System Documentation

1. Introduction

The Leon Evacuation System is a web-based decision support tool built with Streamlit. Its primary purpose is to determine the optimal evacuation route from various barangays in Leon, Iloilo, to a central evacuation center (Poblacion).

The system's core feature is its use of a fuzzy logic model combined with the A* (A-star) pathfinding algorithm. Instead of simply finding the *shortest* path (by distance), it finds the *best* path by evaluating a "cost" based on real-world road characteristics, including:

- Road Slope: How steep or flat the road is.
- Travel Time: The time required to traverse a road segment.
- Path Curvature: How winding or straight the road is.

This allows the system to prioritize routes that are safer and faster, even if they are not the shortest in distance.

2. System Architecture

The application follows a 3-tier architecture, all running within a single Python script environment:

1. Presentation Layer (Frontend):
    - Technology: streamlit
    - Components: The main() function.
    - Responsibility: Renders the web interface, including the title, sidebar, barangay selection dropdown, and metric displays. It's responsible for all user interaction and for displaying the final map generated by the logic layer.
2. Logic Layer (Backend):
    - Technology: EvacuationSystem class, skfuzzy, numpy, heapq.
    - Components: EvacuationSystem class and all its methods (a_star_path, fuzzy_evaluation, get_edge_cost, etc.).
    - Responsibility: Encapsulates all business logic. It holds the application's state (loaded paths, travel data, graph connections). It executes the A* pathfinding, calculates the fuzzy logic costs, and generates the Folium map object.
3. Data Layer (Storage):
    - Technology: pandas, xml.etree.ElementTree, os.

  - Components: The data/ folder, .kml file, and .xlsx files.
  - Responsibility: Provides the raw data. The logic layer interacts with this layer via loader functions (load_kml_paths, load_all_travel_data) to read data into memory.

3. Detailed Component & Function Reference

This section details the key functions and methods used in the system.

3.1. Main Application & Data Loading

These functions control the Streamlit UI and initiate the data loading process.

- main():
  - Purpose: The main entry point for the Streamlit application.
  - Logic: Sets up the page configuration, title, and sidebar. It manages the session state (st.session_state) to hold the EvacuationSystem object. It handles the UI logic for the barangay dropdown and calls the appropriate EvacuationSystem methods to calculate and display the map and metrics.
- load_embedded_files():
  - Purpose: Scans the data/ directory to find the KML and Excel files needed by the system.
  - Returns: A tuple (kml_file, excel_files) containing the path to the KML file (or None) and a list of paths to Excel files (or an empty list).

3.2. EvacuationSystem Class (Core Logic)

This class is the "brain" of the application, holding all data and core methods.

- __init__(self, kml_file_path=None, excel_filepaths=None):
  - Purpose: The class constructor.
  - Logic: Initializes key data structures like self.barangays, self.distances, and self.evacuation_paths. It builds the graph connections (_build_graph_connections) and triggers the data loading methods (load_kml_paths, load_all_travel_data).
- _build_graph_connections(self):
  - Purpose: Defines the static road network graph (adjacency list) for the A* algorithm.
  - Logic: Returns a dictionary where keys are barangay names and values are lists of connected barangays.
- load_kml_paths(self):
  - Purpose: Loads and parses the .kml file.

    - Logic: Uses xml.etree.ElementTree to read the KML. It iterates through <Placemark> tags, extracts the <name> and <coordinates>, and stores the coordinate list in self.evacuation_paths. This data is used for visualization only.
- load_all_travel_data(self):
    - Purpose: Loads and parses all .xlsx files.
    - Logic: Uses pandas to read each Excel file. It parses the filename (e.g., Poblacion_to_Bacolod.xlsx) to identify the destination barangay ("Bacolod"). It reads the "Slope" and "Travel_Time_min" columns, row by row, and stores this segment data in self.travel_data. This data is used for cost calculation.

3.3. Pathfinding & Heuristics (A* Algorithm)

These methods are responsible for finding the optimal path.

- a_star_path(self, start, goal):
    - Purpose: Implements the A* pathfinding algorithm.
    - Logic: Uses a heapq (priority queue) to explore nodes based on the lowest $f(n)$ cost. It tracks the $g(n)$ cost (actual fuzzy cost from start) and $h(n)$ cost (heuristic distance to goal). It calls get_edge_cost to find the cost between nodes and heuristic to estimate the remaining cost.
    - Returns: A tuple (barangay_name, cost, time, distance) for the best path found, or None.
- heuristic(self, node, goal):
    - Purpose: The heuristic function $h(n)$ for A*.
    - Logic: A wrapper that calls haversine_distance to get the straight-line distance (in km) between the current node and the goal node.
- haversine_distance(self, coord1, coord2):
    - Purpose: Calculates the great-circle (straight-line) distance between two latitude/longitude points.
    - Logic: Implements the Haversine formula.
    - Returns: The distance in kilometers.

3.4. Cost Calculation (Fuzzy Logic)

These methods define the "cost" of traveling along a path.

- get_edge_cost(self, from_node, to_node):
    - Purpose: The "actual cost" function $g(n)$ for A*.
    - Logic: If travel data exists for the to_node, it iterates through all its road segments (from self.travel_data) and sums the cost of each segment by

calling fuzzy_evaluation. It also sums the total travel time. If no data exists, it falls back to a simple distance-based cost.

    - Returns: A tuple (total_cost, total_time).

- fuzzy_evaluation(self, slope, travel_time, curvature):
    - Purpose: The core fuzzy logic model.
    - Logic:
        1. Fuzzification: Maps the crisp input values (e.g., slope = 3.5) to fuzzy membership sets (e.g., slope_med = 0.2, slope_high = 0.7) using skfuzzy.trimf and skfuzzy.interp_membership.
        2. Rule Evaluation: Applies fuzzy rules (e.g., IF slope is high OR time is slow THEN cost is high) using np.fmax (OR) and np.fmin (AND).
        3. Defuzzification: Returns a single crisp cost value for the segment.

3.5. Visualization

This method handles the creation of the map.

- create_evacuation_map(self, selected_barangay=None):
    - Purpose: Generates the interactive Folium map.
    - Logic:
        1. Creates a folium.Map centered on Poblacion, using the 'Esri Satellite' tile layer.
        2. Adds folium.Marker for all barangays.
        3. If a selected_barangay is provided:
            - Draws all KML paths in self.evacuation_paths as gray folium.PolyLines.
            - Calls self.a_star_path() to find the best route.
            - Draws the corresponding KML path for the best route as a thick, red folium.PolyLine.
    - Returns: A folium.Map object.

4. Application Workflow

Initialization (on first run):

- main() is called. st.session_state.system is None.
- load_embedded_files() runs, finding the data/ files.
- EvacuationSystem is initialized and stored in st.session_state.system.

- During __init__, load_kml_paths and load_all_travel_data are called, populating the system with data.

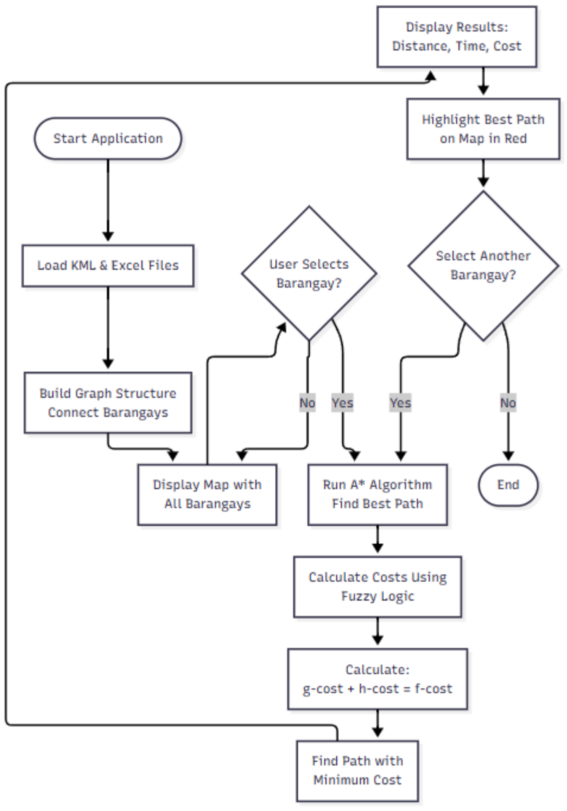
User Interaction (Selecting a Barangay):

- The user selects a barangay (e.g., "Gines") from the st.selectbox.
- Streamlit re-runs the main() script.
- selected_barangay is now "Gines".

Calculation & Rendering:

- main() calls system.a_star_path(system.evacuation_center, selected_barangay) to find the best route to "Gines".
- a_star_path uses get_edge_cost, which in turn uses fuzzy_evaluation, to find the path with the lowest cumulative fuzzy cost.
- main() calls system.create_evacuation_map(selected_barangay).
- create_evacuation_map draws all paths in gray, then draws the best path (Poblacion to Gines) in red.
- The map and the metrics (distance, time) from the a_star_path result are displayed to the user using st_folium and st.metric.

5. System Flowchart


Display Results: Distance, Time, Cost
Start Application
Highlight Best Path on Map in Red
Load KML & Excel Files
User Selects Barangay?
Select Another Barangay?
Build Graph Structure Connect Barangays
No
Yes
Yes
No
Display Map with All Barangays
Run A* Algorithm Find Best Path
End
Calculate Costs Using Fuzzy Logic
Calculate: g-cost + h-cost = f-cost
Find Path with Minimum Cost

The system starts by loading necessary data files, including KML files containing evacuation routes and Excel files with travel data such as road slope, travel time, and curvature. Once the files are loaded, the system builds a graph structure that connects barangays (local administrative districts) based on available road paths.

Next, the application displays a map showing all barangays. The user can then select a barangay to evacuate from. If no selection is made, the system will keep displaying the map until a barangay is chosen.

When a barangay is selected, the system runs the A* algorithm to find the best evacuation path from the evacuation center (Poblacion) to the selected barangay. The A* algorithm calculates the path cost using fuzzy logic, which evaluates road conditions based on slope, travel time, and curvature.

The algorithm combines the actual cost so far (g-cost) with an estimated cost to the goal (h-cost), resulting in a total cost (f-cost). It then identifies the path with the minimum total cost.

After finding the optimal path, the system displays detailed results including distance, travel time, and cost. It also highlights the best evacuation path on the map in red to guide users visually.

Finally, the user can choose to select another barangay to simulate a new evacuation route or end the session. This flow ensures interactive, cost-effective, and visually clear evacuation planning based on real geographic and travel data.