ELeNA - Elevation Navigation System

Varun Kumar Bachi Reddy
Neha Pendem
Somesh Pillay
Rahul Seetharaman

Introduction:

In this development project, we implement ELeNA, an elevation based navigation system. Conventional navigation systems solve the problem of helping the user find the shortest path from a given source and destination, given constraints like traffic. However for the recreational user, who wishes to take a trek, burn some calories, or try a different route these systems often fall short. ELeNA aims to solve the problem of optimizing for constraints like minimum or maximum elevation, given the user has to travel from source to destination. Typical use cases of maximizing elevation include trekking, cycling, workouts and many more. Similarly, minimizing elevation is crucial for people who may have health problems, and wish to avoid bumpy routes.

Requirements Specification:

Functional Requirements:

Functional requirements broadly define the flow of usage of a product/software. They give tangible requirements on how a specific software is expected to behave and how the user interaction is defined. They are typically defined through user stories. In the case of ELeNA the following functional requirements were kept in mind, when implementing the project.

User story: The user needs to select a source and destination with maximizing or minimizing elevation gain

1. The user is able to see a Google Maps like interface
2. The user is able to enter their desired source and destination
3. The user is able to able to enter their desired elevation gain threshold and has the option of maximizing or minimizing the elevation gain

4. The user is able to find the desired route, which is visualized on a Google Maps like interface

Non Functional Requirements

The following non functional requirements were kept in mind, when implementing the project. These are broadly categorized into three categories

Product Use:

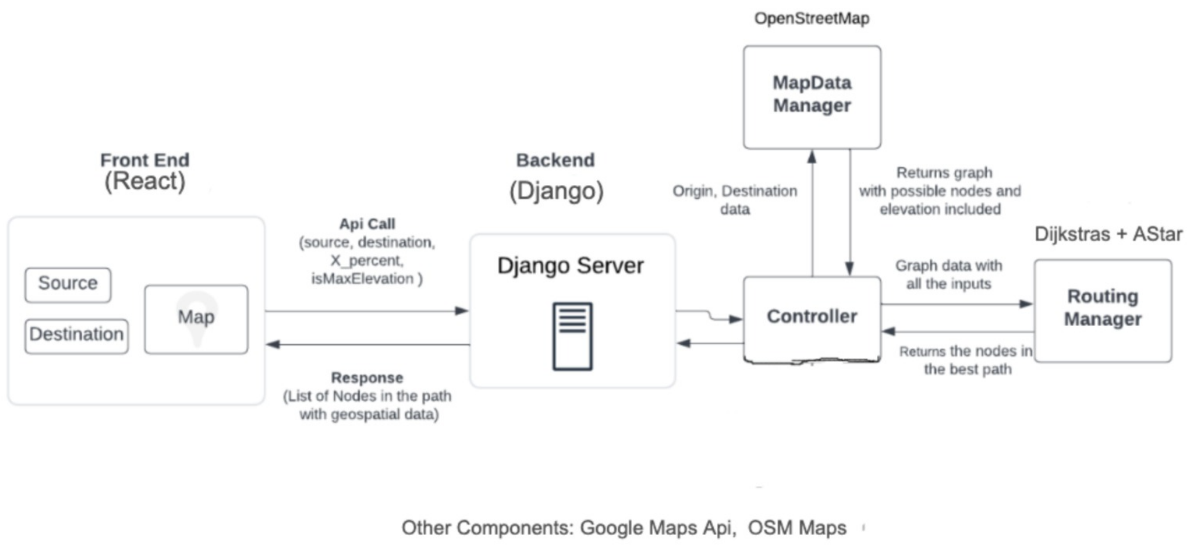
1. Usability: The UI must be extremely intuitive and easy to use. The coloring scheme must be consistent and offer the user a seamless experience from selecting their source and destination to visualizing their routes
2. Efficiency: The application should be designed to work in a real-time scenario with low latencies. The UI should be implemented in a way that it works smoothly and loads fast on a web browser. The response time pertaining to a user's request should be real time and fast.
3. Correctness: Correctness is one of the most critical non functional requirements. The path described by the application should be optimal given the constraints and should serve the user's end goal, be it minimizing elevation, maximizing elevation or simply finding a shortest path given a large space of elevation thresholds.

Product Revision:

This describes the non functional requirements from a developer perspective.

1. Testability: The product must be well tested with sufficient unit test coverage so as to cover all/most of the corner cases. The tests must be written for correctness, and ensuring that the algorithms used in the application adhere to the end goals of the user, which is minimizing or maximizing the elevation gain.
2. Maintainability: The code must be composed of modular components which follows best practices in software development and makes optimal use of design patterns. Maintainability is crucial for the long term code health and making sure developers who are newly onboarded to the project don't have a hard time understanding and making changes to the code as well as adding their own new components.
3. Reusability: The code must be composed of components that are reusable. It must follow the principle of DRY (Don't Repeat Yourself). Every functionality that is represented as code must have an unambiguous representation in the system, and all other components that must use this functionality must be through software abstractions.

Architecture diagram of the proposed system


OpenStreetMap
MapData Manager
Front End (React)
Backend (Django)
Origin, Destination data
Returns graph with possible nodes and elevation included
Dijkstras + AStar
Api Call (source, destination, X_percent, isMaxElevation)
Django Server
Graph data with all the inputs
Source
Map
Destination
Controller
Routing Manager
Returns the nodes in the best path
Response (List of Nodes in the path with geospatial data)
Other Components: Google Maps Api, OSM Maps

The architecture diagram of the project can be broken down into the following major components: Frontend, Backend and algorithms. The front end is implemented using ReactJS, a web framework that is typically used in modern web applications and has several features like Virtual DOM, which make UI rendering fast and responsive. The backend is implemented in Django, a python web framework that is used to write modular code adhering to MVC patterns. Every API call in Django is expressed in terms of routes, which allow incoming HTTP requests and send responses in the form of JSON. The React application then makes HTTP calls to the Django server through REST APIs and passing JSON payloads.

User interaction:
When the user enters a source, destination and their given constraints with respect to the elevation, the latitude and longitude of the source and destination are fetched with Maps APIs. Then the OSMNX APIs (which is an integration of NetworkX and OpenStreetMap) is responsible for fetching the source, destination and intermediate locations as nodes in a graph. OSMNX does this by drawing a bound box around the source and destination nodes and fetching all the intermediary nodes and constructing a graph based on their distances. Once this process is completed, the routing algorithms implemented (Dijkstra and A*) solve the problem of finding the shortest path given the user specific requirements. The total distance and elevation gain is shown along with the map visualization.

Evaluation:

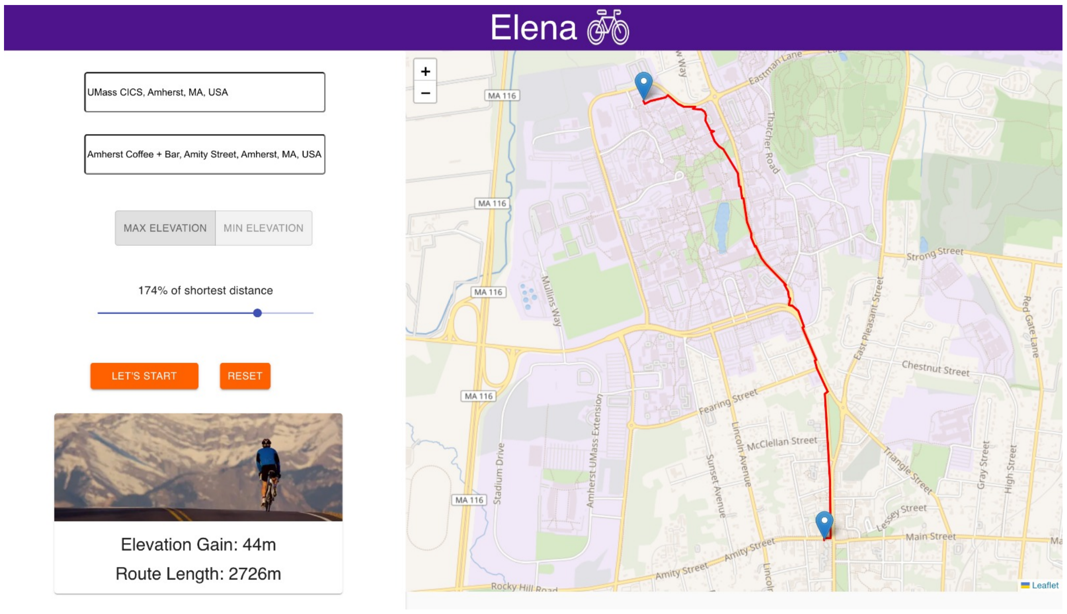
The evaluation criteria is as follows.

Unit testing: Test individual components which are as follows
1. Graph Provider API
2. Routing algorithms like Dijkstra, A*, etc.

Integration testing: Test the application end to end and check for request response latencies, time it takes for the system to complete a user request, correctness of the routes generated and overall usability of the application. To this end, we evaluate the application with respect to several landmarks in and around Amherst, MA

- The front end is tested for its responsiveness by running through all possible user stories, which in this would be both minimization and maximization of elevation gain.
- The routing algorithms are tested with mock graph provider APIs, in order to verify their correctness.
- The graph provider component is tested in tandem with the OSMNX library which is responsible for generating the graph and providing the data model with which the application works.

Application Screenshots:



UMass CICS, Amherst, MA, USA

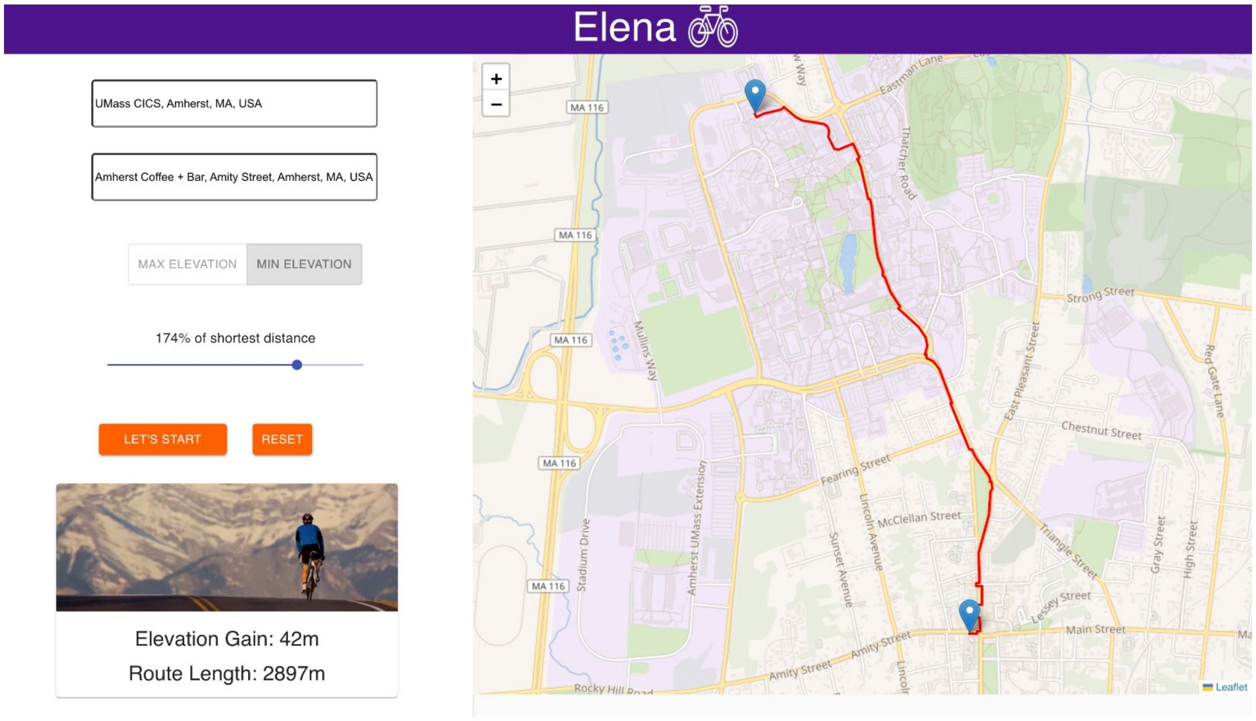
Amherst Coffee + Bar, Amity Street, Amherst, MA, USA

174% of shortest distance

Elevation Gain: 44m

Route Length: 2726m

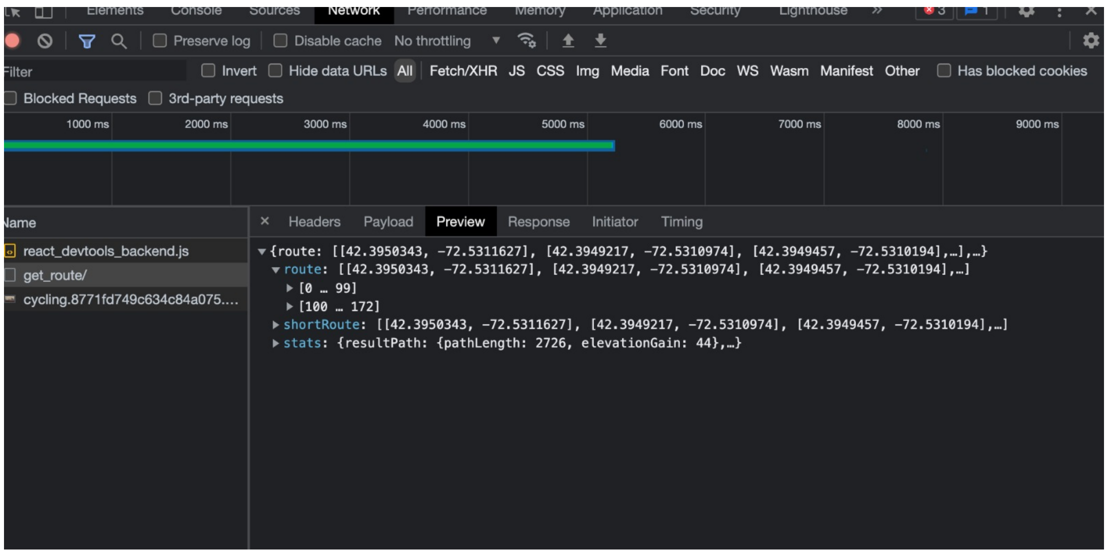
Fig 1: Screenshot describing the max elevation gain user story


Elena
UMass CICS, Amherst, MA, USA
Amherst Coffee + Bar, Amity Street, Amherst, MA, USA
MAX ELEVATION
MIN ELEVATION
174% of shortest distance
LET'S START
RESET
Leaflet

Elevation Gain: 42m
Route Length: 2897m

Fig 2: Screenshot describing the min elevation gain user story

Screenshots of the backend interaction:


Elements Console Sources Network Performance Memory Application Security Lighthouse
Preserve log
Disable cache
No throttling
Filter
Invert
Hide data URLs
All Fetch/XHR JS CSS Img Media Font Doc WS Wasm Manifest Other
Has blocked cookies
Blocked Requests
3rd-party requests
1000 ms 2000 ms 3000 ms 4000 ms 5000 ms 6000 ms 7000 ms 8000 ms 9000 ms
Name
Headers Payload Preview Response Initiator Timing

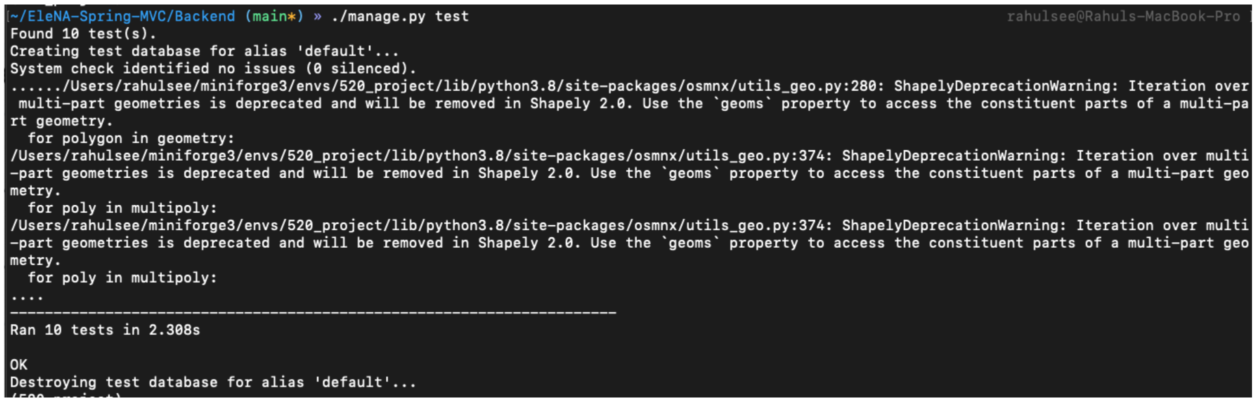
```
react_devtools_backend.js
get_route/
cycling.8771fd749c634c84a075....
```

```
▼{route: [[42.3950343, -72.5311627], [42.3949217, -72.5310974], [42.3949457, -72.5310194],…],…}
  ▼route: [[42.3950343, -72.5311627], [42.3949217, -72.5310974], [42.3949457, -72.5310194],…]
    ▶[0 … 99]
    ▶[100 … 172]
  ▶shortRoute: [[42.3950343, -72.5311627], [42.3949217, -72.5310974], [42.3949457, -72.5310194],…]
  ▶stats: {resultPath: {pathLength: 2726, elevationGain: 44},…}
```

Fig 3: The payload to the Django API server is illustrated in the browser console

The computed route, the shortest paths along with data like path length and elevation gain is returned as a JSON response payload by the Django server.

Screenshots of unit tests



```
~/EleNA-Spring-MVC/Backend (main*) » ./manage.py test                                                      rahulsee@Rahuls-MacBook-Pro
Found 10 test(s).
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....../Users/rahulsee/miniforge3/envs/520_project/lib/python3.8/site-packages/osmnx/utils_geo.py:280: ShapelyDeprecationWarning: Iteration over
 multi-part geometries is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access the constituent parts of a multi-pa
rt geometry.
  for polygon in geometry:
/Users/rahulsee/miniforge3/envs/520_project/lib/python3.8/site-packages/osmnx/utils_geo.py:374: ShapelyDeprecationWarning: Iteration over multi
-part geometries is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access the constituent parts of a multi-part geo
metry.
  for poly in multipoly:
/Users/rahulsee/miniforge3/envs/520_project/lib/python3.8/site-packages/osmnx/utils_geo.py:374: ShapelyDeprecationWarning: Iteration over multi
-part geometries is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access the constituent parts of a multi-part geo
metry.
  for poly in multipoly:
....
----------------------------------------------------------------------
Ran 10 tests in 2.308s

OK
Destroying test database for alias 'default'...
```