Evaluation Document

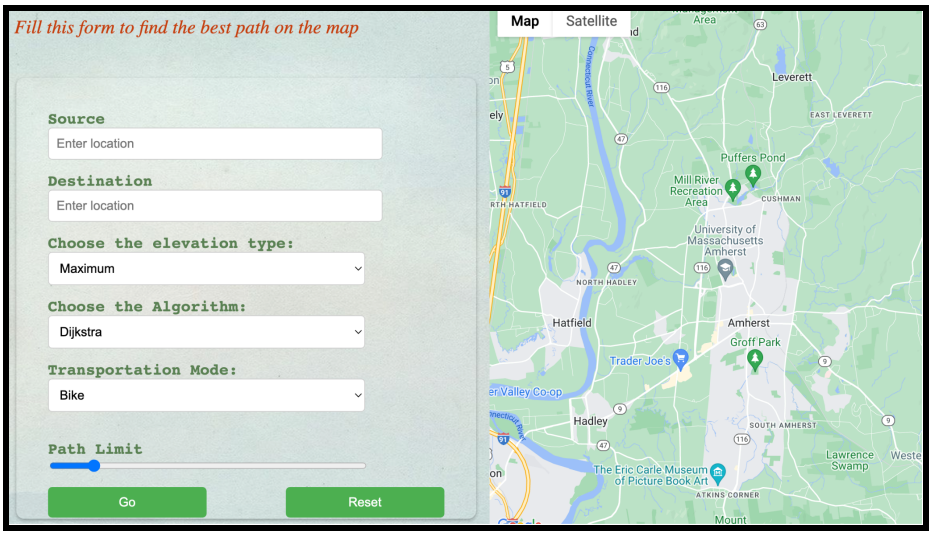
Group name - Kashikoi
Group members - Divya Sharma, Astha Baranwal, Rohan Acharya, Kavya Harlalka
Github link - https://github.com/kavyaharlalka/kashikoi-elena-navigation
Presentation link -
https://drive.google.com/file/d/1kYjDzBjgHvEUk565uJN3MAUknJmqa5kL/view?usp=share_link

Testing

Testability is important to keep our software smooth and free from bugs. We have tested the entire application to ensure that it works according to the specified requirements. We have done testing to ensure that all the functions and modules are tested and all corner cases are covered.

1. **UI Test:** We have also performed UI Design Review and have attempted to incorporate good UI Design Principles like Schneiderman's 8 golden rules in order to make the application more understandable, user-friendly and aesthetically pleasing. Some of these are noted below:

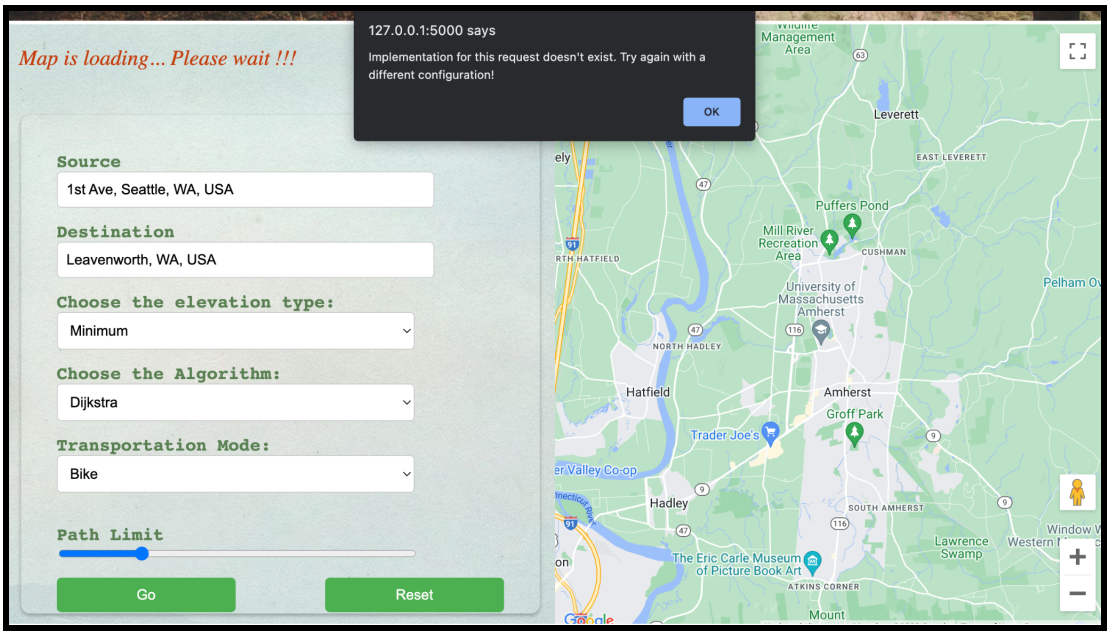
    - Consistency: we have also maintained consistency in designing the various elements such as colors and fonts in our application. For instance, we have used the green color as the theme of our application which is prominent in our background image as well as the label colors for the "Go" and "Reset Map" and the text for the Help menu. This creates a cohesive and professional look for our application.



- Selecting appropriate colors: We have chosen colors that reflect the purpose and branding of our application. Since the EleNA system is designed primarily for bikers, hikers and travelers, we have chosen the green colors as the theme of our application to evoke desired emotions and appeal to the user of the application.

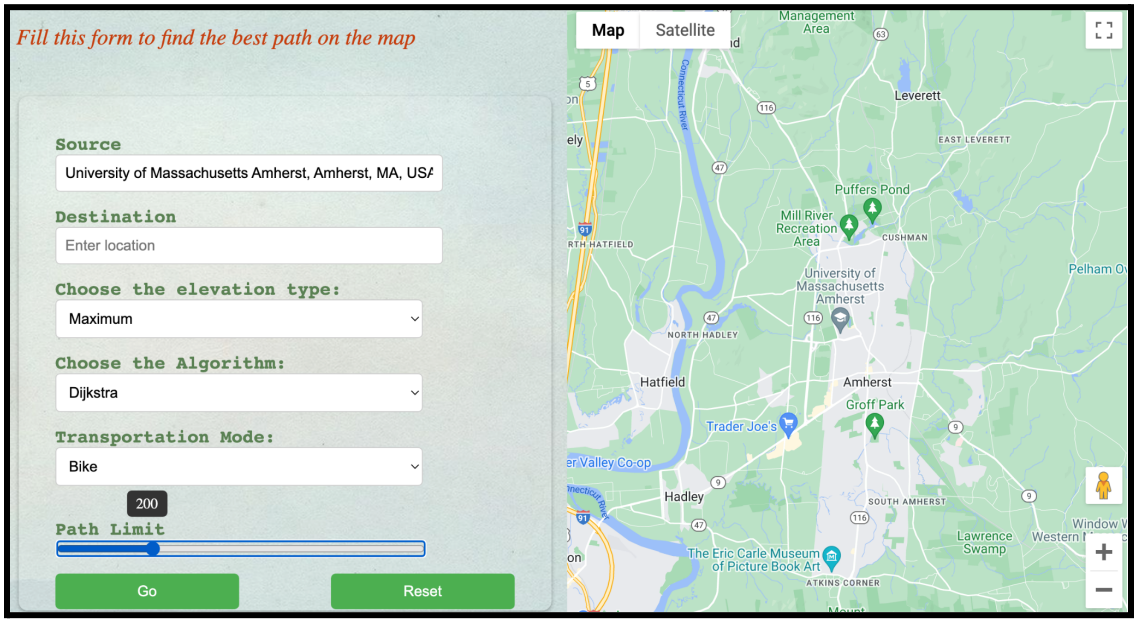
- Ease of use: we have added auto completion feature in our application for the source and destination addresses so that the user need not type in the complete address manually, and is assisted by the system to enter the appropriate address. This reduces the chance of entering invalid addresses to be handled by the application.

- Error handling: If the user enters addresses outside the location scope of our application, we have displayed error messages to prompt the user about the same. We have also added timeout errors if the server takes more than 30 seconds to respond to any API call. These clear and informative error messages help users understand what went wrong and how to correct it.

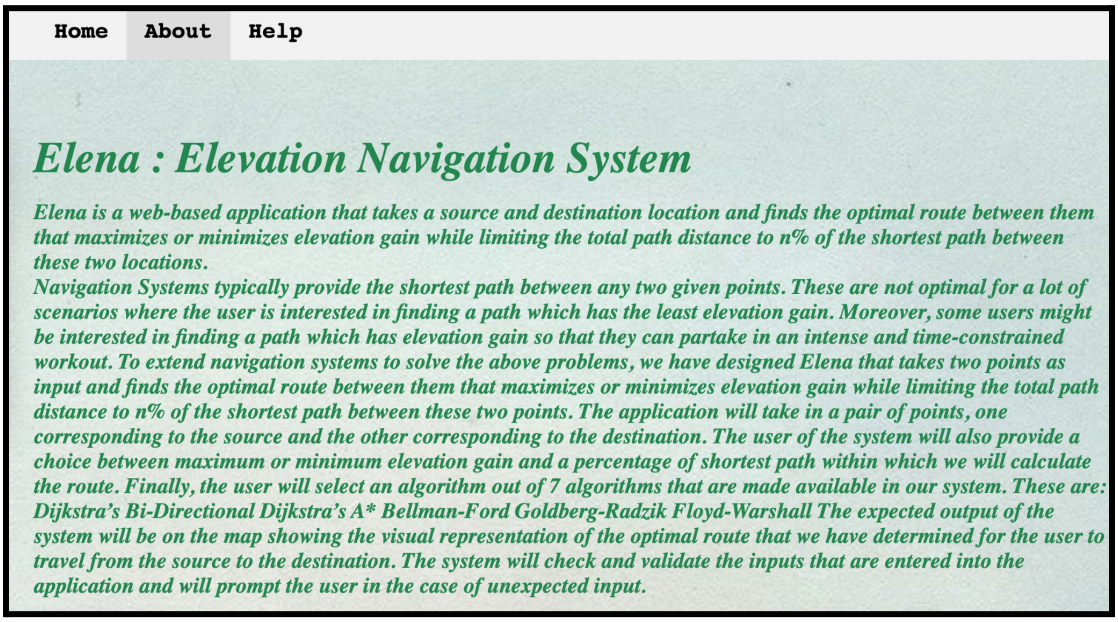


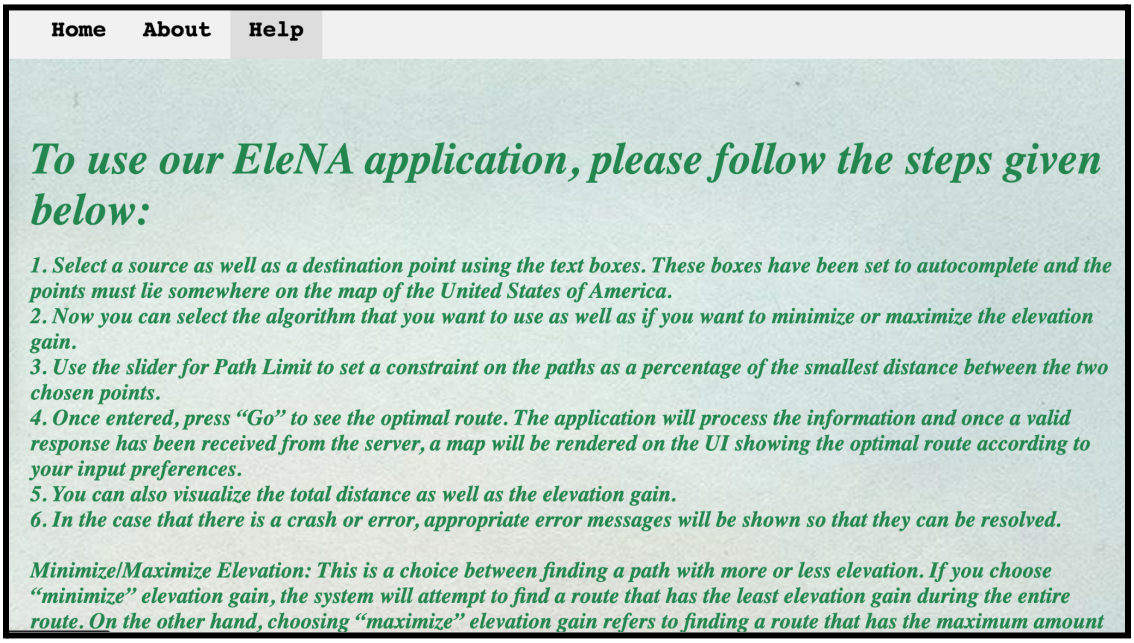
In the above image, on entering a valid address in Seattle, WA, that falls outside a 30 km radius around Amherst, we have displayed that our application does not support this request and that the user should change the configuration and run the application.

- Visual feedback: we have provided visual feedback to users when they interact with the buttons in our application. When they hover over any button, the color dims to reflect that they have chosen the respective button. Moreover, when the user hovers over the path limit, they are able to see the numeric value that is representative of the point that they are viewing.



- We have provided Help and About sections for our application which will allow the user to navigate to these pages in order to get more information about the EleNA system and how to navigate and use the application. These are meant to make the application user-friendly and let the user be in control.

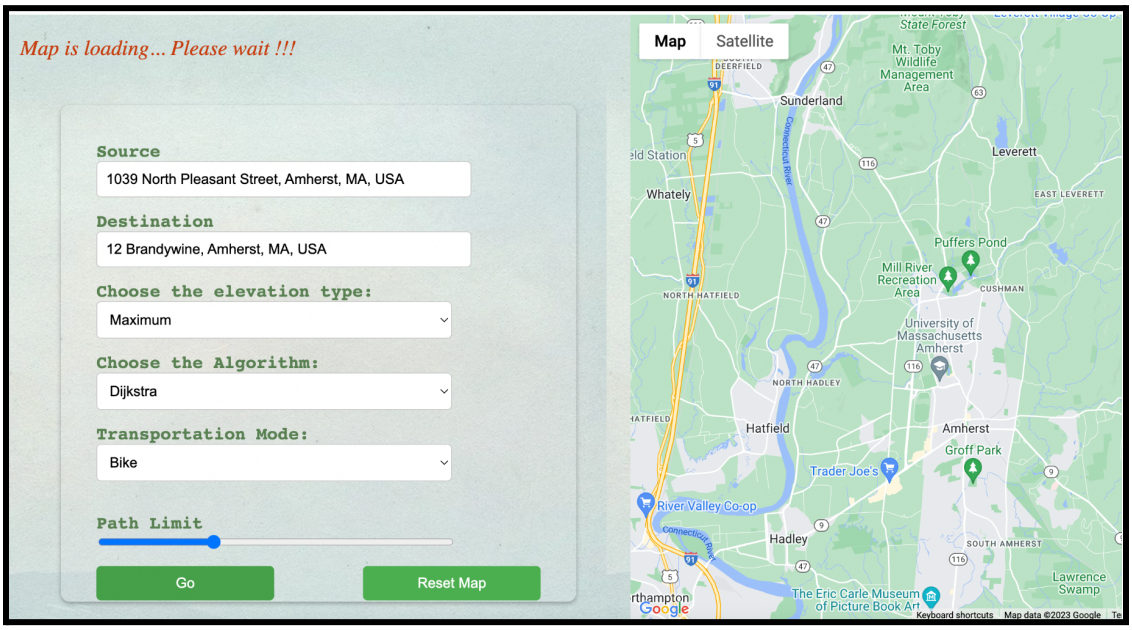




- Shortcuts: We have added a shortcut for the user to click on "Click here to find your journey.." as part of our EleNA tagline to avoid scrolling down and entering the points between which they want to find an optimal route.

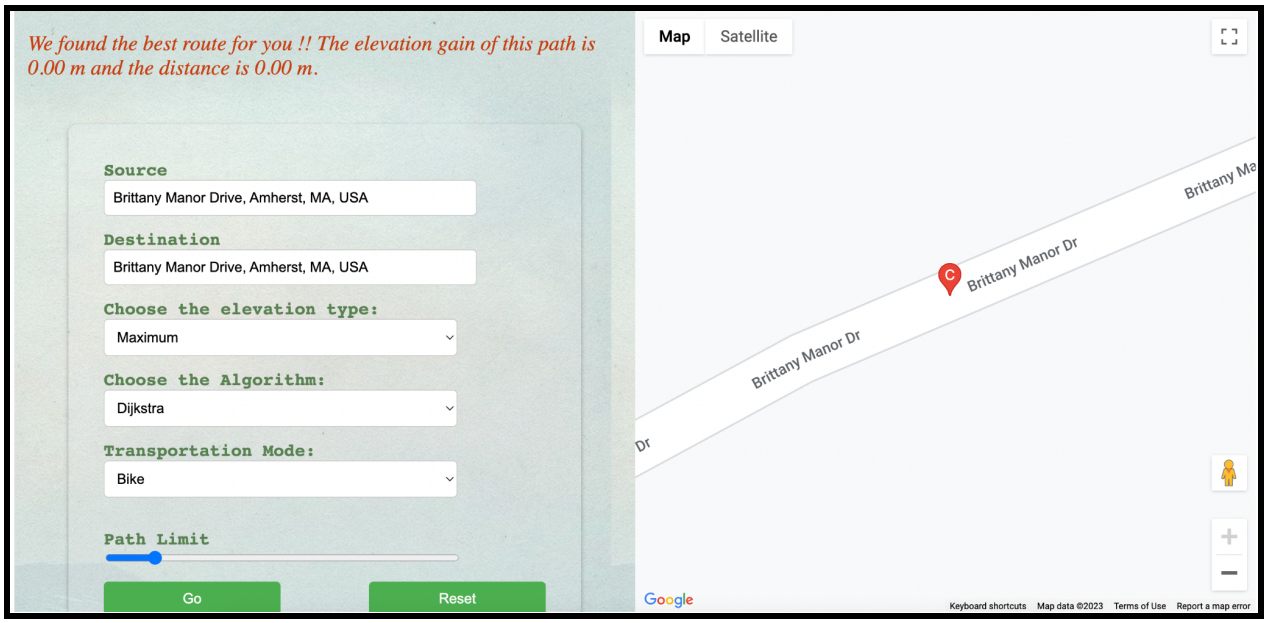
- Permit easy undoing of actions: We have added a reset map button to the user if they want to undo their search action and enter a fresh input to the system.

- Offer informative feedback: When the user clicks on “Go” to fetch a path on the map, we display a “Map is loading… Please wait!!!” prompt to show the user that the request is being processed.

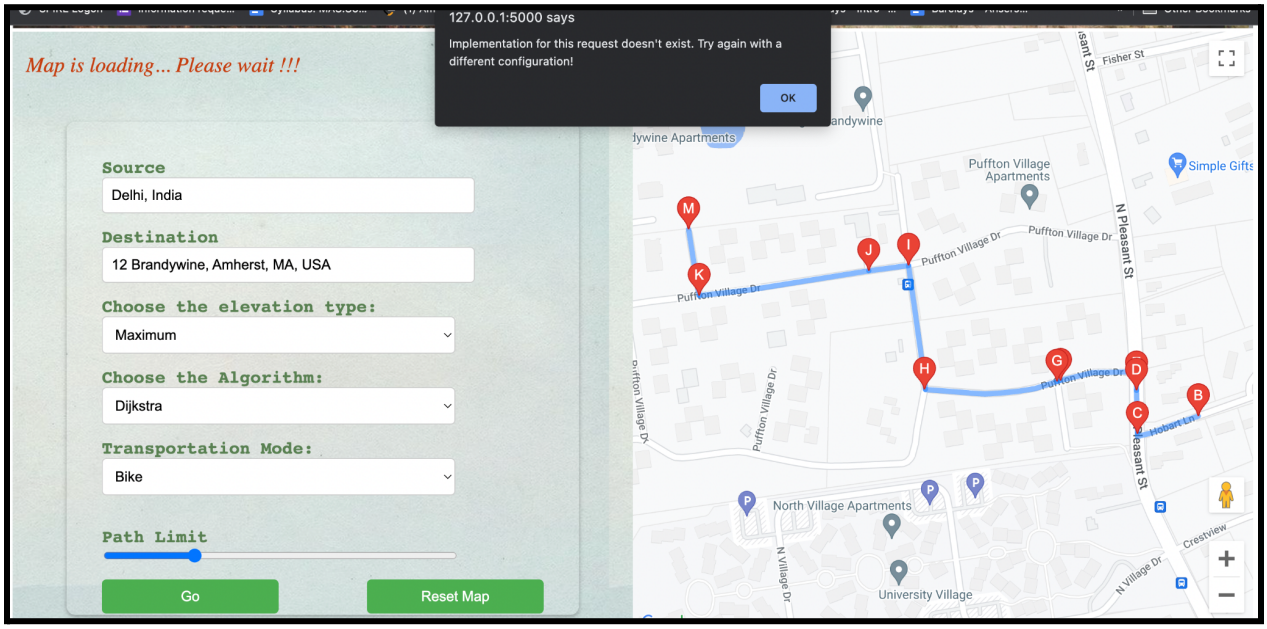


- User testing and feedback: We iteratively worked on frontend prototypes and asked our colleagues and peers for feedback during the mid-point project fair in class. Moreover, we continued incorporating feedback till the final project fair from our peers. This feedback was on the design choices as well as the overall appearance of the application. Based on their feedback, we were able to identify how to make the application more user-friendly and we have incorporated these in the final application.

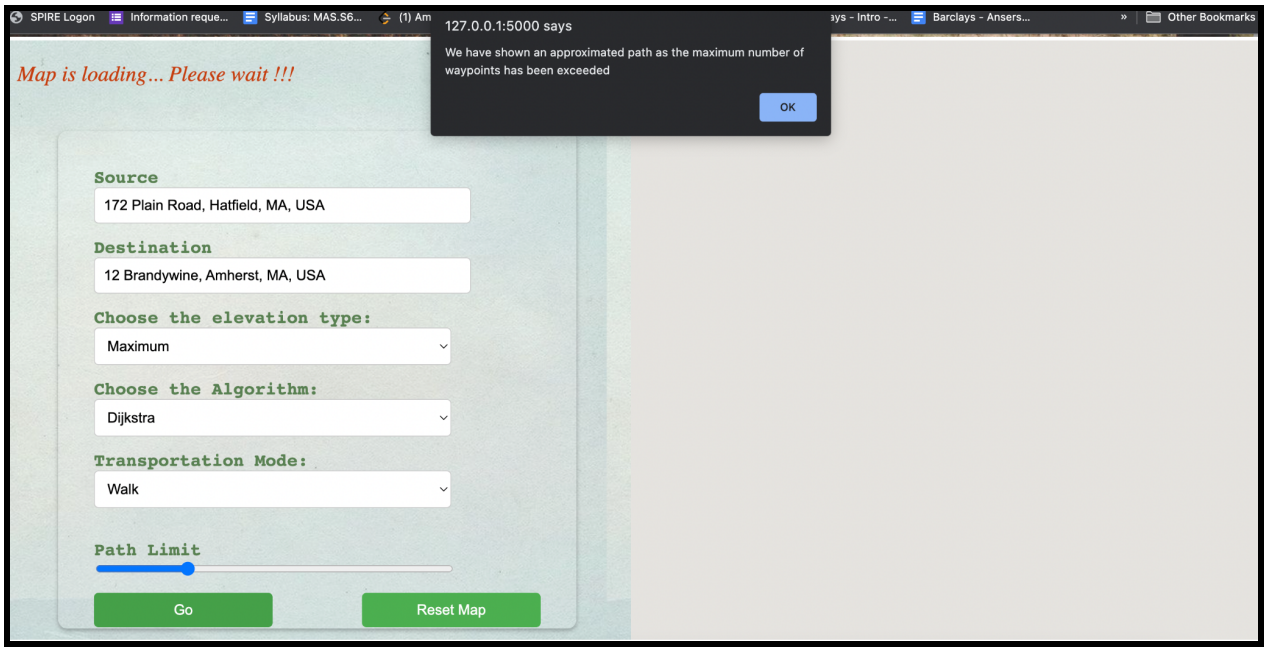
2. **Manual Test:** We have tested edge cases and bugs manually. We have used Postman as well as the UI for testing various situations and source and destination points on the map. The following edge cases were covered in these tests:
    a. In the case that the source and destination nodes are the same, we have handled the situation by displaying only one marker on the map which reflects that the same 2 points have been entered in the application. We have not considered it as an error scenario and hence not added any alert.

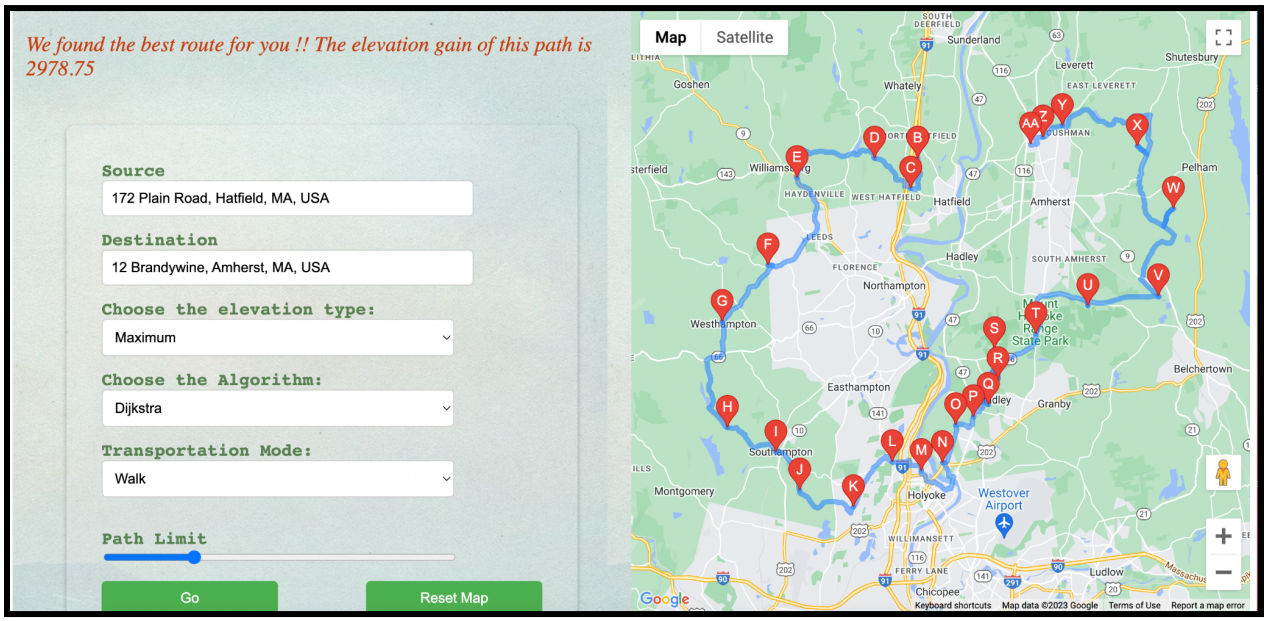


b. In the case that the user has entered a location which is not within the scope of our application (a 30 km radius around Amherst), we have displayed a message that the location cannot be processed.

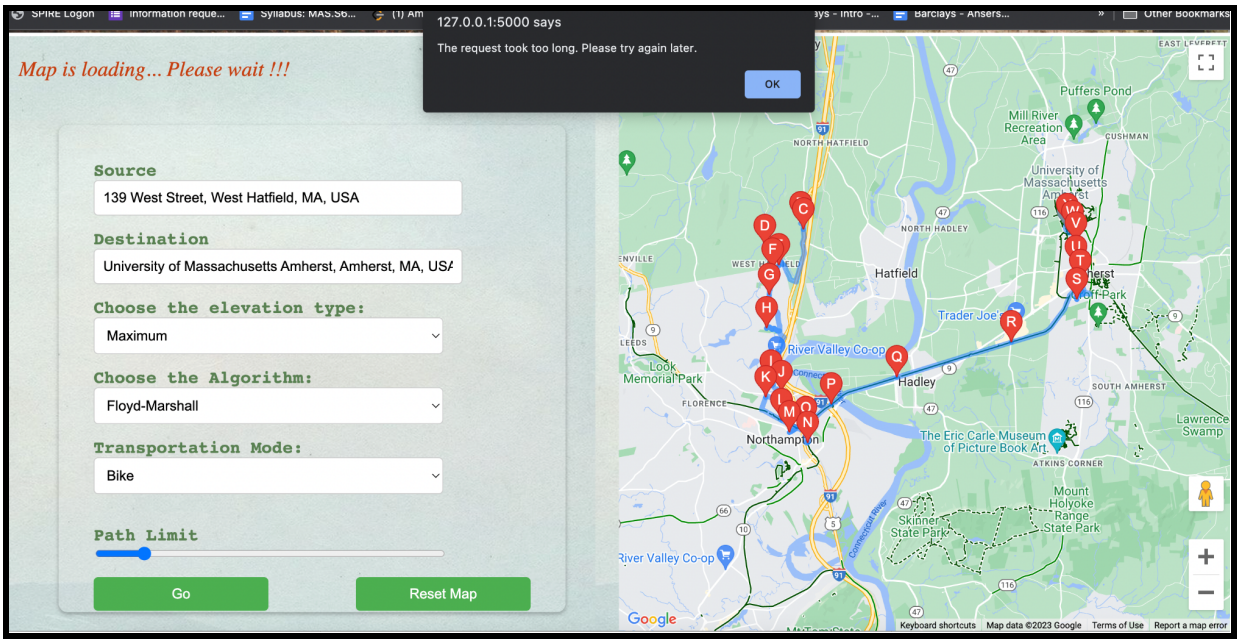


c. In case that the source and destination points entered are such that the number of nodes between them are very large (number of path coordinates >25), our application cannot calculate the exact path since we are using the free version of Google Maps Elevation API. In this case, we have handled this by computing an approximate path for this distance and also alerted the user about the same.

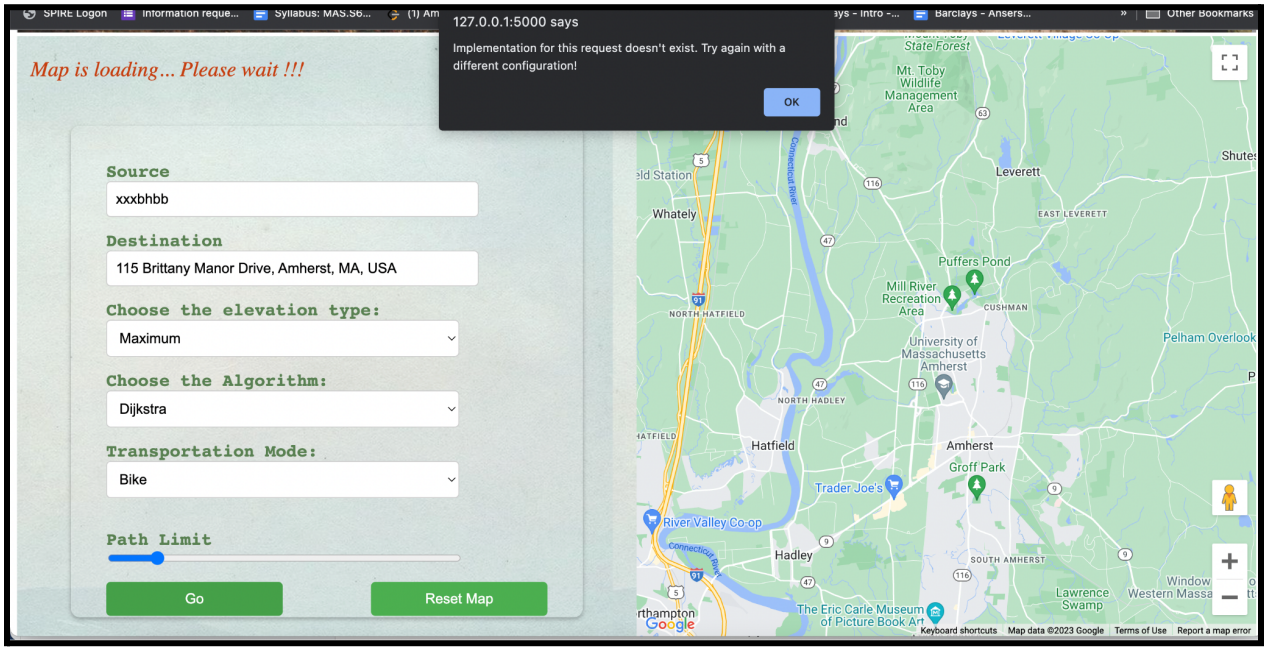




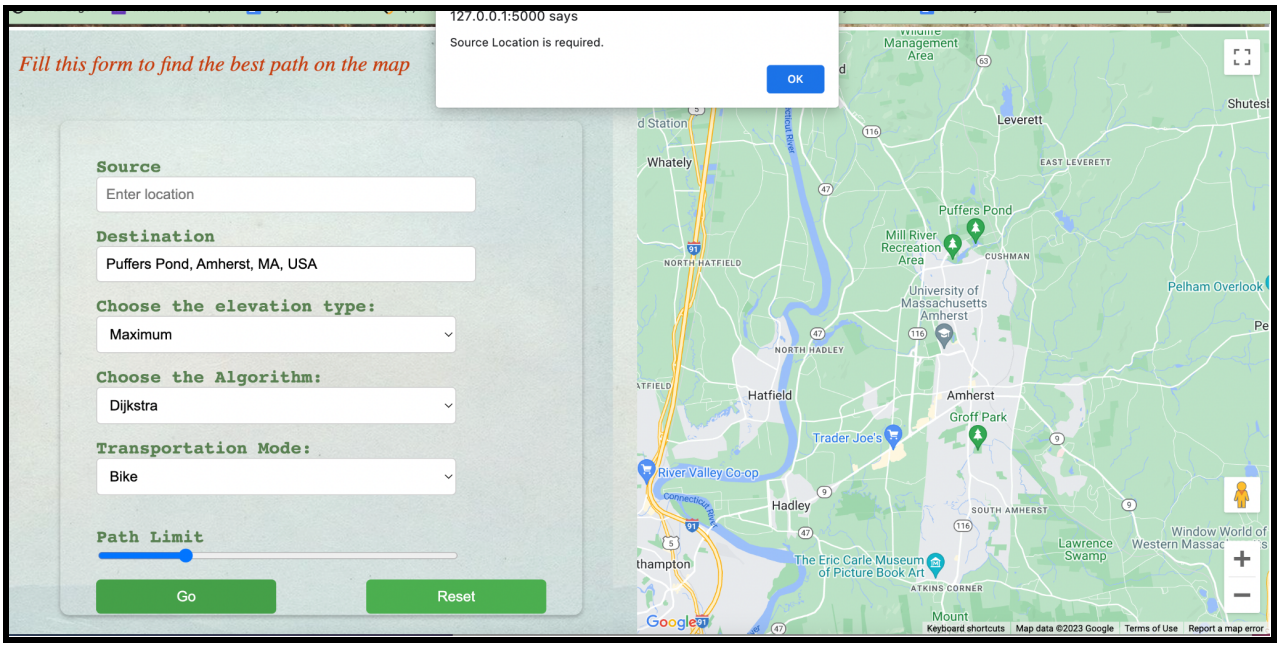
d. If the server is taking a time longer than 30 seconds to handle the request, we display a timeout message to alert the user.



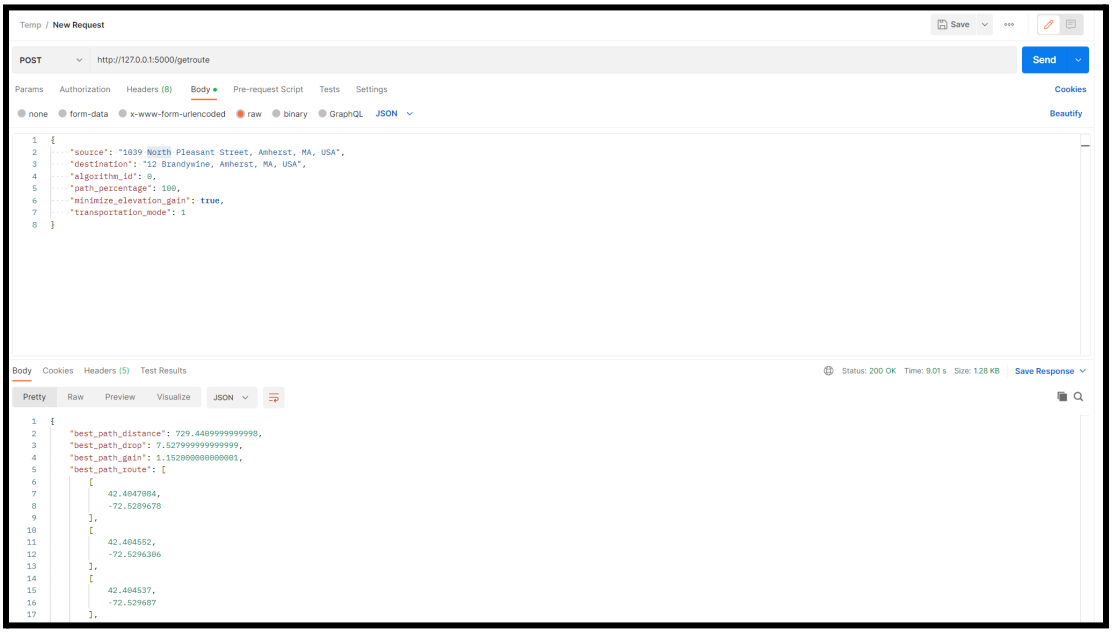
e. We have also tested the cases where the source and destination points are entered randomly, where the user does not use autocomplete to fill in the addresses correctly.



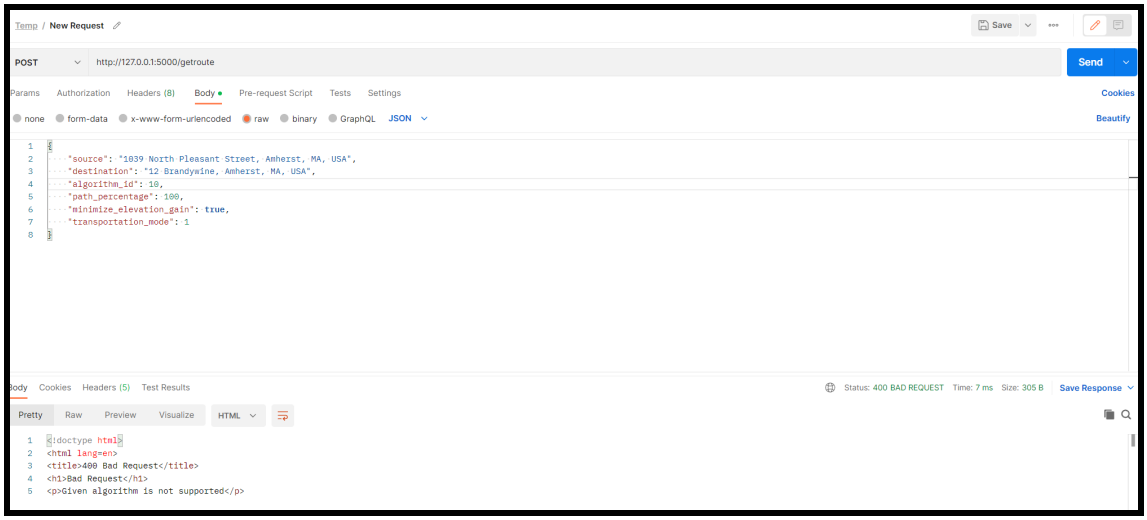
f. We have also tested the cases where the source or destination points are empty and press the go button to fetch the map. In this case, the UI shows an alert message.



We also used Postman to manually send requests and verify its working for valid scenario



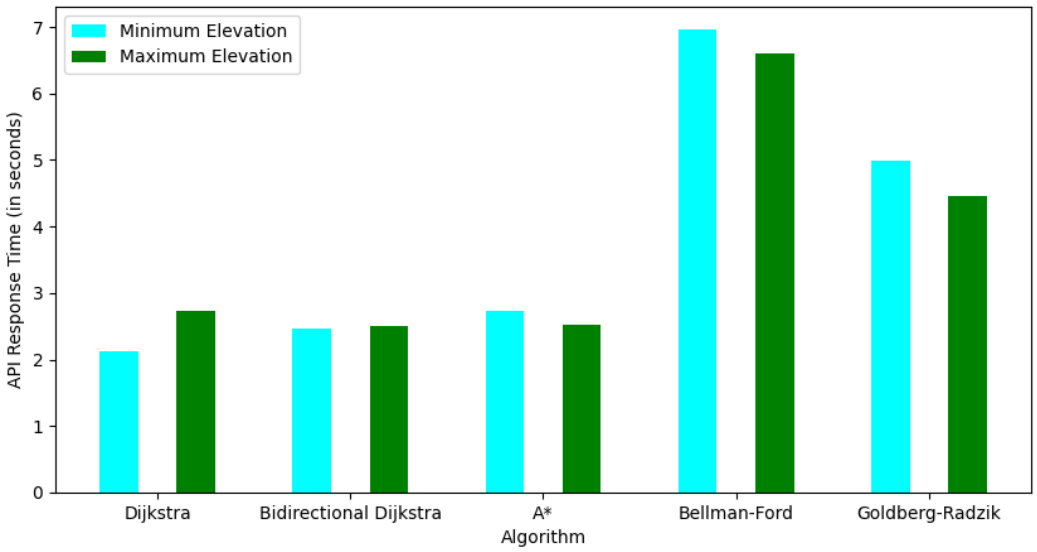
As well as invalid scenarios (we tested all possible scenarios in which data could be considered invalid, but have just pasted one screenshot below for reference)



3. **Performance Test:** We have tested the performance of the /getroute API used in the application for fetching the best path coordinates, from the response-time perspective. We observed the response time taken by the API for different scenarios like response time for different algorithms with minimum and maximum elevation gain setting and response time for different destinations from a particular source and algorithm with minimum and maximum elevation gain.

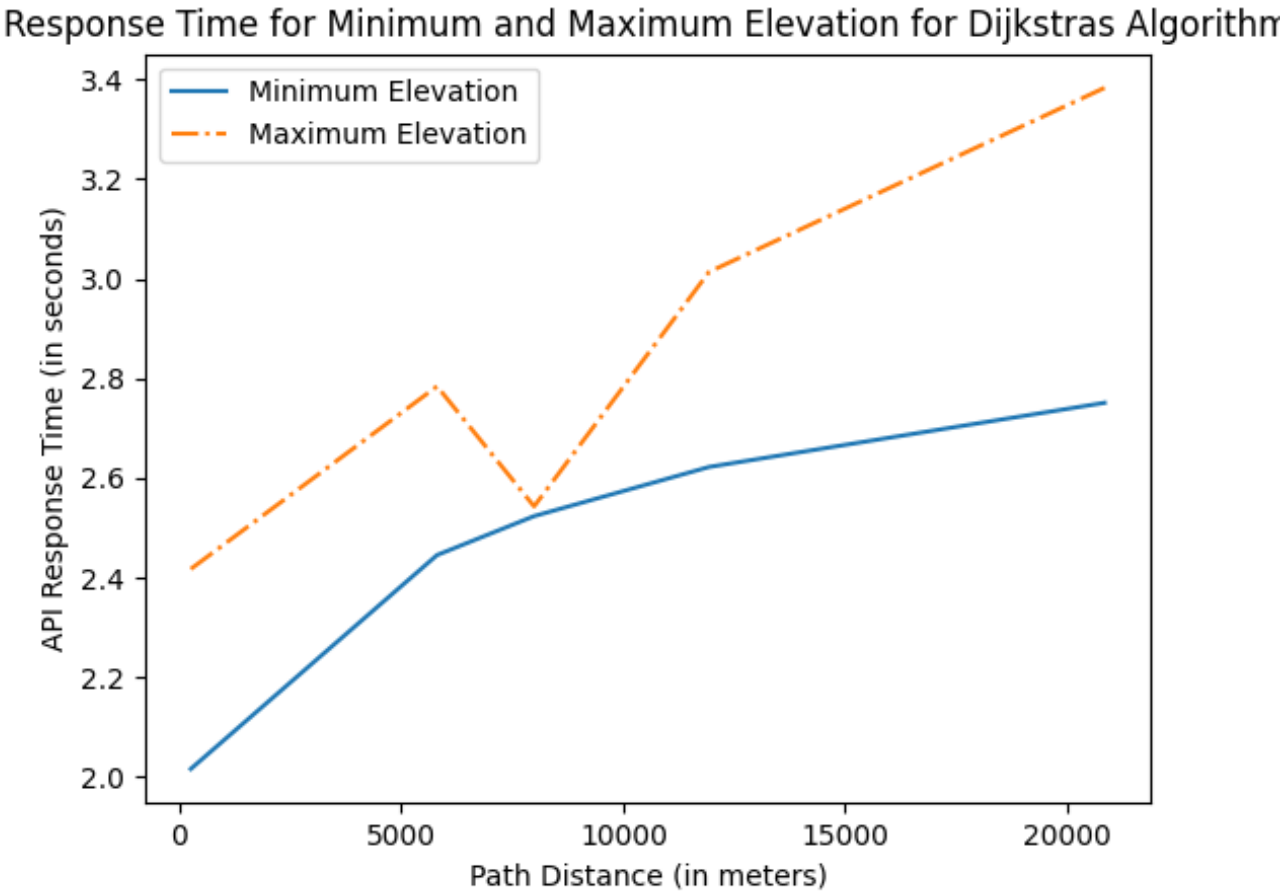
   Following results are obtained :

- We plotted the API response time VS Algorithm graph to evaluate the performance of each algorithm.



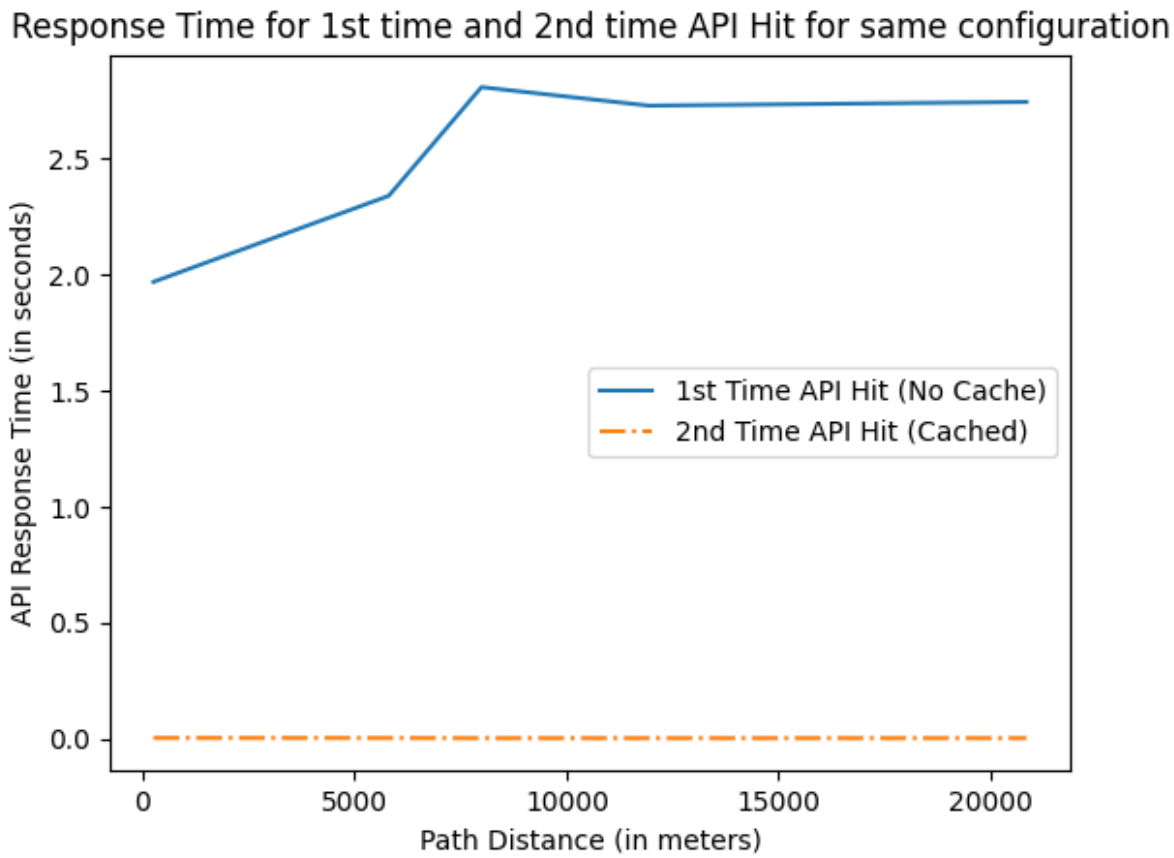
We observed that the Bellman-Ford algorithm takes the highest time to respond with the shortest path as it has more run-time complexity [https://melitadsouza.github.io/pdf/algos.pdf]. Dijkstra, Bidirectional-Dijkstra and A* algorithms take approximately the same time.

- We plotted the API response time VS Path distance graph to evaluate the performance of A* algorithm for different distances for minimum and maximum elevation settings.



We can note that in the case of Maximum Elevation, the API response time is higher. One possible reason for this may be because of the distribution of data. If the majority of data points or regions that we have considered have lower elevations, finding the maximum elevation could involve searching through a larger dataset and performing additional calculations hence having a higher time complexity.

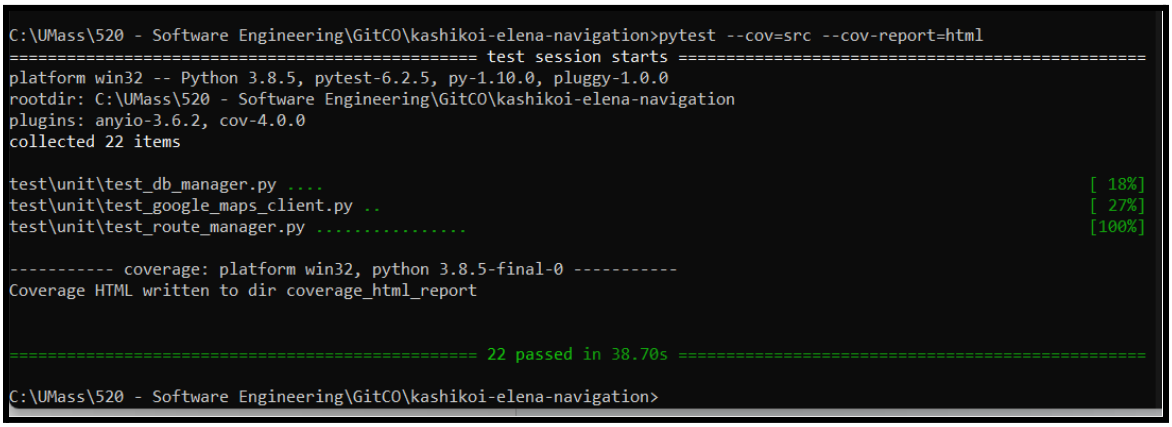
- We plotted the API response time VS Path Distance graph for the case when we first time hit the API and when we second time hit the API from the same set of settings and locations, to observe the effect of the cache mechanism we added in our web-application to improve the response time of the API in case of repeated search.



We can observe in the plot that for the second search the API Response Time is very close to zero seconds.
We have added functionality to save the cache to optimize performance for those scenarios where a user searches for a source and destination location (with the same settings for the other options) which they have entered previously in our application. When they do the search for the second time, the system will be able to reuse the information from the cache and display the optimal route quickly.

4. **Unit Test:** Using pytest and pytest-cov, we have ensured 100% line and branch coverage for our controller and model code. Unit tests are used to test the individual units (methods) in the application. Hence, they are mainly used to test the helper methods and model methods. We have used integration tests to test the API methods. Also, there are certain classes and lines of code that we need to ignore since they are not relevant to the coverage - the app.py file since it is the application startup file which does not need to be tested, the config.py file just loads the config ini to be used throughout the application, main_controller.py file will be tested using integration tests since it is an API, views folder has the UI non pythonic code, routes folder just has the blueprint for flask to call routes in the controller and the __init__.py files are for package initializations and need not be tested. We also ignored certain lines of code in the route_manager.py - the downloading graph file when not cached takes a huge amount of

time because of the size of the graph and ideally would not be needed since our application already has the graphs pre-stored to ensure user has a flawless experience when loading data and the Floyd-Warshall algorithm which is taking a lot of computation power and time because of its nature and is left as a future scope. Our unit tests are extensive and cover all the methods in the controller and model. We have used a hardcoded value of graphs and two nodes from that graph to ensure that the methods are returning what is expected of them. We also test with invalid values to ensure error is thrown in those cases. The sql db manager methods are tested to ensure that insertion and fetching of records are working as expected and in case of invalid data, are throwing data check errors.



5. **Integration/API Test:** We have tested the getroute API which returns information about the optimal path from the server. We have checked a positive case as well as a few illegal cases where our application will not be able to display the optimal route between the source and destination points. A few of these cases are:
    a. Illegal Source Address: The source address must be within a 30 kilometer radius around Amherst. In the case that we enter a source address which is outside this scope, we have tested that the getroute API will return a 400 bad request response status code.
    b. Illegal Destination Address: The destination address must be within a 30 kilometer radius around Amherst. In the case that we enter a destination address which is outside this scope, we have tested that the getroute API will return a 400 bad request response status code.
    c. Null Source Address: The source address cannot be an empty or a null value. In this case, we have tested that the getroute API will return a 400 bad request response status code.
    d. Null Destination Address: The destination address cannot be an empty or a null value. In this case, we have tested that the getroute API will return a 400 bad request response status code.
    e. Illegal Algorithm ID: Our application supports 6 algorithms, namely, Dijkstra's, Bidirectional Dijkstra's, A*, Bellman-Ford, Goldberg-Radzik, and Floyd-Warshall. These correspond to algorithm IDs of 0,1,2,3,4 and 5 respectively. In the case that the application needs to handle algorithm ID less than 0 or greater than 6,

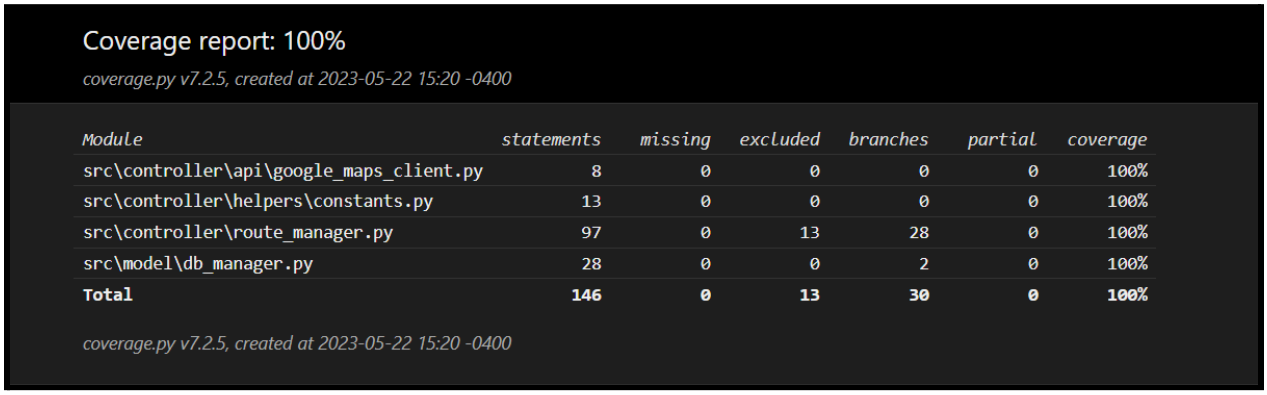
we have tested that the getroute API will return a 400 bad request response status code.

f. Illegal Path Percentage: Our application supports path limit percentage within the range of 100% to 500%. In the case that the application needs to handle a path percentage less than 100 or greater than 500, we have tested that the getroute API will return a 400 bad request response status code.
g. Illegal Transportation Mode: Our application transportation mode of biking and walking, corresponding to 1 and 0 respectively. In the case that the application needs to handle a transportation mode less than 0 or greater than 1, we have tested that the getroute API will return a 400 bad request response status code.

6. **Alpha and Beta Testing:** We have performed extensive alpha testing to test our software internally before releasing it to any external users. This is done to identify any functional or usability issues or any other bugs. The next stage is for beta testing where we will select a group of end-users who could try out our application in their environments and provide feedback. This will further help us understand the software's weaknesses and strengths and make necessary changes as required.

Test Coverage

The test coverage for unit tests was calculated to be 100%.


Coverage report: 100%

coverage.py v7.2.5, created at 2023-05-22 15:20 -0400

Module	statements	missing	excluded	branches	partial	coverage
src\controller\api\google_maps_client.py	8	0	0	0	0	100%
src\controller\helpers\constants.py	13	0	0	0	0	100%
src\controller\route_manager.py	97	0	13	28	0	100%
src\model\db_manager.py	28	0	0	2	0	100%
Total	**146**	**0**	**13**	**30**	**0**	**100%**

coverage.py v7.2.5, created at 2023-05-22 15:20 -0400

We used pytest-cov to calculate our coverage and generate the coverage report. These are the steps we followed :-

1. pip install coverage pytest-cov
2. pytest --cov=src --cov-report=html

The report will be generated in the coverage_html_report folder.