

# CS520 No-Op Group

## Final Report

Members: Jai Puro, Pragya Sarda, Saksham Kumar, Nidhi Chandra

Github usernames: zaiiii, ms-sarda (or Pragya Sarda)<sup>1</sup>, sak2002, n1dhiumass(or Nidhi chandra)

<b>Software Requirement Specification.....</b>	<b>1</b>
Problem Statement.....	1
Overview and Design Requirements.....	1
List of features.....	1
Stakeholders and User-stories.....	1
Functional requirements.....	2
Non-functional requirements.....	2
<b>Design Document.....</b>	<b>3</b>
Overview.....	3
Architecture - High Level Design.....	3
Tech stack and Low Level Details.....	4
Front End - React.....	4
Back End - Python and Flask server.....	4
Database.....	4
Libraries.....	5
Code.....	5
User-Interface.....	6
<b>Features.....</b>	<b>7</b>
Limitations.....	7
Future Work.....	7
References.....	8
<b>Evaluation.....</b>	<b>9</b>
Verifying the functional requirements.....	9
Verifying the non-functional requirements.....	12
<b>Execution Plan and Contributions.....</b>	<b>13</b>

---

<sup>1</sup> Because of the wrong git config setting Pragya and Nidhi have commits from two user names.

# Software Requirement Specification

## Problem Statement

Navigation systems optimize for the shortest or fastest route. However, they do not consider elevation gain. Let's say you are hiking or biking from one location to another. You may want to literally go the extra mile if that saves you a couple thousand feet in elevation gain. Likewise, you may want to maximize elevation gain if you are looking for an intense yet time-constrained workout.

The high-level goal of this project is to develop a software system that determines, given a start and an end location, a route that maximizes or minimizes elevation gain, while limiting the total distance between the two locations to x% of the shortest path.

## Overview and Design Requirements

The aim is to build a web application that takes as input the source and destination along with the user's choice of whether they wish to maximize or minimize elevation gains and the limit on the percentage of the shortest route. It should also take in the kind of transport a user wishes to take - Walk, Bike or Drive and show the difference between the new calculated route satisfying the constraints and the shortest route.

The technical details of the project are discussed in the design document.

### List of features

- Input fields with dropdowns (autocomplete using Google's Location API) for entering the source and destination
- Radio buttons to select min/max elevation gains
- Slider to enter the limit on percentage of the shortest route
- Two maps to showcase the shortest route and the new calculated route
- Result window showing the total calculated distance and elevation of the route
- Radio buttons to select transport - Walk, Bike or Drive

## Stakeholders and User-stories

### Stakeholders

- Runners/Joggers
- Hikers
- Bikers/Cyclists
- Pedestrians

### User Stories

- As a user, I want to be able to enter my source and destination locations
- As a user, I want to be able to see a path from my source to my destination
- As a user, I want to be able to set the elevation gain to maximum or minimum
- As a user, I want to limit the length of the path to a certain fraction of the shortest path

- As a user, I want to be able to see what my shortest path looks like for comparison
- As a user, I want to see what the total distance between my entered locations is
- As a user, I want to see the total elevation between my entered locations is
- As a user, I want to be able to select the kind of transport - Walk, Bike or Drive I want to use

## Functional requirements

The software system should provide a customized and optimized route for hiking, biking or Drive based on the user's preferences (maximize or minimize gain and limit distance till what percentage of the shortest route) and the constraints of the start and end locations. We use Dijkstra and modified version of Dijkstra to find shortest path routes and the shortest path satisfying the user constraints resp.

- **Input:** The system, using the UI, should allow users to input the following information:
  - Start location and End location for their route. Provide suggestions and autocomplete for locations using maps API.
  - The percentage limiting the new route distance from the shortest path.
  - Whether to minimize or maximize the elevation gain.
  - Type of transport - Drive/bike/walk.
- **Output:**
  - The optimized route map based on the user's preference for maximizing or minimizing elevation gain.
  - The shortest route map based on the location system being used (eg: Google Maps) for user's comparison.

## Non-functional requirements

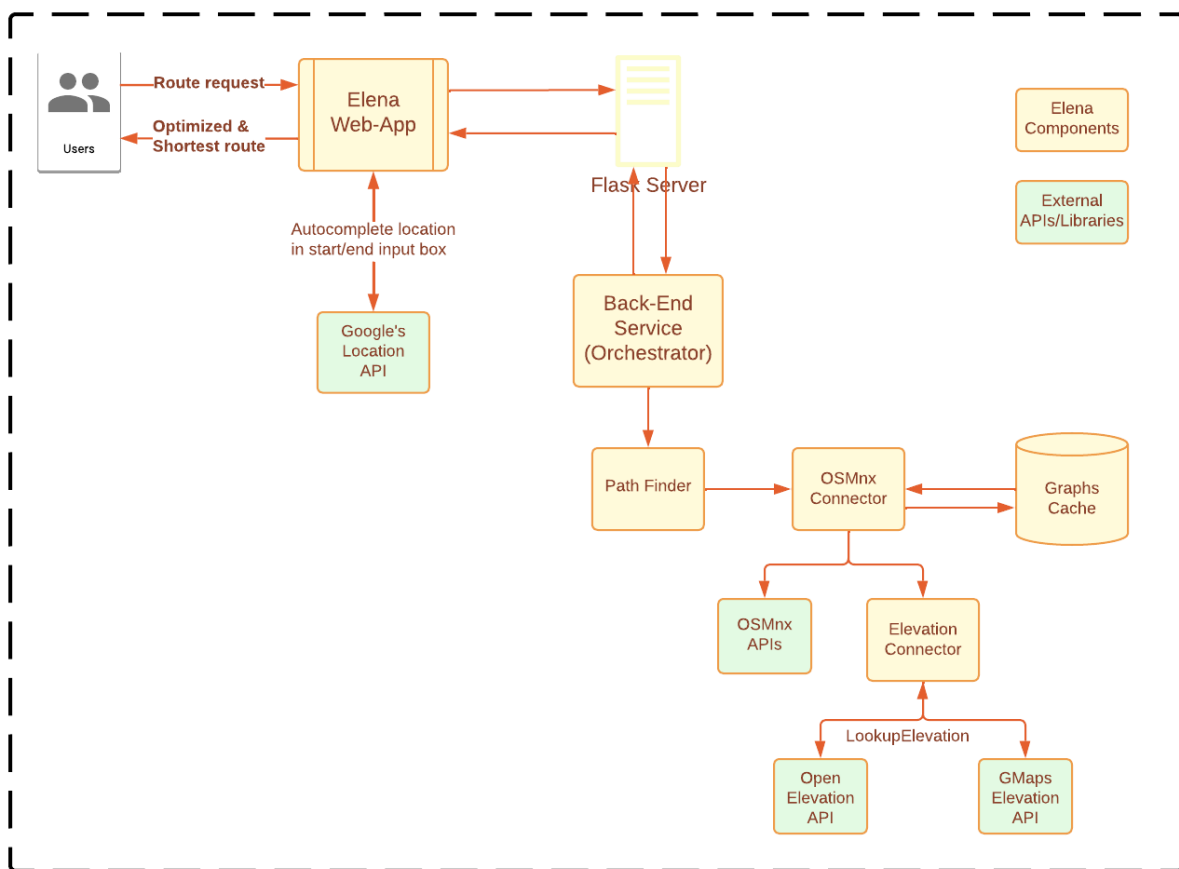
- **Testability** - For developers, covers all edge cases and all integration tests. Have a test plan which is thoroughly executed. 90% line and branch coverage.
- **Readability** and **Documentation** - For developers, easy to manage and maintain. Proper documentation within the code.
- **Usability** - Good and easy to use UI, intuitive
- **Reliability** and **Correctness** - Outputs correct route with the given requirements
- **Performance** - The algorithm should be quick in producing results. The user should not have to wait a long time before getting the result (optimized path) after querying. Exact SLA is not yet decided.
- **Modularity** - Write code in a manageable, modular way which makes it easy to read and extend code. Make multiple layers (front-end, back-end, database) and expose only the required interfaces at each layer.

# Design Document

## Overview

The backend is built using Python, while the frontend is built using React. The website is hosted using a Flask server. We use the OSMnx library to get a city map and various other geo-location data like lat-long of an address. OSMnx in turn uses OpenStreetMap data. We also use the Open Elevation API to determine the elevation at a certain point. However, since Open-Elevation is unreliable (we've had to face a lot of timeouts while developing the project) we use Google Maps api to get the elevation data if Open-Elevation does not work.

## Architecture - High Level Design



**Client Web-app:** It is the client facing part of the application. It exposes the form that the user needs to fill to get an optimized route map. It allows users to enter the start and end location, what kind of elevation gain does he/she want and how much of it can deviate from the shortest route possible. It also allows the user to choose what kind of transport he/she will be using. The optimized route is rendered on this web app along with the shortest route possible (without any constraints) for comparison. We use Google's location API to provide auto-completion for the

location fields. It is also responsible for generating the map for both shortest route and new route and then rendering it.

**Back-end Service:** The backend service is the main brain of the application. It takes the user input from the UI, requests data from the various libraries such as OSMnx, Open-Elevation and Google Maps to create a route between the required locations with the specified constraints.

We use Dijkstra to find the shortest path and use modified Dijkstra to find the optimized shortest path for the user-specified constraints of distance, elevation and type of transport. Both the algorithms have been implemented by us from scratch.

The code is divided into various subcomponents to ensure modularity like connectors to OSMnx, Google, Open-Elevation, Graph creation utils, Orchestrator to bind all of them, etc. The data for a given request will be stored in in-memory objects. This includes request and response from all APIs required to compute the new path. We also support nearest node search and finding paths for addresses that are not exact nodes in the map.

**Database:** We store the graph loaded for a city with the elevation details on all city nodes since this is a time consuming process and may also cost us money if Open-Elevation is not responding. The elevation API from Google is paid.

## Tech stack and Low Level Details

### Front End - React

React is a popular **JavaScript** library used for building user interfaces. We have chosen React for its declarative programming approach and component-based architecture. Another motivation for choosing React was the fact that React allows partial updates to an interface via the use of a Virtual DOM. This allows us to only efficiently update the parts of the UI that change (i.e. the maps) without having to update the entire page.

### Back End - Python and Flask server

Our choice of a Flask server for our backend was inspired by the fact that it is very flexible and light-weight. It is also incredibly easy to scale up a Flask server for any future applications. Apart from this, it is a popular choice for backend servers which means that it has a large community of developers. Lastly, the team members found that the Flask API was simple enough and easy for us to pick up.

Our main algorithm for finding the best route and other backend services will be written in Python.

### Database

We use a simple file system to store the graph files created. We store files in graphml format which is the default format used for OSMnx graphs.

## Libraries

### OSMnx

OSMnx is a Python library that allows users to download geospatial data from OpenStreetMap. Given that we are working with Flask on the backend it is easy to integrate this library into our application. We use OSMnx to download the map of a city in which the start and end destinations are present. We also use it to find the latitude and longitude of various nodes and addresses in a city.

### Open-Elevation

This is an open source API to get elevation of a data point or a set of data points. We use this to get the elevation of all the nodes in the city.

### Google Maps

Since Open-Elevation API is free and open-source it allows limited usage. It also frequently times out. It only supports 150-180 nodes per request and since some cities may have over 4000 nodes, Open Elevation API may take a long amount of time or may take time-out. We therefore use Google Maps as a backup to allow us to run the application without any hiccups.

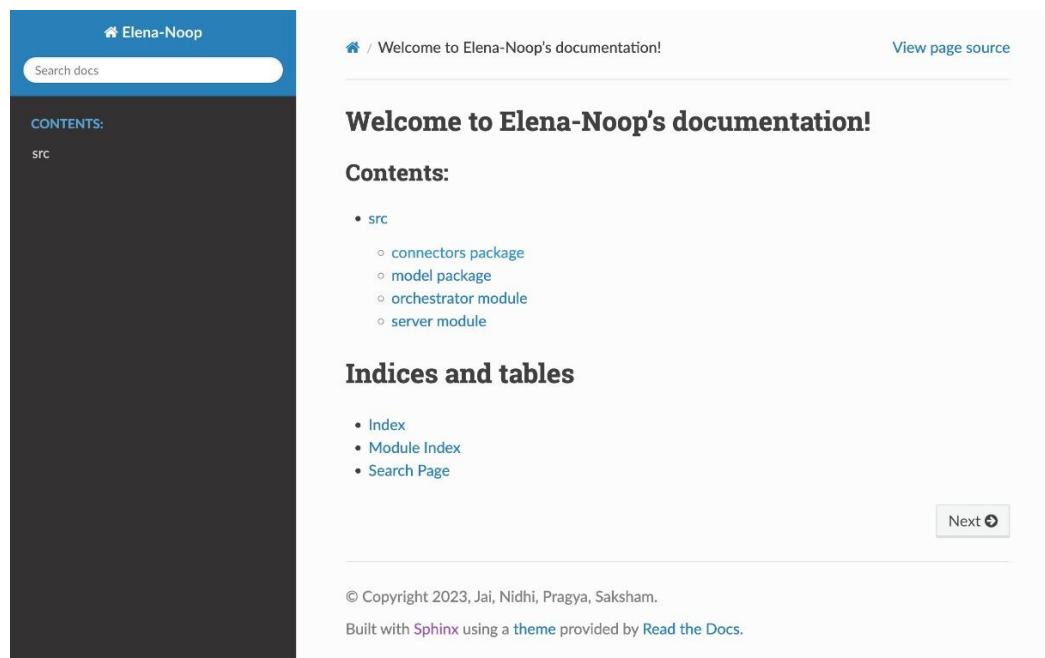
## Code

Github Link to the repo: <https://github.com/ms-sarda/noop-elena>

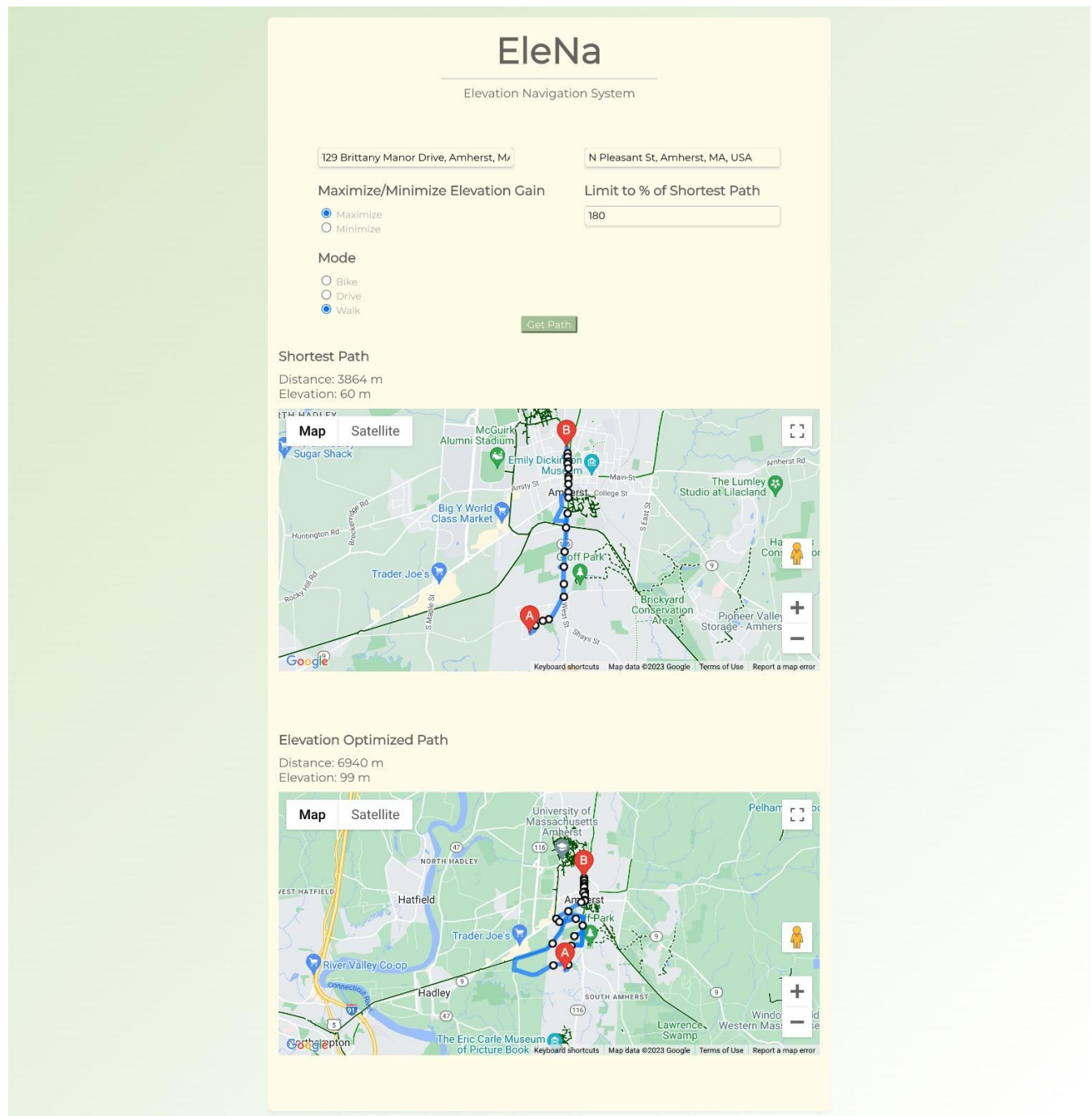
API documentation for developers: <https://github.com/ms-sarda/noop-elena/tree/main/api-doc>

To access the api documentation run:

```
cd api-doc/  
open index.html
```



## User-Interface



## Features

1. We support auto-completion in our UI for which we are using Google Maps API.
2. We render both graphs - shortest and the new optimized graph based user constraints. This allows users to compare and see the difference between the two routes and choose what fits best.
3. The user can choose what kind of transport does s/he want to use for the path - i.e. walk, bike or drive. The graph is generated accordingly.
4. We are caching the graph generated for a city. This allows us to have better performance since generating graphs takes a longer time.
5. Our UI is easy to interact with. It provides radio buttons instead of text fields for users to easily enter the required information. As mentioned in point 1, it also allows users to auto-complete the address for source and destination.
6. We have written code for using both Open-Elevation and Google Maps. A user can select whether to use Google or Open Elevation to get elevation data.

## Limitations

1. The start and end destination need to be within a city. Since OSMnx is an open-source library it is a bit slow. It takes a good enough time to generate a city map. To be able to generate a state map or more will take considerably longer. Therefore, as of now we only take the routes within one city.
2. The algorithm does not generate the best possible path rather it generates one of the possible paths. The problem to generate the best possible path seems to be an NP-Hard problem. We are not sure if there was a better solution using BFS or DFS to solve this problem. Since we had already spent a considerable amount of time implementing the current algorithms we did not get to write A\* which could have generated a better path.
3. The user needs to enter an exact address which is in the form of street name, city, state, country for us to be able to determine a route between the start and end points.

## Future Work

1. Implementing A\* algorithm for finding the optimized path with the specified constraints and then choosing the best path from A\* and Dijkstra.
2. Implementing a multi-threaded server for better performance.
3. If possible, gather data for hiking trails and then provide elevation insights and different paths for trek and trail routes.
4. Have a more interactive UI and better Maps embedding.
5. Cache results of each request for a better performance.



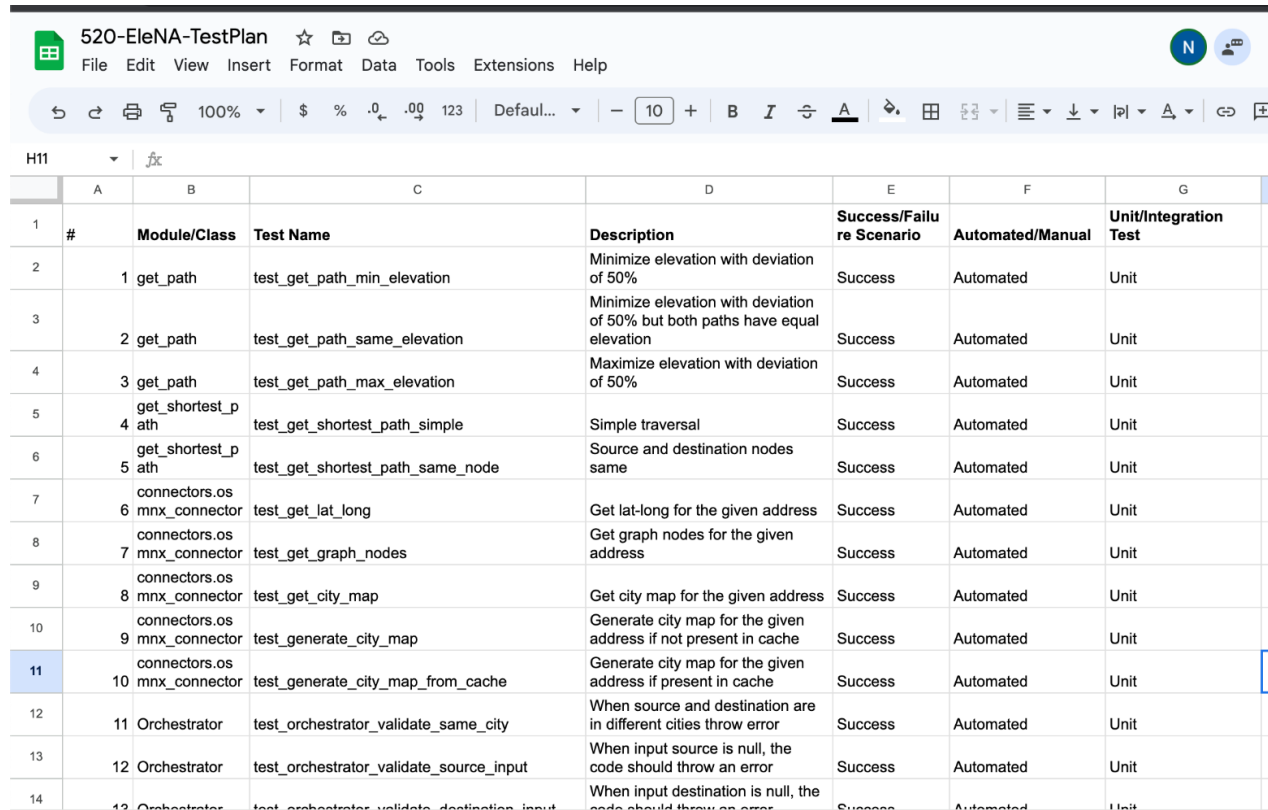
## References

1. Open Elevation - <https://open-elevation.com/>
2. OpenStreetMap - [https://wiki.openstreetmap.org/wiki/Main\\_Page](https://wiki.openstreetmap.org/wiki/Main_Page)
3. OSMnx - <https://osmnx.readthedocs.io/en/stable/>
4. Google Maps Platform - [https://developers.google.com/maps/documentation?\\_gl=1\\*z5osxq\\*\\_ga\\*MTgyNTcyODkwMS4xNjgxOTM0MTA0\\*\\_ga\\_NRWSTWS78N\\*MTY4MTkzNDEwNC4xLjEuMTY4MTkzNDZy4wLjAuMA..](https://developers.google.com/maps/documentation?_gl=1*z5osxq*_ga*MTgyNTcyODkwMS4xNjgxOTM0MTA0*_ga_NRWSTWS78N*MTY4MTkzNDEwNC4xLjEuMTY4MTkzNDZy4wLjAuMA..)
5. Flask - <https://flask.palletsprojects.com/en/2.2.x/>
6. React - <https://react.dev/>

# Evaluation

## Verifying the functional requirements

We have a [test plan document](#) consisting of test cases (including edge cases) for each functionality (per class/function) as well as all integration tests to ensure that the system is working as expected.



#	Module/Class	Test Name	Description	Success/Failure Scenario	Automated/Manual	Unit/Integration Test
1	get_path	test_get_path_min_elevation	Minimize elevation with deviation of 50%	Success	Automated	Unit
2	get_path	test_get_path_same_elevation	Minimize elevation with deviation of 50% but both paths have equal elevation	Success	Automated	Unit
3	get_path	test_get_path_max_elevation	Maximize elevation with deviation of 50%	Success	Automated	Unit
4	get_shortest_path	test_get_shortest_path_simple	Simple traversal	Success	Automated	Unit
5	get_shortest_path	test_get_shortest_path_same_node	Source and destination nodes same	Success	Automated	Unit
6	connectors.os	test_get_lat_long	Get lat-long for the given address	Success	Automated	Unit
7	connectors.os	test_get_graph_nodes	Get graph nodes for the given address	Success	Automated	Unit
8	connectors.os	test_get_city_map	Get city map for the given address	Success	Automated	Unit
9	connectors.os	test_generate_city_map	Generate city map for the given address if not present in cache	Success	Automated	Unit
10	connectors.os	test_generate_city_map_from_cache	Generate city map for the given address if present in cache	Success	Automated	Unit
11	Orchestrator	test_orchestrator_validate_same_city	When source and destination are in different cities throw error	Success	Automated	Unit
12	Orchestrator	test_orchestrator_validate_source_input	When input source is null, the code should throw an error	Success	Automated	Unit
13	Orchestrator	test_orchestrator_validate_destination_input	When input destination is null, the code should throw an error	Success	Automated	Unit

- Unit Tests - We have a line and branch coverage of more than 85% which ensures that all functionality is individually tested. We will also cover all edge cases as a part of this testing.

```
(elena) nidhichandra@Nidhis-MacBook-Air: ~ % pytest --cov=src src/tests
test session starts
platform darwin -- Python 3.9.6, pytest-7.3.1, pluggy-1.0.0
rootdir: /Users/nidhichandra/Developer/cs520/noop-elena
plugins: cov-4.0.0
collected 17 items

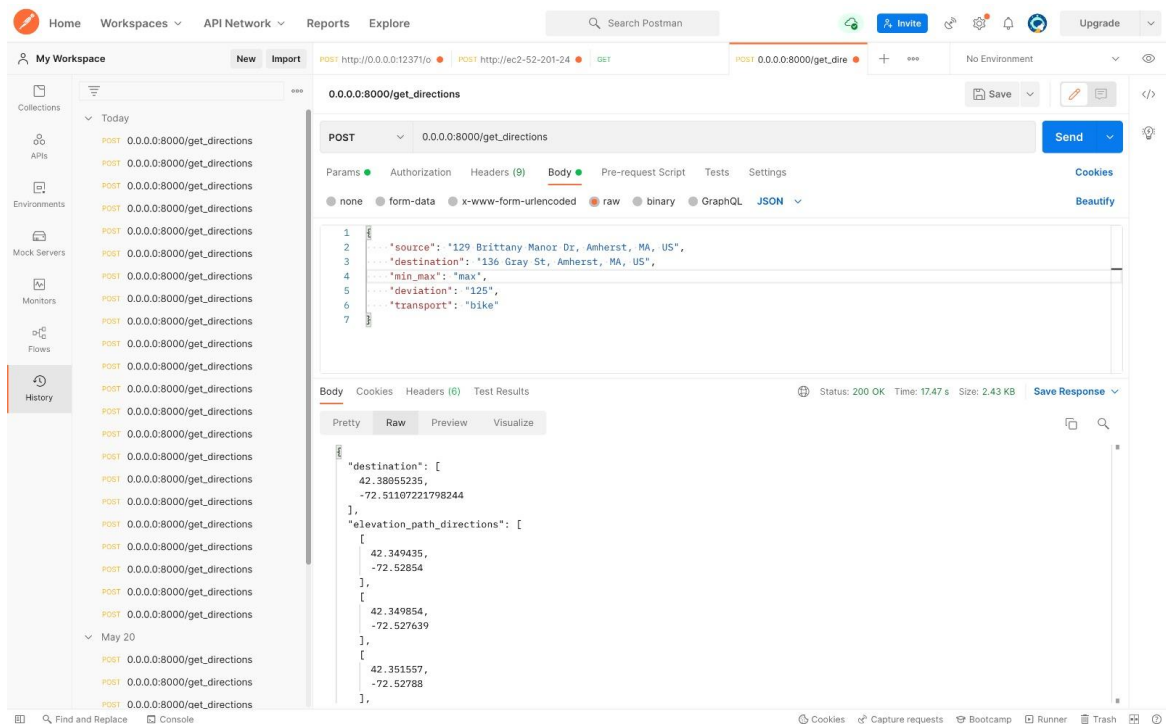
src/tests/test_get_path.py ...
src/tests/test_get_shortest_path.py ..
src/tests/test_orchestrator.py .....
src/tests/test_osmnx_connector.py .....

----- coverage: platform darwin, python 3.9.6-final-0 -----
Name                               Stmts   Miss  Cover
-----
src/connectors/__init__.py           0      0  100%
src/connectors/elevation_connector.py 21     11   48%
src/connectors/osmnx_connector.py    52     20   62%
src/connectors/utlis.py               9      2   78%
src/model/__init__.py                0      0  100%
src/model/path_finder.py            103     0  100%
src/orchestrator.py                  43     0  100%
src/server.py                        26     26    0%
src/tests/__init__.py                0      0  100%
src/tests/test_get_path.py           83     0  100%
src/tests/test_get_shortest_path.py  43     0  100%
src/tests/test_orchestrator.py       70     0  100%
src/tests/test_osmnx_connector.py    44     0  100%
TOTAL                               494     59   88%

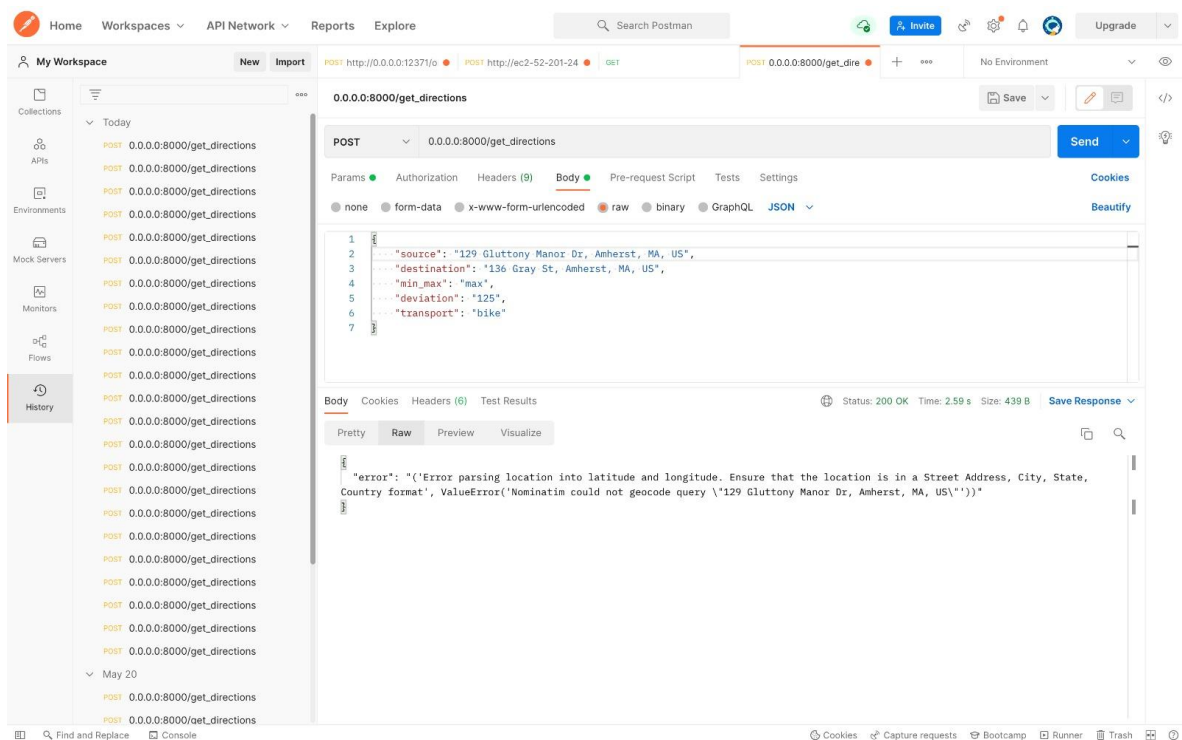
17 passed in 13.99s
```

- Integration Tests - We have covered end-to-end testing of all components based on the test plan manually using Postman. The screenshots are present in the tests folder as well.

### Success Case:

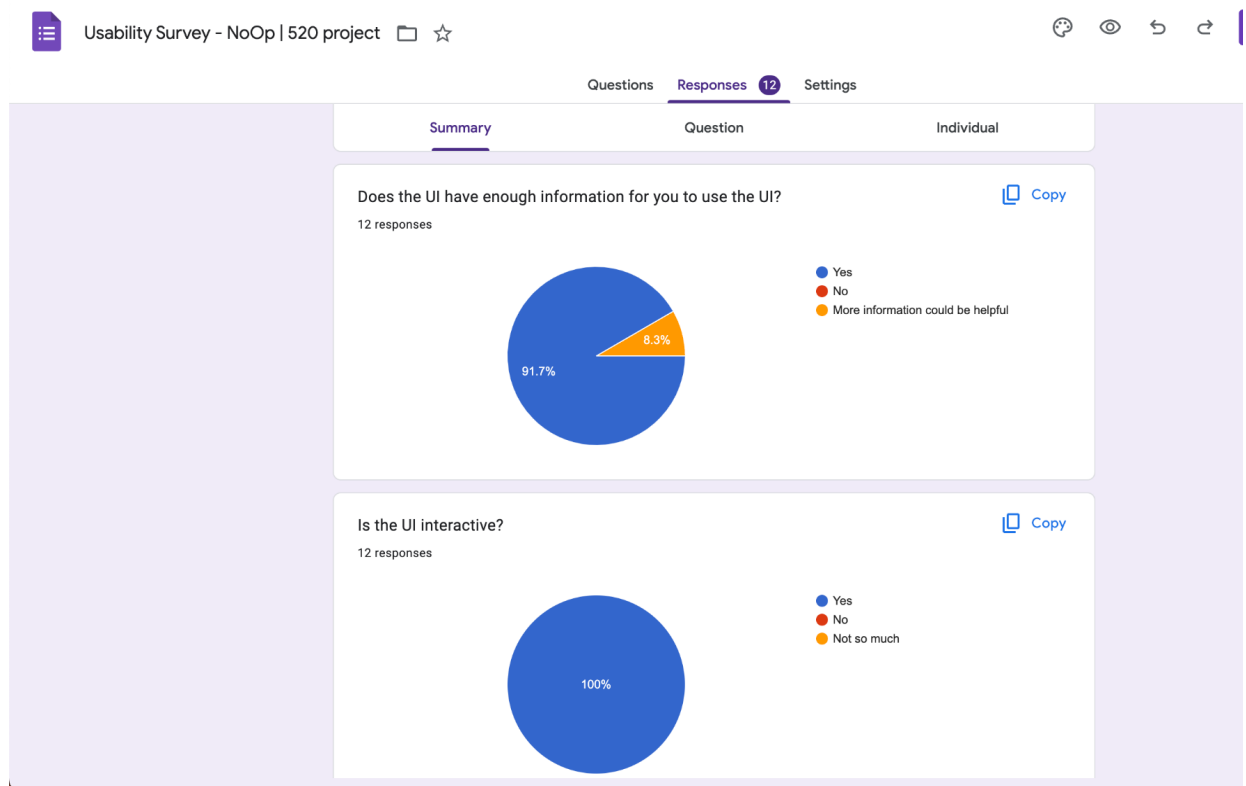


### Error Case:



- UI Tests - We have used alpha/beta testing for UI (which will also cover integration tests) by sharing the application with our friends. We also shared a Usability survey with our friends. We made sure to share the application with our friends outside of this course to avoid any academic dishonesty. The results of it are shown below:

Usability survey responses: [Sheet Link](#), [Response Analysis](#)



Do you have some feedback for us? How could we improve the UI?

10 responses

The UI seems to be good. It could be improved by choosing a more vibrant colour palette for a better aesthetic look.

You could add a slider for the amount elevation. Elevation heatmap along with the current output would be helpful.

I liked the idea of getting elevation details for the path which could be useful for things like biking and walking. This is not the general case with Google maps and seems like a useful feature

UI was Device was helpful sufficient for my use and has a lot of applications in our daily life.

The color contrast could be better. It seems like it is not very disability friendly, like color blindness.

UI can be a bit smooth but it's okay for a start

Side by side comparison instead of stacked comparison is easier to view on small screens.

No. It looks good

Brilliant project! Very intuitive

Add hints for every input field. Makes it easier for the user

## Verifying the non-functional requirements

- We have written code while following peer programming to ensure the code is well-written. As a part of it we also did code reviews among the group members.
- We also conducted usability surveys with our friends to ensure the app is working as expected. The results of it are shared above.

## Execution Plan and Contributions

1. Design - All team members
    - a. UI Mockup - Jai
    - b. Architecture - Pragya
    - c. Algorithm Design - Saksham
    - d. Evaluation Plan - Nidhi
  2. UI development - Jai Puro
  3. Backend-Development - Saksham, Pragya
    - a. Path finding algorithms, interacting with libraries such as OSMnx, Google Maps, Open-Elevation, etc.
  4. Integration - All team members
  5. Evaluation and testing - Nidhi
  6. Project Report and Documentation - All team members
- Because of peer programming each group member was involved in all the components of the application.