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.

The Stages of Software Development Cycle that we've

<mark>Planning stage:</mark>

Requirement Specifications and Analysis:
The requirement specification includes the assumptions about the system, major features of the application, and expected behavior of the system.

The functional requirements that we have considered for this system are as follows:
- The application must provide a valid route between 2 points in such a way that it maximizes or minimizes elevation gain under the constraint that the total distance outputted by the system cannot be more than x% of the shortest path.
  The inputs to the application are -
  1. Source address
  2. Destination address
  3. A choice between maximizing or minimizing elevation gain between the source and destination
  4. The acceptable percentage within which the system can provide the path length
 The expected output of the application is a numerical value for the path length and a map showing the visual representation of the exact route chosen by the system from source to destination.

- The system validates the inputs and instructs the user in case of invalid input values.

The non-functional requirements that we expect the system to fulfill are:

Understandability, Readability, Testability, Portability, Usability, Modularity. We will later explain how each of these requirements are satisfied by the system.

Design and Architecture:
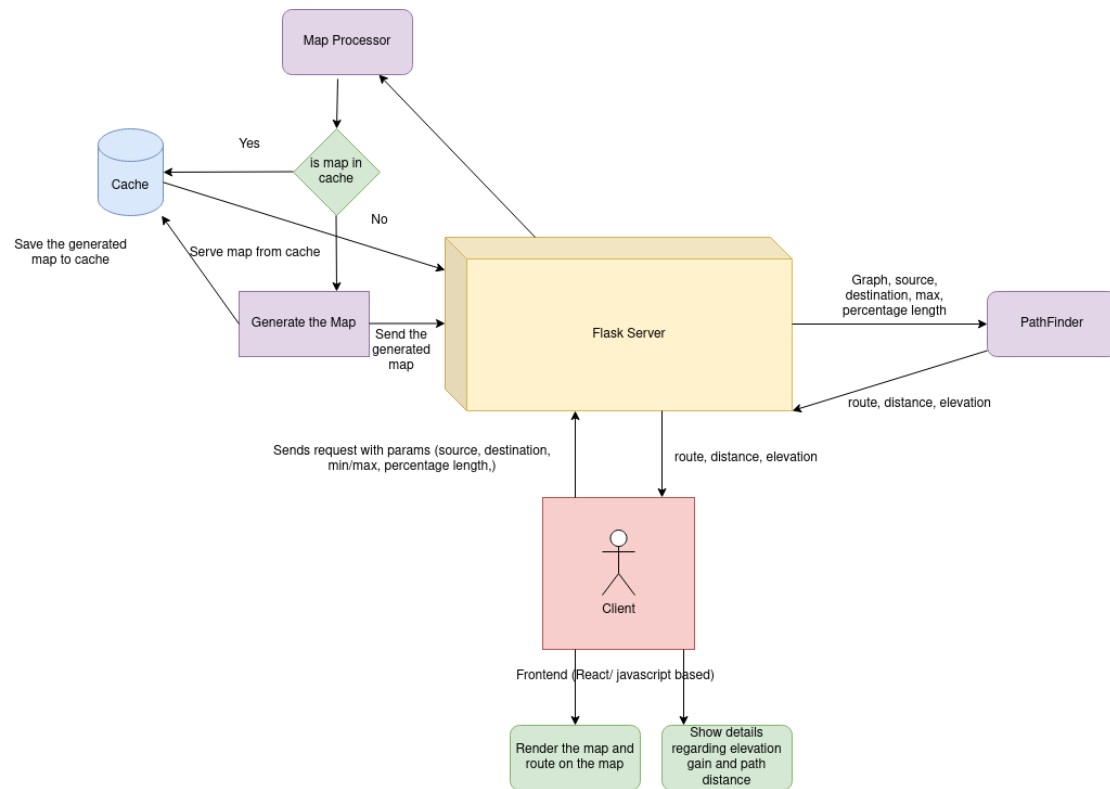
We have proposed the above design for the application. The application follows both Client- Server and MVC architecture patterns. <mark>Should we go into details?</mark>
After analyzing the requirements we concluded that the language of choice for implementing the application would be python for backend and JavaScript for frontend.

Development and Implementation:

The modules of the application are described below:

- **Client** - A Javascript based frontend client will be responsible for sending user input to the backend server. It will also render the map and generated route on the map for the user. It will ensure that only valid data is sent back to the server.
- **Backend Server** - A Flask based backend server. It will receive the input from the

client and call the map generator module to generate a map for the ==specified city.== It will then call the pathfinder module to generate the most optimized route for the given user input. The generated map and optimized route along with elevation gain info will be sent back to the client for frontend rendering.

- **Map Generator** - Map generator module will be responsible for fetching latitude and longitude for source and destination using geoPy and generating map of input ==city and state== using OSMnx library. Osmnx provides a method which attaches elevation data to each node of the generated graph using Google's elevation API. Since Google's elevation API is not free, we have modified this method to use Open-elevation API which is an open source version of Google's elevation API.
  Using the Open-elevation API is a time consuming process, thus to improve the response time of the application we have introduced a cache. This module caches any newly generated maps to avoid redundant calls to the API.
- **PathFinder** - PathFinder module will be responsible for finding an optimized path for a given user input. The optimized path is found through a 2 step process. In the first step, it finds the path elevation based route using 2 algorithms - Dijkstra's and A*. A* is a heuristic based algorithm and hence in certain situations might not yield the most optimized solution. Hence in the second step, the module's algorithm picker analyzes the output of both the algorithms and choses the one which is most aligned with the user's preferences.

Testing:

The testing is done to ensure that the entire application works according to the customer requirements.

An exhaustive test suite has been developed for the application including unit tests, ==integration tests and A/B testing for the UI. Coverage==

Tools and Methodologies -

- Language - Python
- Algorithms - Dijkstra/A*
- Libraries and Frameworks - Flask, osmnx, geoPy, Google Elevation API
- Frontend - React.js/JavaScript
- Testing - pytests