Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Заместитель директора  по учебной работе  З.З. Курмашева  « » 2024 г. |

ПРОЕКТИРОВАНИЕ БАЗЫ ДАННЫХ

ДЛЯ ФОРМИРОВАНИЯ МАРШРУТА ПО АВТОМОБИЛЬНЫМ ДОРОГАМ

Пояснительная записка к курсовому проекту

МДК 11.01 Технология разработки и защиты баз данных

|  |  |
| --- | --- |
|  | Руководитель проекта  Р.Ф. Каримова  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |
|  | Студент гр. 21П-1  Ф.Б. Биктимеров  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г. |

2024

Министерство образования и науки Республики Башкортостан

Государственное автономное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

|  |  |
| --- | --- |
|  | УТВЕРЖДАЮ  Заместитель директора  по учебной работе  З.З. Курмашева  « » 2024 г. |

ЗАДАНИЕ

на курсовой проект студенту дневного отделения, группы 21П-1, специальности 09.02.07 Информационные системы и программирование

Фамилия, имя, отчество: Биктимеров Фаяз Фанилович

Тема курсового проекта: «Проектирование базы данных для формирования маршрута по автомобильным дорогам».

Текст задания:

при выполнении курсового проекта должны быть решены следующие задачи:

1. спроектирована структура базы данных;
2. разработана структура программы;
3. реализованы функции приема и продажи предметов, распределения их по аукционам, составления расписания аукционов, заключения договоров о передаче предметов на продажу и договоров купли-продажи предметов, формирования итоговой сводной ведомости.

В результате выполнения курсового проекта должны быть представлены:

1. пояснительная записка, состоящая из следующих разделов:

Введение

1 Постановка задачи

2 Экспериментальный раздел

Заключение

Приложения

Список сокращений

Список использованных источников

1. электронный носитель, содержащий разработанный программный продукт;
2. презентация курсового проекта в электронном виде.

Список рекомендуемых источников:

1. Култыгин, О. П. Култыгин, О. П. Администрирование баз данных. СУБД MS SQL Server [Текст] : учеб. пособ. / О. П. Култыгин. - М.: МФПА, 2012. - 232 с.
2. Фуфаев, Э.В. Базы данных [Текст]: учеб. пособ. для студ. учрежд. сред. проф. образования / Э.В. Фуфаев, Д.Э. Фуфаев. - 6-е изд., стер. - М.: Издательский центр «Академия», 2012.- 320 с.- (Среднее профессиональное образование)
3. Википедия [Электронный ресурс] // Свободная энциклопедия. – Режим доступа: <http://ru.wikipedia.org/wiki/>, свободный

Задание к выполнению получил «31» января 2024 г.

Студент Иванов Иван Иванович

Срок окончания «31» мая 2024 г.

Руководитель курсового проекта Р.Ф. Каримова

Задание рассмотрено на заседании цикловой комиссии информатики

«11» января 2024 г.

Председатель цикловой комиссии информатики О.В.Фатхулова

Министерство образования и науки Республики Башкортостан

Государственное бюджетное профессиональное образовательное учреждение

Уфимский колледж статистики, информатики и вычислительной техники

ЗАКЛЮЧЕНИЕ

на курсовой проект

Студент Биктимеров Фаяз Фанилович

Группа 21П-1

Специальность 09.02.07 Информационные системы и программирование

Тема Проектирование базы данных для формирования маршрута по автомобильным дорогам

Объем курсового проекта:

количество листов пояснительной записки

количество листов графической части

Заключение о степени соответствия заданию на курсовое проектирование

Характеристика качеств, проявленных студентом при работе над проектом: самостоятельность, дисциплинированность, умение планировать работу и пользоваться литературным материалом и т.д.

Положительные стороны курсового проекта

Недостатки курсового проекта

Характеристика общетехнической и специальной подготовки студента

Заключение и предлагаемая оценка за курсовой проект

Руководитель курсового проекта Каримова Резида Флюновна

«\_\_\_\_» 2024 г.

Подпись

АННОТАЦИЯ

Пояснительная записка к курсовому проекту содержит постановку и программу решения задачи «Проектирование базы данных формирования маршрута по автомобильным дорогам».

Программа Roads.exe написана на языке C# в среде программирования Visual Studio 2019 с использованием системы управления базой данных MS SQL SERVER предназначена для работы в операционной системе MS Windows 10 и выше, отлажена на данных контрольного примера.

СОДЕРЖАНИЕ

|  |  |
| --- | --- |
|  | лист |
| Введение | 4 |
| 1 Постановка задачи | 6 |
| 1.1 Описание предметной области | 6 |
| 1.2 Описание входной информации | 8 |
| 1.3 Описание выходной информации | 8 |
| 1.4 Концептуальное моделирование | 9 |
| 1.5 Логическое моделирование | 10 |
| 1.6 Описание структуры базы данных | 12 |
| 1.7 Контрольный пример | 14 |
| 1.8 Общие требования к программному продукту | 14 |
| 2 Экспериментальный раздел | 16 |
| 2.1 Описание программы | 16 |
| 2.2 Руководство пользователя | 25 |
| 2.3 Протокол тестирования программного продукта | 34 |
| Заключение | 46 |
| Приложения | 47 |
| Список источников | 107 |

ВВЕДЕНИЕ

В современном мире автомобильные дороги играют важную роль в обеспечении мобильности и связности населенных пунктов. Однако для водителей часто возникает необходимость получения информации о маршрутах, условиях движения, а также о наличии сервисных объектов вдоль дороги, таких как авторемонтные пункты и автозаправочные станции. В этом контексте информационная система, способная эффективно обрабатывать данные об автомобильных дорогах и предоставлять пользователю необходимую информацию, становится важным инструментом для автолюбителей.

Данная программа система предназначена для обеспечения удобного и быстрого доступа к информации о маршрутах, условиях движения и сервисных объектах на автомобильных дорогах. Система предоставляет возможности как обычным пользователям, так и администраторам. Обычные пользователи могут просматривать маршруты, осуществлять поиск по заданным критериям и фильтрам, а также получать информацию о населенных пунктах, авторемонтных пунктах и автозаправочных станциях, расположенных вдоль маршрута. Администраторы имеют возможность управлять базой данных, включая добавление и редактирование информации о маршрутах, населенных пунктах, авторемонтных пунктах и автозаправочных станциях.

Целью данного проекта является разработка полнофункциональной информационной системы, способной эффективно удовлетворять потребности пользователей в получении информации о дорожной инфраструктуре и сервисных объектах на автомобильных дорогах. Программа предоставляет пользователю удобный интерфейс для ввода критериев поиска и отображения результатов в виде отчетной документации, содержащей информацию о маршрутах, времени в пути и расположении сервисных объектов.

Задачами курсового проекта являются:

* описать предметную область;
* разработать структуру базы данных;
* разработать приложение;
* провести тестирование приложения.

1. Постановка задачи

* 1. Описание предметной области

Требуется разработать информационную систему для просмотра автомобильных дорог и расположенных на нем заправок, автомастерских и населенных пунктов, которая будет использоваться водителями для более комфортного маршрута. Время поездки рассчитываться по формуле: длинна дороги деленная на среднею скорость, которая равна 90% от предельной скорости.

Данная информационная система предполагает наличие трех групп пользователей: обычный пользователь и администратор.

Не зарегистрированный пользователь имеет возможность:

* поиск по критериям;
* фильтрация по параметрам;
* просматривать уже сформированные маршруты.

Администратор имеет следующие возможности:

* добавление новых маршрутов;
* поиск по критериям;
* редактирование маршрутов;
* фильтрация по параметрам;
* формировать новые маршруты.

Зарегистрированный пользователь имеет следующие возможности:

* поиск по критериям;
* фильтрация по параметрам;
* формировать новые маршруты.

В базе данных должны храниться следующие справочники: населенные пункты, заправочные станции, авторемонтные пункты, категории дороги, пользователи и роли.

В таблице «Дорога» содержится следующая информация:

* идентификатор дороги;
* протяженность в километрах;
* тиа покрытия;
* номер категории;
* ограничение скорости;
* номер начального города;
* номер конечного города.

В таблице «Населенные пункты» содержится следующая информация:

* идентификатор изображения;
* идентификатор лота;
* изображение.

В таблице «Заправочные станции» содержится следующая информация:

* индификатор аукциона;
* название;
* номер города;
* расположение;

В таблице «Авторемонтные пункты» содержится следующая информация:

* идентификатор авторемонтного пункта;
* название;
* номер города;
* расположение.

В таблице «Категории дороги» содержится следующая информация:

* идентификатор категории дороги;
* название.

В таблице «Пользователь» содержится следующая информация:

* идентификатор пользователя;
* логин;
* пароль;
* номер роли.

В таблице «Роли» содержится следующая информация:

* идентификатор роли;
* название роли.

1.2 Описание входной информации

Входной информацией для формирования маршрута это начальный и конечный город.

1.3 Описание выходной информации

* DOCX документ маршрута с информацией о дороге и объеках на нем;
* User Control с информаципей о готовом маршруте.

Выходной информацией является отчет о маршруте (таблица 1.3.1):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Наименование документа (шифр) | Периодичность выдачи документа | Кол-во экз. | Куда передаются | Поля сортировки | Поля группировки | Итоги |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| DOCX документ маршрута с информацией о дороге и объеках на нем | По мере необходимости | 1 | Клиенту | - | - | - |

Таблица 1.3.1

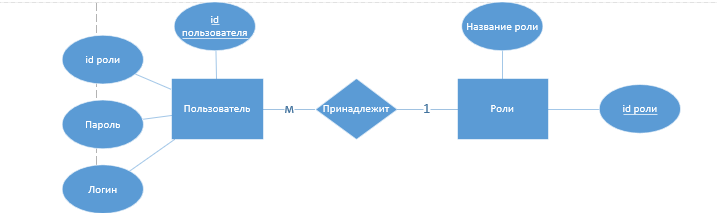
Шаблоны выходных документов представлены в приложении А.

1.4. Концептуальное моделирование

Концептуальная модель базы данных - это некая наглядная диаграмма, нарисованная в принятых обозначениях и подробно показывающая связь между объектами и их характеристиками. Создается концептуальная модель для дальнейшего проектирования базы данных и перевод ее, например, в реляционную базу данных. На концептуальной модели в визуально удобном виде прописываются связи между объектами данных и их характеристиками.

В концептуальной модели есть принятые обозначения элементов. Сущность или объект обозначать прямоугольником, отношения обозначать ромбом, атрибуты объектов, обозначаются овалом. Если сущность связана с отношением, то их связь обозначается прямой линией со стрелкой. Необязательная связь обозначается пунктирной линией. Мощная связь обозначается двойной линией.

Концептуальная модель базы данных представлена в рисунке 1.4.1



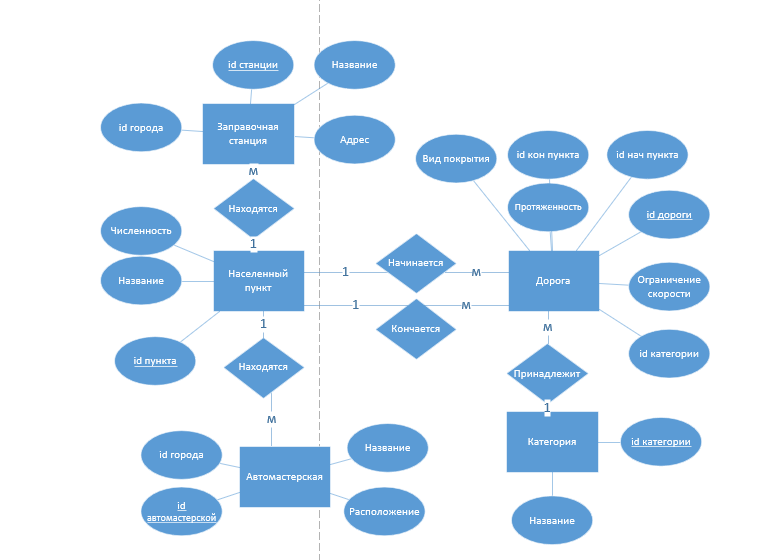


Рисунок 1.4.1 – Концептуальная модель БД

1.5. Логическое моделирование

При логическом моделировании происходит окончательное определение структуры данных, определяются ограничения, накладываемые на эти данные, целью которых является обеспечить целостность данных. Наиболее распространенной моделью данных является реляционная модель. В этой модели данных каждая сущность представляется в виде таблицы.

Логическое моделирование заключается в переходе от концептуальной модели к взаимосвязанным таблицам. Этот переход состоит из следующих шагов:

1. Преобразование сущностей:

- каждая простая сущность становится таблицей.

- каждый атрибут становится столбцом таблицы.

- уникальный идентификатор сущности становится ключом таблицы.

2. Преобразование связи:

- сущности, связанные обязательной связью один к одному можно объединить в одну таблицу.

- связи типа один к одному возможные и связи типа один ко многим реализуются путем переноса ключевых атрибутов таблиц, соответствующих сущностей, стоящих со стороны один в таблице соответствующих сущностей, стоящих со стороны многие.

- связи типа многие ко многим реализуются при помощи промежуточных таблиц, содержащих ключевые атрибуты связываемых таблиц в качестве внешних ключей.

Схема данных – это структура базы данных, описанная на формальном языке, поддерживаемом СУБД (системой управления базы данных). В реляционных базах данных схема определяет таблицы, поля в каждой таблице и ограничения целостности, такие как первичный и внешний ключи.

Схема данных представлена на рисунке 1.5.1.

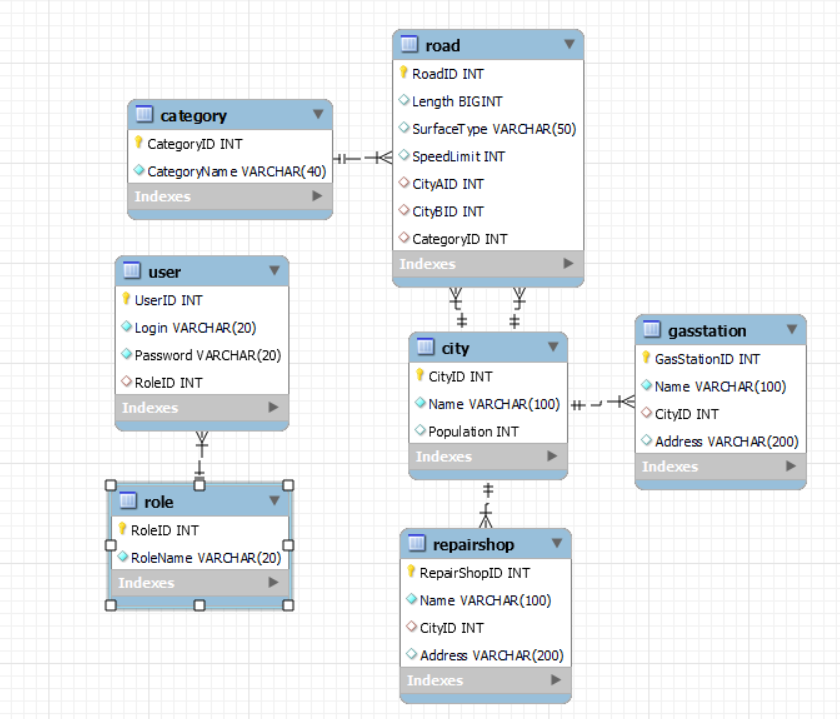


Рисунок 1.5.1 – Схема данных БД

1.6. Описание структуры базы данных

Описание структуры базы данных представлено в таблице 1.6.1.

Таблица 1.6.1 - Описание структуры базы данных

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Имя поля | Описание поля | Тип данных | Размер поля | Тип ключа[[1]](#footnote-1) |
| 1 | 2 | 3 | 4 | 5 |
| Road (Дорога) | | | | |
| RoadID | ID дороги | INT |  | PK |
| Length | Протяженность | BIGINT |  |  |
| SurfaceType | Тип покрытия | NVARCHAR | 50 |  |
| SpeedLimit | Ограничение скорости | INT |  |  |
| CityAID | ID начального города | INT |  | FK |
| CityBID | ID конечного города | INT |  | FK |
| CategoryID | ID категороии дороги | INT |  | FK |
| GasStation (Заправочные станции) | | | | |
| GasStationID | ID заправочной станции | INT |  | PK |
| Name | ID лота | NVARCHAR | 100 |  |
| CityID | ID города | INT |  | FK |
| Address | Адрес | NVARCHAR | 200 |  |
| RepairShop (Авторемонтные пункты) | | | | |
| RepairShopID | ID авторемонтного пункта | INT |  | PK |
| Name | Название | NVARCHAR | 100 |  |

Продолжение таблицы 1.6.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 |
| CityID | ID города | INT |  | FK |
| Address | Адрес | NVARCHAR | 200 |  |
| City (Населенные пункты) | | | | |
| CityID | ID населенного пункта | INT |  | PK |
| Name | Название | NVARCHAR | 100 |  |
| Population | Численность населения | INT |  |  |
| User (Пользователи) | | | | |
| UserID | ID пользователя | INT |  | PK |
| Login | Логин | NVARCHAR | 20 |  |
| Password | Пароль | NVARCHAR | 20 |  |
| RoleID | ID роли | INT |  | FK |
| Category (Категории дороги) | | | | |
| CategoryID | ID категории | INT |  | PK |
| CategoryName | Название категории | NVARCHAR | 40 |  |
| Role (Роли) | | | | |
| RoleID | ID роли | INT |  | PK |
| RoleName | Название роли | NVARCHAR | 20 |  |

1.7. Контрольный пример

Контрольный пример является ручным подсчётом задачи. По составленной программе обрабатываются исходные данные контрольного примера. Полученные результаты сравниваются с известными результатами контрольного примера. При несовпадении результатов производится поиск, исправление ошибок, и снова производится выполнение программы.

Входная информация контрольных примеров представлена в приложении Б.

Выходные данные для контрольных примеров показаны в приложении В.

1.8 Общие требования к программному продукту

Пользователи должны иметь базовые навыки пользования персональным компьютером.

Минимальные требования к техническому обеспечению программного продукта следующие:

* ОС:  Windows 7 Service Pack1/ Windows 7 64Bit Service Pack1/ Windows 8.1 64Bit / Windows 10 64Bit / Mac OS;
* процессор: 2.40 Ггц (четырехъядерный) / AMD Phenom 9850 (четырехъядерный) @ 2.5 Ггц;
* оперативная память: 512МБ (Win 7/Win8 и выше);
* видеокарта: NVIDIA 9800 GT c 512 МБ видеопамяти/ AMD HD 4870 с 1 Гб видеопамяти (DX 9, 10, 10.1);
* жесткий диск: 10 гигабайт свободного места;
* Microsoft DirectX версия 9.0c;

Функциональные возможности программного продукта:

* приложение должно формировать и отображать выходные данные пользователю;
* в приложении должен быть обеспечен просмотр таблиц (справочников) базы данных с возможность добавления, редактирования, удаления данных.

Требования к надежности:

* приложение должно обрабатывать ошибочные действия пользователя и сообщать ему об этом;
* приложение должно обеспечивать контроль входной и выходной информации.

Требования к информационной и программной совместимости: обеспечить работу приложения с таблицами СУБД MySQL Workbench.

2 Экспериментальный раздел

2.1 Описание программы

Программа имеет модульную структуру. При ее запуске выполняется проект на WPF Roads.exe. Программа Roads.exe написана на языке C# в среде программирования Visual Studio 2022 с использованием системы управления базой данных MS SQL Workbench.

Таблица 2.1.1. - Описание модулей

|  |  |
| --- | --- |
| Методы | Назначение |
| 1 | 2 |
| MainWindow.xaml.cs | |
| public MainWindow() | Переход на главную сраницу |
| HomePage.xaml.cs | |
| public HomePage() | Инициализация компонента |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| void AuthOpenBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу авторизации |
| void AboutBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу с информацией о приложении |
| void FeedbackBtn\_MouseDown(object sender, MouseButtonEventArgs e) | Переход на страницу с формой обратной связи |
| private void VseMarBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу с информацией о всех дорогах |
| About .xaml.cs | |
| public MainWindow() | Переход на главную сраницу |
| public About() | Инициализация компонента |
| void WebSite\_MouseDown(object sender, MouseButtonEventArgs e) | Открытие браузера по URL (https://yandex.ru/maps/) |
| AuthPage.xaml.cs | |
| public AuthPage() | Инициализация компонента |
| void BackBtn\_Click(object sender, RoutedEventArgs e) | Возврат на предыдущую страницу |
| void RegistrBtn\_MouseDown(object sender, MouseButtonEventArgs e) | Переход на страницу регистрации |
| void AuthBtn\_Click(object sender, RoutedEventArgs e) | Вызов метода аутентификации пользователя (AuthUserMethod) |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| void AddgasstationBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу добавления заправки |
| void AddrepairshopBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу добавления автомастерской |
| void AddCityBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу добавления города |
| void AddRoadsBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу добавления дорог |
| void EditCityBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу редактирования города |
| void EditGasStationBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу редактирования заправки |
| void EditRepairShopBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу редактирования автомастерской |
| void EditRoadBtn\_Click(object sender, RoutedEventArgs e) | Переход на страницу редактирования дороги |
| void BackBtn\_Click(object sender, RoutedEventArgs e) | Возврат на предыдущую страницу |
| void RegistrBtn\_MouseDown(object sender, MouseButtonEventArgs e) | Переход на страницу регистрации |
| FeedbackPage.xaml.cs | |
| public FeedbackPage() | Инициализация компонента |
| void EmailBtn\_Click(object sender, RoutedEventArgs e) | Отправка электронной почты с использованием SMTP-клиента |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| FullInfoRoad.xaml.cs | |
| public MainWindow() | Переход на главную сраницу |
| FullInfoRoad(double roadLength, int speedLimit) | Инициализация компонента и вычисление времени в пути на основе длины дороги и ограничения скорости |
| double CalculateAverageSpeed(int speedLimit) | Вычисление средней скорости (90% от ограничения скорости) |
| Tuple<int,int> CalculateTimeInTransit(double roadLength, double averageSpeed) | Вычисление времени в пути в часах и минутах на основе длины дороги и средней скорости |
| MainPage.xaml.cs | |
| public MainPage() | Инициализация компонента, заполнение ComboBox городов из базы данных |
| void FindRouteButton\_Click(object sender, RoutedEventArgs e) | Поиск маршрута между выбранными городами, отображение деталей маршрута в ListBox |
| void DownloadReportButton\_Click(object sender, RoutedEventArgs e) | Создание и сохранение отчета о маршруте в формате PDF |

|  |  |
| --- | --- |
| PathFinder(TestDbContext context) | Конструктор класса PathFinder, инициализирует объект контекста базы данных TestDbContext. |
| CalculateAverageSpeed(int speedLimit) | Метод, вычисляющий среднюю скорость на основе предела скорости. Возвращает значение типа double, которое является 90% от переданного предела скорости. |
| CalculateTimeInTransit(double roadLength, double averageSpeed) | Метод, расчитывающий время в пути на основе длины дороги и средней скорости. Возвращает кортеж из двух целых чисел, представляющих часы и минуты в пути. |
| FindPath(int startCityId, int endCityId) | Метод для поиска пути между двумя городами по их идентификаторам. Возвращает список RouteSegment, представляющий найденный маршрут или null, если путь не найден. |
| RegistrPage.xaml.cs |  |
| RegistrPage() | Инициализирует страницу регистрации и создает сервис Servise. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| PathFinder.cs | |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |

|  |  |
| --- | --- |
| RegistrPage() | Инициализирует страницу регистрации и создает сервис Servise. |
| BackBtn\_Click(object sender, RoutedEventArgs e) | Обрабатывает нажатие кнопки "Назад", возвращая пользователя на предыдущую страницу. |
| RegistrBtn\_Click(object sender, RoutedEventArgs e) | Обрабатывает нажатие кнопки "Регистрация", вызывая метод AddUserMethod() для добавления нового пользователя. |

|  |  |
| --- | --- |
| AddUserMethod() | Добавляет нового пользователя на основе введенных данных, проверяя их корректность. В случае успеха переходит на страницу авторизации. Обрабатывает ошибки при добавлении. |
| IsValidEmail(string email) | Проверяет корректность формата email с использованием регулярного выражения. Возвращает true, если email верный, иначе false. |
| RoadCart.xaml.cs | |
| RoadCart() | Конструктор, инициализирующий элемент управления. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |

|  |  |
| --- | --- |
| UserControl\_MouseDoubleClick(object sender, MouseButtonEventArgs e) | Обработчик двойного щелчка, открывающий модальное окно FullInfoRoad с информацией о дороге, если Road доступен в DataContext. |
| Servise.cs | |
| GetCities() | Возвращает список всех городов из базы данных. |
| GetCategories() | Возвращает список всех категорий из базы данных. |
| GetRepairshops() | Возвращает список всех автомастерских из базы данных. |
| GetGasstations() | Возвращает список всех заправок из базы данных. |

|  |  |
| --- | --- |
| GetRoads() | Возвращает список всех дорог из базы данных, включая данные о связанных с ними городах, автомастерских, заправках и категориях. |
| Addrepairshop(Repairshop repairshop) | Добавляет новую автомастерскую в базу данных. |
| Addcity(City city) | Добавляет новый город в базу данных. |
| Addgasstation(Gasstation gasstation) | Добавляет новую заправку в базу данных. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| Addcity(City city) | Добавляет новый город в базу данных. |
| Addgasstation(Gasstation gasstation) | Добавляет новую заправку в базу данных. |
| Addroad(Road road) | Добавляет новую дорогу в базу данных. |
| Adduser(User user) | Добавляет нового пользователя в базу данных. |
| VseRoad.xaml.cs |
| VseRoad() | Конструктор страницы, инициализирует компоненты страницы, загружает дорожные данные и категории, проверяет роль пользователя. |
| UserChek() | Проверяет роль пользователя для управления доступностью кнопки на основе статуса авторизации. |
| LoadRoads() | Загружает список дорог из базы данных с помощью сервиса Servise и применяет текущие фильтры к ним. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| LoadCategories() | Загружает список категорий дорог из базы данных с помощью сервиса Servise, добавляет категорию "Все" в начало списка и устанавливает их в качестве фильтра. |
| ApplyFilters() | Применяет текущие фильтры (категория, поиск, сортировка) к списку дорог и обновляет отображаемые данные в productListView. |
| VseMarshBtn\_Click() | Обработчик нажатия кнопки "На главную", переходит на главную страницу приложения. |
| Button\_Click() | Обработчик нажатия кнопки "Назад", возвращает пользователя на предыдущую страницу и сбрасывает роль пользователя. |
| Button\_Click\_2() | Обработчик нажатия кнопки "Редактировать дороги", переходит на страницу редактирования дорог. |
| Page\_Loaded() | Обработчик события загрузки страницы, повторно загружает список дорог при каждой загрузке страницы. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| Filter\_SelectionChanged\_1() | Обработчик изменения выбора категории в выпадающем списке, обновляет текущую категорию и применяет фильтры к списку дорог. |
| Source\_SelectionChanged() | Обработчик изменения текста в поле поиска, обновляет текущий поиск и применяет фильтры к списку дорог. |
| AddCityPage.xaml.cs | |
| AddCityPage() | Конструктор страницы, инициализирует компоненты страницы. |
| AddButton\_Click() | Обработчик нажатия кнопки "Добавить", создает новый объект City, заполняет его данными из текстовых полей PopulationTextBox и NameTextBox, добавляет город в базу данных с помощью сервиса servise, выводит сообщение об успехе. |
| ExitButton\_Click() | Обработчик нажатия кнопки "Выход", возвращает пользователя на предыдущую страницу. |
| AddRoadsPage.xaml.cs | |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| ExitButton\_Click() | Обработчик нажатия кнопки "Выход", возвращает пользователя на предыдущую страницу. |
| AddButton\_Click() | Обработчик нажатия кнопки "Добавить", создает новую дорогу (Road) и добавляет ее в базу данных через servise. Затем выводит сообщение об успешном добавлении. |
| GasstationPage.xaml.cs | |
| GasstationPage() | Конструктор страницы, инициализирует компоненты и устанавливает источник данных для CityComboBox из базы через Servise.GetCities(). |
| AddButton\_Click() | Обработчик кнопки "Добавить". Создает новую заправочную станцию (Gasstation), заполняет ее данными из полей и выбранного города, добавляет в базу через servise и показывает сообщение об успехе. |
| ExitButton\_Click() | Обработчик кнопки "Выход". Возвращает пользователя на предыдущую страницу. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| RepairshopAddPage.xaml.cs | |
| RepairshopAddPage() | Конструктор страницы, инициализирует компоненты и устанавливает список городов в CityComboBox. |
| AddButton\_Click() | Обработчик нажатия кнопки "Добавить". Создает новый объект Repairshop с данными из полей и выпадающего списка, добавляет в базу данных и выводит сообщение об успехе. |
| ExitButton\_Click() | Обработчик нажатия кнопки "Выход". Возвращает пользователя на предыдущую страницу. |
| EditCityPage.xaml.cs | |
| EditCityPage() | Конструктор страницы, инициализирует компоненты и загружает список городов в CityComboBox. |
| AddButton\_Click() | Обновляет выбранный город данными из полей, сохраняет изменения в базе и выводит сообщение об успехе. |
| ExitButton\_Click() | Возвращает пользователя на предыдущую страницу. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| EditGasStationPage.xaml.cs | |
| EditGasStationPage() | Конструктор страницы, инициализирует компоненты и загружает список заправочных станций и городов в соответствующие выпадающие списки. |
| AdressComboBox\_SelectionChanged() | Заполняет поля данными выбранной заправочной станции при изменении выбора в AdressComboBox. |
| AddButton\_Click() | Обновляет выбранную заправочную станцию данными из полей, сохраняет изменения в базе и выводит сообщение об успехе. |
| ExitButton\_Click() | Возвращает пользователя на предыдущую страницу. |
| RefreshGasStations() | Обновляет список заправочных станций в AdressComboBox из базы данных. |
| EditRepairShopPage.xaml.cs | |
| EditRepairShopPage() | Конструктор страницы, инициализирует компоненты и загружает список автосервисов и городов в соответствующие выпадающие списки. |

Продолжение таблицы 2.1.1

|  |  |
| --- | --- |
| 1 | 2 |
| AddButton\_Click() | Обновляет выбранный автосервис данными из полей, сохраняет изменения в базе и выводит сообщение об успехе. |
| ExitButton\_Click() | Возвращает пользователя на предыдущую страницу. |
| RefreshGasStations() | Обновляет список автосервисов в AdressComboBox из базы данных. |
| EditRoadPage.xaml.cs | |
| public HomePage() | Инициализация компонента |
| EditRoadPage() | Конструктор страницы, инициализирует компоненты и загружает список дорог, городов и категорий в соответствующие выпадающие списки. |
| RoadComboBox\_SelectionChanged() | Заполняет поля данными выбранного маршрута при изменении выбора в RoadComboBox. |
| UpdateButton\_Click() | Обновляет выбранный маршрут данными из полей, сохраняет изменения в базе данных и выводит сообщение об успехе. |
| ExitButton\_Click() | Выход. |

Код программы представлен в приложении Г.

2.2 Протокол тестирования программного продукта

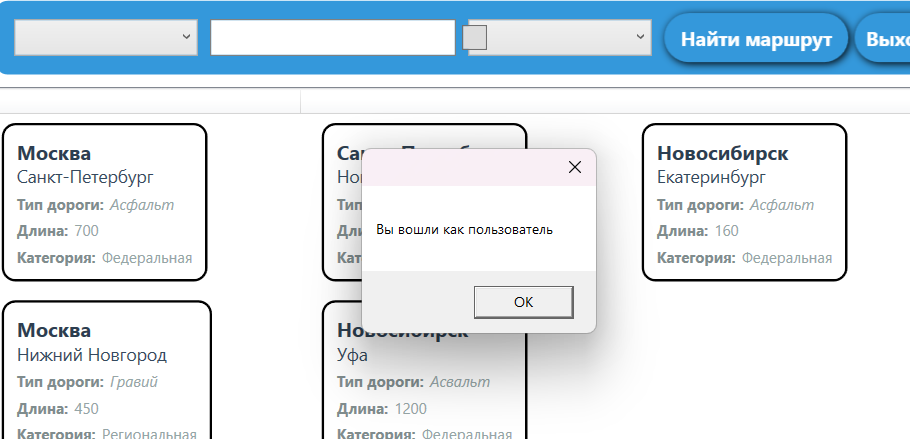
В протоколе тестирования отражаются:

− тестирование на корректных данных;

− тестирование на некорректных данных;

− тестировании продукта на данных контрольного примера.

Тестирование авторизации пользователя при корректных данных, ожидаемое сообщение «Вы вошли» (рисунок 2.2.1).

Рисунок 2.2.1 – Сообщение об успешной авторизации

Тестирование авторизации пользователя при некорректных данных, ожидаемое сообщение «Введите корректные данные» (рисунок 2.2.2).

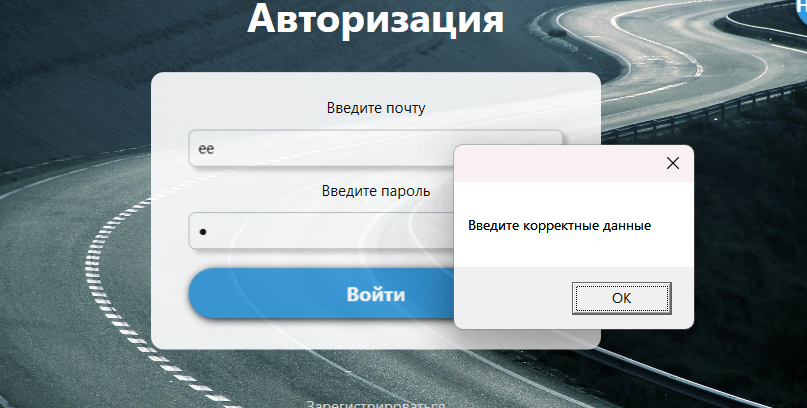


Рисунок 2.2.2 – Сообщение об ошибке при авторизации

Тестирование обратной связи при корректных данных, ожидаемое сообщение «Письмо успешно отправлено!» (рисунок 2.2.3).

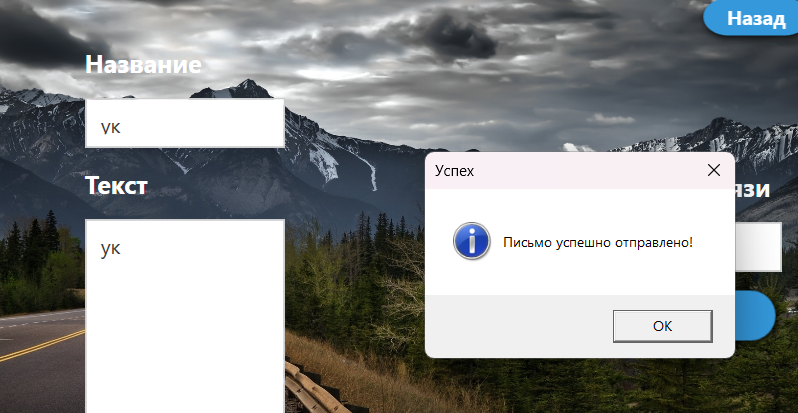


Рисунок 2.2.3 – Сообщение об успешной отправке письма

Тестирование регистрации при корректных данных, ожидаемое сообщение «Вы зарегистрировались» (рисунок 2.2.4).

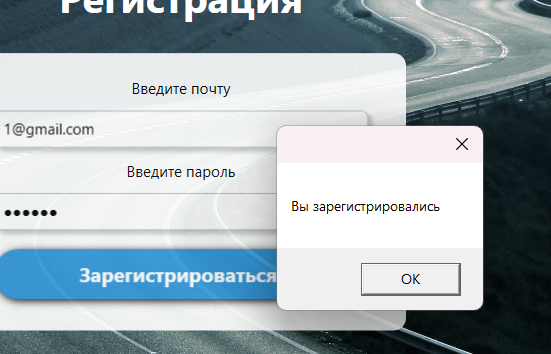


Рисунок 2.2.4 – Сообщение об успешной регистрации

Тестирование регистрации при не корректной почты, ожидаемое сообщение «Некорректный формат почты» (рисунок 2.2.5).

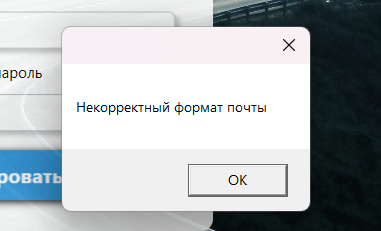


Рисунок 2.2.5 – Сообщение о некорректном форматы почты

Тестирование регистрации при не корректном пароле, ожидаемое сообщение «Пароль должен быть больше 6 символов» (рисунок 2.2.6).

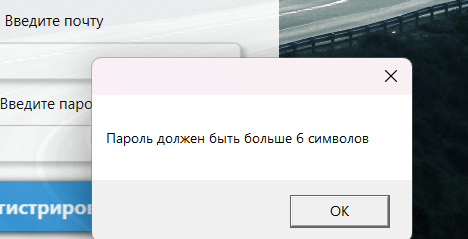


Рисунок 2.2.6 – Сообщение о некорректном пароле

Тестирование сохранении отчета, ожидаемое сообщение «Отчет сохранен как» (рисунок 2.2.7).

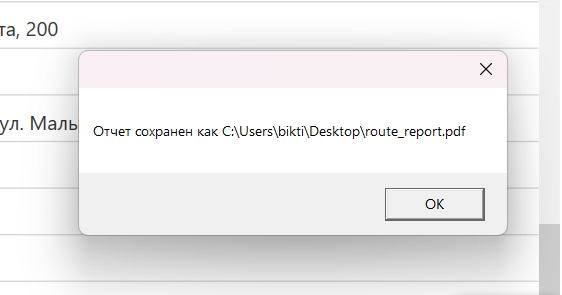
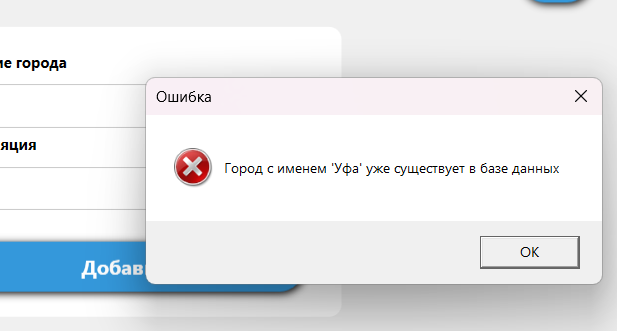


Рисунок 2.2.7 – Сообщение о сохранении отчета

Тестирование добавления города с одинаковым названием, ожидаемое сообщение «Город с именем 'Уфа ' уже существует в базе данных» (рисунок 2.2.8).

 Рисунок 2.2.8 – Сообщение об ошибке при добавления города

Тестирование добавления города при некорректном вводе популяции, ожидаемое сообщение «Ошибка ввода численного значения для населения» (рисунок 2.2.9).

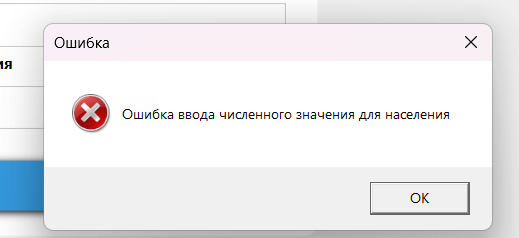


Рисунок 2.2.9 – Сообщение об ошибке при не корректном значении популяции

Тестирование добавления города с корректными данными, ожидаемое сообщение «Город успешно добавлен» (рисунок 2.2.10).

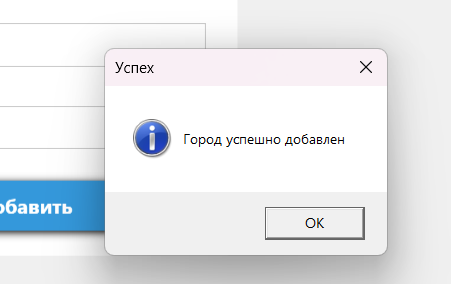


Рисунок 2.2.10 – Сообщение об успешном добавлении города

Тестирование добавления дороги с корректными данными, ожидаемое сообщение «Дорога успешно добавлена» (рисунок 2.2.11).

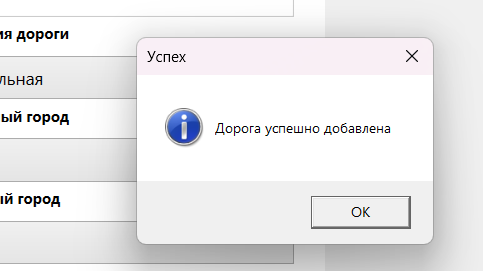


Рисунок 2.2.11 – Сообщение об успешном добавлении дороги

Тестирование добавления дороги с некорректными данными, ожидаемое сообщение «Дорога с такими параметрами уже существует» (рисунок 2.2.12-15).

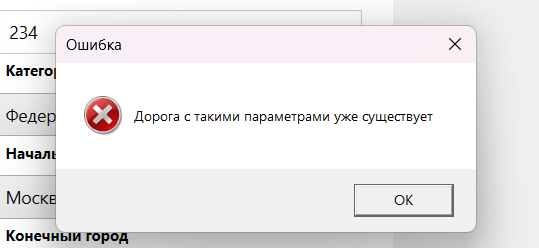


Рисунок 2.2.12 – Сообщение об ошибке при добавлении дороги

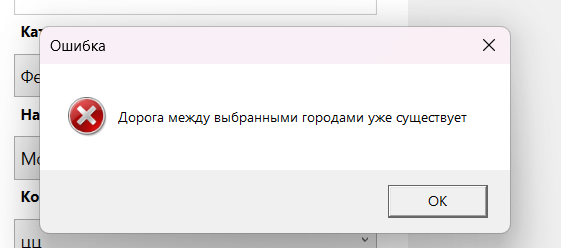


Рисунок 2.2.13 – Сообщение об ошибке при добавлении дороги

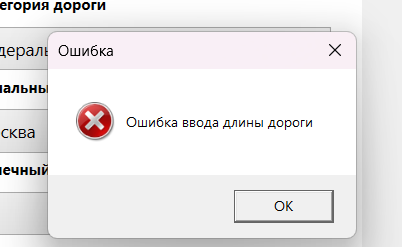


Рисунок 2.2.14 – Сообщение об ошибке при добавлении дороги

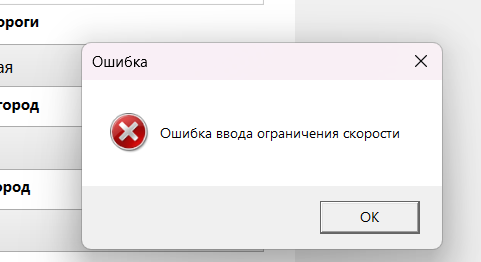


Рисунок 2.2.15 – Сообщение об ошибке при добавлении дороги

Тестирование добавления заправки с некорректными данными, ожидаемое сообщение «Заправка с адресом “1”уже существует.» (рисунок 2.2.16).

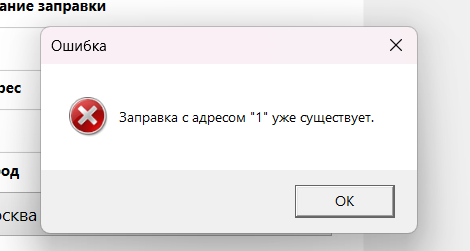


Рисунок 2.2.16 – Сообщение об ошибке при добавлении заправки

Тестирование добавления заправки с корректными данными, ожидаемое сообщение «Вы добавили заправку» (рисунок 2.2.17).

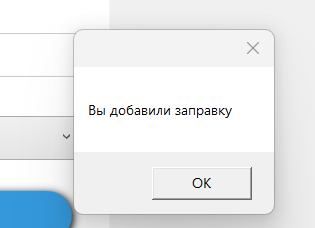


Рисунок 2.2.17 – Сообщение об успешном при добавлении заправки

Тестирование добавления мастерской с некорректными данными, ожидаемое сообщение «Мастерская с адресом “3333” уже существует.» (рисунок 2.2.18).

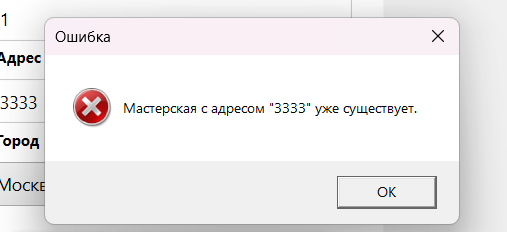


Рисунок 2.2.18 – Сообщение об ошибке при добавлении мастерской

Тестирование добавления мастерской с корректными данными, ожидаемое сообщение «Вы добавили мастерскую» (рисунок 2.2.19).

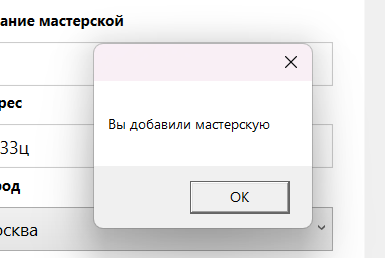


Рисунок 2.2.19 – Сообщение об успешном при добавлении мастерской

При тестировании продукта на данных контрольного примера, результаты полностью совпадают с ожидаемыми (рисунки 2.2.20 – 21)

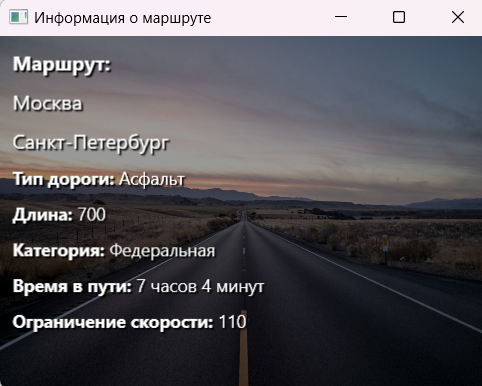


Рисунок 2.2.20 – Информация о маршруте

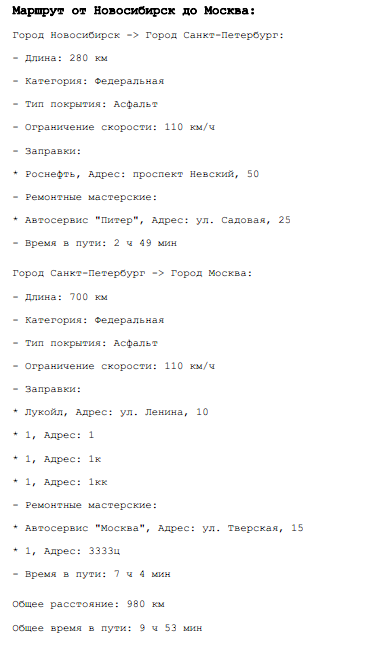


Рисунок 2.2.21 – Информация о сформированном маршруте

2.3 Руководство пользователя

Назначение системы

Программа «Roads» предназначена для удобного просмотра, поиска дорог и формирования маршрутов.

Основной целью данной информационной системы является формирование новых маршрутов, а также предоставление списка доступных маршрутов.

Условия применения системы

Программное обеспечение разрабатывается для персональной вычислительной техники со следующими характеристиками:

* Microsoft Windows 7 / 8 / 10 / 11;
* процессор 1 ГГц;
* 128 МБ ОЗУ;
* 60 МБ свободного пространства на диске;
* разрешение экрана монитора не менее 1920 × 1080;
* дополнительное оборудование
* принтер.

Программа «Roads» предназначена для пользователей, имеющих как минимум первоначальные навыки работы с графической операционной системой, на которой будет запускаться данная программа.

Подготовка системы к работе

Для запуска программы необходимо запустить приложение Roads.exe из каталога, в котором установлен данный программный продукт. После этого открывается начальное окно программы (рисунок 2.3.1).

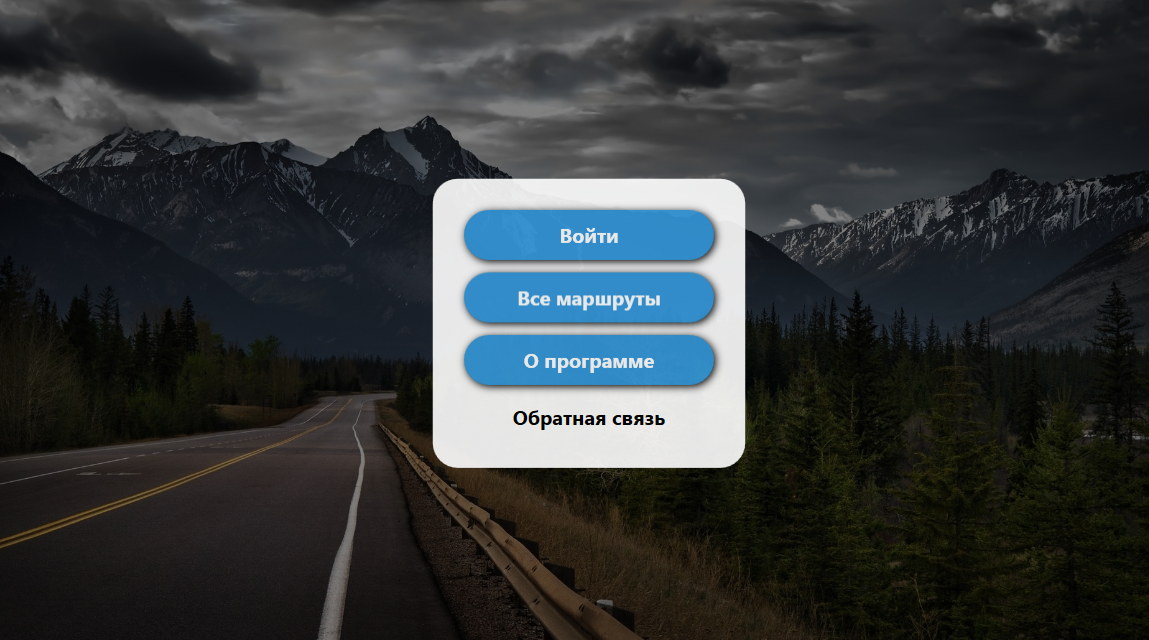


Рисунок 2.3.1 – Начальное окно программы

После запуска программы мы видим 4 кнопки, кнопка “Войти” (рисунок 2.3.2) переносит нас на страницу авторизации, “Все маршруты” (рисунок 2.3.3) на страницу с маршрутами, “О программе” (рисунок 2.3.4) на страницу о программе, “Обратная связь” (рисунок 2.3.5) на окно для отправки письма на почту

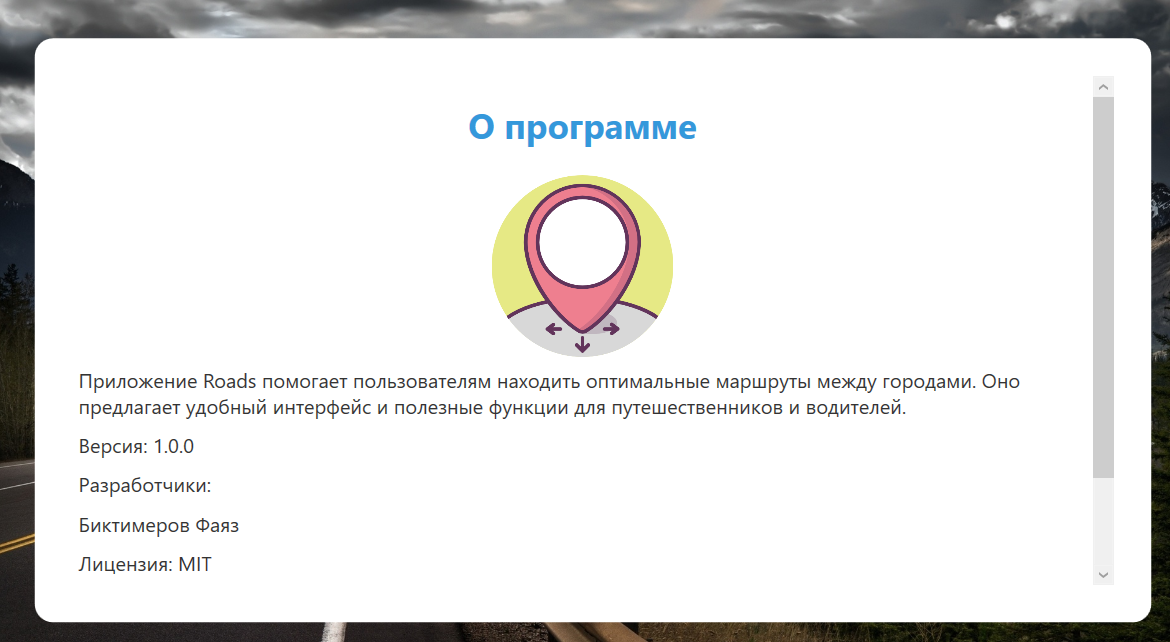


Рисунок 2.3.4 – Окно о программе

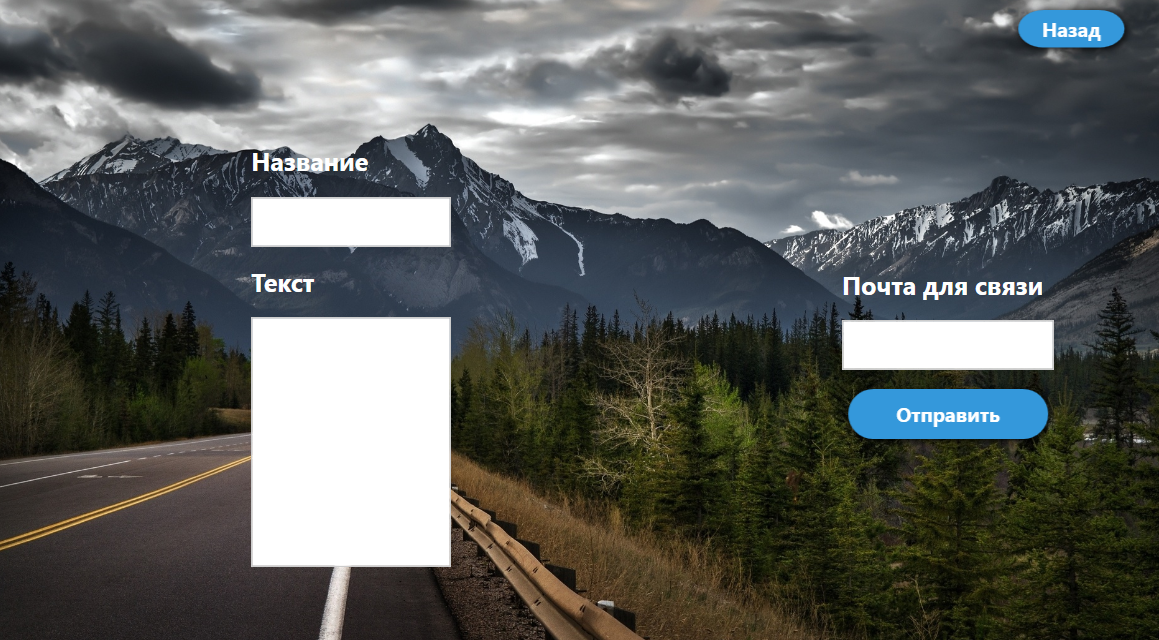


Рисунок 2.3.5 – Окно обратная связь

Описание операций

После ввода правильного пароля и логина пользователь попадает на страницу со всеми маршрутами и ему доступна функция составления маршрута. Если в систему вошел администратор, то ему будет доступна админ панель (рисунок 2.3.6).

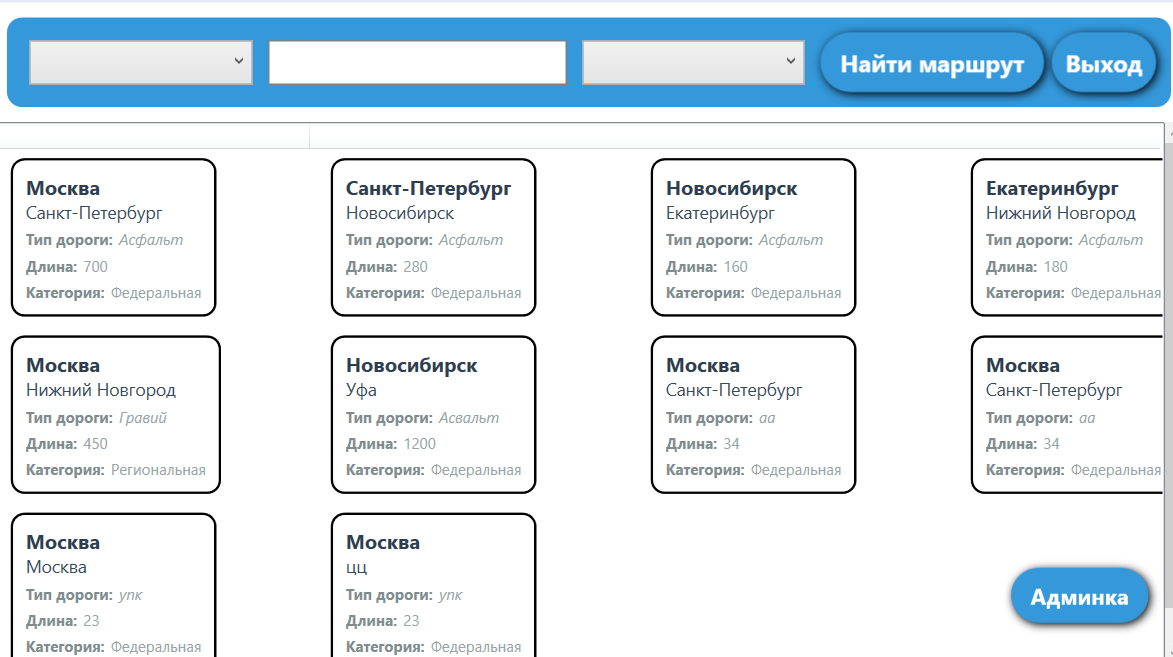


Рисунок 2.3.6 – Окно после входа администратора

В окне авторизации пользователь может нажать на кнопку “Зарегистрироваться” и попадет на окно регистрации (рисунок 2.3.7).



Рисунок 2.3.7 – Окно регистрации

После успешного входа пользователю доступна функция поиска маршрута (рисунок 2.3.8).

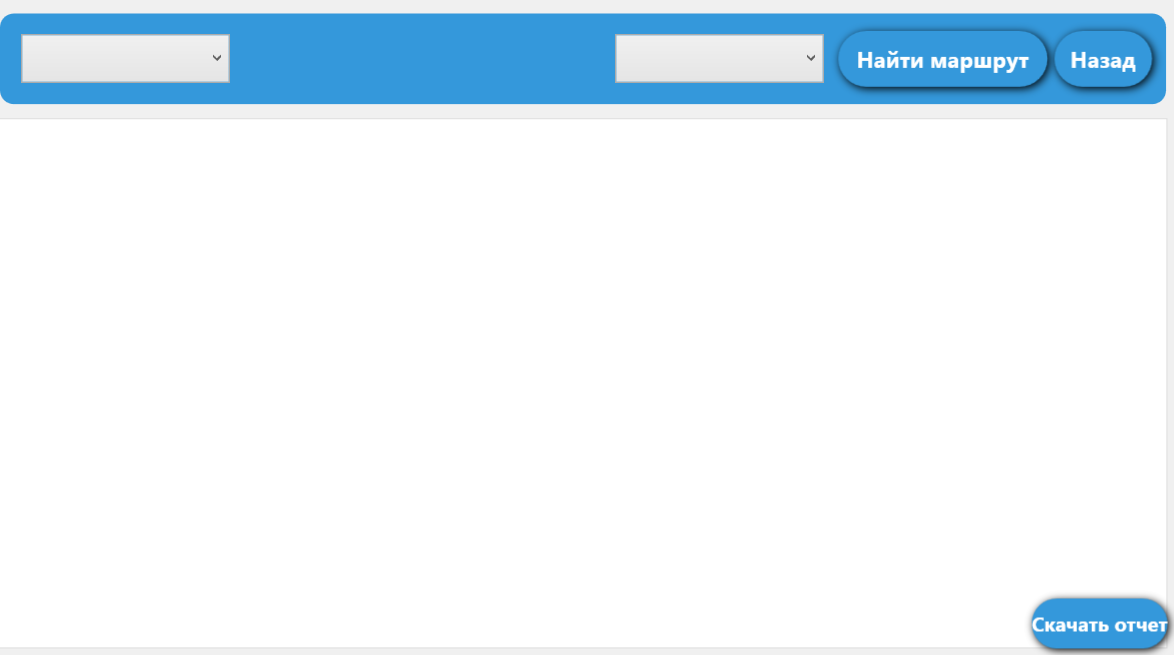


Рисунок 2.3.8 – Окно поиска маршрута

Пользователь в первом ComboBox выбирает начальный город, а во втором конечный и после нажатия кнопки “Найти маршрут” выводиться информация о маршруте (рисунок 2.3.9).

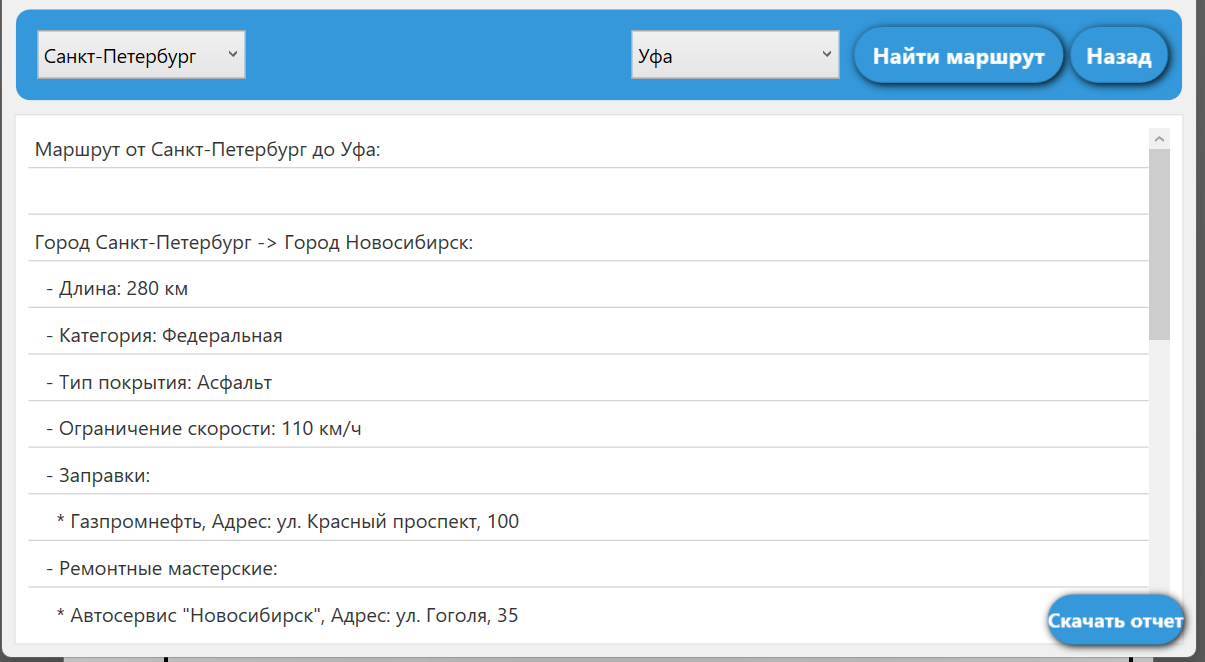


Рисунок 2.3.9 – Найденный маршрут

Найденный маршрут можно сохранить в pdf формате по нажатию кнопки “Скачать отчет” для дальнейшего использования (рисунок 2.3.10).

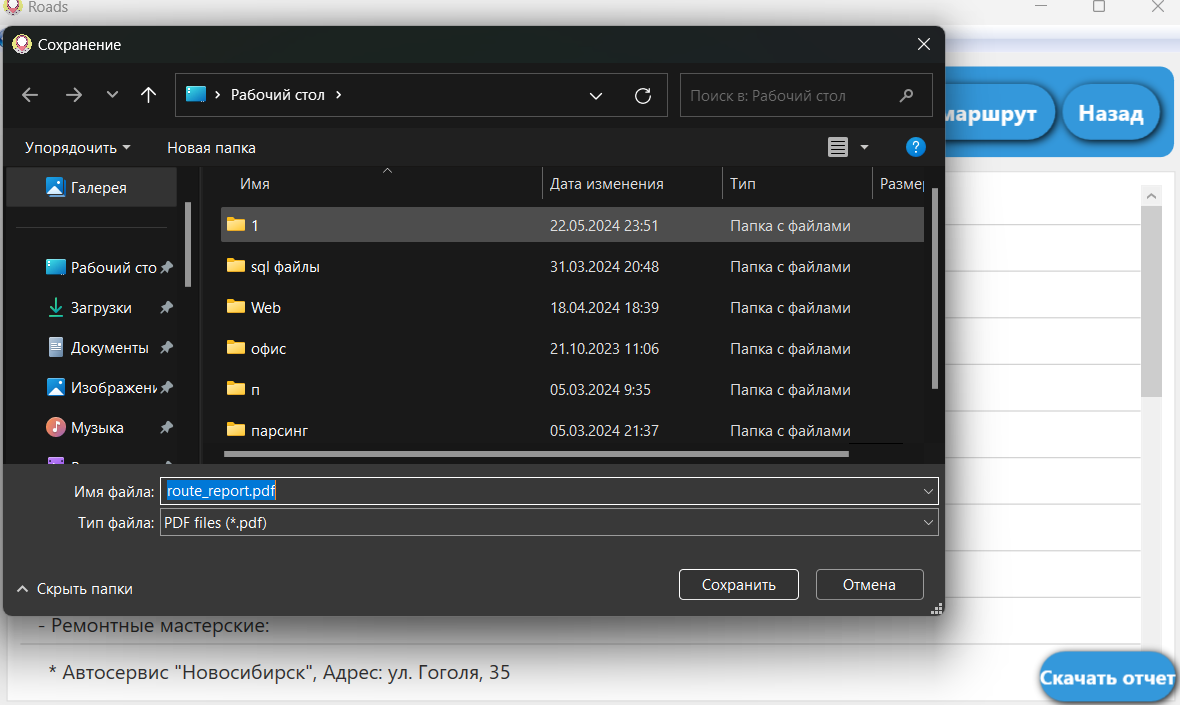


Рисунок 2.3.10 – Окно сохранения маршрута

Администратору доступна админ панель для добавления и редактирования маршрутов и расположенных на нем объектов (рисунок 2.3.11).

****

Рисунок 2.3.11 – Окно с функционалом администратора

Все окна которые доступны администратору (рисунок 2.3.12-19).



Рисунок 2.3.12 – Окно добавления города

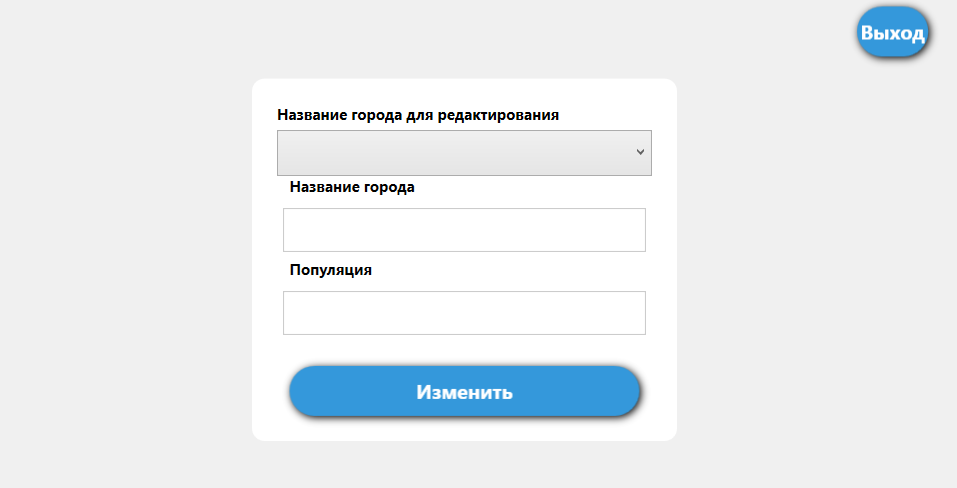


Рисунок 2.3.13 – Окно редактирования города

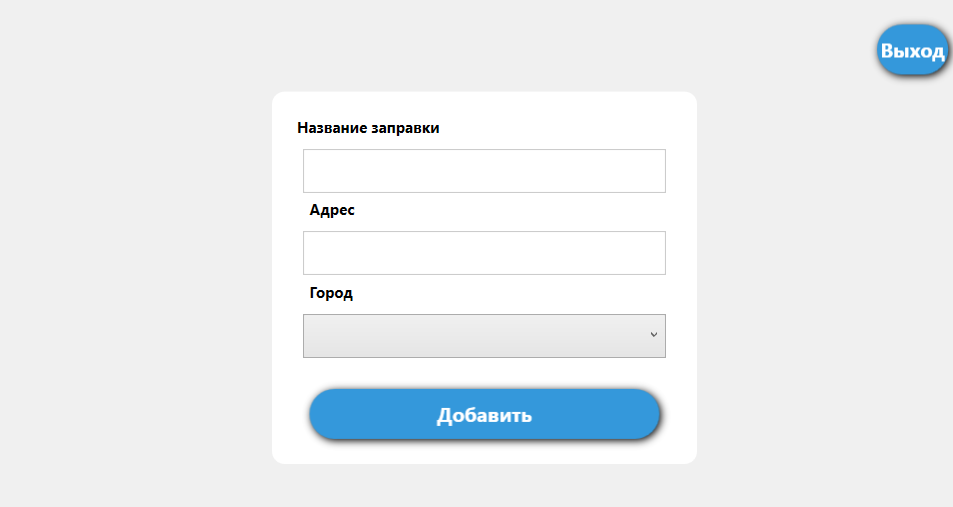


Рисунок 2.3.14 – Окно добавления заправки

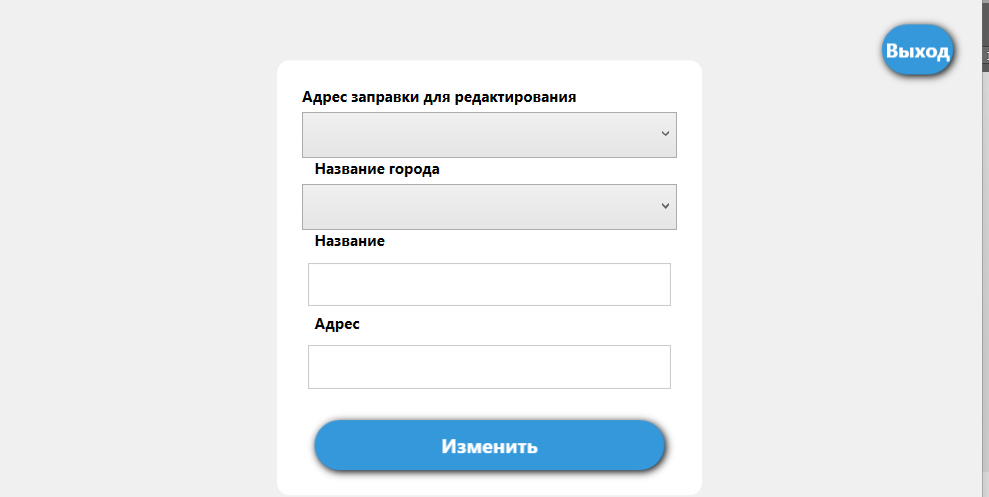


Рисунок 2.3.15 – Окно редактирования заправки

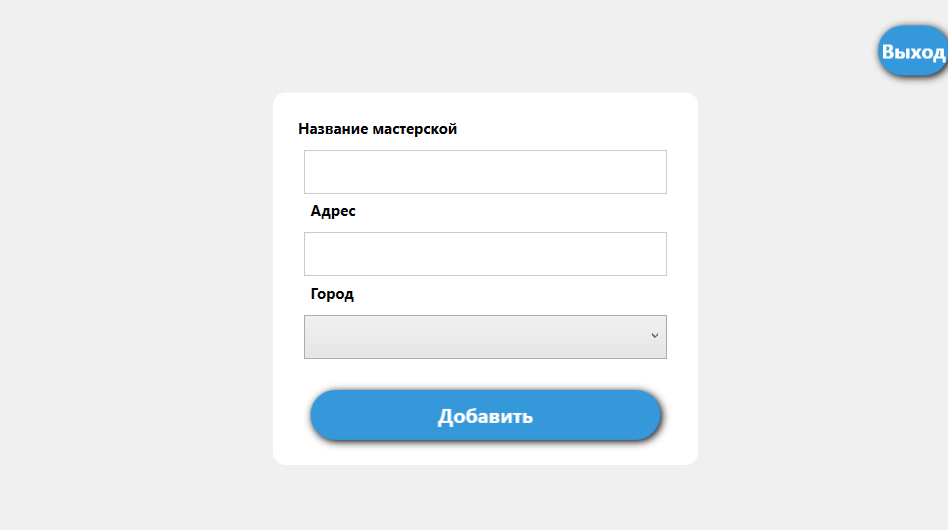


Рисунок 2.3.16 – Окно добавления мастерской

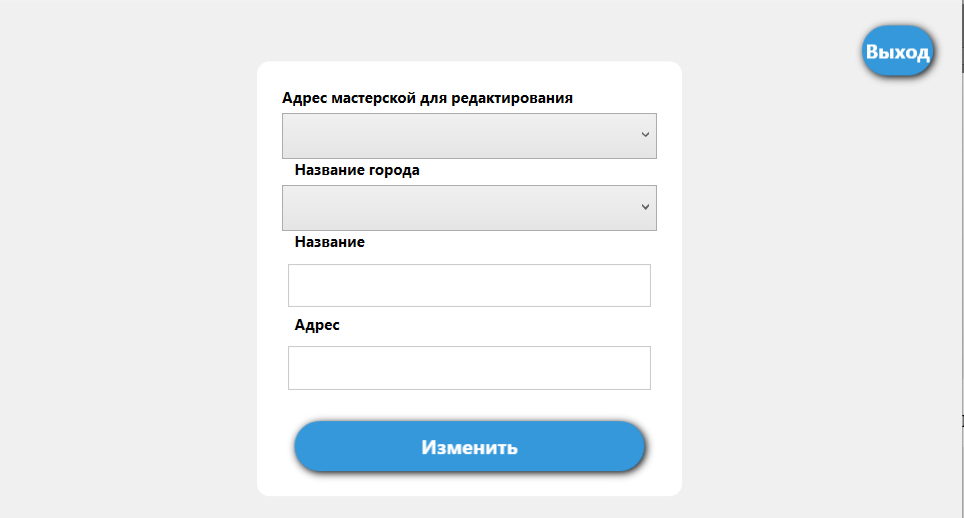


Рисунок 2.3.17 – Окно редактирования мастерской



Рисунок 2.3.18 – Окно добавления маршрута

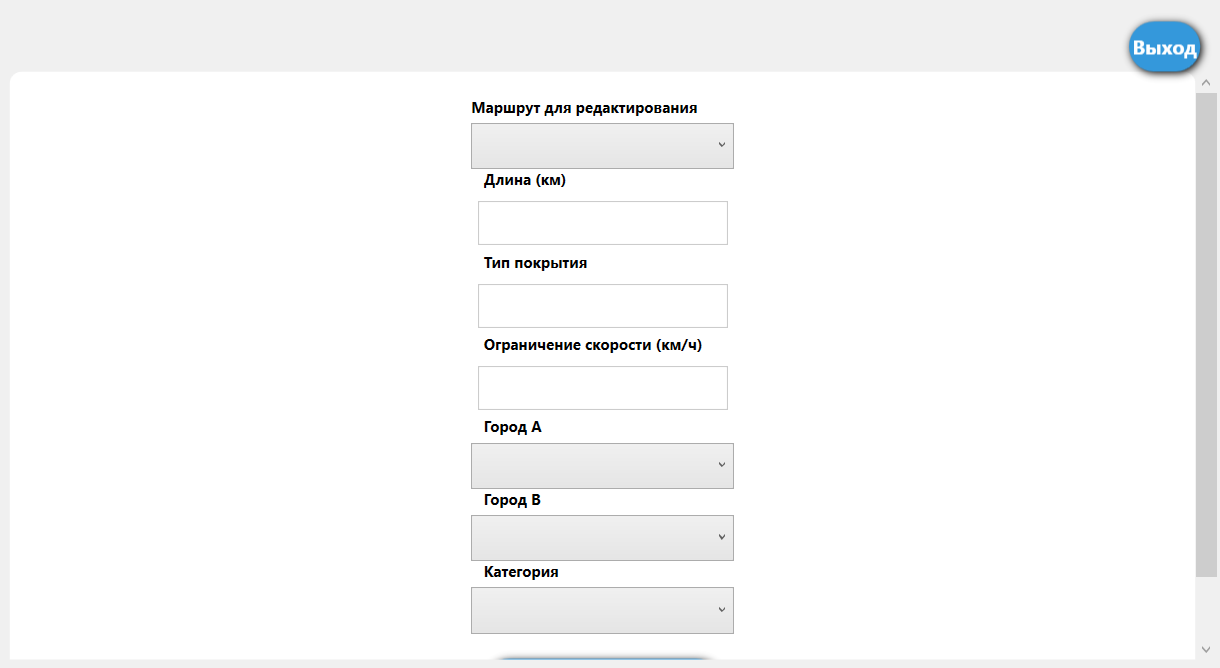


Рисунок 2.3.19 – Окно редактирования маршрута

Рисунок 2.3.14 – Окно добавления заправки

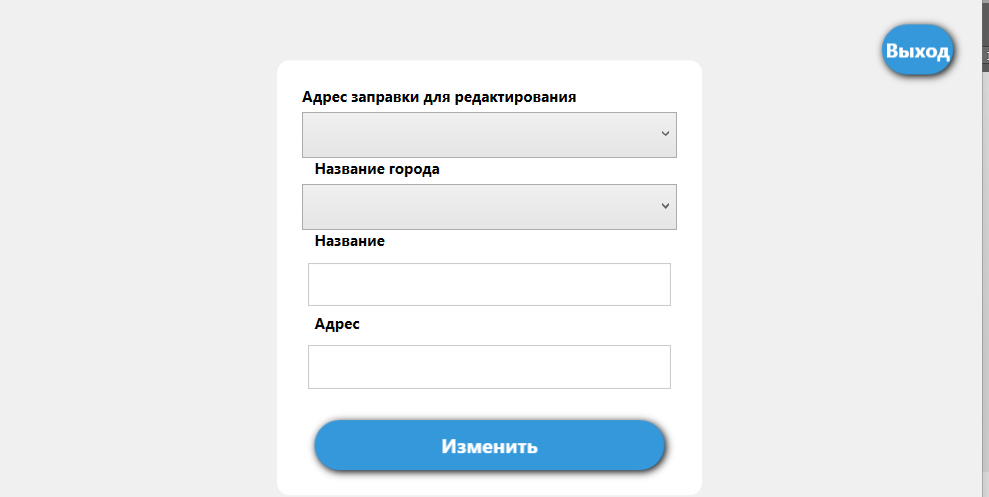


Рисунок 2.3.15 – Окно редактирования заправки

ЗАКЛЮЧЕНИЕ

В процессе выполнения курсового проекта были разработаны структура и алгоритм работы WPF-приложения «Автомобильные дороги».

При этом были изучены особенности реализации компонентов WPF для построения клиентских приложений с визуально привлекательными возможностями взаимодействия с пользователем.

Результатом работы стало создание WPF-приложения для просмотра, поиска дороги и формирования маршрута,.

WPF-приложение написано на языке C# в среде разработки Visual Studio 2022 с использованием языка разметки XAML и системы управления базой данных MS SQL Server 2022.

Были проведены опытная эксплуатация и отладочное тестирование WPF приложения. По результатам отладочного тестирования были устранены некоторые недостатки, в частности были обнаружены и исправлены неточности в реализации алгоритма: усовершенствован контроль на входные данные и отформатирован вывод документов. После этого было написано руководство пользователя.

С помощью приложения на основании данных контрольного примера были получены результаты, которые полностью совпадают с выходной информацией контрольного примера

Шаблоны выходных документов

Рисунок A.1 Карточка маршрута

|  |
| --- |
| {Начальный город}  {Конечный город}  Тип дороги {}  Длинна {}  Категория {} |

|  |
| --- |
| {Начальный город}  {Конечный город}  Тип дороги {}  Длинна {}  Категория {} |

Рисунок A.2 Полная информация о маршруте

|  |
| --- |
| Маршрут  {Начальный город}  {Конечный город}  Тип дороги: {}  Длинна: {}  Категория: {}  Время в пути {}  Ограничения скорости {} |

Рисунок A.3

Маршрут от {Начальный город} до {Конечный город}:

Город {Начальный город} -> Город {Конечный город}:

- Длина: {} км

- Категория: {}

- Тип покрытия: {}

- Ограничение скорости: {}км/ч

- Заправки:

\*

- Ремонтные мастерские:

\*

- Время в пути: 2 ч 49 мин

Город {Начальный город} -> Город {Конечный город}:

- Длина: {} км

- Категория: {}

- Тип покрытия: {}

- Ограничение скорости: {} км/ч

- Заправки:

\*

- Ремонтные мастерские:

\*

- Время в пути {}

Общее расстояние: {} км

Общее время в пути: {}

Входные данные контрольного примера

|  |  |  |
| --- | --- | --- |
| ID города | Название | Население |
| 1 | Москва | 12506468 |
| 2 | Санкт-Петербург | 5384342 |
| 3 | Новосибирск | 1612833 |
| 4 | Екатеринбург | 1493749 |
| 5 | Нижний Новгород | 1252236 |

**Таблица Б.1 – Города**

**Таблица Б.2 – Дороги**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID дороги | Длина (км) | Тип покрытия | Ограничение скорости | ID города A | ID города B | ID категории |
| 1 | 700.00 | Асфальт | 110 | 1 | 2 | 1 |
| 2 | 280.00 | Асфальт | 110 | 2 | 3 | 1 |
| 3 | 160.00 | Асфальт | 110 | 3 | 4 | 1 |
| 4 | 180.00 | Асфальт | 110 | 4 | 5 | 1 |
| 5 | 450.00 | Гравий | 90 | 1 | 5 | 2 |

**Таблица Б.3 – Заправки**

|  |  |  |  |
| --- | --- | --- | --- |
| ID АЗС | Название | ID города | Адрес |
| 1 | Лукойл | 1 | ул. Ленина, 10 |
| 2 | Роснефть | 2 | проспект Невский, 50 |
| 3 | Газпромнефть | 3 | ул. Красный проспект, 100 |
| 4 | Татнефть | 4 | ул. Карла Либкнехта, 200 |
| 5 | Shell | 5 | ул. Большая Покровская, 300 |

**Таблица Б.4 – Мастерские**

|  |  |  |  |
| --- | --- | --- | --- |
| ID автосервиса | Название | ID города | Адрес |
| 1 | Автосервис "Москва" | 1 | ул. Тверская, 15 |
| 2 | Автосервис "Питер" | 2 | ул. Садовая, 25 |
| 3 | Автосервис "Новосибирск" | 3 | ул. Гоголя, 35 |
| 4 | Автосервис "Екатеринбург" | 4 | ул. Малышева, 45 |
| 5 | Автосервис "Нижний" | 5 | ул. Рождественская, 55 |

Выходные данные контрольного примера

Документ для маршрута через три города(таблица В.1):

**Таблица В.1**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Участок маршрута | Длина (км) | Категория | Тип покрытия | Ограничение скорости (км/ч) | Заправки | Ремонтные мастерские | Время в пути |
| Екатеринбург -> Новосибирск | 160 | Федеральная | Асфальт | 110 | Газпромнефть, ул. Красный проспект, 100 | Автосервис "Новосибирск", ул. Гоголя, 35 | 1 ч 36 мин |
| Новосибирск -> Уфа | 1200 | Федеральная | Асфальт | 120 | - | - | 11 ч 6 мин |
| Итого | 1360 | - | - | - | - | - | 12 ч 42 мин |

Документ для маршрута через два города(таблица В.2):

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Участок маршрута | Длина (км) | Категория | Тип покрытия | Ограничение скорости (км/ч) | Заправки | Ремонтные мастерские | Время в пути |
| Петербург | 700 | Федеральная | Асфальт | 110 | 7 ч 4 мин | Петербург | 700 |

**Таблица В.2**

using System.Collections.Generic;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace Roads.Pages

{

public partial class VseRoad : Page

{

private List<Road> roads;

private List<Category> categories;

private string currentFilter = "Все";

private string currentSort = "";

private string currentSearch = "";

public VseRoad()

{

InitializeComponent();

LoadRoads();

LoadCategories();

UserChek();

}

private void UserChek()

{

if(AuthPage.UserRole == 1.ToString())

{

AdminPanelBtn.IsEnabled = false;

VseMarshBtn.IsEnabled = false;

}

else if (AuthPage.UserRole == 2.ToString())

{

AdminPanelBtn.IsEnabled = true;

VseMarshBtn.IsEnabled = true;

}

else

{

VseMarshBtn.IsEnabled = false;

AdminPanelBtn.IsEnabled = false;

}

}

private void LoadRoads()

{

roads = Servise.GetRoads();

ApplyFilters();

}

private void LoadCategories()

{

categories = Servise.GetCategories();

var allCategory = new Category { CategoryId = 0, CategoryName = "Все" };

categories.Insert(0, allCategory);

Filter.ItemsSource = categories;

}

private void ApplyFilters()

{

var filteredRoads = roads.AsQueryable();

if (currentFilter != "Все")

{

filteredRoads = filteredRoads.Where(u => u.Category.CategoryName == currentFilter);

}

if (!string.IsNullOrWhiteSpace(currentSearch))

{

filteredRoads = filteredRoads.Where(u => u.CityA.Name.ToLower().StartsWith(currentSearch));

}

switch (currentSort)

{

case "Сортировать по возрастанию":

filteredRoads = filteredRoads.OrderBy(r => r.Length);

break;

case "Сортировать по убыванию":

filteredRoads = filteredRoads.OrderByDescending(r => r.Length);

break;

}

productListView.ItemsSource = filteredRoads.ToList();

}

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

{

NavigationService.Navigate(new MainPage());

}

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

{

NavigationService.GoBack();

AuthPage.UserRole = null;

}

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

{

NavigationService.Navigate(new EditRoadsPage());

}

private void Page\_Loaded(object sender, RoutedEventArgs e)

{

LoadRoads();

}

private void SortComboBox\_SelectionChanged\_1(object sender, SelectionChangedEventArgs e)

{

if (SortComboBox.SelectedItem is ComboBoxItem selectedItem)

{

currentSort = selectedItem.Content.ToString();

ApplyFilters();

}

}

private void Filter\_SelectionChanged\_1(object sender, SelectionChangedEventArgs e)

{

if (Filter.SelectedItem is Category selectedCategory)

{

currentFilter = selectedCategory.CategoryName;

ApplyFilters();

}

}

private void Source\_SelectionChanged(object sender, RoutedEventArgs e)

{

currentSearch = Source.Text.ToLower();

ApplyFilters();

}

}

}

Страница формирования маршрута

using iText.Kernel.Pdf;

using iText.Layout;

using iText.Layout.Element;

using System;

using System.Collections.Generic;

using System.IO;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

using Microsoft.EntityFrameworkCore;

using Microsoft.Win32;

using Paragraph = iText.Layout.Element.Paragraph;

using iText.Kernel.Font;

namespace Roads.Pages

{

public partial class MainPage : Page

{

public MainPage()

{

InitializeComponent();

using (var context = new TestDbContext())

{

startCityComboBox.ItemsSource = context.Cities.Select(c => c.Name).ToList();

endCityComboBox.ItemsSource = context.Cities.Select(c => c.Name).ToList();

}

}

decimal totalDistance = 0;

int totalHours = 0;

int totalMinutes = 0;

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

{

string startCityName = startCityComboBox.SelectedItem as string;

string endCityName = endCityComboBox.SelectedItem as string;

using (var context = new TestDbContext())

{

var startCity = context.Cities.FirstOrDefault(c => c.Name == startCityName);

var endCity = context.Cities.FirstOrDefault(c => c.Name == endCityName);

if (startCity == null || endCity == null)

{

MessageBox.Show("Один или оба города не найдены");

return;

}

var pathFinder = new PathFinder(context);

var path = pathFinder.FindPath(startCity.CityId, endCity.CityId);

if (path == null)

{

MessageBox.Show("Путь не найден");

}

else

{

routeListBox.Items.Clear();

decimal totalDistance = 0;

int totalHours = 0;

int totalMinutes = 0;

routeListBox.Items.Add($"Маршрут от {startCityName} до {endCityName}:");

routeListBox.Items.Add("");

foreach (var segment in path)

{

routeListBox.Items.Add($"Город {segment.CityA} -> Город {segment.CityB}:");

routeListBox.Items.Add($" - Длина: {segment.Length} км");

routeListBox.Items.Add($" - Категория: {segment.Category}");

routeListBox.Items.Add($" - Тип покрытия: {segment.SurfaceType}");

routeListBox.Items.Add($" - Ограничение скорости: {(segment.SpeedLimit.HasValue ? segment.SpeedLimit.ToString() : "нет данных")} км/ч");

if (segment.Gasstations.Any())

{

routeListBox.Items.Add(" - Заправки:");

foreach (var gasstation in segment.Gasstations)

{

routeListBox.Items.Add($" \* {gasstation.Name}, Адрес: {gasstation.Address}");

}

}

else

{

routeListBox.Items.Add(" - Заправок не найдено");

}

if (segment.Repairshops.Any())

{

routeListBox.Items.Add(" - Ремонтные мастерские:");

foreach (var repairshop in segment.Repairshops)

{

routeListBox.Items.Add($" \* {repairshop.Name}, Адрес: {repairshop.Address}");

}

}

else

{

routeListBox.Items.Add(" - Ремонтных мастерских не найдено");

}

routeListBox.Items.Add($" - Время в пути: {segment.TimeInTransit.Item1} ч {segment.TimeInTransit.Item2} мин");

routeListBox.Items.Add("");

totalDistance += segment.Length ?? 0;

totalHours += segment.TimeInTransit.Item1;

totalMinutes += segment.TimeInTransit.Item2;

}

// Учет превышения минут за 60

totalHours += totalMinutes / 60;

totalMinutes %= 60;

routeListBox.Items.Add($"Общее расстояние: {totalDistance} км");

routeListBox.Items.Add($"Общее время в пути: {totalHours} ч {totalMinutes} мин");

}

}

}

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

{

string startCityName = startCityComboBox.SelectedItem as string;

string endCityName = endCityComboBox.SelectedItem as string;

using (var context = new TestDbContext())

{

var startCity = context.Cities.FirstOrDefault(c => c.Name == startCityName);

var endCity = context.Cities.FirstOrDefault(c => c.Name == endCityName);

if (startCity == null || endCity == null)

{

MessageBox.Show("Один или оба города не найдены");

return;

}

var pathFinder = new PathFinder(context);

var path = pathFinder.FindPath(startCity.CityId, endCity.CityId);

if (path == null)

{

MessageBox.Show("Путь не найден");

}

else

{

// Открываем диалог сохранения файла

SaveFileDialog saveFileDialog = new SaveFileDialog();

saveFileDialog.Filter = "PDF files (\*.pdf)|\*.pdf";

saveFileDialog.FileName = "route\_report.pdf";

if (saveFileDialog.ShowDialog() == true)

{

string filePath = saveFileDialog.FileName;

using (var writer = new PdfWriter(filePath))

{

using (var pdf = new PdfDocument(writer))

{

var document = new Document(pdf);

// Устанавливаем моноширинный шрифт для всего текста

PdfFont font = PdfFontFactory.CreateFont(@"C:\Windows\Fonts\cour.ttf", "Identity-H");

// Заголовок маршрута

Paragraph title = new Paragraph($"Маршрут от {startCityName} до {endCityName}:").SetBold().SetFont(font).SetFontSize(14);

document.Add(title);

// Обнуляем общие значения перед подсчетом

decimal totalDistance = 0;

int totalHours = 0;

int totalMinutes = 0;

foreach (var segment in path)

{

// Информация о сегменте маршрута

Paragraph segmentInfo = new Paragraph($"Город {segment.CityA} -> Город {segment.CityB}:").SetFont(font).SetFontSize(12);

document.Add(segmentInfo);

document.Add(new Paragraph($" - Длина: {segment.Length} км").SetFont(font).SetFontSize(12));

document.Add(new Paragraph($" - Категория: {segment.Category}").SetFont(font).SetFontSize(12));

document.Add(new Paragraph($" - Тип покрытия: {segment.SurfaceType}").SetFont(font).SetFontSize(12));

document.Add(new Paragraph($" - Ограничение скорости: {(segment.SpeedLimit.HasValue ? segment.SpeedLimit.ToString() : "нет данных")} км/ч").SetFont(font).SetFontSize(12));

if (segment.Gasstations.Any())

{

document.Add(new Paragraph(" - Заправки:").SetFont(font).SetFontSize(12));

foreach (var gasstation in segment.Gasstations)

{

document.Add(new Paragraph($" \* {gasstation.Name}, Адрес: {gasstation.Address}").SetFont(font).SetFontSize(12));

}

}

else

{

document.Add(new Paragraph(" - Заправок не найдено").SetFont(font).SetFontSize(12));

}

if (segment.Repairshops.Any())

{

document.Add(new Paragraph(" - Ремонтные мастерские:").SetFont(font).SetFontSize(12));

foreach (var repairshop in segment.Repairshops)

{

document.Add(new Paragraph($" \* {repairshop.Name}, Адрес: {repairshop.Address}").SetFont(font).SetFontSize(12));

}

}

else

{

document.Add(new Paragraph(" - Ремонтных мастерских не найдено").SetFont(font).SetFontSize(12));

}

document.Add(new Paragraph($" - Время в пути: {segment.TimeInTransit.Item1} ч {segment.TimeInTransit.Item2} мин").SetFont(font).SetFontSize(12));

document.Add(new Paragraph("").SetFont(font).SetFontSize(12));

// Обновляем общие значения

totalDistance += segment.Length ?? 0;

totalHours += segment.TimeInTransit.Item1;

totalMinutes += segment.TimeInTransit.Item2;

}

// Учет превышения минут за 60

totalHours += totalMinutes / 60;

totalMinutes %= 60;

// Общая информация о маршруте

document.Add(new Paragraph($"Общее расстояние: {totalDistance} км").SetFont(font).SetFontSize(12));

document.Add(new Paragraph($"Общее время в пути: {totalHours} ч {totalMinutes} мин").SetFont(font).SetFontSize(12));

}

}

MessageBox.Show($"Отчет сохранен как {filePath}");

}

}

}

}

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

{

string startCityName = startCityComboBox.SelectedItem as string;

string endCityName = endCityComboBox.SelectedItem as string;

using (var context = new TestDbContext())

{

var startCity = context.Cities.FirstOrDefault(c => c.Name == startCityName);

var endCity = context.Cities.FirstOrDefault(c => c.Name == endCityName);

if (startCity == null || endCity == null)

{

MessageBox.Show("Один или оба города не найдены");

return;

}

var pathFinder = new PathFinder(context);

var path = pathFinder.FindPath(startCity.CityId, endCity.CityId);

if (path == null)

{

MessageBox.Show("Путь не найден");

}

else

{

routeListBox.Items.Clear();

decimal totalDistance = 0;

int totalHours = 0;

int totalMinutes = 0;

routeListBox.Items.Add($"Маршрут от {startCityName} до {endCityName}:");

routeListBox.Items.Add("");

foreach (var segment in path)

{

routeListBox.Items.Add($"Город {segment.CityA} -> Город {segment.CityB}:");

routeListBox.Items.Add($" - Длина: {segment.Length} км");

routeListBox.Items.Add($" - Категория: {segment.Category}");

routeListBox.Items.Add($" - Тип покрытия: {segment.SurfaceType}");

routeListBox.Items.Add($" - Ограничение скорости: {(segment.SpeedLimit.HasValue ? segment.SpeedLimit.ToString() : "нет данных")} км/ч");

if (segment.Gasstations.Any())

{

routeListBox.Items.Add(" - Заправки:");

foreach (var gasstation in segment.Gasstations)

{

routeListBox.Items.Add($" \* {gasstation.Name}, Адрес: {gasstation.Address}");

}

}

else

{

routeListBox.Items.Add(" - Заправок не найдено");

}

if (segment.Repairshops.Any())

{

routeListBox.Items.Add(" - Ремонтные мастерские:");

foreach (var repairshop in segment.Repairshops)

{

routeListBox.Items.Add($" \* {repairshop.Name}, Адрес: {repairshop.Address}");

}

}

else

{

routeListBox.Items.Add(" - Ремонтных мастерских не найдено");

}

routeListBox.Items.Add($" - Время в пути: {segment.TimeInTransit.Item1} ч {segment.TimeInTransit.Item2} мин");

routeListBox.Items.Add("");

totalDistance += segment.Length ?? 0;

totalHours += segment.TimeInTransit.Item1;

totalMinutes += segment.TimeInTransit.Item2;

}

// Учет превышения минут за 60

totalHours += totalMinutes / 60;

totalMinutes %= 60;

routeListBox.Items.Add($"Общее расстояние: {totalDistance} км");

routeListBox.Items.Add($"Общее время в пути: {totalHours} ч {totalMinutes} мин");

}

}

}

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

{

NavigationService.GoBack();

}

}

}

Страница сервисы

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Roads.Pages

{

class Servise

{

public static List<City> GetCities()

{

using (TestDbContext db = new TestDbContext())

{

var result = db.Cities.ToList();

return result;

}

}

public static List<Category> GetCategories()

{

using (TestDbContext db = new TestDbContext())

{

var result = db.Categories.ToList();

return result;

}

}

public static List<Repairshop> GetRepairshops()

{

using (TestDbContext db = new TestDbContext())

{

var result = db.Repairshops.ToList();

return result;

}

}

public static List<Gasstation> GetGasstations()

{

using (TestDbContext db = new TestDbContext())

{

var result = db.Gasstations.ToList();

return result;

}

}

public static List<Road> GetRoads()

{

using (TestDbContext db = new TestDbContext())

{

db.Cities.ToList();

db.Repairshops.ToList();

db.Gasstations.ToList();

db.Categories.ToList();

var result = db.Roads.ToList();

return result;

}

}

public void Addrepairshop(Repairshop repairshop)

{

using (TestDbContext db = new TestDbContext())

{

db.Repairshops.Add(repairshop);

db.SaveChanges();

}

}

public void Addcity(City city)

{

using (TestDbContext db = new TestDbContext())

{

db.Cities.Add(city);

db.SaveChanges();

}

}

public void Addgasstation(Gasstation gasstation)

{

using (TestDbContext db = new TestDbContext())

{

db.Gasstations.Add(gasstation);

db.SaveChanges();

}

}

public void Addroad(Road road)

{

using (TestDbContext db = new TestDbContext())

{

db.Roads.Add(road);

db.SaveChanges();

}

}

public void Adduser(User user)

{

using (TestDbContext db = new TestDbContext())

{

db.Users.Add(user);

db.SaveChanges();

}

}

public bool CityExists(string cityName)

{

using (TestDbContext db = new TestDbContext())

{

return db.Cities.Any(c => c.Name == cityName);

}

}

public bool RoadExists(int length, string surfaceType, int speedLimit, int categoryId, int cityAId, int cityBId)

{

using (TestDbContext db = new TestDbContext())

{

return db.Roads.Any(r =>

r.Length == length &&

r.SurfaceType == surfaceType &&

r.SpeedLimit == speedLimit &&

r.CategoryId == categoryId &&

r.CityAid == cityAId &&

r.CityBid == cityBId);

}

}

public bool RoadExistsWithCities(int cityAId, int cityBId)

{

using (TestDbContext db = new TestDbContext())

{

return db.Roads.Any(r => (r.CityAid == cityAId && r.CityBid == cityBId) || (r.CityAid == cityBId && r.CityBid == cityAId));

}

}

public bool GasStationExists(string address)

{

using (TestDbContext db = new TestDbContext())

{

string normalizedAddress = address.ToUpper();

return db.Gasstations.Any(g => g.Address.ToUpper() == normalizedAddress);

}

}

public bool RepairShopExists(string address)

{

using (TestDbContext db = new TestDbContext())

{

string normalizedAddress = address.ToUpper();

return db.Repairshops.Any(r => r.Address.ToUpper() == normalizedAddress);

}

}

}

}

Страницы добавления маршрута и объектов

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.Navigation;

using System.Windows.Shapes;

namespace Roads.Pages

{

/// <summary>

/// Логика взаимодействия для RepairshopAddPage.xaml

/// </summary>

public partial class RepairshopAddPage : Page

{

private Servise servise = new Servise();

public RepairshopAddPage()

{

InitializeComponent();

CityComboBox.ItemsSource = Servise.GetCities();

}

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

{

Repairshop repairshop = new Repairshop();

repairshop.Address = AddressTextBox.Text;

repairshop.Name = NameTextBox.Text;

City city = (City)CityComboBox.SelectedItem;

if (servise.RepairShopExists(repairshop.Address))

{

MessageBox.Show($"Мастерская с адресом \"{repairshop.Address}\" уже существует.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

repairshop.CityId = city.CityId;

servise.Addrepairshop(repairshop);

MessageBox.Show("Вы добавили мастерскую");

}

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

{

NavigationService.GoBack();

}

}

}

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.Navigation;

using System.Windows.Shapes;

namespace Roads.Pages

{

/// <summary>

/// Логика взаимодействия для AddCityPage.xaml

/// </summary>

public partial class AddCityPage : Page

{

private Servise servise = new Servise();

public AddCityPage()

{

InitializeComponent();

}

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

{

string cityName = NameTextBox.Text;

if (servise.CityExists(cityName))

{

MessageBox.Show($"Город с именем '{cityName}' уже существует в базе данных", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

City city = new City();

bool isPopulationValid = int.TryParse(PopulationTextBox.Text, out int population);

if (!isPopulationValid)

{

MessageBox.Show("Ошибка ввода численного значения для населения", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

city.Population = population;

city.Name = cityName;

try

{

servise.Addcity(city);

MessageBox.Show("Город успешно добавлен", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при добавлении города: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

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

{

NavigationService.GoBack();

}

}

}

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.Navigation;

using System.Windows.Shapes;

namespace Roads.Pages

{

/// <summary>

/// Логика взаимодействия для AddRoadsPage.xaml

/// </summary>

public partial class AddRoadsPage : Page

{

private Servise servise = new Servise();

public AddRoadsPage()

{

InitializeComponent();

CityAComboBox.ItemsSource = Servise.GetCities();

CityBComboBox.ItemsSource = Servise.GetCities();

CategoruComboBox.ItemsSource = Servise.GetCategories();

}

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

{

NavigationService.GoBack();

}

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

{

int length;

if (!int.TryParse(LengthTextBox.Text, out length))

{

MessageBox.Show("Ошибка ввода длины дороги", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

int speedLimit;

if (!int.TryParse(SpeedLimitTextBox.Text, out speedLimit))

{

MessageBox.Show("Ошибка ввода ограничения скорости", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

Category category = (Category)CategoruComboBox.SelectedItem;

if (category == null)

{

MessageBox.Show("Не выбрана категория дороги", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

City cityA = (City)CityAComboBox.SelectedItem;

if (cityA == null)

{

MessageBox.Show("Не выбран начальный город", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

City cityB = (City)CityBComboBox.SelectedItem;

if (cityB == null)

{

MessageBox.Show("Не выбран конечный город", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

string surfaceType = SurfaceTypeTextBox.Text;

if (servise.RoadExists(length, surfaceType, speedLimit, category.CategoryId, cityA.CityId, cityB.CityId))

{

MessageBox.Show($"Дорога с такими параметрами уже существует", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

if (servise.RoadExistsWithCities(cityA.CityId, cityB.CityId))

{

MessageBox.Show($"Дорога между выбранными городами уже существует", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

Road road = new Road();

road.Length = length;

road.SurfaceType = surfaceType;

road.SpeedLimit = speedLimit;

road.CategoryId = category.CategoryId;

road.CityAid = cityA.CityId;

road.CityBid = cityB.CityId;

try

{

servise.Addroad(road);

MessageBox.Show("Дорога успешно добавлена", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

catch (Exception ex)

{

MessageBox.Show($"Ошибка при добавлении дороги: {ex.Message}", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

}

}

}

}

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.Navigation;

using System.Windows.Shapes;

namespace Roads.Pages

{

/// <summary>

/// Логика взаимодействия для GasstationPage.xaml

/// </summary>

public partial class GasstationPage : Page

{

private Servise servise = new Servise();

public GasstationPage()

{

InitializeComponent();

CityComboBox.ItemsSource = Servise.GetCities();

}

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

{

Gasstation gasstation = new Gasstation();

gasstation.Address = AddressTextBox.Text;

gasstation.Name = NameTextBox.Text;

City city = (City)CityComboBox.SelectedItem;

if (servise.GasStationExists(gasstation.Address))

{

MessageBox.Show($"Заправка с адресом \"{gasstation.Address}\" уже существует.", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

gasstation.CityId = city.CityId;

servise.Addgasstation(gasstation);

MessageBox.Show("Вы добавили заправку");

}

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

{

NavigationService.GoBack();

}

}

}

Страницы редактирования

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace Roads.Pages.EditPage

{

/// <summary>

/// Логика взаимодействия для EditRoadPage.xaml

/// </summary>

public partial class EditRoadPage : Page

{

private ObservableCollection<Road> Roads;

private ObservableCollection<City> Cities;

private ObservableCollection<Category> Categories;

private TestDbContext context = new TestDbContext();

public EditRoadPage()

{

InitializeComponent();

Roads = new ObservableCollection<Road>(Servise.GetRoads());

Cities = new ObservableCollection<City>(Servise.GetCities());

Categories = new ObservableCollection<Category>(Servise.GetCategories());

RoadComboBox.ItemsSource = Roads;

CityAComboBox.ItemsSource = Cities;

CityBComboBox.ItemsSource = Cities;

CategoryComboBox.ItemsSource = Categories;

}

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

{

// Логика для отображения данных выбранного маршрута

Road selectedRoad = RoadComboBox.SelectedItem as Road;

if (selectedRoad != null)

{

LengthTextBox.Text = selectedRoad.Length.ToString();

SurfaceTypeTextBox.Text = selectedRoad.SurfaceType;

SpeedLimitTextBox.Text = selectedRoad.SpeedLimit.ToString();

CityAComboBox.SelectedItem = Cities.FirstOrDefault(c => c.CityId == selectedRoad.CityAid);

CityBComboBox.SelectedItem = Cities.FirstOrDefault(c => c.CityId == selectedRoad.CityBid);

CategoryComboBox.SelectedItem = Categories.FirstOrDefault(cat => cat.CategoryId == selectedRoad.CategoryId);

}

}

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

{

Road selectedRoad = RoadComboBox.SelectedItem as Road;

if (selectedRoad != null)

{

long length;

int speedLimit;

if (!long.TryParse(LengthTextBox.Text, out length) || !int.TryParse(SpeedLimitTextBox.Text, out speedLimit))

{

MessageBox.Show("Длина и ограничение скорости должны быть числами!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

City cityA = CityAComboBox.SelectedItem as City;

City cityB = CityBComboBox.SelectedItem as City;

if (cityA == null || cityB == null)

{

MessageBox.Show("Пожалуйста, выберите начальный и конечный города!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

if (cityA.CityId == cityB.CityId)

{

MessageBox.Show("Начальный и конечный города не могут быть одинаковыми!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

bool routeExists = context.Roads.Any(r => r.CityAid == cityA.CityId && r.CityBid == cityB.CityId && r.RoadId != selectedRoad.RoadId);

if (routeExists)

{

MessageBox.Show("Маршрут между этими городами уже существует!", "Ошибка", MessageBoxButton.OK, MessageBoxImage.Error);

return;

}

selectedRoad.Length = length;

selectedRoad.SurfaceType = SurfaceTypeTextBox.Text;

selectedRoad.SpeedLimit = speedLimit;

selectedRoad.CityAid = cityA.CityId;

selectedRoad.CityBid = cityB.CityId;

selectedRoad.CategoryId = (CategoryComboBox.SelectedItem as Category)?.CategoryId ?? selectedRoad.CategoryId;

context.Roads.Update(selectedRoad);

context.SaveChanges();

RefreshRoads();

RoadComboBox.SelectedItem = Roads.FirstOrDefault(r => r.RoadId == selectedRoad.RoadId);

MessageBox.Show("Маршрут успешно обновлен!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

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

{

NavigationService.GoBack();

}

private void RefreshRoads()

{

// Обновление данных в коллекции Roads

Roads = new ObservableCollection<Road>(Servise.GetRoads());

RoadComboBox.ItemsSource = Roads;

}

}

}

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace Roads.Pages.EditPage

{

/// <summary>

/// Логика взаимодействия для EditCityPage.xaml

/// </summary>

public partial class EditCityPage : Page

{

private ObservableCollection<City> Cities;

private TestDbContext context = new TestDbContext();

public EditCityPage()

{

InitializeComponent();

Cities = new ObservableCollection<City>(Servise.GetCities());

CityComboBox.ItemsSource = Cities;

}

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

{

City selectedCity = CityComboBox.SelectedItem as City;

if (selectedCity != null)

{

selectedCity.Name = CityNameTextBox.Text;

selectedCity.Population = int.Parse(PopulationTextBox.Text);

context.Cities.Update(selectedCity);

context.SaveChanges();

RefreshCities();

CityComboBox.SelectedItem = Cities.FirstOrDefault(c => c.CityId == selectedCity.CityId);

MessageBox.Show("Город успешно обновлен!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

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

{

NavigationService.GoBack();

}

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

{

City selectedCity = CityComboBox.SelectedItem as City;

if (selectedCity != null)

{

CityNameTextBox.Text = selectedCity.Name;

PopulationTextBox.Text = selectedCity.Population.ToString();

}

}

private void RefreshCities()

{

Cities = new ObservableCollection<City>(Servise.GetCities());

CityComboBox.ItemsSource = Cities;

}

}

}

using System;

using System.Collections.ObjectModel;

using System.Linq;

using System.Windows;

using System.Windows.Controls;

namespace Roads.Pages.EditPage

{

/// <summary>

/// Логика взаимодействия для EditGasStationPage.xaml

/// </summary>

public partial class EditGasStationPage : Page

{

private ObservableCollection<Gasstation> GasStations;

private ObservableCollection<City> Cities;

private TestDbContext context = new TestDbContext();

public EditGasStationPage()

{

InitializeComponent();

GasStations = new ObservableCollection<Gasstation>(Servise.GetGasstations());

Cities = new ObservableCollection<City>(Servise.GetCities());

AdressComboBox.ItemsSource = GasStations;

CityNameComboBox.ItemsSource = Cities;

}

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

{

Gasstation selectedGasStation = AdressComboBox.SelectedItem as Gasstation;

if (selectedGasStation != null)

{

CityNameComboBox.SelectedItem = Cities.FirstOrDefault(c => c.CityId == selectedGasStation.CityId);

NameTextBox.Text = selectedGasStation.Name;

AdressTextBox.Text = selectedGasStation.Address;

}

}

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

{

Gasstation selectedGasStation = AdressComboBox.SelectedItem as Gasstation;

if (selectedGasStation != null)

{

selectedGasStation.CityId = (CityNameComboBox.SelectedItem as City)?.CityId ?? selectedGasStation.CityId;

selectedGasStation.Name = NameTextBox.Text;

selectedGasStation.Address = AdressTextBox.Text;

context.Gasstations.Update(selectedGasStation);

context.SaveChanges();

RefreshGasStations();

AdressComboBox.SelectedItem = GasStations.FirstOrDefault(g => g.GasStationId == selectedGasStation.GasStationId);

MessageBox.Show("Заправка успешно обновлена!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

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

{

NavigationService.GoBack();

}

private void RefreshGasStations()

{

GasStations = new ObservableCollection<Gasstation>(Servise.GetGasstations());

AdressComboBox.ItemsSource = GasStations;

}

}

}

using System;

using System.Collections.Generic;

using System.Collections.ObjectModel;

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.Navigation;

using System.Windows.Shapes;

namespace Roads.Pages.EditPage

{

/// <summary>

/// Логика взаимодействия для EditRepairShopPage.xaml

/// </summary>

public partial class EditRepairShopPage : Page

{

private ObservableCollection<Repairshop> Repairshops;

private ObservableCollection<City> Cities;

private TestDbContext context = new TestDbContext();

public EditRepairShopPage()

{

InitializeComponent();

Repairshops = new ObservableCollection<Repairshop>(Servise.GetRepairshops());

Cities = new ObservableCollection<City>(Servise.GetCities());

AdressComboBox.ItemsSource = Repairshops;

CityNameComboBox.ItemsSource = Cities;

}

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

{

Repairshop selectedGasStation = AdressComboBox.SelectedItem as Repairshop;

if (selectedGasStation != null)

{

CityNameComboBox.SelectedItem = Cities.FirstOrDefault(c => c.CityId == selectedGasStation.CityId);

NameTextBox.Text = selectedGasStation.Name;

AdressTextBox.Text = selectedGasStation.Address;

}

}

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

{

Repairshop selectedGasStation = AdressComboBox.SelectedItem as Repairshop;

if (selectedGasStation != null)

{

selectedGasStation.CityId = (CityNameComboBox.SelectedItem as City)?.CityId ?? selectedGasStation.CityId;

selectedGasStation.Name = NameTextBox.Text;

selectedGasStation.Address = AdressTextBox.Text;

context.Repairshops.Update(selectedGasStation);

context.SaveChanges();

RefreshGasStations();

AdressComboBox.SelectedItem = Repairshops.FirstOrDefault(g => g.RepairShopId == selectedGasStation.RepairShopId);

MessageBox.Show("Заправка успешно обновлена!", "Успех", MessageBoxButton.OK, MessageBoxImage.Information);

}

}

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

{

NavigationService.GoBack();

}

private void RefreshGasStations()

{

Repairshops = new ObservableCollection<Repairshop>(Servise.GetRepairshops());

AdressComboBox.ItemsSource = Repairshops;

}

}

}

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. [ГОСТ 28195-89](https://gost.ruscable.ru/Index/11/11212.htm) Оценка качества программных средств. Общие положения: межгосударственный стандарт: дата введения 1990-07-01 - <https://docs.cntd.ru/document/1200009135?ysclid=lroox3amxp882669222> – Текст: электронный.
2. [ГОСТ 28806-90](https://gost.ruscable.ru/Index/10/10605.htm) Качество программных средств. Термины и определения: межгосударственный стандарт: дата введения 1992-01-01 - <https://docs.cntd.ru/document/1200009077?ysclid=lrq6to4u1z728775522> – Текст: электронный.
3. [ГОСТ 34.321-96](https://gost.ruscable.ru/Index/6/6808.htm) Информационные технологии. Система стандартов по базам данных. Эталонная модель управления данными: межгосударственный стандарт: дата введения 2001-07-01 - <https://docs.cntd.ru/document/1200017662?ysclid=lrq6wn65z5753511468> – Текст: электронный.
4. [ГОСТ Р ИСО/МЭК 12207-99. Информационная технология. Процессы жизненного цикла программных средств](https://docs.cntd.ru/document/1200009075#7D20K3): межгосударственный стандарт: дата введения 2000-07-01 -https://docs.cntd.ru/document/1200009075?ysclid=lrool2t1vo124212660. – Текст: электронный.
5. Ботрос, С. MySQL по максимуму / C. Ботрос, Д. Тинли., – 4-е изд., перераб. и доп. – Питер, 2023. – 432 с. – ISBN 978-5-4461-2261-5. – Текст: непосредственный.
6. Доусон, М. Программируем на Python / М. Доусон., перераб. и доп. – Питер, 2022. – 416 с. – ISBN 978-5-4461-1386-6. – Текст: непосредственный.
7. Кириченко, А.В. Web на практике. CSS, HTML, JavaScript, MySQL, PHP для fullstack-разработчиков / А.В. Кириченко, А.П. Никольский, Е.В. Дубовик., перераб. и доп. – Питер, 2021. – 432 с. – ISBN 978-5-94-387271-6. – Текст: непосредственный.
8. Макеев, Г.А. Объектно-ориентированное программирование: с нуля к SOLID и MVC / Г.А. Макеев., перераб. и доп. – БХВ, 2024. – 272 с. – ISBN 978-5-9775-1913-7. – Текст: непосредственный.
9. Постолит, А.В. Python, Django и Bootstrap для начинающих / А.В. Постолит., переаб. и доп. – БХВ, 2023. – 624 с. – ISBN 978-5-9775-1807-9. – Текст: непосредственный
10. Стефанов, С. React. Быстрый старт / С. Стефанов., – 2-е изд., перераб. и доп. – Питер, 2023. – 304 с. – ISBN 978-5-4461-2115-1. – Текст: непосредственный.
11. Тараканов, О.В. Базы данных. Учебник. Студентам ССУЗов / О.В. Тараканов, Л.И. Шустова., 2023. – 304 с. – ISBN 978-5-16-010485-0. - Текст: непосредственный.
12. Трэйси, О. Веб-дизайн для недизайнеров / О. Трэйси., перераб. и доп. – Питер, 2022. – 176 с. – ISBN 978-5-4461-1917-2. – Текст: непосредственный.
13. Документация CSS. – 2024. Текст : электронный. – URL: <https://developer.mozilla.org/ru/docs/Web/CSS/Reference> (дата обращения: 26.01.2024).
14. Документация Django. – 2021. Текст : электронный. – URL: <https://djangodoc.ru/3.2/> (дата обращения: 26.01.2024).
15. Документация HTML. – 2024. Текст : электронный. – URL: <https://developer.mozilla.org/ru/docs/Web/HTML> (дата обращения: 26.01.2024).
16. Документация JavaScript. – 2024. Текст : электронный. – URL: <https://developer.mozilla.org/ru/docs/Web/JavaScript> (дата обращения: 26.01.2024).
17. Документация MySQL. – 2024. Текст : электронный. – URL: <https://dev.mysql.com/doc/> (дата обращения: 26.01.2024).
18. Документация Python. – 2024. Текст : электронный. – URL: <https://docs.python.org/3/index.html> (дата обращения: 26.01.2024).
19. Документация React. – 2023. Текст : электронный. – URL: <https://react.dev/learn> (дата обращения: 26.01.2024).
20. Документация Яндекс Карты API. Текст : электронный. – URL: <https://yandex.ru/maps-api/docs?ysclid=lrq7c8dsns226257630> (дата обращения: 26.01.2024

1. PK-первичный ключ

   FK-внешний ключ [↑](#footnote-ref-1)