Smart Routing System (SRS)

Sam Shtrambrand and Christopher Cleus

1. Description of the Project

The Smart Routing System (SRS) functions as an integrated emergency navigation platform which uses maps to help users locate their closest emergency shelters. The system uses FastAPI backend functionality together with HTML and JavaScript frontend capabilities to achieve real time geolocation tracking and shelter location detection and route calculation. The system first obtains user coordinates by either performing address search or enabling "Use My Location" before it executes a backend model to assess shelters based on their distance and capacity and risk evaluation outcomes. The system shows shelter locations through Leaflet mapping and enables users to access OSRM road directions for navigating to their chosen shelters. The project shows how to build an emergency routing system for safe operations through the combination of computational methods with geospatial algorithms and modern web development tools.

2. Significance of the Project

The SRS project solves an essential issue which affects both emergency response operations and crisis management infrastructure. People face elevated uncertainty levels while their information access becomes limited during natural disasters and hazardous situations while they need to find safe locations immediately. The current tools which include printed evacuation maps and general purpose navigation applications fail to show shelter capacity and risk scores and live hazard assessments so they cannot offer situation specific guidance. The Smart Routing System provides an improved system which determines routes by combining physical distance with safety parameters. The system demonstrates that algorithmic decision systems improve shelter search results through faster search times and route optimization and complex data presentation via an intuitive interface. The project achieves its uniqueness through its combination of routing algorithms with hazard evaluation metrics and real time geolocation and dynamic map visualization into a unified application that runs independently on a local machine.

3. Code Structure

The system uses a modular design which divides operational functions from the user interface elements. The backend system contains all algorithmic operations through its FastAPI application and shelter dataset and graph module which handles route modeling and shelter scoring and correctness verification. The system uses the graph module to construct its road network while performing capacity and hazard assessments and executing Dijkstra for system operations. The frontend operates through a single HTML document which contains all JavaScript code and Leaflet functionality needed to display maps and process user input and marker placement and location storage and route drawing. The run_app.py launcher script allows users to start the complete project by running one command which activates both backend server and local HTTP server for interactive map access. The system design with separate components allows developers to understand its internal operations through independent testing and extension of its components.

Root Directory Structure:

SRS/

├── backend/

│ ├── app.py

│ ├── data/

│ └── graph/

│ ├── build_graph.py

│ ├── dijkstra.py

│ ├── hazard.py

│ ├── capacity.py

│ └── verifier.py

├── frontend/

│ └── index.html

├── run_app.py

└── requirements.txt

4. Description of Algorithms

The Smart Routing System achieves its final version through geospatial distance calculation and external routing services instead of using Dijkstra's algorithm or other graph based pathfinding methods. The operational system determines shelter locations and travel routes by performing geospatial distance calculations and by integrating with external routing services. The system uses the Haversine formula to find the closest shelters to users because this formula calculates the Earth surface distance between two points. The backend system uses coordinate calculations to find the shortest distance between user locations and shelter locations which enables it to generate shelter recommendations based on proximity. The system uses this method to create emergency route suggestions because it offers simple operation with suitable results. The backend system determines the closest shelters by performing geographic calculations but it uses Open Source Routing Machine (OSRM) to generate turn by turn directions through its public API by sending origin destination pairs. The OSRM system takes the origin destination pair from the backend to create a complete road based navigation route which includes route geometry data for interactive map rendering. The system achieves its lightweight backend design through this approach because it uses OSRM's advanced routing engines which run contraction hierarchies and other sophisticated algorithms. The system uses Haversine formula calculations to find nearby shelters and OSRM routing technology to create navigation routes. The system achieves fast and dependable routing operations through this method which eliminates the need to build a custom road network or perform internal pathfinding operations.

5. Verification of Algorithms

The system algorithm verification process checked how well the shelter distance calculations worked and how stable the API routing system was through multiple test cases. The system verified nearest shelter identification through manual distance verification by using controlled toy examples with known geographic points. The Haversine implementation received testing through small coordinate pairs which covered one degree of latitude or longitude to confirm its great circle distance calculations. The system used established city-to-city distance measurements between Philadelphia and New York City to verify that its calculated values matched official geographic data. The OSRM API received mock coordinates to test its route generation capabilities and verify that the produced path geometry matched the GeoJSON output of Leaflet. The system tested three different scenarios which included urban routes near each other and suburban routes that spanned longer distances and situations where OSRM failed to generate routes because of restricted areas or water bodies. The system verified that OSRM produced structured route information when routes existed and it processed error messages correctly when no route was available. The verification process showed that the system produced proper shelter rankings through distance calculations and it generated accurate navigation paths by using OSRM path generation and geospatial distance calculations. The system demonstrates

complete functionality through its main operational elements which provide accurate results and dependable performance and user-friendly interface.

6. Functionalities

The Smart Routing System enables users to access fundamental operational features which they need to operate the system. Users can search for shelters by address entry or the device can detect their current GPS location to start the search. The system fetches the ten closest shelters which appear on the map interface together with their available space and their calculated distance from the user. Users can access the "Get Directions" feature for each shelter to obtain complete driving instructions from OSRM routing servers. The system displays visual signs which show the user's present location. The system maintains real time marker updates while it removes outdated routes to enable users to navigate through its smooth panning system and pop up information markers.

7. Execution Results and Analysis

The system execution proved that the backend and frontend systems functioned as one system because both address based and GPS based search functions operated properly. The system shows nearby shelters through its dynamic map which provides exact routes for navigation. The system produced correct shelter ordering results for multiple location searches through its geographic distance calculation. The system employs an external routing engine which produces routes that follow actual road paths instead of using Euclidean distance calculations. The system logs show that backend requests to /nearest handle multiple consecutive requests which produce fast response times. The final system achieves successful integration between classical algorithms and modern web technologies and geographic APIs to create an operational emergency response system.

8. Conclusions

The Smart Routing System achieves its goal through the combination of geospatial processing with real time user interface functionality which establishes an operational emergency route system. The development process taught me about API connection methods and asynchronous frontend programming and state control systems and algorithm testing procedures. The system achieved success through its integration of real time location tracking with adaptive route generation capabilities. The system faces two main restrictions because it depends on external routing servers and because it uses basic shelter data representations. The present system architecture allows developers to implement new functionality through hazard based routing and multiple shelter assessment approaches and route creation capabilities that operate independently of internet connections.

9. Overall Quality of Report

The Smart Routing System needed multiple software architecture changes and documentation updates and multiple debugging sessions to achieve its final version. The development team built separate system parts which had straightforward design elements while maintaining the existing organizational framework. The editing process aimed to create direct language which shows the project's technical complexity through its detailed documentation. The final output shows complete technical operation while meeting all academic documentation requirements at an exceptional level. The documentation will benefit from additional diagrams and code annotations and testing frameworks integration during future development to enhance its clarity and technical precision.

10. References

https://overpass-turbo.eu/ for shelter collection data

https://leafletjs.com/

https://project-osrm.org/

https://uvicorn.dev/

https://www.movable-type.co.uk/scripts/latlong.html

https://docs.python.org/3/library/http.server.html