EleNa: Elevation based Navigation

Ajinkya Ghadge, Jaydeep Rao, Shashwat Singh, Shashank Srigiri,
Koushik Reddy Bukkasamudram

November 24, 2020

1 Problem Statement

This project aims to create a software that allows a user to find routes that maximizes or minimizes elevation for desired start and end coordinates in Amherst. The software has to do the minimization/maximization of elevation all while limiting the total distance between start and end locations to a percentage of the shortest path, desired by the user. Another objective is to allow the users to choose the type of algorithm to calculate the path in the manner described above.

2 Use cases and motivation

Applications like Google Maps provide the shortest distance between two points or fastest distance between two points. Although the services provided contain elevation information, due to the niche of its application, elevation based route finding doesn't feature on the popular map applications. There are no web applications that factor in elevation for routing. There are several use cases, where such an application would be desirable.

1. Endurance athletes like marathoners and bicyclists often use altitude training for its peculiar advantages like a holistic training experience, improving blood oxygen saturation and strength training.

2. Recreational sports like skateboarding, hikers find elevations useful. A skateboard enthusiast will leverage the height to have a thrilling skateboarding experience, while a hiker might want to avoid too steep paths.

3. For average users, a hill walk is a more intense workout than a walk with no elevation gain. Depending on the age and enthusiasm, elevation based navigation can be useful for an average user as well.

4. Some other commercial use cases over a large area could be planning paths to create new road and rail networks. Rail engines and coaches due to limitations of traction and power cannot tolerate elevation change of more than 5 percent. Although there are several other factors, this is a key factor, and an elevation based navigation feature would be very useful for rail network planning.

3 Challenges

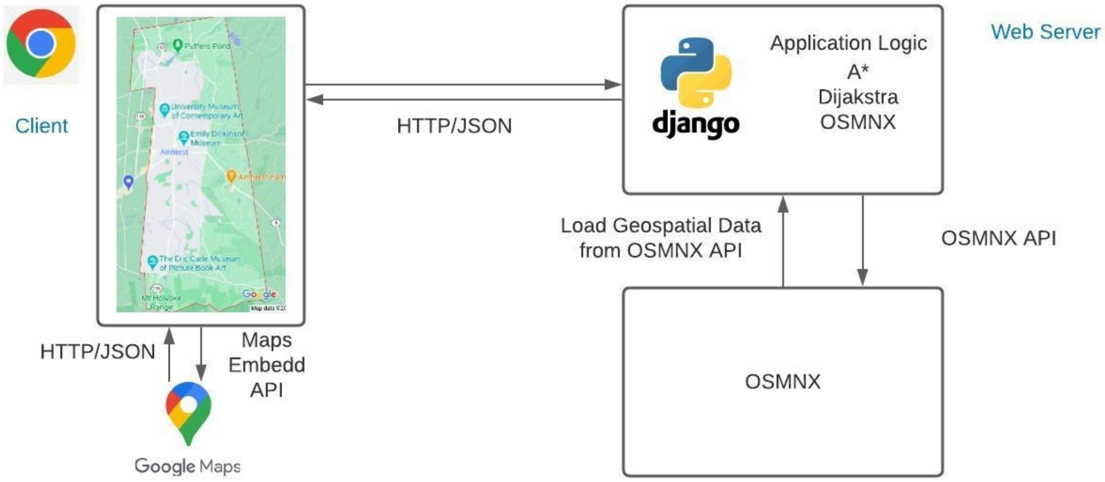
The problem of finding the shortest path between two points, which can be start and end points on a map can be solved in polynomial time by applying Dijkstra's algorithm. In our case, the maximization/minimization of elevation under the constraint of distance makes the desired optimization harder. The optimization is NP-hard and cannot be done in polynomial time. This forces us to use approximation algorithms like A star.

Some other challenges are planning the tasks, developing an optimal development strategy and testing the efficacy of the algorithms used.

4 Architecture

EleNa is designed and developed as a web application. The web application design looks as follows.

Figure 1: Web application architecture



Our implementation of Elena, uses the PAC architecture. PAC stands for presentation abstraction control. The presentation component is the user interface. Abstraction component is the graph with elevations that is loaded for the desired area, which in our case is Amherst. Controller is the set of classes that contain the business logic. In our case the implementation of algorithms and the routing strategies. The abstraction and controller components are contained in the folder, backend in the source code. The presentation component is present in the front end folder. Test cases for the backend are present in the backend folder.

4.1 Front-End

The technology used for front-end is AngularJS and typescript. The design used is material design by google and node-js is used as the package manager. AngularJS is a very stable technology that allows for rapid-testing and has a lot of dependencies built into itself. Additionally, community support as well as unit testing features make it a very reliable platform.

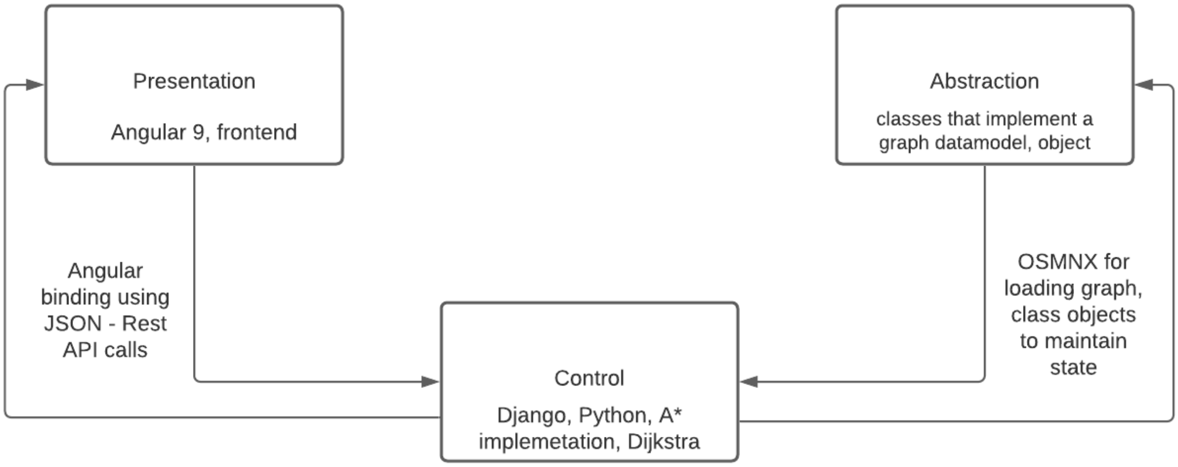
To make the map interactive, we have used Typescript. Like Javascript, Typescript allows interactive code to run on the browser side. For loading the map of Amherst, which is our targeted location, we center the map around Amherst and choose optimal zoom level. Markers are used to obtain the coordinates, which are in turn processed by the backend.

4.2 Back-End

Django is used as the backend. We also make use of OSMNX and Google Maps API. Django, unlike other frameworks like Flask, contains a lot of features, that could be beneficial to scale the application. Django supports MVC as well as URL routing. Django like Flask framework allows for rapid prototyping for web-applications. In addition, it is very well tested, making it a great choice to think of development not just from good design practice, but stability, extensibility and scalability.

OSMnx is a Python package for downloading spatial geometries for modelling, projecting, visualizing, and analyzing streets and routes from OpenStreetMap's APIs. Users can just as easily download and work with landmarks, elevation data, street bearings, travel time, and network routing. OSMnx is very useful for getting elevation information as well as for visualization of the path.

Figure 2: PAC architecture of web application


Presentation
Angular 9, frontend
Abstraction
classes that implement a graph datamodel, object
Angular binding using JSON - Rest API calls
OSMNX for loading graph, class objects to maintain state
Control
Django, Python, A* implemetation, Dijkstra

5 Application flow, Data Model and Algorithms

5.1 Application flow

For the current version of our EleNa implementation, the application flow is qite straightforward. The user on reaching the home page, has to select minimization or maximization option for their desired start and end points.

5.2 Data model

The data model, or the abstraction model for the PAC architecture mentioned above can be explained better with the class diagrams.

1. Nodes and edges are used to represent the graph. The objects formed using these classes can be analyzed or used as the model to query data from memory to obtain the results. This data model encapsulates all the information that is needed by the routing algorithms.

2. For retriving data, we have used OSMNX. OSMNX provides an interface to query OpenStreetMap's API. This package is useful in loading the data initially into our data model.

5.3 Algorithms

Algorithms are used by our application:

1. Dijkstra's algorithm

2. A star algorithm

The algorithms can be briefly described as follows:

1. Dijkstra's algorithm: The application uses Dijkstra's algorithm to calculate the shortest path without optimizing for elevation. The start position of the user is the first node we start with. From this node, the algorithm will calculate the shortest path by considering distance between adjacent nodes as edge weight. The algorithm at each node, starts with the nodes that are closest. It would only stop when the shortest path is calculated or if there is no path between source and destination or if the destination node has the smallest current weight among all non visited nodes.

2. A* algorithm: The application uses this algorithm to calculate the shortest path with optimization for elevation. While Dijkstra's algorithm uses edge weights to optimize, A* leverages the underlying information that our nodes are geographically related. Euclidean distance between the start and end points allows for faster calculation of shortest path as some of the non-path node visits can be avoided. To optimize for elevation, the elevation gain is also multiplied.

Figure 3: Application flow chart

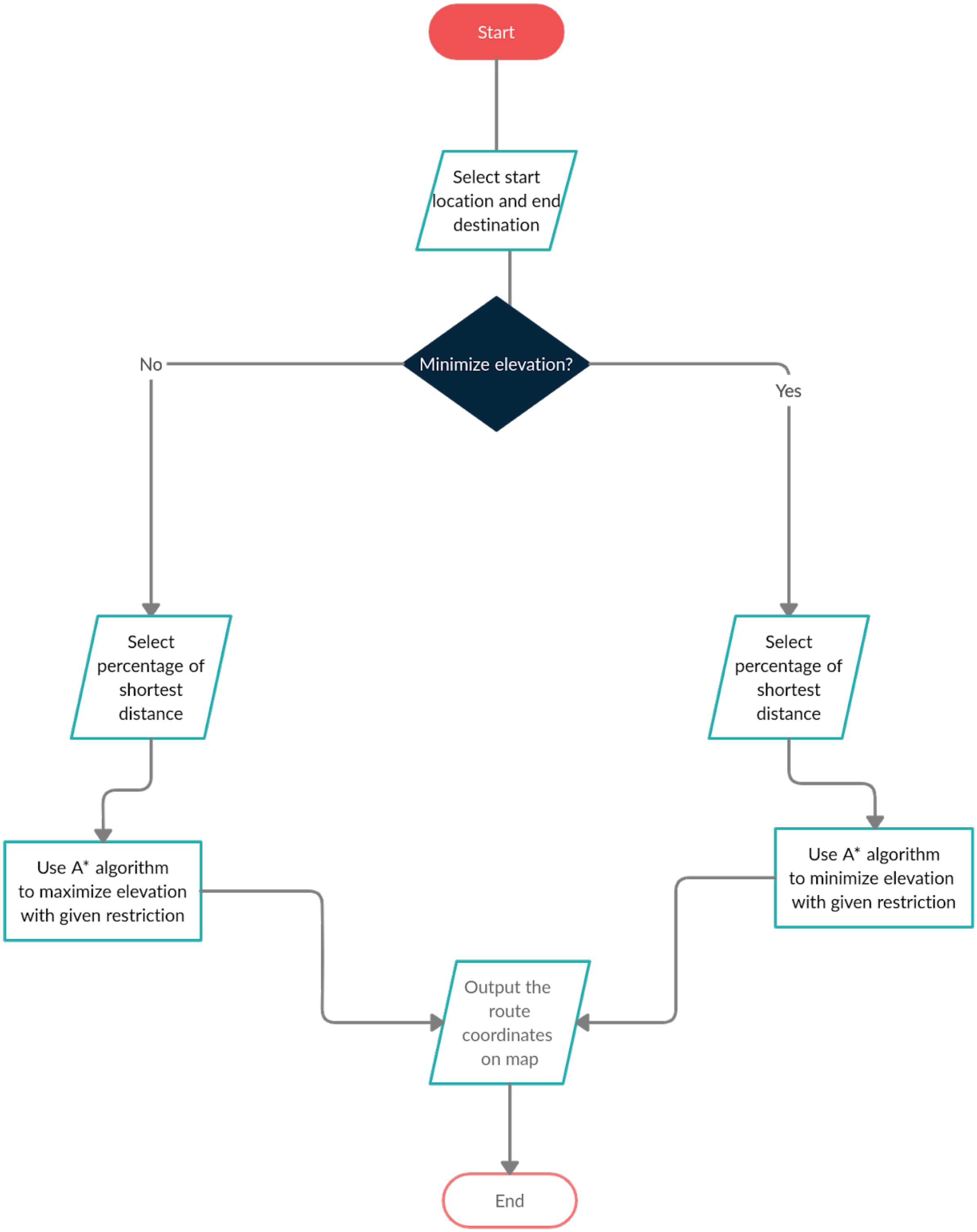
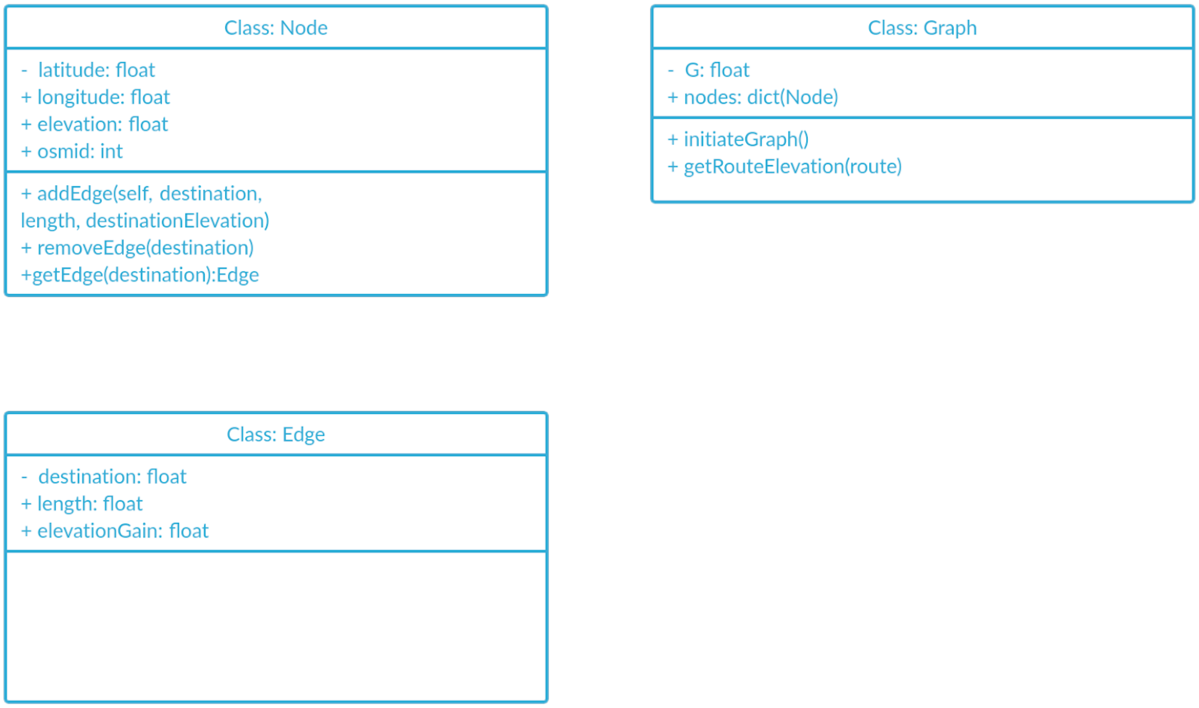

Start
Select start location and end destination
Minimize elevation?
No
Yes
Select percentage of shortest distance
Select percentage of shortest distance
Use A* algorithm to maximize elevation with given restriction
Use A* algorithm to minimize elevation with given restriction
Output the route coordinates on map
End

Figure 4: class diagram



6 Testing

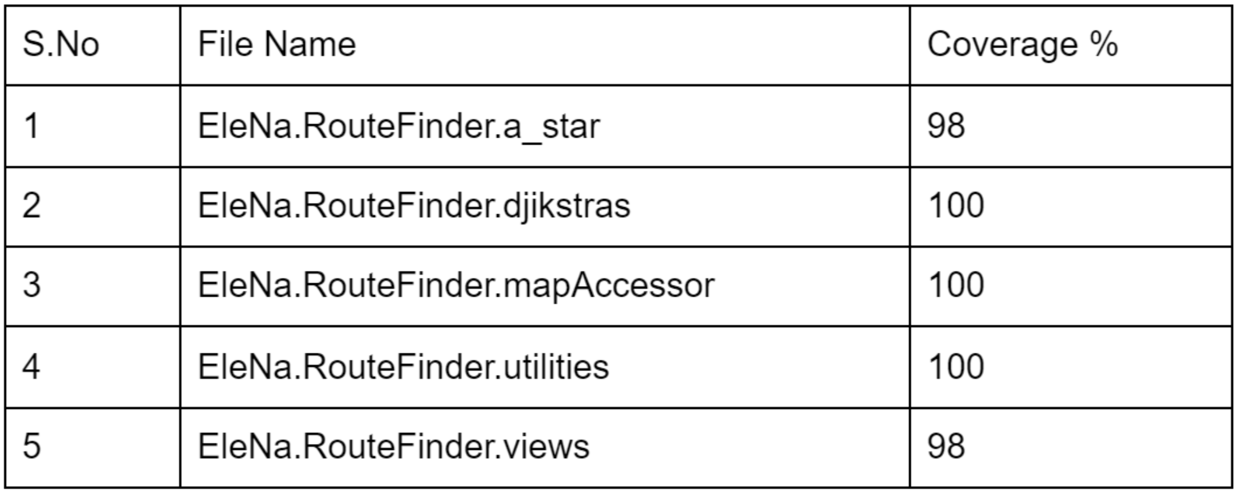
Our goal with testing was to achieve maximum test coverage and have a good testing framework. In our exhaustive testing plan, we have used functional testing and regression testing to test the application. We choose the unittest framework for the consistent xUnit style of unit testing. Nose is a tool that we leverage for unit testing our application whereas coverage is a plugin which helps in visualizing test coverage and provides some functionality to debug issues. Using our testing strategy we have achieved an accuracy of over 98

6.1 Functional testing

We have used functional testing to test each function of our app model and controller. By applying appropriate assertions we are able to verify the functional logic of our code. Each functionality is tested by this methodology, giving very good coverage as well.

Please note that this coverage report takes into account different configuration files such as settings.py, urls.py, admin.py, models.py which does not have any core functional logic embedded in them. So, here are the statistics of coverage for the main logic files.
In the above table, we specify the core functional logic files, which we need to consider for the unit testing, and we specify the amount of coverage in that particular file.To run the unit test cases and also to generate the test coverage report, you need to install nose and coverage using pip or conda.

Figure 5: coverage table



S.No	File Name	Coverage %
1	EleNa.RouteFinder.a_star	98
2	EleNa.RouteFinder.djikstras	100
3	EleNa.RouteFinder.mapAccessor	100
4	EleNa.RouteFinder.utilities	100
5	EleNa.RouteFinder.views	98

Using pip

1. pip install django-nose
2. pip install coverage==3.6

After installing the above dependencies, you can run coverage run manage.py test to get the test coverage report. You can optionally pass in parameters to the above statement in a terminal or in the settings.py file to generate the coverage report in a modified way according to our needs.

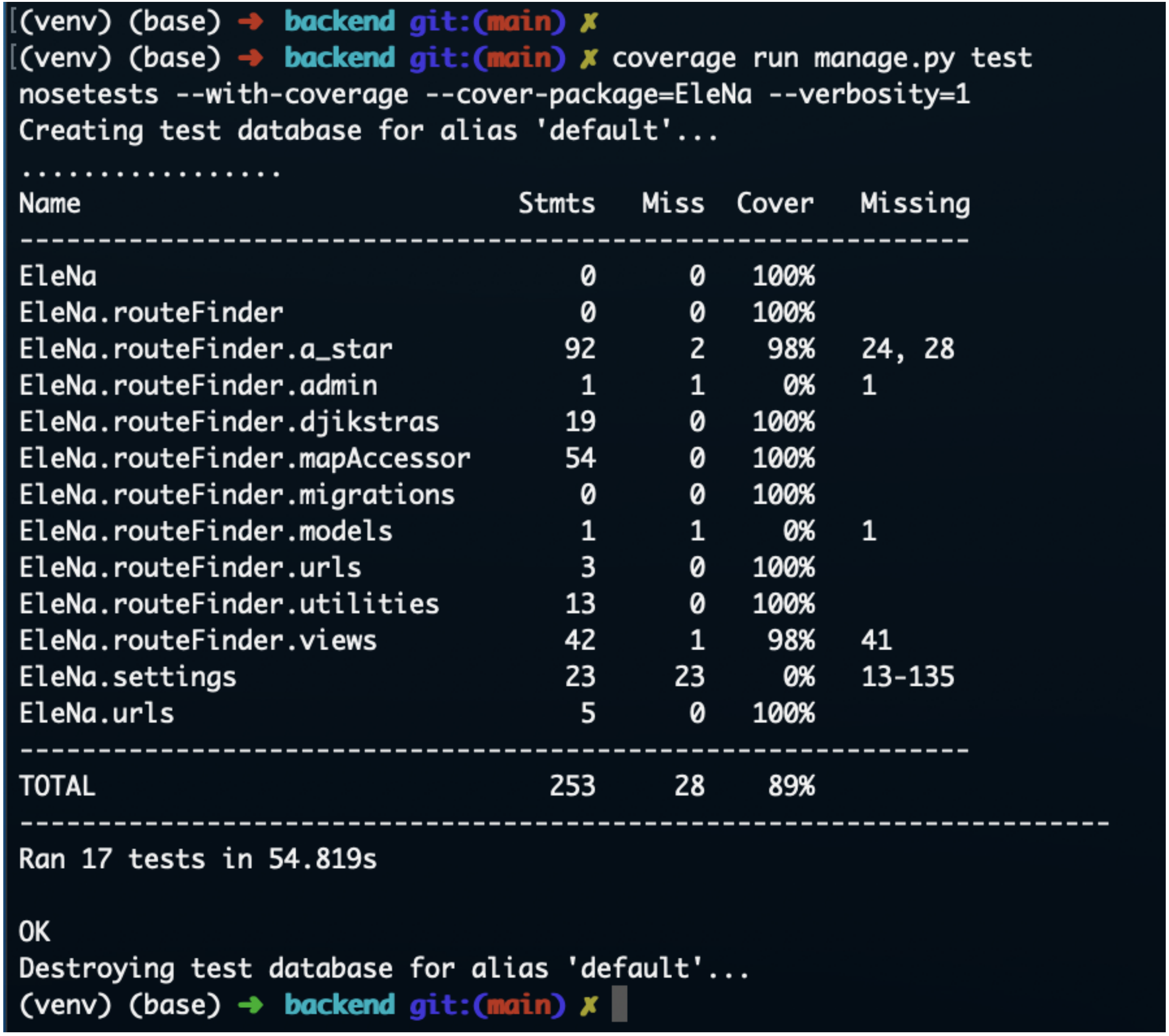
6.2 Regression testing

We implement regression testing to ensure that new feature additions and bug fixes don't break the existing features. Two features that we have tested are loading of the home page and end-to-end obtaining route for our given starting and end-points as our regression test. This is adequate for the scope of the project as these two execution flows cover the entire control flow of the application.

6.3 Testing Algorithm

One of the ways to test the algorithm, their maximization and minimization was to compare the maximization, minimization and standard output. The expected behavior is that for any two points, the algorithm should output more elevation for maximization, lesser for standard elevation and least for minimization. The bounds would be such that, in worst case the three can be at same level, but our algorithms would fail if maximized elevation is lower than standard or minimized elevation. We took a random sample of 40 data points, and using a test script, plotted the elevations. The comparison shows that our algorithms do show desired behavior.

Figure 6: Testing screenshot



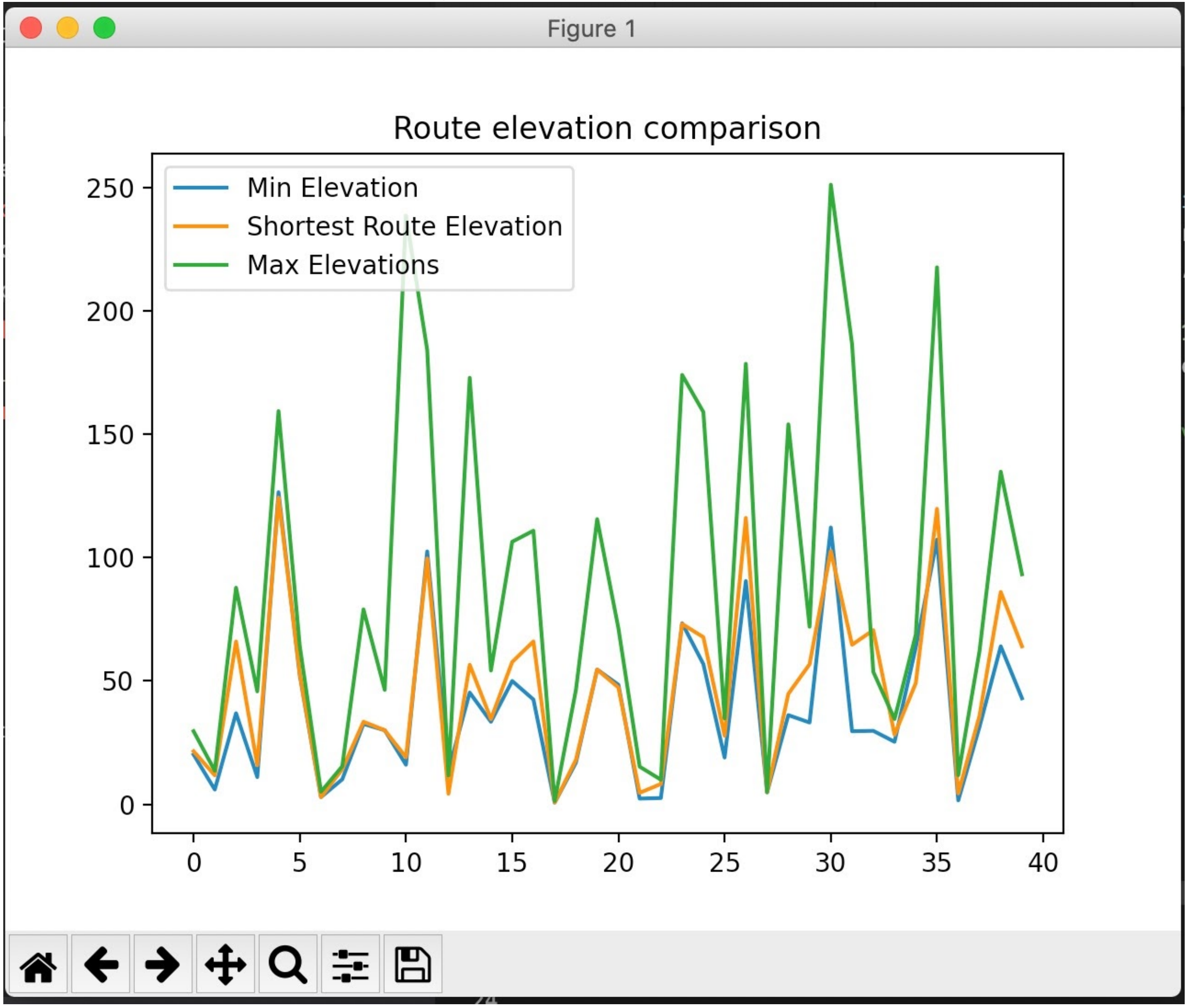
```
[(venv) (base) ➜ backend git:(main) ✗
[(venv) (base) ➜ backend git:(main) ✗ coverage run manage.py test
nosetests --with-coverage --cover-package=EleNa --verbosity=1
Creating test database for alias 'default'...
.................
Name                              Stmts   Miss  Cover   Missing
---------------------------------------------------------------
EleNa                                 0      0   100%
EleNa.routeFinder                     0      0   100%
EleNa.routeFinder.a_star             92      2    98%   24, 28
EleNa.routeFinder.admin               1      1     0%   1
EleNa.routeFinder.djikstras          19      0   100%
EleNa.routeFinder.mapAccessor        54      0   100%
EleNa.routeFinder.migrations          0      0   100%
EleNa.routeFinder.models              1      1     0%   1
EleNa.routeFinder.urls                3      0   100%
EleNa.routeFinder.utilities          13      0   100%
EleNa.routeFinder.views              42      1    98%   41
EleNa.settings                       23     23     0%   13-135
EleNa.urls                            5      0   100%
---------------------------------------------------------------
TOTAL                               253     28    89%
----------------------------------------------------------------------
Ran 17 tests in 54.819s

OK
Destroying test database for alias 'default'...
(venv) (base) ➜ backend git:(main) ✗
```

7 Future Work

We want to build multiple versions of this application, separate versions for average users and premium users. Premium users will get more accurate results by use of more sophisticated algorithms that use more compute power. Premium users will also have personalized advantages like age, height, weight and medical conditions, goals. To implement this we need to have a data model for users as well. We also want users to be able to track calorie expenditure and integrate that with Google Fit API.

Another thing we want to do is, build a more scalable web application. For that we want to use a back-end that uses asynchronous communication using message queues for high throughput of HTTP requests. Along with message queues, we would also want to have multiple workers on the server side instead of a single server, so that fault tolerance can also be built for scalability. This will be especially useful when we use it with the above

Figure 7: Testing algorithms



mentioned features.

Furthermore, we would want to switch to a test driven development methodology so as to ensure better quality of the product and maintain the coverage as the size and functionality of our application grows.

8 Acknowledgement

We thank professor Heather Conboy for helping us with the fundamentals, feedback and direction for this project. Inputs on how to prototype, choosing the area, what to expect from algorithms were particularly useful. Lastly, we really appreciate the valuable feedback provided by the graders during the previous presentation as well.

9 References

1. https://benalexkeen.com/implementing-djikstras-shortest-path-algorithm-with-python/
2. https://www.educative.io/edpresso/what-is-the-a-star-algorithm
3. https://www.youtube.com/watch?v=6TsL96NAZCot=629s
4. https://www.youtube.com/watch?v=GazC3A4OQTEt=329s
5. https://github.com/networkx/networkx/blob/master/networkx/algorithms/shortest_paths/astar.py
6. https://docs.djangoproject.com/en/3.1/topics/testing/overview/
7. https://stackoverflow.com/questions/62137862/what-are-the-units-of-measure-dimensions-of-given-edge-attributes-in-osmnx
8. https://angular-maps.com/api-docs/agm-core/components/agmmap
9. https://angular-maps.com/api-docs/agm-core/directives/agmmarker
10. https://material.angular.io/
11. https://angular.io/guide/two-way-binding
12. https://realpython.com/python-testing/
13. https://python.readthedocs.io/en/stable/library/unittest.html?highlight=re