**Министерство образования Российской Федерации**

**Пензенский государственный университет**

Кафедра «Математическое обеспечение и применение ЭВМ»

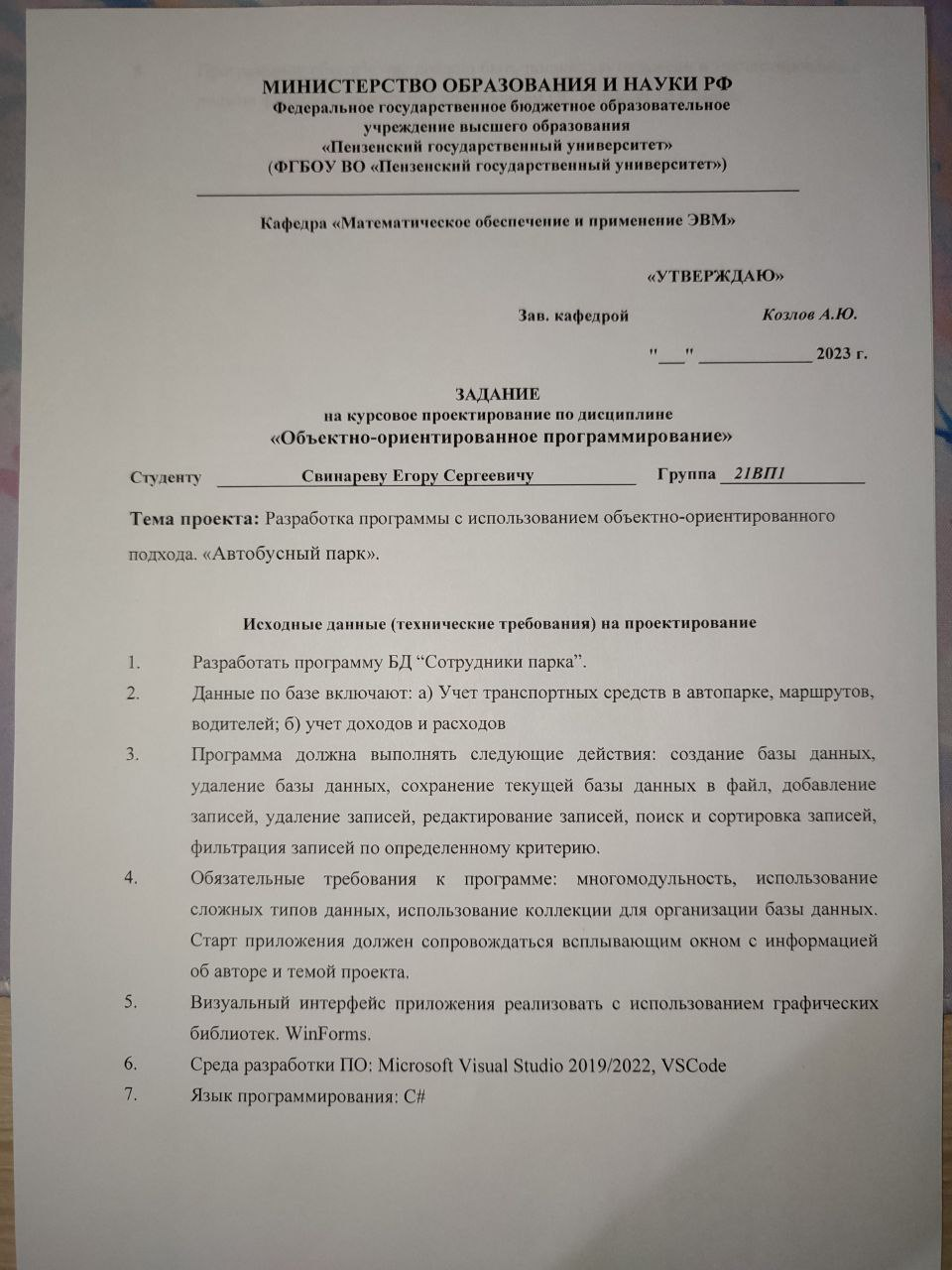
Пояснительная записка к курсовому проекту   
по дисциплине

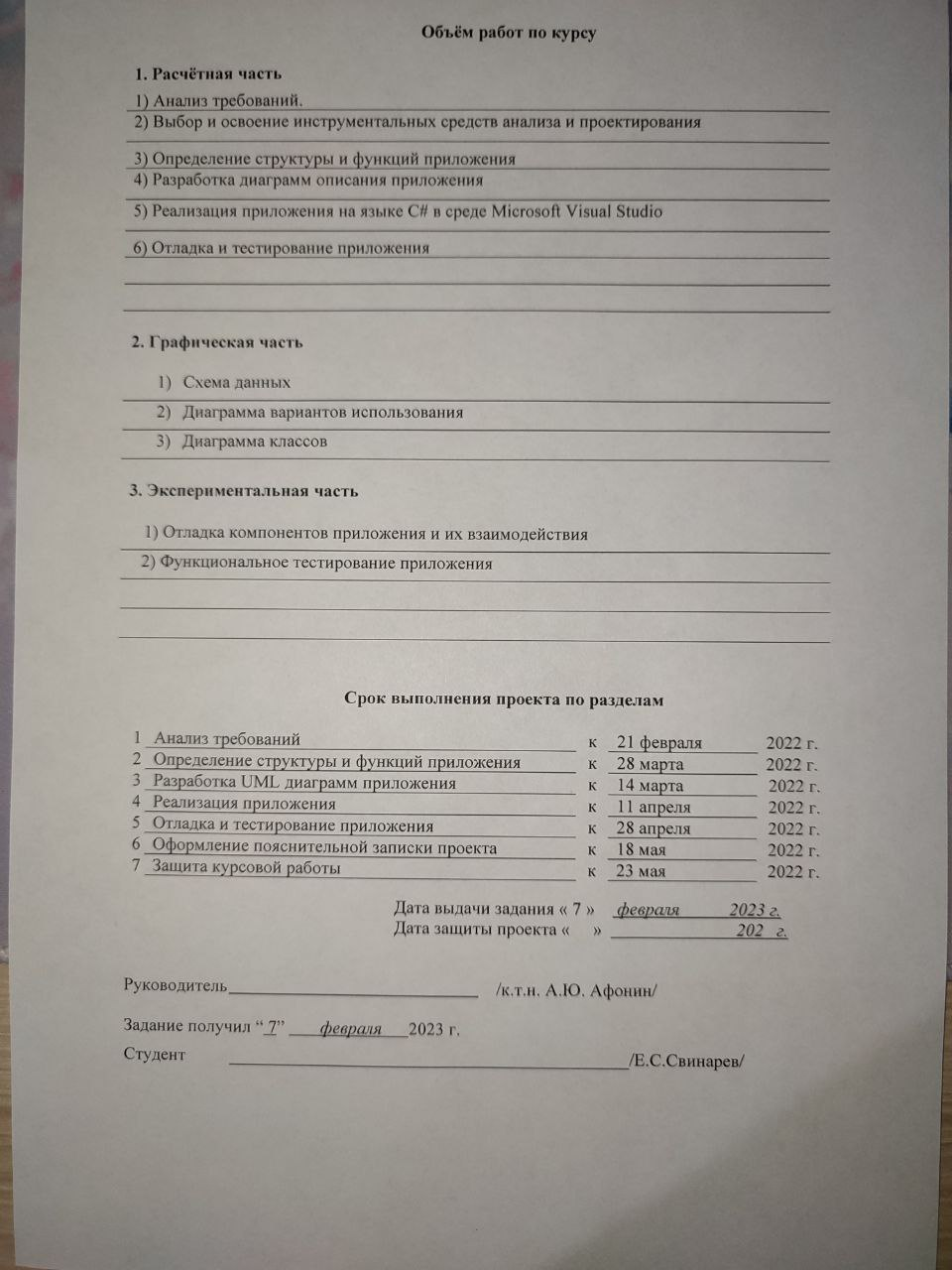
«Объектно-ориентированное программирование»

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

|  |  |
| --- | --- |
| Автор работы: | Свинарев Е.С. |
| Направление бакалавриата | 09.03.04 («Программная инженерия») |
| Обозначение курсовой работы | ПГУ 09.03.04 – 04КР211.25 ПЗ |
| Группа | 21ВП1 |
| Руководитель работы | Афонин А.Ю., к.т.н., доцент |
| Работа защищена «\_\_» \_\_\_\_ 2023 г. | Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_ |

Пенза 2023





Содержание

[**Введение** 5](#_Toc135699004)

[**1 Постановка задачи** 7](#_Toc135699005)

[**2 Выбор решения** 8](#_Toc135699006)

[2.1 Определение необходимых модулей программы 8](#_Toc135699007)

[2.2 Определение структуры файла базы данных 8](#_Toc135699008)

[**3 Описание разработки программы** 9](#_Toc135699009)

[**4 Отладка и тестирование** 13](#_Toc135699010)

[**5 Описание программы** 26](#_Toc135699011)

[5.1 Разработка приложения 26](#_Toc135699012)

[5.2 Разработка меню 27](#_Toc135699013)

[**6 Руководство пользователя** 32](#_Toc135699014)

[**Заключение** 33](#_Toc135699015)

[**Список используемых источников** 34](#_Toc135699016)

[**Приложение А - Руководство пользователя** 35](#_Toc135699017)

[**Приложение В - Исходные тексты программы** 43](#_Toc135699018)

**Введение**

В современном мире одной из наиболее важных проблем является эффективное хранение, поиск и обработка данных. Почти во всех отраслях деятельности человек сталкивается с огромным объемом информации, и информационные системы играют важную роль в структурировании и хранении схожей по тематике информации. Использование информационных систем помогает более эффективно организовывать деятельность организации и полностью или частично отказаться от бумажного документооборота, который, в отличие от электронных средств хранения данных, имеет ограниченный срок службы.

Таким образом, информационная система представляет собой рабочую систему, задача которой состоит в сборе, передаче, хранении, извлечении, обработке и отображении информации.

Цель данной курсовой работы заключается в разработке информационной системы, которая будет хранить данные о транспорте, водителях и маршрутах автобусного парка. Разработанная платформа поможет сотрудникам автобусного парка более эффективно систематизировать информацию об имуществе парка. Приложение позволит пользователям подключаться к единой базе данных, добавлять и удалять записи, осуществлять сортировку, фильтрацию и поиск записей по заданным критериям.

Для достижения поставленной цели нужно решить следующие задачи:

• изучить особенности работы с коллекциями и файлами на С#;

• разработать модель пользовательского интерфейса программного средства;

• разработать основной функционал приложения;

• провести тестирование разработанной программы;

• разработать руководство пользователя.

Разработка программы осуществлялась под управлением операционной системы Windows 11 в среде программирования Visual Studio Community 2022 на C#.

**1 Постановка задачи**

Необходимо разработать программу «Автобусный парк». Программа должна содержать три раздела, первый из которых будет контролировать информацию о водителях автобусного парка, второй будет контролировать информацию о транспорте автобусного парка, а третий будет контролировать информацию о доступных маршрутах автобусного парка.

Информационные поля раздела «Водители»:

* ID;
* Имя;
* Фамилия;
* Возраст;
* Опыт вождения;
* ID Транспорта;
* Транспорт;
* ID Маршрута;
* Маршрут.

Информационные поля раздела «Транспорт»:

* ID;
* Модель;
* Номерной знак;
* Количество мест;
* Дата технического обслуживания;
* Пробег;
* Маршрут;
* Водитель.

Информационные поля раздела «Маршруты»:

* ID;
* Название;
* Стартовая точка;
* Конечная точка;
* Длина маршрута;
* Начало движения;
* Конец движения.

Программа должна предоставить возможности: добавления нового водителя, транспорта и маршрута, изменения существующего водителя, транспорта и маршрута, удаления водителя, транспорта и маршрута, сортировки водителей, транспорта и маршрутов по различным критериям, фильтрации водителей, транспорта и маршрутов по определенным критериям, поиска информации в таблице по доступных информационным полям среди водителей, транспорта и маршрутов, удаления единичного или множественного количества водителей, транспорта и маршрутов.

Для реализации этих требований нужно разработать интуитивно понятный интерфейс пользователя.

Разработанное приложение должно содержать окно с информацией об авторе и теме проекта.

**2 Выбор решения**

**2.1 Определение необходимых модулей программы**

Для создания программы была использована операционная система Windows 10 и среда программирования Visual Studio Community 2022 на основе шаблона приложения Windows Forms (.NET 7.0) для языка программирования C#.

Для работы с файлом базы данных был установлен Entity Framework 7.0.[1]

**2.2 Определение структуры файла базы данных**

В случае использования SQLite, структура файла базы данных имеет следующий вид:

1. Файл базы данных имеет расширение .sqlite или .db
2. Файл базы данных состоит из нескольких таблиц, каждая из которых имеет свое имя.
3. Каждая таблица состоит из столбцов и строк. Столбцы определяют поля таблицы, а строки содержат данные.
4. Каждый столбец имеет свое имя и тип данных, например, INTEGER, TEXT, REAL, BLOB.
5. Каждая строка таблицы содержит данные для каждого столбца в этой таблице.
6. Каждая таблица может иметь связи с другими таблицами, что позволяет объединять данные из нескольких таблиц в один результат запроса.

С использованием EntityFramework для работы с базой данных в C# можно создать модели данных, которые отображают структуру таблиц в базе данных. Модели данных могут быть использованы для выполнения операций CRUD (Create, Read, Update, Delete) с данными в базе данных.

**3 Описание разработки программы**

Разработанное приложение должно выполнить следующие функции:

* Добавить позицию в базу;
* Удалить позицию из базы;
* Сортировать позиции;
* Фильтровать позиции;
* Произвести поиск позиций;
* Удалить все позиции из базы.

В процессе описания разработки программы была составлена диаграмма вариантов использования.

Диаграмма вариантов использования [3], разработанная с помощью средства UML моделирования – Visual Paradigm, приведена на рисунке 1.

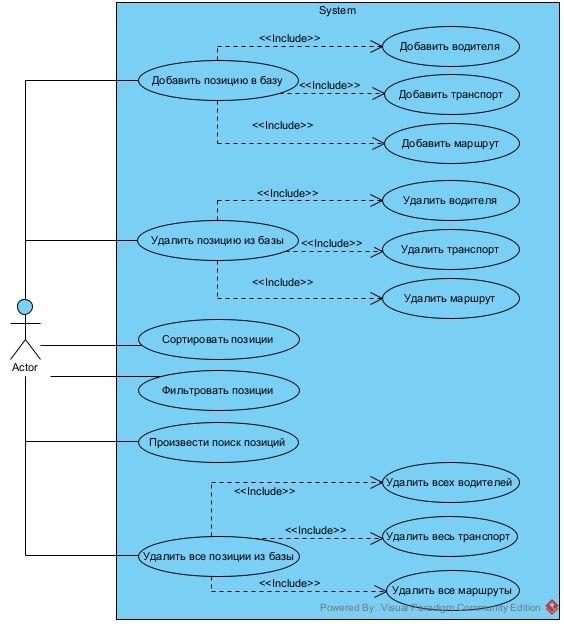


Рисунок 1 – Диаграмма вариантов использования

Описания некоторых спецификаций прецедентов представлены в таблицах 1-3.

Таблица 1 – Спецификация прецедента «Добавить запись в базу»

|  |
| --- |
| **Наименование: Добавить позицию в базу** |
| **ID: 1** |
| **Краткое описание:** пользователь заполняет информационные поля, программа добавляет запись с этой информацией в базу |
| **Действующие лица:** пользователь, программа |
| **Основной поток:**   1. Пользователь заполняет информационные поля. 2. Пользователь нажимает на кнопку «Добавить». 3. Программа проверяет корректность введённых данных. 4. Программа добавляет запись в базу. |
| **Постусловие:** Запись добавлена в базу |

Таблица 2 – Спецификация прецедента «Удалить запись из базы»

|  |
| --- |
| **Наименование: Удалить запись из базы** |
| **ID: 2** |
| **Краткое описание:** пользователь выбирает запись для удаления, программа удаляет выбранную запись |
| **Действующие лица:** программа, пользователь |
| **Основной поток:**   1. Пользователь выбирает строку из таблицы с записями. 2. Пользователь нажимает на кнопку «Удалить». 3. Программа удаляет запись. |
| **Постусловие:** Запись удалена из базы |

Таблица 3 – Спецификация прецедента «Фильтровать позиции»

|  |
| --- |
| **Наименование: Фильтровать позиции** |
| **ID: 3** |
| **Краткое описание:** пользователь вводит текст для фильтрации в текстовое поле, программа производит фильтрацию |
| **Действующие лица:** программа, пользователь |
| **Основной поток:**  1. Пользователь вводит информацию в текстовое поле.  2. Программа выводит в таблицу данные, соответствующие выбору пользователя. |
| **Постусловие:** данные, подходящие под фильтр, выведены в таблицу |

Для описания структуры программного обеспечения была разработана диаграмма классов, представленная на рисунке 2.

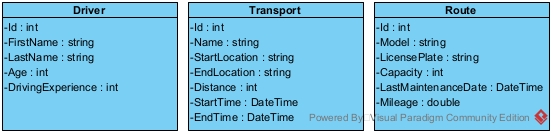


Рисунок 2 – Диаграмма классов

Класс «Driver» представляет собой модель данных для хранения информации о водителе.

Класс содержит следующие свойства:

* "Id": уникальный идентификатор водителя, тип данных - int.
* "FirstName": имя водителя, тип данных - string.
* "LastName": фамилия водителя, тип данных - string.
* "Age": возраст водителя, тип данных - int.
* "DrivingExperience": опыт вождения водителя в годах, тип данных - int.

Класс «Route» представляет собой модель данных для хранения информации о маршруте.

Класс содержит следующие свойства:

* "Id": уникальный идентификатор маршрута, тип данных - int.
* "Name": название маршрута, тип данных - string.
* "StartLocation": начальное местоположение маршрута, тип данных - string.
* "EndLocation": конечное местоположение маршрута, тип данных - string.
* "Distance": длина маршрута в километрах, тип данных - int.
* "StartTime": время начала движения по маршруту, тип данных - DateTime.
* "EndTime": время окончания движения по маршруту, тип данных - DateTime.

Класс «Transport» представляет собой модель данных для хранения информации о транспортном средстве.

Класс содержит следующие свойства:

* "Id": уникальный идентификатор транспортного средства, тип данных - int.
* "Model": модель транспортного средства, тип данных - string.
* "LicensePlate": государственный регистрационный номер транспортного средства, тип данных - string.
* "Capacity": вместимость транспортного средства, тип данных - int.
* "LastMaintenanceDate": дата последнего технического обслуживания транспортного средства, тип данных - DateTime.
* "Mileage": текущий пробег транспортного средства, тип данных - double.

**4 Отладка и тестирование**

В курсовой работе было выполнено функциональное тестирование разработанного программного обеспечения. План тестирования приведен в таблице 4.

Таблица 4 – План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| 1 | Добавить позицию в таблицу. | Позиция занесена в таблицу | Позиция отобразилась в таблице (Рисунок 3) |
| 2 | Удалить водителя из таблицы. | Водитель удален из таблицы, отображено сообщение об удалении | Водитель из таблицы удален, выведено сообщение «Удалены водители с ID: 1!» (Рисунок 4) |
| 3 | Добавить транспорт существующему водителю. | Транспорт добавлен существующему водителю | Транспорт отобразился в таблице (Рисунок 5) |
| 4 | Ввести некорректные данные при создании позиции. | Сообщение о некорректных данных. | Выведено сообщение, о том, что данные в полях для информации некорректны (Рисунок 6) |
| 5 | Отсортировать водителей в порядке возрастания по полю «ID». | Записи отсортированы | Записи в таблице упорядочены в порядке возрастания по полю «ID» (Рисунок 7) |

Таблица 4 – План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| 6 | Отфильтровать водителей по максимальному возрасту. | Отображены водители с возрастом, который меньше максимального | Водители, соответствующие критерию фильтрации, выведены в таблицу (Рисунок 8) |
| 7 | Сбросить фильтрацию. | В таблице появились все водители | В таблице появились все существующие водители (Рисунок 9) |
| 8 | Удалить всех водителей. | Удалены все водители, выведено сообщение об удалении | Таблица очищена, выведено сообщение «Удалены водители с ID: …» (Рисунок 10) |
| 9 | Нажать кнопку «Очистить» на форме добавления водителя. | Очищены поля формы добавления водителя | Применена очистка к полям формы (Рисунок 11) |
| 10 | Нажать кнопку «Сгенерировать» на форме добавления водителя. | Заполнены поля формы добавления водителя | Применена генерация данных к полям формы (Рисунок 12) |
| 11 | Применить фильтр отображения колонок таблицы «Водители». | Применен фильтр для отображения колонок таблицы «Водители» | Применен фильтр для отображения колонок таблицы «Водители» (Рисунок 13) |
| 12 | Нажать кнопку «Изменить водителя». | Открыта форма изменения выбранного водителя | Открыта форма изменения выбранного водителя (Рисунок 14) |

Таблица 4 – План тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| 13 | Произвести поиск на таблице «Водители». | Отображены данные, соответствующие искомому значению. | Отображены данные, соответствующие искомому значению (Рисунок 15) |

На рисунках 3–15, приведены скриншоты, отражающие результаты работы программы в процессе тестирования.

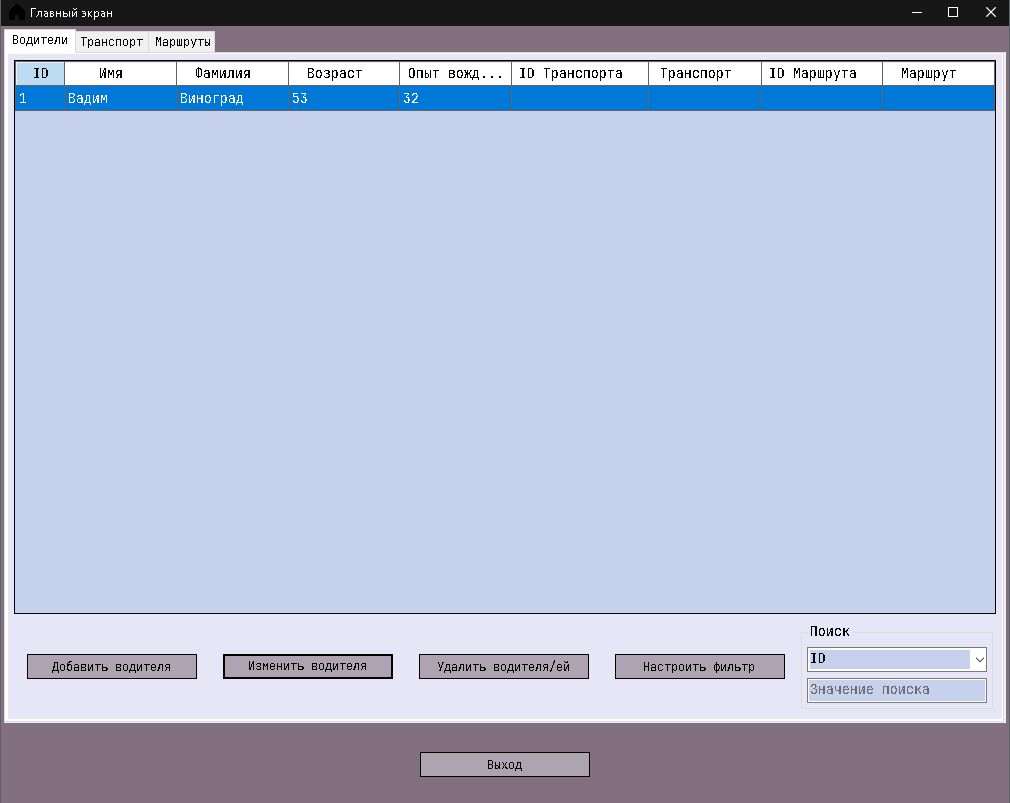
****

Рисунок 3 – Добавление позиции в таблицу

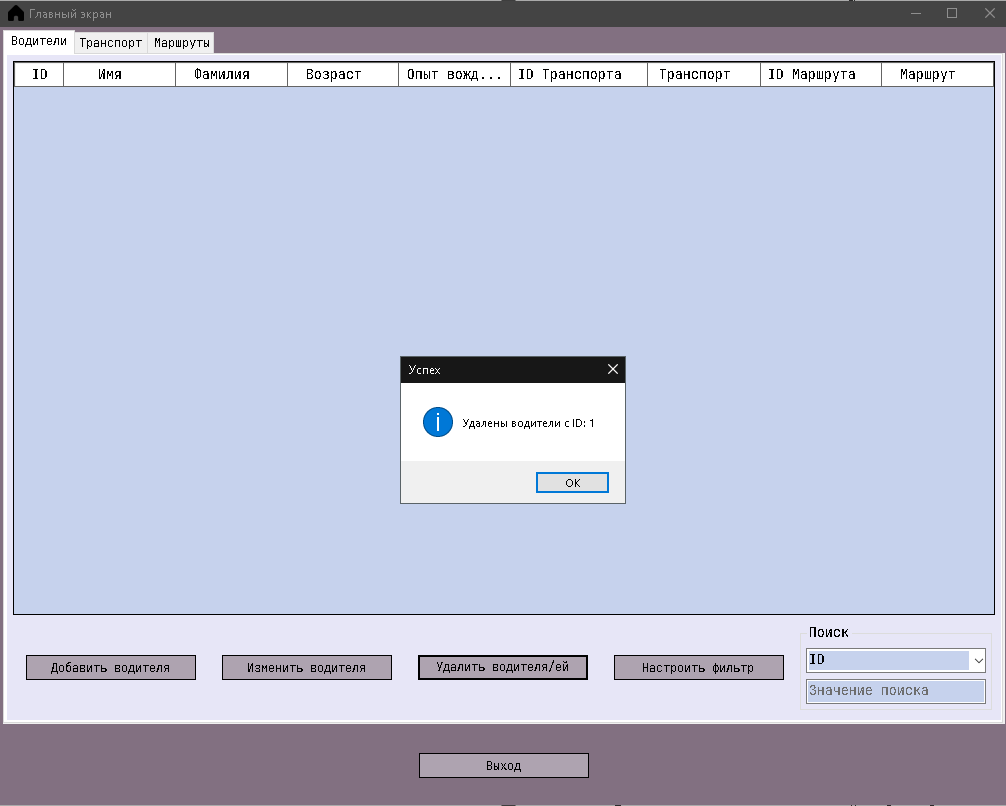
****

Рисунок 4 – Удаление водителя из таблицы

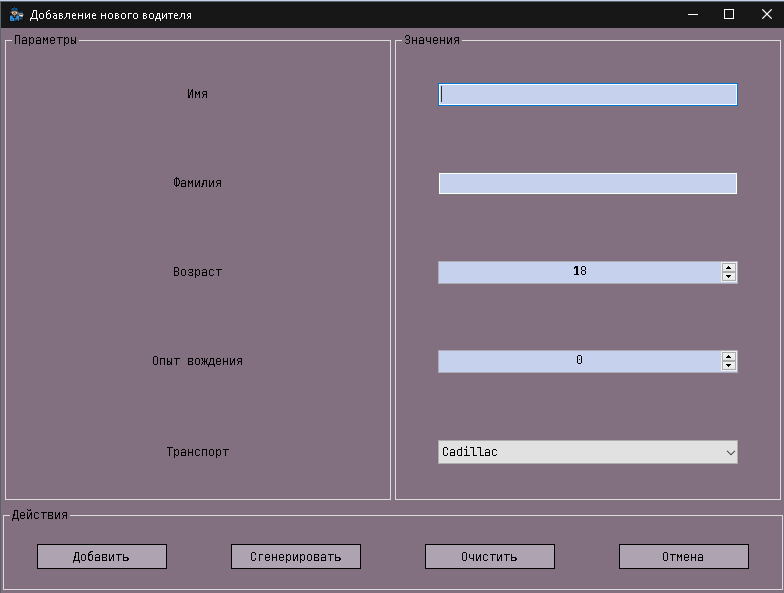


Рисунок 5 – Добавление транспорта существующему водителю

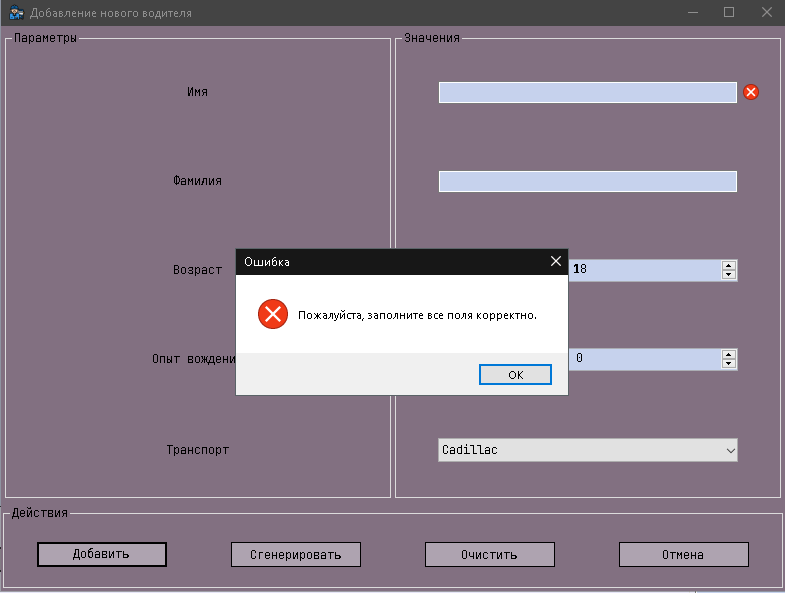


Рисунок 6 – Ввод некорректных данных при добавлении позиции

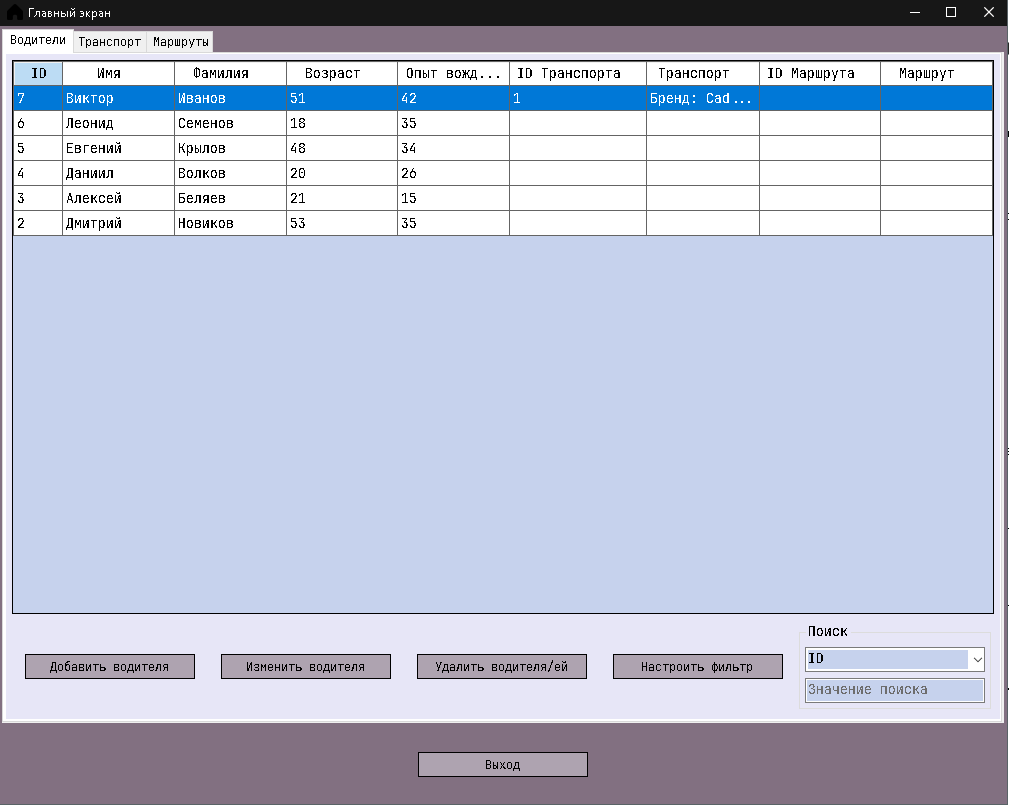


Рисунок 7 – Сортировка водителей по полю «ID»

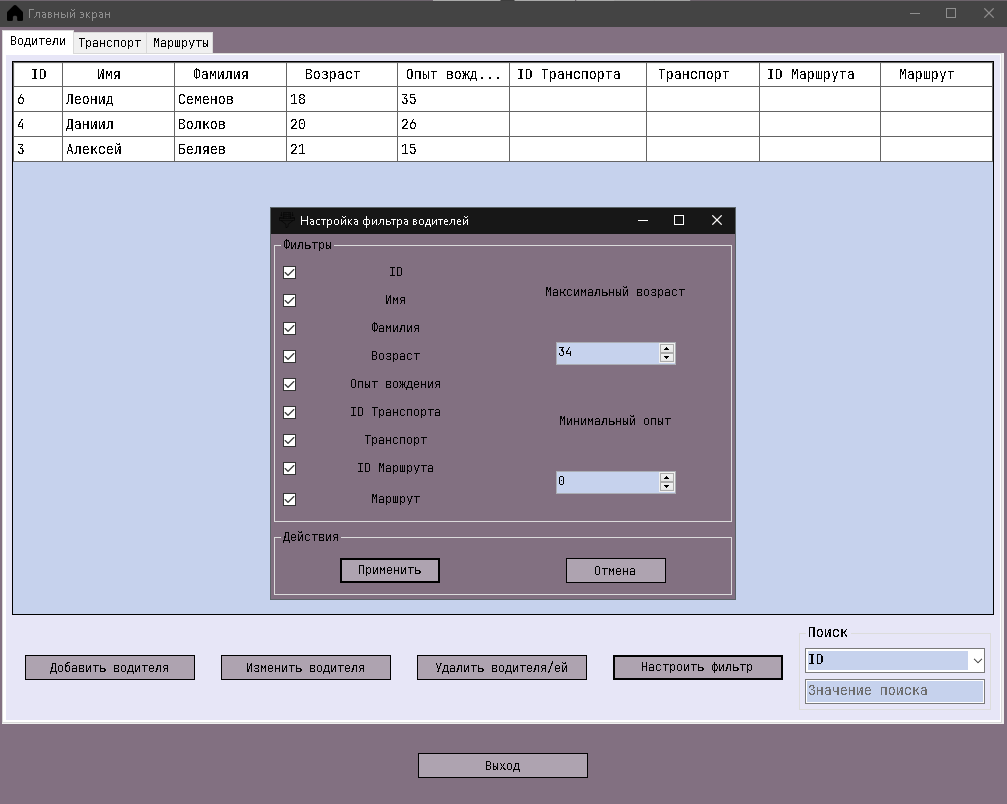


Рисунок 8 – Фильтрация водителей по максимальному возрасту

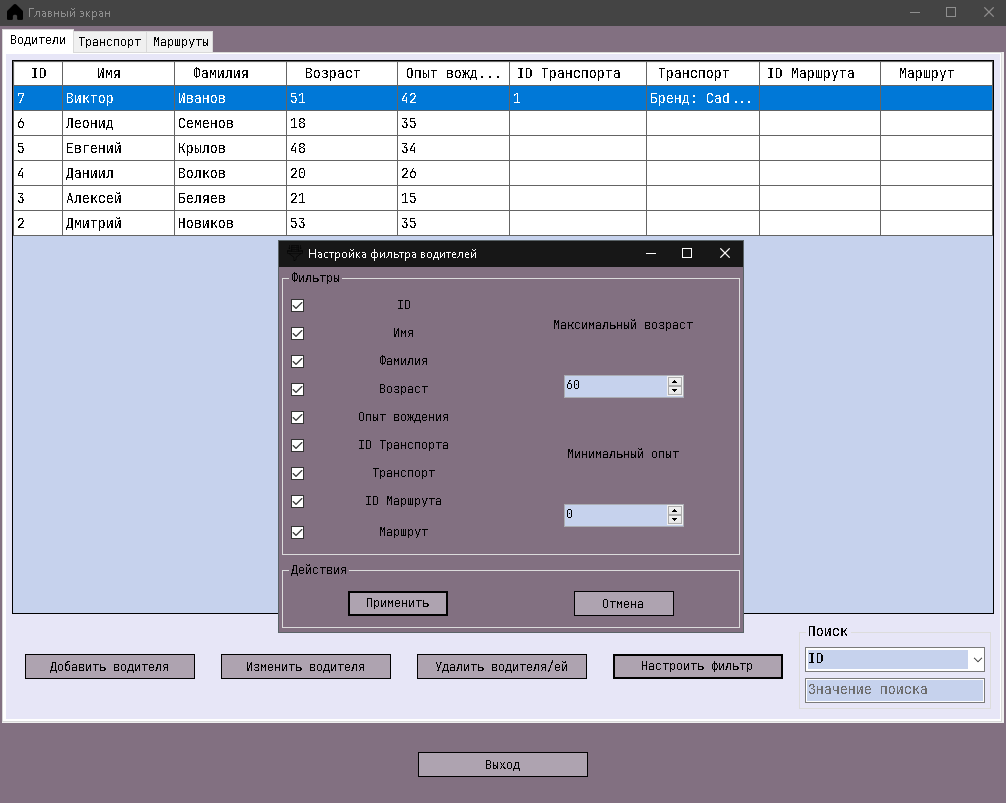


Рисунок 9 – Сброс фильтрации

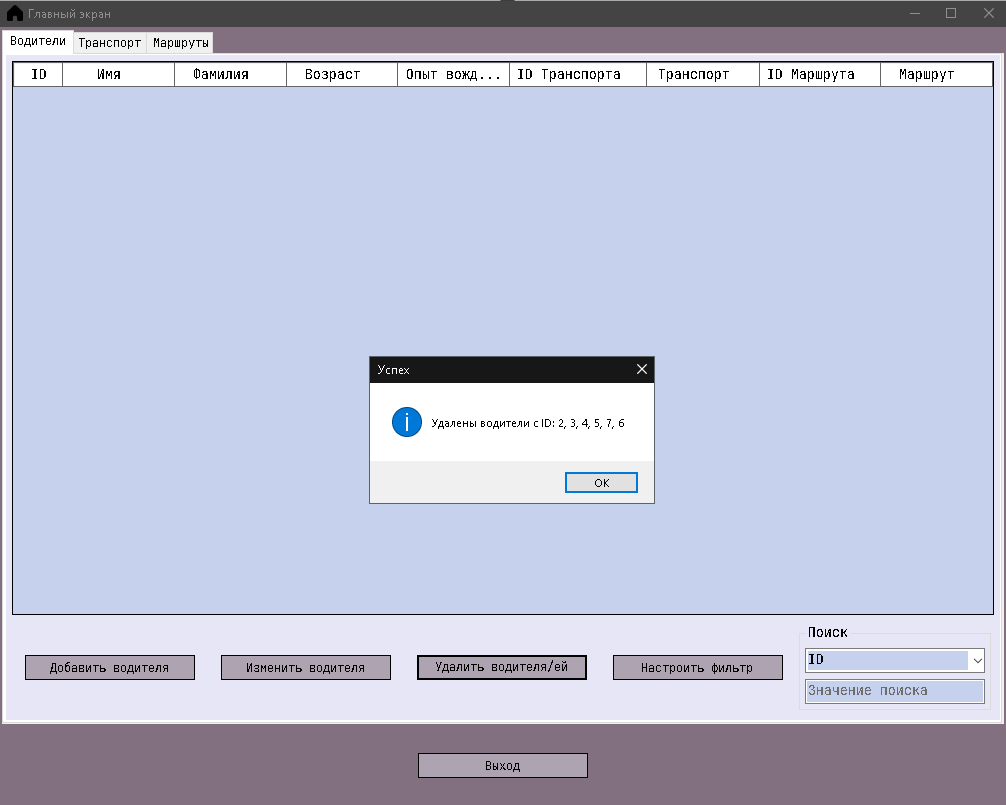


Рисунок 10 – Удаление всех водителей

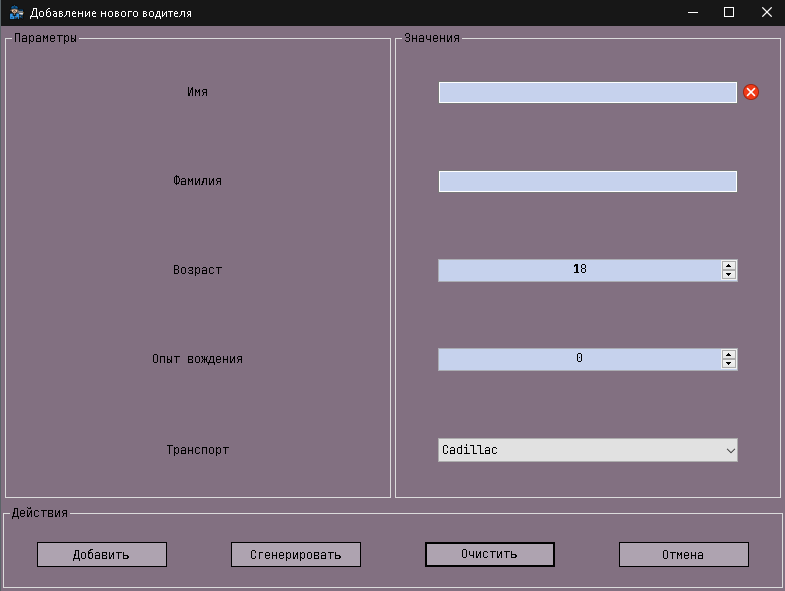


Рисунок 11 – Применений очистки на форме добавления водителя

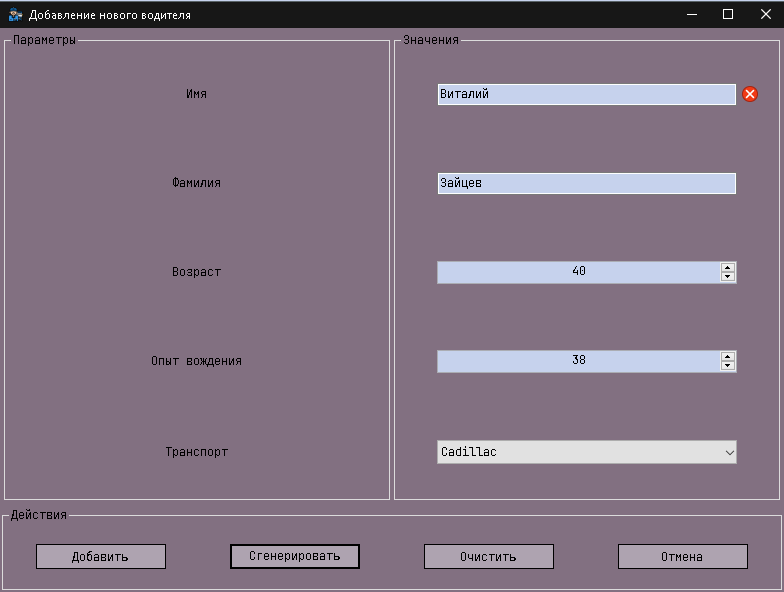


Рисунок 12 – Применение генерации на форме добавления водителя

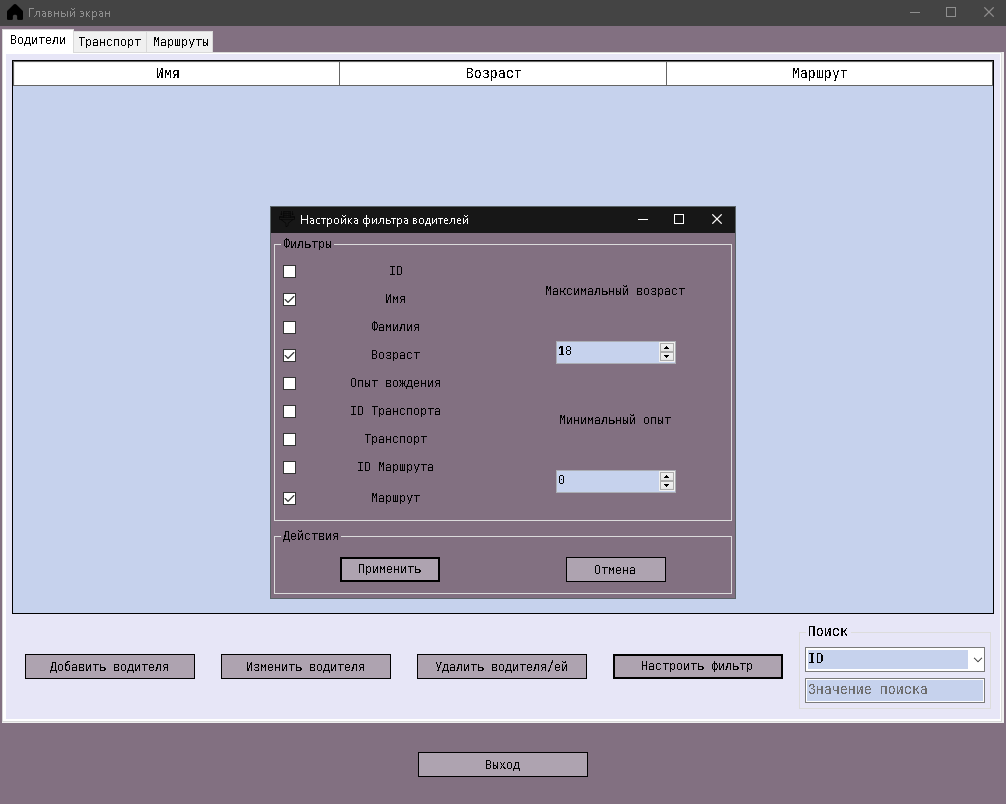


Рисунок 13 – Применение фильтрации отображаемых колонок таблицы «Водители»

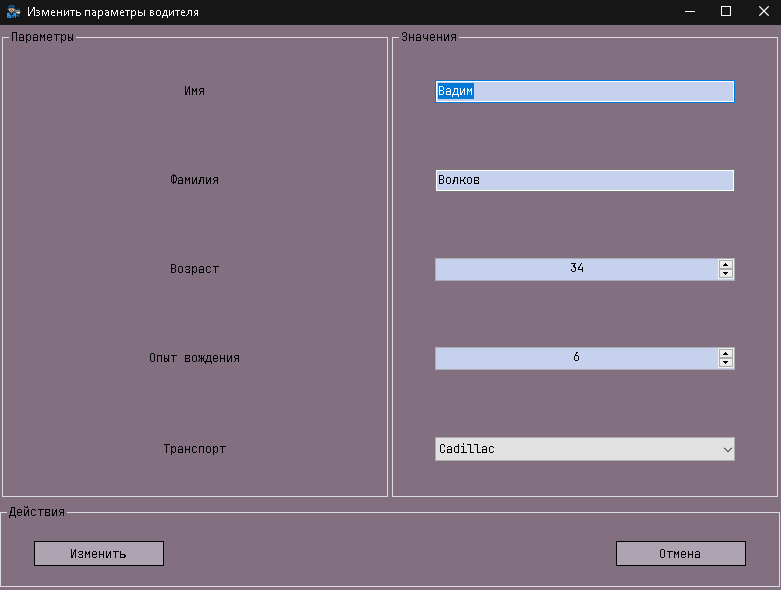


Рисунок 14 – Отображение формы изменения водителей

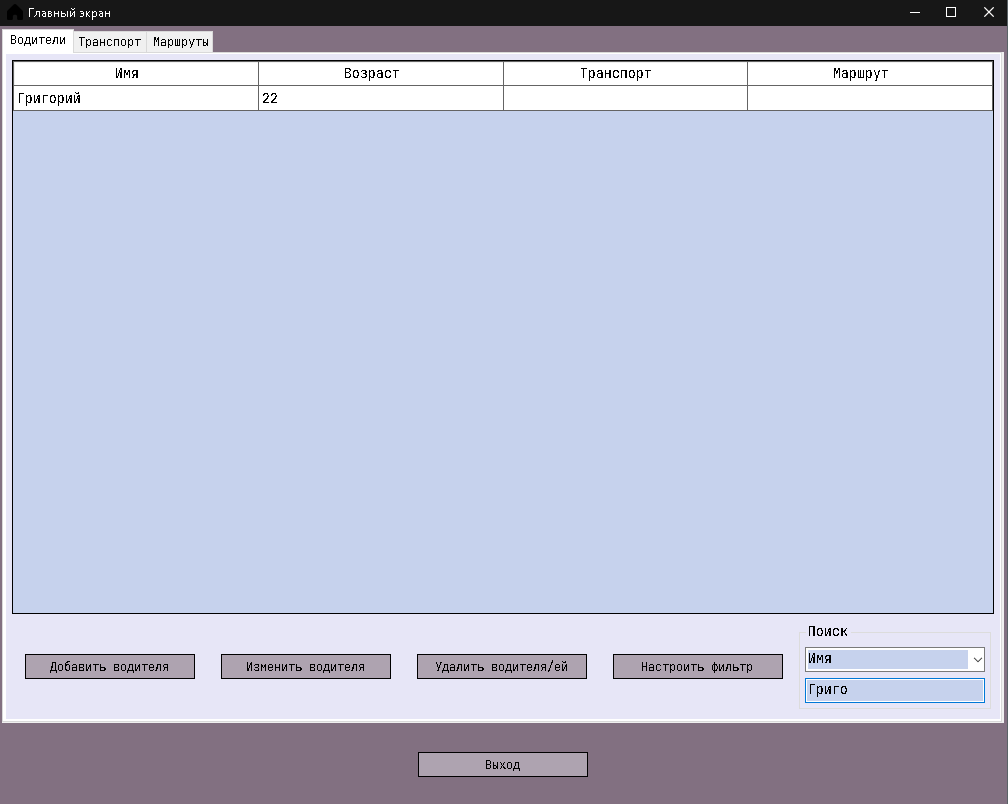


Рисунок 15 – Поиск информации на таблице «Водители»

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

**5 Описание программы**

**5.1 Разработка приложения**

В процессе выполнения курсовой работы была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты и связи между ними. Диаграмма компонентов приведена на рисунке 16.

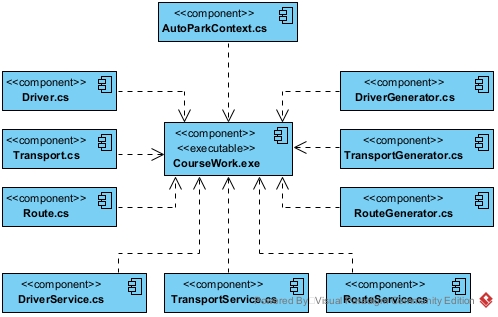


Рисунок 16 – Диаграмма компонентов

Описание компонентов приведено в таблице 5.

Таблица 5 – Описание компонентов

|  |  |
| --- | --- |
| Компонент | Назначение |
| CourseWork.exe | Исполняемый файл приложения |
| AutoParkContext.cs | Класс для работы с контекстом базы данных |
| Driver.cs | Класс, представляющий сущность «Водитель» |
| Transport.cs | Класс, представляющий сущность «Транспорт» |
| Route.cs | Класс, представляющий сущность «Маршрут» |
| DriverService.cs | Класс, представляющий сервис для работы с сущностью «Водитель» |

Таблица 5 – Описание компонентов

|  |  |
| --- | --- |
| TransportService.cs | Класс, представляющий сервис для работы с сущностью «Транспорт» |
| RouteService.cs | Класс, представляющий сервис для работы с сущностью «Маршрут» |
| DriverGenerator.cs | Класс, представляющий генератор данных для сущности «Водитель» |
| TransportGenerator.cs | Класс, представляющий генератор данных для сущности «Транспорт» |
| RouteGenerator.cs | Класс, представляющий генератор данных для сущности «Маршрут» |

**5.2 Разработка меню**

Приложение состоит из 11 различных форм.

На рисунке 17 представлен интерфейс формы информации об авторе

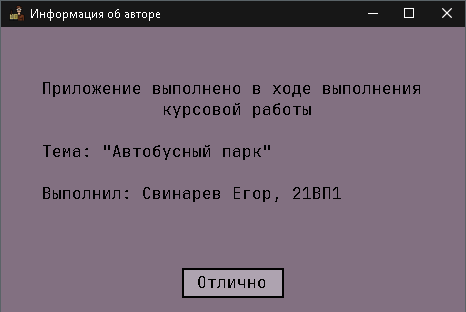
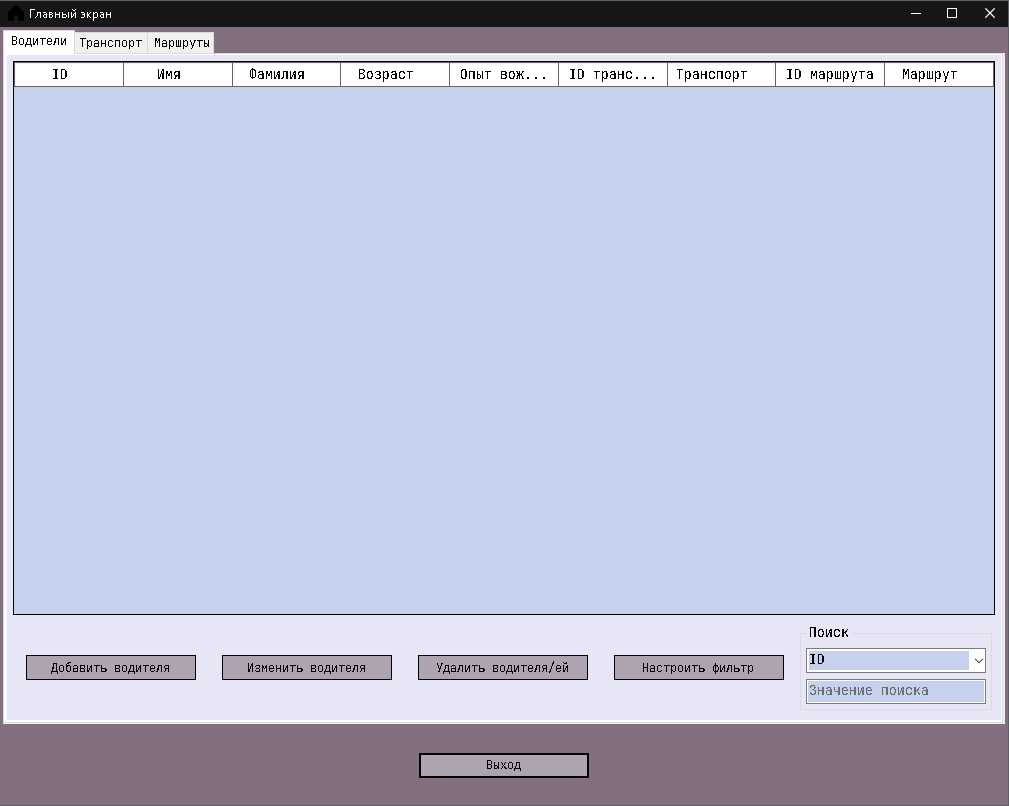


Рисунок 17 – Интерфейс формы информации об авторе

На рисунке 18 представлен интерфейс главного экрана приложения.

Данная форма содержит следующие компоненты:

1. доступные страницы таблицы;
2. текущая таблица;
3. кнопка открытия формы добавления позиции;
4. кнопка открытия формы изменения выбранной позиции;
5. кнопка удаления выбранных позиций;
6. кнопка открытия формы настройки фильтра для текущей таблицы;
7. выпадающий список выбора колонки для выполнения поиска;
8. значение поиска информации;
9. кнопка закрытия формы.



9

8

7

6

5

4

3

2

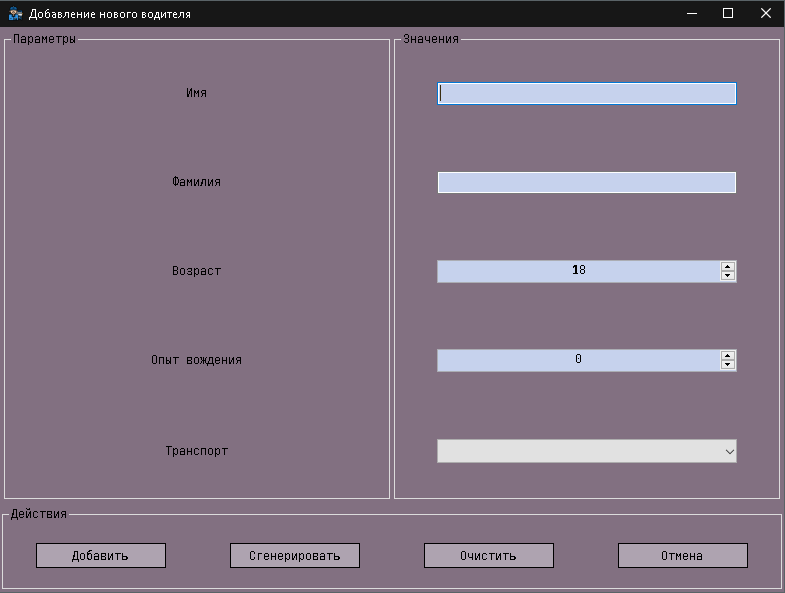
1

Рисунок 18 – Интерфейс формы главного экрана приложения

На рисунке 19 представлен интерфейс формы добавления позиции.

Данная форма содержит следующие компоненты:

1. описание поля ввода информации;
2. поле для ввода информацией;
3. кнопка добавления позиции с указанной информацией;
4. кнопка генерации информации для полей ввода информации;
5. кнопка очистки полей для ввода информации;
6. кнопка закрытия формы.

****

6

5

4

3

2

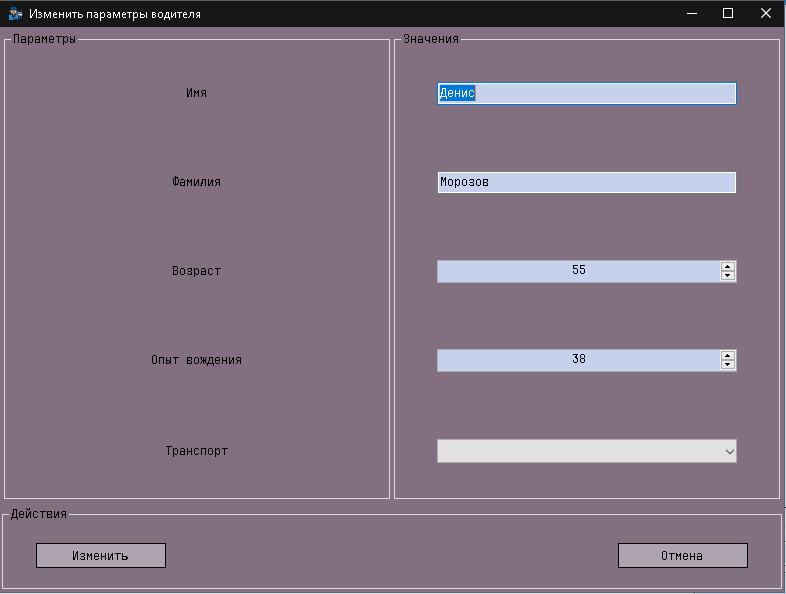
1

Рисунок 19 – Интерфейс формы добавления позиции

На рисунке 20 представлен интерфейс формы изменения информации о позиции.

Данная форма содержит следующие компоненты:

1. описание поля ввода информации;
2. поле для ввода информации;
3. кнопка применения изменения позиции;
4. кнопка закрытия формы.



4

3

2

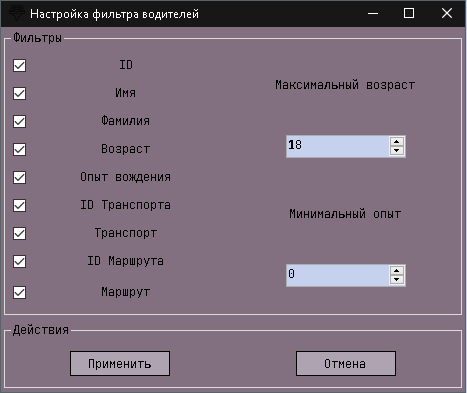
1

Рисунок 20 – Интерфейс формы изменения информации о позиции

На рисунке 21 представлен интерфейс формы задания фильтра.

Данная форма содержит следующие компоненты:

1. выбор отображения колонки таблицы
2. описание поля ввода информации;
3. поле для ввода информации;
4. кнопка применения фильтра;
5. кнопка закрытия формы.

****

5

4

3

2

1

Рисунок 21 – Интерфейс формы задания фильтра

**6 Руководство пользователя**

Руководство пользователя является неотъемлемой частью любой программы, предназначенной для использования людьми. Оно содержит информацию, необходимую для того, чтобы пользователи могли эффективно работать с программой и использовать ее возможности. Основная цель руководства пользователя заключается в том, чтобы обеспечить пользователя необходимой информацией для самостоятельной работы с программой.

В руководстве пользователя приводятся подробные инструкции и рекомендации, которые помогают пользователям быстро разобраться в функциональности программы и решить возникающие проблемы. Руководство пользователя содержит описание шагов, которые необходимо выполнить для достижения конкретной цели.

Текст руководства пользователя приведен в приложении A.

**Заключение**

В ходе выполнения курсового проекта была создана программа под названием "CourseWork", которая позволяет пользователям работать с базой данных, содержащей записи, связанные с тематикой "Автобусный парк". Приложение обеспечивает основные инструменты для выполнения следующих операций: добавление, удаление, фильтрация, поиск и сортировка записей. Разработанное программное обеспечение соответствует всем требованиям, установленным для выполнения проекта. Результаты тестирования подтверждают правильную работу приложения. В приложении В приведен код программы.

В рамках выполнения курсового проекта было разработано руководство пользователя, которое позволит пользователям быстро освоить функциональность созданного приложения.

Завершающим этапом разработки приложения стало создание инсталлятора.

На исходный код проекта был создан github репозиторий по следующей ссылке: https://github.com/3982580943570984/oop-course-work

**Список используемых источников**

1. "Entity Framework: основы работы с ORM-фреймворком" (статья на сайте "Хабр"). URL: https://habr.com/ru/company/microsoft/blog/301380/
2. Болотов, А. И. Основы разработки программного обеспечения. СПб.: Питер, 2019. - 528 с.
3. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624с
4. Проектирование баз данных на примере SQLite. [Электронный ресурс]. URL: https://habr.com/ru/post/464585/
5. "C# Учебник: Основы языка C#. Работа с базами данных" (статья на сайте "metanit.com"). [Электронный ресурс]. URL: https://metanit.com/sharp/tutorial/8.1.php

**Приложение А - Руководство пользователя**

Программа CourseWork.exe предназначена для хранения информации о водителях, транспорте и маршрутах. Программа имеет интуитивно понятный интерфейс, оснащена вкладками для выбора желаемой отображаемой таблицы и поддерживает операции добавления, изменения, удаления, фильтрации и поиска информации.

При запуске программы происходит вывод формы, содержащей информацию об авторе и теме проекта.

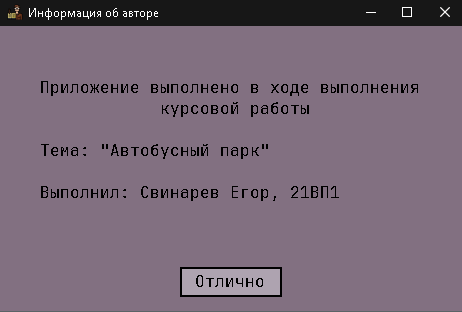


Рисунок А.1 – Информационное окно

Для добавления нового водителя, транспорта или маршрута в таблицу необходимо нажать кнопку «Добавить <водителя|транспорт|маршрут>», после чего откроется форма для добавления позиции. После ввода данных в поля формы необходимо нажать «Добавить».

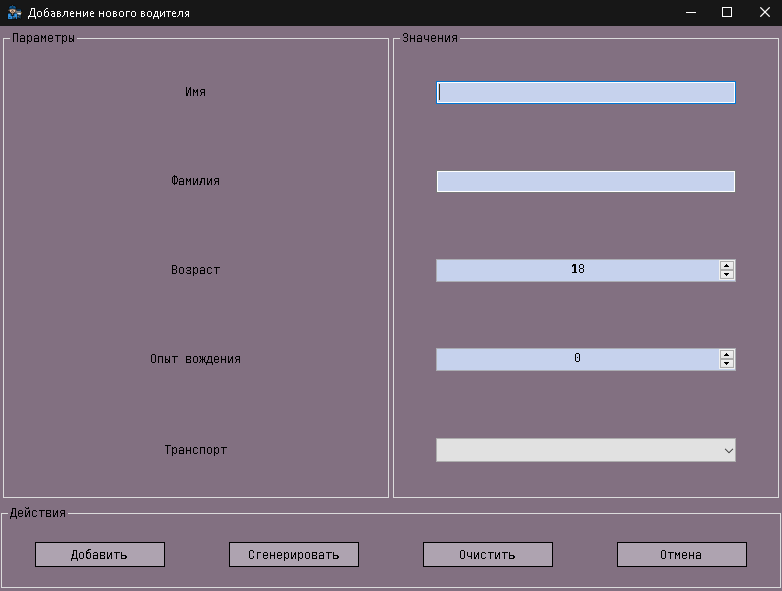


Рисунок А.2 – Форма добавления водителя

При попытке добавить водителя, транспорт или маршрут с частично заполненными полями или же некорректными данными, на экран будут выведены надписи с советами, а неправильно заполненные поля будут подсвечены. Все неправильные поля нужно и справить, иначе позиция не будет добавлена.

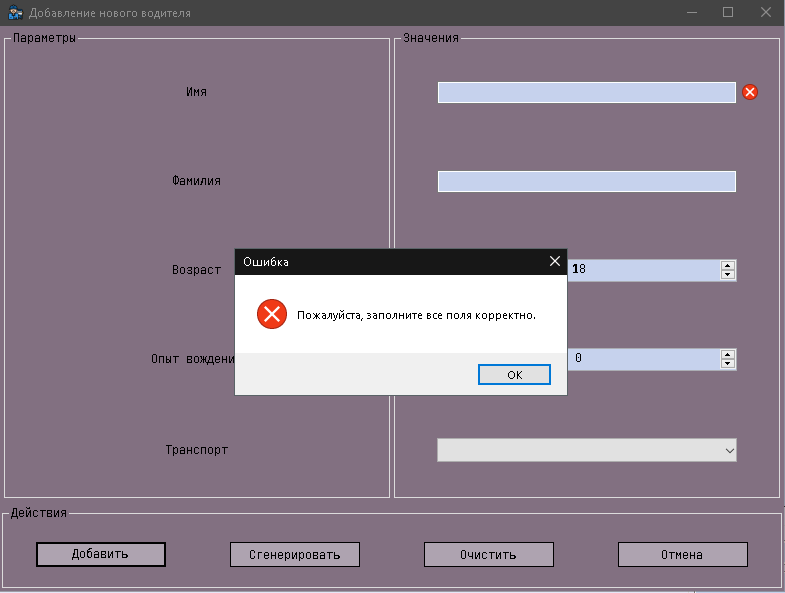


Рисунок А.3 – Форма добавления водителя при наличии неверных полей

Если же при нажатии на кнопку «Добавит» появилось информационное сообщение об успехе, то добавление позиции прошло успешно и запись отобразится в таблице.

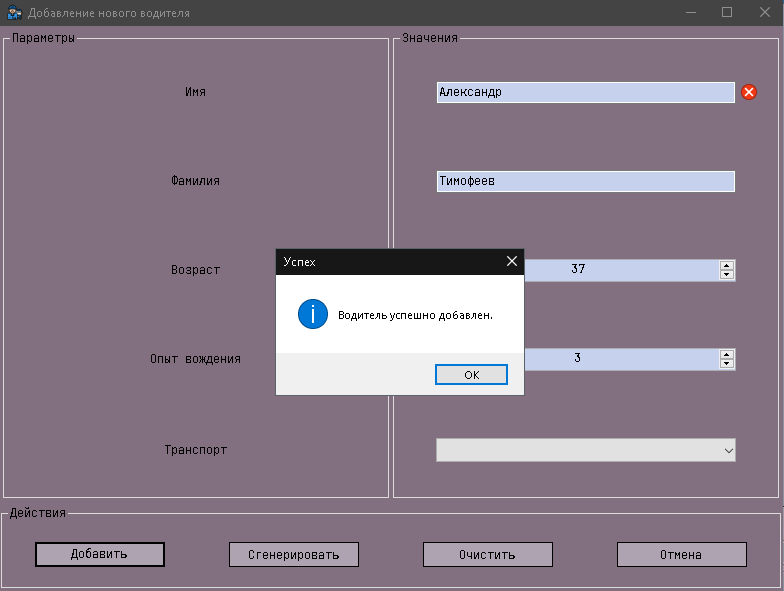


Рисунок А.4 – Форма добавления водителя при успешной валидации

Для удаления позиции нужно выбрать строку в таблице, которую необходимо удалить и нажать на кнопку «Удалит».

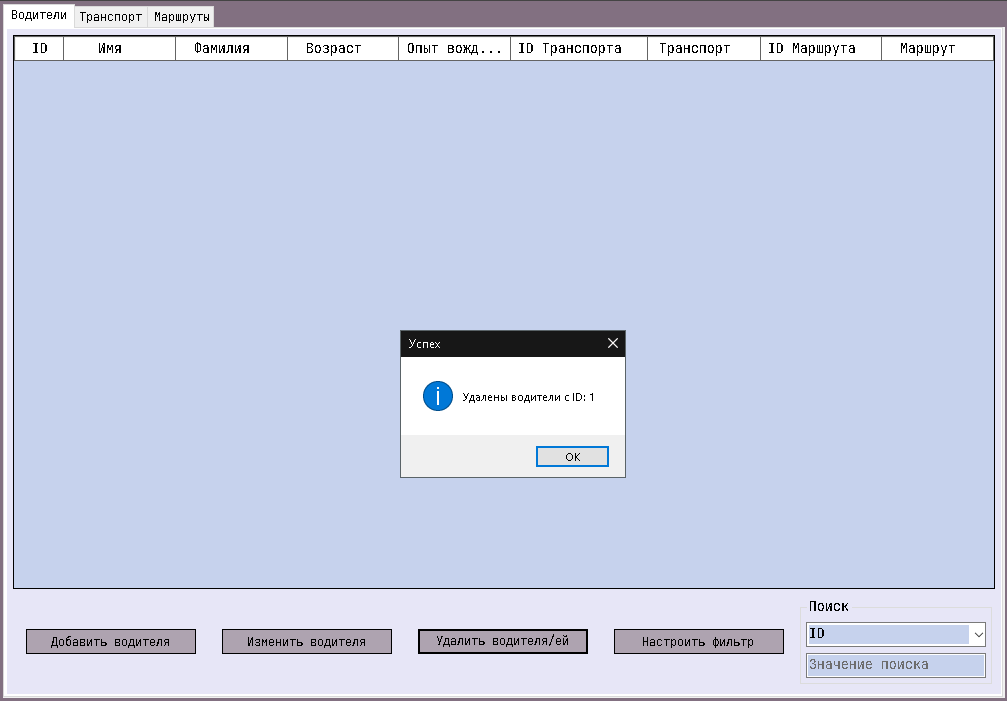


Рисунок А.5 – Удаление водителя

Для фильтрации водителей по максимальному возрасту или минимальному опыту необходимо ввести соответствующее значение в текстовое поле. После нажатия на кнопку «Применить» в таблице отобразятся позиции, удовлетворяющие критерию.

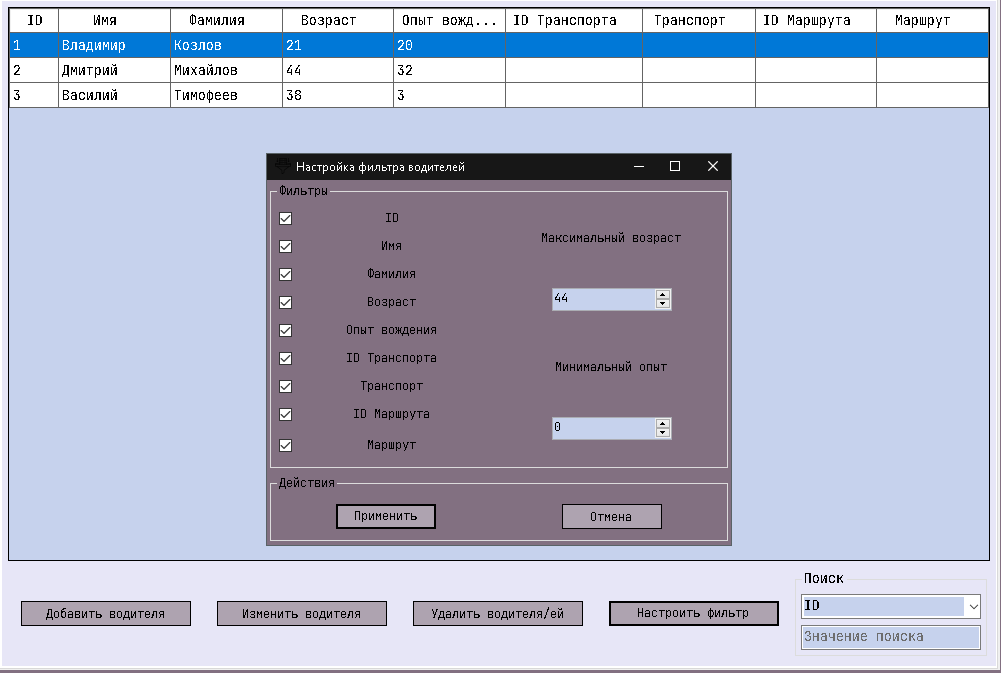


Рисунок А.6 – Применение фильтрации по максимальному возрасту

Чтобы показать все существующие позиции необходимо вернуть значения фильтра в их стандартное значение.

Для сортировки позиций необходимо нажать на заголовок той колонки, по которой необходимо отсортировать записи.

При повторном нажатии на заголовки позиции будут отсортированы в обратном порядке.

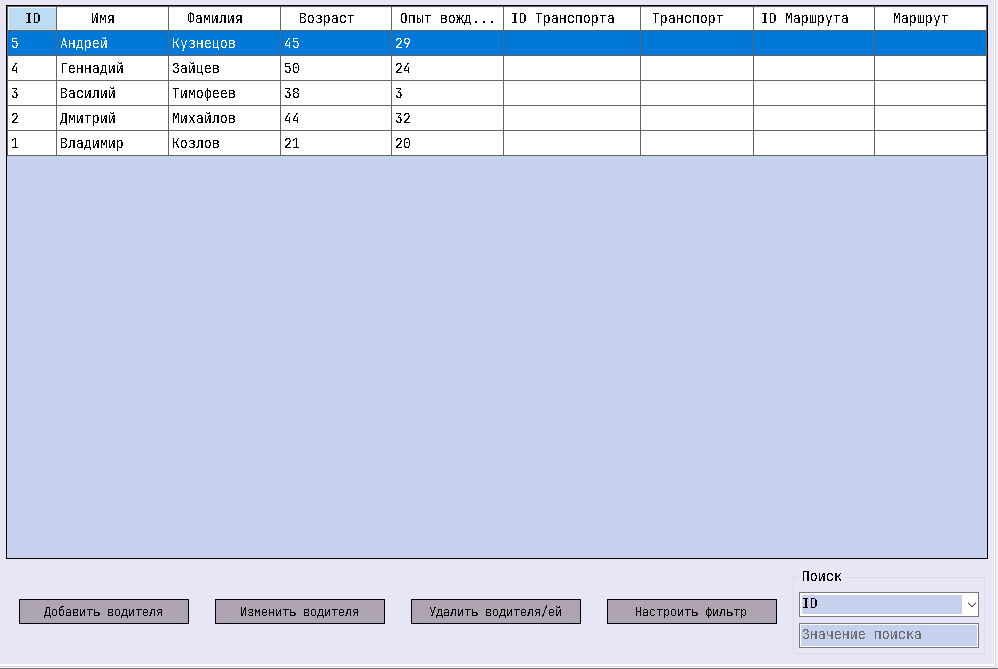


Рисунок А.7 – Первое нажатие на колонку «ID»

Для произведения поиска в таблице необходимо выбрать интересующую колонку и ввести значение для поиска. После нажатия на клавиатуре кнопки «Enter» в таблице останутся позиции, удовлетворяющие критерию.



Рисунок А.8 – Применение поиска по «ID»

Чтобы отменить поиск, достаточно очистить поле, хранящее значение для поиска и нажать на клавиатуре кнопку «Enter»

Для удаления всех существующих данных необходимо выделить всех водителей и нажать кнопку «Удалить». В случае успешного удаления таблица станет пустой.

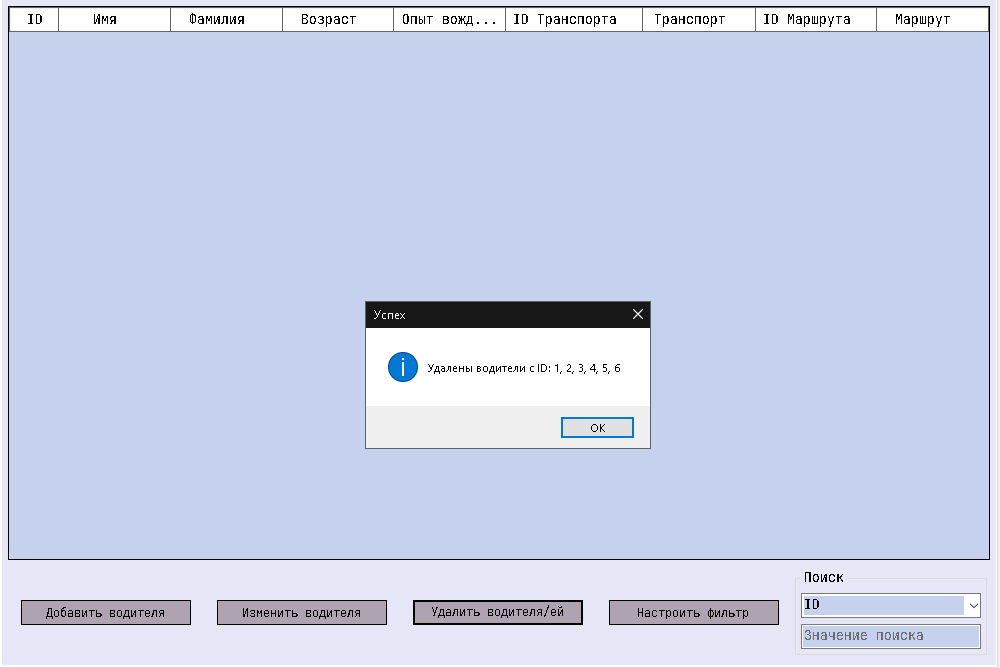


Рисунок А.9 – Вид таблицы после удаления всех водителей

**Приложение В - Исходные тексты программы**

Файл AuthorInfoForm.cs

namespace CourseWork.Forms.ForMain;

public partial class AuthorInfoForm : Form

{

public AuthorInfoForm() => InitializeComponent();

private void ButtonExit\_Click(object sender, EventArgs e) => Close();

}

Файл MainForm.cs

using CourseWork.Contexts;

using CourseWork.Entities;

using CourseWork.Forms.ForDrivers;

using CourseWork.Forms.ForMain;

using CourseWork.Forms.ForRoutes;

using CourseWork.Forms.ForTransports;

using CourseWork.Services;

using Microsoft.EntityFrameworkCore;

namespace CourseWork;

public partial class MainForm : Form

{

/// <summary>

/// Контекст базы данных автопарка

/// </summary>

public static readonly AutoParkContext autoParkContext = new();

/// <summary>

/// Конструктор класса MainForm

/// </summary>

public MainForm() => InitializeComponent();

/// <summary>

/// Метод OnLoad

/// </summary>

/// <param name="e">Аргументы события</param>

protected override void OnLoad(EventArgs e)

{

base.OnLoad(e);

autoParkContext.Database.EnsureDeleted();

autoParkContext.Database.EnsureCreated();

autoParkContext.Drivers.Load();

DriverBindingSource.DataSource = autoParkContext.Drivers.Local.ToBindingList();

UpdateDriverSearchTypes();

autoParkContext.Transports.Load();

TransportBindingSource.DataSource = autoParkContext.Transports.Local.ToBindingList();

UpdateTransportSearchTypes();

autoParkContext.Routes.Load();

RouteBindingSource.DataSource = autoParkContext.Routes.Local.ToBindingList();

UpdateRouteSearchTypes();

AuthorInfoForm authorInfoForm = new();

authorInfoForm.ShowDialog();

}

private void UpdateDriverSearchTypes()

{

ComboBoxDriversSearchTypes.Items.Clear();

foreach (DataGridViewColumn column in DataGridViewDrivers.Columns)

if (column.Visible)

ComboBoxDriversSearchTypes.Items.Add(column.HeaderText);

ComboBoxDriversSearchTypes.SelectedIndex = 0;

}

private void UpdateTransportSearchTypes()

{

ComboBoxTransportSearchTypes.Items.Clear();

foreach (DataGridViewColumn column in DataGridViewTransportes.Columns)

if (column.Visible)

ComboBoxTransportSearchTypes.Items.Add(column.HeaderText);

ComboBoxTransportSearchTypes.SelectedIndex = 0;

}

private void UpdateRouteSearchTypes()

{

ComboBoxRoutesSearchTypes.Items.Clear();

foreach (DataGridViewColumn column in DataGridViewRoutes.Columns)

if (column.Visible)

ComboBoxRoutesSearchTypes.Items.Add(column.HeaderText);

ComboBoxRoutesSearchTypes.SelectedIndex = 0;

}

/// <summary>

/// Обработчик нажатия на кнопку добавления водителя

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

AddDriverForm addDriverForm = new();

addDriverForm.ShowDialog();

}

/// <summary>

/// Обработчик нажатия на кнопку изменения информации о водителе

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

if (DriverBindingSource.Current != null)

{

Driver currentDriver = DriverBindingSource.Current as Driver;

UpdateDriverForm updateDriverForm = new(currentDriver);

updateDriverForm.ShowDialog();

DataGridViewDrivers.Refresh();

}

else

{

MessageBox.Show("Выберите водителя для редактирования.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик нажатия на кнопку удаления выбранных водителей

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

private async void ButtonDeleteSelectedDrivers\_Click(object sender, EventArgs e)

{

if (DataGridViewDrivers.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите водителей для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Создаем сервис для удаления водителей

DriverService driverService = new(autoParkContext);

string deletedDriversIds = "";

// Проходимся по каждой выбранной строке и удаляем соответствующего водителя

foreach (DataGridViewRow selectedRow in DataGridViewDrivers.SelectedRows)

{

int driverId = (int)selectedRow.Cells[0].Value;

try

{

await driverService.DeleteDriverAsync(driverId);

// Если удаление прошло успешно, добавляем ID удаленного водителя в строку deletedDriversIds

if (deletedDriversIds == "")

deletedDriversIds = driverId.ToString();

else

deletedDriversIds += ", " + driverId.ToString();

}

catch (Exception ex)

{

// Если произошла ошибка при удалении водителя, выводимсообщение с ошибкой

string errorMessage = $"Не удалось удалить водителя с ID {driverId}: {ex.Message}";

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

}

}

// Создаем сообщение об успешном выполнении операции и показываем его, используя список удаленных ID водителей

string successMessage = $"Удалены водители с ID: {deletedDriversIds}";

MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик нажатия на кнопку настройки фильтрации водителей

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

FilterDriverForm filterDriverForm = new(DataGridViewDrivers);

filterDriverForm.ShowDialog();

/\*UpdateDriverSearchTypes();\*/

}

/// <summary>

/// Обработчик нажатия на кнопку добавления транспорта

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

AddTransportForm addTransportForm = new();

addTransportForm.ShowDialog();

}

/// <summary>

/// Обработчик нажатия на кнопку обновления информации о транспорте

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

if (TransportBindingSource.Current != null)

{

Transport currentTransport = TransportBindingSource.Current as Transport;

UpdateTransportForm updateTransportForm = new(currentTransport);

updateTransportForm.ShowDialog();

DataGridViewTransportes.Refresh();

}

else

{

MessageBox.Show("Выберите транспорт для редактирования.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик нажатия на кнопку удаления выбранных транспортов

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

private async void ButtonDeleteSelectedTransportes\_Click(object sender, EventArgs e)

{

if (DataGridViewTransportes.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите транспорт для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Создаем сервис для удаления транспорта

TransportService transportService = new(autoParkContext);

string deletedTransportsIds = "";

// Проходимся по каждой выбранной строке и удаляем соответствующий транспорт

foreach (DataGridViewRow selectedRow in DataGridViewTransportes.SelectedRows)

{

int transportId = (int)selectedRow.Cells[0].Value;

try

{

await transportService.DeleteTransportAsync(transportId);

// Если удаление прошло успешно, добавляем ID удаленного транспорта в строку deletedDriversIds

if (deletedTransportsIds == "")

deletedTransportsIds = transportId.ToString();

else

deletedTransportsIds += ", " + transportId.ToString();

}

catch (Exception ex)

{

// Если произошла ошибка при удалении транспорта, выводим сообщение с ошибкой

string errorMessage = $"Не удалось удалить транспорт с ID {transportId}: {ex.Message}";

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

}

}

// Создаем сообщение об успешном выполнении операции и показываем его, используя список удаленных ID транспортов

string successMessage = $"Удалены транспорты с ID: {deletedTransportsIds}";

MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик нажатия на кнопку фильтрации транспортов

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

FilterTransportForm filterTransportForm = new(DataGridViewTransportes);

filterTransportForm.ShowDialog();

/\*UpdateTransportSearchTypes();\*/

}

/// <summary>

/// Обработчик нажатия на кнопку добавления маршрута

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

AddRouteForm addRouteForm = new();

addRouteForm.ShowDialog();

}

/// <summary>

/// Обработчик нажатия на кнопку обновления информации о маршруте

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

if (RouteBindingSource.Current != null)

{

Route currentRoute = RouteBindingSource.Current as Route;

UpdateRouteForm updateRouteForm = new(currentRoute);

updateRouteForm.ShowDialog();

DataGridViewRoutes.Refresh();

}

else

{

MessageBox.Show("Выберите маршрут для редактирования.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

}

/// <summary>

/// Обработчик нажатия на кнопку удаления выбранных маршрутов

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

private async void ButtonDeleteRoutes\_Click(object sender, EventArgs e)

{

if (DataGridViewRoutes.SelectedRows.Count == 0)

{

MessageBox.Show("Выберите маршруты для удаления.", "Ошибка", MessageBoxButtons.OK, MessageBoxIcon.Error);

return;

}

// Создаем сервис для удаления маршрута

RouteService routeService = new(autoParkContext);

string deletedRotuesIds = "";

// Проходимся по каждой выбранной строке и удаляем соответствующий маршрут

foreach (DataGridViewRow selectedRow in DataGridViewRoutes.SelectedRows)

{

int routeId = (int)selectedRow.Cells[0].Value;

try

{

await routeService.DeleteRouteAsync(routeId);

// Если удаление прошло успешно, добавляем ID удаленного маршрута в строку deletedDriversIds

if (deletedRotuesIds == "")

deletedRotuesIds = routeId.ToString();

else

deletedRotuesIds += ", " + routeId.ToString();

}

catch (Exception ex)

{

// Если произошла ошибка при удалении маршрута, выводим сообщение с ошибкой

string errorMessage = $"Не удалось удалить маршрут с ID {routeId}: {ex.Message}";

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

}

}

// Создаем сообщение об успешном выполнении операции и показываем его, используя список удаленных ID маршрутов

string successMessage = $"Удалены маршруты с ID: {deletedRotuesIds}";

MessageBox.Show(successMessage, "Успех", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

/// <summary>

/// Обработчик нажатия на кнопку фильтрации маршрутов

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

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

{

FilterRouteForm filterRouteForm = new(DataGridViewRoutes);

filterRouteForm.ShowDialog();

/\*UpdateRouteSearchTypes();\*/

}

/// <summary>

/// Обработчик нажатия на кнопку выхода из приложения

/// </summary>

/// <param name="sender">Отправитель события</param>

/// <param name="e">Аргументы события</param>

private void ButtonExit\_Click(object sender, EventArgs e) => Application.Exit();

private void TextBoxDriversSearchValue\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar != (char)Keys.Enter)

return;

string searchText = TextBoxDriversSearchValue.Text.ToLower();

CurrencyManager currencyManager = (CurrencyManager)BindingContext[DataGridViewDrivers.DataSource];

currencyManager.SuspendBinding();

foreach (DataGridViewRow row in DataGridViewDrivers.Rows)

{

row.Visible = false;

DataGridViewCell searchCell = row.Cells[ComboBoxDriversSearchTypes.SelectedIndex];

if (searchCell != null && searchCell.Value.ToString().ToLower().Contains(searchText))

row.Visible = true;

}

currencyManager.ResumeBinding();

e.Handled = true;

}

private void TextBoxTransportsSearchValues\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar != (char)Keys.Enter)

return;

string searchText = TextBoxTransportsSearchValues.Text.ToLower();

CurrencyManager currencyManager = (CurrencyManager)BindingContext[DataGridViewTransportes.DataSource];

currencyManager.SuspendBinding();

foreach (DataGridViewRow row in DataGridViewTransportes.Rows)

{

row.Visible = false;

DataGridViewCell searchCell = row.Cells[ComboBoxTransportSearchTypes.SelectedIndex];

if (searchCell != null && searchCell.Value.ToString().ToLower().Contains(searchText))

row.Visible = true;

}

currencyManager.ResumeBinding();

e.Handled = true;

}

private void TextBoxRoutesSearchValue\_KeyPress(object sender, KeyPressEventArgs e)

{

if (e.KeyChar != (char)Keys.Enter)

return;

string searchText = TextBoxRoutesSearchValue.Text.ToLower();

CurrencyManager currencyManager = (CurrencyManager)BindingContext[DataGridViewRoutes.DataSource];

currencyManager.SuspendBinding();

foreach (DataGridViewRow row in DataGridViewRoutes.Rows)

{

row.Visible = false;

DataGridViewCell searchCell = row.Cells[ComboBoxRoutesSearchTypes.SelectedIndex];

if (searchCell != null && searchCell.Value.ToString().ToLower().Contains(searchText))

row.Visible = true;

}

currencyManager.ResumeBinding();

e.Handled = true;

}

}

Файл AutoParkContext.cs

using CourseWork.Entities;

using Microsoft.EntityFrameworkCore;

namespace CourseWork.Contexts;

/// <summary>

/// Контекст базы данных для приложения автопарк.

/// </summary>

public class AutoParkContext : DbContext

{

/// <summary>

/// Коллекция объектов транспортных средств в базе данных.

/// </summary>

public DbSet<Transport> Transports { get; set; }

/// <summary>

/// Коллекция объектов водителей в базе данных.

/// </summary>

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

/// <summary>

/// Коллекция объектов маршрутов в базе данных.

/// </summary>

public DbSet<Route> Routes { get; set; }

/// <summary>

/// Путь к файлу базы данных.

/// </summary>

public string DbPath { get; }

/// <summary>

/// Инициализирует новый экземпляр класса AutoParkContext.

/// </summary>

public AutoParkContext()

{

var folder = Environment.SpecialFolder.LocalApplicationData;

var path = Environment.GetFolderPath(folder);

DbPath = Path.Join(path, "autopark.db");

}

/// <summary>

/// Настраивает параметры подключения к базе данных.

/// </summary>

/// <param name="optionsBuilder">Строитель опций для контекста базы данных.</param>

protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)

=> optionsBuilder.UseSqlite($"Data Source={DbPath}");

/// <summary>

/// Настраивает модель данных для базы данных.

/// </summary>

/// <param name="modelBuilder">Строитель модели для контекста базы данных.</param>

protected override void OnModelCreating(ModelBuilder modelBuilder)

{

// Связь один-ко-многим между Route и Transport

modelBuilder.Entity<Route>()

.HasMany(r => r.Transports) // каждый Route может иметь множество связанных сущностей Transport

.WithOne(t => t.Route) // каждая сущность Transport может быть связана только с одной сущностью Route

.HasForeignKey(t => t.RouteId); // внешний ключ для связи находится в свойстве RouteId в сущности Transport

// Связь один-к-одному между Transport и Driver

modelBuilder.Entity<Transport>()

.HasOne(t => t.Driver) // каждая сущность Transport может быть связана только с одной сущностью Driver

.WithOne(d => d.Transport) // каждая сущность Driver может быть связана только с одной сущностью Transport

.HasForeignKey<Driver>(d => d.TransportId); // внешний ключ для связи находится в свойстве TransportId в сущности Driver

}

}

Файл Driver.cs

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace CourseWork.Entities;

/// <summary>

/// Представляет водителя в системе управления автопарком.

/// </summary>

[Table("Drivers")]

public class Driver

{

/// <summary>

/// Уникальный идентификатор водителя.

/// </summary>

[Key]

[DisplayName("ID")]

public int Id { get; set; }

/// <summary>

/// Имя водителя.

/// </summary>

[Required]

[MaxLength(50)]

[DisplayName("Имя")]

public string? FirstName { get; set; }

/// <summary>

/// Фамилия водителя.

/// </summary>

[Required]

[MaxLength(50)]

[DisplayName("Фамилия")]

public string? LastName { get; set; }

/// <summary>

/// Возраст водителя.

/// </summary>

[Required]

[DisplayName("Возраст")]

public int Age { get; set; }

/// <summary>

/// Опыт вождения водителя в годах.

/// </summary>

[Required]

[Range(0, int.MaxValue)]

[DisplayName("Опыт вождения в годах")]

public int DrivingExperience { get; set; }

/// <summary>

/// Внешний ключ, связывающий водителя с транспортным средством.

/// </summary>

[DisplayName("ID Транспорта")]

public int? TransportId { get; set; }

/// <summary>

/// Навигационное свойство, которое позволяет получить объект транспортного средства, связанный с данным водителем.

/// </summary>

[DisplayName("Транспорт")]

public Transport? Transport { get; set; }

/// <summary>

/// Внешний ключ, связывающий водителя с маршрутом.

/// </summary>

[DisplayName("ID Маршрута")]

public int? RouteId { get; set; }

/// <summary>

/// Коллекция маршрутов, которые может обслуживать данный водитель.

/// </summary>

[DisplayName("Маршрут")]

public Route? Route { get; set; }

/// <summary>

/// Возвращает строковое представление объекта водителя, содержащее его идентификатор, имя, фамилию, возраст и опыт вождения.

/// </summary>

/// <returns>Строковое представление объекта водителя.</returns>

public override string ToString()

=> $"{FirstName} {LastName}, возраст: {Age}, стаж: {DrivingExperience}";

}

Файл Route.cs

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace CourseWork.Entities;

/// <summary>

/// Представляет маршрут в системе управления автопарком.

/// </summary>

[Table("Routes")]

public class Route

{

/// <summary>

/// Уникальный идентификатор маршрута.

/// </summary>

[Key]

[DisplayName("ID")]

public int Id { get; set; }

/// <summary>

/// Название маршрута.

/// </summary>

[Required]

[MaxLength(50)]

[DisplayName("Название")]

public string? Name { get; set; }

/// <summary>

/// Начальное местоположение маршрута.

/// </summary>

[Required]

[MaxLength(255)]

[DisplayName("Стартовая точка")]

public string? StartLocation { get; set; }

/// <summary>

/// Конечное местоположение маршрута.

/// </summary>

[Required]

[MaxLength(255)]

[DisplayName("Конечная точка")]

public string? EndLocation { get; set; }

/// <summary>

/// Длина маршрута в километрах.

/// </summary>

[Required]

[Range(0, int.MaxValue)]

[DisplayName("Длина маршрута")]

public int Distance { get; set; }

/// <summary>

/// Время начала движения по маршруту.

/// </summary>

[Required]

[DisplayName("Начало движения")]

public DateTime StartTime { get; set; }

/// <summary>

/// Время окончания движения по маршруту.

/// </summary>

[Required]

[DisplayName("Конец движения")]

public DateTime EndTime { get; set; }

/// <summary>

/// Коллекция транспортных средств, которые могут обслуживать данный маршрут.

/// </summary>

[DisplayName("Транспорт")]

public List<Transport?> Transports { get; set; } = new();

/// <summary>

/// Коллекция водителей, которые могут обслуживать данный маршрут.

/// </summary>

[DisplayName("Водители")]

public List<Driver?> Drivers { get; set; } = new();

/// <summary>

/// Возвращает строковое представление объекта маршрута, содержащее его идентификатор, название, начальное и конечное местоположение, длину, время начала и окончания движения по маршруту.

/// </summary>

/// <returns>Строковое представление объекта маршрута.</returns>

public override string ToString()

=> $"Название: {Name}, Нач. точка: {StartLocation}, Конеч. точка: {EndLocation}, Расстояние: {Distance}, Время начала: {StartTime}, Время конца: {EndTime}";

}

Файл Transport.cs

using System.ComponentModel;

using System.ComponentModel.DataAnnotations;

using System.ComponentModel.DataAnnotations.Schema;

namespace CourseWork.Entities;

/// <summary>

/// Представляет транспортное средство в системе управления автопарком.

/// </summary>

[Table("Transportes")]

public class Transport

{

/// <summary>

/// Уникальный идентификатор транспортного средства.

/// </summary>

[Key]

[DisplayName("ID")]

public int Id { get; set; }

/// <summary>

/// Модель транспортного средства.

/// </summary>

[Required]

[MaxLength(50)]

[DisplayName("Модель")]

public string Model { get; set; } = string.Empty;

/// <summary>

/// Государственный регистрационный номер транспортного средства.

/// </summary>

[Required]

[MaxLength(6)]

[DisplayName("Номерной знак")]

public string LicensePlate { get; set; } = string.Empty;

/// <summary>

/// Вместимость транспортного средства.

/// </summary>

[Required]

[Range(2, int.MaxValue)]

[DisplayName("Кол-во мест")]

public int Capacity { get; set; }

/// <summary>

/// Дата последнего технического обслуживания транспортного средства.

/// </summary>

[Required]

[DisplayName("Дата тех. обслуживания")]

public DateTime LastMaintenanceDate { get; set; }

/// <summary>

/// Текущий пробег транспортного средства.

/// </summary>

[Required]

[Range(0, double.MaxValue)]

[DisplayName("Пробег")]

public double Mileage { get; set; }

/// <summary>

/// Идентификатор маршрута, на котором работает транспортное средство.

/// </summary>

[DisplayName("ID Маршрута")]

public int? RouteId { get; set; }

/// <summary>

/// Маршрут, на котором работает транспортное средство.

/// </summary>

[DisplayName("Маршрут")]

public Route? Route { get; set; }

/// <summary>

/// Идентификатор водителя, который управляет транспортным средством.

/// </summary>

[DisplayName("ID Водителя")]

public int? DriverId { get; set; }

/// <summary>

/// Водитель, который управляет транспортным средством.

/// </summary>

[DisplayName("Водитель")]

public Driver? Driver { get; set; }

/// <summary>

/// Возвращает строковое представление объекта транспортного средства, содержащее его идентификатор, модель, государственный регистрационный номер и вместимость.

/// </summary>

/// <returns>Строковое представление объекта транспортного средства.</returns>

public override string ToString()

=> $"Бренд: {Model}, Номер. знак: {LicensePlate}, Вместимость: {Capacity}";

}

Файл DriverServiec.cs

using CourseWork.Contexts;

using CourseWork.Entities;

using Microsoft.EntityFrameworkCore;

namespace CourseWork.Services;

/// <summary>

/// Сервис для выполнения операций с водителями в базе данных.

/// </summary>

public class DriverService

{

private readonly AutoParkContext \_dbContext = new();

/// <summary>

/// Инициализирует новый экземпляр класса DriverService.

/// </summary>

/// <param name="dbContext">Контекст базы данных.</param>

public DriverService(AutoParkContext dbContext) => \_dbContext = dbContext;

/// <summary>

/// Возвращает список всех водителей в базе данных.

/// </summary>

/// <returns>Список всех водителей в базе данных.</returns>

public async Task<List<Driver>> GetAllDriversAsync() => await \_dbContext.Drivers.ToListAsync();

/// <summary>

/// Возвращает водителя по его идентификатору.

/// </summary>

/// <param name="id">Идентификатор водителя.</param>

/// <returns>Водитель с указанным идентификатором.</returns>

public async Task<Driver?> GetDriverByIdAsync(int id) => await \_dbContext.Drivers.FindAsync(id);

/// <summary>

/// Добавляет нового водителя в базу данных.

/// </summary>

/// <param name="driver">Новый водитель.</param>

/// <returns>Асинхронная задача.</returns>

public async Task AddDriverAsync(Driver driver)

{

\_dbContext.Drivers.Add(driver);

await \_dbContext.SaveChangesAsync();

}

/// <summary>

/// Обновляет существующего водителя в базе данных.

/// </summary>

/// <param name="driver">Водитель для обновления.</param>

/// <returns>Асинхронная задача.</returns>

public async Task UpdateDriverAsync(Driver driver)

{

\_dbContext.Update(driver);

await \_dbContext.SaveChangesAsync();

}

/// <summary>

/// Удаляет водителя из базы данных по его идентификатору.

/// </summary>

/// <param name="id">Идентификатор водителя.</param>

/// <returns>Асинхронная задача.</returns>

public async Task DeleteDriverAsync(int id)

{

var driver = await GetDriverByIdAsync(id);

if (driver != null)

{

\_dbContext.Drivers.Remove(driver);

await \_dbContext.SaveChangesAsync();

}

}

}

Файл TransportServiec.cs

using CourseWork.Contexts;

using CourseWork.Entities;

using Microsoft.EntityFrameworkCore;

namespace CourseWork.Services;

/// <summary>

/// Сервис для выполнения операций с транспортом в базе данных.

/// </summary>

public class TransportService

{

private readonly AutoParkContext \_dbContext;

/// <summary>

/// Инициализирует новый экземпляр класса TransportService.

/// </summary>

/// <param name="dbContext">Контекст базы данных.</param>

public TransportService(AutoParkContext dbContext) => \_dbContext = dbContext;

/// <summary>

/// Возвращает список всех транспортных средств в базе данных.

/// </summary>

/// <returns>Список всех транспортных средств в базе данных.</returns>

public async Task<List<Transport>> GetAllTransportsAsync() => await \_dbContext.Transports.ToListAsync();

/// <summary>

/// Возвращает транспортное средство по его идентификатору.

/// </summary>

/// <param name="id">Идентификатор транспортного средства.</param>

/// <returns>Транспортное средство с указанным идентификатором.</returns>

public async Task<Transport?> GetTransportByIdAsync(int id) => await \_dbContext.Transports.FindAsync(id);

/// <summary>

/// Добавляет новое транспортное средство в базу данных.

/// </summary>

/// <param name="transport">Новое транспортное средство.</param>

/// <returns>Асинхронная задача.</returns>

public async Task AddTransportAsync(Transport transport)

{

\_dbContext.Transports.Add(transport);

await \_dbContext.SaveChangesAsync();

}

/// <summary>

/// Обновляет существующее транспортное средство в базе данных.

/// </summary>

/// <param name="transport">Транспортное средство для обновления.</param>

/// <returns>Асинхронная задача.</returns>

public async Task UpdateTransportAsync(Transport transport)

{

\_dbContext.Update(transport);

await \_dbContext.SaveChangesAsync();

}

/// <summary>

/// Удаляет транспортное средство из базы данных по его идентификатору.

/// </summary>

/// <param name="id">Идентификатор транспортного средства.</param>

/// <returns>Асинхронная задача.</returns>

public async Task DeleteTransportAsync(int id)

{

var transport = await GetTransportByIdAsync(id);

if (transport != null)

{

\_dbContext.Transports.Remove(transport);

await \_dbContext.SaveChangesAsync();

}

}

}

Файл RouteServiec.cs

using CourseWork.Contexts;

using CourseWork.Entities;

using Microsoft.EntityFrameworkCore;

namespace CourseWork.Services;

/// <summary>

/// Сервис для выполнения операций с маршрутами в базе данных.

/// </summary>

public class RouteService

{

private readonly AutoParkContext \_dbContext;

/// <summary>

/// Инициализирует новый экземпляр класса RouteService.

/// </summary>

/// <param name="dbContext">Контекст базы данных.</param>

public RouteService(AutoParkContext dbContext) => \_dbContext = dbContext;

/// <summary>

/// Возвращает список всех маршрутов в базе данных.

/// </summary>

/// <returns>Список всех маршрутов в базе данных.</returns>

public async Task<List<Route>> GetAllRoutesAsync() => await \_dbContext.Routes.ToListAsync();

/// <summary>

/// Возвращает маршрут по его идентификатору.

/// </summary>

/// <param name="id">Идентификатор маршрута.</param>

/// <returns>Маршрут с указанным идентификатором.</returns>

public async Task<Route?> GetRouteByIdAsync(int id) => await \_dbContext.Routes.FindAsync(id);

/// <summary>

/// Добавляет новый маршрут в базу данных.

/// </summary>

/// <param name="route">Новый маршрут.</param>

/// <returns>Асинхронная задача.</returns>

public async Task AddRouteAsync(Route route)

{

\_dbContext.Routes.Add(route);

await \_dbContext.SaveChangesAsync();

}

/// <summary>

/// Обновляет существующий маршрут в базе данных.

/// </summary>

/// <param name="route">Маршрут для обновления.</param>

/// <returns>Асинхронная задача.</returns>

public async Task UpdateRouteAsync(Route route)

{

\_dbContext.Update(route);

await \_dbContext.SaveChangesAsync();

}

/// <summary>

/// Удаляет маршрут из базы данных по его идентификатору.

/// </summary>

/// <param name="id">Идентификатор маршрута.</param>

/// <returns>Асинхронная задача.</returns>

public async Task DeleteRouteAsync(int id)

{

var route = await GetRouteByIdAsync(id);

if (route != null)

{

\_dbContext.Routes.Remove(route);

await \_dbContext.SaveChangesAsync();

}

}

}