Elevation-based navigation system (EleNA)

SDE CORPS

Github Repository - https://github.com/shrey96goyal/520-SWECorps

1. SOFTWARE REQUIREMENT SPECIFICATIONS (SRS)

Overview

Idea

The project is about implementing a navigation system that considers the elevation gain. A typical system like Apple or Google Maps just takes the start and end points from the user and presents the shortest distance between them using standard algorithms such as Dijkstra's or A*. The goal of this project is to develop a software system that determines a route to minimize/maximize the elevation gain depending on the start and end locations of the user. The elevation gain increment/decrement can change the distance to up to x% of the original shortest route.

Stakeholders

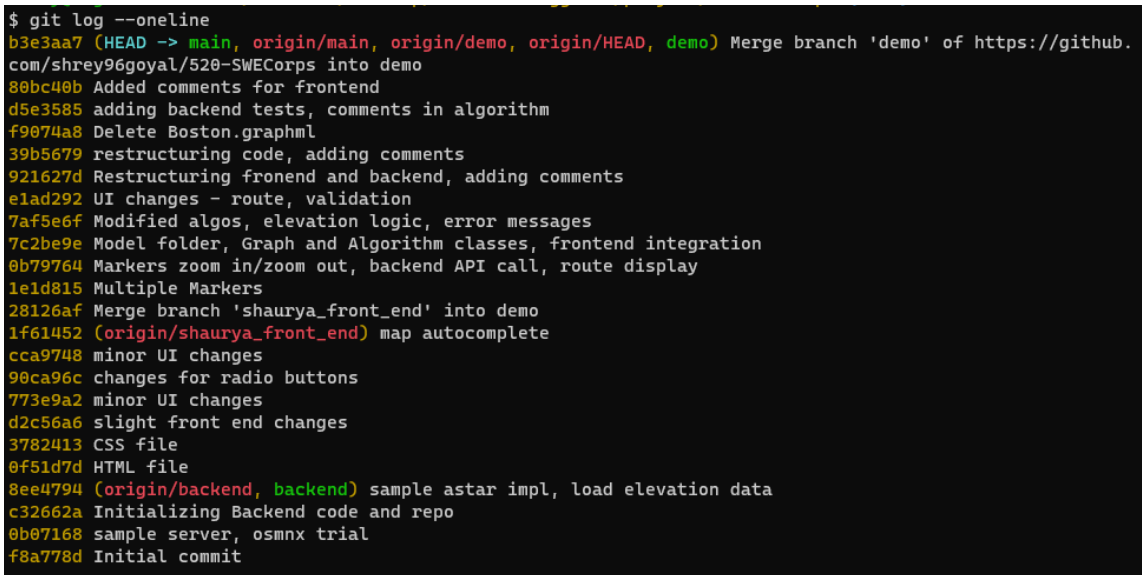
The stakeholders for this software system may include the following groups-

1) **Users -** The users will be using the app to find their way from point A to B at the desired elevation. The user has to input the start and end locations and they get the shortest route which can change depending on the change in elevation gain.
2) **Developers -** This group is responsible for planning, designing, and developing the project. It should meet the needs of the users and all functions should work properly.
3) **Managers -** This group guides the developers in their work. They supervise and manage the development cycle. They ensure that the project is completed on time, under financial constraints, and meets the needs of the users.
4) **Quality Assurance/Testing team -** This group tests the application to ensure it meets the standards of quality. They identify errors and bugs which are reported to the developers.
5) **Marketers -** The team promotes the app among potential users. Various ways to do this include social media, TV/radio, and other promotional activities.
6) **Investors -** This group provides funding for the activities of other groups in the stakeholders in exchange for a return on their investment as profits.

Non-functional Requirements

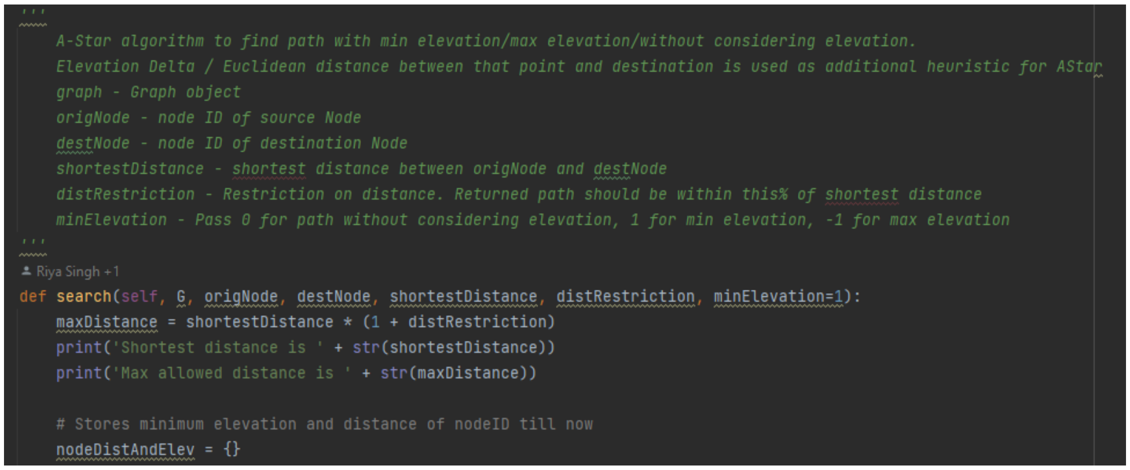
The following non-functional requirements are satisfied in this project -

1) **Version Control -** Git is used for this project for version control. Each change in the codebase is committed on GitHub with an appropriate message. This also allows multiple developers to work on the same codebase.



```
$ git log --oneline
b3e3aa7 (HEAD -> main, origin/main, origin/demo, origin/HEAD, demo) Merge branch 'demo' of https://github.
com/shrey96goyal/520-SWECorps into demo
80bc40b Added comments for frontend
d5e3585 adding backend tests, comments in algorithm
f9074a8 Delete Boston.graphml
39b5679 restructuring code, adding comments
921627d Restructuring fronend and backend, adding comments
e1ad292 UI changes - route, validation
7af5e6f Modified algos, elevation logic, error messages
7c2be9e Model folder, Graph and Algorithm classes, frontend integration
0b79764 Markers zoom in/zoom out, backend API call, route display
1e1d815 Multiple Markers
28126af Merge branch 'shaurya_front_end' into demo
1f61452 (origin/shaurya_front_end) map autocomplete
cca9748 minor UI changes
90ca96c changes for radio buttons
773e9a2 minor UI changes
d2c56a6 slight front end changes
3782413 CSS file
0f51d7d HTML file
8ee4794 (origin/backend, backend) sample astar impl, load elevation data
c32662a Initializing Backend code and repo
0b07168 sample server, osmnx trial
f8a778d Initial commit
```

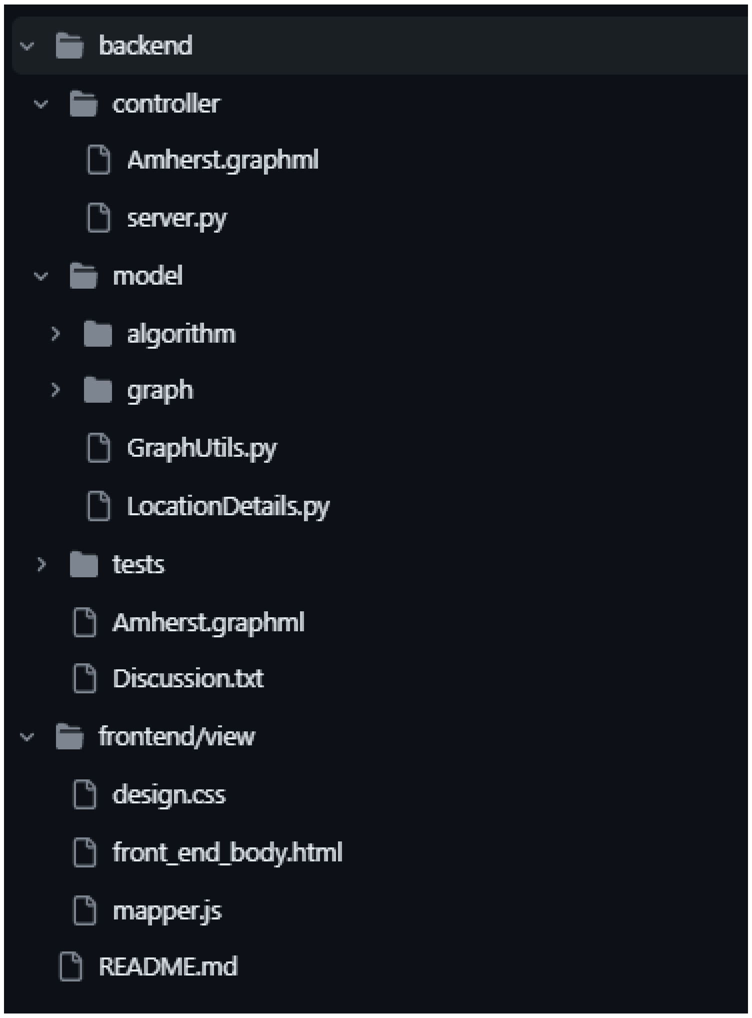
2) **Understandability -** The project implements understandability as follows
   a) **Comments -** Comments have been added for each method to explain their functionality and their parameters. Inline comments have also been added as necessary.



```
'''
    A-Star algorithm to find path with min elevation/max elevation/without considering elevation.
    Elevation Delta / Euclidean distance between that point and destination is used as additional heuristic for AStar
    graph - Graph object
    origNode - node ID of source Node
    destNode - node ID of destination Node
    shortestDistance - shortest distance between origNode and destNode
    distRestriction - Restriction on distance. Returned path should be within this% of shortest distance
    minElevation - Pass 0 for path without considering elevation, 1 for min elevation, -1 for max elevation
'''
Riya Singh +1
def search(self, G, origNode, destNode, shortestDistance, distRestriction, minElevation=1):
    maxDistance = shortestDistance * (1 + distRestriction)
    print('Shortest distance is ' + str(shortestDistance))
    print('Max allowed distance is ' + str(maxDistance))

    # Stores minimum elevation and distance of nodeID till now
    nodeDistAndElev = {}
```

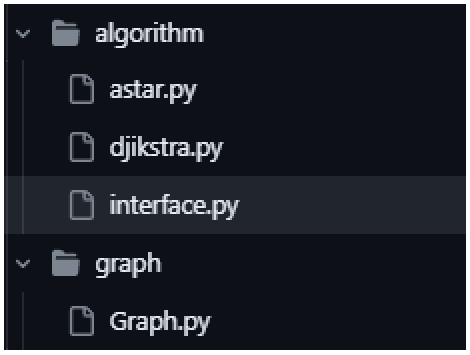
b) **README** - A README file has been provided which outlines the functionality provided in the project, and provides instructions on how to run the application

3) **Modularity -** The codebase is modular. There is a front-end and a back-end, and the application follows the MVC architecture. The codebase is further split into modules for each component, each with its focussed functionality like algorithm and graph classes in the model.



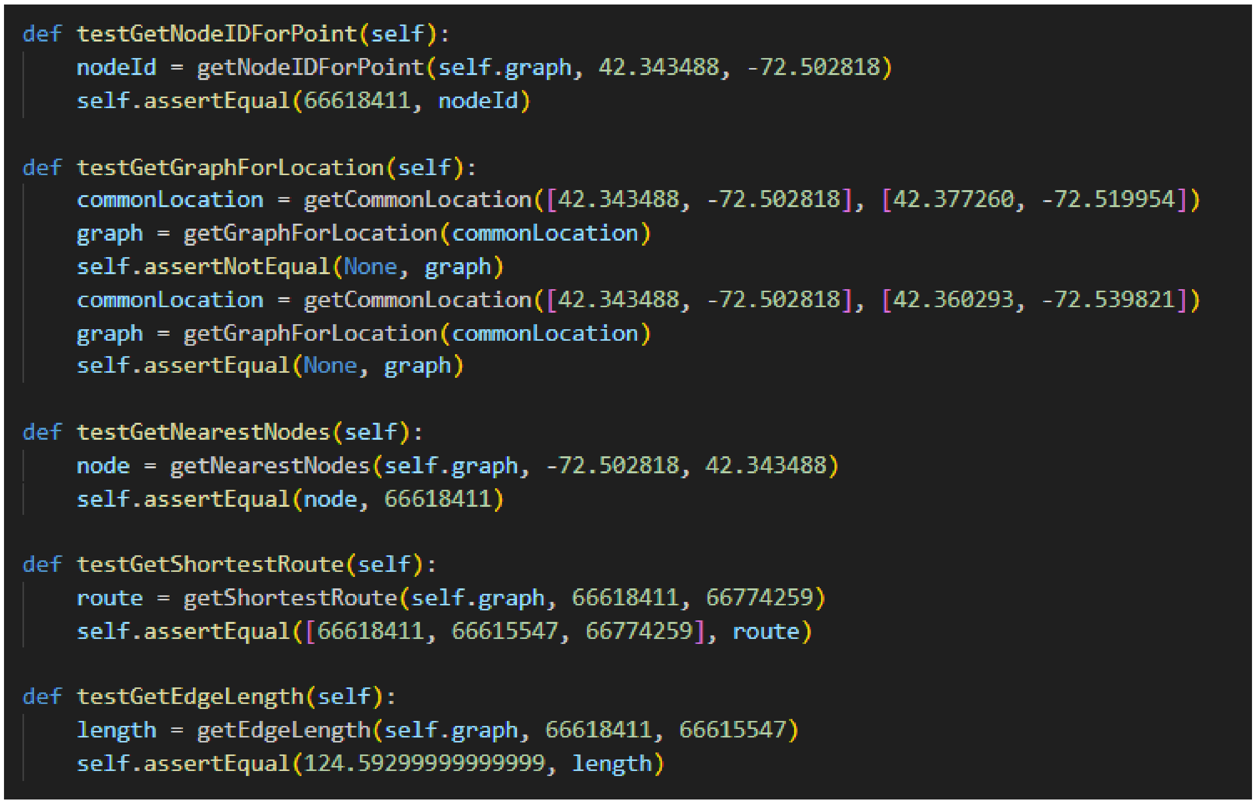
```
backend
  controller
    Amherst.graphml
    server.py
  model
    algorithm
    graph
    GraphUtils.py
    LocationDetails.py
  tests
  Amherst.graphml
  Discussion.txt
frontend/view
  design.css
  front_end_body.html
  mapper.js
README.md
```

Front-end, Backend and MVC



Modules for algorithm and graph

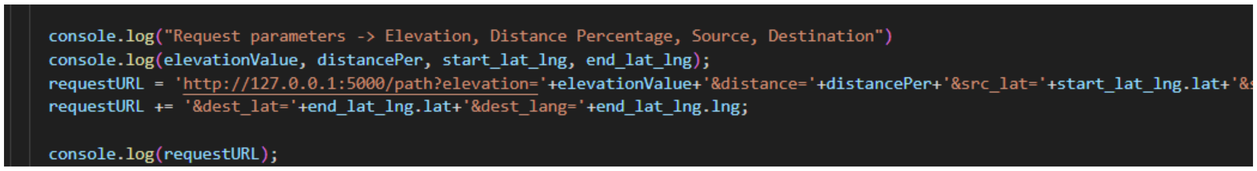
4) **Testability -** We have written unit tests for the backend (in *backend/tests/test_unit.py)*. We have also written integration tests where we call the API and check the response for validity (*backend/tests/test_API.py)*. We did beta testing with the help of our peers for the user interface and user experience



```
def testGetNodeIDForPoint(self):
    nodeId = getNodeIDForPoint(self.graph, 42.343488, -72.502818)
    self.assertEqual(66618411, nodeId)

def testGetGraphForLocation(self):
    commonLocation = getCommonLocation([42.343488, -72.502818], [42.377260, -72.519954])
    graph = getGraphForLocation(commonLocation)
    self.assertNotEqual(None, graph)
    commonLocation = getCommonLocation([42.343488, -72.502818], [42.360293, -72.539821])
    graph = getGraphForLocation(commonLocation)
    self.assertEqual(None, graph)

def testGetNearestNodes(self):
    node = getNearestNodes(self.graph, -72.502818, 42.343488)
    self.assertEqual(node, 66618411)

def testGetShortestRoute(self):
    route = getShortestRoute(self.graph, 66618411, 66774259)
    self.assertEqual([66618411, 66615547, 66774259], route)

def testGetEdgeLength(self):
    length = getEdgeLength(self.graph, 66618411, 66615547)
    self.assertEqual(124.59299999999999, length)
```

Some unit tests for GraphUtils

5) **Debuggability -** Print statements and console logs have been added in the code to assist debuggability. The backend also sends informative responses for failures which are displayed on the UI.



```
console.log("Request parameters -> Elevation, Distance Percentage, Source, Destination")
console.log(elevationValue, distancePer, start_lat_lng, end_lat_lng);
requestURL = 'http://127.0.0.1:5000/path?elevation='+elevationValue+'&distance='+distancePer+'&src_lat='+start_lat_lng.lat+'&
requestURL += '&dest_lat='+end_lat_lng.lat+'&dest_lang='+end_lat_lng.lng;

console.log(requestURL);
```

Logs in mapper.js in the frontend

Functional Requirements

Input requirements:

1.) **Input of Start and End Location:** Users can enter the start and end location by inputting the addresses of start and end points.

2.) **Distance Limitation:** Users can set the maximum distance they are willing to travel between the start and end points, limiting it to a percentage (x%) of the shortest route between the two points.

3.) **Elevation Gain Minimization/Maximization:** Users can minimize, maximize or simply get the shortest path between the start and end points.

Output requirements:

4.) **Path Information:** Users can see relevant information about the suggested path, including distance and elevation gain. The optimized path is highlighted on the map to help the user visualize the journey better.

Processing requirements:

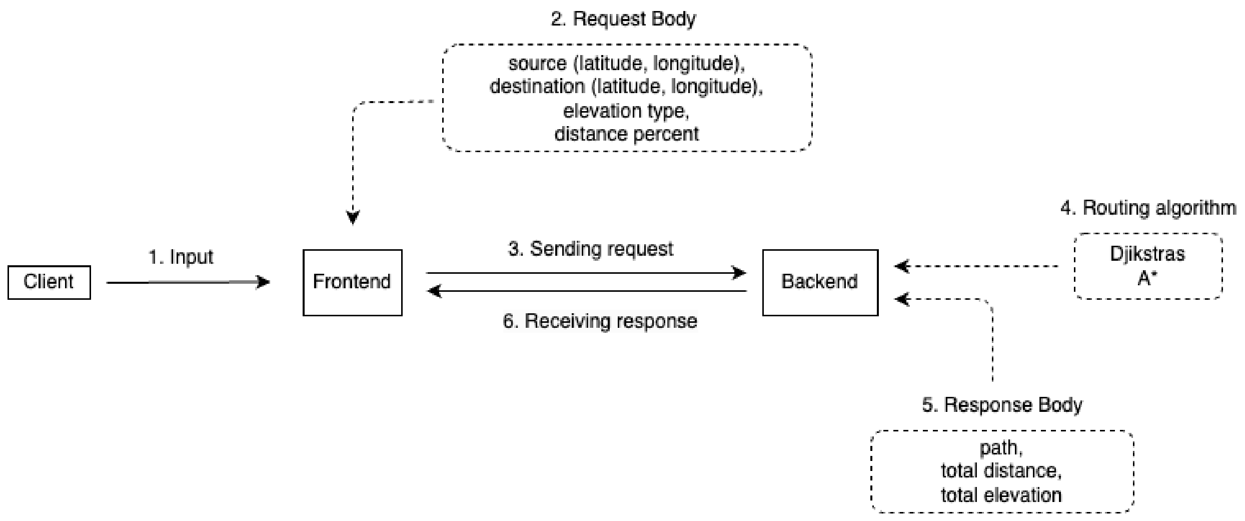
5.) **Path Selection Algorithm:** The system uses Dijkstra's algorithm and AStar algorithm to calculate and suggest the best possible route for the user based on the user's preferences for elevation gain and distance limitation.

Features

1. **Auto-complete for start and end location:** This feature allows users to type their desired location. On pressing enter, the best match is taken as input for the location and is shown on a text field.
2. **Markers for start and end point on the map**: Once users input their location, the appropriate marker for the location is shown on the map. The map is zoomed in/out accordingly
3. **Highlighting the path:** This feature allows users to see the path which the model has returned on the map.

2. DESIGN

2.1 Architecture/DataModel



2.2 Front-end Details

- We have used HTML and CSS to render the user interface of the web application.
- Javascript has been used for simple rendering logic within the front-end. It has also been used for sending HTTP GET requests using the *fetch* API.
- Leaflet.js has been used to add map functionality to the application. It allows operations like showing markers, highlighting the path and capturing latitude and longitude
- Users give the source and destination locations as input using the 2 search icons. The best matching location is used as input, and the appropriate marker is shown on the map. For example, putting the input as "Boulders Amherst" takes the location to The Boulders in Amherst. This uses Leaflet Geocoder with Google Maps API, through which we can find the latitude and longitude of the location
- Users can also choose among 3 path types - minimum elevation, maximum elevation or shortest path. They can then give the distance percentage as an input, whose default value is 0
- On clicking the 'Get Path' button, a request is sent to the backend.
    - Sample request - 'http://127.0.0.1:5000/path?elevation=0&distance=10&src_lat=42.3495879&src_lang=-72.5283459&dest_lat=42.3996673&dest_lang=-72.5267929'
        - Server is running on '127.0.0.1:5000'
        - 'elevation=0' - Shortest path. Value of 1 means min elevation, 2 means max elevation
        - 'distance=10' - Distance percentage of 10. Paths will be searched for within 10% of the shortest distance between source and destination
        - 'src_lat=42.3495879&src_lang=-72.5283459' - Source latitude and longitude

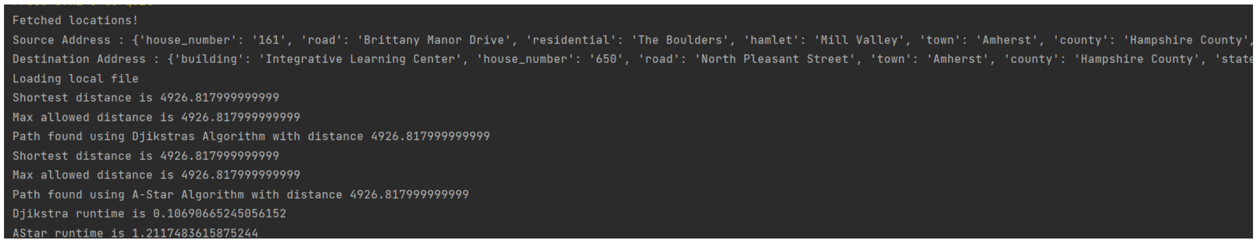
        - ‘dest_lat=42.3996673&dest_lang=-72.5267929’ - Destination latitude and longitude
- Once the response is received, the UI is updated to show the details within the response.
    - For a successful request, the path, distance and elevation is shown
    - If unsuccessful, the error message is shown

2.3 Backend Details

- We have used Python and Flask to run the backend server
- It listens to GET requests on <localhost>/path
- We have used the OSMnx module to load the geoLocation data and load a graph. We have used Google’s Elevation API to load elevation data for nodes and edges within this graph
- Nodes represent places within the graph with information like latitude, longitude, elevation
- Edges represent paths between 2 nodes. Each edge has its own length and elevation gradient
- On receiving a request, we do the following
    - Parse the request to get arguments - source location, destination location, path type and distance percentage
    - Create a *Graph* object, which loads the relevant graph for our task using OSMnx. It also loads elevation data
    - Call *Graph.getPath()* to get the required path. It returns the path with a list of latitude and longitude, the path distance, and the path elevation
- These details are then sent as the response
- Internally, Dijkstra’s and AStar algorithms have been used to compute the required routes. We have also computed the runtime for both, which are printed at the backend level. This helps in evaluating which algorithm is giving better performance, and can be chosen in the future.
- Below is a possible successful response

```
{"distance":4926.818,
"elevation":2.825,
"route":[[42.350747,-72.527387],[42.35076,-72.527387],.....,[42.390239,-72.5251188],[42.39075
7,-72.5252034]]}
```

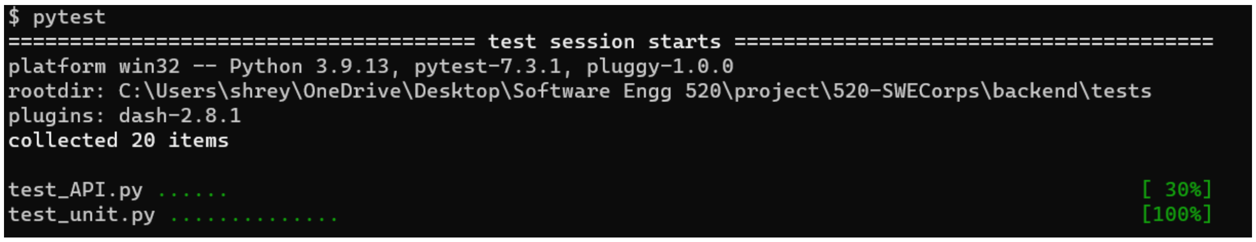
- Example of possible backend console



```
Fetched locations!
Source Address : {'house_number': '161', 'road': 'Brittany Manor Drive', 'residential': 'The Boulders', 'hamlet': 'Mill Valley', 'town': 'Amherst', 'county': 'Hampshire County',
Destination Address : {'building': 'Integrative Learning Center', 'house_number': '650', 'road': 'North Pleasant Street', 'town': 'Amherst', 'county': 'Hampshire County', 'state
Loading local file
Shortest distance is 4926.817999999999
Max allowed distance is 4926.817999999999
Path found using Djikstras Algorithm with distance 4926.817999999999
Shortest distance is 4926.817999999999
Max allowed distance is 4926.817999999999
Path found using A-Star Algorithm with distance 4926.817999999999
Djikstra runtime is 0.10690665245056152
AStar runtime is 1.2117483615875244
```

2.4 Testing and Evaluation

- **Unit Tests**
    - We have used the PyTest testing framework to write tests
    - We have written unit tests for functions and classes in the *backend* in *test_unit.py*
    - They evaluate success and failure cases of methods, output of algorithms etc
    - We have set some predefined source and destination points. The source truth for the shortest path has been picked from Google maps. We evaluate the correctness of our shortest path algorithm output using those.
    - To evaluate the correctness of our minimum and maximum elevation paths, we check whether the distance along the path is within our limits, and whether the elevation is less/more (as required) than the elevation of the shortest path
    - Testing has been done majorly for points around Amherst
- **Integration Tests**
    - We have written mock API calls to the backend with valid and invalid parameters in test_API
    - For valid parameters we have a predefined source and destination (as above).

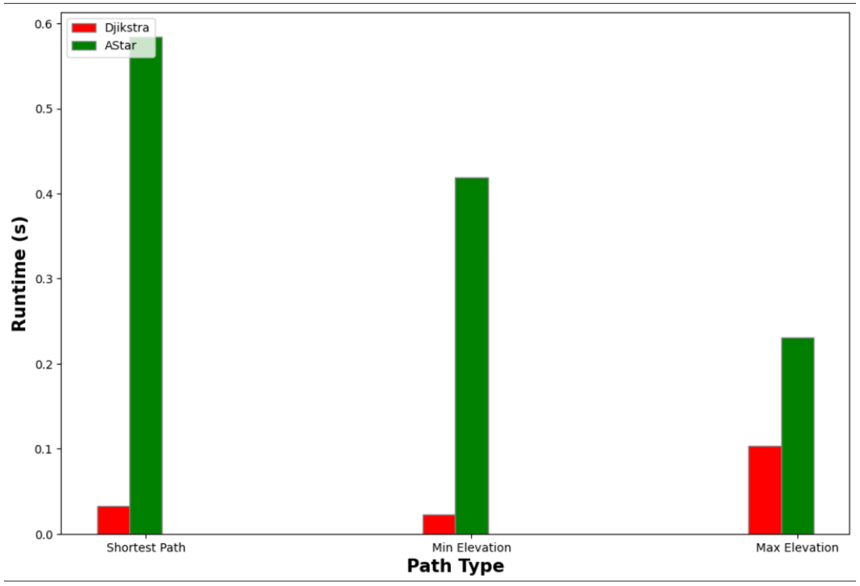


```
$ pytest
===================================== test session starts ======================================
platform win32 -- Python 3.9.13, pytest-7.3.1, pluggy-1.0.0
rootdir: C:\Users\shrey\OneDrive\Desktop\Software Engg 520\project\520-SWECorps\backend\tests
plugins: dash-2.8.1
collected 20 items

test_API.py ......                                                                        [ 30%]
test_unit.py ..............                                                               [100%]
```

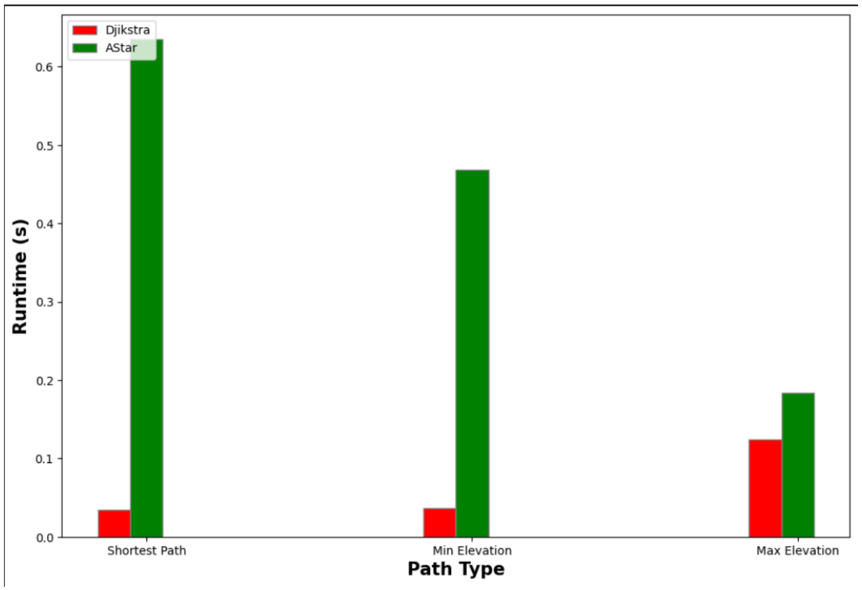
Unit tests and Integration Tests

- **Peer evaluation for Frontend**
    - We showed Beta versions of the application to peers and worked on the UI based on that. The final working prototype has been shown in the README

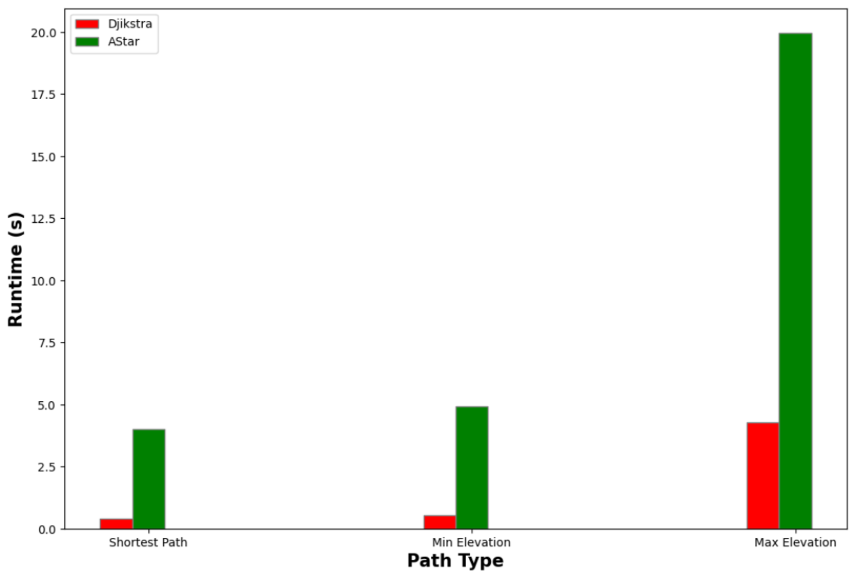
    - This also helped us find bugs within the UI which we could solve. For example - Our highlighted route on the map was not getting cleared on selecting the source and destination multiple times.
- **Algorithm Runtime Evaluation**
    - We have calculated the runtime of our implementation of Djikstras and AStar algorithms for a combination of 5 different source and destination pairs.
    - For each source and destination, we select different elevation option from amongst shortest path, minimum elevation, maximum elevation
    - Thus, we show 5 different bar graphs for comparing our algorithms



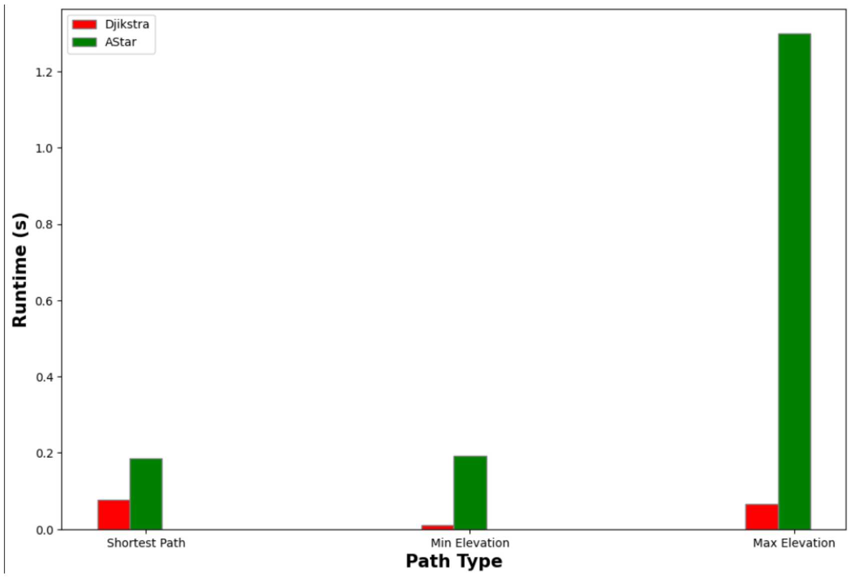
Runtime comparison from Boulders to DuBois library



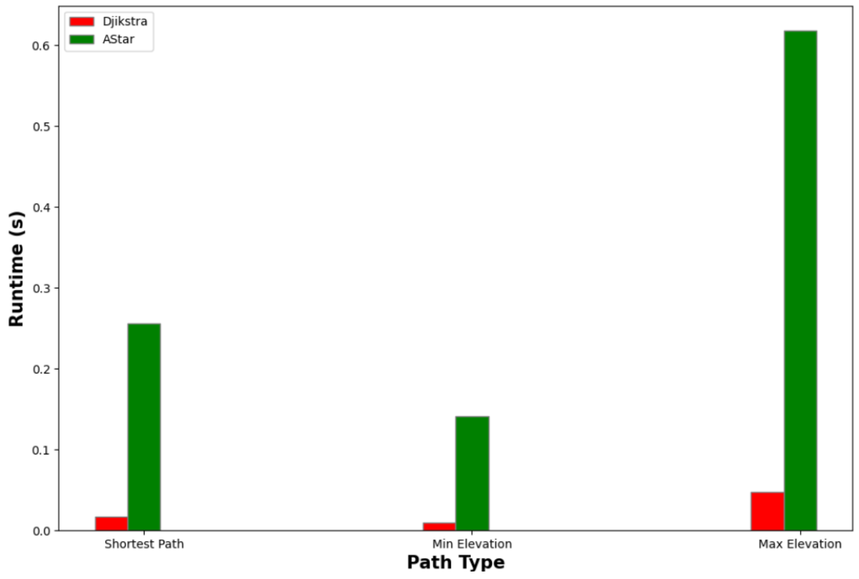
Runtime comparison from Boulders to Amherst Common



Runtime comparison from Boulders to Amherst Common



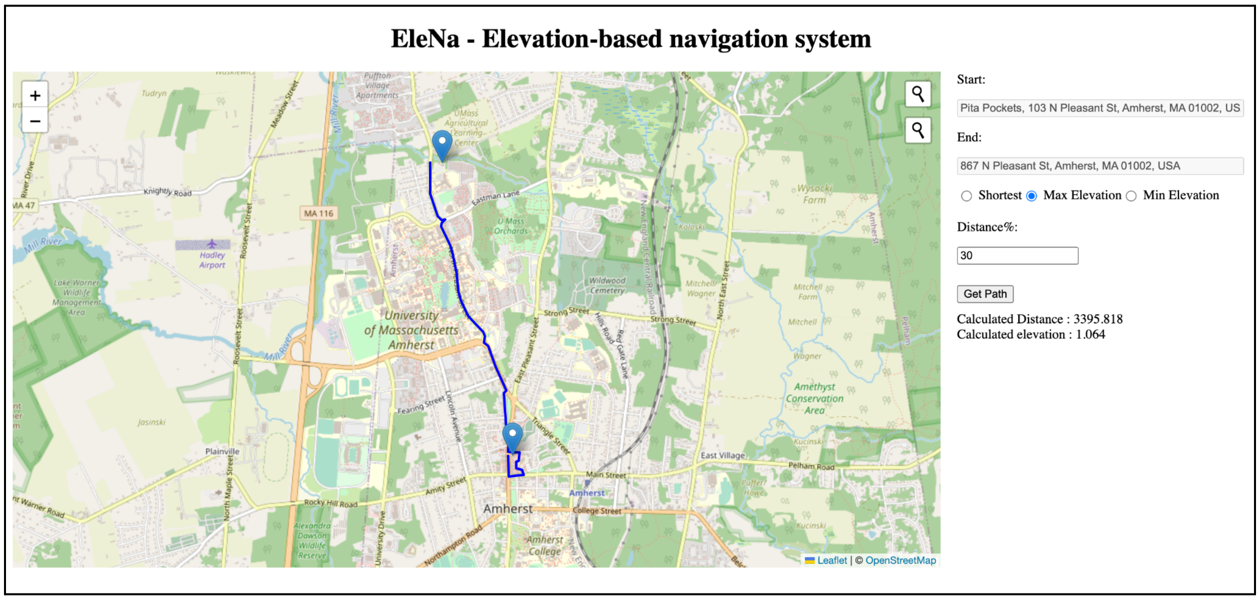
Runtime comparison from Boulders to Amherst Common



Runtime comparison from Mullins center to Emily Dickinson Museum

2.5 UI

- Path is displayed on the map and total distance and elevation is displayed along the path



EleNa - Elevation-based navigation system

Start:

Pita Pockets, 103 N Pleasant St, Amherst, MA 01002, US

End:

867 N Pleasant St, Amherst, MA 01002, USA

Shortest [✓ Max Elevation] Min Elevation

Distance%:

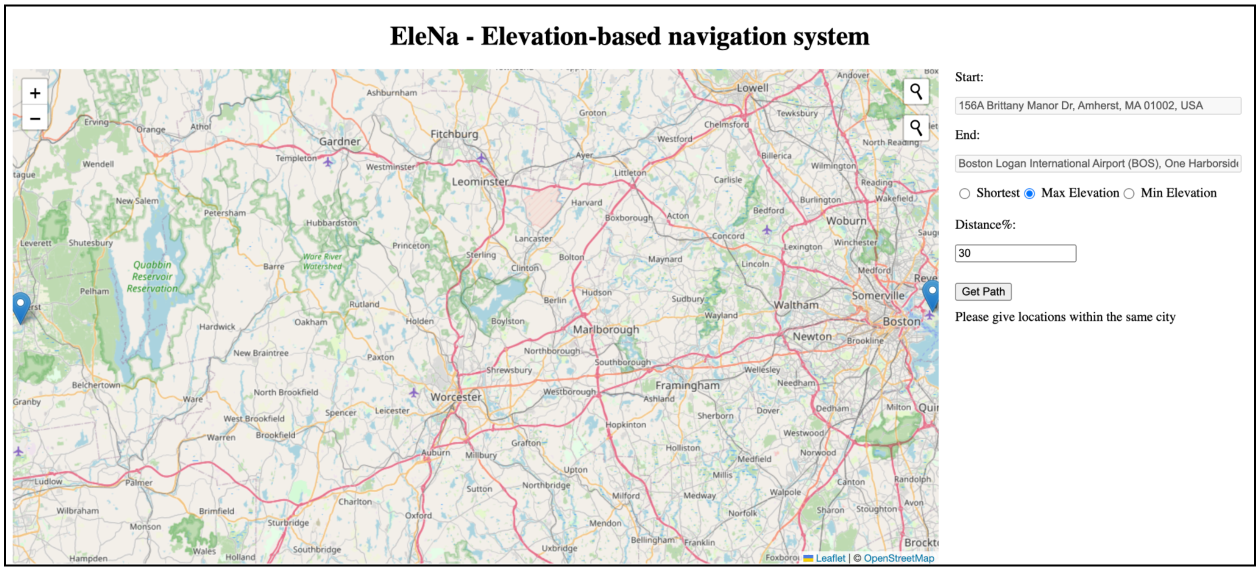
30

Get Path

Calculated Distance : 3395.818
Calculated elevation : 1.064

University of Massachusetts Amherst
Amherst
Amherst College
Leaflet | © OpenStreetMap

- If the user enters source/target locations from different cities, the corresponding error message will be displayed



EleNa - Elevation-based navigation system

Start:

156A Brittany Manor Dr, Amherst, MA 01002, USA

End:

Boston Logan International Airport (BOS), One Harborside

Shortest [✓ Max Elevation] Min Elevation

Distance%:

30

Get Path

Please give locations within the same city