Частное учреждение образования

«Колледж бизнеса и права»

РАЗРАБОТКА ПРОГРАММОГО СРЕДСТВА ДЛЯ РАСЧЕТА СТОИМОСТИ ПОЕЗДКИ НА ЛИЧНОМ АВТОТРАНСПОРТЕ

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту по дисциплине

«Базы данных и системы управления базами данных»

КП Т.091016.401

Руководитель проекта (Н.В. Ржеутская)

Учащийся (А.С. Самойлов)

2024

*Изм.*

*Лист*

*№ докум.*

*Подпись*

*Дата*

*Лист*

*3*

*КП Т.091016.401*

*Разраб.*

*Самойлов А.С.*

*Провер.*

*Ржеутская Н.В.*

*Реценз.*

*Н. Контр.*

*Утверд.*

*Разработка программного средства для расчета стоимости поездки на личном автотранспорте*

*Лит.*

*Листов*

*КБП*

*у*

Оглавление

[Введение 4](#_Toc136948623)

[1 Описание задачи 4](#_Toc136948624)

[1.1 Анализ предметной области 6](#_Toc136948625)

[1.2 Постановка задачи 7](#_Toc136948626)

[2 Проектирование системы 8](#_Toc136948627)

[2.1 Требования к приложению 8](#_Toc136948628)

[2.2 Проектирование модели 8](#_Toc136948629)

[2.3 Концептуальный прототип 9](#_Toc136948631)

[3 Описание реализации программного средства 16](#_Toc136948632)

[3.1 Инструменты разработки и применяемые технологии 16](#_Toc136948633)

[3.2 Порядок авторизации пользователей 17](#_Toc136948634)

[3.3 Организация данных 17](#_Toc136948635)

[4 Применение 25](#_Toc136948636)

[4.1 Назначение программного средства 25](#_Toc136948637)

[Заключение 26](#_Toc136948638)

[Список информационных источников 27](#_Toc136948639)

[ПРИЛОЖЕНИЕ А 28](#_Toc136948640)

[ПРИЛОЖЕНИЕ В 76](#_Toc136948641)

Введение

В настоящее время все больше людей используют различные виды транспорта для перемещения по городу или между городами. В связи с этим возникает потребность в удобных и быстрых инструментах для определения стоимости поездки. Разработка приложения поможет упростить процесс расчета стоимости поездки для конечных пользователей.

Актуальность поставленной задачи обусловлена ростом спроса на необходимость расчета стоимости поездки. Также автоматизация данного процесса, который позволит сократить время, затрачиваемое на ручной расчет, и уменьшить вероятность ошибок в расчетах.

При разработке программного средства составлена записка, в которой будут описаны следующие разделы.

В пояснительной записке содержатся четыре раздела.

Первый раздел «Описание задачи» содержит в себе анализ предметной области, в которой автор проекта исследует предметную область, в которой описывает бизнес-процессы происходящие в предметной области, описывает алгоритмы получения промежуточных и итоговых показателей, обосновывает необходимость компьютерной обработки информации или процессов, определяет потенциальную аудиторию, а так же содержит постановку задачи в котором определяет функции которые должны быть автоматизированы на основе проанализированных бизнес-процессов и бизнес-задач, описаны существующих аналогов с указание отличий которые будут реализованы в разрабатываемом программном средстве.

Второй раздел «Проектирование системы» содержит в себе требования к аппаратным и операционным ресурсам, средства защиты будущего приложения, накладываемые ограничения на интерфейс. Также стоит отобразить в разделе диаграммы: вариантов использования с указанием типов связей между вариантами использования, диаграмма классов с выделением необходимых атрибутов и/или методов, диаграмма последовательности или деятельности. Необходимо отразить описание внешнего пользовательского интерфейса и также должны приведены графические изображения макетов и прототипов диалоговых окон.

Третий «Описание реализации программного средства» содержит в себе инструменты разработки и применяемые технологии, порядок авторизации пользователей, организация данных, функции логические и физическая организация, входные и выходные данные, функциональное тестирование, описание справочной системы.

Четвертый раздел «Применение» содержит в себе назначение программного средства, условия применения программного средства.

В заключении будет проанализировано созданное программное средства, определена степень соответствия поставленной задачи и выполненной работы.

Приложения будут содержать текст программных модулей и результаты работы программы.

Графическая часть содержит в себе диаграмму вариантов использования, диаграмма сущность-связь (концептуальная модель), диаграмма сущность-связь (логическая модель), диаграмма деятельности.

# Описание задачи

## **Анализ предметной области**

Целью является разработка программного средства на языке C# для автоматизации расчета стоимости поездки на личном автотранспорте.

Создание такого приложения имеет актуальность в настоящее время в связи с увеличивающейся потребностью в экономии времени и ресурсов, а также повышенной необходимостью контроля расходов на транспорт. Большинство людей понимают, что иметь возможность автоматически рассчитывать стоимость поездки в режиме реального времени может значительно облегчить их повседневную жизнь и повысить эффективность использования транспортных средств.

Алгоритм для расчета стоимости поездки будет включать в себя такие параметры, как расстояние между местом отправления и местом назначения, октановое число потребляемого топлива, расход потребления топлива, средняя скорость, характеристики выбранного автомобиля. Расчетное время прибытия рассчитывается по формуле (1).

*(1)*

где t – время, часы;

S– расстояние, км;

V – скорость, км/ч.

Формула стоимости поездки рассчитывается по формуле (2).

*(2)*

где С – стоимость поездки, руб;

R – расстояние, км;

P – потребление топлива на 100 км, литр;

Pp – цена топлива, руб.

Формула затраченного топлива на поездку (3)

*(3)*

где Q – затраченное топливо, литр;

S – расстояние, км;

P – потребление топлива на 100 км, литр.

Такая программа будет максимально облегчать процесс расчета и анализа данных за определенный срок времени.

Это программное средство может быть полезным для широкого круга пользователей, как для частных лиц, так и для предприятий и туристических агентств. Оно может помочь им в быстром и точном расчете стоимости поездок. Безусловно, в наше время все больше людей уделяют внимание здоровому образу жизни и правильному питанию, и это вполне понятно. Также, многие компании и предприятия стараются контролировать свои расходы, включая расходы на поездки.

## **Постановка задачи**

Необходимо разработать программное средство для автоматизации расчета стоимости поездки на личном автотранспорте, в который будут реализованы следующие функции:

* ведение справочников типов топлива, цен на выбранный тип топлива, списка транспортных средств, списком маршрутов, списком «любимых маршрутов»;
* автоматический расчет стоимости поездки по введенным критериям пользователя;
* сохранения результатов расчетов в базу данных, а также формирование выходных файлов в виде отчета MSOffice;
* формирование статистики расходов пользователя за разные промежутки времени.

В результате исследования предметной области были рассмотрены такие аналоги как: «Avtonov», «Transportdepo», «SV-auto», «Fuelio», «TripLog» были обнаружены такие отличия как:

* наличие обширных баз данных с автомобилями и маршрутами, что обеспечивает более точный и полный расчет стоимости поездки;
* кроссплатформенность, что позволяет использовать программное средство на различных операционных системах и устройствах;
* большое количество параметров для подсчета результата, что увеличивает точность расчета и гибкость программы;
* лаконичный и простой интерфейс, что делает использование программы более удобным для пользователя.

Разработанное программное средство будет обладать такими функциями как:

* возможность добавления транспортного средства, маршрута, топлива;
* возможность редактирования транспортного средства, маршрута, топлива;
* наличии истории поездок, что делает возможным отслеживание статистики и анализ прошлых поездок;
* наличие выходных данных как отчеты, что может предоставить возможность анализа данных.

# Проектирование системы

## **Требования к приложению**

Для защиты программного средства будет реализована функция регистрации и авторизации в системе для обеспечения и удобства работы пользователя.

Для работы программного средства необходимо устройство, которое будет иметь следующие характеристики:

* процессор тактовой частотой не менее 2 ГГц, архитектурой x64;
* оперативную память не менее 4 ГБ;
* место на диске от 512 МБ;
* клавиатура проводная;
* мышь проводная;
* видеопамять от 256 МБ;
* операционная система Windows 10 или Windows 11.

Для корректной работы программного средства подключение к интернету не требуется.

Ограничения на интерфейс программы включают в себя:

* размер окна: окно приложения должно быть достаточного размера, чтобы вместить все элементы управления и содержимого;
* разрешение экрана: размер элементов управления и текста должен соответствовать разрешению экрана, для корректного отображения элементов управления на разных устройствах с разным разрешением экрана;
* расположение элементов: необходимо расположить их в интуитивно понятных местах для пользователя и не загромождать окна ненужными элементами;
* шрифт и размер: тест должен быть читаем заметен на форме.

Общие требования к интерфейсу:

* использование стандартных элементов управления, таких как кнопки, текстовые поля;
* понятный и легкий интерфейс для пользователя, чтобы интерфейс был понятен пользователю и не мешал его работе в программе.

## **Проектирование модели**

Диаграммы вариантов использования описывают взаимоотношения и зависимости между группами вариантов использования и действующих лиц, участвующими в процессе. Данная диаграмма включает в себя актера, который называется «Пользователь». Данная диаграмма включает в себя следующие функции include: «ввести данные», «ввести среднюю скорость», «ввести кол-во пассажиров» «выбрать данные», «осуществить формирование результата», «сформировать результат по времени в пути», «сформировать результат по потреблению топлива», «сформировать результат по необходимой сумме денег», «осуществить работу со справочником маршрутов», «осуществить работу со справочником любимых маршрутов», «осуществить работу со справочником топлива», «осуществить работу со справочником автомобилей» Также данная диаграмма включает в себя функции extend: «предоставить информацию об автомобилях», «предоставить информацию о типах топлива», «предоставить информацию о типах маршрутах», «выбрать тип топлива», «выбрать расход топлива», «выбрать любимый маршрут», «выбрать маршрут», «выбрать тип потребления топлива», «выбрать вид топлива», «выбрать автомобиль». Данная диаграмма изображена на листе 3 в графической части.

Диаграмма деятельности представляет собой визуализацию последовательности действий или процессов в системе. Она позволяет понять, какие шаги выполняются в системе и как они взаимодействуют между собой. Данная диаграмма описывается процесс формирования поездки на автомобиле. Диаграмма деятельности представлена в графической части на листе 4.

## **Проектирование структуры базы данных**

Исходя из исследования предметной области были выделены следующие сущности: «Автомобиль», «Пользователь», «Поездка», «Маршрут», «Топливо», «Любимые маршруты».

Для сущности «Автомобиль» существуют такие атрибуты как:

* название автомобиля;
* год выпуска;
* класс автомобиля;
* максимальная скорость;
* количество мест;
* вид топлива;
* тип топлива;
* расход на 100 км смешанный.

Для сущности «Пользователь» существуют такие атрибуты как:

* логин;
* пароль.

Для сущности «Поездка» существуют такие атрибуты как:

* пользователь;
* автомобиль;
* маршрут;
* средняя скорость;
* расстояние;
* потрачено топлива;
* средний расход на 100 км;
* дата поездки;
* время отправки;
* время прибытия;
* сумма денег.

Для сущности «Маршрут» существуют такие атрибуты как:

* точка отправления;
* точка прибытия;
* расстояние.

Для сущности «Топливо» существуют такие атрибуты как:

* вид топлива;
* октановое число;
* стоимость.

Для сущности «Любимые маршруты» существуют такие атрибуты как:

* пользователь;
* маршрут.

Существует связь между сущностями «Автомобиль» и «Поездка» («используется для»), «Поездка» и «Пользователь» («совершает»), «Маршрут» и «Поездка» («проходит по»), «Автомобиль» и «Топливо» («использует»), «Пользователь» и «Любимые маршруты» («сохраняет»).

Разработанная диаграмма сущность-связь концептуальный прототип представлен в графической части на листе 1.

## **Концептуальный прототип**

При запуске программного средства будет отображаться окно аутентификации, в которое будет состоять из текстовых полей и кнопок управления. Прототип окна аутентификации изображен на рисунке 2.1.

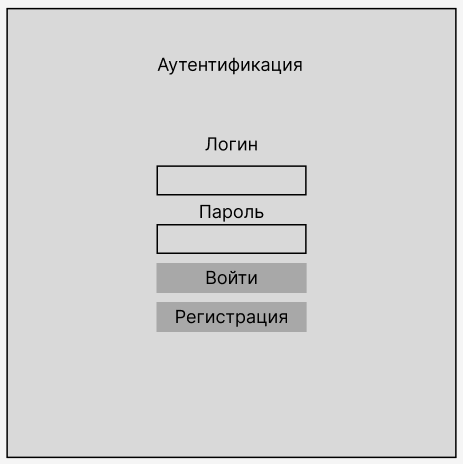


Рисунок 2.1 – Прототип окна аутентификации

В данном окне будет находится два поля для ввода данных и одна кнопка. В поля вводятся логин и пароль пользователя, который используется программой. При нажатии на кнопку «Регистрация» будет открыто окно регистрации, изображенное на рисунке 2.2.

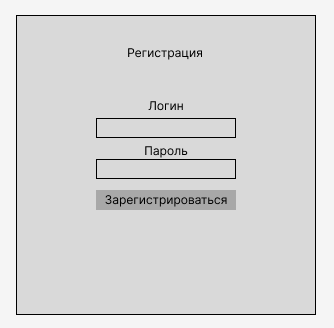


Рисунок 2.2 – Прототип окна регистрации

Окно регистрации будет иметь два поля для ввода данных о пользователе и кнопка. При нажатии «Зарегистрироваться» происходит создание нового пользователя и запись его данных в систему для последующего входа в аккаунт.

Вернемся к окну аутентификации и рассмотрим кнопку «Войти», при нажатии на которую произойдет открытие окна главной формы, изображенного на рисунке 2.3.

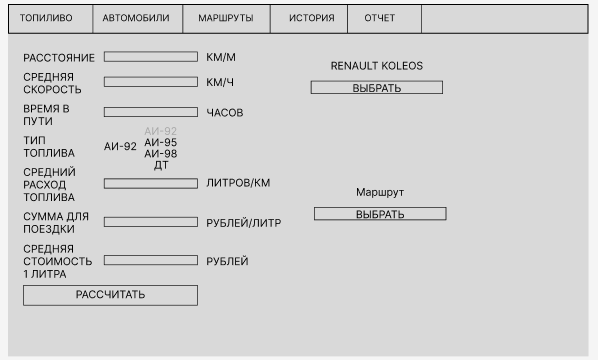


Рисунок 2.3 – Прототип главного окна

Вернемся на главное окно и рассмотрим верхнее меню. Нажмем на кнопку «Топливо» программа отобразит меню с 2 пунктами «Добавить топливо» и «Редактировать топливо». При нажатии на кнопку «Добавить топливо» отобразиться окно добавления топлива в систему, изображенное на рисунке 2.4.

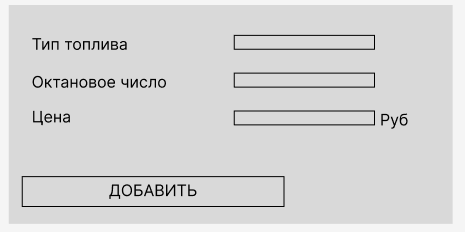


Рисунок 2.4 – Прототип окна добавления топлива

При выборе пункта «Редактировать топливо» отобразиться окно редактирования топлива в системе, изображенное на рисунке 2.5.

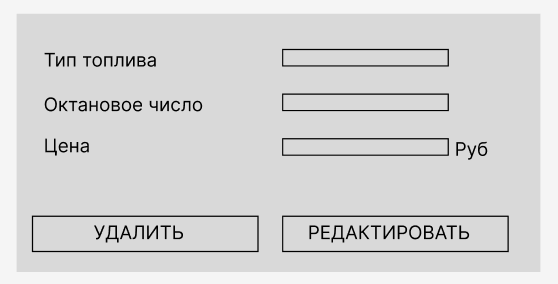


Рисунок 2.5 – Прототип окна редактирования топлива

Вернемся на главное окно и рассмотрим верхнее меню. Нажмем на кнопку «Автомобили» программа отобразит меню с 2 пунктами «Добавить автомобиль» и «Редактировать автомобиль». При нажатии на кнопку «Добавить автомобиль» отобразиться окно добавления автомобиля в систему, изображенное на рисунке 2.6.

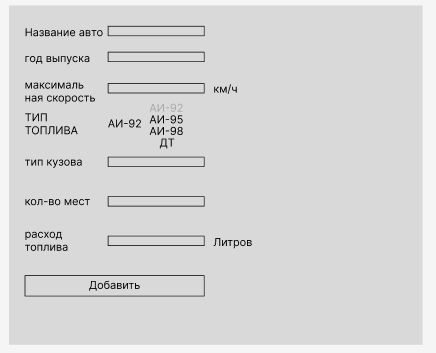


Рисунок 2.6 – Прототип окна добавления автомобиля

При выборе пункта «Редактировать автомобиль» отобразиться окно редактирования автомобиля в системе, изображенное на рисунке 2.7.

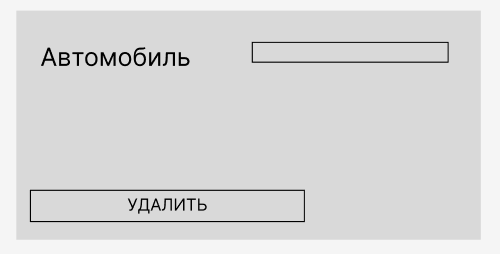


Рисунок 2.7 – Прототип окна редактирования автомобиля

Вернемся к главному окну и рассмотрим кнопку «Маршруты» при нажатии на которую программа отобразит меню с 2 пунктами «Добавить маршрут» и «Редактировать маршрут». При нажатии на кнопку «Добавить маршрут» отобразится окно для добавления нового маршрута в систему, изображенное на рисунке 2.8.

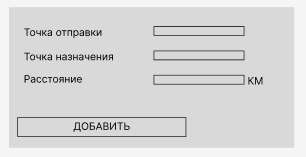


Рисунок 2.8 – Прототип окна добавления маршрута

При выборе пункта «Редактировать маршрут» отобразиться окно редактирования маршрута в системе, изображенное на рисунке 2.9.

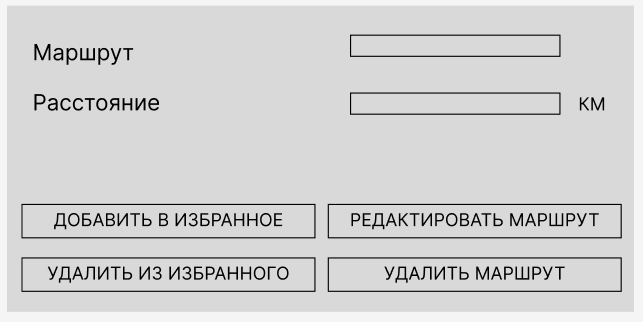


Рисунок 2.9 – Прототип окна редактирования маршрута

Вернемся к главному окну и рассмотрим кнопку «История», при нажатии на которую программа отобразит окно истории поездок с таблицей и полями для фильтрации для данных в таблице. Данное окно, изображенное на рисунке 2.10.

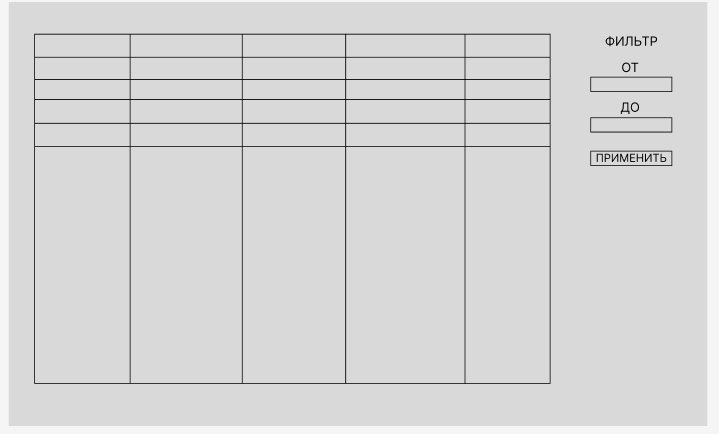


Рисунок 2.10 – Прототип окна истории поездок

Вернемся к главному окну и рассмотрим кнопку «Отчет» при нажатии на которую программа отобразит окно для создания отчета по поездкам. Окно будет иметь поля для фильтрации данных для отчета. Данное окно, изображенное на рисунке 2.11.



Рисунок 2.11 – Прототип окна создания отчета

# Описание реализации программного средства

## **Инструменты разработки и применяемые технологии**

В ходе реализации программного средства были использованы следующие инструменты:

* операционная система Windows 10;
* среда разработки MS Visual Studio 2022;
* язык программирования C#;
* язык программирования SQL;
* технология WPF;
* Microsoft Word;
* Microsoft SQL Server Management Studio 2018.

Windows – группа семейств коммерческих операционных систем корпорации Microsoft. Данная операционная система является универсальной и может использовать для разных целей. Данная операционная система позволяет пользователю компьютера работать с файлами, пользоваться Интернетом, просматривать видеозаписи, прослушивать музыку и многое другое. В данное время операционные системы Windows являются самыми распространенными операционными системами в мире.

Для разработки приложения использовались среда разработки Visual Studio Community. Так как данное программное обеспечение очень удобно и практично для разработки приложения на WPF.

Программное средство было разработано языке программирования C# с использованием WPF для создания окон программы. C# - является компилируемым, статистически типизированным языком программирования общего назначения.

Для выбора языка C# стало его быстродействие в выполнении поставленных задач и желание улучшить навыки владения данным языком программирования.

SQL (Structured Query Language) – это язык программирования, который используется для управления данными в реляционных базах данных. Он предоставляет различные операторы и команды для создания, изменения, извлечения и управления данными в базе данных. SQL широко используется во многих типах приложений, таких как веб-сайты, бизнес-приложения, аналитические системы и многое другое.

Для реализации нашего приложения была выбрана технология Windows Presentation Foundation (WPF) в сочетании с инструментами разработки из экосистемы Microsoft. WPF представляет собой фреймворк для создания графического интерфейса пользователя (GUI) в операционной системе Windows. Его основное назначение состоит в разработке интерактивных и эстетически привлекательных приложений с использованием мощного набора инструментов и функциональности. В проектируемом приложении WPF используется для создания интерфейса пользователя. Благодаря богатым возможностям WPF, разработчики могут создавать элементы управления, анимацию, стили и многое другое, что позволяет создать привлекательный и интуитивно понятный пользовательский интерфейс, соответствующий требованиям нашего проекта. В проектируемом приложении WPF используется для создания интерфейса пользователя.

Microsoft Word — [текстовый процессор](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D0%BF%D1%80%D0%BE%D1%86%D0%B5%D1%81%D1%81%D0%BE%D1%80), предназначенный для создания, просмотра, редактирования и форматирования текстов статей, деловых бумаг, а также иных [документов](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D0%BE%D0%B2%D1%8B%D0%B9_%D1%84%D0%B0%D0%B9%D0%BB), с локальным применением простейших форм [таблично](https://ru.wikipedia.org/wiki/%D0%A2%D0%B0%D0%B1%D0%BB%D0%B8%D1%86%D0%B0)-[матричных](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) алгоритмов. Выпускается [корпорацией Microsoft](https://ru.wikipedia.org/wiki/Microsoft) в составе [пакета](https://ru.wikipedia.org/wiki/%D0%9E%D1%84%D0%B8%D1%81%D0%BD%D1%8B%D0%B9_%D0%BF%D0%B0%D0%BA%D0%B5%D1%82) [Microsoft Office](https://ru.wikipedia.org/wiki/Microsoft_Office).

SQL Server Management Studio 18 – это интегрированная среда для управления любой инфраструктурой SQL, от SQL Server до баз данных SQL Azure. SSMS предоставляет средства для настройки, наблюдения и администрирования экземпляров SQL Server и баз данных. Используется SSMS для развертывания, мониторинга и обновления компонентов уровня данных, используемых приложениями, и создания запросов и скриптов.

## **Порядок авторизации пользователей**

В программном средстве реализована роль пользователя. Для данного приложения необходима аутентификация и авторизация пользователей.

Роль пользователя имеет такие функции как:

* расчет стоимости поездки, потребления топлива, времени поездки;
* добавление и редактирование автомобилей в базе данных;
* добавление и редактирование маршрутов в базе данных;
* добавление и редактирование типов топлива в базе данных;
* формирование статистики поездок пользователя за разные периоды времени;
* формирование отчета за определенную поездку или определенный период;
* ведение справочников типов топлива, цен топлива, списка транспортных средств, маршрутов.

## **Организация данных**

Хранение данных программы реализовано при помощи реляционной базы данных SQL. Данная база данных хранит в себе 6 сущностей такие как: «Пользователь», «Автомобиль», «Маршрут», «Топливо», «Поездки», «Любимые маршруты».

Архитектура базы данных спроектирована таким образом, чтобы обеспечить эффективное хранение и управление информацией, а также обеспечить целостность данных. Каждая таблица базы данных отражает определенный аспект функциональности приложения и содержит соответствующие поля для хранения данных.

Сущность «Пользователь» хранит в себе информацию обо всех пользователях необходимую для аутентификации в приложении. Описание всех полей данной сущности описывается в таблице 3.1.

Таблица 3.1 – Информация о сущности «Пользователь»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Размер, байт | Описание |
| UserID | INT | 4 | Идентификатор пользователя |
| Login | NVARCHAR (50) | До 100 | Логин пользователя |
| Password | NVARCHAR (50) | До 100 | Пароль пользователя |

Сущность «Автомобиль» хранит в себе информацию обо всех автомобилях в системе необходимых для работы приложения. Описание всех полей данной сущности описаны в таблице 3.2.

Таблица 3.2 – Информация о сущности «Автомобиль»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Размер, байт | Описание |
| IdCar | INT | 4 | Идентификатор автомобиля |
| Name | NVARCHAR(MAX) | 100 | Название автомобиля |
| Year | INT | 4 | Год выпуска автомобиля |
| TypeCar | NVARCHAR(MAX) | 100 | Кузов автомобиля |
| MaxSpeed | INT | 4 | Максимальная скорость автомобиля |
| SeatingCapacity | INT | 4 | Кол-во мест в автомобиле |
| TypeFuel | NVARCHAR (50) | До 100 | Тип топлива автомобиля |
| FuelOctan | NVARCHAR (50) | До 100 | Октановое число топлива |
| ConsumptionFuel | NUMERIC (10, 2) | 100 | Потребление топлива |

Сущность «Маршрут» хранит в себе информацию обо всех маршрутах в системе необходимых для работы приложения. Описание всех полей данной сущности описаны в таблице 3.3.

Таблица 3.3 – Информация о сущности «Маршрут»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Размер, байт | Описание |
| RouteID | INT | 4 | Идентификатор маршрута |
| PointA | NVARCHAR (100) | До 200 | Точка A конца маршрута |
| PointB | NVARCHAR (100) | До 200 | Точка B конца маршрута |
| RouteLength | DECIMAL (10, 2) | 100 | Дистанция маршрута в километрах |

Сущность «Топливо» хранит в себе информацию о видах топлива и его цене. Данная таблица используется для формирования стоимости поездки. Описание всех полей данной сущности описаны в таблице 3.4.

Таблица 3.4 – Информация о сущности «Топливо»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Размер, байт | Описание |
| FuelId | INT | 4 | Идентификатор вида топлива |
| FuelType | NVARCHAR (50) | До 100 | Тип топлива |
| OctaneNumber | NVARCHAR (20) | До 40 | Октановое число топлива |
| FuelPrice | NUMERIC (10, 2) | 100 | Цена топлива за литр |

Сущность «Поездка» хранит в себе информацию о поездках пользователя. Данная таблица используется для хранения истории всех поездок и формировании отчетов по поездкам. Описание всех полей данной сущности описаны в таблице 3.5.

Таблица 3.5 – Информация о сущности «Поездка»

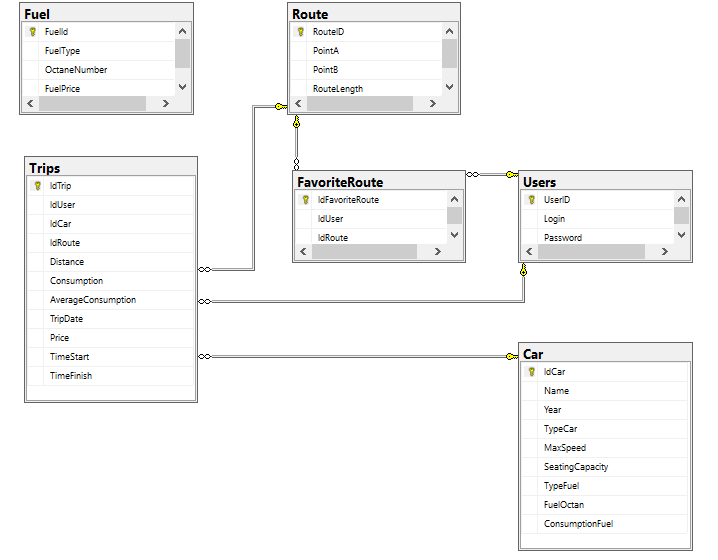
|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Размер, байт | Описание |
| IdTrip | INT | 4 | Идентификатор поездки |
| IdUser | INT | 4 | Идентификатор пользователя |
| IdCar | INT | 4 | Идентификатор автомобиля |
| IdRoute | INT | 4 | Идентификатор маршрута |
| Distance | NUMERIC (10, 2) | 100 | Дистанция поездки в километрах |
| Consumption | NUMERIC (10, 2) | 100 | Общий расход топлива в поездке |
| AverageConsumption | NUMERIC (10, 2) | 100 | Средний расход топлива на 100 км |
| TripDate | DATE | 3 | Дата поездки |
| Price | NUMERIC (10, 2) | 100 | Стоимость поездки |
| TimeStart | DATETIME | 8 | Время начала поездки |
| TimeFinish | DATETIME | 8 | Примерное время окончания поездки |

Сущность «Любимый маршрут» хранит в себе информацию о любимых маршрутах пользователя. Данная таблица используется для хранения любимых маршрутов пользователя. Описание всех полей данной сущности описаны в таблице 3.6.

Таблица 3.6 – Информация о сущности «Любимые маршруты»

|  |  |  |  |
| --- | --- | --- | --- |
| Поле | Тип | Размер, байт | Описание |
| IdFavoriteRoute | INT | 4 | Идентификатор избранного маршрута |
| IdUser | INT | 4 | Идентификатор пользователя |
| OctaneNumber | INT | 4 | Идентификатор маршрута |

Также для данного раздела была разработана физическая модель базы данных. Физическая модель базы данных представлена на рисунке 3.1.

 Рисунок 3.1 – Физическая модель базы данных

* 1. **Функции: логическая и физическая организация**

В базе данных для работы с таблице «Trips» была разработана функция GetTotalTripPricesForUser. Функция предназначена для подсчета общий суммы затрат на поездки для заданного пользователя. Функция принимает в качестве входного параметра идентификатор пользователя (@UserId) и возвращает общую сумму затрат в виде числа с фиксированной точкой (NUMERIC(10,2)).Код функции GetTotalTripPricesForUser представлен ниже.

CREATE FUNCTION dbo.GetTotalTripPricesForUser(

@UserId INT

)

RETURNS NUMERIC(10, 2)

AS

BEGIN

DECLARE @TotalPrice NUMERIC(10, 2);

SELECT @TotalPrice = SUM(Price)

FROM Trips

WHERE IdUser = @UserId;

RETURN @TotalPrice;

END;

В данном программном средстве была реализована процедура GetFuelByFuelType. Процедура предназначена для извлечения данных о топливе из таблицы Fuel по заданному виду толпива. Процедура принимает в качестве входного параметра тип топлива (@FuelType) и возвращает все записи из таблицы Fuel, где значение столбца FuelType соответствует переданному параметру. Код процедуры представлен ниже.

CREATE PROCEDURE GetFuelByFuelType

@FuelType NVARCHAR(50)

AS

BEGIN

SELECT \* FROM Fuel WHERE FuelType = @FuelType;

END;

Также был разработан триггер CountCarsAfterDelete для таблицы Cars. Триггер срабатывает после удаления автомобиля из таблицы Car. Его основная цель – отслеживание изменений в таблице и информирование о количестве автомобилей в системе после каждой операции удаления. Код триггера представлен ниже.

CREATE TRIGGER CountCarsAfterDelete

ON Car

AFTER DELETE

AS

BEGIN

DECLARE @CarCount INT;

SELECT @CarCount = COUNT(\*) FROM Car;

PRINT 'Количество автомобилей в системе после удаления: ' + CAST(@CarCount AS NVARCHAR);

END;

* 1. **Входные и выходные данные**

Для роли пользователя определены следующие входные данные:

* логин;
* пароль;
* добавление данных о маршрутах;
* добавление данных об автомобилях;
* добавление данных о типах топлива;
* добавление данных о ценах на топлива;
* данные для расчета о поездках.

Для пользователя выходные данные будут:

* отчет за текущую поездку;
* отчет расходов пользователя за определенный срок.
  1. **Функциональное тестирование**

Функциональное тестирование – это тестирование программного обеспечения (ПО) в целях проверки реализации функциональных требований в определенных условиях решать задачи, поставленные пользователем.

В данное программное средстве реализованная система ситуаций и ошибок, которые обрабатывается в программное средство (ПС), информация о ошибке выводится в выпадающем окне.

Для функции регистрации, аутентификации, добавления маршрута, расчета стоимости поездки составлены тест-кейсы, отображенные в таблице 3.7.

Таблица 3.7 – Тест-кейсы для функций

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль/функция | Шаги воспроизведения | Результат |
| 1 | Регистрация пользователя | 1. Осуществить переход по кнопке «Зарегистрироваться».  2.Заполнить требуемые поля валидными данными:  - «Логин» – Samsik;  - «Пароль» – 123123.  3.Нажать на кнопку «Зарегистрироваться». | Ожидаемый: отображается уведомление о успешной регистрации. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены Результаты тестирования приведены на Рисунке Б.1. |

Продолжение таблицы 3.7

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль/функция | Шаги воспроизведения | Результат |
| 2 | Авторизация в приложении | 1. Осуществить переход в окно аутентификации.  2. Заполнить требуемые поля необходимыми данными:  - «Логин» – Samsik;  - «Пароль» – Не заполнен.  3. Нажать кнопку «войти». | Ожидаемый: отображается уведомление о некорректных данных. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.2. |
| 3 | Добавление маршрута | 1. Выбрать кнопку «Добавить маршрут» в окне «Окно администратора».  2. Заполнить требуемые поля необходимыми данными:  - «Точка отправления» – Минск;  - «Точка прибытия» – Сочи;  - «Расстояние» - 2660;  3. Нажать кнопку "Добавить маршрут". | Ожидаемый: отображается уведомление о добавлении нового маршрута. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.3. |
| 4 | Расчет стоимости поездки | 1. Осуществить переход в окно «Главная».  2. Заполнить все поля необходимыми корректными данными.  3. Нажать кнопку «Рассчитать». | Ожидаемый: отображается уведомление о расчете стоимости поездки и вывод рассчитанных данных на окно. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.4. |
| 5 | Расчет стоимости поездки | 1. Осуществить переход в окно «Главная».  2. Заполнить некоторые поля некорректными и некоторые корректными данными.  3. Нажать кнопку "Рассчитать". | Ожидаемый:  отображается уведомление о некорректно введенных данных и отображение некорректно введённых данных. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.5. |

Продолжение таблицы 3.7

|  |  |  |  |
| --- | --- | --- | --- |
| № | Модуль/функция | Шаги воспроизведения | Результат |
| 6 | Добавление автомобиля | 1. Выбрать кнопку «Добавить автомобиль» в окне «Окно администратора».  2. Заполнить требуемые поля необходимыми корректными данными.  3. Нажать кнопку «Добавить автомобиль». | Ожидаемый: отображается уведомление о добавлении нового автомобиля. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.6. |
| 7 | Добавление любимого маршрута | 1. Выбрать кнопку «Редактировать маршрут» в окне «Главное окно».  2. Заполнить требуемые поля необходимыми корректными данными.  3. Нажать кнопку «Добавить в избранное» | Ожидаемый: отображается уведомление о добавлении нового автомобиля. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.7. |
| 8 | Удаление любимого маршрута | 1. Выбрать кнопку «Редактировать маршрут» в окне «Главное окно».  2. Заполнить требуемые поля необходимыми корректными данными.  3. Нажать кнопку «Удалить из избранного» | Ожидаемый: отображается уведомление о добавлении нового автомобиля. |
| Фактический: фактический результат соответствует ожидаемому, все условия соблюдены. Результаты тестирования приведены на рисунке Б.7. |

# Применение

## **Назначение программного средства**

Программное средство разработано для автоматизации расчета стоимости поездки личном автотранспорте. Данное программное средства называется «TripNode». Программное средство может быть использовано как обычными водителями для планирования бюджета поездки, так и компаниями, занимающимися организацией транспортных услуг, для более точного расчета затрат на топливо и маршруты. Для запуска программного средства установка каких-то дополнительных библиотек не требуется, все необходимые библиотеки устанавливаются автоматически.

* 1. **Условия применения**

Средства, ограничивающие работу приложения, не были выявлена. Для корректной работы программного средства требуется соблюдать минимальные системные требования программы, а именно:

* процессор тактовой частотой не менее 2 ГГц, архитектурой x64;
* оперативную память не менее 4 ГБ;
* место на диске от 512 МБ;
* клавиатура проводная;
* мышь проводная;
* видеопамять от 256 МБ;
* операционная система Windows 10 или Windows 11.

Для работы с программным средством необходимо установить Microsoft

.Net Framework 4.8. Данный компонент будет предложено установить в ходе установки программы.

# Заключение

В результате работы над курсовым проектом было разработано и создано программное средство, реализующее программу для автоматизации стоимости поездки.

Для разработки программного средства было изучено и сформировано значение о WPF, а также значение об основах взаимодействия десктоп приложений с базами данных SQL. В ходе тестирования приходилось корректировать уже созданный алгоритм для более точного расчета.

В данном программном средства реализованы такие задачи как: регистрация и авторизация в системе, расчет затрат на поездку, ведение справочников системы, формирование статистика поездок, формирование статистики пользователя определенных промежуток времени.

Преимуществом данного программного средства является легкость в освоении, понятный интерфейс и низкие системные требования.

Недостатками данного ПС является такие свойства как: ограниченный функционал, отсутствие кроссплатформенности, отсутствие обновление, отсутствие обновления цен на топливо в реальном времени.

Разработанное программное средство является готовым продуктом, готовым к использованию.

# Список информационных источников

1. Агуров, Павел. C#. Сборник рецептов / Павел Агуров. - М.: «БХВ-Петербург», **2012**. - 432 c
2. Альфред, В. Ахо. Компиляторы. Принципы, технологии и инструментарий / Альфред В. Ахо и др. - М.: Вильямс, 2015. **- 266** c.
3. Багласова, Т.Г. Методические указания по выполнению дипломного проекта для учащихся по специальности 2-40 01 01 «Программное обеспечение технологий» / Т.Г. Багласова. – Минск : КБП, 2024. – 38 c.
4. Багласова, Т.Г. Методические указания по оформлению курсовых и дипломных проектов / Т.Г. Багласова, К.О. Якимович. – Минск : КБП, 2022. – 41 c.
5. Бондарь, А.Г. Microsoft SQL Server 2012 / А.Г. Бондарь. – СПб. : БХВ-Петербург, 2013. – 608 с.
6. Петкович, Д. Microsoft SQL Server 2012. Руководство для начинающих: пер. с английского / Д. Петкович. – СПб.: БХВ-Петербург, 2013. – 816 с.: ил.
7. Подбельский В.В. Программирование. Базовый курс С#: учебник для вузов/ В.В. Подбельский. – Москва: Издательство Юрайт, 2020. – 369 c.
8. Подбельский, В. В. Язык С#. Базовый курс / В.В. Подбельский. - М.: Финансы и статистика, Инфра-М, 2011. - 384 c
9. Прайс, Джейсон Visual C# 2.0. Полное руководство / Джейсон Прайс, Майк Гандэрлой. - М.: Век +, Корона-Век, Энтроп, **2010**. - 736 c.
10. Metanit [Электронный ресурс] – Metanit, 2023. – Режим доступа https://metanit.com/sharp/wpf/. – Дата доступа : 11.03.2004.
11. Microsoft Learn [Электронный ресурс] – Microsoft, 2024. – Режим доступа: https://learn.microsoft.com/ru-ru/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16. – Дата доступа : 11.03.2024.

# ПРИЛОЖЕНИЕ А

**(обязательное)**

**Текст программных модулей**

<Window x:Class="TripNode.UserForm.AuthForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Title="Аутентификация"

Icon="/Icon/car.ico"

Height="456" MinHeight="456" MaxHeight="456"

Width="456" MinWidth="456" MaxWidth="456" >

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="104\*"/>

<RowDefinition Height="55\*"/>

<RowDefinition Height="61\*"/>

<RowDefinition Height="61\*"/>

<RowDefinition Height="55\*"/>

<RowDefinition Height="111\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="23\*"/>

<ColumnDefinition Width="22\*"/>

<ColumnDefinition Width="22\*"/>

<ColumnDefinition Width="23\*"/>

</Grid.ColumnDefinitions>

<Button x:Name="AuthButton" Content="Войти" Margin="38,12,38,0" VerticalAlignment="Top" Grid.ColumnSpan="2" Height="25" Grid.Row="3" Grid.Column="1" Click="AuthButton\_Click" />

<TextBox x:Name="textBoxLogin" Grid.ColumnSpan="2" Grid.Row="1" VerticalAlignment="Center" Height="30" Grid.Column="1" FontSize="20"/>

<Button x:Name="RegButton" Grid.Column="1" Content="Зарегистрироваться" Margin="38,47,38,0" Grid.Row="3" VerticalAlignment="Top" Grid.ColumnSpan="2" Grid.RowSpan="2" Height="25" Click="RegButton\_Click"/>

<PasswordBox x:Name="textBoxPassword" Grid.ColumnSpan="2" Grid.Row="2" VerticalAlignment="Top" Height="30" Grid.Column="1" Margin="0,33,0,0" Grid.RowSpan="2" FontSize="20"/>

<Label Grid.Column="1" Content="Логин" Grid.ColumnSpan="2" HorizontalContentAlignment="Center" FontSize="18" Margin="55,67,55,0" Height="35" VerticalAlignment="Top"/>

<Label Grid.Column="1" Content="Пароль" Grid.Row="1" Grid.ColumnSpan="2" HorizontalContentAlignment="Center" Height="32" VerticalAlignment="Top" FontSize="18" Margin="52,48,58,0" Grid.RowSpan="2"/>

</Grid>

</Window>

using System.Windows;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для AuthForm.xaml

/// </summary>

public partial class AuthForm : Window

{

private readonly Database dbManager;

private readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

public AuthForm()

{

InitializeComponent();

dbManager = new Database(connectionString);

}

private void RegButton\_Click(object sender, RoutedEventArgs e)

{

UserForm.RegForm regForm = new UserForm.RegForm();

regForm.ShowDialog();

}

private void AuthButton\_Click(object sender, RoutedEventArgs e)

{

if (textBoxLogin.Text != "" && textBoxPassword.Password != "")

{

if (dbManager.CheckUser(textBoxLogin.Text, textBoxPassword.Password))

{

MainWindow userForm = new MainWindow(dbManager.GetIdUser(textBoxLogin.Text,textBoxPassword.Password));

userForm.ShowDialog();

}

}

else

{

MessageBox.Show("Заполните все поля и повторите попытку!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

<Window x:Class="TripNode.UserForm.RegForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Регистрация"

Height="456" MinHeight="456" MaxHeight="456"

Width="456" MinWidth="456" MaxWidth="456" >

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="104\*"/>

<RowDefinition Height="55\*"/>

<RowDefinition Height="61\*"/>

<RowDefinition Height="61\*"/>

<RowDefinition Height="55\*"/>

<RowDefinition Height="111\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="23\*"/>

<ColumnDefinition Width="22\*"/>

<ColumnDefinition Width="22\*"/>

<ColumnDefinition Width="23\*"/>

</Grid.ColumnDefinitions>

<TextBox x:Name="textBoxLogin" Grid.ColumnSpan="2" Grid.Row="1" VerticalAlignment="Center" Height="30" Grid.Column="1" FontSize="20"/>

<Button x:Name="RegButton" Grid.Column="1" Content="Зарегистрироваться" Margin="38,47,38,0" Grid.Row="3" VerticalAlignment="Top" Grid.ColumnSpan="2" Grid.RowSpan="2" Click="RegButtonClick" Height="25"/>

<PasswordBox x:Name="textBoxPassword" Grid.ColumnSpan="2" Grid.Row="2" VerticalAlignment="Top" Height="30" Grid.Column="1" Margin="0,33,0,0" Grid.RowSpan="2" FontSize="20"/>

<Label Grid.Column="1" Content="Логин" Grid.ColumnSpan="2" HorizontalContentAlignment="Center" FontSize="18" Margin="55,67,55,0" Height="35" VerticalAlignment="Top"/>

<Label Grid.Column="1" Content="Пароль" Grid.Row="1" Grid.ColumnSpan="2" HorizontalContentAlignment="Center" Height="32" VerticalAlignment="Top" FontSize="18" Margin="52,48,58,0" Grid.RowSpan="2"/>

</Grid>

</Window>

using System.Text.RegularExpressions;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для RegForm.xaml

/// </summary>

public partial class RegForm : Window

{

private readonly Database dbManager;

public static readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

public RegForm()

{

InitializeComponent();

dbManager = new Database(connectionString);

}

private void RegButtonClick(object sender, RoutedEventArgs e)

{

// Проверяем валидность введенного логина и пароля

if (TextBoxValid(textBoxLogin) & PasswordValid(textBoxPassword))

{

User user = new User(textBoxLogin.Text, textBoxPassword.Password);

// Пытаемся добавить пользователя в базу данных

if (dbManager.RegisterUser(user))

{

// Выводим уведомление об успешном добавлении пользователя

MessageBox.Show("Пользователь добавлен", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

this.Close(); // Закрываем текущее окно регистрации

}

else

{

// Выводим сообщение об ошибке, если логин или пароль уже заняты

MessageBox.Show("Данный логин или пароль занят. Повторите попытку", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

// Выводим сообщение об ошибке, если введены некорректные данные

MessageBox.Show("Пользователь не добавлен. Возможно ошибка в том, что:\n· Пустое поле\n· Длина поля меньше 3 символов\n· Использование запрещенных символов", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private bool TextBoxValid(TextBox text)

{

if (text.Text != "" & Regex.IsMatch(text.Text, @"^[a-zA-Zа-яА-Я\s\d\-\_]+$") & text.Text.Length > 3)

{

text.BorderBrush = Brushes.Gray;

return true;

}

else

{

text.BorderBrush = Brushes.Red;

return false;

}

}

private bool PasswordValid(PasswordBox text)

{

if (text.Password != "" & Regex.IsMatch(text.Password, @"^[a-zA-Zа-яА-Я\s\d\-\_]+$") & text.Password.Length > 3 )

{

text.BorderBrush = Brushes.Gray;

return true;

}

else

{

text.BorderBrush = Brushes.Red;

return false;

}

}

}

}

<Window x:Class="TripNode.UserForm.AddCarForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Title="Добавить автомобиль"

Icon="/Icon/car.ico"

Height="450" MaxHeight="450" MinHeight="450"

Width="800" MaxWidth="800" MinWidth="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="30\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="47\*"/>

<RowDefinition Height="16\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="25\*"/>

<ColumnDefinition Width="180\*"/>

<ColumnDefinition Width="227\*"/>

<ColumnDefinition Width="187\*"/>

<ColumnDefinition Width="181"/>

</Grid.ColumnDefinitions>

<TextBox x:Name="textBoxName" FontSize="15" Grid.Column="2" Grid.Row="1" Margin="10,0,10,0" Height="27" VerticalAlignment="Center"/>

<Label Content="Марка автомобиля" Grid.Column="1" Grid.Row="1" Margin="10,10,10,0" Height="29" VerticalAlignment="Top" FontSize="15"/>

<TextBox x:Name="textBoxYear" FontSize="15" Grid.Column="2" Grid.Row="2" Margin="10,0,10,0" Height="27" VerticalAlignment="Center"/>

<Label Content="Год выпуска" Grid.Column="1" Grid.Row="2" Margin="10,0,10,0" Height="29" VerticalAlignment="Center" FontSize="15"/>

<Label x:Name="LabelTypeCar" Content="Тип кузова автомобиля" Grid.Column="1" Grid.Row="4" Margin="5,9,0,0" Height="29" VerticalAlignment="Top" FontSize="15" HorizontalContentAlignment="Center" HorizontalAlignment="Left" Width="175"/>

<TextBox x:Name="textBoxPlace" FontSize="15" Grid.Column="2" Grid.Row="5" Margin="10,0,10,0" Height="27" VerticalAlignment="Center"/>

<Label Content="Кол-во мест" Grid.Column="1" Grid.Row="5" Margin="10,0,10,0" Height="29" VerticalAlignment="Center" FontSize="15"/>

<Label x:Name="LabelTypeFuel" Content="Вид топлива" Grid.Column="1" Grid.Row="6" Margin="10,0,10,0" Height="29" VerticalAlignment="Center" FontSize="15"/>

<Label Content="Расход топлива" Grid.Column="3" Grid.Row="1" Margin="18,11,9,0" Height="29" VerticalAlignment="Top" FontSize="15"/>

<TextBox x:Name="textBoxFuelConsumptionGeneral" Grid.Column="4" Margin="0,8,0,15" Grid.Row="1" FontSize="15" HorizontalAlignment="Center" Width="161">

<TextBox.ToolTip>

Смешанный расход

</TextBox.ToolTip>

</TextBox>

<ComboBox x:Name="comboBoxFuelType" Grid.Column="2" Grid.Row="6" VerticalAlignment="Center" Height="27" Margin="10,0,10,0" SelectionChanged="comboBoxFuelType\_SelectionChanged"/>

<ComboBox x:Name="comboBoxCarType" Grid.Column="2" Grid.Row="4" VerticalAlignment="Center" Height="27" Margin="10,0,10,0"/>

<Button Grid.Column="3" Content="Добавить автомобиль" HorizontalAlignment="Center" Grid.Row="2" VerticalAlignment="Center" Click="Button\_Click" Height="27" Width="165"/>

<Label Content="Максимальная скорость" Grid.Column="1" Grid.Row="3" Margin="10,0,10,0" Height="29" VerticalAlignment="Center" FontSize="13"/>

<TextBox x:Name="textBoxMaxSpeed" FontSize="15" Grid.Column="2" Grid.Row="3" Margin="10,0,10,0" Height="27" VerticalAlignment="Center"/>

<Label x:Name="OctanLabel" Content="Октановое число" Grid.Column="1" Grid.Row="7" Margin="10,0,10,0" Height="29" VerticalAlignment="Center" FontSize="15"/>

<ComboBox x:Name="comboBoxOctan" Grid.Column="2" Grid.Row="7" VerticalAlignment="Center" Height="27" Margin="10,0,10,0"/>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для AddCarForm.xaml

/// </summary>

public partial class AddCarForm : Window

{

private readonly Database dbManager;

private readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

public event EventHandler UpdateValue;

public AddCarForm()

{

InitializeComponent();

InitComboBox();

dbManager = new Database(connectionString);

}

private List<string> Fuel = new List<string>() { "Бензин", "Дизельное топливо" };

private List<string> CarType = new List<string>() { "Седан", "Купе", "Хэтчбек", "Универсал", "Внедорожник", "Кроссовер", "Кабриолет", "Лифтбек", "Фургон", "Минивэн", "Пикап" };

public void InitComboBox()

{

comboBoxFuelType.ItemsSource = Fuel;

comboBoxCarType.ItemsSource = CarType;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (ValidValue())

{

string name = this.textBoxName.Text;

int year = Convert.ToInt32(this.textBoxYear.Text);

string typeCar = this.comboBoxCarType.SelectedItem.ToString();

int seatingCapacity = Convert.ToInt32(this.textBoxPlace.Text);

int maxSpeed = Convert.ToInt32(this.textBoxMaxSpeed.Text);

string fuel = this.comboBoxFuelType.SelectedItem.ToString();

string fuelType;

if (fuel == "Дизельное топливо" & comboBoxOctan.SelectedIndex == 0)

{

fuelType = "";

}

else

{

fuelType = comboBoxOctan.SelectedItem.ToString();

}

double fuelConsumptionGeneral = DoubleNull(textBoxFuelConsumptionGeneral);

Car car = new Car(name, year, typeCar, maxSpeed, seatingCapacity, fuel, fuelType, fuelConsumptionGeneral);

if (dbManager.InsertCar(car))

{

MessageBox.Show($"Автомобиль: {name} был добавлен в систему.", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show($"Автомобиль: {name} не был добавлен в систему. Автомобиль уже находится в системе.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show("Некорректные данные", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void comboBoxFuelType\_SelectionChanged(object sender, SelectionChangedEventArgs e) // обработчик изменение значения в comboBox

{

string selectedItem = comboBoxFuelType.SelectedItem.ToString();

if (selectedItem == "Бензин")

{

List<Fuel> fuelOctan = dbManager.GetFuelList("Бензин");

foreach(var fuel in fuelOctan)

{

comboBoxOctan.Items.Add(fuel.OctaneNumber);

}

}

else if (selectedItem == "Дизельное топливо")

{

List<Fuel> fuelOctan = dbManager.GetFuelList("Дизельное топливо");

foreach (var fuel in fuelOctan)

{

comboBoxOctan.Items.Add(fuel.OctaneNumber);

}

}

comboBoxOctan.SelectedIndex = 1;

}

private bool ValidValue()

{

if (TextBoxValid(textBoxName) & ValidFuel() & СonsumptionСheck() & LogicCheck())

{

return true;

}

return false;

}

private bool TextBoxValid(TextBox text)// проверка значения в textBox

{

if (text.Text == "")

{

text.BorderBrush = Brushes.Red;

return false;

}

else

{

text.BorderBrush = Brushes.Gray;

return true;

}

}

private bool ValidFuel() // проверка корректности выбора топлива

{

string selectedItem = IsValidComboBox(comboBoxFuelType);

if (selectedItem == "Бензин" || selectedItem == "Дизельное топливо")

{

if (comboBoxFuelType.SelectedIndex != -1 && textBoxFuelConsumptionGeneral.Text != "")

{

return true;

}

}

return false;

}

private bool СonsumptionСheck()

{

string selectedItem = IsValidComboBox(comboBoxFuelType);

double ConsumptionGeneral;

bool isNumericOne = IsValidDoubleInput(textBoxFuelConsumptionGeneral, 0, 100, out ConsumptionGeneral);

if (selectedItem == "Бензин" || selectedItem == "Дизельное топливо")

{

if (isNumericOne)

{

textBoxFuelConsumptionGeneral.BorderBrush = Brushes.Gray;

return true;

}

}

return false;

}

private bool LogicCheck() // логическая проверка значений в полях

{

string selectedItem = IsValidComboBox(comboBoxCarType);

int year, speed, place, minSeats = 2, maxSeats = 5;

bool IsYear, IsSpeed, IsSeats;

IsYear = IsValidIntInput(textBoxYear, 1980, 2024, out year);

IsSpeed = IsValidIntInput(textBoxMaxSpeed, 0, 500, out speed);

IsSeats = IsValidIntInput(textBoxPlace, 0, 11, out place);

if (selectedItem == "Фургон")

{

minSeats = 2;

maxSeats = 10;

}

else if (selectedItem == "Универсал" || selectedItem == "Внедорожник" || selectedItem == "Кроссовер" || selectedItem == "Минивэн")

{

minSeats = 5;

maxSeats = 8;

}

if (place >= minSeats && place <= maxSeats)

{

textBoxPlace.BorderBrush = Brushes.Gray;

if (IsYear && IsSpeed && IsSeats)

{

comboBoxCarType.BorderBrush = Brushes.Gray;

return true;

}

return false;

}

else

{

if (selectedItem == "Ошибка")

{

textBoxPlace.BorderBrush = Brushes.Red;

return false;

}

MessageBox.Show($"Ошибка количества мест. Требуется от {minSeats} до {maxSeats} мест для автомобиля типа {selectedItem}");

textBoxPlace.BorderBrush = Brushes.Red;

return false;

}

}

private bool IsValidIntInput(TextBox text, int min, int max, out int value) // проверка корректности введенных целочисленных значений

{

if (text.Text == "")

{

value = 0;

text.BorderBrush = Brushes.Red; // смена цвета границы элемента

return false;

}

bool isNumeric = int.TryParse(text.Text, out value);

if (isNumeric)

{

if (value > min && value < max)

{

text.BorderBrush = Brushes.Gray;

return true;

}

else

{

text.BorderBrush = Brushes.Red;

return false;

}

}

else

{

text.BorderBrush = Brushes.Red;

return false;

}

}

private bool IsValidDoubleInput(TextBox box, int min, int max, out double value)// проверка корректности введенных вещественное значений

{

if (box.Text == "")

{

value = 0;

box.BorderBrush = Brushes.Red;

return false;

}

bool isNumeric = double.TryParse(FixStr(box.Text), out value);

if (isNumeric)

{

if (value > min && value < max)

{

box.BorderBrush = Brushes.Gray;

return true;

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

private string IsValidComboBox(ComboBox box)// проверка корректности выбора значения в comboBox

{

if (box.SelectedIndex == -1)

{

return "Ошибка";

}

else

{

return box.SelectedItem.ToString();

}

}

private string FixStr(string input)

{

return input.Replace('.', ',');

}

private double DoubleNull(TextBox text)

{

if (text.Text != "")

{

return Convert.ToDouble(FixStr(text.Text));

}

else

{

return 0;

}

}

}

}

<Window x:Class="TripNode.UserForm.AddFuelForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Добавление топлива"

Height="260" MinHeight="260" MaxHeight="280"

Width="400" MinWidth="400" MaxWidth="280" >

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="13\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="51\*"/>

<RowDefinition Height="36\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<ComboBox x:Name="comboBoxFuel" FontSize="15" Margin="10,12,10,0" Height="27" VerticalAlignment="Top" Grid.Column="1" Grid.Row="1"/>

<Label Content="Вид топлива" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="1"/>

<TextBox x:Name="textBoxOctan" FontSize="15" Margin="10,11,10,0" Height="27" VerticalAlignment="Top" Grid.Row="2" Grid.Column="1"/>

<Label Content="Октановое число" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="2"/>

<TextBox x:Name="textBoxPrice" FontSize="15" Margin="10,0,0,0" Height="27" VerticalAlignment="Center" Grid.Row="3" Grid.Column="1" Width="145" HorizontalAlignment="Left"/>

<Label Content="Цена топлива" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="3"/>

<Button x:Name="ButtonAddFuel" Content="Добавить топиливо" Margin="18,0,18,0" Click="ButtonAddRoute\_Click" Grid.Row="4" Height="28" VerticalAlignment="Center" />

<Label Grid.Column="1" Content="Руб" HorizontalAlignment="Left" Margin="160,0,0,0" Grid.Row="3" VerticalAlignment="Center"/>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Media;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для AddFuelForm.xaml

/// </summary>

public partial class AddFuelForm : Window

{

public static readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

private readonly Database dbManager = new Database(connectionString);

List<string> fuel = new List<string>() { "Бензин", "Дизельное топливо" };

public event EventHandler UpdateValue;

public AddFuelForm()

{

InitializeComponent();

InitComboBox();

}

public void InitComboBox()

{

foreach(string fuelItem in fuel)

{

comboBoxFuel.Items.Add(fuelItem);

}

}

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

{

double fuelPrice;

if(ValidValue() & IsValidDoubleInput(textBoxPrice,0,20, out fuelPrice))

{

string fuelType = comboBoxFuel.SelectedItem.ToString();

string octanNumber = textBoxOctan.Text;

Fuel fuel = new Fuel(fuelType, octanNumber, fuelPrice);

if(dbManager.InsertFuel(fuel))

{

MessageBox.Show($"Топливо было добавлено в систему.", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show($"Топливо не было добавлено в систему. Автомобиль уже находится в системе.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show("Некорректные данные", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private bool IsValidDoubleInput(TextBox box, int min, int max, out double value)// проверка корректности введенных вещественное значений

{

if (box.Text == "")

{

value = 0;

box.BorderBrush = Brushes.Red;

return false;

}

bool isNumeric = double.TryParse(FixStr(box.Text), out value);

if (isNumeric)

{

if (value > min && value < max)

{

box.BorderBrush = Brushes.Gray;

return true;

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

private bool ValidValue()

{

if (TextBoxValid(textBoxPrice) & TextBoxValid(textBoxOctan) & ValidFuel())

{

return true;

}

return false;

}

private bool TextBoxValid(TextBox text)// проверка значения в textBox

{

if (text.Text == "")

{

text.BorderBrush = Brushes.Red;

return false;

}

else

{

text.BorderBrush = Brushes.Gray;

return true;

}

}

private bool ValidFuel() // проверка корректности выбора топлива

{

string selectedItem = IsValidComboBox(comboBoxFuel);

if (selectedItem == "Бензин" || selectedItem == "Дизельное топливо")

{

if (comboBoxFuel.SelectedIndex != -1)

{

return true;

}

}

return false;

}

private string IsValidComboBox(ComboBox box)// проверка корректности выбора значения в comboBox

{

if (box.SelectedIndex == -1)

{

return "Ошибка";

}

else

{

return box.SelectedItem.ToString();

}

}

private double DoubleNull(TextBox text)

{

if (text.Text != "")

{

return Convert.ToDouble(FixStr(text.Text));

}

else

{

return 0;

}

}

private string FixStr(string input)

{

return input.Replace('.', ',');

}

}

}

<Window x:Class="TripNode.UserForm.AddRouteForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Добавление маршрута"

Height="260" MinHeight="260" MaxHeight="280"

Width="400" MinWidth="400" MaxWidth="280" >

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="13\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="51\*"/>

<RowDefinition Height="36\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<TextBox x:Name="textBoxPointA" FontSize="15" Margin="10,12,10,0" Height="27" VerticalAlignment="Top" Grid.Column="1" Grid.Row="1"/>

<Label Content="Точка отправления" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="1"/>

<TextBox x:Name="textBoxPointB" FontSize="15" Margin="10,0,10,0" Height="27" VerticalAlignment="Center" Grid.Row="2" Grid.Column="1"/>

<Label Content="Точка назначения" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="2"/>

<TextBox x:Name="textBoxDistance" FontSize="15" Margin="10,0,0,0" Height="27" VerticalAlignment="Center" Grid.Row="3" Grid.Column="1" Width="145" HorizontalAlignment="Left"/>

<Label Content="Расстояние" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="3"/>

<Button x:Name="ButtonAddRoute" Content="Добавить маршрут" Margin="18,0,18,0" Click="ButtonAddRoute\_Click" Grid.Row="4" Height="28" VerticalAlignment="Center" />

<Label Grid.Column="1" Content="Км" HorizontalAlignment="Left" Margin="160,0,0,0" Grid.Row="3" VerticalAlignment="Center"/>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для AddRouteForm.xaml

/// </summary>

public partial class AddRouteForm : Window

{

private readonly Database dbManager;

public readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

public event EventHandler UpdateValue;

public AddRouteForm()

{

InitializeComponent();

dbManager = new Database(connectionString);

}

private void ButtonAddRoute\_Click(object sender, RoutedEventArgs e)//добавление маршурта

{

string pointA = textBoxPointA.Text.TrimEnd();

string pointB = textBoxPointB.Text.TrimEnd();

double distance = 0;

bool IsValidValue = IsValidDoubleInput(textBoxDistance, 0, 40000, out distance);

if ((TextBoxValid(textBoxPointA) & TextBoxValid(textBoxPointB) & IsValidValue) & (pointA != pointB))

{

Route route = new Route(pointA, pointB, distance);

if (dbManager.InsertRoute(route))

{

MessageBox.Show($"Маршрут {pointA} - {pointB} добавлен в систему.", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show($"Маршрут {pointA} - {pointB} не добавлен, так как уже есть в системе.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show("Ошибка в маршруте, проверьте маршрут и повторите попытку.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private bool IsValidDoubleInput(TextBox box, int min, int max, out double value)//проверка вещественных значений

{

if (box.Text == "")

{

value = 0;

box.BorderBrush = Brushes.Red;

return false;

}

bool isNumeric = double.TryParse(FixStr(box.Text), out value);

if (isNumeric)

{

if (value > min && value < max)

{

box.BorderBrush = Brushes.Gray;

return true;

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

private bool TextBoxValid(TextBox text)//проверка кооректносоти данных в textBox

{

if (text.Text == "")

{

text.BorderBrush = Brushes.Red;

return false;

}

else

{

text.BorderBrush = Brushes.Gray;

return true;

}

}

private string FixStr(string input)//замена . на ,

{

return input.Replace('.', ',');

}

}

}

<Window x:Class="TripNode.UserForm.EditCarForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Редактирование автомобиля"

Height="200" MinHeight="200" MaxHeight="200"

Width="400" MinWidth="400" MaxWidth="280">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="13\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Content="Автомобиль" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="1"/>

<Button x:Name="DeleteCarButton" Content="Удалить автомбиль" Grid.Row="2" VerticalAlignment="Center" Height="24" Grid.Column="1" Margin="10,0,5,0" Click="DeleteCarButton\_Click"/>

<ComboBox x:Name="comboBoxCar" Grid.Column="1" Grid.Row="1" VerticalAlignment="Center" Height="28" MouseDoubleClick="comboBoxCar\_MouseDoubleClick" Margin="10,0,10,0"/>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.ConstrainedExecution;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для EditCarForm.xaml

/// </summary>

public partial class EditCarForm : Window

{

private readonly Database dbManager;

public static readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

public static List<Car> carList = new List<Car>();

public event EventHandler UpdateValue;

public EditCarForm()

{

InitializeComponent();

dbManager = new Database(connectionString);

carList = dbManager.GetAllCars();

InitComboBox();

}

private void InitComboBox()

{

carList = dbManager.GetAllCars();

comboBoxCar.Items.Clear();

foreach (var car in carList)

{

comboBoxCar.Items.Add(car.name);

}

}

private void DeleteCarButton\_Click(object sender, RoutedEventArgs e)

{

if(comboBoxCar.SelectedIndex != -1)

{

Car car = carList[comboBoxCar.SelectedIndex];

if (dbManager.DeleteCar(car))

{

MessageBox.Show($"Автомобиль {car.name} удален","Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show($"Автомобиль {car.name} не удален", "Ошибка",MessageBoxButton.OK,MessageBoxImage.Error);

}

InitComboBox();

}

else

{

MessageBox.Show($"Выберите автомобиль, который хотите удалить", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void comboBoxCar\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

comboBoxCar.SelectedIndex = -1;

}

}

}

<Window x:Class="TripNode.UserForm.EditFuelForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Редактирование топлива"

Height="260" MinHeight="200" MaxHeight="280"

Width="400" MinWidth="400" MaxWidth="280">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="13\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="51\*"/>

<RowDefinition Height="36\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Content="Вид топилва" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="1"/>

<ComboBox x:Name="comboBoxFuel" Grid.Column="1" HorizontalAlignment="Center" Grid.Row="1" VerticalAlignment="Center" Width="180" Height="28" SelectionChanged="comboBoxFuel\_SelectionChanged"/>

<Label Content="Тип топлива" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="2"/>

<Label Content="Цена топлива" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="3"/>

<TextBox x:Name="textBoxPrice" FontSize="15" Height="27" VerticalAlignment="Top" Grid.Row="3" Grid.Column="1" Width="130" HorizontalAlignment="Left" Margin="10,12,0,0"/>

<Label Grid.Column="1" Content="Руб" HorizontalAlignment="Left" Margin="151,13,0,0" Grid.Row="3" VerticalAlignment="Top"/>

<ComboBox x:Name="comboBoxOctane" Grid.Column="1" HorizontalAlignment="Center" Grid.Row="2" VerticalAlignment="Top" Width="180" Height="28" Margin="0,8,0,0" SelectionChanged="comboBoxOctane\_SelectionChanged"/>

<Button x:Name="EditFuelButton" Content="Редактировать топливо" Margin="10,10,10,13" Grid.Row="4" Grid.Column="1" Click="EditFuelButton\_Click"/>

<Button x:Name="DeleteButton" Content="Удалить топилво" Margin="10,10,10,13" Grid.Row="4" Click="DeleteButton\_Click"/>

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.ConstrainedExecution;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для EditFuelForm.xaml

/// </summary>

public partial class EditFuelForm : Window

{

public static readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

private readonly Database dbManager = new Database(connectionString);

public static List<Fuel> fuelList = new List<Fuel>();

List<string> fuel = new List<string>() { "Бензин", "Дизельное топливо" };

public event EventHandler UpdateValue;

public EditFuelForm()

{

InitializeComponent();

InitComboBox();

}

private void InitComboBox()

{

comboBoxFuel.Items.Clear();

foreach (string fuelItem in fuel)

{

comboBoxFuel.Items.Add(fuelItem);

}

comboBoxOctane.Items.Clear();

textBoxPrice.Text = "";

}

private void DeleteButton\_Click(object sender, RoutedEventArgs e)

{

if(comboBoxFuel.SelectedIndex != -1 && comboBoxOctane.SelectedIndex != -1)

{

string fuel = comboBoxFuel.SelectedItem.ToString();

string octaneNumber = comboBoxOctane.SelectedItem.ToString();

if (dbManager.DeleteFuel(fuel, octaneNumber))

{

MessageBox.Show($"Топливо {fuel} - {octaneNumber} удалено", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

InitComboBox();

}

else

{

MessageBox.Show($"Топливо {fuel} - {octaneNumber} не удалено", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show($"Выберите топливо для удаления", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void EditFuelButton\_Click(object sender, RoutedEventArgs e)

{

double fuelPrice;

if (comboBoxFuel.SelectedIndex != -1 && comboBoxOctane.SelectedIndex != -1 & IsValidDoubleInput(textBoxPrice, 0, 20, out fuelPrice))

{

string fuel = comboBoxFuel.SelectedItem.ToString();

string octaneNumber = comboBoxOctane.SelectedItem.ToString();

Fuel newFuel = new Fuel(fuel, octaneNumber, fuelPrice);

if (dbManager.UpdateFuelPrice(newFuel))

{

MessageBox.Show($"Топливо {fuel} - {octaneNumber} обновлено", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

InitComboBox();

}

else

{

MessageBox.Show($"Топливо {fuel} - {octaneNumber} не обновлено", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show($"Введите корректные значения", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private bool IsValidDoubleInput(TextBox box, int min, int max, out double value)// проверка корректности введенных вещественное значений

{

if (box.Text == "")

{

value = 0;

box.BorderBrush = Brushes.Red;

return false;

}

bool isNumeric = double.TryParse(FixStr(box.Text), out value);

if (isNumeric)

{

if (value > min && value < max)

{

box.BorderBrush = Brushes.Gray;

return true;

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

private string FixStr(string input)//замена . на ,

{

return input.Replace('.', ',');

}

private void comboBoxFuel\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if(comboBoxFuel.SelectedIndex != -1)

{

fuelList = dbManager.GetFuelList(comboBoxFuel.SelectedItem.ToString());

foreach( Fuel fuel in fuelList)

{

comboBoxOctane.Items.Add(fuel.OctaneNumber);

}

}

else

{

comboBoxOctane.SelectedIndex = -1;

}

}

private void comboBoxOctane\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if(comboBoxOctane.SelectedIndex != -1)

{

textBoxPrice.Text = fuelList[comboBoxFuel.SelectedIndex].FuelPrice.ToString();

}

else

{

textBoxPrice.Text = "";

}

}

}

}

<Window x:Class="TripNode.UserForm.EditRouteForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Редактирование маршрута"

Height="260" MinHeight="260" MaxHeight="280"

Width="400" MinWidth="400" MaxWidth="280">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="13\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="48\*"/>

<RowDefinition Height="51\*"/>

<RowDefinition Height="36\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Label Content="Машрут" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="1"/>

<TextBox x:Name="textBoxDistance" FontSize="15" Margin="10,13,0,0" Height="27" VerticalAlignment="Top" Grid.Row="2" Grid.Column="1" Width="145" HorizontalAlignment="Left"/>

<Label Content="Расстояние" Margin="24,0,24,0" Height="29" VerticalAlignment="Center" FontSize="15" Grid.Row="2"/>

<Button x:Name="EditRouteButton" Content="Редактировать маршрут" Margin="10,10,10,10" Grid.Row="3" Grid.Column="1" Click="EditRouteButton\_Click" />

<Button x:Name="DeleteRouteButton" Content="Удалить маршрут" Grid.Row="4" Grid.Column="1" Click="DeleteRouteButton\_Click" RenderTransformOrigin="0.5,-1.571" Margin="10,12,10,11"/>

<ComboBox x:Name="comboBoxRoute" Grid.Column="1" HorizontalAlignment="Center" Grid.Row="1" VerticalAlignment="Center" Width="180" Height="28" MouseDoubleClick="comboBoxPointOne\_MouseDoubleClick" SelectionChanged="comboBoxRoute\_SelectionChanged"/>

<Label Grid.Column="1" Content="Км" HorizontalAlignment="Left" Margin="160,13,0,0" Grid.Row="2" VerticalAlignment="Top"/>

<Button x:Name="FavoriteRouteButton" Content="Добавить в избранное" Margin="10,10,10,10" Grid.Row="3" Click="FavoriteRouteButton\_Click"/>

<Button x:Name="DeleteFavoriteRouteButton" Content="Удалить из избранного" Margin="10,12,10,11" Grid.Row="4" Click="DeleteFavoriteRouteButton\_Click" />

</Grid>

</Window>

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using TripNode.Classes;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для EditRouteForm.xaml

/// </summary>

public partial class EditRouteForm : Window

{

int IdUser;

private readonly Database dbManager;

public static readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

private static List<Route> routeList = new List<Route>();

public event EventHandler UpdateValue;

public EditRouteForm(int idUser)

{

InitializeComponent();

dbManager = new Database(connectionString);

routeList = dbManager.GetAllRoutes();

InitComboBox();

IdUser = idUser;

}

private void InitComboBox()

{

routeList = dbManager.GetAllRoutes();

comboBoxRoute.Items.Clear();

foreach (var route in routeList)

{

comboBoxRoute.Items.Add($"{route.cityOne} - {route.cityTwo}");

}

}

private void EditRouteButton\_Click(object sender, RoutedEventArgs e)

{

if(comboBoxRoute.SelectedIndex != -1)

{

if (Convert.ToDouble(textBoxDistance.Text) > 0 && Convert.ToDouble(textBoxDistance.Text) < 10000 && comboBoxRoute.SelectedIndex != 1)

{

Route newRoute = routeList[comboBoxRoute.SelectedIndex];

newRoute.distance = Convert.ToDouble(textBoxDistance.Text);

if (dbManager.UpdateRoute(newRoute))

{

MessageBox.Show($"Маршрут {newRoute.cityOne} - {newRoute.cityTwo} обновлен","Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show($"Маршрут {newRoute.cityOne} - {newRoute.cityTwo} не обновлен","Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show($"Ошибка, проверьте введенные данные","Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

InitComboBox();

}

else

{

MessageBox.Show("Выберите маршрут для редактирования", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void DeleteRouteButton\_Click(object sender, RoutedEventArgs e)

{

if (comboBoxRoute.SelectedIndex != -1)

{

Route route = routeList[comboBoxRoute.SelectedIndex];

string pointA = route.cityOne;

string pointB = route.cityTwo;

int idRoute = dbManager.GetIdRoute(pointA, pointB);

bool favoriteCheck = dbManager.DeleteFavoriteRoute(IdUser, idRoute);

if (dbManager.DeleteRoute(route))

{

if (favoriteCheck)

{

MessageBox.Show($"Маршрут {route.cityOne} - {route.cityTwo} удален и так же удалени из избранных", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show($"Маршрут {route.cityOne} - {route.cityTwo} удален", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

}

else

{

MessageBox.Show($"Маршрут {route.cityOne} - {route.cityTwo} не удален", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

InitComboBox();

}

else

{

MessageBox.Show("Выберите маршрут для удалени", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void comboBoxPointOne\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

comboBoxRoute.SelectedIndex = -1;

}

private void comboBoxRoute\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (comboBoxRoute.SelectedIndex != -1)

{

int selectedIndex = comboBoxRoute.SelectedIndex;

textBoxDistance.Text = routeList[selectedIndex].distance.ToString();

}

else

{

textBoxDistance.Text = string.Empty;

}

}

private void FavoriteRouteButton\_Click(object sender, RoutedEventArgs e)

{

if(comboBoxRoute.SelectedIndex != -1)

{

Route route = routeList[comboBoxRoute.SelectedIndex];

string pointA = route.cityOne;

string pointB = route.cityTwo;

int idRoute = dbManager.GetIdRoute(pointA, pointB);

if(dbManager.InsertFavoriteRoute(IdUser, idRoute))

{

MessageBox.Show("Маршрут добавлен в избранное","Уведомление",MessageBoxButton.OK,MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show("Маршрут не добавлен в избранное","Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show("Выберите маршрут для добавления", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private void DeleteFavoriteRouteButton\_Click(object sender, RoutedEventArgs e)

{

if (comboBoxRoute.SelectedIndex != -1)

{

Route route = routeList[comboBoxRoute.SelectedIndex];

string pointA = route.cityOne;

string pointB = route.cityTwo;

int idRoute = dbManager.GetIdRoute(pointA, pointB);

if (dbManager.DeleteFavoriteRoute(IdUser, idRoute))

{

MessageBox.Show("Маршрут удален из избранного", "Уведомление", MessageBoxButton.OK, MessageBoxImage.Information);

UpdateValue?.Invoke(this, EventArgs.Empty);

}

else

{

MessageBox.Show("Маршрут не удален из избранного", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

MessageBox.Show("Выберите маршрут для удаления", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

<Window x:Class="TripNode.UserForm.History"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="История поездок"

Height="450" MaxHeight="450" MinHeight="450"

Width="800" MaxWidth="800" MinWidth="800">

<Grid>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="3\*"/>

<ColumnDefinition Width="\*"/>

</Grid.ColumnDefinitions>

<DataGrid Name="dataGrid" ItemsSource="{Binding TripDetailsList}" Grid.Column="0">

</DataGrid>

<StackPanel Grid.Column="1">

<Label Content="Фильтр" HorizontalContentAlignment="Center" FontSize="15" FontWeight="Bold"/>

<Label Content="От" HorizontalContentAlignment="Center"/>

<DatePicker Name="DateOne" Width="120" HorizontalAlignment="Center" Margin="0,10,0,0" VerticalAlignment="Top" />

<Label Content="До" HorizontalContentAlignment="Center"/>

<DatePicker Name="DateTwo" Width="120" HorizontalAlignment="Center" Margin="0,10,0,0" VerticalAlignment="Top" />

<Button Content="Применить фильтр" Width="120" Margin="10" Click="Button\_Click"/>

<Label Content="" x:Name="FullPrice" Margin="0 200 0 0" Width="200"/>

</StackPanel>

</Grid>

</Window>

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using TripNode.Classes;

using Xceed.Document.NET;

using Xceed.Words.NET;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для History.xaml

/// </summary>

public partial class History : Window

{

private readonly Database dbManager;

private int idUser;

public History(int idUser)

{

InitializeComponent();

this.idUser = idUser;

string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

dbManager = new Database(connectionString);

List<TripInfo> trips = dbManager.GetTripsByUserId(idUser);

dataGrid.ItemsSource = trips;

dataGrid.AutoGenerateColumns = false;

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Имя пользователя", Binding = new Binding("UserLogin") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Название машины", Binding = new Binding("CarName") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Точка А маршрута", Binding = new Binding("RoutePointA") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Точка B маршрута", Binding = new Binding("RoutePointB") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Пройденное расстояние", Binding = new Binding("Distance") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Расход топлива", Binding = new Binding("Consumption") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Средний расход", Binding = new Binding("AverageConsumption") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Дата поездки", Binding = new Binding("TripDate") { StringFormat = "dd.MM.yyyy" } });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Цена", Binding = new Binding("Price") });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Начало поездки", Binding = new Binding("TimeStart") { StringFormat = "HH.mm.ss" } });

dataGrid.Columns.Add(new DataGridTextColumn() { Header = "Окончание поездки", Binding = new Binding("TimeFinish") { StringFormat = "HH.mm.ss" } });

this.FullPrice.Content = $"Затраты за месяц {dbManager.GetTotalTripPricesForUser(idUser).ToString()}";

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

this.FullPrice.Content = "";

DateTime? startDate = DateOne.SelectedDate;

DateTime? endDate = DateTwo.SelectedDate;

if (startDate != null && endDate != null)

{

if (startDate <= endDate)

{

// Обработка, если startDate меньше или равен endDate

List<TripInfo> data = dbManager.GetTripsBetweenDate(idUser, startDate.Value, endDate.Value);

dataGrid.ItemsSource = data;

}

else

{

// Обработка, если startDate больше endDate

MessageBox.Show("Дата начала периода не может быть больше даты конца периода", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else if (startDate != null && endDate == null)

{

// Выполнить скрипт для работы с выбранной только первой датой startDate

List<TripInfo> data = dbManager.GetTripsBetweenDate(idUser, startDate.Value, null);

dataGrid.ItemsSource = data;

}

else if (startDate == null && endDate != null)

{

// Выполнить скрипт для работы с выбранной только второй датой endDate

List<TripInfo> data = dbManager.GetTripsBetweenDate(idUser, null, endDate.Value);

dataGrid.ItemsSource = data;

}

else if (startDate == null && endDate == null)

{

// Обе даты не заполнены

List<TripInfo> dateFilterData = dbManager.GetTripsBetweenDate(idUser, null, null);

dataGrid.ItemsSource = dateFilterData;

}

}

}

}

<Window x:Class="TripNode.UserForm.ReportForm"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:TripNode.UserForm"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Отчеты"

Height="300" MaxHeight="300" MinHeight="300"

Width="400" MaxWidth="400" MinWidth="400">

<Grid>

<StackPanel HorizontalAlignment="Center" VerticalAlignment="Center">

<TextBlock HorizontalAlignment="Center" TextWrapping="Wrap" Text="Создание отчета" VerticalAlignment="Center" FontSize="20" Margin="10"/>

<Label Content="Начало отчета" HorizontalContentAlignment="Center"/>

<DatePicker Name="DateOne" Margin="10"/>

<Label Content="Конец отчета" HorizontalContentAlignment="Center"/>

<DatePicker Name="DateTwo" Margin="10"/>

<Button Content="Создать отчет" Margin="10" Click="Button\_Click" />

</StackPanel>

</Grid>

</Window>

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Markup;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Shapes;

using TripNode.Classes;

using Xceed.Document.NET;

using Xceed.Words.NET;

namespace TripNode.UserForm

{

/// <summary>

/// Логика взаимодействия для ReportForm.xaml

/// </summary>

public partial class ReportForm : Window

{

private readonly Database dbManager;

int idUser;

string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

public ReportForm(int idUser)

{

InitializeComponent();

dbManager = new Database(connectionString);

this.idUser = idUser;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

DateTime? startDate = DateOne.SelectedDate;

DateTime? endDate = DateTwo.SelectedDate;

if (startDate != null && endDate != null)

{

if (startDate <= endDate)

{

// Обработка, если startDate меньше или равен endDate

List<TripInfo> data = dbManager.GetTripsBetweenDate(idUser, startDate.Value, endDate.Value);

GenerateReport(data);

}

else

{

// Обработка, если startDate больше endDate

MessageBox.Show("Дата начала периода не может быть больше даты конца периода", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else if (startDate != null && endDate == null)

{

// Выполнить скрипт для работы с выбранной только первой датой startDate

List<TripInfo> data = dbManager.GetTripsBetweenDate(idUser, startDate.Value, null);

GenerateReport(data);

}

else if (startDate == null && endDate != null)

{

// Выполнить скрипт для работы с выбранной только второй датой endDate

List<TripInfo> data = dbManager.GetTripsBetweenDate(idUser, null, endDate.Value);

GenerateReport(data);

}

else if (startDate == null && endDate == null)

{

// Обе даты не заполнены

List<TripInfo> dateFilterData = dbManager.GetTripsBetweenDate(idUser, null, null);

GenerateReport(dateFilterData);

}

}

public void GenerateReport(List<TripInfo> trips)

{

// Создание диалогового окна сохранения файла

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Документ Word (\*.docx)|\*.docx";

saveFileDialog.FileName = "TripReport";

// Если пользователь выбрал место сохранения файла и нажал "Сохранить"

if (saveFileDialog.ShowDialog() == true)

{

// Создание документа Word

using (DocX document = DocX.Create(saveFileDialog.FileName))

{

// Добавление заголовка

document.InsertParagraph("Отчет о поездках").FontSize(20d).Bold().Alignment = Alignment.center;

// Создание переменной для хранения суммы расходов за месяц

decimal totalConsumption = 0;

int count = 1;

// Добавление информации о каждой поездке и вычисление суммы расходов за месяц

foreach (TripInfo trip in trips)

{

// Добавление информации о поездке

document.InsertParagraph($"Поекздка номер: {count}");

document.InsertParagraph($"Имя пользователя: {trip.UserLogin}");

document.InsertParagraph($"Название машины: {trip.CarName}");

document.InsertParagraph($"Точка A маршрута: {trip.RoutePointA}");

document.InsertParagraph($"Точка B маршрута: {trip.RoutePointB}");

document.InsertParagraph($"Пройденное расстояние: {trip.Distance}");

document.InsertParagraph($"Расход топлива: {trip.Consumption}");

document.InsertParagraph($"Средний расход: {trip.AverageConsumption}");

document.InsertParagraph($"Дата поездки: {trip.TripDate.ToString("dd.MM.yyyy")}");

document.InsertParagraph($"Цена: {trip.Price}");

document.InsertParagraph($"Начало поездки: {trip.TimeStart.ToString("HH.mm.ss")}");

document.InsertParagraph($"Окончание поездки: {trip.TimeFinish.ToString("HH.mm.ss")}");

document.InsertParagraph($"");

totalConsumption += trip.Price;

count++;

}

// Добавление строки с суммой расходов за месяц

document.InsertParagraph($"Общий расход за текущий период: {totalConsumption}");

document.Save();

MessageBox.Show("Документ сохранен");

}

}

}

}

}

<Application x:Class="TripNode.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:TripNode"

StartupUri="UserForm\AuthForm.xaml">

<Application.Resources>

</Application.Resources>

</Application>

<Window x:Class="TripNode.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:xctk="http://schemas.xceed.com/wpf/xaml/toolkit"

xmlns:local="clr-namespace:TripNode"

mc:Ignorable="d"

Icon="/Icon/car.ico"

Title="Главная страница"

Height="400" MinHeight="400" MaxHeight="450"

Width="855" MinWidth="855" MaxWidth="855"

WindowStartupLocation="CenterScreen" >

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="30\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="49\*"/>

<RowDefinition Height="47\*"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="210"/>

<ColumnDefinition Width="145"/>

<ColumnDefinition Width="95"/>

<ColumnDefinition Width="165"/>

<ColumnDefinition Width="160"/>

<ColumnDefinition Width="80"/>

<ColumnDefinition Width="0\*"/>

</Grid.ColumnDefinitions>

<Label Content="Расстояние" Grid.Column="1" Grid.Row="1" FontSize="15" Height="27" Margin="10,0,10,0" VerticalAlignment="Center"/>

<TextBox x:Name="textBoxDistance" Grid.Column="2" Grid.Row="1" FontSize="15" Height="27" Margin="10,0,10,0" VerticalAlignment="Center" IsEnabled="False"/>

<Label Grid.Column="3" Content="Км" Grid.Row="1" FontSize="15" Height="27" Margin="28,0,22,0" VerticalAlignment="Center" HorizontalContentAlignment="Center"/>

<Label Content="Средняя скорость" Grid.Column="1" Grid.Row="2" FontSize="15" Height="34" Margin="10,11,10,0" VerticalAlignment="Top"/>

<TextBox x:Name="textBoxAverSpeed" Grid.Column="2" Grid.Row="2" FontSize="15" Height="27" Margin="10,0,10,0" VerticalAlignment="Center"/>

<Label Grid.Column="3" Content="Км/Ч" Grid.Row="2" FontSize="15" Height="27" Margin="19,0,18,0" VerticalAlignment="Center" HorizontalContentAlignment="Center"/>

<Label Content="Примерное время в пути" Grid.Column="1" Grid.Row="3" FontSize="15" Height="33" Margin="10,0,10,0" VerticalAlignment="Center"/>

<TextBox x:Name="textBoxTime" Grid.Column="2" Grid.Row="3" FontSize="15" Height="27" Margin="11,0,10,0" VerticalAlignment="Center" IsReadOnly="True"/>

<Label Grid.Column="3" Content="Часов" Grid.Row="3" FontSize="15" Height="27" Margin="17,0,10,0" VerticalAlignment="Center" HorizontalContentAlignment="Center"/>

<Label Content="Тип топлива" Grid.Column="1" Grid.Row="4" FontSize="15" Height="27" Margin="10,0,10,0" VerticalAlignment="Center"/>

<ComboBox x:Name="comboBoxFuelType" Grid.Column="2" Grid.Row="4" FontSize="15" Height="27" Margin="10,0,10,0" VerticalAlignment="Center" IsEnabled="False"/>

<Label Content="Расход топлива на 100 км" Grid.Column="1" Grid.Row="5" FontSize="15" Height="27" Margin="10,9,10,0" VerticalAlignment="Top"/>

<TextBox x:Name="textBoxConsumption" Grid.Column="2" Grid.Row="5" FontSize="15" Height="27" Margin="10,9,10,0" VerticalAlignment="Top" IsEnabled="False"/>

<Label x:Name="LabelConsumption" Grid.Column="3" Content="Литров" Grid.Row="5" FontSize="15" Height="29" Margin="5,7,5,0" VerticalAlignment="Top" HorizontalContentAlignment="Center"/>

<Label Content="Итоговая сумма" Grid.Column="1" Grid.Row="6" FontSize="15" Height="30" Margin="10,0,10,0" VerticalAlignment="Center"/>

<TextBox x:Name="textBoxPrice" Grid.Column="2" Grid.Row="6" FontSize="15" Height="27" Margin="10,10,10,0" VerticalAlignment="Top" IsReadOnly="True"/>

<Label x:Name="LabelConsumption\_Copy" Grid.Column="3" Content="Рублей" Grid.Row="6" FontSize="15" Height="29" Margin="5,0,5,0" VerticalAlignment="Center" HorizontalContentAlignment="Center"/>

<Button x:Name="ButtonCalculate" Grid.Column="1" Grid.Row="7" Content="Рассчитать" Margin="10,10,10,10" Click="ButtonCalculate\_Click" />

<Label Content="Автомобиль" Grid.Column="4" Grid.Row="1" FontSize="15" Height="27" Margin="15,0,15,0" VerticalAlignment="Center"/>

<ComboBox x:Name="comboBoxCar" Grid.Column="5" Grid.Row="1" FontSize="15" Height="27" Margin="18,0,18,0" VerticalAlignment="Center" SelectionChanged="comboBoxCar\_SelectionChanged" MouseDoubleClick="comboBoxCar\_MouseDoubleClick"/>

<Label Content="Кол-во Пассажиров" Margin="15,10,0,0" Grid.Column="4" Grid.Row="2" VerticalAlignment="Top" FontSize="15"/>

<TextBox x:Name="textBoxPeople" Grid.Column="5" Grid.Row="2" FontSize="15" Height="27" Margin="18,0,18,0" VerticalAlignment="Center"/>

<Label Content="Маршрут" Grid.Column="4" Grid.Row="3" FontSize="15" Height="29" Margin="15,13,5,0" VerticalAlignment="Top"/>

<ComboBox x:Name="comboBoxRoute" Grid.Column="5" Grid.Row="3" FontSize="15" Height="27" Margin="18,0,18,0" VerticalAlignment="Center" SelectionChanged="comboBoxPointOne\_SelectionChanged" MouseDoubleClick="comboBoxPointOne\_MouseDoubleClick"/>

<Label Content="Цена топлива" Grid.Column="4" Grid.Row="4" FontSize="15" Height="29" Margin="15,14,5,0" VerticalAlignment="Top"/>

<TextBox x:Name="textBoxFuelPrice" Grid.Column="5" Grid.Row="4" FontSize="15" Height="27" Margin="18,0,18,0" VerticalAlignment="Center" IsEnabled="false"/>

<Label Grid.Column="6" Content="Рублей" Grid.Row="4" FontSize="15" Width="64" Margin="0,13,0,0" VerticalAlignment="Top" HorizontalAlignment="Left"/>

<Menu Grid.ColumnSpan="7">

<MenuItem Header="Автомобили" Width="auto" Height="29" Margin="10,0,0,0" BorderThickness="0">

<MenuItem Header="Добавить автомобиль" Click="MenuItem\_ClickCar"/>

<MenuItem Header="Редактировать автомобиль" Click="MenuItem\_ClickCar2"/>

</MenuItem>

<MenuItem Header="Маршрут" BorderThickness="0">

<MenuItem Header="Добавить маршрут" Click="MenuItem\_Click\_1"/>

<MenuItem Header="Редактировать маршрут" Click="MenuItem\_Click\_2"/>

</MenuItem>

<MenuItem Header="Топливо" Width="70" Height="29" Margin="10,0,0,0">

<MenuItem Header="Добавить топливо" Click="MenuItem\_Click\_4"/>

<MenuItem Header="Редактировать топливо" Click="MenuItem\_Click\_5"/>

</MenuItem>

<MenuItem Header="История" Width="70" Height="29" Margin="10,0,0,0" Click="MenuItem\_ClickHistory"/>

<MenuItem Header="Отчет" Click="MenuItem\_Click\_3"/>

<MenuItem Header="Очистить" Width="70" Height="29" Margin="10,0,0,0" Click="MenuItem\_Click"/>

</Menu>

<Label Content="Использовано топлива" Grid.Column="4" Grid.Row="5" FontSize="15" Height="29" Margin="0,8,147,0" VerticalAlignment="Top" Grid.ColumnSpan="2"/>

<TextBox x:Name="textBoxUsedFuel" Grid.Column="5" Grid.Row="5" FontSize="15" Height="27" Margin="18,9,18,0" VerticalAlignment="Top" IsReadOnly="True"/>

<Label x:Name="LabelLitr" Grid.Column="6" Content="Литров" Grid.Row="5" FontSize="15" Width="64" VerticalAlignment="Top" HorizontalAlignment="Left" Margin="0,8,0,0"/>

<Label Content="Дата отправки" Grid.Column="4" Grid.Row="6" FontSize="15" Height="29" Margin="10,8,10,0" VerticalAlignment="Top"/>

<DatePicker x:Name="DataPickerFirstData" Grid.Column="5" Grid.Row="6" Width="124" HorizontalAlignment="Center" VerticalAlignment="Top" Margin="0,11,0,0"/>

<Label Content="Время отправки" Grid.Column="4" Grid.Row="7" FontSize="15" Height="29" Margin="15,12,5,0" VerticalAlignment="Top"/>

<xctk:TimePicker Name="timePicker" Grid.Column="5" Grid.Row="7" Margin="18,10,18,8" />

</Grid>

</Window>

using Microsoft.Win32;

using System;

using System.Collections.Generic;

using System.Data.OleDb;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Input;

using System.Windows.Media;

using TripNode.Classes;

using TripNode.UserForm;

using Xceed.Document.NET;

using Xceed.Words.NET;

using Xceed.Wpf.Toolkit;

namespace TripNode

{

/// <summary>

/// Логика взаимодействия для MainWindow.xaml

/// </summary>

///

public partial class MainWindow : Window

{

public static readonly string connectionString = "Data Source=(LocalDB)\\MSSQLLocalDB;AttachDbFilename=|DataDirectory|\\CarDB.mdf;Integrated Security=True;";

private readonly Database dbManager = new Database(connectionString);

int idUser;

List<Car> carList = new List<Car>();

List<Route> routes = new List<Route>();

List<FavoriteRoute> favoriteRoutes = new List<FavoriteRoute>();

List<string> fuelList = new List<string>() { "Бензин", "Дизельное топливо" };

Car selectCar = new Car();

Route selectRoute = new Route();

List<Fuel> fuelPrice = new List<Fuel>();

public MainWindow(int idUser)

{

InitializeComponent();

this.idUser = idUser;

InitComboBox();

}

public void InitComboBox()

{

carList = dbManager.GetAllCars();

routes = dbManager.GetAllRoutes();

favoriteRoutes = dbManager.GetFavoriteRoutes(idUser);

comboBoxCar.Items.Clear();

foreach (var carItem in carList)

{

comboBoxCar.Items.Add(carItem.name);

}

comboBoxFuelType.Items.Clear();

foreach (string fuelItem in fuelList)

{

comboBoxFuelType.Items.Add(fuelItem);

}

comboBoxRoute.Items.Clear();

foreach (Route route in routes)

{

string itemText = $"{route.cityOne} - {route.cityTwo}";

bool isFavorite = IsFavoriteRoute(route);

// Создаем элемент с пользовательским форматированием

TextBlock textBlock = new TextBlock();

textBlock.Text = itemText;

if (isFavorite)

{

textBlock.Foreground = Brushes.Green;

}

comboBoxRoute.Items.Add(textBlock);

}

}

private bool IsFavoriteRoute(Route route)

{

foreach (var favoriteRoute in favoriteRoutes)

{

if (favoriteRoute.PointA == route.cityOne && favoriteRoute.PointB == route.cityTwo)

{

return true;

}

}

return false;

}

private void MenuItem\_Click(object sender, RoutedEventArgs e)

{

textBoxDistance.Clear();

textBoxAverSpeed.Clear();

textBoxTime.Clear();

textBoxConsumption.Clear();

textBoxPrice.Clear();

textBoxFuelPrice.Clear();

textBoxUsedFuel.Clear();

textBoxDistance.BorderBrush = Brushes.Gray;

textBoxFuelPrice.BorderBrush = Brushes.Gray;

textBoxConsumption.BorderBrush = Brushes.Gray;

comboBoxFuelType.SelectedIndex = -1;

comboBoxCar.SelectedIndex = -1;

comboBoxRoute.SelectedIndex = -1;

}

private void comboBoxCar\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (comboBoxCar.SelectedIndex != -1)

{

selectCar = dbManager.GetCarData(comboBoxCar.Items[comboBoxCar.SelectedIndex].ToString());

comboBoxFuelType.SelectedIndex = fuelList.IndexOf(selectCar.fuel);

textBoxConsumption.Text = selectCar.fuelConsumptionGeneral.ToString();

fuelPrice = dbManager.GetFuelList(comboBoxFuelType.SelectedItem.ToString());

textBoxFuelPrice.Text = fuelPrice.FirstOrDefault(fuel => fuel.FuelType == comboBoxFuelType.SelectedItem.ToString() && fuel.OctaneNumber == selectCar.fuelOctan)?.FuelPrice.ToString() ?? "0";

comboBoxFuelType.IsEnabled = false;

textBoxConsumption.IsEnabled = false;

textBoxFuelPrice.IsEnabled = false;

}

else

{

comboBoxFuelType.SelectedIndex = -1;

comboBoxFuelType.IsEnabled = false;

textBoxConsumption.IsEnabled = false;

textBoxFuelPrice.IsEnabled = false;

textBoxFuelPrice.Clear();

textBoxConsumption.Clear();

}

}

private void comboBoxCar\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

comboBoxCar.SelectedIndex = -1;

selectCar = null;

}

private void comboBoxPointOne\_MouseDoubleClick(object sender, MouseButtonEventArgs e)

{

comboBoxRoute.SelectedIndex = -1;

selectRoute = null;

}

private void comboBoxPointOne\_SelectionChanged(object sender, SelectionChangedEventArgs e)

{

if (comboBoxRoute.SelectedIndex != -1)

{

textBoxDistance.Text = routes[comboBoxRoute.SelectedIndex].distance.ToString();

textBoxDistance.IsEnabled = false;

selectRoute = routes[comboBoxRoute.SelectedIndex];

}

else

{

textBoxDistance.IsEnabled = false;

}

}

private void ButtonCalculate\_Click(object sender, RoutedEventArgs e)

{

//Проверяем валидность введенных значений

if (ValidValue())

{

// Получаем значения из комбо-боксов

string car = SetValue(comboBoxCar);

string route = SetValue(comboBoxRoute);

// Получаем выбранный тип топлива

string fuelType = comboBoxFuelType.SelectedItem.ToString();

// Получаем числовые значения из текстовых полей

double distance = Convert.ToDouble(this.textBoxDistance.Text);

double fuelPrice = Convert.ToDouble(this.textBoxFuelPrice.Text);

double consumption = Convert.ToDouble(this.textBoxConsumption.Text);

double averageSpeed = Convert.ToDouble(this.textBoxAverSpeed.Text);

int people = Convert.ToInt32(this.textBoxPeople.Text);

// Проверяем условие и увеличиваем расход топлива, если средняя скорость превышает 140 км/ч

if (averageSpeed > 140 && car == "Неизвестно")

{

consumption += 1;

}

// Выполняем необходимые расчеты

double usedFuel = Math.Round((distance \* consumption) / 100.0,2);

double result = distance / averageSpeed;

double fullPrice = Math.Round((distance / 100) \* fuelPrice \* consumption, 2);

// Получаем выбранную дату из DatePicker

DateTime dateOne = (DateTime)DataPickerFirstData.SelectedDate;

DateTime? selectedTime = timePicker.Value;

DateTime startDate;

string date;

date = dateOne.ToString("dd.MM.yyyy");

if (selectedTime.HasValue)

{

startDate = dateOne.Date + selectedTime.Value.TimeOfDay;

}

else

{

System.Windows.MessageBox.Show("Выберите время отправки", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

TimeSpan timeSpan = TimeSpan.FromHours(result);

DateTime endTime = startDate;

endTime = endTime.Add(timeSpan);

// Заполняем текстовые поля с расчетами

textBoxUsedFuel.Text = usedFuel.ToString();

textBoxUsedFuel.BorderBrush = Brushes.Green;

textBoxTime.Text = Math.Round(result, 2).ToString();

textBoxTime.BorderBrush = Brushes.Green;

textBoxPrice.Text = fullPrice.ToString();

textBoxPrice.BorderBrush = Brushes.Green;

// Получаем идентификаторы пользователя, автомобиля и маршрута из базы данных

Car selectCar = carList[comboBoxCar.SelectedIndex];

int idCar = dbManager.GetIdCar(dbManager.GetCarData(selectCar.name));

Route selectRoute = routes[comboBoxRoute.SelectedIndex];

int idRoute = dbManager.GetIdRoute(selectRoute);

// Создаем объект поездки

Trips trip = new Trips(idUser, idCar, idRoute, distance, usedFuel, selectCar.fuelConsumptionGeneral, date, fullPrice, startDate, endTime);

// Добавляем поездку в базу данных

if (dbManager.InsertTrip(trip))

{

// Выводим сообщение с подтверждением

var res = System.Windows.MessageBox.Show("Хотите сохранить данную поездку в .docx?", "Подтверждение", MessageBoxButton.YesNo, MessageBoxImage.Question);

// Проверка выбора пользователя

if (res == MessageBoxResult.Yes)

{

// Если пользователь выбрал "Да", вызываем функцию для создания документа

GenerateReport(trip,selectCar,selectRoute);

}

}

else

{

// Выводим сообщение об ошибке, если поездка не была оформлена

System.Windows.MessageBox.Show("Поездка не оформлена. Так как данный автомобиль занят другим пользователем на этот день", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

else

{

System.Windows.MessageBox.Show("Заполните все поля и повторите попытку", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

private bool ValidValue()

{

double distance, averageSpeed, consumption, fuelPrice;

if (IsValidDoubleInput(textBoxDistance, 0, 10000, out distance) &

IsValidDoubleInput(textBoxAverSpeed, 0, (selectCar == null) ? selectCar.maxSpeed : 300, out averageSpeed) &

IsValidDoubleInput(textBoxConsumption, 0, 100, out consumption) &

IsValidDoubleInput(textBoxFuelPrice, 0, 1000, out fuelPrice) &

IsValidPlacesInput(textBoxPeople) &

IsValidDataInput(DataPickerFirstData))

{

return true;

}

else

{

return false;

}

}

private string SetValue(ComboBox comboBox)

{

if (comboBox.SelectedIndex != -1)

{

return comboBox.SelectedItem.ToString();

}

else { return "Неизвестно"; }

}

private bool IsValidPlacesInput(TextBox textBoxPeople)

{

if (comboBoxCar.SelectedIndex != -1)

{

if (int.TryParse(textBoxPeople.Text, out int people) && people > 0 && people <= selectCar.seatingCapacity)

{

textBoxPeople.BorderBrush = Brushes.Gray;

return true;

}

textBoxPeople.BorderBrush = Brushes.Red;

return false;

}

textBoxPeople.BorderBrush = Brushes.Gray;

return true;

}

private bool IsValidDoubleInput(TextBox box, int min, int max, out double value)

{

if (box.Text == "")

{

value = 0;

box.BorderBrush = Brushes.Red;

return false;

}

bool isNumeric = double.TryParse(FixStr(box.Text), out value);

if (isNumeric)

{

if (value > min && value < max)

{

box.BorderBrush = Brushes.Gray;

return true;

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

else

{

box.BorderBrush = Brushes.Red;

return false;

}

}

private string FixStr(string input)

{

return input.Replace('.', ',');

}

private bool IsValidDataInput(DatePicker picker)

{

if (picker.SelectedDate.HasValue & picker.SelectedDate >= DateTime.Now.Date)

{

picker.BorderBrush = Brushes.Gray;

return true;

}

else

{

picker.BorderBrush = Brushes.Red;

return false;

}

}

private void MenuItem\_ClickHistory(object sender, RoutedEventArgs e)

{

UserForm.History history = new UserForm.History(idUser);

history.ShowDialog();

}

private void MenuItem\_ClickCar(object sender, RoutedEventArgs e)

{

UserForm.AddCarForm carForm = new UserForm.AddCarForm();

carForm.UpdateValue += EditRoute\_UpdateValue;

carForm.ShowDialog();

}

private void MenuItem\_Click\_1(object sender, RoutedEventArgs e)

{

UserForm.AddRouteForm newRoute = new UserForm.AddRouteForm();

newRoute.UpdateValue += EditRoute\_UpdateValue;

newRoute.ShowDialog();

}

private void MenuItem\_Click\_2(object sender, RoutedEventArgs e)

{

UserForm.EditRouteForm editRoute = new UserForm.EditRouteForm(idUser);

editRoute.UpdateValue += EditRoute\_UpdateValue;

editRoute.ShowDialog();

}

private void EditRoute\_UpdateValue(object sender, EventArgs e)

{

InitComboBox();

}

private void MenuItem\_ClickCar2(object sender, RoutedEventArgs e)

{

UserForm.EditCarForm editCarForm = new UserForm.EditCarForm();

editCarForm.UpdateValue += EditRoute\_UpdateValue;

editCarForm.ShowDialog();

}

private void MenuItem\_Click\_3(object sender, RoutedEventArgs e)

{

UserForm.ReportForm report = new UserForm.ReportForm(idUser);

report.ShowDialog();

}

private void MenuItem\_Click\_4(object sender, RoutedEventArgs e)

{

UserForm.AddFuelForm addForm = new UserForm.AddFuelForm();

addForm.UpdateValue += EditRoute\_UpdateValue;

addForm.ShowDialog();

}

private void MenuItem\_Click\_5(object sender, RoutedEventArgs e)

{

UserForm.EditFuelForm editFuel = new UserForm.EditFuelForm();

editFuel.UpdateValue += EditRoute\_UpdateValue;

editFuel.ShowDialog();

}

private void GenerateReport(Trips trip, Car car, Route route)

{

// Создание диалогового окна сохранения файла

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "Документ Word (\*.docx)|\*.docx";

saveFileDialog.FileName = "TripReport";

// Если пользователь выбрал место сохранения файла и нажал "Сохранить"

if (saveFileDialog.ShowDialog() == true)

{

// Создание документа Word

using (DocX document = DocX.Create(saveFileDialog.FileName))

{

// Добавление заголовка

document.InsertParagraph("Отчет о поездке").FontSize(20d).Bold().Alignment = Alignment.center;

// Добавление информации о поездке

document.InsertParagraph($"Название машины: {car.name}");

document.InsertParagraph($"Точка A маршрута: {route.cityOne}");

document.InsertParagraph($"Точка B маршрута: {route.cityTwo}");

document.InsertParagraph($"Пройденное расстояние: {route.distance}");

document.InsertParagraph($"Расход топлива: {trip.consumption}");

document.InsertParagraph($"Средний расход: {trip.averageConsumption}");

document.InsertParagraph($"Дата поездки: {trip.date}");

document.InsertParagraph($"Цена: {trip.price}");

document.InsertParagraph($"Начало поездки: {trip.timeStart.ToString("HH.mm.ss")}");

document.InsertParagraph($"Окончание поездки: {trip.timeFinish.ToString("HH.mm.ss")}");

document.Save();

System.Windows.MessageBox.Show("Документ сохранен");

}

}

}

}

}<?xml version="1.0" encoding="utf-8"?>

<packages>

<package id="DocX" version="3.0.0" targetFramework="net48" />

<package id="Extended.Wpf.Toolkit" version="4.6.0" targetFramework="net48" />

</packages>

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TripNode.Classes

{

public class Car

{

public string name { get; set; } // Название автомобиля

public int year { get; set; } // Год выпуска

public string typeCar { get; set; } // Кузов автомобиля

public int maxSpeed { get; set; } // Максимальная скорость

public int seatingCapacity { get; set; } // Кол-во мест

public string fuel { get; set; } // Вид топилва

public string fuelOctan { get; set; } // Октановое число

public double fuelConsumptionGeneral { get; set; } //среднее потребление топлива

public Car() { }

public Car(string name, int year, string typeCar, int maxSpeed, int seatingCapacity, string fuel, string fuelOctan, double fuelConsumptionGeneral)

{

this.name = name;

this.year = year;

this.typeCar = typeCar;

this.maxSpeed = maxSpeed;

this.seatingCapacity = seatingCapacity;

this.fuel = fuel;

this.fuelOctan = fuelOctan;

this.fuelConsumptionGeneral = fuelConsumptionGeneral;

}

public string FixStr(double x)

{

return x.ToString().Replace('.', ',');

}

}

}

using System;

using System.Collections.Generic;

using System.Data.SqlClient;

using System.Runtime.ConstrainedExecution;

using System.Windows;

namespace TripNode.Classes

{

class Database

{

private readonly string connectionString;

public Database(string connectionString)

{

this.connectionString = connectionString;

}

public bool RegisterUser(User user)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

// Проверка наличия пользователя с таким логином

using (var checkCommand = new SqlCommand("SELECT COUNT(\*) FROM Users WHERE Login = @Login", connection))

{

checkCommand.Parameters.AddWithValue("@Login", user.login);

int existingUsersCount = (int)checkCommand.ExecuteScalar();

if (existingUsersCount > 0)

{

// Пользователь с таким логином уже существует

return false;

}

}

// Если пользователь с таким логином не найден, выполняем вставку

using (var insertCommand = new SqlCommand("INSERT INTO Users (Login, Password) VALUES (@Login, @Password)", connection))

{

insertCommand.Parameters.AddWithValue("@Login", user.login);

insertCommand.Parameters.AddWithValue("@Password", user.password);

int rowsAffected = insertCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool CheckUser(string login, string password)

{

// Создаем соединение с базой данных MS SQL Server

using (SqlConnection connection = new SqlConnection(connectionString))

{

// Открываем соединение

connection.Open();

// Создаем команду для проверки пароля в базе данных

using (SqlCommand checkPasswordCommand = new SqlCommand(

"SELECT COUNT(\*) FROM Users WHERE Login = @Login AND Password = @Password", connection))

{

// Добавляем параметры для защиты от SQL-инъекций

checkPasswordCommand.Parameters.AddWithValue("@Login", login);

checkPasswordCommand.Parameters.AddWithValue("@Password", password);

// Получаем количество записей, удовлетворяющих условиям запроса

int count = Convert.ToInt32(checkPasswordCommand.ExecuteScalar());

// Проверяем, совпадает ли пароль

if (count == 0)

{

// Выводим сообщение об ошибке

MessageBox.Show("Неверный логин или пароль.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return false; // Возвращаем false, так как пароль не совпадает

}

}

// Закрываем соединение с базой данных

connection.Close();

// Возвращаем true, так как пароль совпадает

return true;

}

}

public int GetIdUser(string user, string password)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand getIdCommand = new SqlCommand(

"SELECT UserID FROM Users WHERE Login = @Login and Password = @Password;", connection))

{

getIdCommand.Parameters.AddWithValue("@Login", user);

getIdCommand.Parameters.AddWithValue("@Password", password);

using (SqlDataReader reader = getIdCommand.ExecuteReader())

{

while (reader.Read())

{

return Convert.ToInt32(reader["UserID"]);

}

}

}

return -1; // Если пользователь не найден

}

}

public bool InsertCar(Car car)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

using (var checkCommand = new SqlCommand("SELECT COUNT(\*) FROM Car WHERE Name = @Name AND Year = @Year", connection))

{

checkCommand.Parameters.AddWithValue("@Name", car.name);

checkCommand.Parameters.AddWithValue("@Year", car.year);

int existingCarsCount = (int)checkCommand.ExecuteScalar();

if (existingCarsCount > 0)

{

return false;

}

}

using (var insertCommand = new SqlCommand("INSERT INTO Car (Name, Year, TypeCar, MaxSpeed, SeatingCapacity, TypeFuel, FuelOctan, ConsumptionFuel) VALUES (@Name, @Year, @TypeCar, @MaxSpeed, @SeatingCapacity, @TypeFuel, @FuelOctan, @ConsumptionFuel)", connection))

{

insertCommand.Parameters.AddWithValue("@Name", car.name);

insertCommand.Parameters.AddWithValue("@Year", car.year);

insertCommand.Parameters.AddWithValue("@TypeCar", car.typeCar);

insertCommand.Parameters.AddWithValue("@MaxSpeed", car.maxSpeed);

insertCommand.Parameters.AddWithValue("@SeatingCapacity", car.seatingCapacity);

insertCommand.Parameters.AddWithValue("@TypeFuel", car.fuel);

insertCommand.Parameters.AddWithValue("@FuelOctan", car.fuelOctan);

insertCommand.Parameters.AddWithValue("@ConsumptionFuel", car.fuelConsumptionGeneral); // Ensure the decimal separator is correct

int rowsAffected = insertCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool DeleteCar(Car car)

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand checkRouteCommand = new SqlCommand(

"SELECT COUNT(\*) FROM Car WHERE Name = @Name;", connection))

{

checkRouteCommand.Parameters.AddWithValue("@Name", car.name);

int count = Convert.ToInt32(checkRouteCommand.ExecuteScalar());

if (count == 0)

{

return false;

}

}

using (SqlCommand deleteCarCommand = new SqlCommand(

"DELETE FROM Car WHERE Name = @Name;", connection))

{

deleteCarCommand.Parameters.AddWithValue("@Name", car.name);

deleteCarCommand.ExecuteNonQuery();

}

}

}

catch (SqlException ex)

{

var result = MessageBox.Show("Ошибка при удалении автмобиля: Автомобиль уже использовался когда-либо. Хотите удалить связанные поездки?", "Ошибка", MessageBoxButton.YesNo, MessageBoxImage.Error);

if (result == MessageBoxResult.Yes)

{

if (DeleteTripsRelatedToCar(GetIdCar(car)))

{

DeleteCar(car);

return true;

}

}

return false;

}

return true;

}

public bool DeleteTripsRelatedToCar(int idCar)

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string deleteTripsCommandText = "DELETE FROM Trips WHERE IdCar = @IdRoute;";

using (SqlCommand deleteTripsCommand = new SqlCommand(deleteTripsCommandText, connection))

{

deleteTripsCommand.Parameters.AddWithValue("@IdRoute", idCar);

deleteTripsCommand.ExecuteNonQuery();

}

}

return true;

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при удалении поездок, связанных с автомобилем");

}

return false;

}

public Car GetCarData(string name)

{

Car selectCar = new Car();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand selectCarCommand = new SqlCommand("SELECT \* FROM Car WHERE Name = @name", connection))

{

selectCarCommand.Parameters.AddWithValue("@name", name);

using (SqlDataReader reader = selectCarCommand.ExecuteReader())

{

if (reader.Read())

{

selectCar = new Car(

reader["Name"].ToString(),

Convert.ToInt32(reader["Year"]),

reader["TypeCar"].ToString(),

Convert.ToInt32(reader["MaxSpeed"]),

Convert.ToInt32(reader["SeatingCapacity"]),

reader["TypeFuel"].ToString(),

reader["FuelOctan"].ToString(),

Convert.ToDouble(reader["ConsumptionFuel"])

);

}

}

}

connection.Close();

}

return selectCar;

}

public int GetIdCar(Car car)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand getIdCommand = new SqlCommand(

"SELECT IdCar FROM Car WHERE Name = @Name and Year = @Year;", connection))

{

getIdCommand.Parameters.AddWithValue("@Name", car.name);

getIdCommand.Parameters.AddWithValue("@Year", car.year);

using (SqlDataReader reader = getIdCommand.ExecuteReader())

{

while (reader.Read())

{

return Convert.ToInt32(reader["IdCar"]);

}

}

}

return -1;

}

}

public List<Car> GetAllCars()

{

List<Car> cars = new List<Car>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// SQL-запрос для выбора всех автомобилей из таблицы Car

using (SqlCommand selectCarsCommand = new SqlCommand("SELECT \* FROM Car", connection))

{

using (SqlDataReader reader = selectCarsCommand.ExecuteReader())

{

while (reader.Read())

{

Car car = new Car

{

name = reader["Name"].ToString(),

year = Convert.ToInt32(reader["Year"]),

typeCar = reader["TypeCar"].ToString(),

maxSpeed = Convert.ToInt32(reader["MaxSpeed"]),

seatingCapacity = Convert.ToInt32(reader["SeatingCapacity"]),

fuel = reader["TypeFuel"].ToString(),

fuelOctan = reader["FuelOctan"].ToString(),

fuelConsumptionGeneral = Convert.ToDouble(reader["ConsumptionFuel"])

};

cars.Add(car);

}

}

}

connection.Close();

}

return cars;

}

public bool InsertRoute(Route route)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Проверяем, существует ли маршрут уже

using (var checkRouteCommand = new SqlCommand(

"SELECT COUNT(\*) FROM Route WHERE ( PointA = @CityOne AND PointB = @CityTwo ) OR ( PointA = @CityTwo AND PointB = @CityOne );", connection))

{

checkRouteCommand.Parameters.AddWithValue("@CityOne", route.cityOne);

checkRouteCommand.Parameters.AddWithValue("@CityTwo", route.cityTwo);

int count = Convert.ToInt32(checkRouteCommand.ExecuteScalar());

if (count > 0)

{

return false;

}

}

// Если маршрут не существует, добавляем его

using (var insertRouteCommand = new SqlCommand(

"INSERT INTO Route (PointA, PointB, RouteLength) VALUES (@CityOne, @CityTwo, @Distance);", connection))

{

insertRouteCommand.Parameters.AddWithValue("@CityOne", route.cityOne);

insertRouteCommand.Parameters.AddWithValue("@CityTwo", route.cityTwo);

insertRouteCommand.Parameters.AddWithValue("@Distance", route.distance);

int rowsAffected = insertRouteCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool UpdateRoute(Route route)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// Проверяем, существует ли маршрут

using (SqlCommand checkRouteCommand = new SqlCommand(

"SELECT COUNT(\*) FROM Route WHERE PointA = @CityOne AND PointB = @CityTwo;", connection))

{

checkRouteCommand.Parameters.AddWithValue("@CityOne", route.cityOne);

checkRouteCommand.Parameters.AddWithValue("@CityTwo", route.cityTwo);

int count = Convert.ToInt32(checkRouteCommand.ExecuteScalar());

if (count == 0)

{

return false; // Маршрут не найден, возвращаем false

}

}

// Если маршрут существует, обновляем его

using (SqlCommand updateRouteCommand = new SqlCommand(

"UPDATE Route SET RouteLength = @Distance WHERE PointA = @CityOne AND PointB = @CityTwo;", connection))

{

updateRouteCommand.Parameters.AddWithValue("@CityOne", route.cityOne);

updateRouteCommand.Parameters.AddWithValue("@CityTwo", route.cityTwo);

updateRouteCommand.Parameters.AddWithValue("@Distance", route.distance);

updateRouteCommand.ExecuteNonQuery();

}

return true; // Обновление маршрута прошло успешно

}

}

public bool DeleteRoute(Route route)

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand checkRouteCommand = new SqlCommand(

"SELECT COUNT(\*) FROM Route WHERE PointA = @PointA AND PointB = @PointB;", connection))

{

checkRouteCommand.Parameters.AddWithValue("@PointA", route.cityOne);

checkRouteCommand.Parameters.AddWithValue("@PointB", route.cityTwo);

int count = Convert.ToInt32(checkRouteCommand.ExecuteScalar());

if (count == 0)

{

return false;

}

}

using (SqlCommand deleteRouteCommand = new SqlCommand(

"DELETE FROM Route WHERE PointA = @CityOne AND PointB = @CityTwo;", connection))

{

deleteRouteCommand.Parameters.AddWithValue("@CityOne", route.cityOne);

deleteRouteCommand.Parameters.AddWithValue("@CityTwo", route.cityTwo);

deleteRouteCommand.ExecuteNonQuery();

}

}

return true;

}

catch (SqlException ex)

{

var result = MessageBox.Show("Ошибка при удалении маршрута: Маршрут уже использовался когда-либо. Хотите удалить связанные поездки?", "Ошибка", MessageBoxButton.YesNo, MessageBoxImage.Error);

if (result == MessageBoxResult.Yes)

{

if (DeleteTripsRelatedToRoute(GetIdRoute(route)))

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

using (SqlCommand deleteRouteCommand = new SqlCommand(

"DELETE FROM Route WHERE PointA = @CityOne AND PointB = @CityTwo;", connection))

{

deleteRouteCommand.Parameters.AddWithValue("@CityOne", route.cityOne);

deleteRouteCommand.Parameters.AddWithValue("@CityTwo", route.cityTwo);

deleteRouteCommand.ExecuteNonQuery();

}

}

return true;

}

}

return false;

}

}

public int GetIdRoute(Route route)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand getIdCommand = new SqlCommand(

"SELECT RouteID FROM Route WHERE PointA = @PointA and PointB = @PointB ;", connection))

{

getIdCommand.Parameters.AddWithValue("@PointA", route.cityOne);

getIdCommand.Parameters.AddWithValue("@PointB ", route.cityTwo);

using (SqlDataReader reader = getIdCommand.ExecuteReader())

{

while (reader.Read())

{

return Convert.ToInt32(reader["RouteID"]);

}

}

}

return -1;

}

}

public List<Route> GetAllRoutes()

{

List<Route> routesWithIds = new List<Route>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// SQL-запрос для выбора всех маршрутов из таблицы Route

using (SqlCommand selectRoutesCommand = new SqlCommand("SELECT \* FROM Route", connection))

{

using (SqlDataReader reader = selectRoutesCommand.ExecuteReader())

{

while (reader.Read())

{

Route route = new Route

{

cityOne = reader["PointA"].ToString(),

cityTwo = reader["PointB"].ToString(),

distance = Convert.ToDouble(reader["RouteLength"])

};

routesWithIds.Add(route);

}

}

}

connection.Close();

}

return routesWithIds;

}

/\*public List<Fuel> GetFuelList(string fuelType)

{

List<Fuel> fuels = new List<Fuel>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

string query = "SELECT \* FROM Fuel WHERE FuelType = @FuelType";

SqlCommand command = new SqlCommand(query, connection);

command.Parameters.AddWithValue("@FuelType", fuelType);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Fuel fuel = new Fuel

(

Convert.ToInt32(reader["FuelId"]),

Convert.ToString(reader["FuelType"]),

Convert.ToString(reader["OctaneNumber"]),

Convert.ToDouble(reader["FuelPrice"])

);

fuels.Add(fuel);

}

reader.Close();

}

return fuels;

}\*/

//использование процедуры

public List<Fuel> GetFuelList(string fuelType)

{

List<Fuel> fuels = new List<Fuel>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

SqlCommand command = new SqlCommand("GetFuelByFuelType", connection);

command.CommandType = System.Data.CommandType.StoredProcedure;

command.Parameters.AddWithValue("@FuelType", fuelType);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Fuel fuel = new Fuel

(

Convert.ToInt32(reader["FuelId"]),

Convert.ToString(reader["FuelType"]),

Convert.ToString(reader["OctaneNumber"]),

Convert.ToDouble(reader["FuelPrice"])

);

fuels.Add(fuel);

}

reader.Close();

}

return fuels;

}

public bool InsertFuel(Fuel fuel)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

// Проверка наличия пользователя с таким логином

using (var checkCommand = new SqlCommand("SELECT COUNT(\*) FROM Fuel WHERE FuelType = @FuelType AND OctaneNumber = @OctaneNumber", connection))

{

checkCommand.Parameters.AddWithValue("@FuelType", fuel.FuelType);

checkCommand.Parameters.AddWithValue("@OctaneNumber", fuel.OctaneNumber);

int existingUsersCount = (int)checkCommand.ExecuteScalar();

if (existingUsersCount > 0)

{

return false;

}

}

using (var insertCommand = new SqlCommand("INSERT INTO Fuel (FuelType, OctaneNumber, FuelPrice) VALUES (@FuelType, @OctaneNumber,@FuelPrice)", connection))

{

insertCommand.Parameters.AddWithValue("@FuelType", fuel.FuelType);

insertCommand.Parameters.AddWithValue("@OctaneNumber", fuel.OctaneNumber);

insertCommand.Parameters.AddWithValue("@FuelPrice", fuel.FuelPrice);

int rowsAffected = insertCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool UpdateFuelPrice(Fuel fuel)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

using (var checkCommand = new SqlCommand("SELECT COUNT(\*) FROM Fuel WHERE FuelType = @FuelType AND OctaneNumber = @OctaneNumber", connection))

{

checkCommand.Parameters.AddWithValue("@FuelType", fuel.FuelType);

checkCommand.Parameters.AddWithValue("@OctaneNumber", fuel.OctaneNumber);

int existingFuelCount = (int)checkCommand.ExecuteScalar();

if (existingFuelCount == 0)

{

// Запись о топливе не найдена, генерируем исключение

MessageBox.Show("Топливо не найдено для обновления цены.");

return false; // Прерываем выполнение метода, так как топливо не найдено

}

}

// Обновление цены на топливо

using (var updateCommand = new SqlCommand("UPDATE Fuel SET FuelPrice = @FuelPrice WHERE FuelType = @FuelType AND OctaneNumber = @OctaneNumber", connection))

{

updateCommand.Parameters.AddWithValue("@FuelType", fuel.FuelType);

updateCommand.Parameters.AddWithValue("@OctaneNumber", fuel.OctaneNumber);

updateCommand.Parameters.AddWithValue("@FuelPrice", fuel.FuelPrice);

int rowsAffected = updateCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool DeleteFuel(string fuelType, string octaneNumber)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

// Проверка использования октанового числа в таблице Car

using (var checkUsageCommand = new SqlCommand("SELECT COUNT(\*) FROM Car WHERE FuelOctan = @OctaneNumber", connection))

{

checkUsageCommand.Parameters.AddWithValue("@OctaneNumber", octaneNumber);

int carCountUsingFuel = (int)checkUsageCommand.ExecuteScalar();

if (carCountUsingFuel > 0)

{

// Топливо используется в таблице Car, выводим сообщение

MessageBox.Show("Невозможно удалить топливо, так как оно используется в таблице Car.");

return false; // Прерываем выполнение метода, так как топливо используется

}

}

// Удаление записи о топливе

using (var deleteCommand = new SqlCommand("DELETE FROM Fuel WHERE FuelType = @FuelType AND OctaneNumber = @OctaneNumber", connection))

{

deleteCommand.Parameters.AddWithValue("@FuelType", fuelType);

deleteCommand.Parameters.AddWithValue("@OctaneNumber", octaneNumber);

int rowsAffected = deleteCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool DeleteTripsRelatedToRoute(int idRoute)

{

try

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

string deleteTripsCommandText = "DELETE FROM Trips WHERE IdRoute = @IdRoute;";

using (SqlCommand deleteTripsCommand = new SqlCommand(deleteTripsCommandText, connection))

{

deleteTripsCommand.Parameters.AddWithValue("@IdRoute", idRoute);

deleteTripsCommand.ExecuteNonQuery();

}

}

return true;

}

catch (SqlException ex)

{

MessageBox.Show("Ошибка при удалении поездок, связанных с маршрутом");

}

return false;

}

public List<Trips> GetAllTrips()

{

List<Trips> trips = new List<Trips>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand command = new SqlCommand("SELECT \* FROM Trips", connection))

{

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

Trips trip = new Trips

{

idUser = reader.GetInt32(0),

idCar = reader.GetInt32(1),

idRoute = reader.GetInt32(2),

distance = reader.GetDouble(3),

consumption = reader.GetDouble(4),

averageConsumption = reader.GetDouble(5),

date = reader.GetString(6),

price = reader.GetDouble(7),

timeStart = reader.GetDateTime(8),

timeFinish = reader.GetDateTime(9)

};

trips.Add(trip);

}

}

}

return trips;

}

public List<TripInfo> GetTripsByUserId(int userId)

{

List<TripInfo> trips = new List<TripInfo>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

string sqlQuery = @"SELECT

u.Login,

c.Name,

r.PointA,

r.PointB,

t.Distance,

t.Consumption,

t.AverageConsumption,

t.TripDate,

t.Price,

t.TimeStart,

t.TimeFinish

FROM

Trips t

JOIN

Users u ON t.IdUser = u.UserID

JOIN

Car c ON t.IdCar = c.IdCar

JOIN

Route r ON t.IdRoute = r.RouteID

WHERE

t.IdUser = @UserId";

SqlCommand command = new SqlCommand(sqlQuery, connection);

command.Parameters.AddWithValue("@UserId", userId);

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

TripInfo trip = new TripInfo(

reader["Login"].ToString(),

reader["Name"].ToString(),

reader["PointA"].ToString(),

reader["PointB"].ToString(),

Convert.ToDouble(reader["Distance"]),

Convert.ToDouble(reader["Consumption"]),

Convert.ToDouble(reader["AverageConsumption"]),

Convert.ToDateTime(reader["TripDate"]),

Convert.ToDecimal(reader["Price"]),

Convert.ToDateTime(reader["TimeStart"]),

Convert.ToDateTime(reader["TimeFinish"])

);

trips.Add(trip);

}

reader.Close();

}

return trips;

}

public List<TripInfo> GetTripsBetweenDate(int userId, DateTime? startDate, DateTime? endDate)

{

List<TripInfo> trips = new List<TripInfo>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

string sqlQuery = @"SELECT

t.IdTrip,

u.Login,

c.Name AS CarName,

r.PointA AS RoutePointA,

r.PointB AS RoutePointB,

t.Distance,

t.Consumption,

t.AverageConsumption,

t.TripDate,

t.Price,

t.TimeStart,

t.TimeFinish

FROM

Trips t

JOIN

Users u ON t.IdUser = u.UserID

JOIN

Car c ON t.IdCar = c.IdCar

JOIN

Route r ON t.IdRoute = r.RouteID

WHERE

t.IdUser = @UserId";

// Создаем команду SQL с предварительно добавленными параметрами

SqlCommand command = new SqlCommand(sqlQuery, connection);

command.Parameters.AddWithValue("@UserId", userId);

if (endDate.HasValue)

{

sqlQuery += " AND t.TripDate <= @EndDate";

command.Parameters.AddWithValue("@EndDate", endDate.Value);

}

if (startDate.HasValue)

{

sqlQuery += " AND t.TripDate >= @StartDate";

command.Parameters.AddWithValue("@StartDate", startDate.Value);

}

// Используем команду для выполнения запроса SQL

command.CommandText = sqlQuery; // Устанавливаем текст запроса

connection.Open();

SqlDataReader reader = command.ExecuteReader();

while (reader.Read())

{

TripInfo trip = new TripInfo(

reader["Login"].ToString(),

reader["CarName"].ToString(),

reader["RoutePointA"].ToString(),

reader["RoutePointB"].ToString(),

Convert.ToDouble(reader["Distance"]),

Convert.ToDouble(reader["Consumption"]),

Convert.ToDouble(reader["AverageConsumption"]),

Convert.ToDateTime(reader["TripDate"]),

Convert.ToDecimal(reader["Price"]),

Convert.ToDateTime(reader["TimeStart"]),

Convert.ToDateTime(reader["TimeFinish"])

);

trips.Add(trip);

}

reader.Close();

}

return trips;

}

public bool InsertTrip(Trips trip)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

using (var insertCommand = new SqlCommand("INSERT INTO Trips (IdUser, IdCar, IdRoute, Distance, Consumption, AverageConsumption, TripDate, Price, TimeStart, TimeFinish) VALUES (@IdUser, @IdCar, @IdRoute, @Distance, @Consumption, @AverageConsumption, @TripDate, @Price, @TimeStart, @TimeFinish)", connection))

{

insertCommand.Parameters.AddWithValue("@IdUser", trip.idUser);

insertCommand.Parameters.AddWithValue("@IdCar", trip.idCar);

insertCommand.Parameters.AddWithValue("@IdRoute", trip.idRoute);

insertCommand.Parameters.AddWithValue("@Distance", trip.distance);

insertCommand.Parameters.AddWithValue("@Consumption", trip.consumption);

insertCommand.Parameters.AddWithValue("@AverageConsumption", trip.averageConsumption);

insertCommand.Parameters.AddWithValue("@TripDate", DateTime.Parse(trip.date));

insertCommand.Parameters.AddWithValue("@Price", trip.price);

insertCommand.Parameters.AddWithValue("@TimeStart", trip.timeStart);

insertCommand.Parameters.AddWithValue("@TimeFinish", trip.timeFinish);

int rowsAffected = insertCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public int GetIdRoute(string pointA, string pointB)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (SqlCommand getIdCommand = new SqlCommand(

"SELECT RouteID FROM Route WHERE PointA = @PointA and PointB = @PointB;", connection))

{

getIdCommand.Parameters.AddWithValue("@PointA", pointA);

getIdCommand.Parameters.AddWithValue("@PointB", pointB);

using (SqlDataReader reader = getIdCommand.ExecuteReader())

{

while (reader.Read())

{

return Convert.ToInt32(reader["RouteID"]);

}

}

}

return -1;

}

}

public bool InsertFavoriteRoute(int IdUser, int IdRoute)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (var checkRouteCommand = new SqlCommand(

"SELECT COUNT(\*) FROM FavoriteRoute WHERE IdUser = @IdUser AND IdRoute = @IdRoute;", connection))

{

checkRouteCommand.Parameters.AddWithValue("@IdUser", IdUser);

checkRouteCommand.Parameters.AddWithValue("@IdRoute", IdRoute);

int count = Convert.ToInt32(checkRouteCommand.ExecuteScalar());

if (count > 0)

{

return false;

}

}

using (var insertRouteCommand = new SqlCommand(

"INSERT INTO FavoriteRoute (IdUser, IdRoute) VALUES (@IdUser, @IdRoute);", connection))

{

insertRouteCommand.Parameters.AddWithValue("@IdUser", IdUser);

insertRouteCommand.Parameters.AddWithValue("@IdRoute", IdRoute);

int rowsAffected = insertRouteCommand.ExecuteNonQuery();

return rowsAffected > 0;

}

}

}

public bool DeleteFavoriteRoute(int IdUser, int IdRoute)

{

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

using (var checkRouteCommand = new SqlCommand(

"SELECT COUNT(\*) FROM FavoriteRoute WHERE IdUser = @IdUser AND IdRoute = @IdRoute;", connection))

{

checkRouteCommand.Parameters.AddWithValue("@IdUser", IdUser);

checkRouteCommand.Parameters.AddWithValue("@IdRoute", IdRoute);

int count = Convert.ToInt32(checkRouteCommand.ExecuteScalar());

if (count == 0)

{

return false;

}

}

using (var deleteRouteCommand = new SqlCommand(

"DELETE FROM FavoriteRoute WHERE IdUser = @IdUser AND IdRoute = @IdRoute;", connection))

{

deleteRouteCommand.Parameters.AddWithValue("@IdUser", IdUser);

deleteRouteCommand.Parameters.AddWithValue("@IdRoute", IdRoute);

deleteRouteCommand.ExecuteNonQuery();

}

}

return true;

}

public List<FavoriteRoute> GetFavoriteRoutes(int idUser)

{

List<FavoriteRoute> favoriteRoutes = new List<FavoriteRoute>();

using (SqlConnection connection = new SqlConnection(connectionString))

{

connection.Open();

// SQL-запрос для выбора всех маршрутов из таблицы FavoriteRoute для указанного IdUser

string sqlQuery = @"

SELECT FR.IdFavoriteRoute, FR.IdUser, FR.IdRoute, R.PointA, R.PointB

FROM FavoriteRoute FR

JOIN Route R ON FR.IdRoute = R.RouteID

WHERE FR.IdUser = @IdUser";

using (SqlCommand selectFavoriteRoutesCommand = new SqlCommand(sqlQuery, connection))

{

selectFavoriteRoutesCommand.Parameters.AddWithValue("@IdUser", idUser);

using (SqlDataReader reader = selectFavoriteRoutesCommand.ExecuteReader())

{

while (reader.Read())

{

FavoriteRoute route = new FavoriteRoute(

reader["PointA"].ToString(),

reader["PointB"].ToString()

);

favoriteRoutes.Add(route);

}

}

}

connection.Close();

}

return favoriteRoutes;

}

public decimal GetTotalTripPricesForUser(int userId)

{

using (var connection = new SqlConnection(connectionString))

{

connection.Open();

using (var command = new SqlCommand("SELECT dbo.GetTotalTripPricesForUser(@UserId)", connection))

{

command.Parameters.AddWithValue("@UserId", userId);

var result = command.ExecuteScalar();

return result == DBNull.Value ? 0 : Convert.ToDecimal(result);

}

}

}

}

}using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TripNode.Classes

{

public class FavoriteRoute

{

public string PointA; // Точка отправки

public string PointB; // Точка прибытия

public FavoriteRoute(string pointA, string pointB)

{

PointA = pointA;

PointB = pointB;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TripNode.Classes

{

public class Fuel

{

public int FuelId { get; set; }

public string FuelType { get; set; }

public string OctaneNumber { get; set; }

public double FuelPrice { get; set; }

public Fuel(string fuelType, string octaneNumber, double fuelPrice)

{

FuelType = fuelType;

OctaneNumber = octaneNumber;

FuelPrice = fuelPrice;

}

public Fuel(int fuelId, string fuelType, string octaneNumber, double fuelPrice)

{

FuelId = fuelId;

FuelType = fuelType;

OctaneNumber = octaneNumber;

FuelPrice = fuelPrice;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TripNode.Classes

{

class Route

{

public string cityOne { get; set; } // Точка отправления

public string cityTwo { get; set; } // Точка прибытия

public double distance { get; set; } // Расстояние

public Route() { }

public Route(string cityOne, string cityTwo, double distance)

{

this.cityOne = cityOne;

this.cityTwo = cityTwo;

this.distance = distance;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TripNode.Classes

{

public class TripInfo

{

public string UserLogin { get; set; }

public string CarName { get; set; }

public string RoutePointA { get; set; }

public string RoutePointB { get; set; }

public double Distance { get; set; }

public double Consumption { get; set; }

public double AverageConsumption { get; set; }

public DateTime TripDate { get; set; }

public decimal Price { get; set; }

public DateTime TimeStart { get; set; }

public DateTime TimeFinish { get; set; }

public TripInfo(string userLogin, string carName, string routePointA, string routePointB,

double distance, double consumption, double averageConsumption, DateTime tripDate,

decimal price, DateTime timeStart, DateTime timeFinish)

{

UserLogin = userLogin;

CarName = carName;

RoutePointA = routePointA;

RoutePointB = routePointB;

Distance = distance;

Consumption = consumption;

AverageConsumption = averageConsumption;

TripDate = tripDate;

Price = price;

TimeStart = timeStart;

TimeFinish = timeFinish;

}

}

}

using System;

using System.Security.RightsManagement;

namespace TripNode.Classes

{

class Trips

{

public int idUser { get; set; } // id Пользователя

public int idCar { get; set; } // id Автомобиля

public int idRoute { get; set; } // id Маршрута

public double distance { get; set; } // Расстоярние поездки

public double consumption { get; set; } // Потребление общее

public double averageConsumption { get; set; } // Потребление за км

public string date { get; set; } // Дата совершения поезкди

public double price { get; set; } // Сумма поездки

public DateTime timeStart { get; set; } // Дата отправки

public DateTime timeFinish { get; set; } // Дата прибытия

public Trips() { }

public Trips(int idUser, int idCar, int idRoute, double distance, double consumption, double averageConsumption, string date, double price, DateTime timeStart, DateTime timeFinish)

{

this.idUser = idUser;

this.idCar = idCar;

this.idRoute = idRoute;

this.distance = distance;

this.consumption = consumption;

this.averageConsumption = averageConsumption;

this.date = date;

this.price = price;

this.timeStart = timeStart;

this.timeFinish = timeFinish;

}

}

}

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace TripNode.Classes

{

class User

{

public string login { get; set; } //Логин пользователя

public string password { get; set; } // Пароль пользователя

public User() { }

public User(string login, string password)

{

this.login = login;

this.password = password;

}

}

}

# ПРИЛОЖЕНИЕ Б

**(справочное)**

**Результат работы программы**

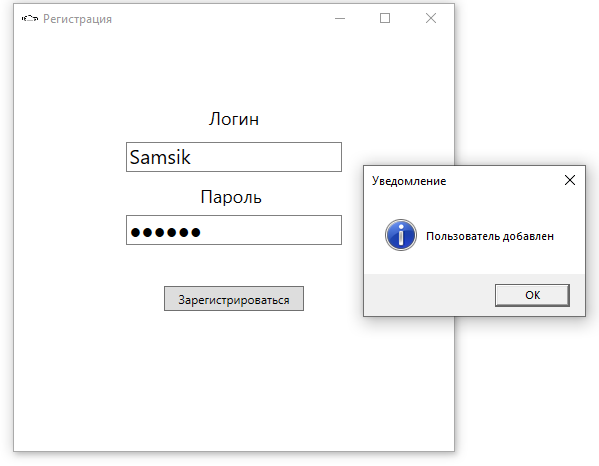


Рисунок Б.1 – Результат действия «Регистрация пользователя»

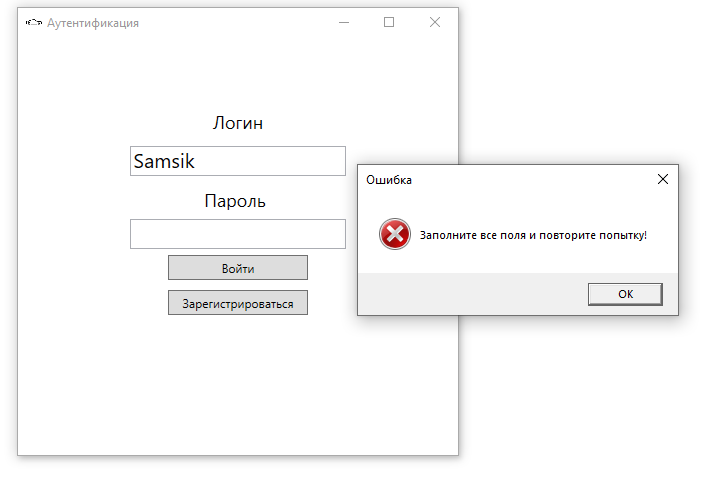


Рисунок Б.2 – Результат действия «Авторизация в приложении»

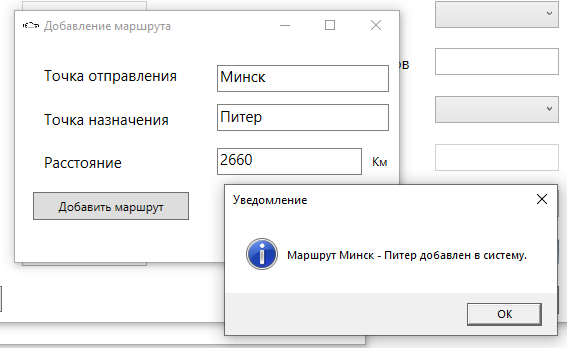


Рисунок Б.3 – Результат действия «Добавление маршрута»

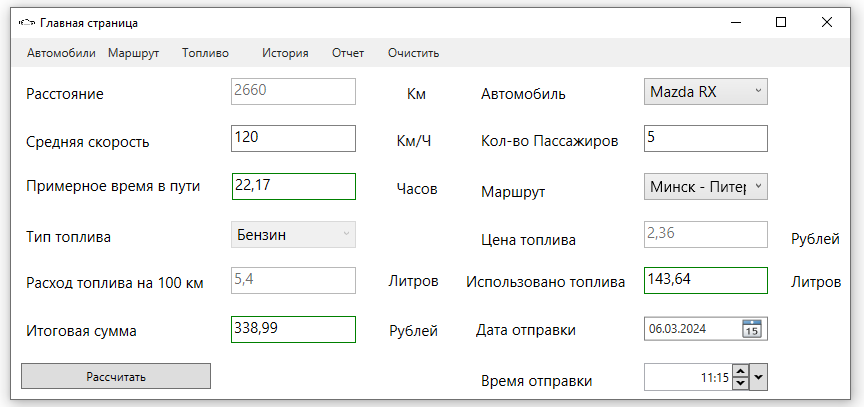


Рисунок Б.4 – Результат действия «Расчет стоимости поездки»

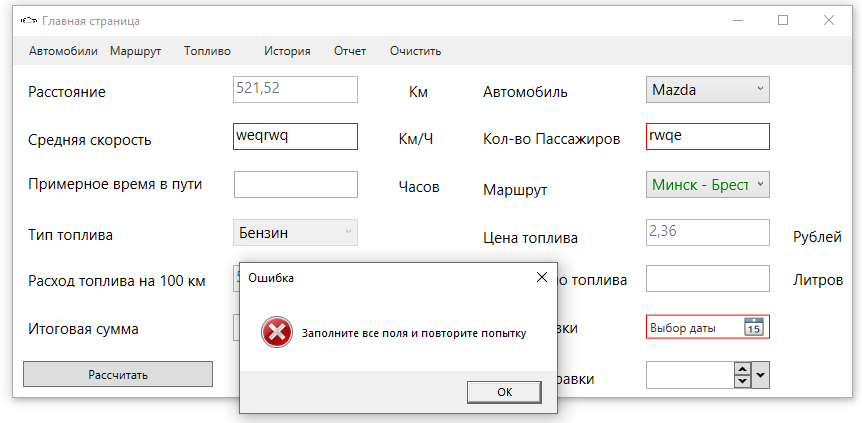


Рисунок Б.5 – Результат действия «Расчет стоимости поездки»

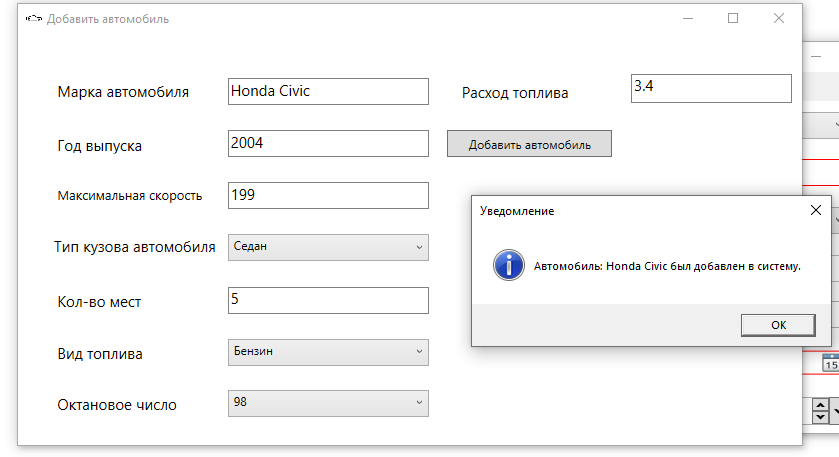


Рисунок Б.6 – Результат действия «Добавление автомобиля»

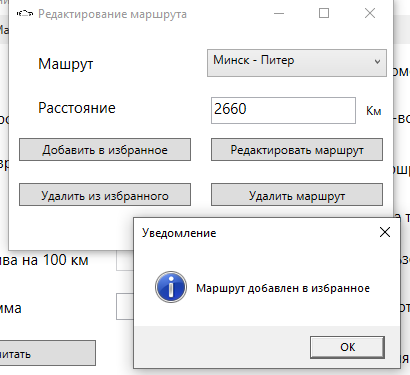


Рисунок Б.7 – Результат действия «Добавление любимого маршрута»

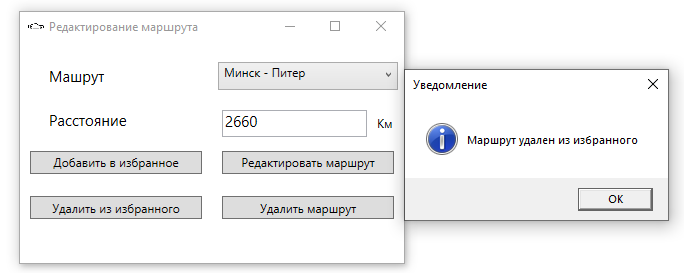
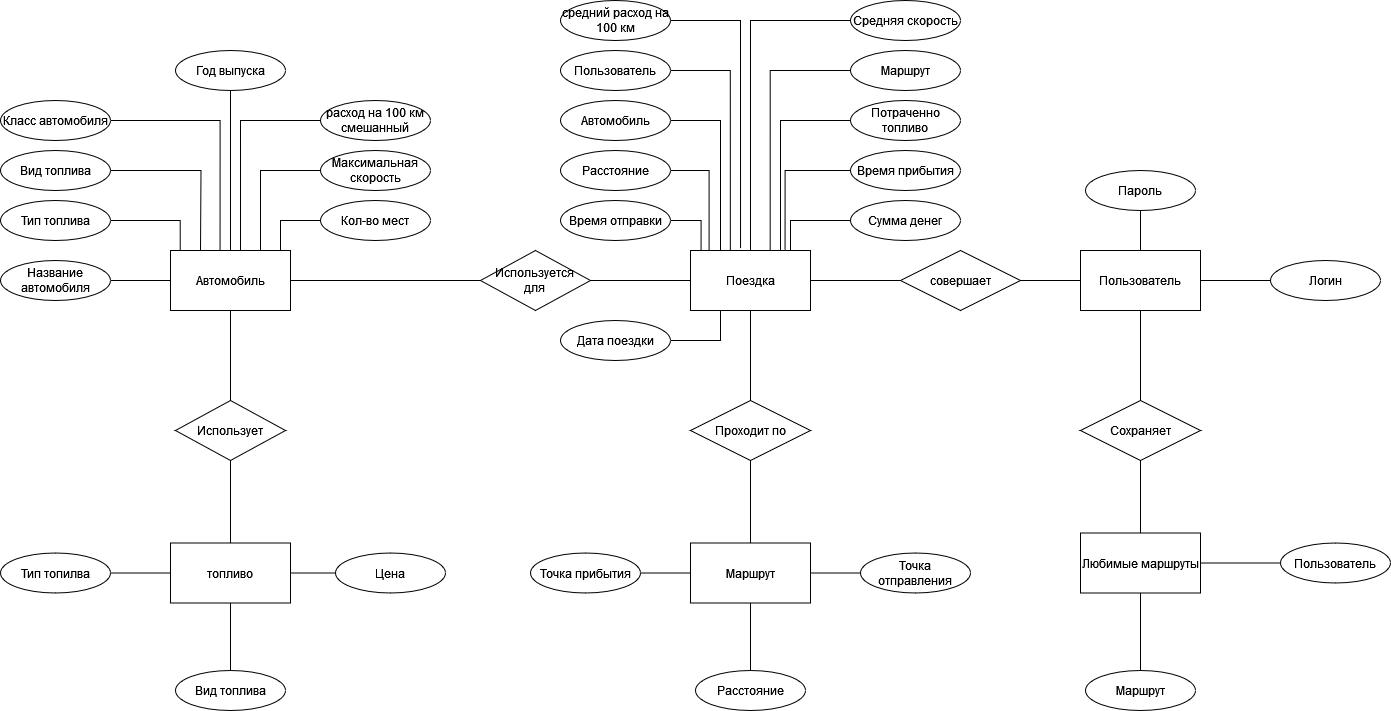


Рисунок Б.8 – Результат действия «Удаление любимого маршрута»



Масса

Лиcт.

Масштаб

Утверд.

Реценз.

Провер.

Ржеутская Н.В.

№ докум.

Подпись

Дата

Изм.

Лист

*Разработка программного средства для расчета стоимости поездки на личном автотранспорте*

*Диаграмма сущность-связь (концептуальная модель)*

Разраб.

Самойлов А.С.

Лист 1

Подп. и дата

Инв.№ дубл.

Инв.№подл.

*КП Т.091016.401 ГЧ*

*КП Т.091016.401 ГЧ*

Т. Контр.

Листов 4

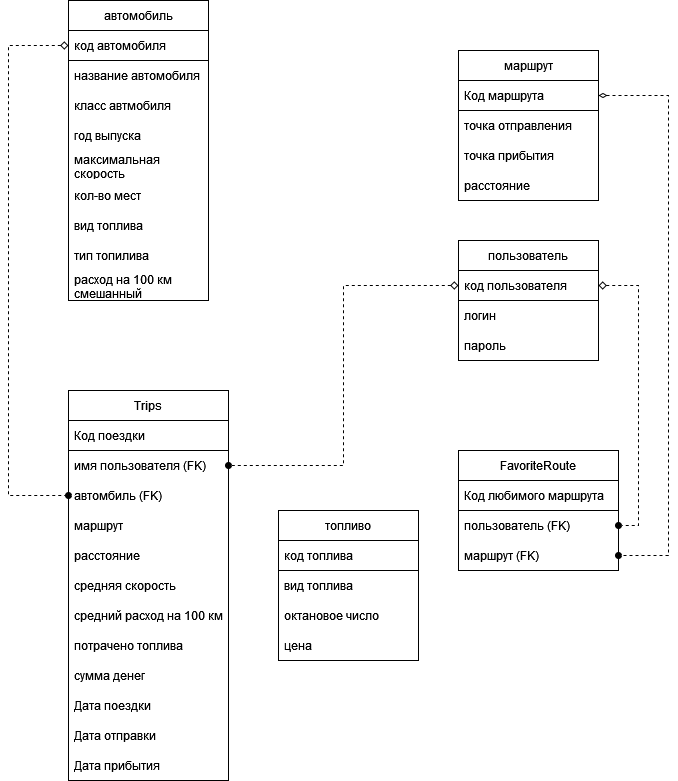
У

КБП

Н. Контр.

Взам.инв.№

Подп. и дата



Утверд.

Т. Контр.

Реценз.

Н. Контр.

№ докум.

Подпись

Дата

Изм.

Лист

Провер.

Ржеутская Н.В.

Разраб.

Самойлов А.С.

Лист 2

Подп. и дата

Инв.№ дубл.

Инв.№подл.

*КП Т.091016.401 ГЧ*

*КП Т.091016.401 ГЧ*

Листов 4

*Разработка программного средства для расчета стоимости поездки на личном автотранспорте*

*Диаграмма сущность-связь (логическая модель)*

У

Масса

Лиcт.

Масштаб

КБП

Взам.инв.№

Подп. и дата

Лист 3

Подп. и дата

Инв.№ дубл.

Инв.№подл.

*КП Т.091016.401 ГЧ*

*КП Т.091016.401 ГЧ*

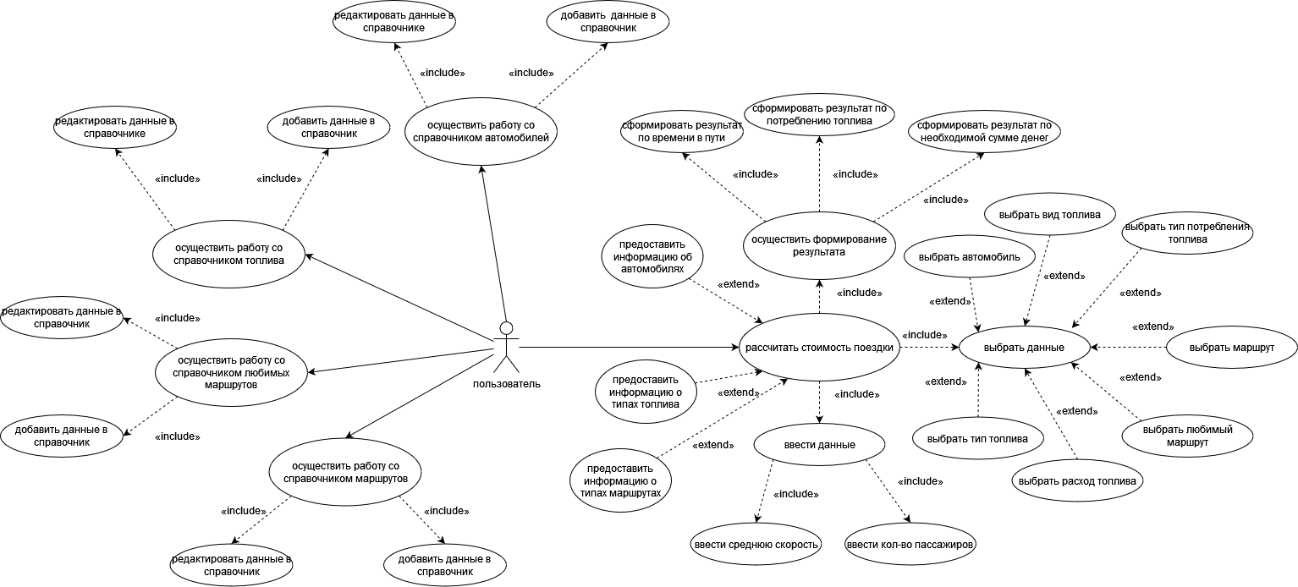
Листов 4

У

КБП

Взам.инв.№

Подп. и дата



Утверд.

Т. Контр.

Реценз.

Н. Контр.

Разраб.

Самойлов А.С.

№ докум.

Подпись

Дата

Изм.

Лист

Масса

Лиcт.

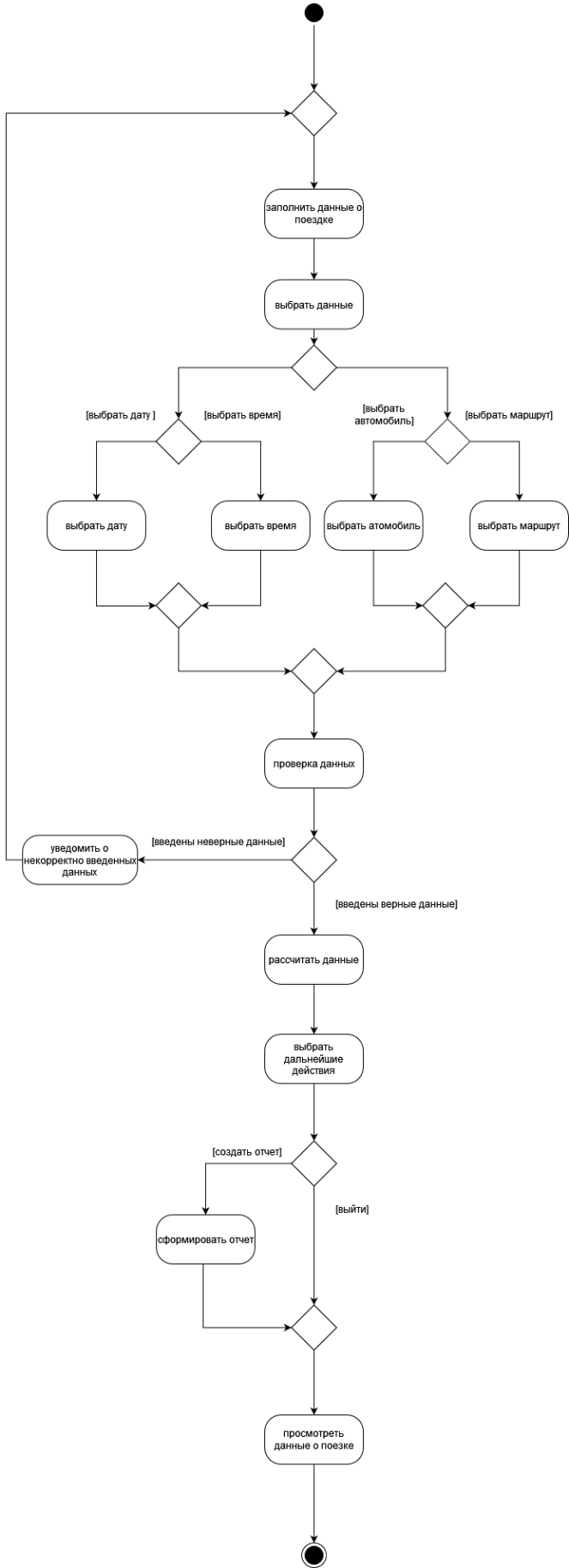
Масштаб

Ржеутская Н.В.

Провер.

*Разработка программного средства для расчета стоимости поездки на личном автотранспорте*

*Диаграмма вариантов использования*



Масса

Лиcт.

Масштаб

Утверд.

Т. Контр.

Реценз.

Н. Контр.

№ докум.

Подпись

Дата

Изм.

Лист

*Разработка программного средства для расчета стоимости поездки на личном автотранспорте*

*Диаграмма деятельности по функции создание поездки*

Провер.

Ржеутская Н.В.

Разраб.

Самойлов А.С.

Лист 4

Подп. и дата

Инв.№ дубл.

Инв.№подл.

*КП Т.091016.401 ГЧ*

*КП Т.091016.401 ГЧ*

Листов 4

У

КБП

Взам.инв.№

Подп. и дата

**Этикетка**

для курсового проекта

**Курсовой проект**

Тема « Разработка программного средства для расчета стоимости поездки на личном автотранспорте»

КП Т.091016.401

Разработан

Утвержден

Разработчик Самойлов А.С.

Руководитель: Ржеутская Н.В.

Технические средства: Процессор 1.3 ГГц, ОЗУ 6 ГБ, 500 ГБ свободного места

Программные средства: MS Visual Studio 2022 Community, MS Word, Microsoft SQL Server Management Studio 18

**Состав документа:**

Пояснительная записка – ПЗ\_СамойловТ091.docх

Графическая часть – Диаграмма «Сущность-связь» (концептуальная модель).docx, Диаграмма «Сущность-связь» (логическая модель).docx, Диаграмма вариантов использования.docx, Диаграмма деятельности по функции создание поездки.docx.

Папка с проектом – TripNode

Установочный пакет программного средства – TripNodeSetup.exe

**Удостоверяющий лист**

электронного документа – курсовой проект

Тема « Разработка программного средства для расчета стоимости поездки на личном автотранспорте»

Обозначение\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ КП Т.091016.401\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Разработчик Самойлов А.С. Руководитель Ржеутская Н.В.

(Ф.И.О.) (Ф.И.О.)

Подписи лиц, ответственных за разработку электронного документа

|  |  |  |
| --- | --- | --- |
| Состав электронного документа | Разработчик | Руководитель |
| Пояснительная записка (на бумажном носителе формата А4), ПЗ\_СамойловТ091.docх |  |  |
| ГЧ, Диаграмма «Сущность-связь» (концептуальная модель).docx |  |  |
| ГЧ, Диаграмма «Сущность-связь» (логическая модель).docx |  |  |
| ГЧ, Диаграмма вариантов использования.docx |  |  |
| ГЧ, Диаграмма деятельности по функции создание поездки.docx |  |  |
| Папка с проектом «TripNode» |  |  |
| Установочный пакет программного средства «TripNodeSetup.exe» |  |  |
| Тип носителя: оптический компакт-диск |  |  |