Trail Finder

Overview

Trail Finder is a web application designed to help users search for and filter trails based on various criteria, such as difficulty level, available amenities, and more. The application loads trail data from a CSV file and displays the results on a web page. The project is written in Go and uses the standard library for HTTP handling and CSV parsing.

Features

Filter by Address: Search for trails by specifying an address.

Filter by Difficulty: Narrow down trails based on difficulty levels such as Easy, Moderate, and Difficult.

Amenity Filters: Filter trails based on available restrooms, picnic areas, fishing spots, fees, and bike racks.

Responsive Design: The application is user-friendly and responsive across different devices.

Prerequisites

Go Programming Language: Ensure Go is installed on your system. You can download it from the official Go website (https://go.dev/dl/).

CSV File: The trail data is stored in a CSV file (`BoulderTrailHeads.csv`), which should be placed in the root directory of the project.

Installation

Clone the Repository:

```
git clone https://github.com/Saryu96/sap-eb-take-home-problem.git
cd trail-finder
```

Run the Application:

Execute the following command to start the server:

```
go run mail.go
```

Access the Application:
Open your web browser and navigate to `http://localhost:8080/trails` to access the application.

File Structure

mail.go: The main Go application file. It handles loading data from the CSV file, filtering the trails based on user input, and rendering the results using an HTML template.

trails.html: The HTML template file used to render the filtered trails in a table format.

BoulderTrailHeads.csv: The CSV file containing trail data such as name, address, difficulty level, and available amenities.

Code Documentation

mail.go

Trail Struct:

- Represents a single trail with various attributes like name, access type, restrooms, picnic, fishing, address, fee, bike rack, and difficulty.

mapdifficulty(class string) string:

- Converts the trail class code from the CSV file into a human-readable difficulty level such as "Easy," "Moderate," "Difficult," or "Most Difficult."

loadTrailsData(filename string) ([]Trail, error):

- Reads trail data from the specified CSV file and returns a slice of `Trail` structs.
- It maps the data from the CSV columns to the appropriate fields in the `Trail` struct and converts certain fields to boolean values.

filterTrails(trails []Trail, address, difficulty string, restrooms, picnic, fishing, fee, bikeRack bool) []Trail:

- Filters the trails based on the user's input criteria. It checks each trail against the provided filter values and returns only those trails that match the criteria.

handleTrails(w http.ResponseWriter, r *http.Request):

- Handles the HTTP request for the trails route.
- Parses user input from the request, loads trail data, filters the data based on user input, and renders the filtered results using the HTML template.

main():

- The entry point of the application. It sets up the HTTP routes and starts the server on port `8080`.

trails.html

HTML Structure:

- The HTML template includes a form for users to input their search criteria and a table to display the filtered trails.

Form:

- The form includes fields for address, difficulty, and checkboxes for restrooms, picnic, fishing, fee, and bike rack. When submitted, the form sends a GET request to the trails route with the user's input.

Table:

- The table dynamically displays the filtered trail data. Each row corresponds to a trail, and each column displays a specific attribute such as address, difficulty, and amenities (e.g., restrooms, picnic).