ELeNA(Elevation Based Navigation)

520_SweSquad
Ankita Sahoo
Beena Kumari
Gaurav Chandra
Mostafa Kamal

Objective

Elevation-based Navigation(Elena) is a map based application that takes into consideration the route's elevation with the distance as well. We build a navigation software to find the route based on the user's preferred elevation. The route for maximum or minimum elevation is calculated within x% of shortest path, where x is an input from the user. This type of system can be especially helpful for activities like hiking, mountain climbing, or paragliding where the elevation of the terrain can have a significant impact on the journey and route.

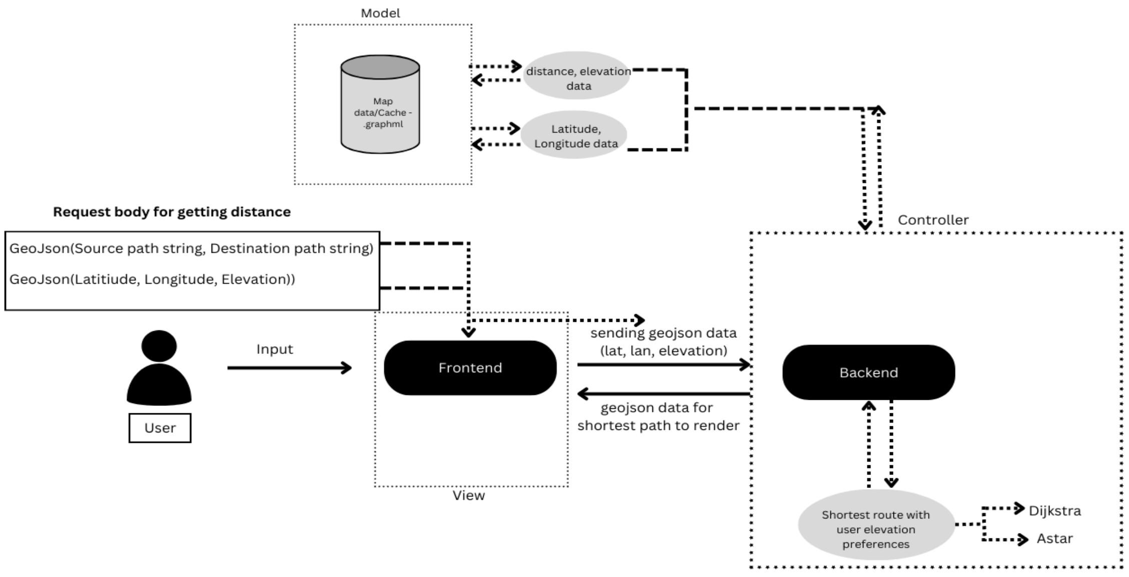
Soft metrics of ELeNA

- Software Requirements Satisfied
    - Readability -> The code repository has a Readme file to understand the steps involved in running the application. The code also uses meaning variable and function names ensuring readability.
    - Testability -> Both frontend and backend code have well defined tests.
    - Modularity and Understandability -> The files are properly packaged within folders and functions are well defined to improve understandibility.
    - Error Handling -> Proper errors are rendered on the screen for different error cases while fetching the results. Try/catch blocks are added for promise calls.

Features

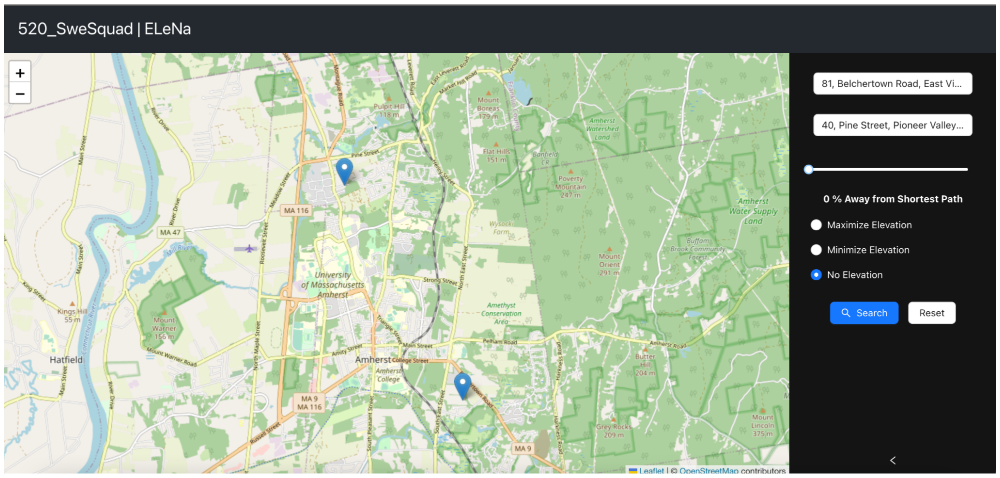
1. The start and destination end location is provided as an input by the user. OpenStreetMapProvider is used to provide support for autocomplete in start and end addresses for user entered locations.
2. Get the shortest distance between source and destination.
3. Get the route with maximum or minimum elevation within x% of shortest distance, x is an input from the user.
4. The resulting path is highlighted on the map.
5. Everytime there's a search performed on the UI screen for a city/town whose data is not saved on our system. We create graphs for the same and save it such that the next calls become seamless, hence implementing a cache system.
6. The UI shows the route statistics, important for comparing the results for different cases.

Design


Model
Map data/Cache - .graphml
distance, elevation data
Latitude, Longitude data
Request body for getting distance
GeoJson(Source path string, Destination path string)
GeoJson(Latitiude, Longitude, Elevation))
Controller
Input
Frontend
sending geojson data (lat, lan, elevation)
geojson data for shortest path to render
Backend
User
View
Shortest route with user elevation preferences
Dijkstra
Astar

Frontend:

- The web platform is built on ReactJs. Maps are rendered using leaflet.js (React-leaflet library) which is basically an encapsulation of the openstreetmap apis. The application also uses ant-design library for building the UI components.
- The web platform renders a map on the screen (zoomed on *Amherst, MA, USA* which is also our sample set for this app) where the user can add start and destination marker by typing the desired locations in the search typeaheads.



81, Belchertown Road, East Vi...

40, Pine Street, Pioneer Valley...

0 % Away from Shortest Path

Maximize Elevation

Minimize Elevation

[✓ No Elevation]

n

- The user also has the provision to select from maximum elevation, minimum elevation and no elevation. We also have a drag button to limit the distance to the x% of shortest path.
- The above inputs are sent to the backend server through rest api calls. The code uses axios library to perform the GET request.

  GET Api URL

  ${backend_url}/metadata?**src=-72.49977940790353,42.37086435**&**dest=-72.52908918181818,42.41020340909091**&**flag=3**&**percent=0**

  src => start address coordinates

  dest => end address coordinates

  flag => 1 (maximize elevation), 2 (minimize elevation), 3 (no elevation)

  percent => integer value for x% of shortest path

Backend:

- On receiving the GET api call, the backend server parses the URL parameters and tries to find the shortest path using the different algorithms. We calculate the required path using the algorithms - A* and Dijkstra.
- The geojson path created from the algorithms is then returned to the frontend for rendering the path on the UI.

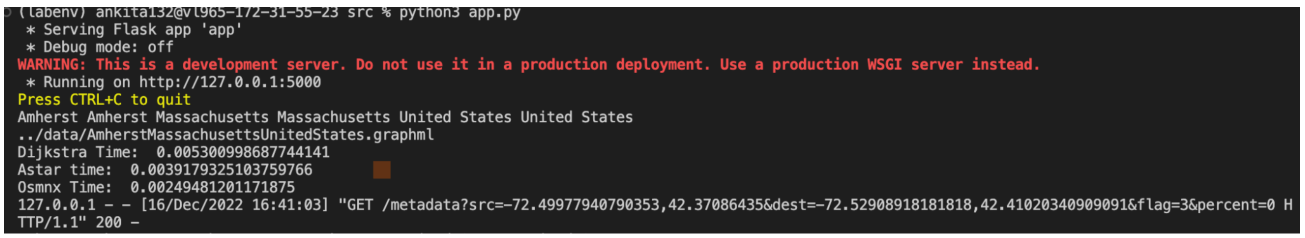
<u>Api returned response</u>

```
{data: {

          distance: ‘6785.999’

          elevation: ‘54.78’

          path: [[42.372559, -72.499927], [42.3729074,
-72.5011874], [42.372965, -72.504849]]}}
```



```
(labenv) ankita132@vl965-172-31-55-23 src % python3 app.py
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
Amherst Amherst Massachusetts Massachusetts United States United States
../data/AmherstMassachusettsUnitedStates.graphml
Dijkstra Time:  0.005300998687744141
Astar time:  0.0039179325103759766
Osmnx Time:  0.00249481201171875
127.0.0.1 - - [16/Dec/2022 16:41:03] "GET /metadata?src=-72.49977940790353,42.37086435&dest=-72.52908918181818,42.41020340909091&flag=3&percent=0 H
TTP/1.1" 200 -
```

We use the OSMnx module, it uses nodes as points on a map that represent specific locations. These nodes are one of the three basic elements of the OpenStreetMap (OSM) data model, along with ways (linear features such as roads) and relations (logical groupings of other elements). Nodes can be used to represent a variety of features, including buildings, traffic lights, and landmarks. When retrieving a street network from OSM using OSMnx, the nodes in the network represent intersections or junctions between the roads. To plot data onto a map using osmnx, we retrieve the street network and extract the nodes.

Testing

1. Unit Testing for Frontend
    - For the UI screen, we created tests to validate the UI components, i.e. Header, Sider and Map components.
    - The axios calls are validated by making mock api calls.

2. Unit Testing for Backend
    - For the backend, we tested the path rendered from different algorithms (Dijkstra, Astar).

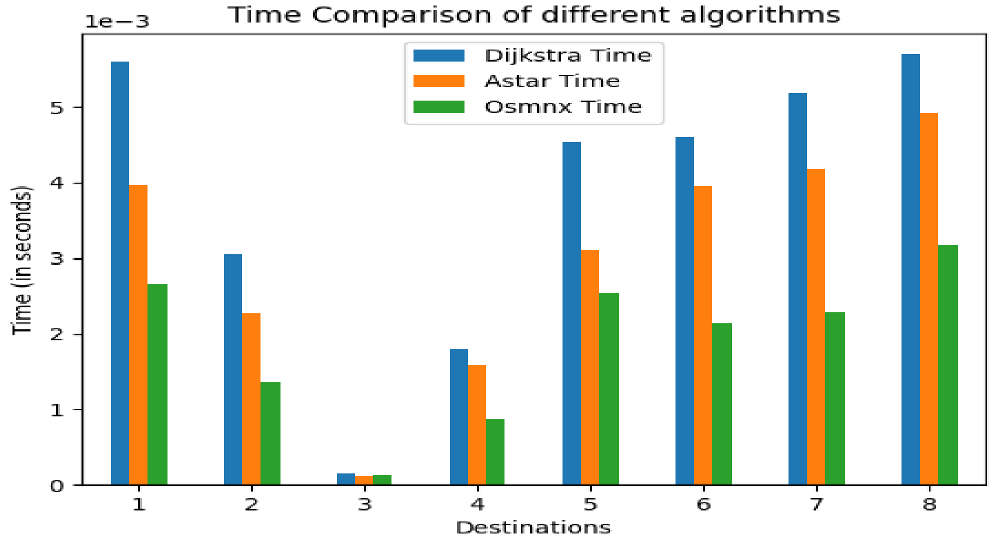
3. Integration Testing
    - The graph and path of certain source and destination points are imitated and tested for correctness from the google maps.
    - Currently we used destination points around Amherst to validate the shortest paths rendered.

4. Beta Testing
    - Beta1.0 and Beta2.0 releases for testing the versions and getting feedback from colleagues on different versions of the application.

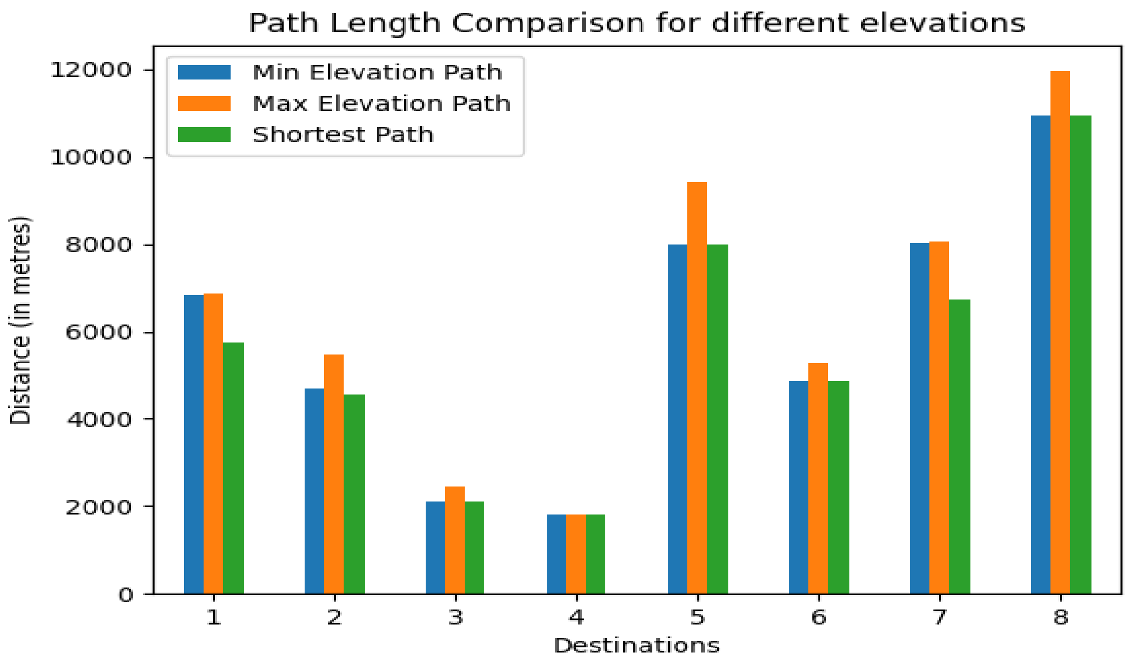
5. Alpha Testing
    - We did thorough testing of the entire platform before deploying, log the debugs and ensure the platform gives the correct results.

Experiments and Results

1. Comparison of different algorithms run times to calculate shortest paths.



2. Path length comparison of different elevation preferences. The minimum and maximum elevation paths are within 20% of the shortest path.


Path Length Comparison for different elevations
Min Elevation Path
Max Elevation Path
Shortest Path
Distance (in metres)
Destinations