Pranav, Shubham, Dipti, Neeharika (Developers)

Software requirements specification Document

Elevation-based Navigation System (EleNa)

(Source Code in main_project branch)

Introduction

The EleNa (Elevation-based Navigation) project aims to develop a cutting-edge application that provides users with optimal route recommendations based on individual preferences and requirements. Traditional navigation systems focus solely on the shortest path between two points, neglecting the impact of elevation changes. However, for activities such as hiking, mountain climbing, or paragliding, where elevation plays a crucial role, a specialized approach is necessary.

Purpose of the document:

The purpose of this document is to provide a comprehensive overview of the EleNa project, including its objectives, features, functionalities, and technical specifications. It serves as a reference guide for project stakeholders, developers, and other individuals involved in the project's planning, development, and implementation. The document outlines the project's problem statement, goals, and the proposed solution to address the identified challenges.

Project Overview:

The EleNa project focuses on developing an advanced map-based application that integrates elevation data into the route calculation process. The application aims to provide users with route recommendations that go beyond conventional navigation systems. By factoring in elevation dynamics, users can make informed decisions based on a comprehensive understanding of both distance and elevation changes along the route.

The project's core features include:

- **Elevation Optimization**: Users can specify their preferred elevation criteria, such as maximizing or minimizing elevation gain along the route. This allows outdoor enthusiasts to tailor their journeys based on their specific preferences and activity requirements.
- **Tolerance Percentage**: The application allows users to set a tolerance percentage, denoted as "x%," to ensure that the recommended route deviates within a desired range of distance from the shortest path. This feature strikes a balance between route optimization and adherence to user-specific preferences.
- **User Interface**: The EleNa application provides a user-friendly interface where users can input their source and destination locations, specify elevation preferences, and view the recommended route on an interactive map. The interface is designed to be intuitive, visually appealing, and easy to navigate.

Document Conventions and Terminology:

To ensure clarity and consistency throughout this document, the following conventions and terminology will be used:

- **Project-specific terminology**: Any domain-specific terms or acronyms relevant to the EleNa project will be defined and used consistently throughout the document.
- **Formatting conventions:** Headings, subheadings, bullet points, and numbering will be used to organize and structure the document for ease of reading and comprehension.

Scope

Project Scope Description:

The scope of the EleNa project encompasses the development of a comprehensive application that offers users optimal route recommendations based on elevation considerations. The application will provide functionality for users to input their source and destination locations, specify their elevation preferences (maximum or minimum gain), and set a tolerance percentage for the deviation from the shortest path distance. The application will calculate and display the recommended route that meets the user's criteria. Additionally, the application will incorporate map visualization, allowing users to view the recommended route and relevant elevation data.

Stakeholders and User Roles:

- **Users:** The primary stakeholders and users of the EleNa application are outdoor enthusiasts, such as hikers, climbers, and paragliders, who require specialized route recommendations based on elevation considerations.
- **Developers:** The development team (our team of “Developers”) responsible for designing, implementing, and maintaining the EleNa application. We were also responsible for overseeing the project, coordinating resources, and ensuring timely delivery.

Assumptions and Dependencies:

- **Availability of Elevation Data:** The EleNa application depends on accurate and up-to-date elevation data for the specified geographic areas. The project assumes the availability of reliable elevation datasets or APIs to access this data.
- **Internet Connectivity:** Users will require an internet connection to access and utilize the EleNa application.
- **User Input:** The accuracy and correctness of user-provided input, such as source and destination locations, elevation preferences, and tolerance percentage, are assumed for generating accurate route recommendations.

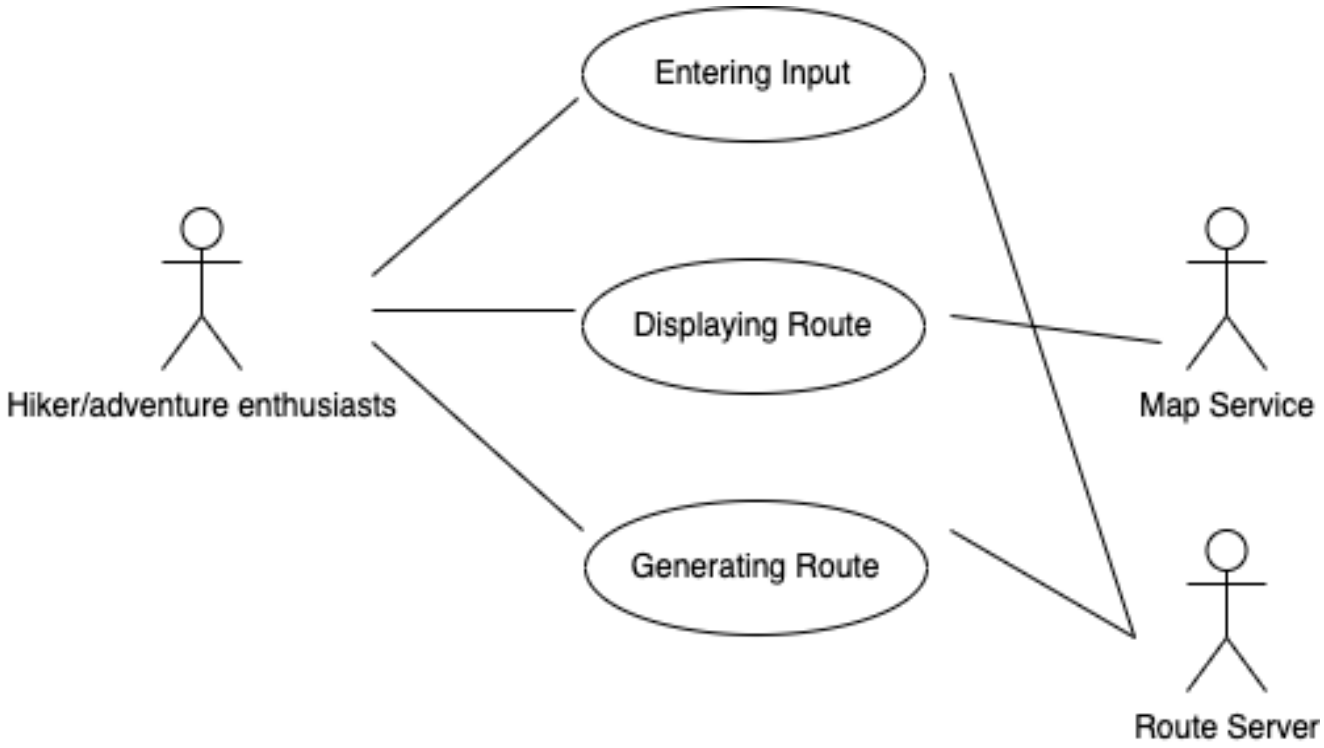
Product Features and Functionalities:

- **Source and Destination Input:** Users can enter their desired source and destination locations to generate route recommendations.
- **Elevation Preferences:** Users can specify whether they want to optimize for maximum or minimum elevation gain along the route.
- **Tolerance Percentage:** Users can set a tolerance percentage, allowing the recommended route to deviate within a desired range of distance from the shortest path.

- **Route Calculation:** The application will calculate the recommended route based on the user's input, considering elevation data and the specified criteria.
- **Map Visualization:** The recommended route and relevant elevation data will be visually displayed on a map interface for users to view and analyze.
- **User Feedback:** Users may provide feedback on the recommended routes and overall application experience to improve future iterations.
- **Error Handling:** The application will include appropriate error handling mechanisms to handle invalid inputs, network errors, and other exceptional scenarios.

Functional Requirements

Use case diagram



Use Case Descriptions:

Enter Source:

- Description: User enters the starting point of their journey.
- Actors: User
- Preconditions: None
- Postconditions: Starting point is recorded.

Enter Destination:

- Description: User enters the destination point of their journey.

- Actors: User
- Preconditions: None
- Postconditions: Destination point is recorded.

Choose Algorithm:
- Description: User selects the algorithm to be used for calculating the optimal route.
- Actors: User
- Preconditions: Starting point and destination are recorded.
- Postconditions: Algorithm choice is recorded.

Choose Elevation Type:
- Description: User selects whether to maximize or minimize elevation changes along the route.
- Actors: User
- Preconditions: Starting point, destination, and algorithm choice are recorded.
- Postconditions: Elevation type preference is recorded.

Set Tolerance Percentage:
- Description: User sets the tolerance percentage for the deviation from the shortest path distance.
- Actors: User
- Preconditions: Starting point, destination, algorithm choice, and elevation type preference are recorded.
- Postconditions: Tolerance percentage is recorded.

Set Elevation Preference:
- Description: User sets the elevation preference (maximum or minimum gain) along the route.
- Actors: User
- Preconditions: Starting point, destination, algorithm choice, and tolerance percentage are recorded.
- Postconditions: Elevation preference is recorded.

Generate Optimal Route:
- Description: The system calculates the most optimal route based on the user's inputs.
- Actors: System
- Preconditions: Starting point, destination, algorithm choice, elevation type preference, tolerance percentage, and elevation preference are recorded.
- Postconditions: Optimal route is generated.

Display Route on Map:
- Description: The system displays the optimal route and relevant elevation information on a map.
- Actors: System

- Preconditions: Optimal route is generated.
- Postconditions: Route and elevation information are displayed on the map.

System Features and Interactions:

- User interacts with the system by entering the source and destination points.
- User selects the algorithm (A* or Dijkstra's) to be used for route calculation.
- User chooses the elevation type (maximize or minimize) preference.
- User sets the tolerance percentage for deviation from the shortest path distance.
- Users set the elevation preference (maximum or minimum gain) along the route.
- System calculates the optimal route based on user inputs using the selected algorithm.
- System generates a map displaying the optimal route and relevant elevation information.
- Users can view the recommended route and analyze elevation data on the map.
- The system handles invalid inputs, such as incorrect locations or missing preferences, and provides appropriate error messages to the user.

Non-Functional Requirements:

Performance Requirements:

- Response Time: The application should provide a responsive user interface with fast response times for route calculations and map visualization.
- Scalability: The application should be able to handle a growing number of users and accommodate increased data processing requirements without significant performance degradation.
- Efficiency: The route calculation algorithm should be optimized for efficient computation, minimizing processing time and resource utilization.

Security Requirements:

- Data Privacy: The application should ensure the privacy and security of user input and sensitive data, such as origin and destination addresses.
- Authentication and Authorization: If user accounts or personalized settings are implemented, appropriate authentication and authorization mechanisms should be in place to protect user data and ensure authorized access to the application's features.
- Secure Communication: The application should use secure communication protocols (e.g., HTTPS) to transmit data between the client and server, preventing unauthorized access or tampering.

Usability Requirements:

- Intuitive User Interface: The application should have a user-friendly interface, with clear and intuitive input options for origin and destination addresses.
- Clear Output Presentation: The recommended routes and elevation information should be presented in a clear and understandable manner, allowing users to easily interpret and compare the results.
- Reset Function: The application should provide a reset function that allows users to revert to default settings, enhancing usability and user control.

Compatibility Requirements:

- Cross-Platform Compatibility: The application should be compatible with major web browsers and operating systems, including desktop and mobile platforms, ensuring a consistent user experience across devices.
- Data Source Compatibility: The application should support integration with various elevation data sources, allowing for flexibility in accessing elevation information from different providers or APIs.

Scalability Requirements:

- **Concurrent User Support:** The application should be able to handle multiple concurrent users, providing accurate and timely route recommendations without performance degradation.
- **Data Handling:** The application should be able to process large datasets efficiently, accommodating increased data volumes as the user base grows.
- **Algorithm Extensibility:** The application's architecture should allow for easy integration of additional algorithms or optimization techniques, ensuring scalability in terms of incorporating future enhancements or customizations.

System Architecture:

High-level system architecture overview:

The EleNa project follows the Model-View-Controller (MVC) architecture pattern. The View component includes user interface elements such as origin and destination inputs, algorithm selection, and path limit options. The Model component is responsible for generating the graph and populating nodes with elevation attributes using the OpenStreetMap API and Google Maps Elevation API. The Model performs path computations and communicates the results to the View for display.

Component breakdown and description:

- View: The View component handles the user interface elements and user interactions. It includes input fields for origin and destination locations, options to choose the algorithm (A* or Dijkstra's), and the path limit. The View communicates user input to the Controller and displays the computed results.
- Model: The Model component is responsible for data management and computations. It interacts with external APIs, such as the OpenStreetMap API and Google Maps Elevation API, to generate the graph and retrieve elevation data. The Model uses the selected algorithm to perform path calculations, considering elevation changes. It communicates the results to the View for display.
- Controller: The Controller component acts as an intermediary between the View and Model. It handles user input and triggers appropriate actions in the Model. The Controller receives user input from the View, validates and processes it, and communicates with the Model to initiate graph generation and path computation. It also updates the View with the computed results.

Data flow and interaction diagrams:

The data flow in the EleNa system follows these general steps:

1. User provides origin and destination locations, selects algorithm and path limit options through the View.
2. The View sends the user input to the Controller.
3. The Controller validates and processes the input, triggering actions in the Model.
4. The Model interacts with external APIs to generate the graph and retrieve elevation data.
5. The Model performs path calculations using the selected algorithm, considering elevation changes.
6. The Model communicates the computed results to the Controller.
7. The Controller updates the View with the results, displaying the recommended routes and elevation information to the user.

User Interface Design:

Wireframes/Mockups of Key Screens:

- Homepage: The homepage will feature input fields for origin and destination locations, options to choose the algorithm (A* or Dijkstra's), and a path limit setting.
- Route Result: This screen will display the recommended route on a map, including elevation information. It will also show the shortest path and the path considering elevation changes, allowing users to compare and make informed decisions.
- Settings: The settings screen will allow users to customize their preferences, such as selecting units of measurement, adjusting elevation thresholds, or specifying a tolerance percentage for the route distance deviation.
- Error/Validation Messages: These screens will display error or validation messages when users input invalid or incomplete information, ensuring a smooth user experience.

UI Design Guidelines and Standards:

- Consistency: Maintain a consistent design throughout the application, including colors, typography, and layout.
- Clarity: Use clear and concise labels, headings, and instructions to guide users through the application.
- Accessibility: Follow accessibility guidelines to ensure that the application is usable by individuals with disabilities, including proper color contrast, keyboard navigation support, and alternative text for images.

User Interaction and Navigation:

- Input Interaction: Users will interact with input fields to enter origin and destination addresses, select algorithm options, and set path limits.
- Map Interaction: Users will be able to interact with the map, zooming in and out, panning, and clicking on markers to view additional information.

- Button Interaction: Users will click buttons to submit input, trigger route calculations, navigate between screens, and apply settings changes.
- Navigation Menu: Provide a menu or navigation bar for easy access to different screens and features within the application.
- Feedback and Confirmation: Provide visual feedback and confirmation messages to inform users of successful actions or notify them of any errors or issues.

Data Management:

Data Entities and Attributes:

- Location: Represents a geographical location and consists of attributes such as latitude, longitude, and elevation.
- Route: Represents a calculated route and includes attributes such as the list of waypoints, distance, elevation gain/loss, and algorithm used.
- User Preferences: Stores user-specific preferences, such as algorithm selection, path limit, and tolerance percentage.

Database Schema Design:

- Locations Table: Stores information about geographical locations, including latitude, longitude, and elevation.
- Routes Table: Stores calculated routes, including origin, destination, waypoints, distance, elevation gain/loss, and algorithm used.
- User Preferences Table: Stores user-specific preferences, including algorithm selection, path limit, and tolerance percentage.

Data Storage and Retrieval Requirements:

- Elevation Data: Retrieve elevation data from the Google Maps Elevation API or a suitable elevation data source and store it in a database for efficient retrieval during route calculations.
- Location Data: Store and retrieve location data, including latitude, longitude, and elevation, from a database to support route calculations and map visualization.
- Route Data: Store calculated routes in a database for future reference and quick retrieval when requested by users.

External Interfaces:

Third-party integrations:

- Google Maps API: The frontend of the application integrates with the Google Maps API provided by the @react-google-maps/api library. This API allows you to embed Google Maps into your application, access map features, and interact with the map.
- Google Maps Elevation API: The Model component utilizes the Google Maps Elevation API to retrieve elevation data for each node in the graph. This API provides accurate elevation information, enabling the application to consider elevation changes when calculating the recommended route.

API specifications and documentation:

- Google Maps API Documentation: The documentation for the Google Maps API provides detailed information on the available endpoints, parameters, and responses. It explains how to use the API to display maps, add markers, calculate routes, and access elevation data. You can refer to the official Google Maps API documentation for specific details and usage examples.
- Google Maps Elevation API Documentation: The Google Maps Elevation API documentation specifies the endpoints, parameters, and responses for retrieving elevation data. It explains how to make requests to the API and interpret the elevation information returned. You can find more information in the official Google Maps Elevation API documentation.

Hardware or software dependencies:

- Internet Connectivity: The application requires a stable internet connection to access the Google Maps API and retrieve map data, calculate routes, and retrieve elevation data.

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:

- Heading - UI consists of a component called as the Heading and we tested if the Heading component is loaded with the styles provided.
- Input Field - Source - UI consists of different inputs and tested if the input field source is loaded and displayed properly with a placeholder.
- Input Field - Destination - UI consists of different inputs and tested if the input field destination is loaded and displayed properly with a placeholder.
- 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.
- 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.
- 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.
- 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.
- Radio Buttons - Min/Max elevation - UI consists of radio buttons to choose either a minimum or maximum elevation from the source to the destination.

- 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.

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

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

- 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.
- 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.
- 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.
- 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

- 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.
- 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.