Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

«Уральский федеральный университет имени первого Президента России Б.Н. Ельцина » (УрФУ)

Институт «Физико-технологический»

Кафедра «Техническая физика»

Оценка

Руководитель курсового

проектирования

Члены комиссии \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата защиты 23.12.2024

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

по теме: «Разработка подсказчика, сопровождающего автотуристов и работающего на принципах СУБД»

Студент Чирков Андрей Сергеевич

(ФИО) Подпись

Группа ФТ-320007

Екатеринбург

2024

Оглавление

[Введение 3](#_Toc185792431)

[Описание и актуальность 3](#_Toc185792432)

[Цели и задачи 3](#_Toc185792433)

[Основная часть 5](#_Toc185792434)

[Проектирование базы данных 5](#_Toc185792435)

[Информационная модель 6](#_Toc185792436)

[Создание интерфейса 13](#_Toc185792437)

[Реализация 16](#_Toc185792438)

[Заключение 18](#_Toc185792439)

[Список источников 19](#_Toc185792440)

[Приложение А. Заполнение таблиц и схема маршрутов 20](#_Toc185792441)

[Приложение Б: Исходный код программы 23](#_Toc185792442)

Введение

Описание и актуальность

Данный проект представляет собой программу для планирования маршрутов, записи и доступа к истории поездок с использованием базы данных Microsoft Access. Необходимо предоставить пользователю интерфейс для регистрации, авторизации и расчета оптимальных маршрутов между заданными точками.

В современном мире, где задачи логистики и перемещения становятся всё более сложными, автоматизированные системы расчета маршрутов играют ключевую роль. Разработка такого приложения предоставляет пользователям возможность быстро находить оптимальные пути, учитывать остановки для дозаправки и ночлега.

Цель и задачи

**Цель работы:**

Разработать подсказчик, сопровождающий автотуристов и работающий на принципах СУБД.

**Задачи работы:**

1. **Проектирование структуры данных:**

Разработать базу данных для хранения информации о городах, маршрутах, объектах инфраструктуры, истории поездок и данных пользователей.

Определить взаимосвязи между данными (поля и ключи таблиц) для эффективной работы приложения.

1. **Реализация модуля расчёта маршрутов:**

Внедрить алгоритм поиска кратчайшего пути на основе графов.

Рассчитать параметры поездки: расстояние, время в пути, количество остановок (на ночлег и дозаправку).

1. **Создание пользовательского интерфейса:**

Разработать окно регистрации и авторизации.

Реализовать главное окно для ввода исходных данных и запуска расчёта маршрута.

Разработать окно для отображения истории поездок пользователя.

1. **Обеспечение взаимодействия с СУБД:**

Настроить подключение к базе данных и выполнение запросов для извлечения и записи информации.

Автоматизировать сохранение данных о маршрутах в историю поездок.

1. **Разработка модуля фильтрации:**

Реализовать механизм отображения истории маршрутов только для авторизованного пользователя.

1. **Реализация функций интеллектуального подсказчика:**

Вывод информации о заправках, отелях и других объектах инфраструктуры вдоль маршрута.

Автоматическое предложение оптимальных остановок в зависимости от расстояния и времени в пути.

1. **Документирование:**

Описать архитектуру программы, функциональные модули и структуру базы данных.

Основная часть

Проектирование базы данных

Изображение выглядит как текст, диаграмма, снимок экрана, линия

Автоматически созданное описание

Рисунок 1 - Схема данных

Всего у базы данных есть 6 таблиц, связанных между собой по Id. Однако таблицы Users и TripsHistory связаны только между собой через UserID. Ниже описаны атрибуты каждой таблицы:

**Таблица «Cities»:**

CityID INTEGER PRIMARY KEY – Уникальный идентификатор города

CityName VARCHAR – Название города

**Таблица «Routes»:**

RouteID INTEGER PRIMARY KEY – Уникальный идентификатор маршрута (пути между городами)

FromCityID INTEGER – Идентификатор города отправления (Связан с таблицей Cities через CityID)

ToCityID INTEGER – Идентификатор города отправления (Связан с таблицей Cities через CityID)

DistanceKm REAL – Расстояние между двумя городами

**Таблица «ObjectTypes»:**

ObjectTypeID INTEGER PRIMARY KEY – Уникальный идентификатор типа объекта (Связан с таблицей InfrastructureObjects через ObjectType)

ObjectTypeName VARCHAR – Наименование типа объекта

**Таблица «InfrastructureObjects»:**

ObjectID INTEGER PRIMARY KEY – Уникальный идентификатор объекта

ObjectType INTEGER – Идентификатор типа объекта (таблица ObjectTypes)

ObjectName VARCHAR – Название объекта

CityID INTEGER – Идентификатор города

DistanceFromCity REAL – Расстояние между городами

OnRouteID INTEGER – Идентификатор маршрута (связан с таблицей Routes через RouteID)

**Таблица «Users»:**

UserID INTEGER AUTOINCREMENT PRIMARY KEY – Уникальный идентификатор пользователя

Username VARCHAR – Уникальный логин пользователя

Password VARCHAR – Пароль пользователя

**Таблица «TripsHistory»**

TripID INTEGER AUTOINCREMENT PRIMARY KEY - Уникальный идентификатор созданного маршрута

UserID INTEGER – Идентификатор пользователя (связан с таблицей Users через UserID)

RouteDate DATE – Дата создания маршрута

RouteDetails VARCHAR – Детали маршрута

Информационная модель

Изображение выглядит как текст, снимок экрана, диаграмма, Шрифт

Автоматически созданное описание

Рисунок 2 – Информационная модель

Программа состоит из 4 модулей: main, route\_logic, hash\_finc и db\_utils.

В главном модуле создается весь интерфейс программы и происходят следующие процессы: регистрация, аутентификация пользователей и предоставление доступа к функционалу: расчету маршрута и истории поездок. Функционал истории поездок находится здесь же.

В модуле db\_utils находятся функции связей между программой и базой данных.

В модуле route\_logic описаны основные алгоритмы программы – расчета маршрута, описания объектов, находящихся на нем и предоставление рекомендаций. Они задействованы с главным модулем

В модуле hash\_func находятся функции, связанные с регистрацией и авторизацией пользователей – хэширование пароля при регистрации и проверка вводимого пароля при авторизации.

Функциональные блоки

Часть функции, которая запрашивает у пользователя его логин и пароль для регистрации, по которым он в будущем сможет зайти.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 3 – Код функции регистрации

Часть функции, которая отвечает за авторизацию пользователя, запрашивает те же логин и пароль.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 4 - Код функции авторизации пользователя

Код функции шифрования паролей путем хэширования, а так же код верификации пароля (введенный пароль хэшируется той же функцией и, если хэши совпадают, происходит авторизация).

Изображение выглядит как текст, снимок экрана, программное обеспечение, Операционная система

Автоматически созданное описание

Рисунок 5 – Код функции хэширования

Данная функция осуществляет подключение к базе данных и выводит ошибку при неудаче.

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 6 – Код функции подключения к базе данных

Часть кода функции регистрации пользователя и записи его данных в базу данных – логина и хэшированного пароля. (логин проверяется на уникальность).

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 7 – Код функции регистрации пользователя и занесения в базу данных

Часть функции, которая осуществляет авторизацию пользователя, сравнивая вводимые данные с данными в базе данных.

Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Автоматически созданное описание

Рисунок 8 – Код функции авторизации пользователя

Функции, проверяющие наличие городов в базе данных (в соответствии выбранному маршруту), также функция получения наименования типа объекта через его ID.

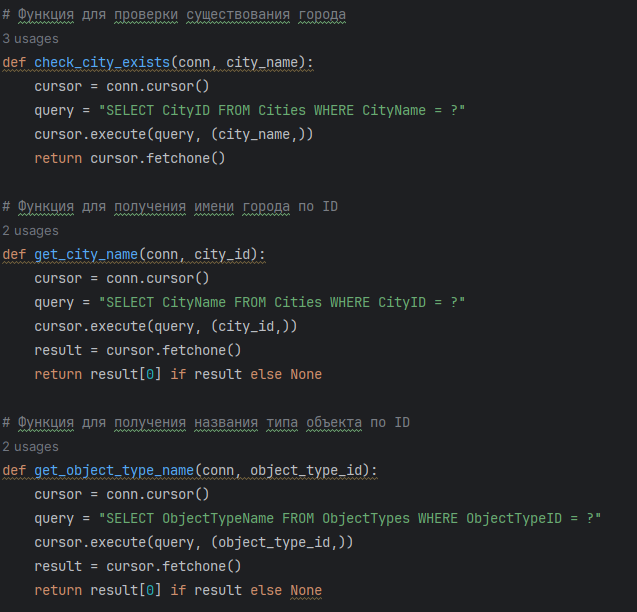


Рисунок 9 – Код функции проверки наличия города в базе данных и код функции получения наименования объектов инфраструктуры

Данная функция представляет все возможные пути из города отправки в город прибытия. Затем находит наикратчайший путь.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Автоматически созданное описание

Рисунок 10 – Код функции реализации логики поиска маршрута

Данная функция находит на построенном маршруте объекты инфраструктуры и выводит их параметры: наименования, типы, нахождения в км от ближайшего города.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 11 – код функции поиска объектов инфраструктуры на маршруте

Часть кода, реализующая расчет данных маршрута: дистанцию, время прибытия, рекомендации остановок на ночлег и дозаправку, а также города, АЗС, отели и кафе, которые находятся на пути.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Рисунок 12 – Часть кода функции расчета

Часть кода, осуществляет обращение к базе данных для загрузки истории поездок.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Автоматически созданное описание

Рисунок 13 – Обращение к базе данных для записи истории

Код функции, осуществляющий запись данных в историю поездки пользователя:

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Рисунок 14 – Запись истории в базу данных

Создание интерфейса

Сначала попадаем в окно, в котором можно выбрать регистрацию, авторизацию или выйти из программы.

Изображение выглядит как текст, снимок экрана, дисплей, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 15 – Окно выбора регистрации и авторизации

Переходим в регистрацию и заполняем имя пользователя (логин – поле с уникальным содержимым) и пароль (повторяем для проверки). Также можем вернуться на начальное окно.

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 16 – Окно регистрации пользователя

После успешной регистрации выводится следующее сообщение, далее мы попадаем в окно авторизации.

Изображение выглядит как текст, снимок экрана, программное обеспечение, дисплей

Автоматически созданное описание

Рисунок 17 – Окно успешной регистрации

В этом окне пользователь вводит данные и авторизуется в систему.

Изображение выглядит как текст, снимок экрана, дисплей, число

Автоматически созданное описание

Рисунок 18 – Окно авторизации

Далее переходим в главное окно программы, в котором находятся поля для ввода данных, необходимы для создания маршрута. Ниже находится окно вывода расчетов, а справа представлена схема городов и путевых связей между ними. Также ниже поля вывода расчетов находятся две кнопки: выход (для возврата на начальный экран) и История поездок (для перехода в окно с историей поездок).

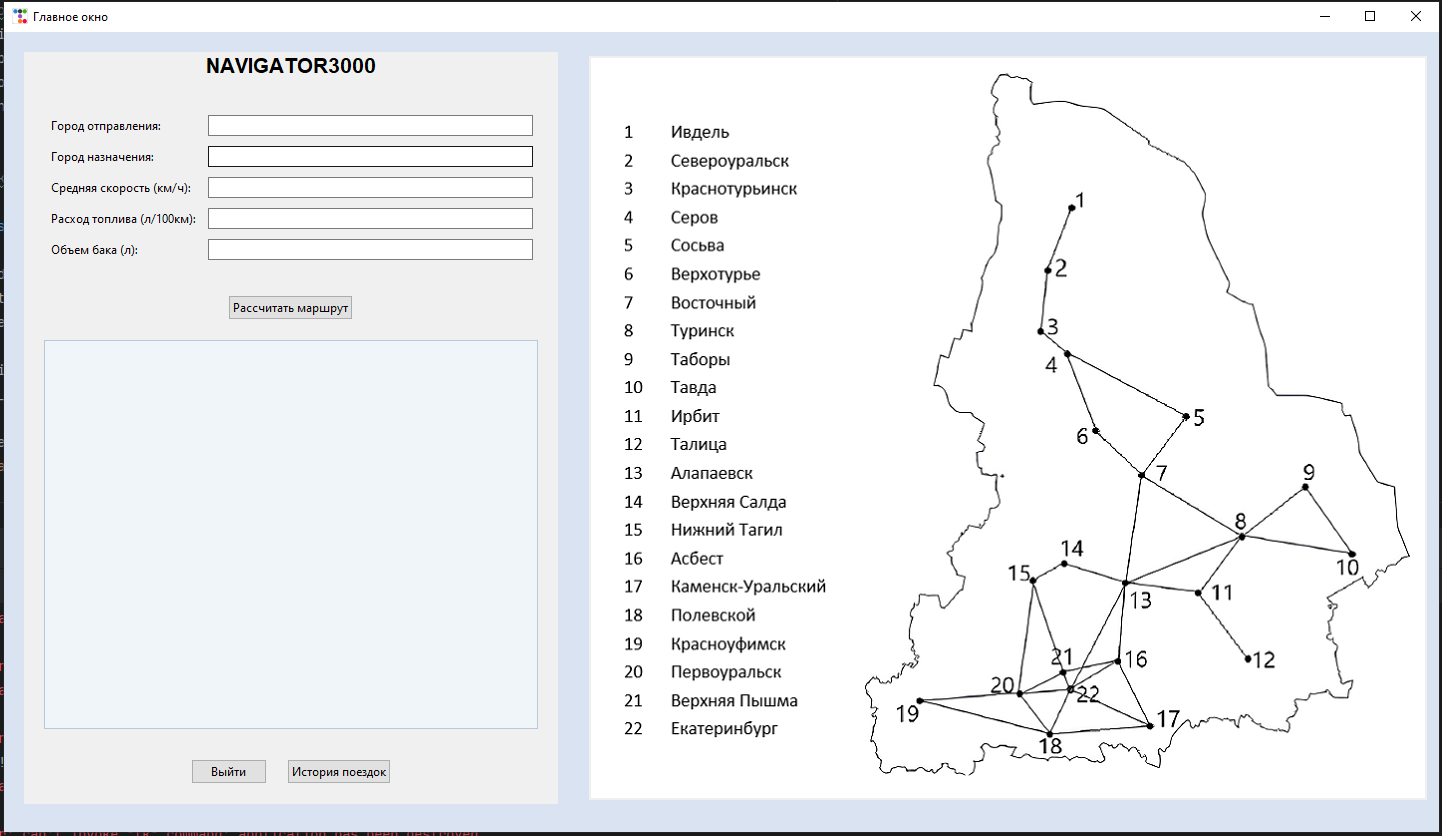


Рисунок 19 – Главное окно программы

Окно с историей поездок: здесь будут отображаться такие параметры как ID построенного маршрута, дата и детали (у каждого пользователя своя история маршрутов). Также есть кнопка для возврата на главный экран

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 20 – Окно истории поездок

Реализация

Регистрируемся:

Изображение выглядит как текст, снимок экрана, программное обеспечение, Значок на компьютере

Автоматически созданное описание

Рисунок 21 – Регистрация

Вводим данные и рассчитываем маршрут:  
Изображение выглядит как текст, диаграмма, карта, снимок экрана

Автоматически созданное описание

Рисунок 22 – Построение маршрута

Переходим в Историю поездок:

Изображение выглядит как текст, снимок экрана, дисплей, программное обеспечение

Автоматически созданное описание

Рисунок 23 – История поездок с внесенными данными

# **Заключение**

В этой работе была реализована система с подсказчиком для автотуристов, работающего на принципах СУБД.

Была разработана информационная модель данных, включающая таблицы

городов, объектов инфраструктуры, пользователей, маршрутов и типов объектов.

Была реализована функциональная модель с основными возможностями: регистрация, авторизация и использование самой программы, ввод параметров, необходимых для расчета маршрута, расчет и вывод рассчитанных данных.

Для хранения данных была использована реляционная СУБД MS Access. Программа реализована с использованием языка программирования Python, библиотеки для работы с графическим интерфейсом ttkbootstrap и модуля pyodbc для взаимодействия с базой данных.

В целом в работе были продемонстрированы навыки проектирования и реализации приложения с использованием SQL. Полученный опыт будет способствовать дальнейшему развитию компетенций в области разработки ПО.

# **Список источников**

1) Учебное пособие Язые реляционных баз данных В.Ю. Кара-Ушанов (дата обращения 10.12.2024) — Режим доступа[электронный]: <https://elar.urfu.ru/bitstream/10995/40612/1/978-5-7996-1622-9_2016.pdf>

2) Руководство по работе с MS Access на Python (дата обращения 12.12.2024) — Режим доступа[электронный]: <https://pythoninoffice.com/python-ms-access-database-pyodbc/>

# **Приложение А. Заполнение таблиц и схема маршрутов**

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описаниеИзображение выглядит как текст, снимок экрана, Шрифт, линия

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, программное обеспечение

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, Параллельный

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, число, меню

Автоматически созданное описание

Изображение выглядит как текст, диаграмма, карта

Автоматически созданное описание

# **Приложение Б: Исходный код программы**

**Модуль Main:**

import ttkbootstrap as ttk  
from tkinter import messagebox  
from db\_utils import connect\_to\_database, register\_user, login\_user, history\_save\_db  
from route\_logic import calculate\_route  
from PIL import Image, ImageTk  
  
def center\_window(window, width, height):  
 screen\_width = window.winfo\_screenwidth()  
 screen\_height = window.winfo\_screenheight()  
 x = (screen\_width - width) // 2  
 y = (screen\_height - height) // 2  
 window.geometry(f"{width}x{height}+{x}+{y}")  
  
  
  
class StartWindow(ttk.Window):  
 def \_\_init\_\_(self, db\_path):  
 super().\_\_init\_\_(title="Добро пожаловать!")  
 self.db\_path = db\_path  
 self.conn = None  
 center\_window(self, 300, 200)  
 self.resizable(False, False)  
  
 # Лэйбл заголовка  
 ttk.Label(self, text="Выберите действие", font=("Arial", 14), anchor="center").grid(row=0, column=0, columnspan=2, pady=20, sticky="ew")  
  
 # Кнопка регистрации  
 reg\_button = ttk.Button(self, text="Регистрация", command=self.show\_registration)  
 reg\_button.grid(row=1, column=0, padx=10, pady=10, sticky="ew")  
  
 # Кнопка авторизации  
 auth\_button = ttk.Button(self, text="Авторизация", command=self.show\_login)  
 auth\_button.grid(row=1, column=1, padx=10, pady=10, sticky="ew")  
  
 # Кнопка для выхода  
 exit\_button = ttk.Button(self, text="Выйти", command=self.exit\_program, bootstyle="danger-outline")  
 exit\_button.grid(row=2, column=0, columnspan=2, padx=10, pady=10) # Изменено columnspan  
  
 # Настройка строк и столбцов для адаптивности  
 self.grid\_columnconfigure(0, weight=1)  
 self.grid\_columnconfigure(1, weight=1)  
  
 self.protocol("WM\_DELETE\_WINDOW", self.on\_closing)  
  
 def show\_registration(self):  
 self.destroy()  
 reg\_window = RegistrationWindow(self.db\_path)  
 reg\_window.mainloop()  
  
 def show\_login(self):  
 self.destroy()  
 login\_window = LoginWindow(self.db\_path)  
 login\_window.mainloop()  
  
 def exit\_program(self):  
 self.destroy()  
  
 def on\_closing(self):  
 if self.conn:  
 self.conn.close()  
 print("Соединение с БД закрыто.")  
 self.destroy()  
  
class RegistrationWindow(ttk.Window):  
 def \_\_init\_\_(self, db\_path):  
 super().\_\_init\_\_(title="Регистрация")  
 self.db\_path = db\_path  
 center\_window(self, 300, 250)  
 self.resizable(False, False)  
  
 # Лэйбл для имени пользователя  
 ttk.Label(self, text="Имя пользователя:", anchor="w").grid(row=0, column=0, padx=10, pady=5, sticky="ew")  
 self.username\_entry = ttk.Entry(self)  
 self.username\_entry.grid(row=0, column=1, padx=10, pady=5, sticky="ew")  
  
 # Лэйбл для пароля  
 ttk.Label(self, text="Пароль:", anchor="w").grid(row=1, column=0, padx=10, pady=5, sticky="ew")  
 self.password\_entry = ttk.Entry(self, show="\*") # Маскируем пароль  
 self.password\_entry.grid(row=1, column=1, padx=10, pady=5, sticky="ew")  
  
 # Лэйбл для подтверждения пароля  
 ttk.Label(self, text="Подтвердите пароль:", anchor="w").grid(row=2, column=0, padx=10, pady=5, sticky="ew")  
 self.confirm\_password\_entry = ttk.Entry(self, show="\*") # Маскируем пароль  
 self.confirm\_password\_entry.grid(row=2, column=1, padx=10, pady=5, sticky="ew")  
  
 # Кнопка регистрации  
 reg\_button = ttk.Button(self, text="Зарегистрироваться", command=self.register, bootstyle="primary")  
 reg\_button.grid(row=3, column=0, columnspan=2, padx=10, pady=15, sticky="ew")  
  
 # Кнопка назад  
 back\_button = ttk.Button(self, text="Назад", command=self.go\_back, bootstyle="secondary-outline")  
 back\_button.grid(row=4, column=0, columnspan=2, padx=10, pady=5, sticky="ew")  
  
 # Настройка строк и столбцов для адаптивности  
 self.grid\_columnconfigure(0, weight=1)  
 self.grid\_columnconfigure(1, weight=1)  
  
 def register(self):  
 username = self.username\_entry.get()  
 password = self.password\_entry.get()  
 if not username or not password:  
 messagebox.showerror("Ошибка", "Пожалуйста, заполните все поля.")  
 return  
 conn = connect\_to\_database(self.db\_path)  
 if conn:  
 user\_id = register\_user(conn, username, password)  
 if user\_id:  
 messagebox.showinfo("Успех", "Регистрация прошла успешно!")  
 conn.close()  
 self.go\_back()  
 else:  
 conn.close()  
 messagebox.showerror("Ошибка", "Ошибка регистрации.")  
  
 def go\_back(self):  
 self.destroy()  
 start\_window = StartWindow(self.db\_path)  
 start\_window.mainloop()  
  
class LoginWindow(ttk.Window):  
 def \_\_init\_\_(self, db\_path):  
 super().\_\_init\_\_(title="Авторизация", themename="morph")  
 self.db\_path = db\_path  
 center\_window(self, 300, 250)  
 self.resizable(False, False)  
  
 # Лэйбл для имени пользователя  
 ttk.Label(self, text="Имя пользователя:", anchor="w").grid(row=0, column=0, padx=10, pady=5, sticky="ew")  
 self.username\_entry = ttk.Entry(self)  
 self.username\_entry.grid(row=0, column=1, padx=10, pady=5, sticky="ew")  
  
 # Лэйбл для пароля  
 ttk.Label(self, text="Пароль:", anchor="w").grid(row=1, column=0, padx=10, pady=5, sticky="ew")  
 self.password\_entry = ttk.Entry(self, show="\*")  
 self.password\_entry.grid(row=1, column=1, padx=10, pady=5, sticky="ew")  
  
 # Кнопка войти  
 login\_button = ttk.Button(self, text="Войти", command=self.login, bootstyle="success")  
 login\_button.grid(row=2, column=0, columnspan=2, padx=10, pady=15, sticky="ew")  
  
 # Кнопка назад  
 back\_button = ttk.Button(self, text="Назад", command=self.go\_back, bootstyle="secondary-outline")  
 back\_button.grid(row=3, column=0, columnspan=2, padx=10, pady=5, sticky="ew")  
  
 # Настройка строк и столбцов для адаптивности  
 self.grid\_columnconfigure(0, weight=1)  
 self.grid\_columnconfigure(1, weight=1)  
  
 def login(self):  
 username = self.username\_entry.get()  
 password = self.password\_entry.get()  
 if not username or not password:  
 messagebox.showerror("Ошибка", "Пожалуйста, заполните все поля.")  
 return  
  
 conn = connect\_to\_database(self.db\_path)  
 if conn:  
 user\_id = login\_user(conn, username, password)  
 if user\_id:  
 self.destroy()  
 main\_window = MainWindow(self.db\_path, user\_id)  
 main\_window.mainloop()  
 else:  
 conn.close()  
 messagebox.showerror("Ошибка", "Неверный логин или пароль.")  
  
 def go\_back(self):  
 self.destroy()  
 start\_window = StartWindow(self.db\_path)  
 start\_window.mainloop()  
  
  
class MainWindow(ttk.Window):  
 def \_\_init\_\_(self, db\_path, user\_id):  
 super().\_\_init\_\_(title="Главное окно", themename="morph")  
 self.db\_path = db\_path  
 self.user\_id = user\_id  
 center\_window(self, 1435, 800)  
 self.grid\_columnconfigure(0, weight=0)  
 self.grid\_columnconfigure(1, weight=1)  
  
 # Фрейм для элементов (текста и кнопок)  
 left\_frame = ttk.Frame(self)  
 left\_frame.grid(row=0, column=0, padx=20, pady=20, sticky="nw")  
  
 welcome\_label = ttk.Label(left\_frame, text="NAVIGATOR3000", font=("Arial", 16, "bold"), anchor="center")  
 welcome\_label.grid(row=0, column=0, columnspan=2, pady=(0, 10), sticky="ew")  
  
 try:  
 image = Image.open("map\_final.png")  
 self.tk\_image = ImageTk.PhotoImage(image)  
 self.image\_label = ttk.Label(self, image=self.tk\_image)  
 self.image\_label.grid(row=0, column=1, padx=2, pady=16)  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Ошибка загрузки изображения: {e}")  
  
 # Фрейм для ввода данных  
 frame = ttk.Frame(left\_frame)  
 frame.grid(row=1, column=0, padx=20, pady=20, sticky="ew")  
  
 # Поля ввода и лейблы внутри frame  
 ttk.Label(frame, text="Город отправления:").grid(row=0, column=0, padx=5, pady=5, sticky="w")  
 self.from\_city\_entry = ttk.Entry(frame)  
 self.from\_city\_entry.grid(row=0, column=1, padx=5, pady=5, sticky="ew")  
  
 ttk.Label(frame, text="Город назначения:").grid(row=1, column=0, padx=5, pady=5, sticky="w")  
 self.to\_city\_entry = ttk.Entry(frame)  
 self.to\_city\_entry.grid(row=1, column=1, padx=5, pady=5, sticky="ew")  
  
 ttk.Label(frame, text="Средняя скорость (км/ч):", anchor="w").grid(row=2, column=0, padx=5, pady=5, sticky="w")  
 self.speed\_entry = ttk.Entry(frame)  
 self.speed\_entry.grid(row=2, column=1, padx=5, pady=5, sticky="ew")  
  
 ttk.Label(frame, text="Расход топлива (л/100км):", anchor="w").grid(row=3, column=0, padx=5, pady=5, sticky="w")  
 self.fuel\_consumption\_entry = ttk.Entry(frame)  
 self.fuel\_consumption\_entry.grid(row=3, column=1, padx=5, pady=5, sticky="ew")  
  
 ttk.Label(frame, text="Объем бака (л):", anchor="w").grid(row=4, column=0, padx=5, pady=5, sticky="w")  
 self.fuel\_tank\_capacity\_entry = ttk.Entry(frame)  
 self.fuel\_tank\_capacity\_entry.grid(row=4, column=1, padx=5, pady=5, sticky="ew")  
  
 frame.columnconfigure(1, weight=1) # Растягиваем поле ввода  
  
 # Кнопка расчета маршрута  
 calc\_button = ttk.Button(left\_frame, text="Рассчитать маршрут", command=self.calculate\_route,  
 bootstyle="primary")  
 calc\_button.grid(row=2, column=0, columnspan=2, pady=10)  
  
 # Поле для вывода результата  
 self.result\_text = ttk.Text(left\_frame, height=25, wrap="word", state="disabled")  
 self.result\_text.grid(row=3, column=0, columnspan=2, padx=20, pady=10, sticky="nsew")  
  
 # Фрейм для кнопок "История" и "Выйти"  
 button\_frame = ttk.Frame(left\_frame)  
 button\_frame.grid(row=4, column=0, columnspan=2, pady=10, sticky="ew")  
  
 # Кнопка для выхода  
 exit\_button = ttk.Button(button\_frame, text="Выйти", command=self.go\_back, bootstyle="danger-outline")  
 exit\_button.grid(row=0, column=0, padx=10, pady=10, sticky="e")  
  
 # Кнопка истории  
 history\_button = ttk.Button(button\_frame, text="История поездок", command=self.show\_trip\_history,  
 bootstyle="info")  
 history\_button.grid(row=0, column=1, padx=10, pady=10, sticky="w")  
  
 # Настраиваем столбцы button\_frame  
 button\_frame.columnconfigure(0, weight=1)  
 button\_frame.columnconfigure(1, weight=1)  
  
 # Настраиваем столбцы left\_frame  
 left\_frame.columnconfigure(0, weight=1)  
  
 def calculate\_route(self):  
 city\_from = self.from\_city\_entry.get()  
 city\_to = self.to\_city\_entry.get()  
 try:  
 avg\_speed = float(self.speed\_entry.get())  
 fuel\_consumption = float(self.fuel\_consumption\_entry.get())  
 fuel\_tank\_capacity = float(self.fuel\_tank\_capacity\_entry.get())  
 except ValueError:  
 messagebox.showerror("Ошибка",  
 "Пожалуйста, введите корректные числовые значения для параметров автомобиля.")  
 return  
  
 conn = connect\_to\_database(self.db\_path)  
 try:  
 if conn:  
 route\_data = calculate\_route(conn, city\_from, city\_to, avg\_speed, fuel\_consumption, fuel\_tank\_capacity)  
 if route\_data:  
 self.display\_route\_data(route\_data, city\_from, city\_to)  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Ошибка расчета маршрута: {e}")  
 finally:  
 if conn:  
 conn.close()  
  
  
  
 def display\_route\_data(self, route\_data, city\_from, city\_to):  
 if not route\_data:  
 messagebox.showerror("Ошибка", "Маршрут не найден или произошла ошибка при расчёте.")  
 return  
 city\_names, total\_distance, total\_time, stops\_for\_fuel, stops\_for\_sleep, infrastructure = route\_data  
  
 # Формируем текст для вывода  
 output\_text = f"Маршрут: {' -> '.join(city\_names)}\n"  
 output\_text += f"Общее расстояние: {total\_distance:.2f} км\n"  
 output\_text += f"Общее время в пути: {total\_time:.2f} часов\n"  
  
 history\_text = f"{city\_from} -> {city\_to}\t"  
 history\_text += f"Расстояние: {total\_distance:.2f} км\t"  
 history\_text += f"Общее время: {total\_time:.2f} часов"  
  
 if stops\_for\_fuel:  
 output\_text += "\nРекомендуемые остановки на заправку:\n"  
 for station, address in stops\_for\_fuel:  
 output\_text += f"- {station} (км от ближайшего города: {address})\n"  
  
 if stops\_for\_sleep:  
 output\_text += "\nРекомендуемые остановки на ночлег:\n"  
 for hotel, address in stops\_for\_sleep:  
 output\_text += f"- {hotel} (км от ближайшего города: {address})\n"  
  
 output\_text += "\nОбъекты инфраструктуры на маршруте:\n"  
 for obj\_info in infrastructure:  
 obj\_type, obj\_name, distance, start\_end = obj\_info  
 output\_text += f"- {obj\_type}: {obj\_name} (на {distance} км от начала маршрута, {start\_end})\n"  
  
  
  
 # Выводим результат в текстовое поле  
 self.result\_text.configure(state='normal')  
 self.result\_text.delete("1.0", ttk.END)  
 self.result\_text.insert(ttk.END, output\_text)  
 self.result\_text.configure(state='disabled')  
  
 # Сохраняем историю поездки  
 def history\_save(user, route\_details):  
 conn = connect\_to\_database(self.db\_path)  
 if conn:  
 history\_save\_db(conn, user, route\_details)  
  
 history\_save(self.user\_id, history\_text)  
  
 # Добавить метод для перехода в окно истории  
 def show\_trip\_history(self):  
 self.destroy()  
 history\_window = TripHistoryWindow(self.db\_path, self.user\_id)  
 history\_window.mainloop()  
  
 def go\_back(self):  
 self.destroy()  
 start\_window = StartWindow(self.db\_path)  
 start\_window.mainloop()  
  
 def exit\_program(self):  
 self.destroy()  
  
class TripHistoryWindow(ttk.Window):  
 def \_\_init\_\_(self, db\_path, user\_id):  
 super().\_\_init\_\_(title="История поездок", themename="morph")  
 self.db\_path = db\_path  
 self.user\_id = user\_id  
 center\_window(self, 800, 600)  
 self.resizable(False, False)  
  
 ttk.Label(self, text="История поездок", font=("Arial", 16), anchor="center").pack(pady=10)  
  
 # Список для отображения маршрутов  
 self.history\_listbox = ttk.Treeview(  
 self,  
 columns=("ID", "Date", "Details"),  
 show="headings",  
 height=20  
 )  
 self.history\_listbox.heading("ID", text="ID маршрута")  
 self.history\_listbox.heading("Date", text="Дата")  
 self.history\_listbox.heading("Details", text="Детали маршрута")  
 self.history\_listbox.column("ID", width=50, anchor="center")  
 self.history\_listbox.column("Date", width=150, anchor="center")  
 self.history\_listbox.column("Details", width=600, anchor="w")  
 self.history\_listbox.pack(padx=10, pady=10, fill="both", expand=True)  
  
 # Загрузка данных истории  
 self.load\_trip\_history()  
  
 # Кнопка возврата  
 back\_button = ttk.Button(self, text="Назад", command=self.go\_back, bootstyle="secondary-outline")  
 back\_button.pack(pady=10)  
  
 def load\_trip\_history(self):  
 conn = connect\_to\_database(self.db\_path)  
 if not conn:  
 messagebox.showerror("Ошибка", "Не удалось подключиться к базе данных.")  
 return  
  
 try:  
 cursor = conn.cursor()  
 query = "SELECT TripID, RouteDate, RouteDetails FROM TripsHistory WHERE UserID = ? ORDER BY RouteDate DESC"  
 cursor.execute(query, (self.user\_id,))  
 trips = cursor.fetchall()  
  
 # Очистить текущие записи  
 for row in self.history\_listbox.get\_children():  
 self.history\_listbox.delete(row)  
  
 # Добавить данные в список  
 for trip in trips:  
 trip\_id, date, details = trip  
 self.history\_listbox.insert("", "end", values=(trip\_id, date, details))  
 except Exception as e:  
 messagebox.showerror("Ошибка", f"Ошибка при загрузке истории поездок: {e}")  
 finally:  
 conn.close()  
  
 def go\_back(self):  
 self.destroy()  
 main\_window = MainWindow(self.db\_path, self.user\_id)  
 main\_window.mainloop()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 MDB\_PATH = r"C:/users/andrew/Desktop/Проект БАЗЫ ДАННЫХ/project.accdb" # Укажите путь к вашей БД  
 start\_window = StartWindow(MDB\_PATH)  
 start\_window.mainloop()

**Модуль Routes\_logic:**

from tkinter import messagebox  
import heapq  
from db\_utils import get\_object\_type\_name  
  
# Функция для построения графа городов  
def build\_graph(conn):  
 cursor = conn.cursor()  
 query = "SELECT FromCityID, ToCityID, DistanceKm FROM Routes"  
 cursor.execute(query)  
 graph = {}  
 for from\_city, to\_city, distance in cursor.fetchall():  
 if from\_city not in graph:  
 graph[from\_city] = []  
 graph[from\_city].append((to\_city, distance))  
 # Двусторонний маршрут  
 if to\_city not in graph:  
 graph[to\_city] = []  
 graph[to\_city].append((from\_city, distance))  
 return graph  
  
  
# Алгоритм поиска кратчайшего пути (Dijkstra)  
def find\_shortest\_path(graph, start, end):  
 distances = {node: float('inf') for node in graph}  
 previous\_nodes = {node: None for node in graph}  
 distances[start] = 0  
 priority\_queue = [(0, start)]  
  
 while priority\_queue:  
 current\_distance, current\_node = heapq.heappop(priority\_queue)  
  
 if current\_node == end:  
 break  
  
 for neighbor, weight in graph.get(current\_node, []):  
 distance = current\_distance + weight  
 if distance < distances[neighbor]:  
 distances[neighbor] = distance  
 previous\_nodes[neighbor] = current\_node  
 heapq.heappush(priority\_queue, (distance, neighbor))  
  
 # Восстановление пути  
 path, current = [], end  
 while previous\_nodes[current] is not None:  
 path.append(current)  
 current = previous\_nodes[current]  
 path.append(start)  
 path.reverse()  
  
 return path, distances[end]  
  
# Функция для получения объектов инфраструктуры на маршруте  
def get\_infrastructure\_on\_route(conn, route\_id):  
 cursor = conn.cursor()  
 query = """  
 SELECT ObjectType, ObjectName, DistanceFromCity  
 FROM InfrastructureObjects  
 WHERE OnRouteID = ?  
 """  
 cursor.execute(query, (route\_id,))  
 results = cursor.fetchall()  
 infrastructure = []  
 for obj\_type, obj\_name, distance\_from\_city in results:  
 obj\_type\_name = get\_object\_type\_name(conn, obj\_type) # из db\_utils  
 infrastructure.append((obj\_type\_name, obj\_name, distance\_from\_city))  
 return infrastructure  
  
  
# Основная логика маршрута с расчётами  
def calculate\_route(conn, city\_from, city\_to, avg\_speed, fuel\_consumption, fuel\_tank\_capacity):  
 # Проверка наличия городов в базе данных  
 from db\_utils import check\_city\_exists, get\_city\_name # Импорт функций для db\_utils в локальный scope  
 from\_city = check\_city\_exists(conn, city\_from)  
 to\_city = check\_city\_exists(conn, city\_to)  
  
 if not from\_city or not to\_city:  
 messagebox.showerror("Ошибка", "Пожалуйста, заполните оба поля.")  
 return  
  
 from\_city\_id = from\_city[0]  
 to\_city\_id = to\_city[0]  
  
 # Построение графа и поиск кратчайшего пути  
 graph = build\_graph(conn)  
 shortest\_path, total\_distance = find\_shortest\_path(graph, from\_city\_id, to\_city\_id)  
  
 # Расчёты  
 total\_time = total\_distance / avg\_speed # Время в пути (часы)  
 fuel\_range = fuel\_tank\_capacity / fuel\_consumption \* 100 # Запас хода на полном баке (км)  
 stops\_for\_fuel = []  
 stops\_for\_sleep = []  
 current\_distance = 0  
 driving\_time = 0  
  
 all\_infrastructure = []  
 city\_names = [get\_city\_name(conn, city\_id) for city\_id in shortest\_path]  
  
 for i in range(len(shortest\_path) - 1):  
 cursor = conn.cursor()  
 query = """  
 SELECT RouteID, DistanceKm FROM Routes  
 WHERE (FromCityID = ? AND ToCityID = ?) OR (FromCityID = ? AND ToCityID = ?)  
 """  
 cursor.execute(query, (shortest\_path[i], shortest\_path[i + 1], shortest\_path[i + 1], shortest\_path[i]))  
 route = cursor.fetchone()  
 if route:  
 route\_id, segment\_distance = route  
 current\_distance += segment\_distance  
 driving\_time += segment\_distance / avg\_speed  
  
 infrastructure = get\_infrastructure\_on\_route(conn, route\_id)  
 all\_infrastructure.extend(  
 [(obj[0], obj[1], obj[2], f"{city\_names[i]} -> {city\_names[i + 1]}") for obj in infrastructure])  
  
 if current\_distance > fuel\_range:  
 fuel\_stations = [obj for obj in infrastructure if obj[0] == "АЗС"]  
 if fuel\_stations:  
 stops\_for\_fuel.append((fuel\_stations[0][1], fuel\_stations[0][2])) # Название и адрес первой АЗС  
 current\_distance = 0  
  
 if driving\_time > 9: # Максимум 9 часов за рулём  
 hotels = [obj for obj in infrastructure if obj[0] == "Отель"]  
 if hotels:  
 stops\_for\_sleep.append((hotels[0][1], hotels[0][2])) # Название и адрес отеля  
 driving\_time = 0  
  
 overnight\_stays = total\_time // 8 # Остановки на ночлег  
  
 return (city\_names, total\_distance, total\_time, stops\_for\_fuel, stops\_for\_sleep, all\_infrastructure)

**Модуль Db\_utils:**

from hash\_func import \*  
import pyodbc  
  
# Функция для подключения к базе данных  
def connect\_to\_database(db\_path):  
 connection\_string = (  
 f"DRIVER={{Microsoft Access Driver (\*.mdb, \*.accdb)}};"  
 f"DBQ={db\_path};"  
 )  
 try:  
 conn = pyodbc.connect(connection\_string)  
 return conn  
 except pyodbc.Error as e:  
 print(f"Ошибка при подключении к базе данных: {e}")  
 return None  
  
def history\_save\_db(conn, UserID, RouteDetails):  
 cursor = conn.cursor()  
 try:  
 query\_insert = "INSERT INTO TripsHistory (UserID, RouteDetails) VALUES (?, ?)"  
 cursor.execute(query\_insert, UserID, RouteDetails)  
 conn.commit()  
 print("В истории успешно записано")  
 except pyodbc.Error as e:  
 conn.rollback()  
 return None  
  
def register\_user(conn, username, password):  
 cursor = conn.cursor()  
 hashed\_password = hash\_password(password) # вызываем функцию хэширования из файла hash\_func.py  
  
 # Проверяем, существует ли пользователь с таким именем  
 try:  
 query\_check = "SELECT UserID FROM Users WHERE Username = ?"  
 cursor.execute(query\_check, (username,))  
 user = cursor.fetchone()  
 if user:  
 print("Пользователь с таким именем уже существует.")  
 return None  
 except pyodbc.Error as e:  
 print(f"Ошибка при проверке пользователя: {e}")  
 return None  
  
  
 # Добавляем нового пользователя в базу данных  
 try:  
 query\_insert = "INSERT INTO Users (Username, Password) VALUES (?, ?)"  
 cursor.execute(query\_insert, (username, hashed\_password))  
 conn.commit()  
 print("Регистрация успешна!")  
 except pyodbc.Error as e:  
 print(f"Ошибка при регистрации пользователя: {e}")  
 conn.rollback()  
 return None  
  
 # Получаем ID нового пользователя  
 try:  
 query\_get\_id = "SELECT UserID FROM Users WHERE Username = ?"  
 cursor.execute(query\_get\_id, (username,))  
 user\_id = cursor.fetchone()[0]  
 return user\_id  
 except (pyodbc.Error, TypeError) as e:  
 print(f"Ошибка при получении ID пользователя: {e}")  
 return None  
  
def login\_user(conn, username, password):  
 cursor = conn.cursor()  
 # Получаем хэш пароля из базы данных  
 query = "SELECT UserID, Password FROM Users WHERE Username = ?"  
 cursor.execute(query, (username,))  
 user\_data = cursor.fetchone()  
  
 if user\_data:  
 user\_id, stored\_hashed\_password = user\_data  
  
 # Проверяем пароль с помощью verify\_password  
 if verify\_password(password, stored\_hashed\_password): # вызываем функцию из hash\_func.py  
 print("Авторизация успешна!")  
 return user\_id  
 else:  
 print("Неверный пароль.")  
 return None  
 else:  
 print("Неверное имя пользователя.")  
 return None  
  
# Функция для проверки существования города  
def check\_city\_exists(conn, city\_name):  
 cursor = conn.cursor()  
 query = "SELECT CityID FROM Cities WHERE CityName = ?"  
 cursor.execute(query, (city\_name,))  
 return cursor.fetchone()  
  
# Функция для получения имени города по ID  
def get\_city\_name(conn, city\_id):  
 cursor = conn.cursor()  
 query = "SELECT CityName FROM Cities WHERE CityID = ?"  
 cursor.execute(query, (city\_id,))  
 result = cursor.fetchone()  
 return result[0] if result else None  
  
# Функция для получения названия типа объекта по ID  
def get\_object\_type\_name(conn, object\_type\_id):  
 cursor = conn.cursor()  
 query = "SELECT ObjectTypeName FROM ObjectTypes WHERE ObjectTypeID = ?"  
 cursor.execute(query, (object\_type\_id,))  
 result = cursor.fetchone()  
 return result[0] if result else None

**Модуль Hash\_func:**

import hashlib

def hash\_password(password: str) -> str:  
 if not isinstance(password, str):  
 raise TypeError()  
 encoded\_password = password.encode('utf-8')  
 hashed\_password = hashlib.sha256(encoded\_password).hexdigest()  
 return hashed\_password  
  
def verify\_password(password: str, hashed\_password: str) -> bool:  
 if not isinstance(password, str):  
 raise TypeError()  
 if not isinstance(hashed\_password, str):  
 raise TypeError()  
  
 rehashed\_password = hash\_password(password)  
 return rehashed\_password == hashed\_password