Elevation-based navigation system (EleNa)

Team Developers
- Pranav
- Shubham
- Dipti
- Neeharika

The Problem

A Software program designed to guide a user from a starting point to a destination by either minimizing or maximizing changes in elevation while also keeping the overall distance traveled between the two locations to x% of the shortest possible route.

Individuals who enjoy hiking or jogging have the option to use the app and choose the type of workout they prefer, ranging from low intensity to high intensity, depending on their personal preferences.

Project Requirements

Function Requirements:

- The app has the capability to obtain the necessary information from users needed to calculate the optimal route, including the starting point, the ending point, and preferences for maximizing or minimizing elevation.
- The program generates a map that displays the most optimal route between the starting point and the ending point, calculated through the algorithm that corresponds to the user's selected preferences. Users are provided with the option to select either the A* or Dijkstra's algorithm for determining the best route.

Project Requirements

Non- Functional requirements:

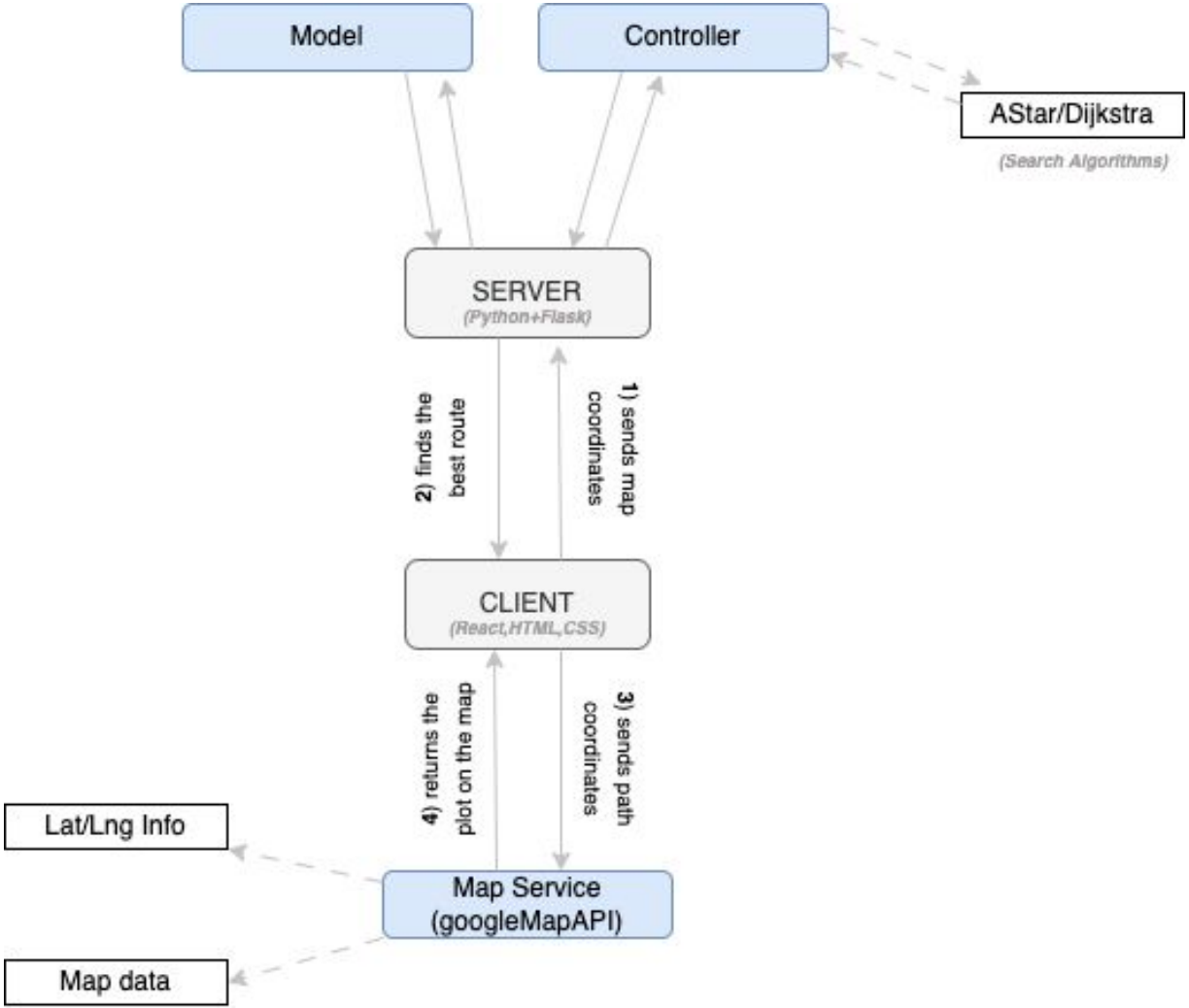
- Usability : The design of the application prioritizes user-friendliness by incorporating the following features:
  - Input options: Users can mark specific points on the map or enter an address using a text box.
  - Output information: The program displays both the shortest route and the route that takes elevation changes into account.
  - Reset function: Users can easily revert the application back to its default settings.
- Debuggability
- Modularity of code

- Extensibility : By utilizing the Strategy design pattern, the program can easily incorporate additional algorithms into its implementation.Other potential extensions include increasing the geographic region by adjusting the radius of consideration
    - Additionally, the Observer design pattern ensures that the program is capable of accommodating multiple users concurrently, resulting in a more efficient and effective implementation.

- Testability
    - The implementation of the MVC design pattern ensures that the application can be easily tested.
    - The incorporation of the unittest feature enables the automation of test cases, while UI white box testing is also employed.

Design Patterns:

- Model-View-Controller (MVC) architecture pattern has been used.
- The View has :
  - Origin and Destination location
  - Path Limit (X%)
  - Select one of the two algorithms - Dijkstra and A*
  - Option to choose between Minimum and Maximum Elevation
- React’s Google Maps API
- Google Maps Elevation API


Model
Controller
AStar/Dijkstra
(Search Algorithms)
SERVER
(Python+Flask)
2) finds the best route
1) sends map coordinates
CLIENT
(React,HTML,CSS)
4) returns the plot on the map
3) sends path coordinates
Lat/Lng Info
Map Service
(googleMapAPI)
Map data

Design Patterns

- Regarding the program's design patterns, we utilized the Strategy pattern to employ two distinct algorithms (A* and Dijkstra), as well as multiple heuristics through a shared template.
- Furthermore, we implemented the Observer pattern in our architecture, where the model serves as the observable element and the view acts as the observer.

The Evaluation

- Unit Test Cases have been added for backend and integration testing.
- Unit Test Cases have been added for frontend using React testing library.
- We did Integration Testing.
- To verify the usability of the application, it undergoes user acceptance testing, which involves having peers run and test the program.

The Plan

- Backend : Pranav and Shubham
  - We successfully implemented the path finding algorithm using A*, Dijkstra Elevation API.
- FrontEnd : Dipti and Neeharika
  - We successfully implemented the implementation of the UI.
- Unit tests : successfully implemented
- Integration Testing : successfully implemented
- Bugs Fixing & Error Handling: successfully implemented