Elevation-based navigation system (EleNA)

Team name: SAM and Team Members are

Akhil Reddy Anthireddy - aanthireddy@umass.edu

Sanjana Radhakrishna - sanjanaradha@umass.edu

Maanusri Balasubramanian: mbalasubrama@umass.edu

Introduction

Problem statement

The goal of this project is to build an application that returns the best route between an user-entered source and destination considering user choices and requirements, like maximizing/minimizing the elevation gain in the route, route distance being within x% of the shortest path between the source and destination.

Focus Areas

- Experimented by employing A* and Dijkstra's routing algorithms to find the best path between the source and the destination given the user preferences.
- Developed a sophisticated, intuitive and responsive UI, where the computed routes are rendered properly on the map, making it easily perceivable to the user.

Project Requirements

Functional Requirements:

- The application is able to receive user inputs required for the best route computation process, like source, destination, elevation maximize or minimize requirement.
- A map to render the best path between the source and destination that is computed by the algorithm employed based on the entered user preferences.
- User is given a chance to choose the algorithm to be used in computing the best route, either the A* or the Dijkstra's algorithm

Non-Functional Requirements:

- An intuitive and responsive UI that allows the user to key in their query and displays the best suitable possible in a perceivable/understandable manner.
- Algorithmic performance

Technologies Used

Frontend

- HTML/CSS
- Javascript
- Mapbox GL JS

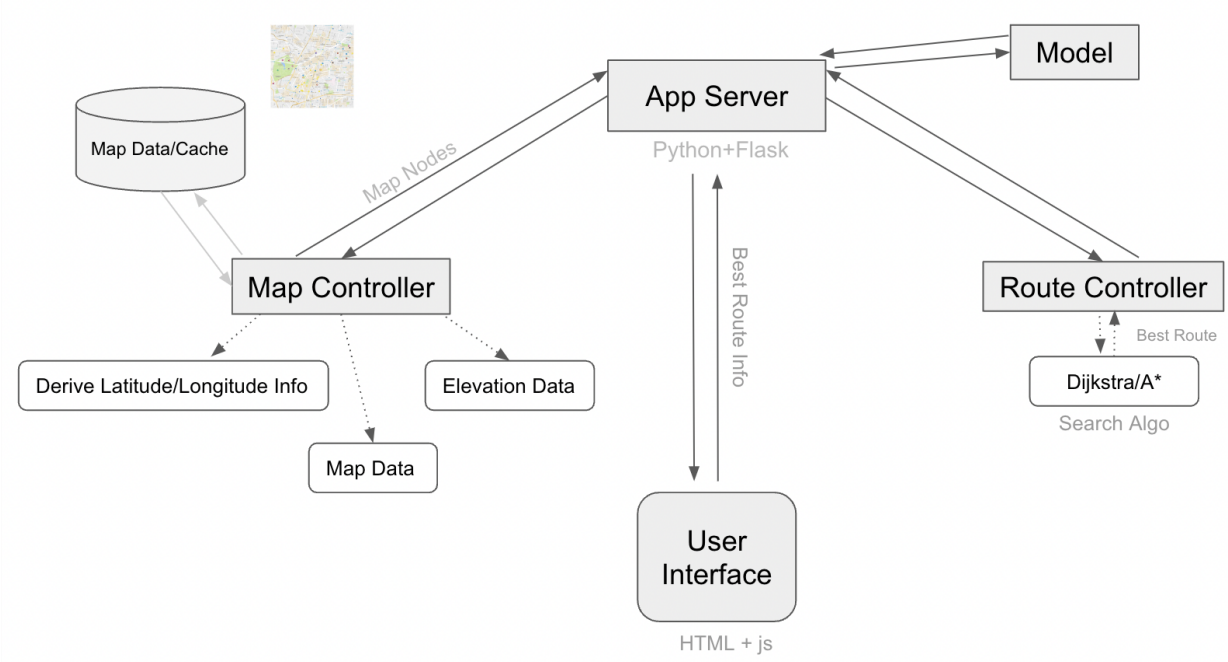
Backend

- Python
- Flask
- Osmnx

Planning:

High Level Architecture:

We are using the Model-View-Controller(MVC) architecture pattern where the model, view, and controller components interact as shown in the below architecture diagram.



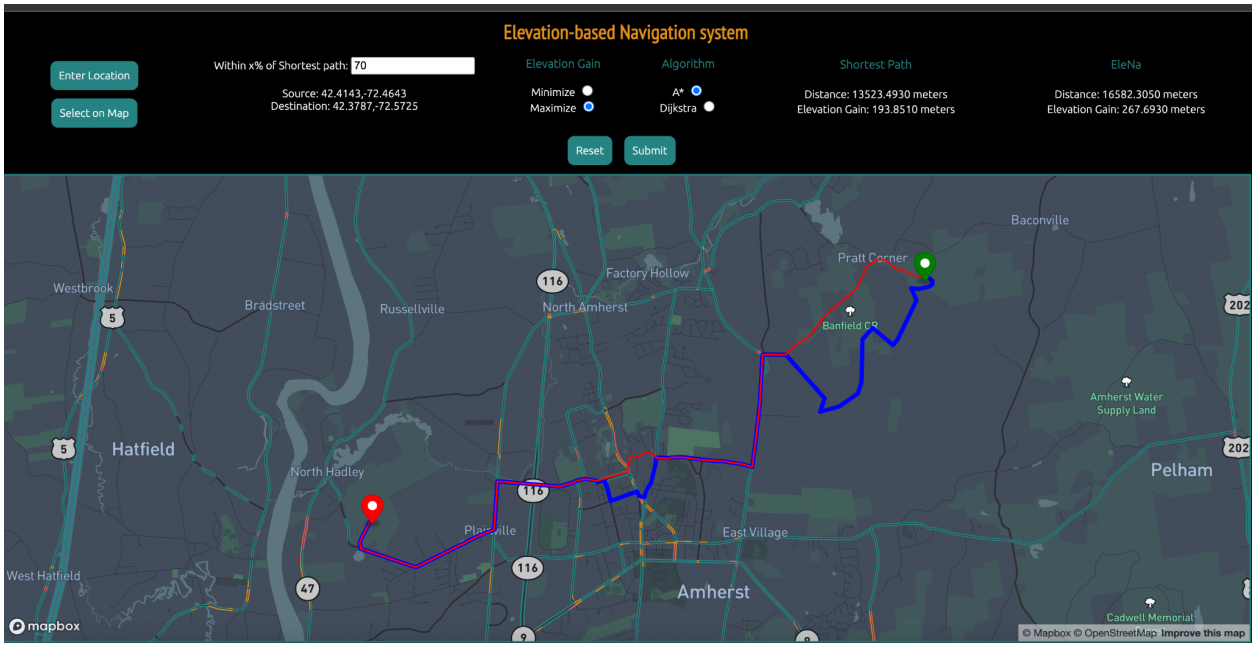
Design patterns used:

1. **Strategy design pattern**: We use two separate algorithms using strategy design patterns. Additionally, we employ numerous heuristics while utilizing a standard template.

2. **Observer design pattern:** View acts as an observer and model acts as observable.

Implementation:

FrontEnd:



Users have 2 options to check the shortest path.

1. **Select on map**: It is used to select source and destination address on the map.
2. **Enter location**: It is used to manually enter the source and destination address.

Total distance between the two locations to x% of the shortest path can be given using **Within X% of Shortest Path**. This actually tells how much extra they are willing to travel in comparison to the shortest non-elevation based path. Users can either select maximum or minimum elevation gain using maximum and minimum toggle buttons. Similarly, users can either select Astar or Dijkstra Algorithm using the Algorithm toggle button. We have used Javascript, html and CSS.

Backend:

For the backend, we are using Flask. It provides a built-in development server and a fast debugger. Scope of our implementation is the Amherst Massachusetts geographical area which can be extended by increasing the radius. We have used the Open Elevation API provided for adding the elevation.
After processing the input json, the backend gives the output json which has the shortest paths with and without elevation gain.

Algorithmic Approach:

For any source and destination coordinates, using OSMNX's function(get_nearest_node), we first fetch the nearest graph nodes to these coordinates.

```
self.source, distance_1 = ox.get_nearest_node(graph, point=start, return_dist=True)
self.destination, distance_2 = ox.get_nearest_node(graph, point=end, return_dist=True)
```

OSMNX's shortest path(shortest_path) function calculates the shortest path between source and destination.

```
self.short_path = nx.shortest_path(graph, source=self.source, target=self.destination,
                                   weight='length')
```

Dijkstra's algorithm allows us to find the shortest path between any two vertices of a graph. The algorithm uses a greedy approach in the sense that we find the next best solution hoping that the end result is the best solution for the whole problem.

A* is just like **Dijkstra**, the only difference is that A* tries to look for a better path by using a heuristic function which gives priority to nodes that are supposed to be better than others while Dijkstra's just explores all possible paths. A* is considered a "best first search" because it greedily chooses which vertex to explore next, according to the value of f(v) [f(v) = h(v) + g(v)] - where h is the heuristic and g is the cost so far.

Testing and Evaluation

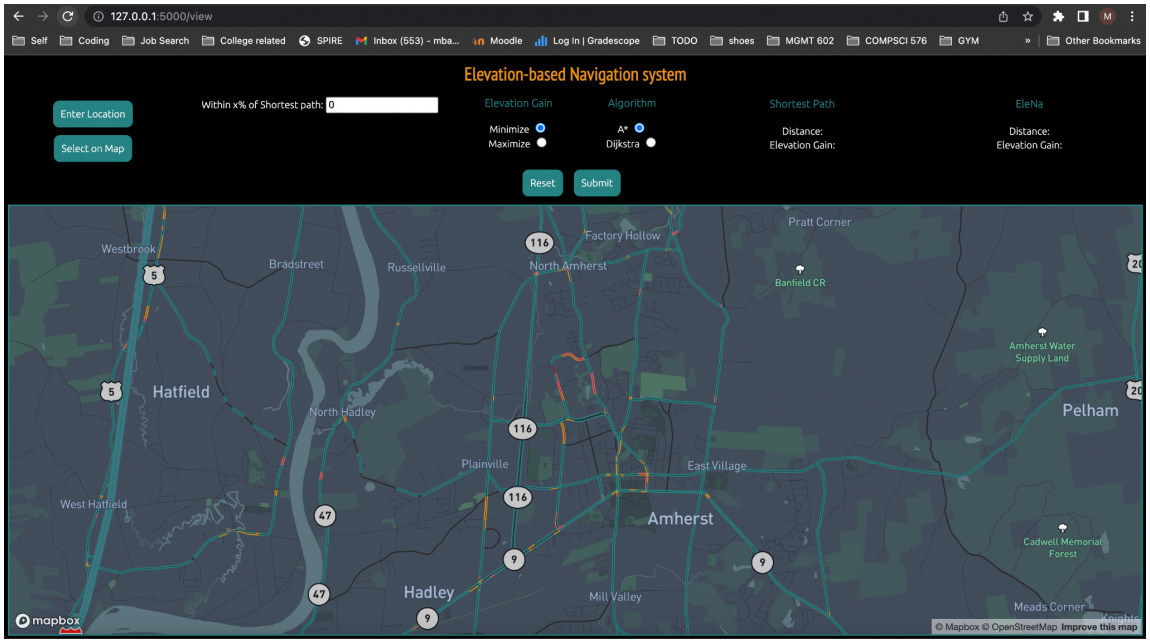
Use of MVC architecture application ensures that application is easily testable.

Backend testing:
We have used the python unittest library for backend testing.

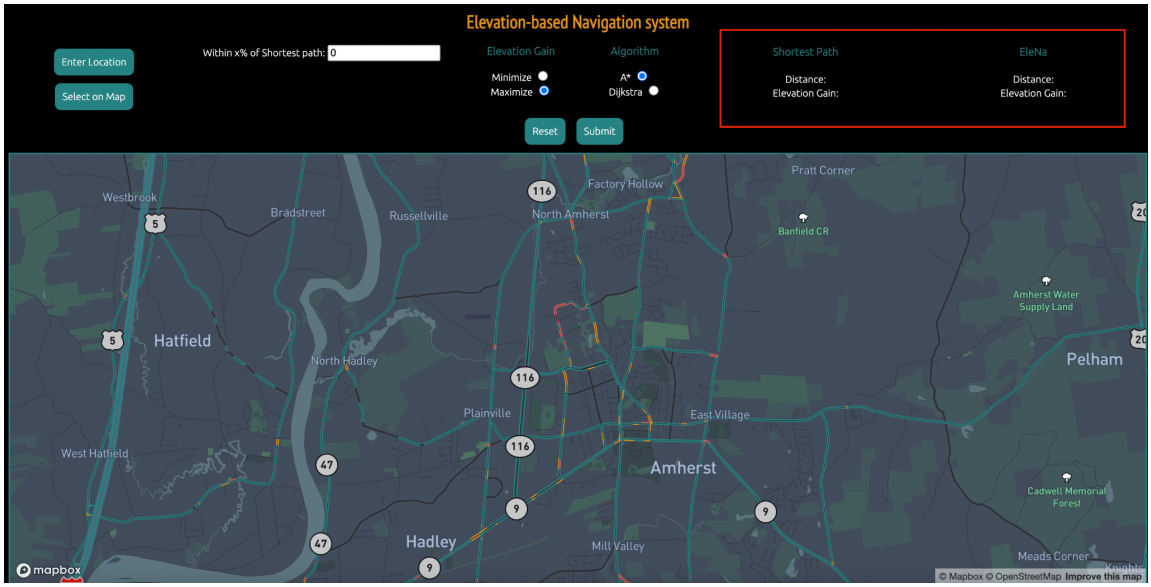
Frontend testing:

The following use cases were manually verified and screenshots have been attached wherever possible.

1. On page load, the fields to show the source and destination values are not visible

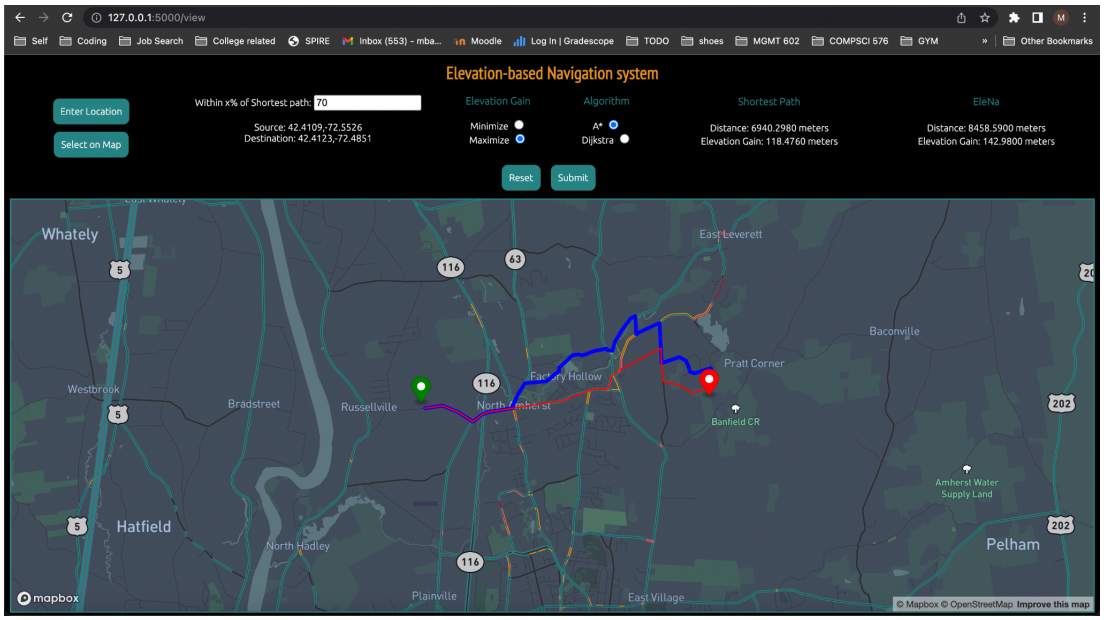


2. On page load, UI fields that display the Distance and Elevation Gain under Shortest Path and EleNa are empty

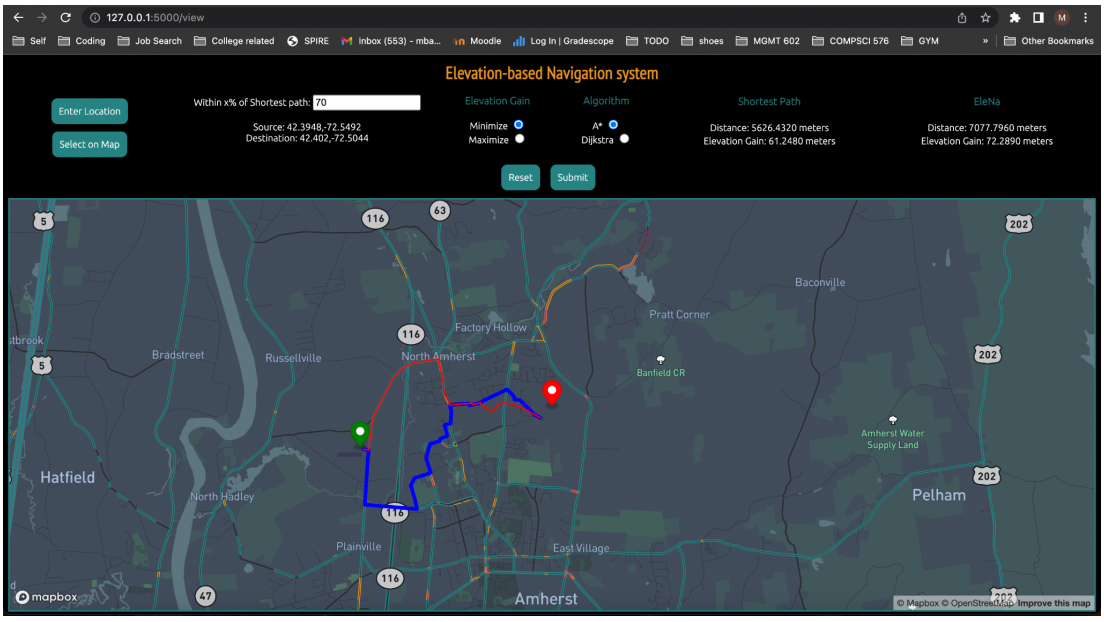


3. Able to toggle between the Minimize and Maximize options under Elevation Gain

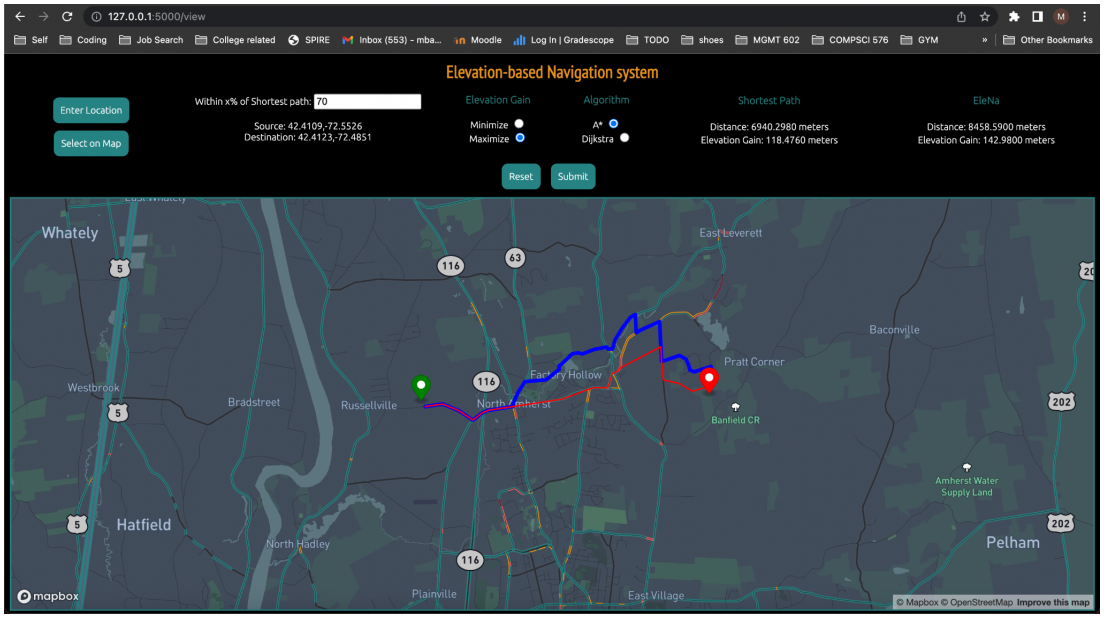
Maximize

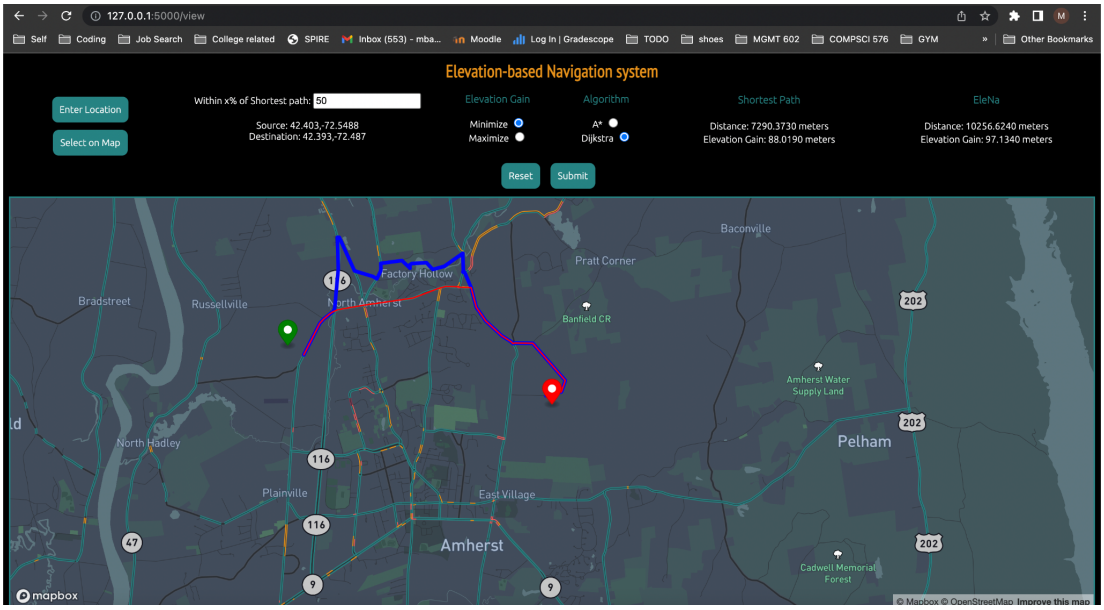


Minimize

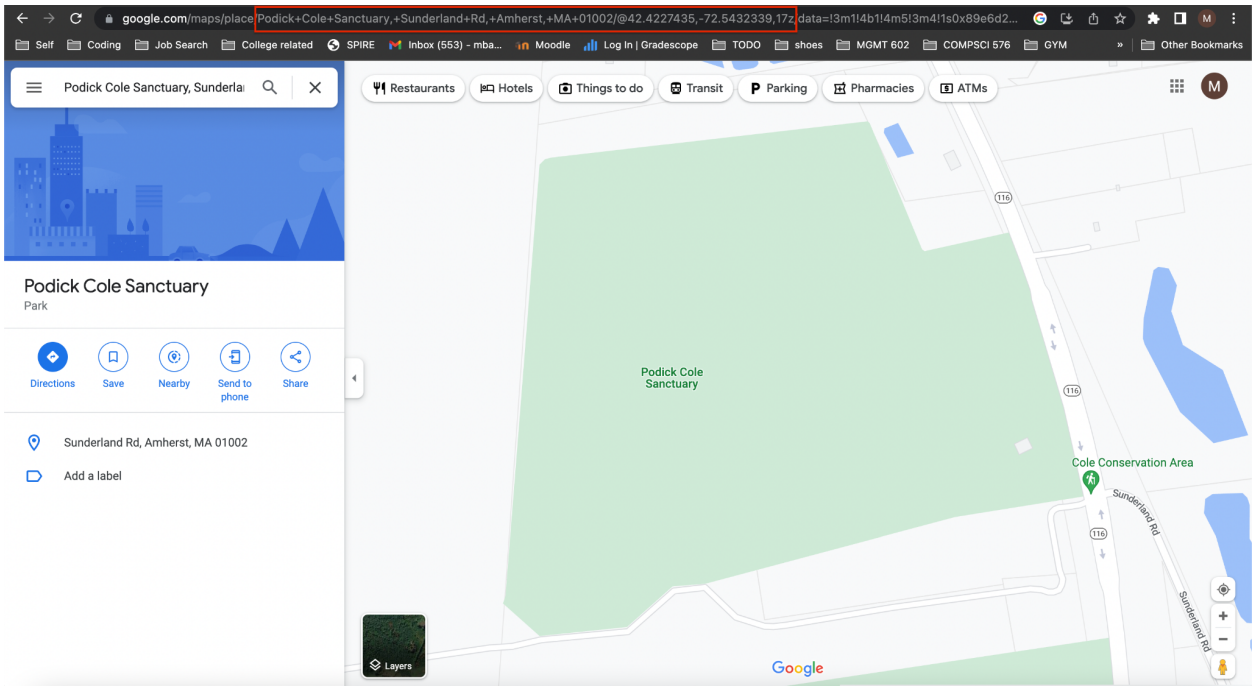


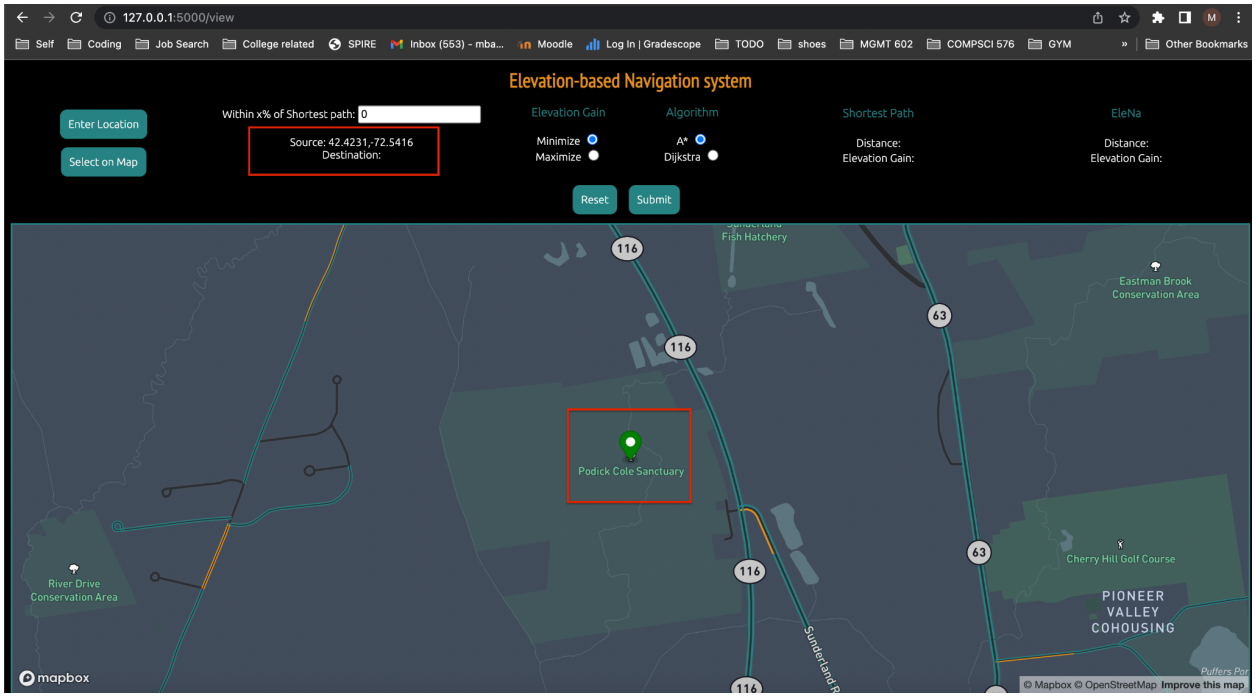
4. Able to toggle between A* and Dijkstra options under Algorithms



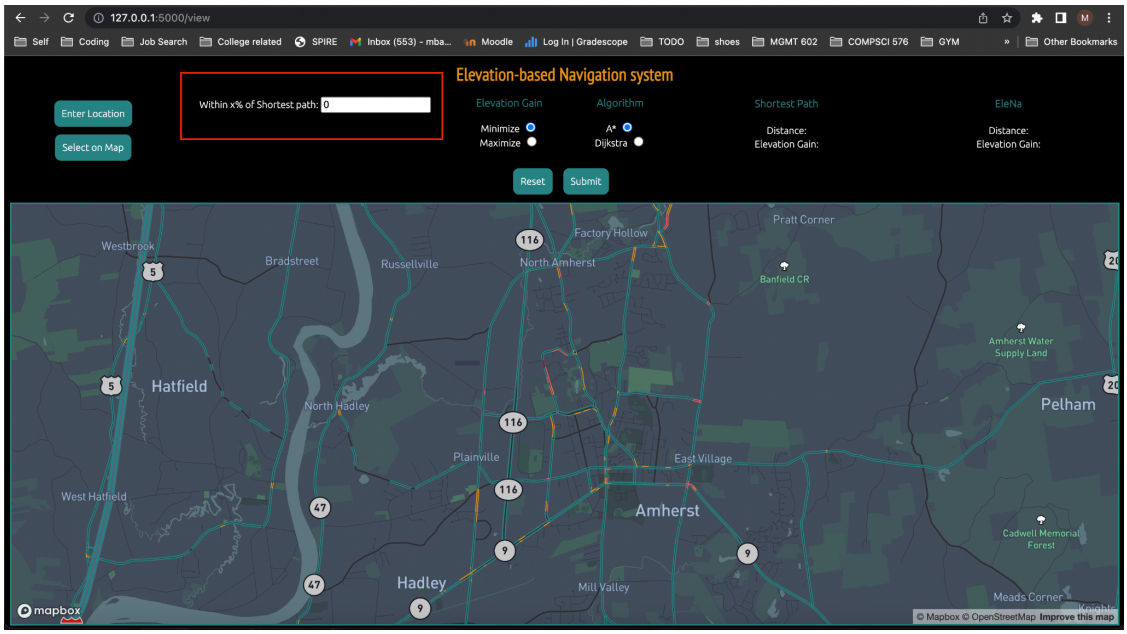


5. Correct coordinates are printed when marker is placed on the map (Compared with the results from Google maps)

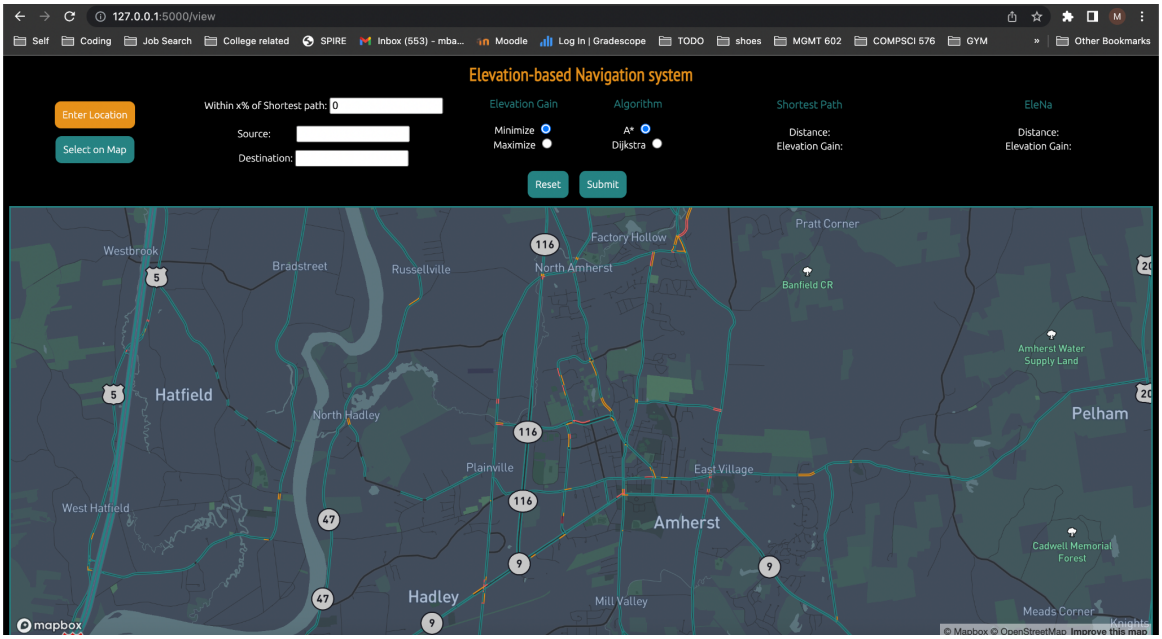




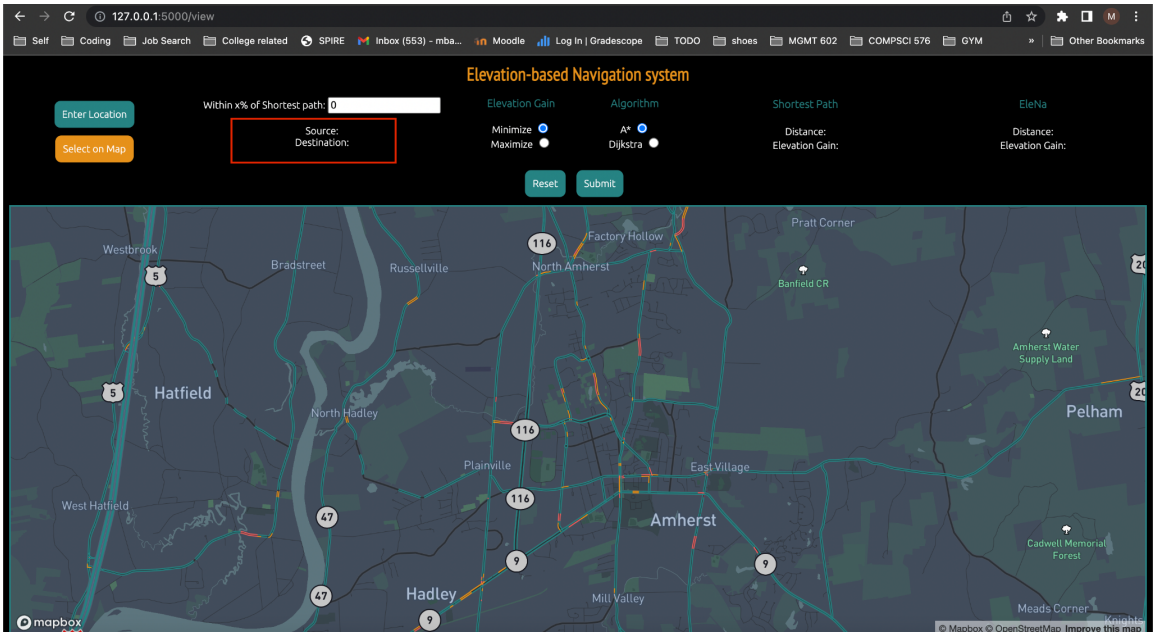
6. On page load, the input field corresponding to within x% of Shortest path is set to 0



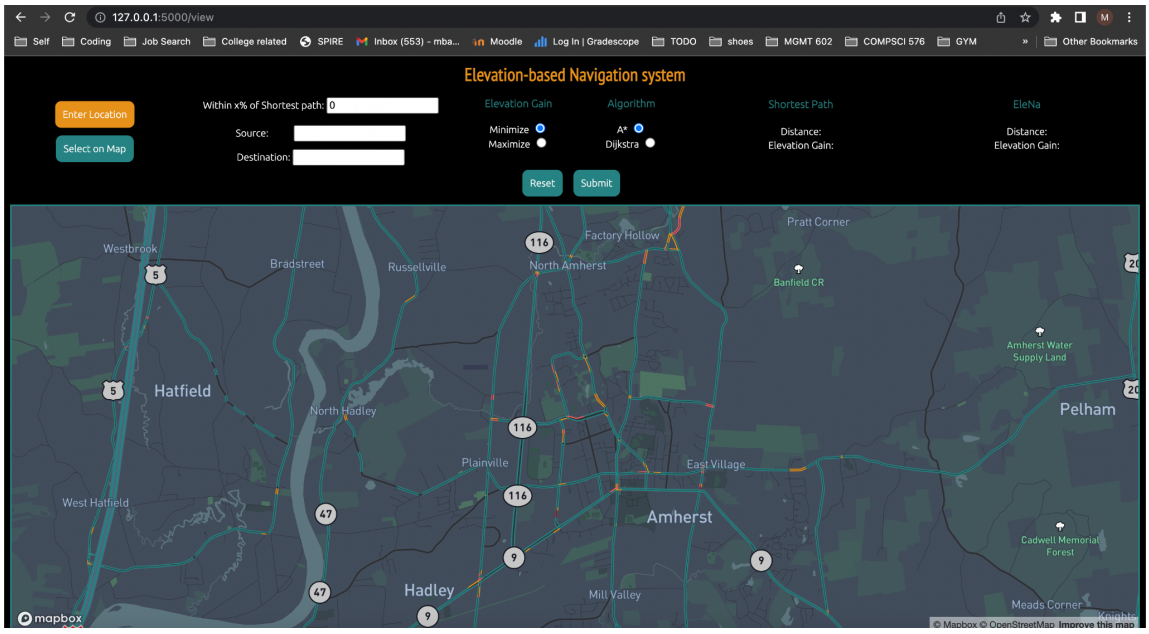
7. After page load, clicking “Enter Location” displays the input fields for entering the source and destination addresses



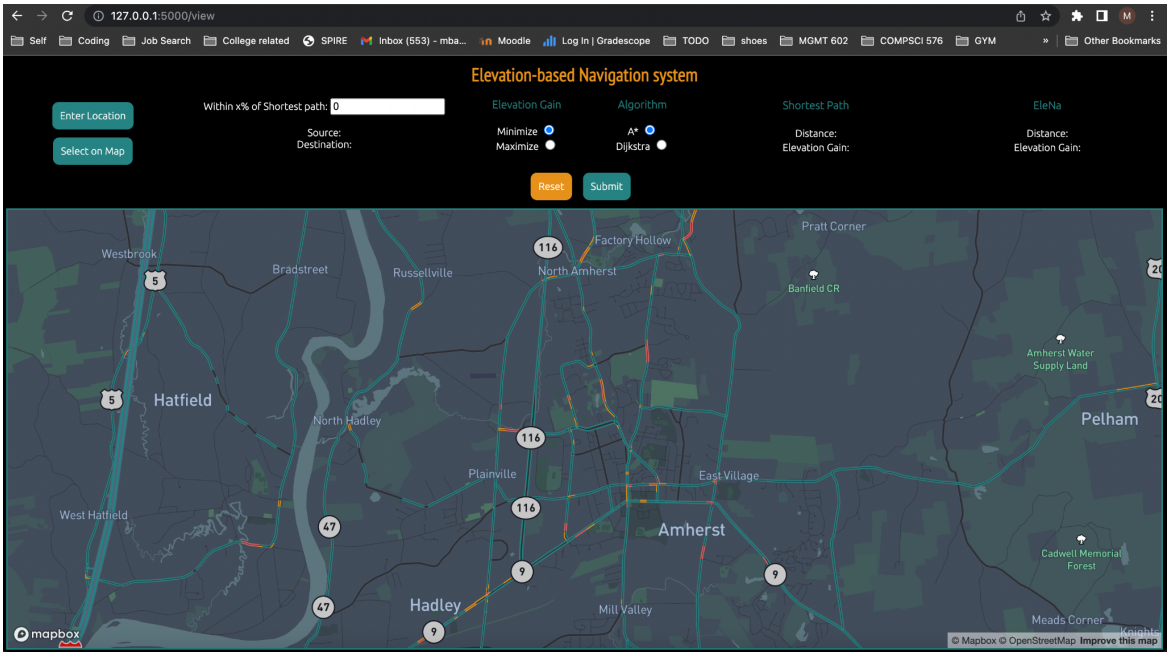
8. After page load, clicking “Select on Map”, displays a UI element that displays the latitude and longitude of the about to be selected location appears



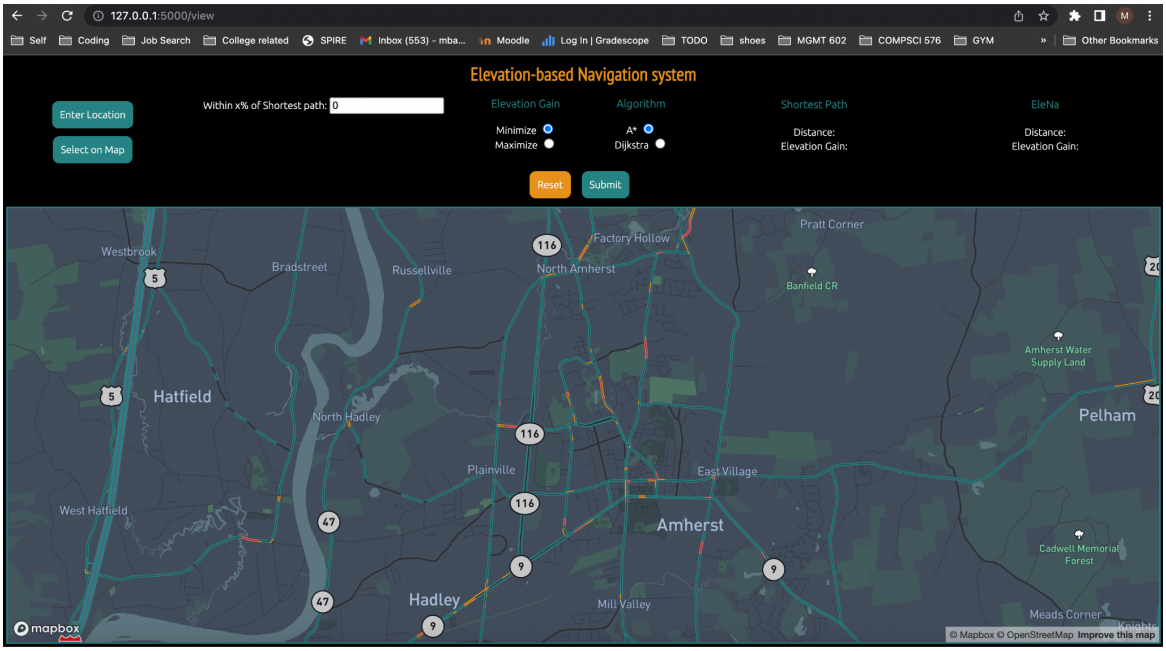
9. When “Enter Location” in chosen and then the user chooses “Select on Map”, the input fields for entering the source & destination addresses disappear and a UI element that displays the latitude and longitude of the about to be selected location appears



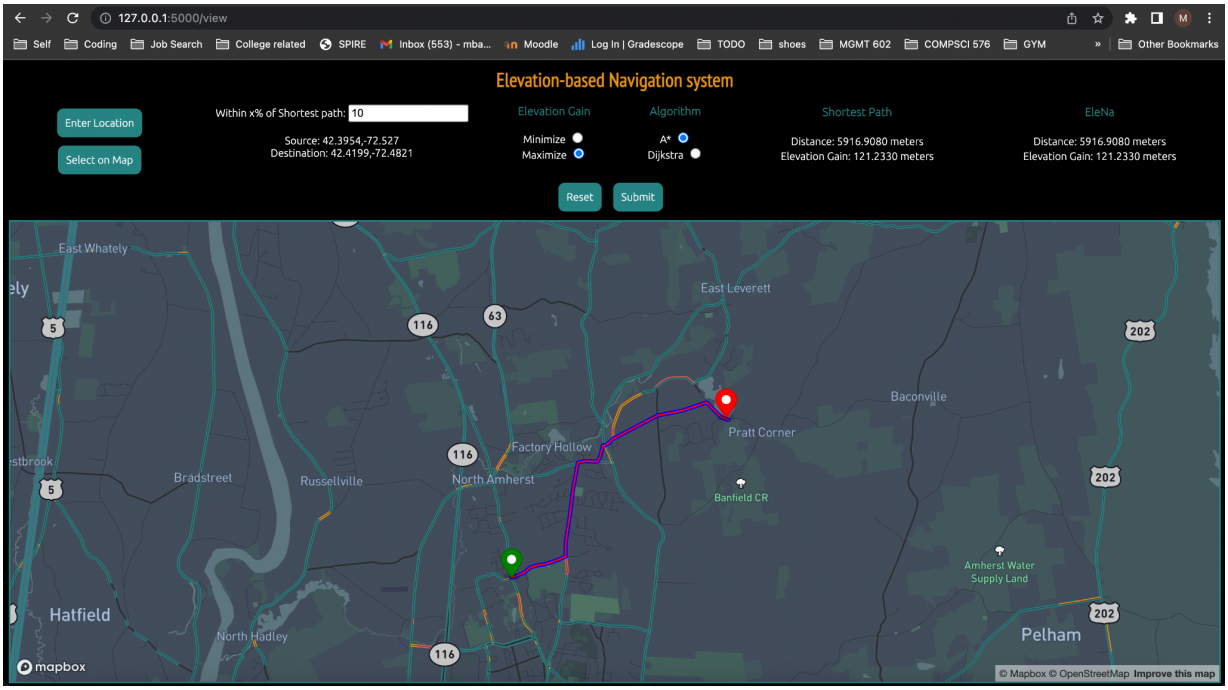
10. Hovering on any of the 4 buttons (Enter Location, Select on Map, Reset/Submit) changes the color of the button

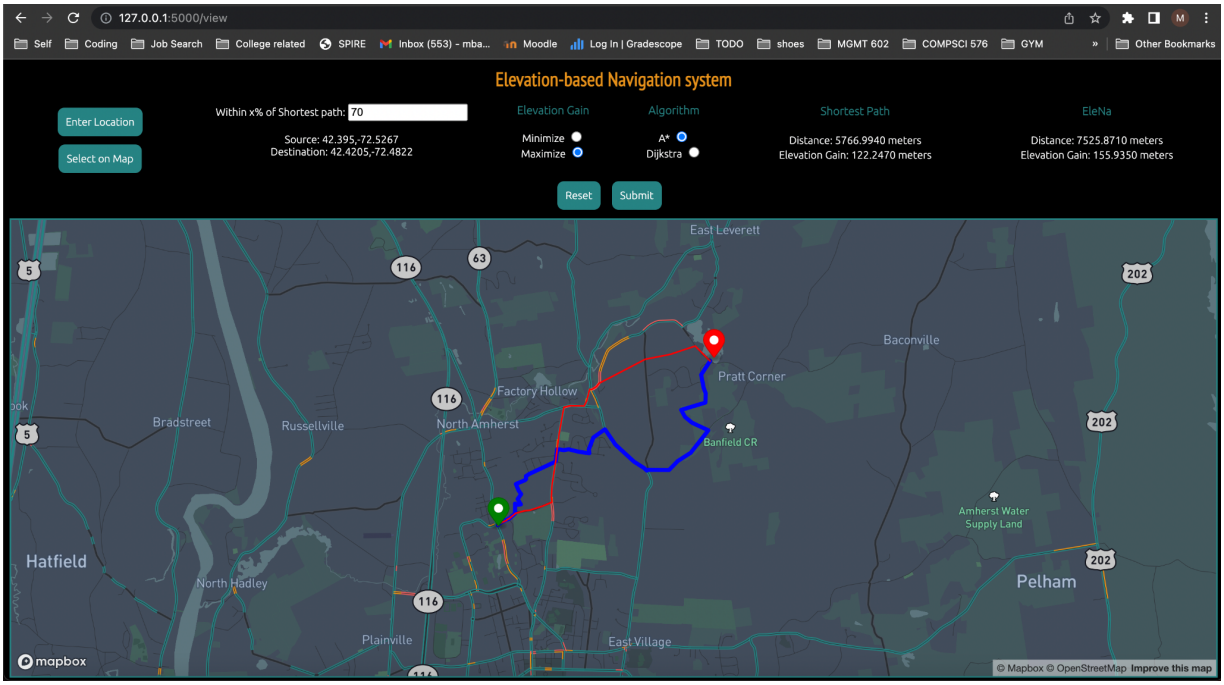


11. Clicking on the “Reset” button, removes the source and destination latitude/longitude input fields and information, sets value in the input field corresponding to x% of shortest path to 0 and clears the markers on the map

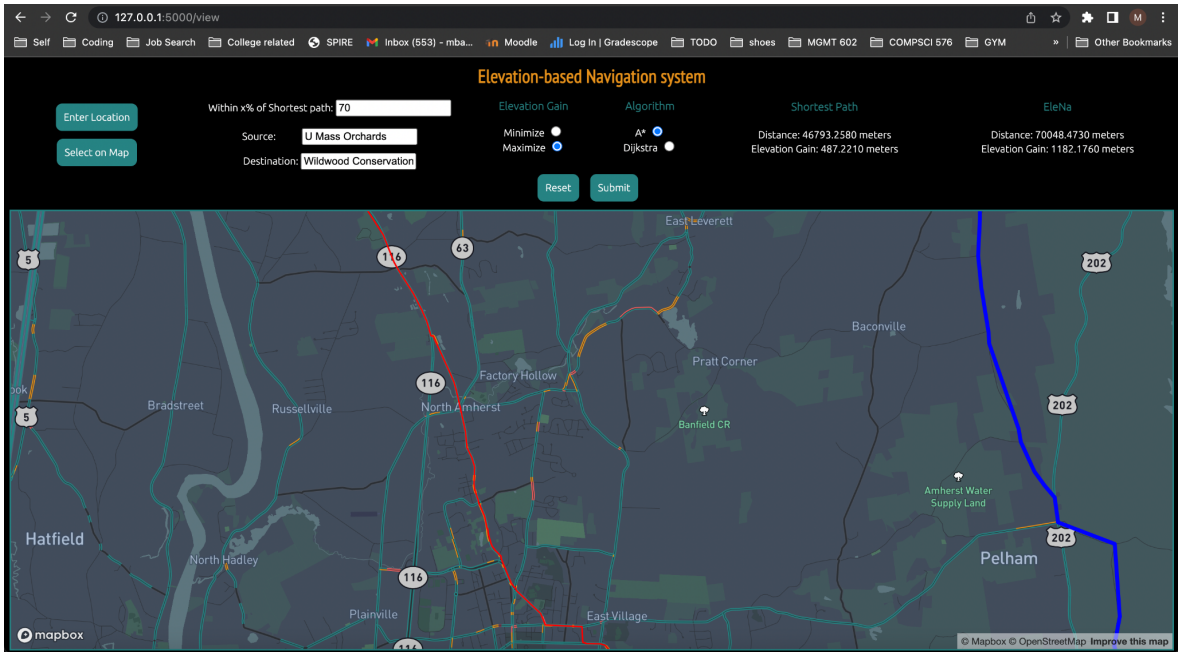


12. Entering different values in the input field corresponding to x% of shortest path, gives different values when all other settings remain the same

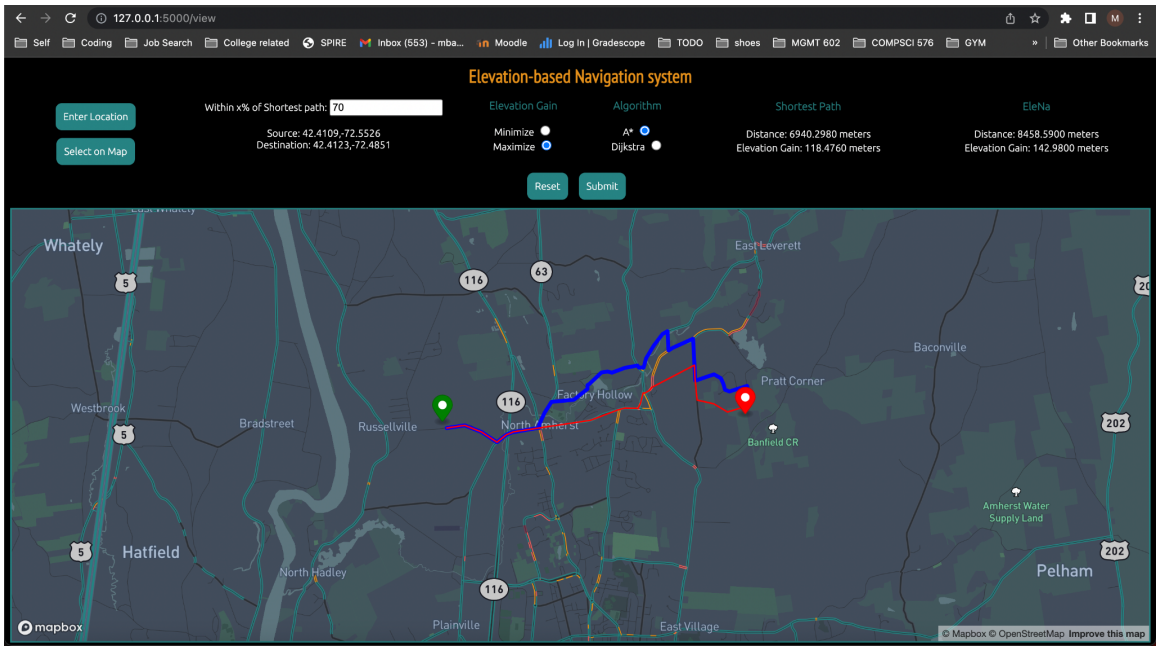




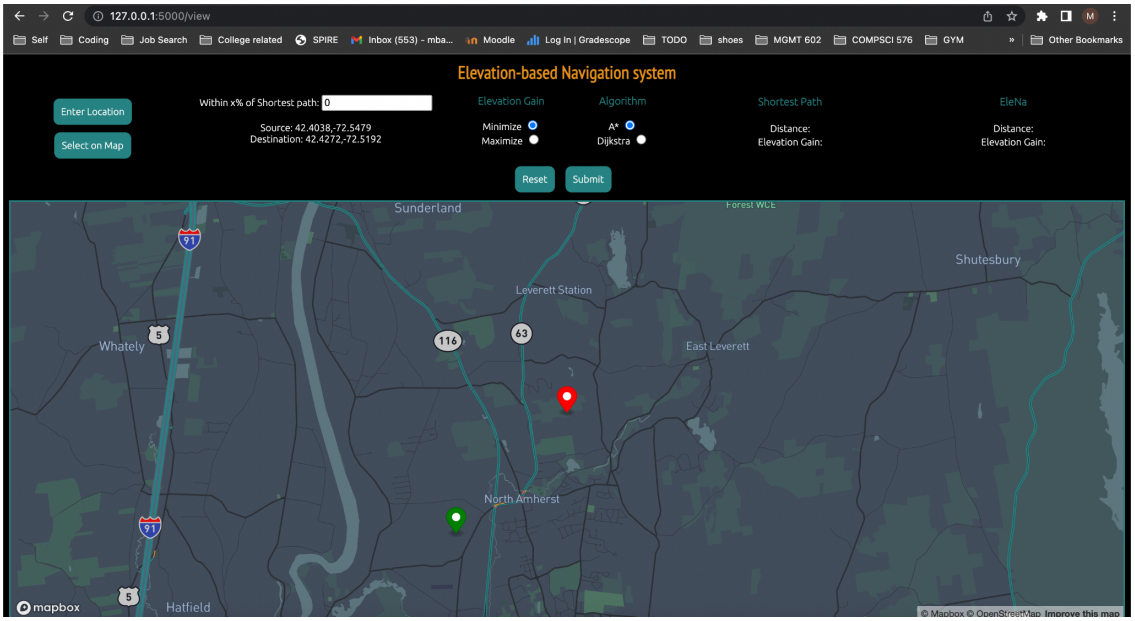
13. Manually entering the source and destination and all information and clicking on “Submit”, displays the Distance and Elevation Gain under Shortest Path and EleNa and the corresponding paths in the map connecting the source and destination



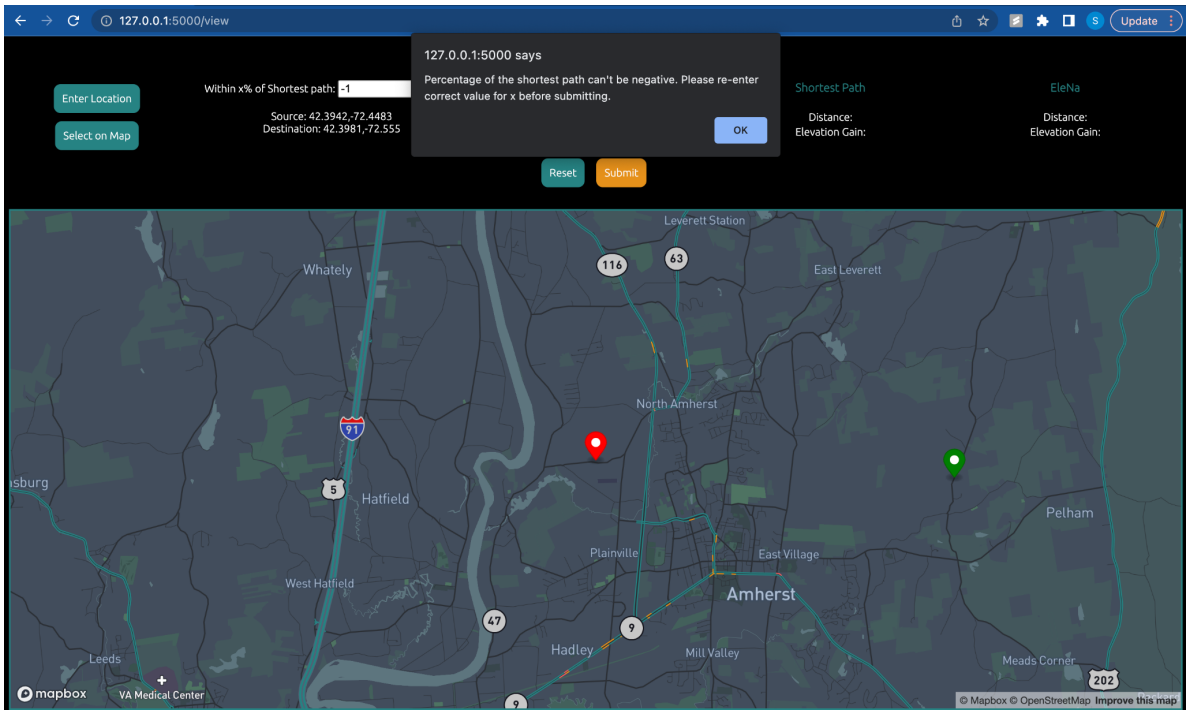
14. Selecting source and destination on the map and clicking on “Submit”, displays the Distance and Elevation Gain under Shortest Path and EleNa and a the corresponding paths in the map connecting the source and destination



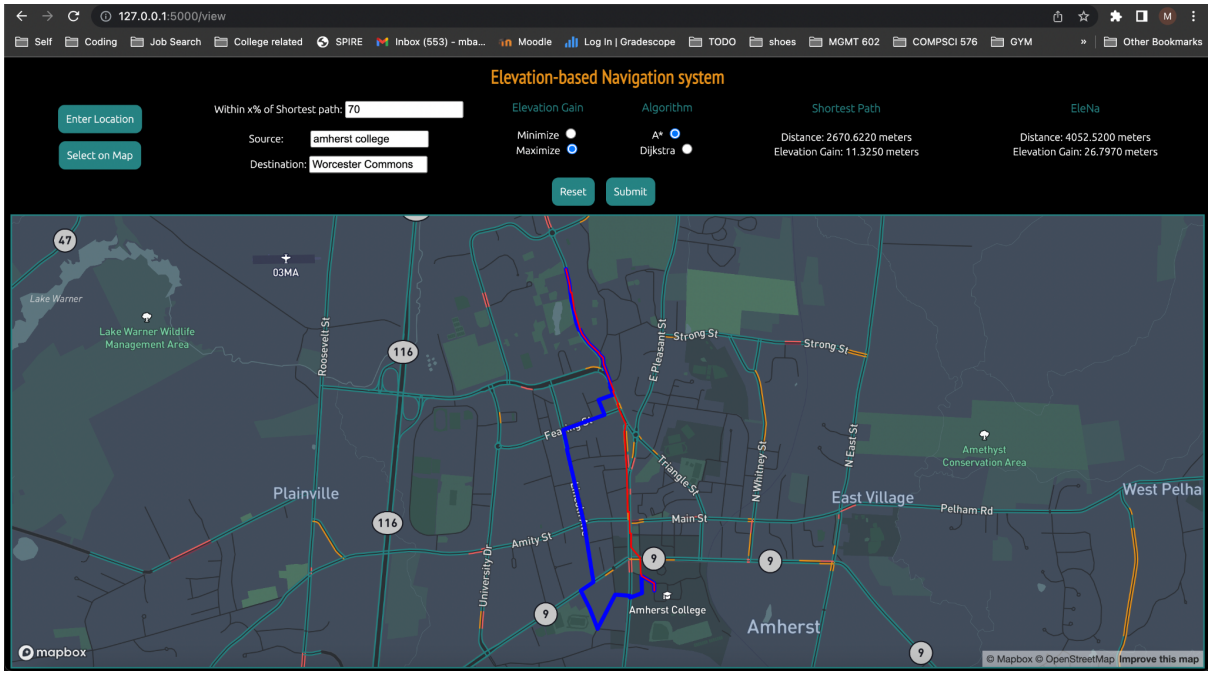
15. On choosing “Select on map” and a click on the map puts pointers marking the location selected

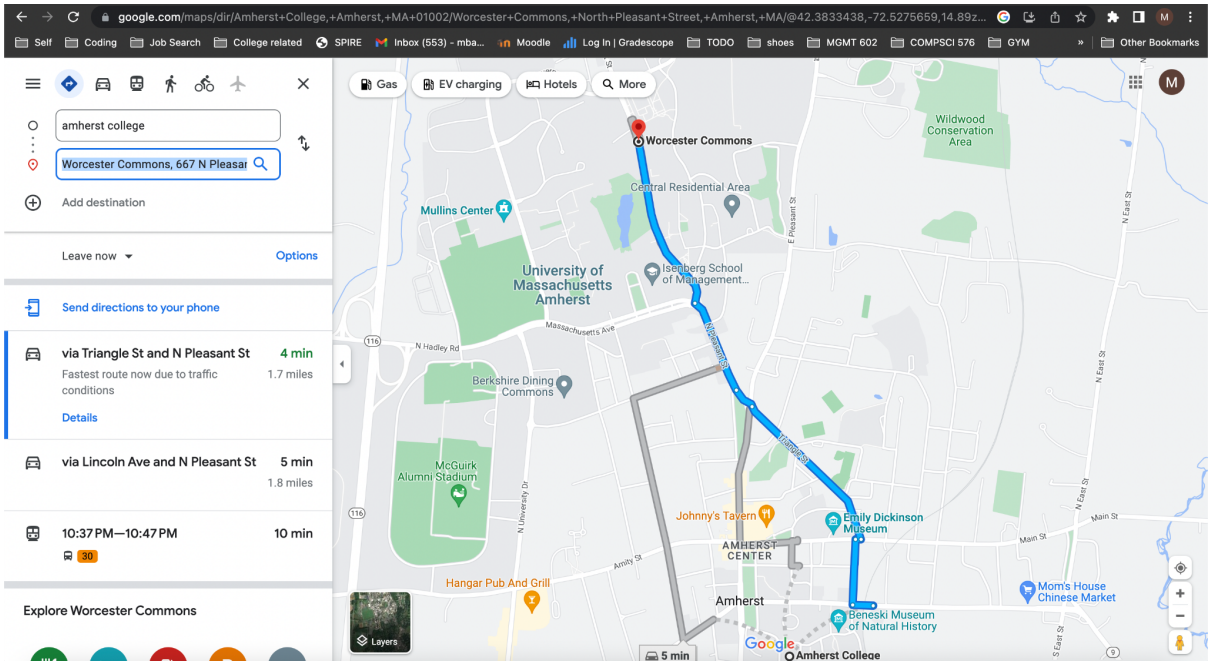


16. An error popup appears when either Source address/Destination address/both have invalid entries/are left blank by the user and “Submit” is pressed



17. The shortest path returned by this application matches with the path returned by Google Maps





18. Selecting “Enter Location” doesn’t allow user to click on any points in the map

19. When using “Select on map”, the user is allowed to select a maximum of 2 points only (that are marked by the pointers)

Performance testing:

We compute the execution times that the application takes to return the best route based on the user preferences and the shortest path. We make the request to the algorithm 30 times, each in the case of placing the pointers and manually entering the address and recording the execution time for this.

Mean Time for the path computations when:
Pointers were placed on the map: 1.445 seconds
Address is manually entered by the user: 2.041 seconds

Usability:

a. Option to select the location the map
b. Reset option to reset the values
c. Option to enter source and destination manually
d. Visualization of shortest paths with and without elevation gain.
e. Option to maximize and minimize the elevation gain.

How our EleNa application stand out:

Our application helps the user visualize both the shortest path and the best path given the elevation preference of the user, giving them a better understanding of their options (Both the routes are highlighted on the map)

Extensibility:

a. It is simple to add extra algorithms to the implementation as the Strategy design pattern is used.
b. The solution is made efficient for concurrent access by numerous users by the usage of the Observer design pattern.

Appendix:

Github repo link:
https://github.com/akhil-2709/EleNA-sam-520

Youtube video link is available in the README file:
https://github.com/akhil-2709/EleNA-sam-520#readme