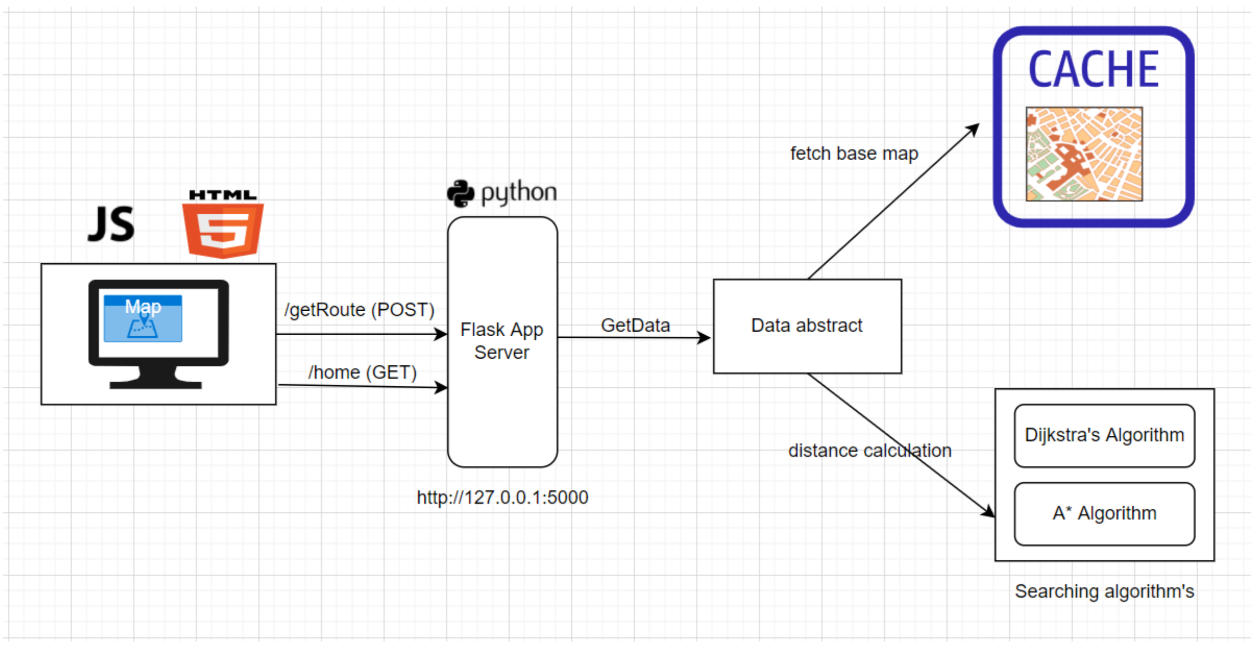
1. **API endpoints:**
    a. /home: This is a GET request which loads the home page of the app.
    b. /route: This is a POST request which accepts the following parameters in the request payload:
        i. Start location: latitude
        ii. Start location longitude
        iii. End location latitude
        iv. End location longitude
        v. Elevation ratio
        vi. Elevation type: This parameter takes whether we want to maximize elevation or minimize elevation. (Values: min, max)

2. **Caching**: We are using Python Pickle in order to implement caching. Downloading the entire graph whenever a user searches for the shortest path from UI is extremely time-consuming and requires lots of bandwidth and is overall a bad user experience. So we have developed a caching mechanism in order to mitigate this issue. Whenever a user searches for the shortest path from UI, the backend system downloads the map of that area and saves it as bytes data in a pickle file called “graph.p”. Pickle module allows us to deserialize this bytes data back to the graph for performing distance calculation. So every time user performs a query on the UI for a different start and endpoint, the system doesn’t download the entire graph again but has prefetched data downloaded which will make the /route API faster and more efficient.

    Here we are assuming that the user will search for queries in the Amherst area and hence we have a pre-download graph of Amherst in file graph.p. If a user wishes to query for any other area, then they would need to delete the existing graph.p and update UI. The new graph.p will then be downloaded by the system and that area will be in our cache henceforth.

3. **Architecture**:



We have modeled our system according to Client-Server architecture. Inside the server, we have implemented the entire backend system using Python, Flask, Photon, OpenStreetMap, and Map Box API.

Backend: The backend system has 2 APIs as mentioned in the API endpoints subsection.
The main method calls the data_abstract python file's getData method which has all the shortest path calculations encapsulated in it.

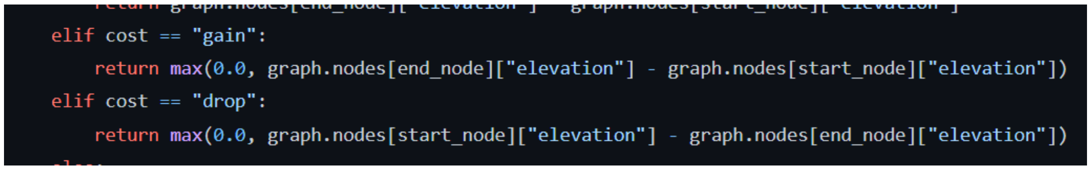
a. **Data_abstract's getData method:**
   1. Initializes Photon which is a geocoding service that aims to let users search while typing with OpenStreetMap.
   2. Initializes AbstractMap i.e loading graph from graph pickle file and deserialize it into a graph data structure.
   3. Invokes distance calculation from class "distance_calculate" and fetch the shortest path
   4. Get data from the shortest path in a model containing
      a. Start location,
      b. End location
      c. Elevation route
      d. Shortest route
      e. Shortest Route's distance, elevation gain, and elevation drop
      f. Elevated Route's distance, elevation gain, and elevation drop

b. **Distance Calculation:**
   1. We first calculate the shortest path between locations specified by the user using the Networkx python library.

   2. We then calculate the Elevated path using Dijkstra’s algorithm and the A* algorithm. In the case of maximizing elevation gain, whichever algorithm returns the maximum elevation gain route will be returned as the API response from the backend whereas, In the case of minimizing elevation gain, the minimum elevated route will be returned.

4. **A* Heuristics:**
   The A* family of algorithms which are often used for path finding and graph traversals, use a cost score of the form (to choose which path to continue on, in every iteration until at final node):
   f(n) = g(n) + h(n)
   Where :
   The g(n) function tells us the cost of the path to any vertex n from the start node.
   The heuristic function h(n) tells A* an estimate of the minimum cost from any vertex n to the goal.

   The heuristic function h(n) is the problem specific one, where it is defined as per the objective that needs to be accomplished. In our case as we looking to maximize or minimize the elevation, we use the get_edge_cost() function in our code to model our requirements as a cost function as follows(snippet of function implementation quoted below):



```
    elif cost == "gain":
        return max(0.0, graph.nodes[end_node]["elevation"] - graph.nodes[start_node]["elevation"])
    elif cost == "drop":
        return max(0.0, graph.nodes[start_node]["elevation"] - graph.nodes[end_node]["elevation"])
```

   For maximizing the elevation the get_edge_cost() is used in “drop” mode as it returns the minimum cost of zero for any route choice which has the elevation increasing and the elevation difference value in case the elevation is decreasing, hence increasing the cost(penalty) proportionately to the difference in elevation decrease of that sub-route.

   For minimizing the elevation the get_edge_cost() is used in “gain mode as it returns the minimum cost of zero for any route choice which has the elevation decreasing and the elevation difference value in case the elevation is increasing, hence increasing the cost(penalty) proportionately to the difference in elevation increase of that sub-route.

5. **Test Plan:**
   We perform unit tests to check the functionality of the necessary functions powering our implementation of the EleNa application in the tests/test.py.

```
    get_graph(end)
    get_route(A)
    get_shortest_path()
    get_Elevation(A)
    get_cost(A)
    get_geojson(start)
    get_data(start, end)
```

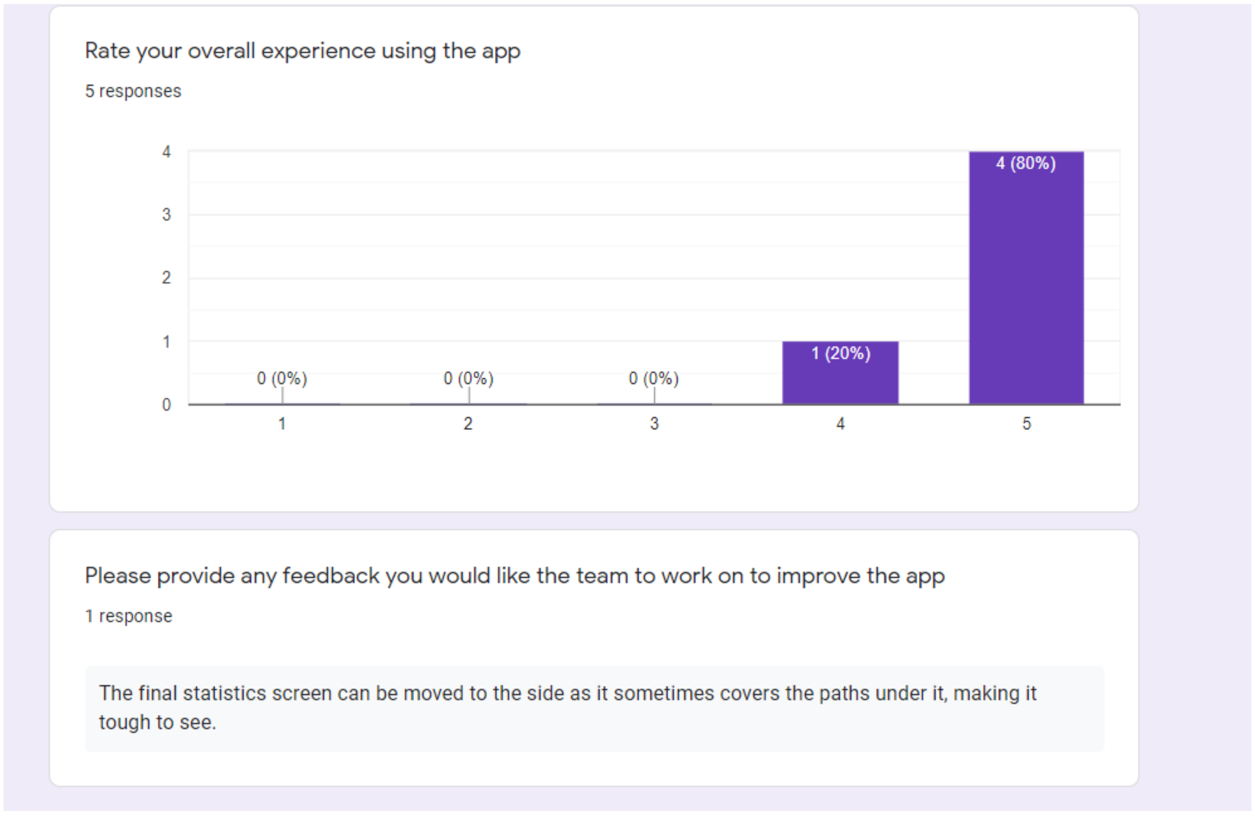
We test the above functions of our codebase and their associated functionality with our test cases.

6. **Experimental Evaluation:**

   We collected user feedback on the UI and the general functionality of our Elena app using the following Google Form -

   https://forms.gle/JSzvqAFf1og3DmzXA

   The feedback was mostly very positive (mostly 5 on overall app experience), with only one of the users wanting to move the final route stats box, away from the center of the screen rating the overall experience as 4.


Rate your overall experience using the app

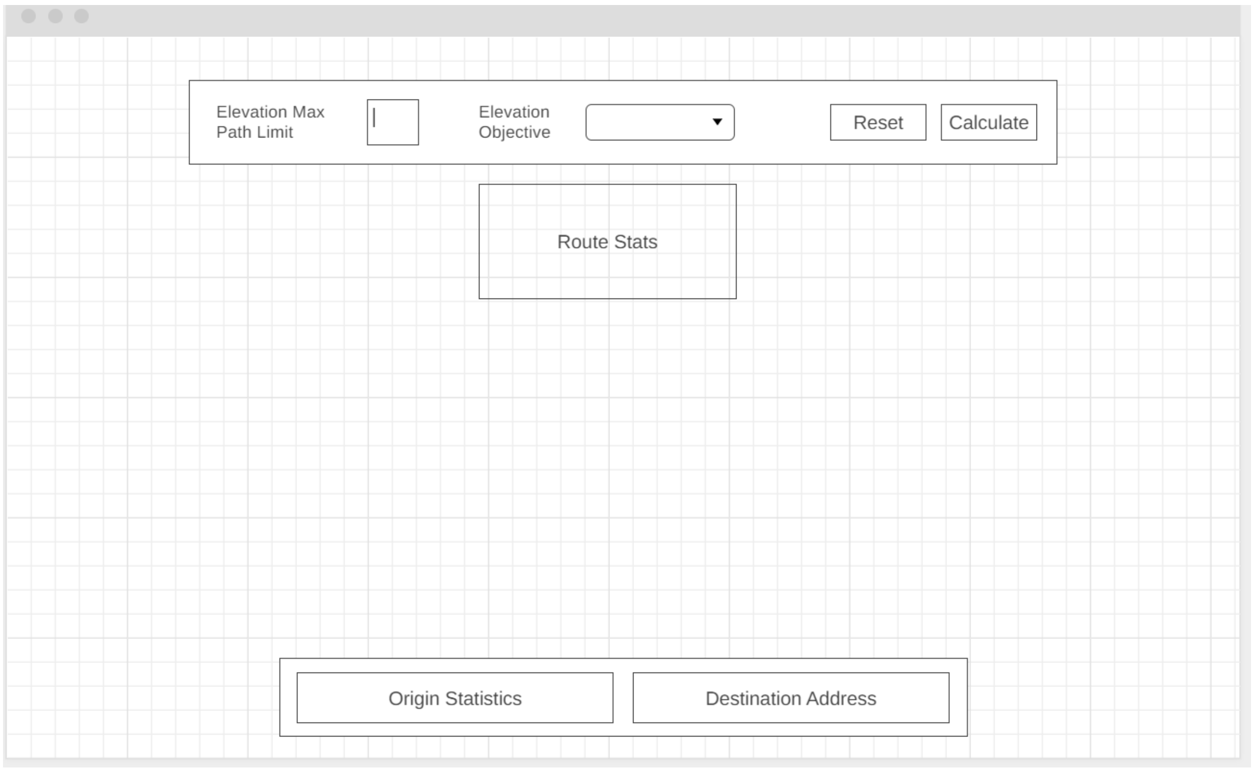
5 responses

Please provide any feedback you would like the team to work on to improve the app

1 response

The final statistics screen can be moved to the side as it sometimes covers the paths under it, making it tough to see.

7. **Initial WireFrame:**


Elevation Max Path Limit
Elevation Objective
Reset
Calculate
Route Stats
Origin Statistics
Destination Address