MessageBox.Show($"Маршрут с номером {deleteRouteNumber} удален", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

textBox1.Text = "";

textBox2.Text = "";

textBox3.Text = "";

}

else

{

MessageBox.Show($"Маршрут с номером {deleteRouteNumber} не найден", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при удалении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

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

{

try

{

string routenumber = textBox1.Text;

string start = textBox2.Text;

string end = textBox3.Text;

RouteNumber newRouteNumber = new RouteNumber(routenumber, start, end);

RouteNumber routeNumberToUpdate = null;

foreach (RouteNumber routeNumber in RouteNumber.RouteNumbers)

{

if (routeNumber.routeNumber == routenumber)

{

routeNumberToUpdate = newRouteNumber;

break;

}

}

if (routeNumberToUpdate != null)

{

routeNumberToUpdate.Start = start;

routeNumberToUpdate.End = end;

ShowObjectsOnDataGridView();

MessageBox.Show($"Обновлены для маршрута с номером: {routenumber}", "Сообщение", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

else

{

MessageBox.Show($"Не найден маршрут с номером: {routenumber}", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка при изменении: " + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

private void ShowObjectsOnDataGridView()

{

dataGridView1.Rows.Clear();

try

{

foreach (RouteNumber routenumber in RouteNumber.RouteNumbers)

{

dataGridView1.Rows.Add(

routenumber.routeNumber,

routenumber.Start,

routenumber.End

);

}

}

catch (Exception ex)

{

MessageBox.Show("Ошибка:" + ex.Message, "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

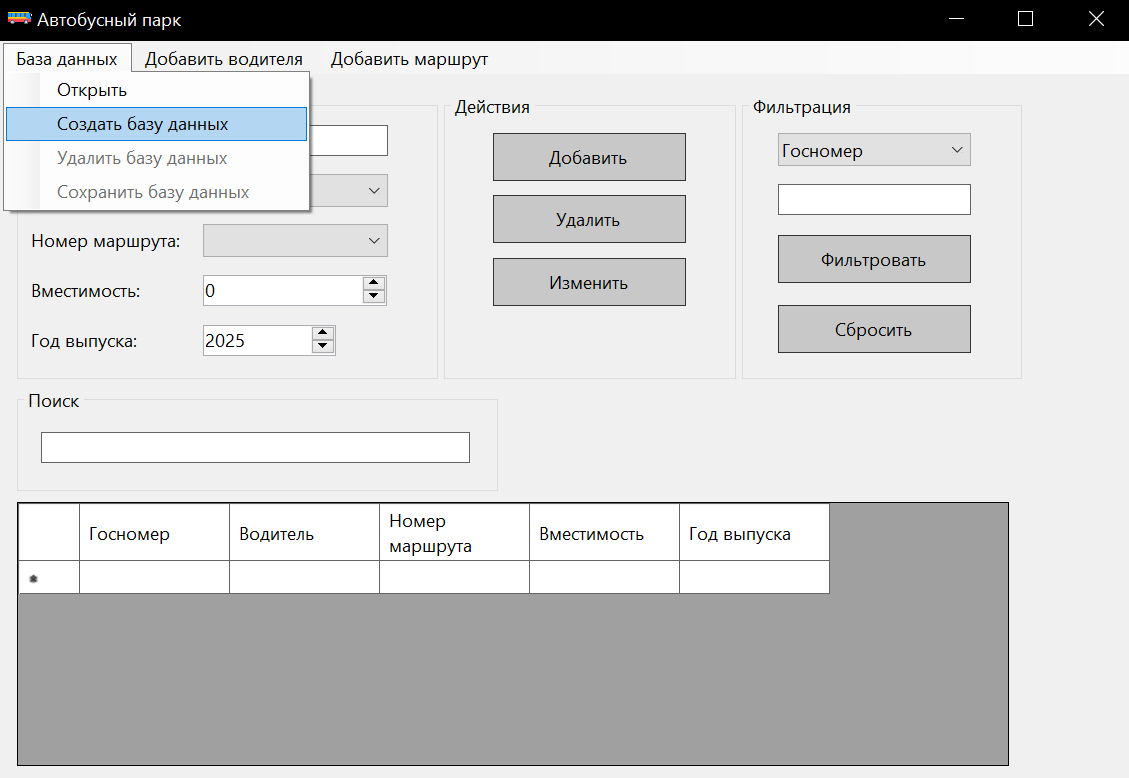
}

}

# Приложение B – Руководство пользователя

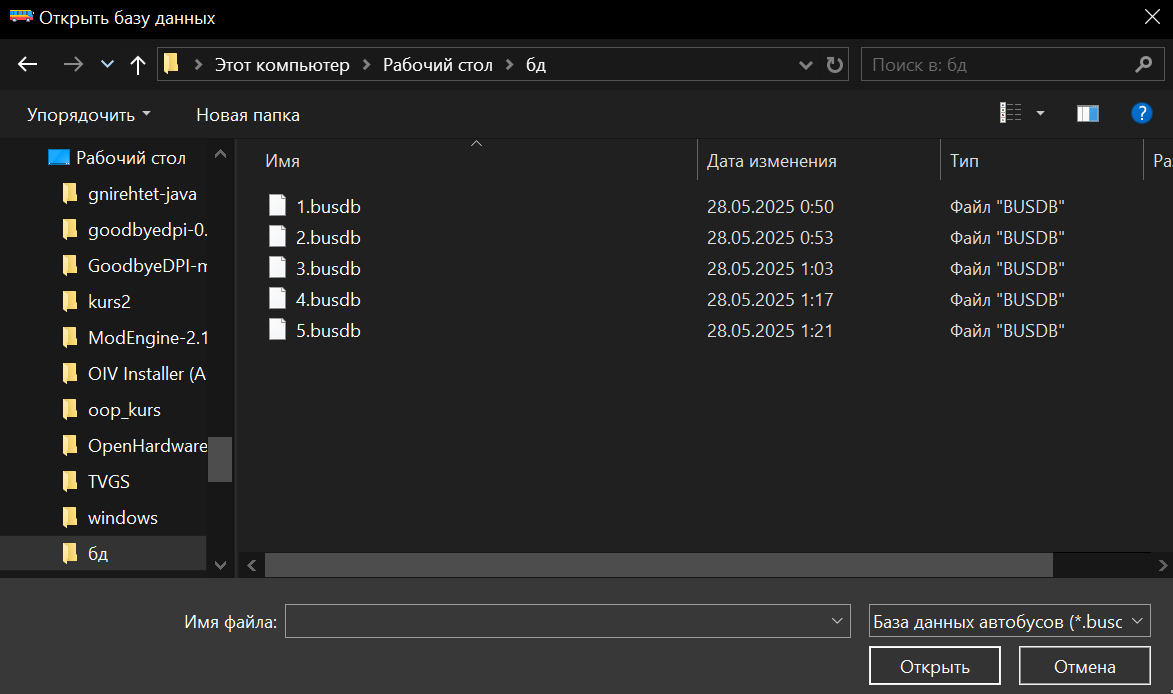
Программа Petrunin\_23VP2\_BusPark.exe предназначена для хранения информации о автобусах, водителях и маршрутах. Программа имеет интуитивно понятный интерфейс и поддерживает такие операции как добавления новой записи, удаления записи, редактирования записи, а также поиск, фильтрация и сортировка записи, сохранение существующей БД.

Для создания новой БД необходимо выбрать пункт меню «Создать базу данных» во вкладке «База данных» (рисунок В.1).



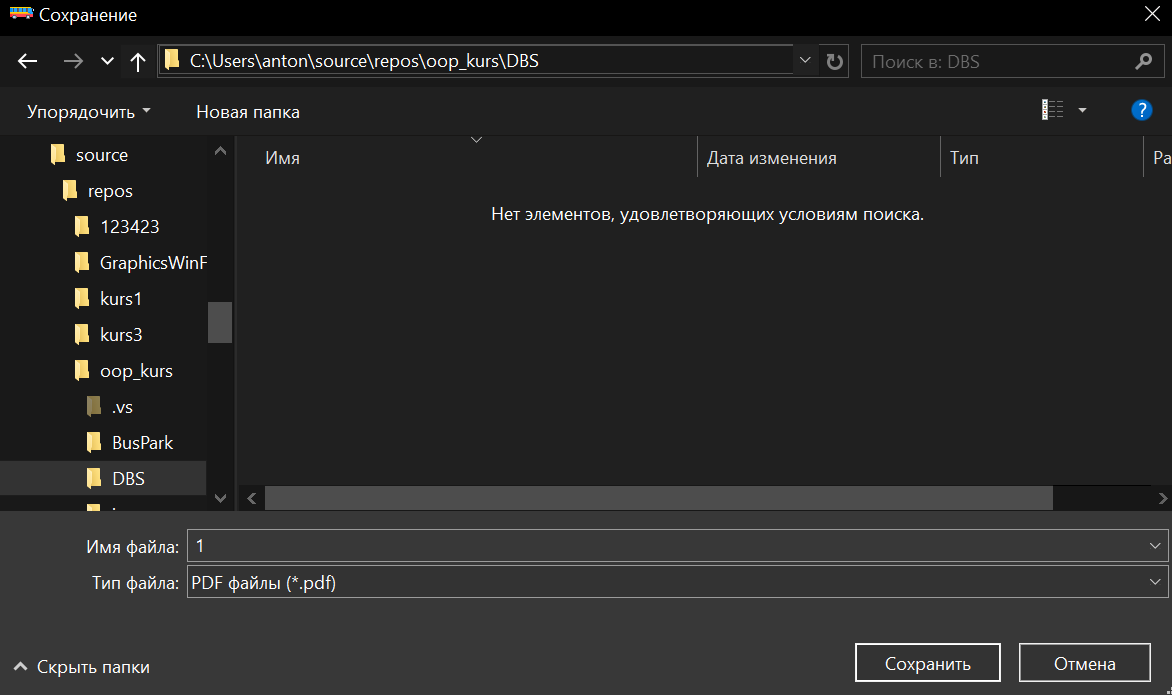
**Рисунок В.1 – Создание новой БД**

Для открытия и вывода на экран существующей БД необходимо выбрать пункт меню «Открыть» во вкладке «База данных» (рисунок В.1). Будет предложено выбрать файл из каталога.



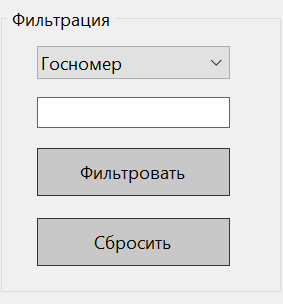
**Рисунок В.2 – Открытие существующей БД**

Для сохранения существующего каталога необходимо выбрать пункт меню «Сохранить базу данных.» во вкладке «База данных». На экране появиться окно с выбором директории сохранения и именем файла.

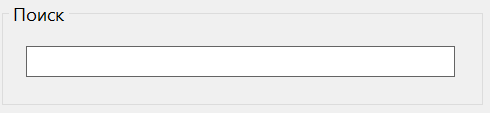


**Рисунок В.3 – Сохранение БД в файл**

Для поиска записи или фильтрации в каталоге необходимо выбрать критерий, далее ввести в поле поиск или фильтрации значения по котором вы хотите получить данные.

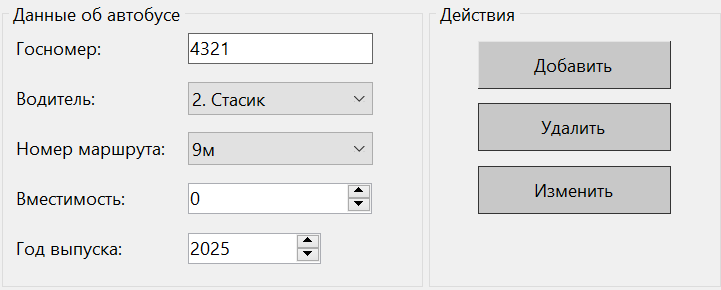


**Рисунок В.4 – Меню фильтрации**

****

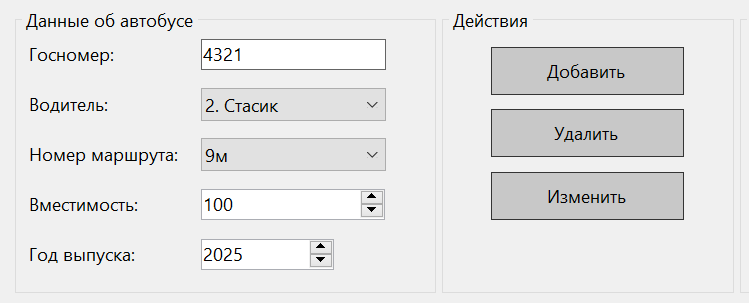
**Рисунок В.5 – Меню поиска**

Для добавления новой записи необходимо ввести нужные данные в соответствии с их типом данных и нажать кнопку «Добавить».

****

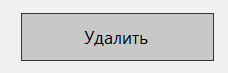
**Рисунок В.6 – Меню добавления**

Для редактирования данных нужно ввести госномер автобуса, затем нажать кнопку «Изменить», ввести новые данные для редактирования.

****

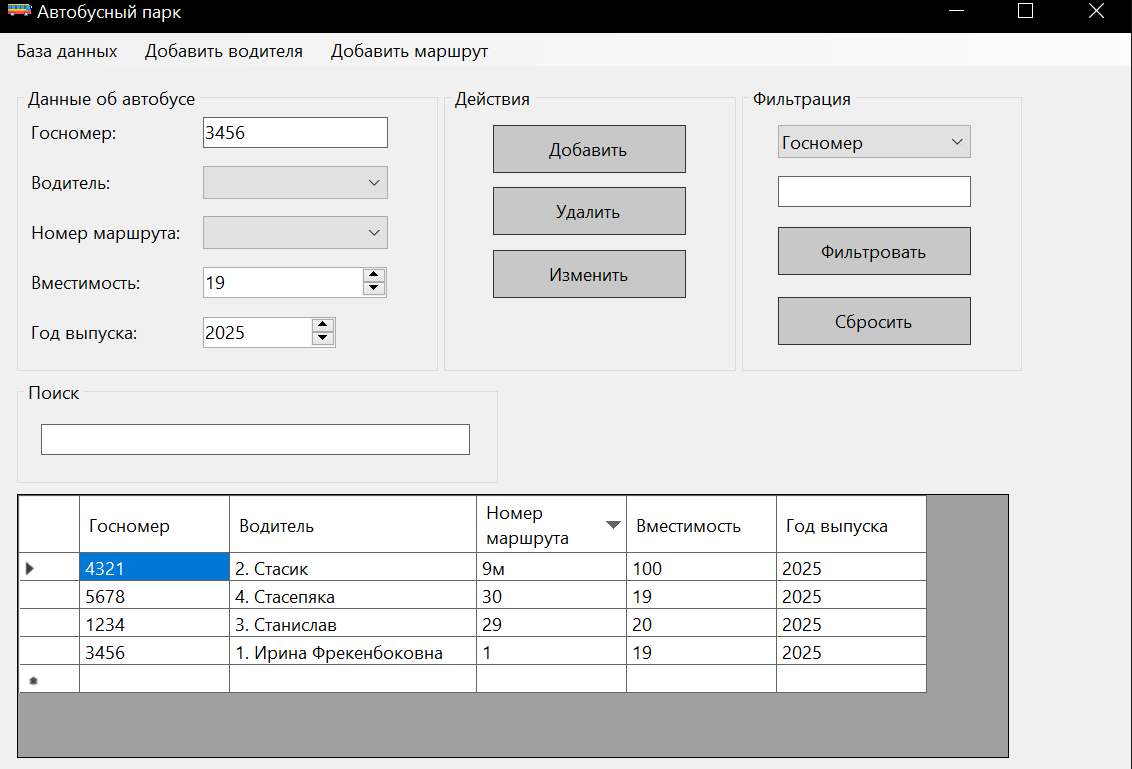
**Рисунок В.7 – Меню редактирования**

Для удаления записи необходимо ввести госномер автобуса и нажать кнопку удалить «Удалить».

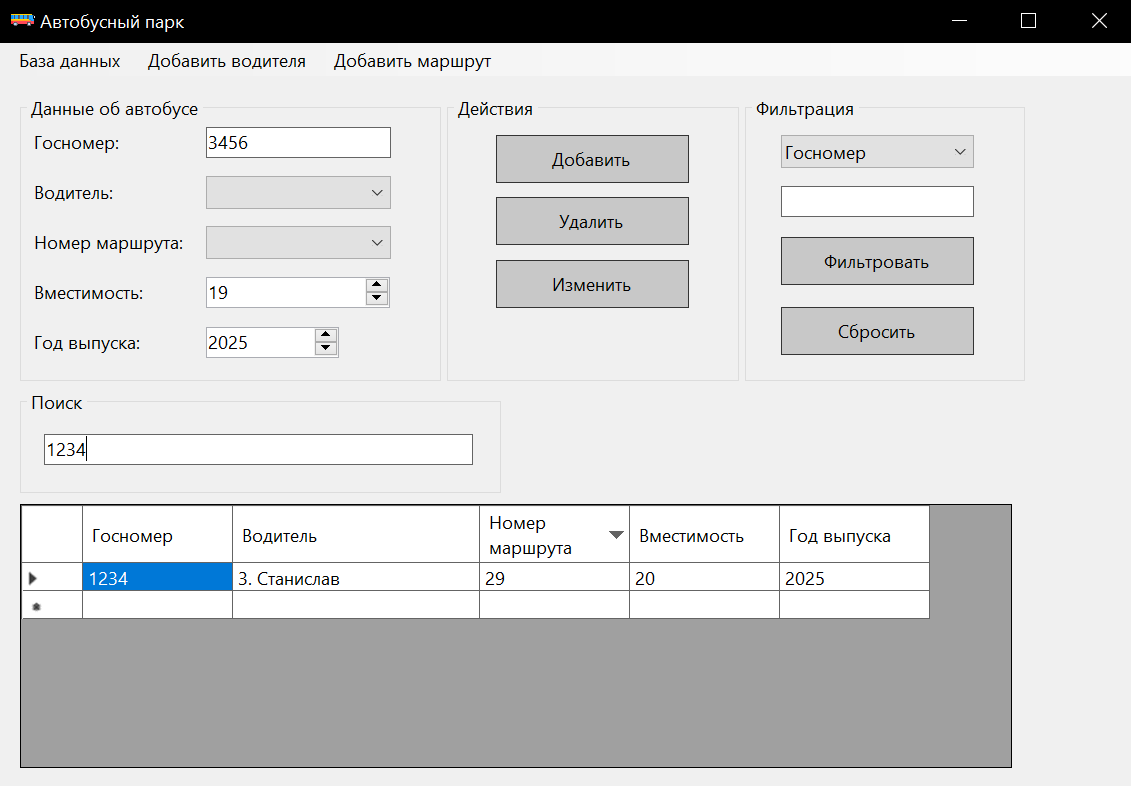


**Рисунок В.7 – Кнопка удаления элемента**

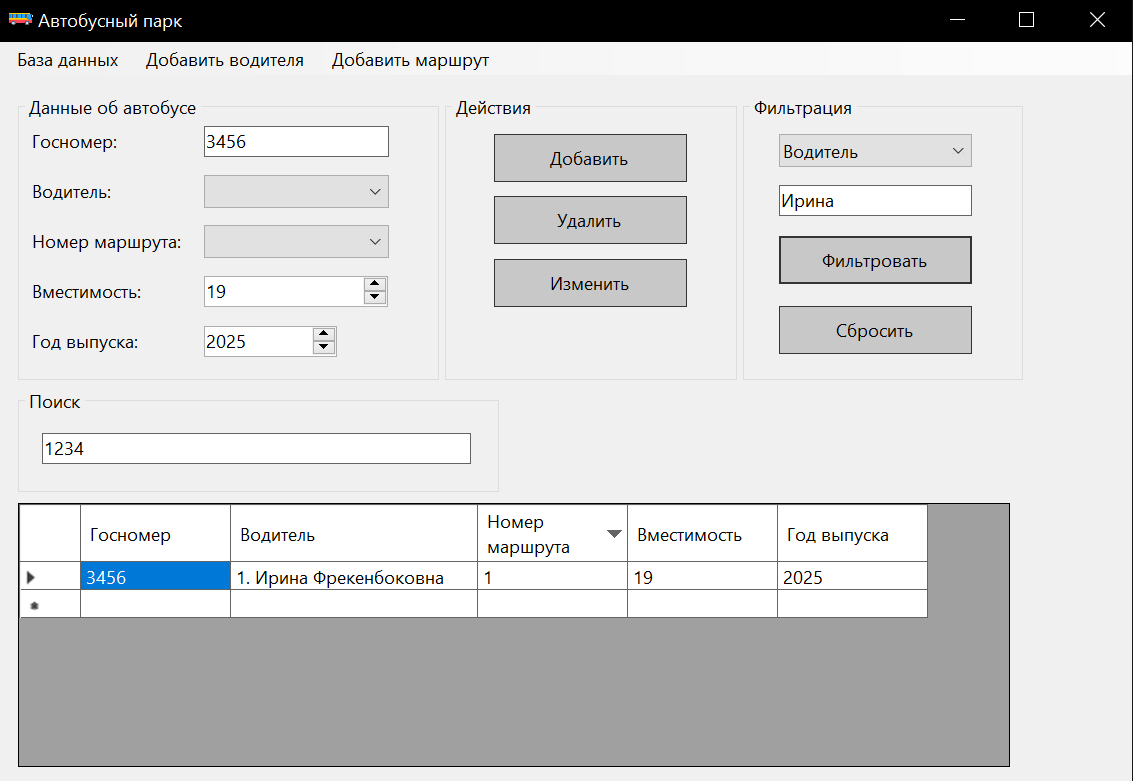
Для сортировки нужно нажать на заголовок таблицы и данные от сортируются по выбранному параметру, для изменения критерия сортировки (по убыванию или по возрастанию) кликните повторно на заголовок.

**  
Рисунок В.8 – Сортировка**

Для поиска нужно ввести нужное значение для поиска. Пользователю будут удалены неподходящие строки и останутся только нужные.

**  
Рисунок В.9 – Поиск**

Для фильтрации нужно выбрать критерий нахождения и ввести нужное значение для фильтрации. Пользователю будут удалены неподходящие строки и останутся только нужные.

**  
Рисунок В.9 – Поиск**