SpotterAssessment PDF Guide

Next-Gen Fuel Route Optimization System

1. Introduction

1.1 Overview

The Fuel Route Optimization System is a **high-performance** travel assistant that calculates the most fuel-efficient route between two locations in the USA. The system automatically selects **cost-effective fuel stops**, generates an interactive route map, and provides **real-time cost estimates** for the journey.

NOTE: Due to plan limitations and large CSV size only a chunk of csv data is cached and stored.(100 Locations). The caching script has multi-async logic where with multiple keys the complete csv can be cached and loaded.

1.2 Key Features

- **Cache powered route optimization for faster responses.**
- **Automated fuel stop selection**
- **Interactive route visualization**
- **Optimized performance with caching**
- **Enterprise-grade location finding logic**
- **Best visual experience with detailed documented guide**

2. Tech Stack

Backend Technologies

- **Django + Django REST Framework** – API development
- **Celery + Redis** – Background task execution -(***To add Celery for cache updating worker)***
- **GraphHopper API** – Route calculation - [FREE Plan]
- **OpenCage Geocoder API** – Geolocation services - [FREE Plan]

Frontend & Visualization

- **Folium** – Interactive map generation
- **Modern UI/UX principles** applied
- **HTML CSS and JS for Basic UI**
- **Swagger and Redoc Integration is done for better API playground support.**

Data & Intelligence

- **Fuel Prices CSV** – Live fuel price data (*Provided in assessment email*)
- **ML-ready architecture** for predictive analytics
- **Indepth and aggressive algorithms using powerful cache mechanisms**
- **Detailed API for integrating**

---

3. System Workflow

3.1 Route Processing

1. **User enters start and finish locations**
2. System validates locations as **within the USA**
3. **GraphHopper API** generates the optimized route
4. **Fuel stops are optimized** for cost efficiency and spacing
5. System calculates **total fuel cost**
6. **Response includes:**

- **Route map with fuel stops**
- **List of fuel stops & real-time prices**
- **Total trip fuel cost**

7. Strong Cache layer will cache if new request and use cached request if present in cache.

---

4. API Endpoints

Endpoint	Method	Description
`/get-route/`	`GET`	Fetch optimized route data
`/route/`	`POST`	Returns full route & fuel stops
`/route/simple/`	`GET`	Fetch a simplified fuel route

---

5. Backend Architecture

5.1 Performance Optimization

- **Redis caching** for reduced API response times
- **Parallel processing** for rapid computation

5.2 Fuel Stop Selection Strategy

- **Prioritizes lowest fuel prices along the route**
- **Ensures no refuel gap exceeds 500 miles**
- **Stops selected within 10-mile range of the route**

5.3 Security & Reliability

- **JWT Authentication** for secure API access
- **Built-in error handling** for stability

- **Scalable architecture** for high-load support

6. Deployment Strategy

- Hosted on **Ubuntu servers** with **Docker containers**
- **NGINX** for optimized request handling
- **Gunicorn** for scalable WSGI execution

7. Future Enhancements

- **AI-driven fuel price forecasting**
- **Mobile application integration**
- **User profiles & route history tracking**

8. Visuals & Documentation

Company Logo: *(Insert image here)*

Route Map Example: *(Insert screenshot here)*

Fuel Stop Data Sample: *(Insert data table here)*

Database Schema Diagram: *(Attach ER diagram here)*

9. Additional Notes

To create *cache_initializer*

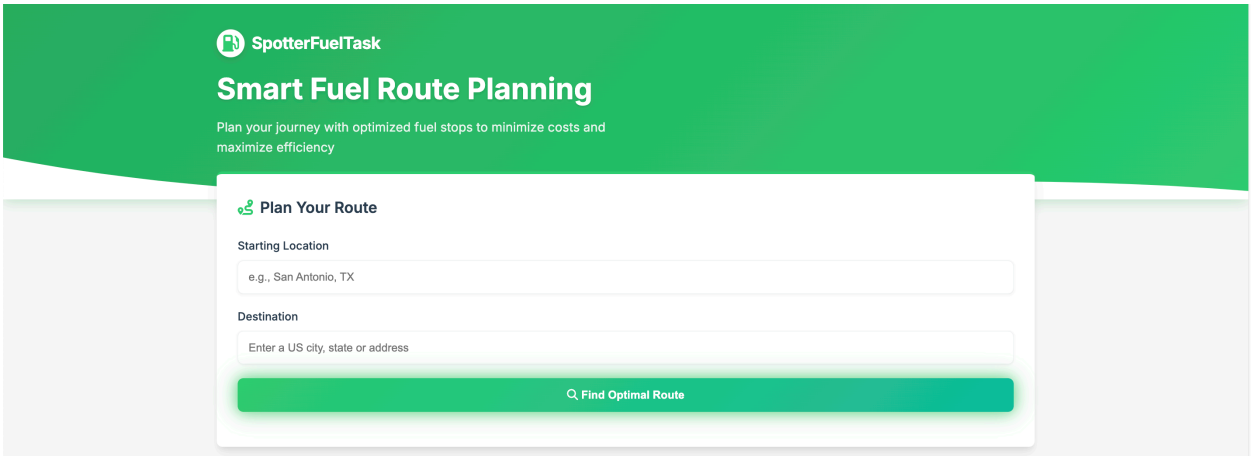
```
mkdir -p FuelRouter/management/commands
touch FuelRouter/management/commands/
init .py
touch FuelRouter/management/commands/initialize_cache.py
```

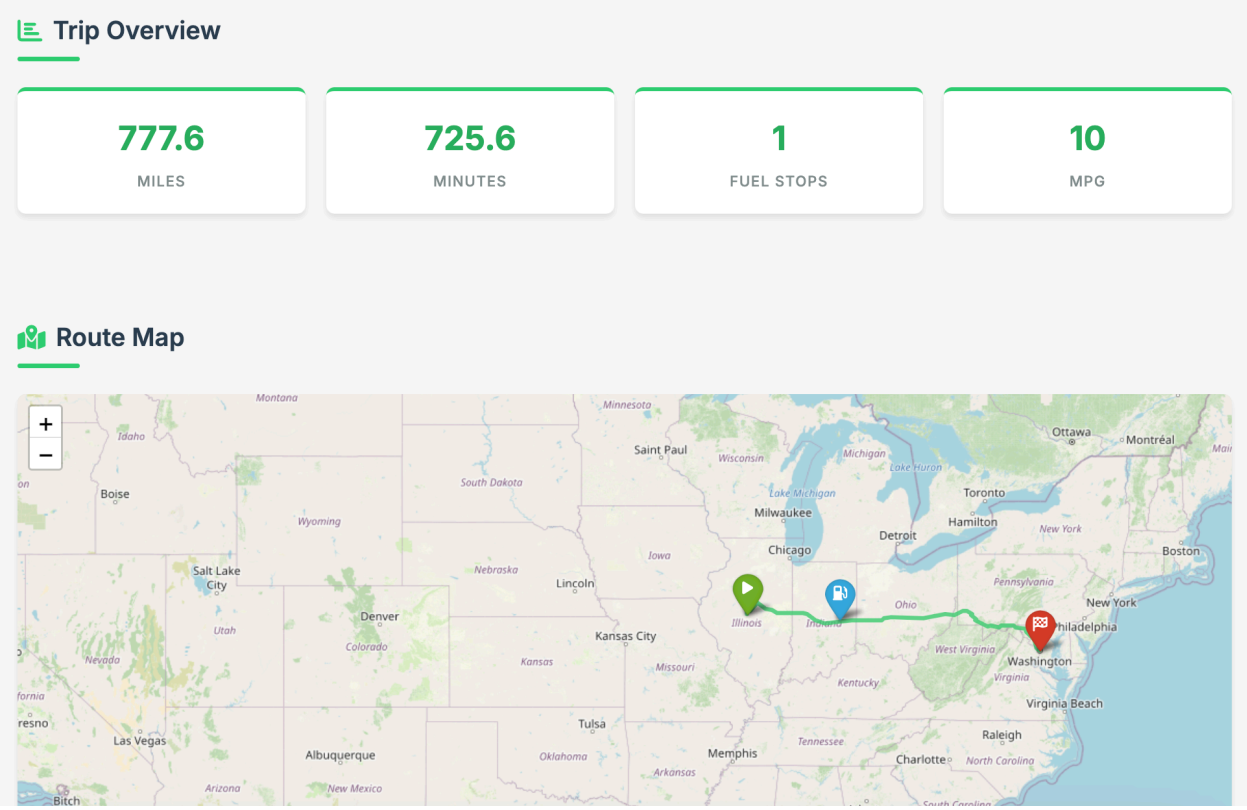
10.Local CURL Commands

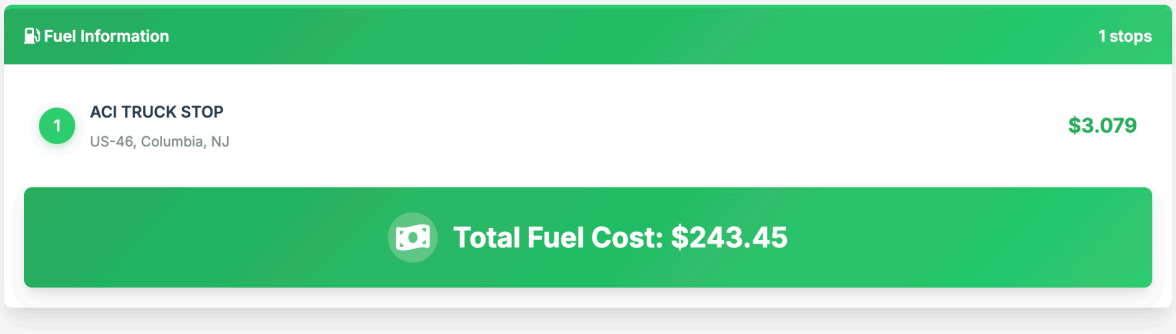
```
curl --location ' http://127.0.0.1:8000/api/route/ ' \
--header 'Content-Type: application/json' \
--data '{
"start_location": "Chicago, IL, USA",
"end_location": "New York, NY, USA"
}'
```

11.Local UI Demo

http://127.0.0.1:8000/api/spotter-fuel-dashboard/







The UI and the API uses caching layer. Cached responses are less than 1s and are accurate.

The complete csv can be cached at once with more keys as async logic is in place.

POSTMAN collection is in the codebase.

If you have any questions please contact me on ***rcviit4196@gmail.com***

Thanks. Looking forward to work with you.