Pranav, Shubham, Dipti, Neeharika (Developers)

Evaluation and Design Document

Elevation-based Navigation System (EleNa)

(Source Code in main_project branch)

Problem Statement:

The objective of this project is to develop a cutting-edge application that offers users the optimal route between a user-defined source and destination, considering individual preferences and requirements. This includes the ability to maximize or minimize elevation gain along the route while ensuring that the route distance falls within a specified percentage of the shortest path between the source and destination.

Overview Explanation:

Elevation-based Navigation (Elena) is an advanced map-based application specifically engineered to cater to the needs of users seeking route recommendations that go beyond conventional navigation systems. By integrating elevation data into the route calculation process, Elena empowers users to make informed decisions based on a comprehensive understanding of both distance and elevation dynamics.

Traditionally, navigation systems have focused solely on providing the shortest path without taking into account the impact of elevation changes. However, for activities such as hiking, mountain climbing, or paragliding, where the terrain's elevation plays a pivotal role, a specialized approach is necessary. Elena serves as a groundbreaking solution by offering users the ability to define their preferred elevation criteria, such as optimizing for maximum or minimum elevation gain along the route.

Moreover, Elena goes a step further by allowing users to specify a tolerance percentage (denoted as "x%"), ensuring that the recommended route deviates within the desired range of distance from the shortest path. This intelligent flexibility strikes a balance between route optimization and adhering to user-specific preferences.

The result is an unparalleled navigation experience, empowering outdoor enthusiasts to plan their journeys with unparalleled precision and efficiency. By factoring in both elevation and distance considerations, Elena revolutionizes the way individuals engage in outdoor activities, enabling them to chart the most suitable course that aligns seamlessly with their unique requirements.

Features

1. **Elevation data**: EleNa uses elevation data to provide users with the most efficient route based on elevation changes.
2. **Route planning**: Users can input their starting point and destination, and EleNa will calculate the most efficient route based on elevation changes.
3. **Offline maps**: EleNa can provide offline maps,so users can navigate without an internet connection.
4. **Multiple path options**: Elena gives users an option to select between the elevation path and the shortest path.
5. **Path Limits**: The user can set up a threshold value(x) which limits the elevation path to that x% of the shortest path.
6. **Error handling:** If the user enters a wrong value, error messages will pop up and ask the user to re-enter the wrongly entered value.
7. **Autocomplete Location**: To enhance user experience, Elena has integrated Google's Places API for autocomplete functionality of the entered location.
8. **Algorithm Selection**: The user can enter the algorithm they want to use for generating the path.

Functional Requirements

- The application is designed with the capability to retrieve essential information from its users, which is then utilized to calculate the most optimal route. This information typically includes details about the starting point, the destination point, and the user's preferences with respect to either maximizing or minimizing elevation changes along the route.
- The program generates a map that will display the most optimal route between the starting point and the ending point calculated by the algorithm chosen by the user along with the elevation gain.

- The users are given a choice to choose either the A* or Dijkstra's algorithm.

Non-functional Requirements

- **Usability -** The design of the application is focused on providing a user-friendly experience, which is achieved through the incorporation of several key features. One such feature is the input options that are available to the user. The user can enter an address using a text box which has an autocomplete feature. Additionally, the output information that the program provides is designed to be clear and informative. Users are able to view both the shortest possible route and the route that takes elevation changes into account, giving them a comprehensive view of their journey. Finally, the application includes a reset function that allows users to easily revert the program back to its default settings, providing them with greater flexibility and control.

- **Debuggability -** Adding proper comments that explain the purpose of the code can make it easier for other developers to understand the code and its functionality. This can help to reduce confusion and ensure that future updates or modifications to the code are made correctly. Additionally, adding logger statements can help to identify errors or issues in the code by tracking the program's execution and providing a detailed log of the program's behavior. This can be particularly helpful in troubleshooting and debugging the code, as it provides valuable information about the program's execution and can help to pinpoint the source of any errors or issues that arise. By combining these two approaches, developers can create code that is both easier to understand and easier to debug, ultimately resulting in a more robust and reliable software application.

- **Modularity of Code -** Model-View-Controller (MVC) design pattern is used to ensure the modularity of the code. This pattern separates the different aspects of the code into distinct modules, each with a specific responsibility. The Model module is responsible for managing the data and logic of the application, while the View module is responsible for rendering the user interface. The Controller

module acts as an intermediary between the Model and View, processing user input and updating the Model and View as necessary.

- **Extensibility -** The Strategy design pattern will be utilized to ensure the application's extensibility, allowing for easy incorporation of additional algorithms. Other potential extensions, such as adjusting the radius of consideration to include geographic regions, can also be accommodated. Additionally, the Observer design pattern will be implemented to enable the program to efficiently and effectively handle multiple users concurrently.

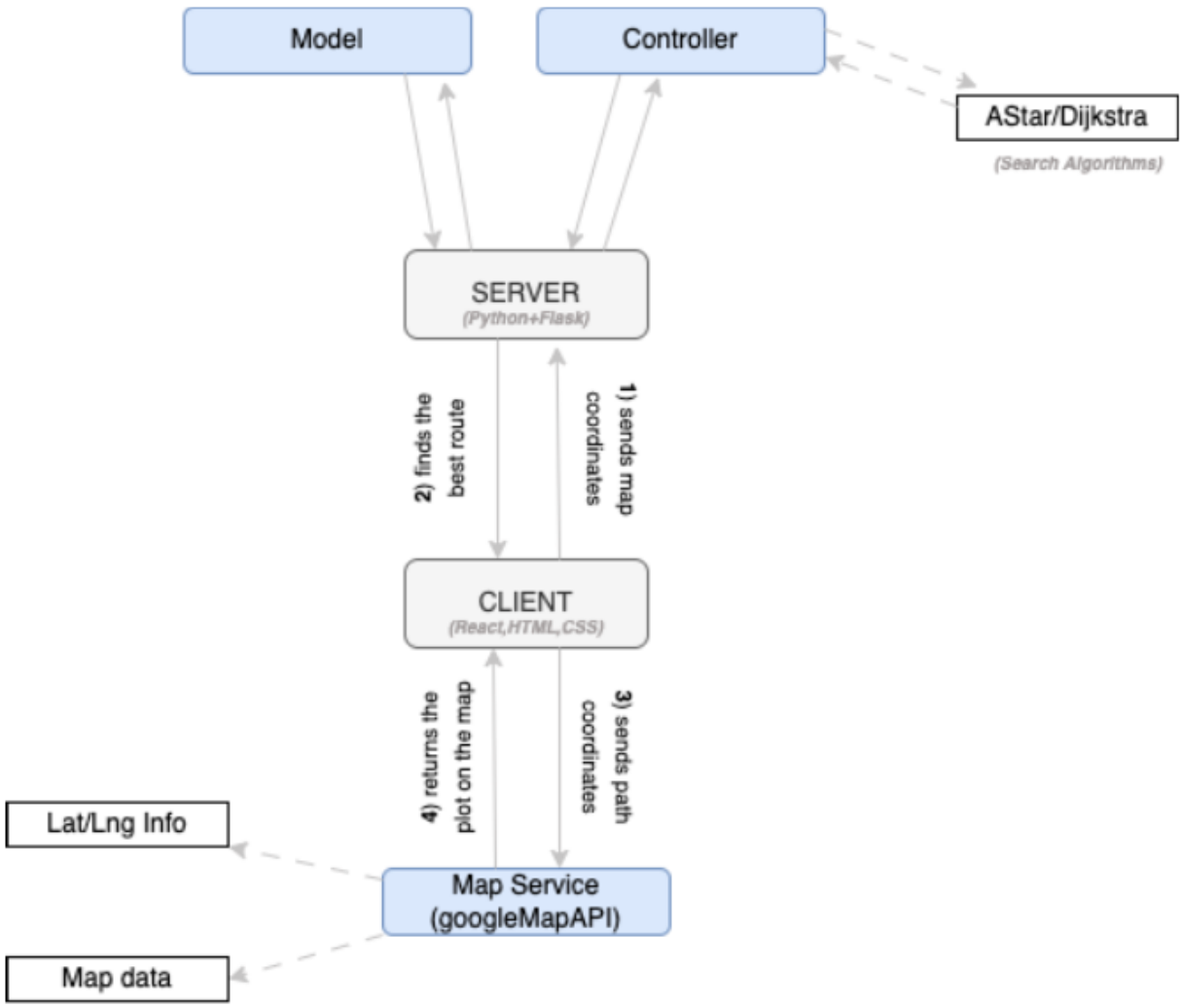
- **Testability -** The use of the Model-View-Controller (MVC) design pattern in the application architecture provides the benefit of easier testing. The Model module, responsible for data and logic management, can be tested separately from the View and Controller modules, avoiding any impact on the Model tests from changes in the View or Controller. For automated testing, the application incorporates the unittest feature, facilitating the quick and easy execution of automated test cases. Additionally, UI white box testing is employed to ensure proper functioning of the user interface. These testing measures ensure the application's reliability and quality.

Architecture

- Model-View-Controller(MVC) architecture pattern is used.
- The View will have :
    - Origin and the destination location
    - Option to choose the algorithm - A* or Dijkstra’s
    - Path Limit
    - Option to choose between minimum and maximum elevation

The generation of the graph and population of the nodes with elevation attributes are accomplished by the react library: React-google-maps/api.

Tech Stack along with UI design mockup and data model



The technology stack chosen for EleNa, our elevation-based navigation system, has been carefully selected to deliver a professional and efficient application. Here's why this stack is well-suited for our project:

Frontend Development:

- To create a modern and interactive user interface, we have opted for React, a widely adopted JavaScript library. React's component-based architecture, efficient rendering, and reusable code components allow us to build dynamic and engaging web pages. With React, we can provide users with a seamless and responsive navigation experience.

Backend Development:

- Python, a versatile and powerful programming language, serves as the foundation for our backend development. Its extensive library ecosystem and user-friendly syntax make it an excellent choice for building web applications. We leverage Flask, a lightweight and flexible web framework, to handle the backend logic and build RESTful APIs. Flask's simplicity and modular design enable us to develop efficient and scalable web applications with ease.

Map Service:

- To handle longitude, latitude, and elevation values, we have used Google Maps API. This modular component separates the map-related functionality, making our code more organized and maintainable. By decoupling map-related operations, we can focus on providing accurate and reliable route calculations based on elevation changes.

Route Calculation:

- For optimal route calculation, we employ either the Dijkstra or A* algorithm. These well-established algorithms are widely recognized for their ability to find the shortest path in a graph efficiently. By considering elevation changes, our algorithms can determine the most efficient route for our users. These algorithms excel in performance and can handle large datasets, ensuring a smooth and reliable navigation experience.

Overall, our chosen technology stack combines the versatility and interactivity of React on the frontend, the power and flexibility of Python on the backend, and the efficiency of the selected algorithms. This combination enables us to deliver a professional and high-performing elevation-based navigation system, providing users with a seamless and optimized journey..

Design Patterns

- We made use of the Strategy design pattern to implement the two different algorithms and employ multiple heuristics using a shared template.

- In addition, the application's architecture incorporates the Observer design pattern, which involves the Model serving as the observed element and the View as the observing element.

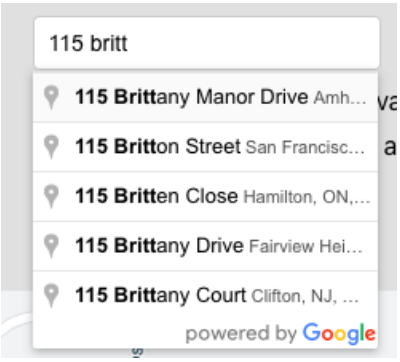
Evaluation

- The application is equipped with unit test cases to ensure the correctness and functionality of each module and component.
- Unit Test Cases have been added for backend and integration testing.
- Unit Test Cases have been added for frontend using React testing library.
- We also performed Integration testing.
- To ensure that the application is user friendly and meets the intended requirements, it is subjected to user acceptance testing. This process will involve having individuals from the target user group test the application and provide feedback on its usability and functionality.

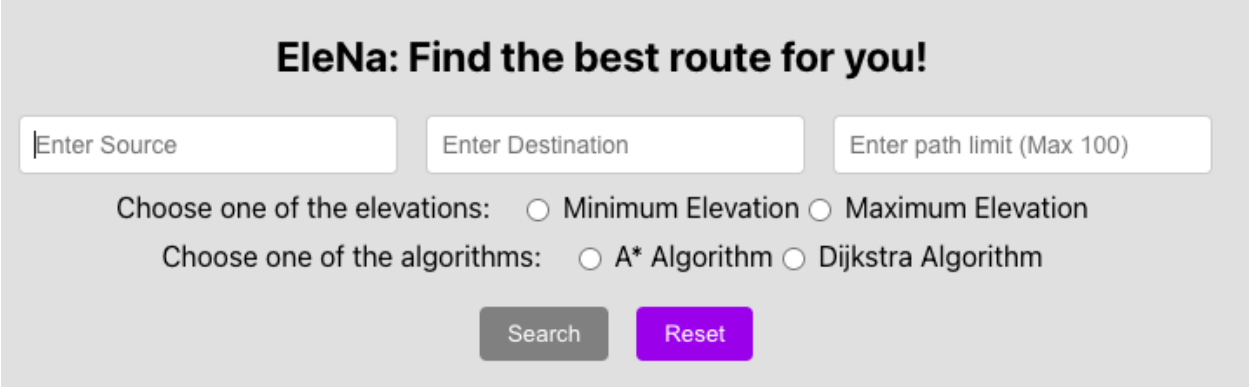
Implementation:

Frontend:

The user of Elena, will input the source and destination locations, as well as three mandatory inputs: Path limit, Minimum/Maximum elevation, and an algorithm selection. To assist the user in selecting the desired locations, an autocomplete feature powered by Google's Places API will provide suggestions as she types. Once the user has entered the source and destination, the system will retrieve the corresponding coordinates (latitude and longitude) from the Places API. These coordinates are crucial for the server to calculate the best route based on the specified parameters.



Autocomplete Input



Input panel of UI

The client packages a response and sends it to the server in the format below:

```
const responseToBackend = {
    origin_location: [src.geometry.location.lat(), src.geometry.location.lng()],
    destination_location: [dest.geometry.location.lat(),
dest.geometry.location.lng()],
    elevation_option: elevation,
    selected_algorithm: algorithm,
    path_limit: threshold
```

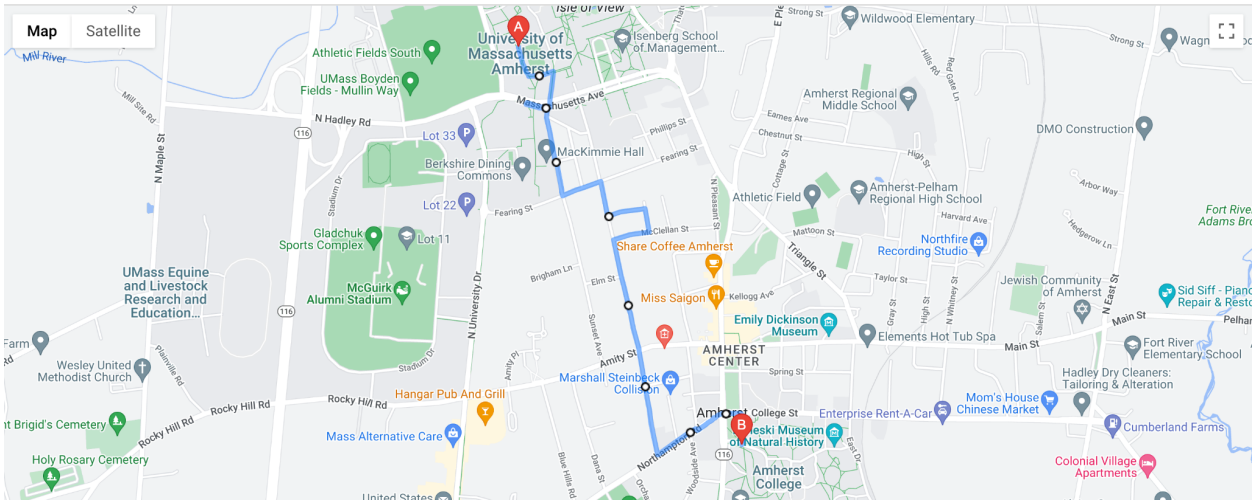
Frontend Response

After calculating the best route according to the response sent by the client, the server sends a json response with 2 calculated paths - Elena path and the shortest path. The response from the backend looks like this.

```
pranavjain@vl965-172-31-136-220 src % ./runserver.sh
 * Serving Flask app 'server/backend.py'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 597-694-927
127.0.0.1 - - [22/May/2023 21:17:29] "OPTIONS /get_elena_path HTTP/1.1" 200 -
Loading the offline map.... ../offlineStreetMap.p
Shortest route between source and destination has been calculated.
---------------
Route Details
Approach: Shortest Route
Route Distance: 2490.565999999999
Elevation Gain: 50.02000000000002
---------------
---------------
Route Details
Approach: AStar
Route Distance: 2839.7699999999995
Elevation Gain: 50.903
---------------
Elevation Output Paths:  {'path_elevation': {'metadata': {}, 'shape': {'categoryType': 'LineString', 'data_points': [[-
72.53016, 42.3869153], [-72.529962, 42.3863936], [-72.5295766, 42.3858026], [-72.5294147, 42.3857955], [-72.5291043, 42
.3858516], [-72.5292577, 42.3853328], [-72.5293725, 42.3843417], [-72.5295353, 42.3847339], [-72.5287791, 42.3846255],
[-72.5287837, 42.3844569], [-72.5286169, 42.3836623], [-72.528536, 42.383668], [-72.528253, 42.382628], [-72.5272908, 4
2.3827924], [-72.526921, 42.381636], [-72.525996, 42.381885], [-72.525604, 42.380582], [-72.525331, 42.3797], [-72.5250
64, 42.378797], [-72.524935, 42.378361], [-72.5245971, 42.3772539], [-72.5240821, 42.3757133], [-72.5240445, 42.3756115
], [-72.5240061, 42.3754923], [-72.523719, 42.374218], [-72.5233471, 42.3724717], [-72.5231617, 42.3716005], [-72.52211
, 42.372123], [-72.521438, 42.372461], [-72.5211667, 42.3726155], [-72.5200234, 42.3731753], [-72.519809, 42.373212], [
-72.5196664, 42.3732169], [-72.5196797, 42.3730901], [-72.519189, 42.3723861], [-72.5188338, 42.3721427], [-72.518841,
42.372027], [-72.5187658, 42.3720136], [-72.5175993, 42.3715067], [-72.5175895, 42.3710233]]}, 'categoryType': 'ELEMENT
'}, 'path_shortest': {'metadata': {}, 'shape': {'categoryType': 'LineString', 'data_points': [[-72.53016, 42.3869153],
[-72.529962, 42.3863936], [-72.5295766, 42.3858026], [-72.5294147, 42.3857955], [-72.5291043, 42.3858516], [-72.5283609
, 42.3860189], [-72.5277209, 42.386181], [-72.5276688, 42.386207], [-72.5276419, 42.3861387], [-72.5275085, 42.3858522]
, [-72.5273844, 42.385564], [-72.5272171, 42.3852478], [-72.5271854, 42.3851632], [-72.527095, 42.3849387], [-72.527066
7, 42.3848588], [-72.5267302, 42.3840772], [-72.5267029, 42.3840081], [-72.52661, 42.383749], [-72.526476, 42.383347],
[-72.5260145, 42.3819395], [-72.525996, 42.381885], [-72.525604, 42.380582], [-72.525331, 42.3797], [-72.525064, 42.378
797], [-72.524935, 42.378361], [-72.5245971, 42.3772539], [-72.5240821, 42.3757133], [-72.5240445, 42.3756115], [-72.52
29257, 42.3757769], [-72.5227132, 42.3757923], [-72.5226619, 42.3757943], [-72.5222831, 42.3758025], [-72.5221964, 42.3
758029], [-72.5215644, 42.3758061], [-72.5215537, 42.3757177], [-72.5215229, 42.3754627], [-72.521505, 42.3753145], [-7
2.521492, 42.3752068], [-72.5214742, 42.3750594], [-72.5214374, 42.3747548], [-72.521416, 42.374593], [-72.5213832, 42.
3743512], [-72.521331, 42.373932], [-72.5210695, 42.3739465], [-72.5207739, 42.3739628], [-72.520719, 42.3739659], [-72
.5205993, 42.3739725], [-72.5199656, 42.3739981], [-72.519821, 42.374004], [-72.5198112, 42.3733559], [-72.5197036, 42.
3733586], [-72.5196499, 42.3733165], [-72.5196664, 42.3732169], [-72.5196797, 42.3730901], [-72.519189, 42.3723861], [-
72.5188338, 42.3721427], [-72.5187689, 42.37213], [-72.5187658, 42.3720136], [-72.5175993, 42.3715067], [-72.5175895, 4
2.3710233]]}, 'categoryType': 'ELEMENT'}, 'shortest_distance': 2490.565999999999, 'shortest_elevation_gain': 50.0200000
0000002, 'shortest_elevation_drop': 0, 'start_location': 'University of Massachusetts Amherst, Mullins Way, Hadley, Ham
pshire County, Massachusetts, 01003, United States', 'end_location': 'Amherst College, Norwottuck Rail Trail, Amherst,
Hampshire County, Massachusetts, 01002, United States', 'elevation_path_distance': 2839.7699999999995, 'elevation_path_
gain': 50.903, 'elevation_path_drop': 0, 'boolean_flag': 2}
127.0.0.1 - - [22/May/2023 21:17:31] "POST /get_elena_path HTTP/1.1" 200 -
```

Backend Response

Then we use the react library - react-google-maps/api, and its components - GoogleMap, Marker, DirectionsRenderer, to plot the elena path on the map.



Map and Route rendering on UI

Backend:

We utilize Flask as the backend framework, which offers a convenient development server and efficient debugging capabilities. The backend implementation of the EleNa project is responsible for handling API requests and processing the data to calculate the optimal route based on the provided parameters. Let's explore how the backend implementation works based on the given code. To incorporate elevation information, we make use of the Open Elevation API.

Importing Dependencies:

- The json module is imported to handle JSON data.
- The googlemaps library is imported to interact with the Google Maps API.
- The necessary modules and classes from Flask are imported to create a Flask application and handle HTTP requests.
- The Model, AStarController, DijkstraController, and NotificationHandler classes are imported from the respective modules.

Flask Application Setup:

- An instance of the Flask application is created with the name app.
- Cross-Origin Resource Sharing (CORS) is enabled using the flask_cors extension to allow requests from a specific origin (http://localhost:3000).
- The root route ('/') returns a simple greeting message indicating that the Developers EleNa Server is running.

API Endpoint for Getting Elena Path:

- The '/get_elena_path' route is defined with the POST method to handle requests for obtaining the optimal route.
- The request body is expected to contain JSON data with the following attributes: 'origin_location', 'destination_location', 'path_limit', 'elevation_option', and 'selected_algorithm'.
- The JSON data is retrieved using request.get_json(force=True).
- The origin and destination addresses are extracted from the JSON data.
- The necessary parameters for route calculation, such as origin, destination, path limit, elevation strategy, and selected algorithm, are assigned to variables.
- An instance of the Model class is created.
- An instance of the NotificationHandler class is created and added as an observer to the model.
- Depending on the selected algorithm, an instance of either AStarController or DijkstraController is created.
- The model, start location, end location, path limit, and elevation strategy are set for the controller.
- The controller manipulates the route model by executing the algorithm and considering elevation options.
- The result, obtained from the view in the form of a JSON output, is stored in the output variable.
- The JSON output, including the optimal path and relevant information, is returned as the API response.

Upon processing the input JSON data, the backend generates an output JSON containing the shortest paths, both with and without elevation gain.

Algorithm:

Dijkstra's algorithm is a method used to determine the shortest path between two vertices in a graph. It employs a greedy approach, where the algorithm selects the next best solution with the hope that it will lead to the overall best solution for the entire problem.

A* The algorithm is similar to Dijkstra's algorithm, but with one key difference. A* incorporates a heuristic function that assigns priority to nodes based on their expected superiority compared to others. Unlike Dijkstra's algorithm, which explores all possible paths, A* algorithm selectively explores vertices by greedily choosing the next vertex to explore based on the value of f(v) [f(v) = h(v) + g(v)]. Here, h represents the heuristic function, and g represents the accumulated cost up to that point.

In summary, A* algorithm is often referred to as a "best first search" because it intelligently selects the next vertex to explore by considering both the heuristic value and the current cost.

Testing and Evaluation:

Frontend testing

We have used the package from react called as the “@testing-library/react” in order to perform the frontend testing. This package provides all the utilities to test the different React components. It mainly helps in writing the tests which are based on the user actions and interactions with the various components instead of depending on the implementation details. The tests performed on the frontend are as follows:

1. **Heading -** UI consists of a component called as the Heading and we tested if the Heading component is loaded with the styles provided.

2. **Input Field - Source -** UI consists of different inputs and tested if the input field source is loaded and displayed properly with a placeholder.

3. **Input Field - Destination -** UI consists of different inputs and tested if the input field destination is loaded and displayed properly with a placeholder.

4. **Input Field - Path Limit -** UI consists of different inputs and tested if the input field path limit is loaded and displayed properly with a placeholder.

5. **Search Button Disabled -** UI consists of a Search button to find the path where the button has to be disabled before all of the required fields and the options are selected.

6. **Reset Button Disabled -** UI consists of a Reset button to reset all of the entered values from the different input fields so that the user can start his new search. This button has to be disabled when all of the input fields are empty.

7. **Metrics -** UI consists of different metrics that are displayed like the EleNa Path distance, elevation gain, shortest path and the elevation gain of the shortest path. All of these labels and values have to be displayed on the UI.

8. **Radio Buttons - Min/Max elevation -** UI consists of radio buttons to choose either a minimum or maximum elevation from the source to the destination.

9. **Radio Buttons - A*/Dijkstra -** UI consists of radio buttons to choose from two different algorithms namely A* and Dijkstra from the source to the destination.

All of the above mentioned tests have passed and here is the screenshot of the same.

```
PASS src/App.test.js
  ✓ Heading is Loaded (9 ms)
  ✓ Value in the input field – Source (2 ms)
  ✓ Value in the input field – Destination (2 ms)
  ✓ Value in the input field – Path Limit (1 ms)
  ✓ Search button is disabled when the input fields are empty (15 ms)
  ✓ Reset button is disabled when the input fields are empty (5 ms)
  ✓ Shortest Path Metrics are rendered correctly (4 ms)
  ✓ Testing the radio buttons of the min-max elevation (4 ms)
  ✓ Testing the radio buttons of the algorithm (5 ms)

Test Suites: 1 passed, 1 total
Tests:       9 passed, 9 total
Snapshots:   0 total
Time:        0.805 s, estimated 1 s
Ran all test suites.

Watch Usage: Press w to show more.
```

Backend / Integration Testing

We have used the unit-test framework in order to perform the backend/integration testing. The unit-test framework is a library which helps in providing a set of functionalities and also different structures which makes it easy to write, execute as well as organize the different test cases. It creates automated tests for individual pieces of code which can be either at function level, class level or can also be a functionality. The tests performed on the backend are as follows:

Backend

1. **Map Rendering -** Checked if the map is being rendered correctly on the frontend with no errors and has a pin(center).

2. **Coordinates - Address -** It is essential for the coordinates of a particular address to get converted into the address in order to display it on the UI.

Integration

1. **A* - Min elevation -** The path which has been generated by the algorithm A* has minimum/lesser elevation compared to that of the shortest path between the source and destination.

2. **Dijkstra - Min elevation -** The path which has been generated by the algorithm Dijkstra has minimum/lesser elevation compared to that of the shortest path between the source and destination.

3. **A* - Max elevation -** The path which has been generated by the algorithm A* has maximum/greater elevation compared to that of the shortest path between the source and destination.

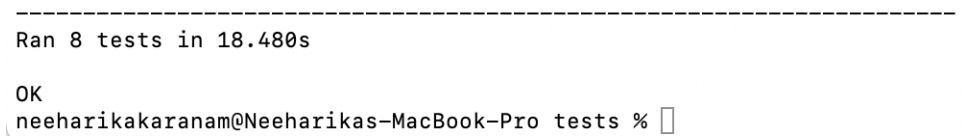
4. **Dijkstra - Max elevation -** The path which has been generated by the algorithm Dijkstra has maximum/greater elevation compared to that of the shortest path between the source and destination.

MVC architecture

1. **A* controller - Model -** The application uses MVC architecture, so we checked if the controller of the algorithm A* communicates with the model and makes the required changes.

2. **Dijkstra controller - Model -** The application uses MVC architecture, so we checked if the controller of the algorithm Dijkstra communicates with the model and makes the required changes.

All of the above mentioned tests have passed and here is the screenshot of the same.



Performance Evaluation Test

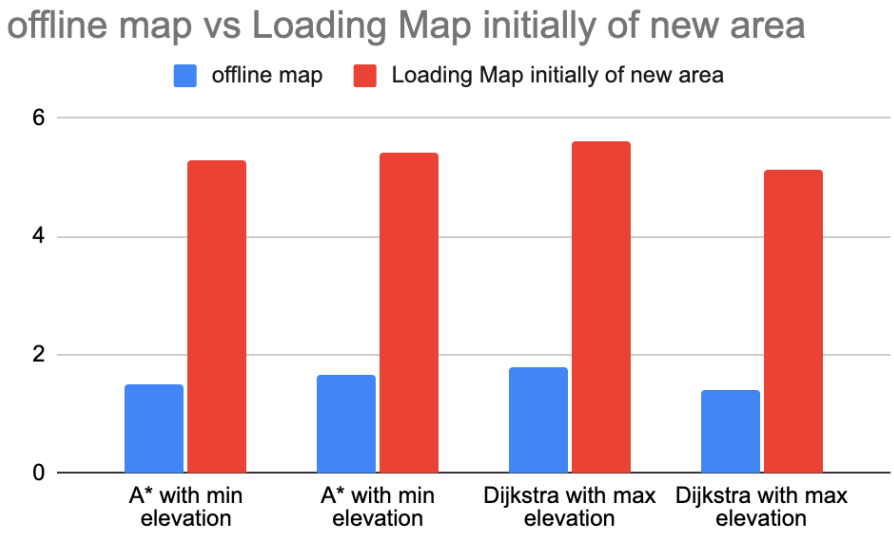
The performance of the "/get_elena_path" API, responsible for retrieving optimal path coordinates, was thoroughly tested in our application. The evaluation encompassed several crucial aspects, including the response time of the API under various scenarios, taking into account different algorithms with minimum and maximum elevation gain settings, as well as different destinations from a specific source utilizing various algorithms and elevation gain parameters.

Furthermore, it is worth mentioning that the EleNa project incorporated both offline and online map functionalities. The offline map feature significantly contributed to enhancing user experience by facilitating quick access to path information. By enabling users to download the map for offline usage, they could retrieve path details swiftly, even in areas with limited or no internet connectivity. This offline map

capability was particularly beneficial in situations where immediate access to directions was crucial.

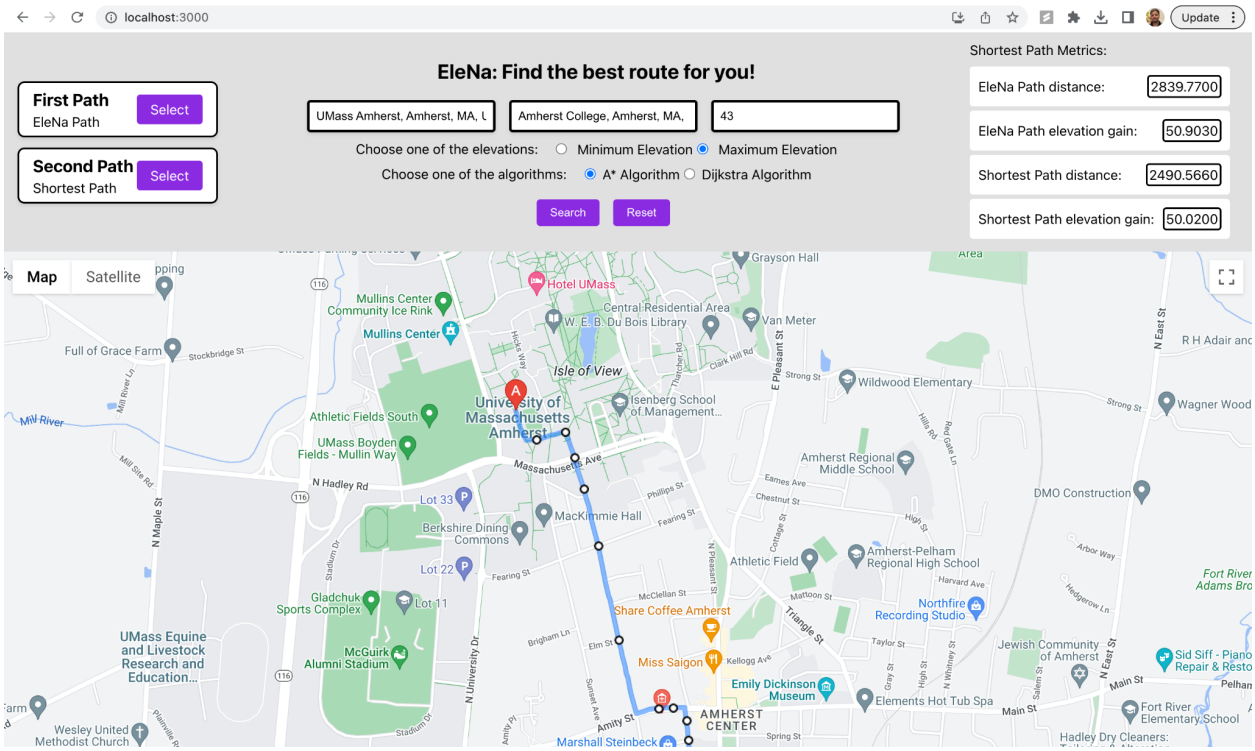
The performance evaluation, therefore, considered the effectiveness and efficiency of the offline map feature, analyzing its impact on reducing response time and providing users with expedited access to path information.

Overall, the performance evaluation aimed to provide comprehensive insights into the EleNa project, ensuring the API's responsiveness, optimizing path retrieval, and leveraging the advantages of both offline and online map functionalities to enhance user experience.



X axis : Algorithms
Y axis: Time is seconds

Pair programming teams:
Shubham and Pranav (Backend):
Github usernames: (shubhampatel007889, PranavJain23)
Dipti and Neeharika (Frontend and Tests):
Github usernames: (DiptiLohia, Neeharikaranam)



Final Web View

Links

Project Description Presentation Video :
https://drive.google.com/file/d/163dqVol35NZNQvTPCL6FvH84GUfvj8Np/view?usp=sharing

Github repo link:
https://github.com/DiptiLohia/developers-elena-520/tree/main_project