Geographic information System (GIS) and Global Positioning System (GPS) based Application Development

Final Project Report

DiverVenture: Interactive Mountain Hiking Trail Guide

Dimas Budi Nugraha (001201700023)

IS-2021 Class 2

❖ About DiverVenture

Diverventure is a web-based Geographic Information System (GIS) application designed to enhance the hiking experience by providing comprehensive information about hiking trails and points of interest in mountainous areas. With Diverventure, outdoor enthusiasts can explore and navigate hiking routes with ease, making their adventures safer and more enjoyable.

❖ Objectives

Diverventure is a GIS-based web application that focuses on mapping hiking trails and points of interest in mountainous areas. Its key goals include providing comprehensive trail information such as difficulty level, length, and duration to aid hikers in planning their trips effectively. The application also features a direction feature that guides users from their location to the chosen trail marker, promoting seamless navigation. Additionally, Diverventure highlights landmarks and viewpoints, enhancing the overall hiking experience. Diverventure aims to create a safe and enjoyable environment for hikers, empowering them to explore, plan, and navigate with confidence in mountainous terrains.

❖ Scope

- Mapping Hiking Trails:
    - The scope includes mapping out hiking trails in mountainous areas using GIS technology.
    - The application will display trail markers on the map to represent different hiking routes.

- Trail Information Display:
    - Users will be able to click on a trail marker to view detailed information about the hiking route.
    - The displayed information will include the difficulty level, length, and estimated duration of the hike.

- Direction Features:
    - The application will provide a direction feature that guides users from their current location to the selected trail marker.
    - Users will receive step-by-step directions to reach the starting point of the hiking trail.

- Points of Interest:
    - The scope includes highlighting points of interest, such as landmarks and viewpoints, within the mountainous areas.
    - Users will have access to additional information about these points of interest.

- User Interface:
    - The web application will have a user-friendly interface that allows for easy navigation and interaction.
    - The interface will be designed to provide a seamless user experience across different devices.

- Data Management:
    - The scope includes managing and updating the trail information and points of interest in the application's database.
    - Regular data maintenance and updates will ensure the accuracy and relevancy of the information.

The scope of the Diverventure project encompasses mapping hiking trails, displaying trail information, providing direction features, highlighting points of interest, designing a user-friendly interface, managing data, and ensuring scalability and performance. These features aim to deliver an immersive and efficient hiking experience for users exploring mountainous areas.

❖ Project Completion Status: 60%

This project update represents the final completion status of the Diverventure project, which stands at 60% of its intended scope. While significant progress has been made, it is important to acknowledge that certain aspects of the project remain unfinished. The completed portions of the project include:

- Hiking Trail Mapping:
  - The GIS-based mapping functionality has been implemented, allowing for the display of hiking trails on the map.

- Trail Marker Interaction:
  - Users can click on trail markers to access basic trail information, including difficulty level, length, and estimated duration.

- Initial Direction Features:
  - The initial version of the direction feature has been implemented, enabling users to receive directions from their current location to selected trail markers.

However, the following components of the project's scope remain incomplete:

- Trail Information Enhancement:
  - Detailed trail information, such as elevation profiles, points of interest along the trail, and user reviews, has not been fully integrated.

- Advanced Direction Features:
  - Enhancements to the direction feature, including real-time navigation, alternative route suggestions, and distance/time estimations, have yet to be implemented.

- Points of Interest Integration:
    - Additional points of interest, such as camping sites, water sources, and notable landmarks, have not been included in the application.

- User Interface Refinement:
    - The user interface design requires further refinement to enhance usability, visual appeal, and responsiveness across different devices.

- Data Management and Maintenance:
    - Ongoing data management tasks, such as updating trail information, adding new trails, and ensuring data accuracy, need to be established.

❖ Architecture and Technology Used for Diverventure

Diverventure is built using the PERN stack (PostgreSQL, Express.js, React, Node.js), along with the integration of the Google Maps API. The chosen architecture and technologies provide a robust and efficient foundation for developing a GIS-based web application. Here is an overview of the architecture and technologies used:

- Front-End:

React: The front-end of Diverventure is developed using React, a popular JavaScript library for building user interfaces. React enables the creation of interactive and responsive components for the web application.

- Back-End:

Node.js: Diverventure's back-end is powered by Node.js, a runtime environment that allows server-side JavaScript execution. Node.js enables the handling of server-side logic and API requests.

Express.js: Express.js, a lightweight web application framework for Node.js, is utilized to build the server-side application structure, handle routing, and manage HTTP requests.

- Database:

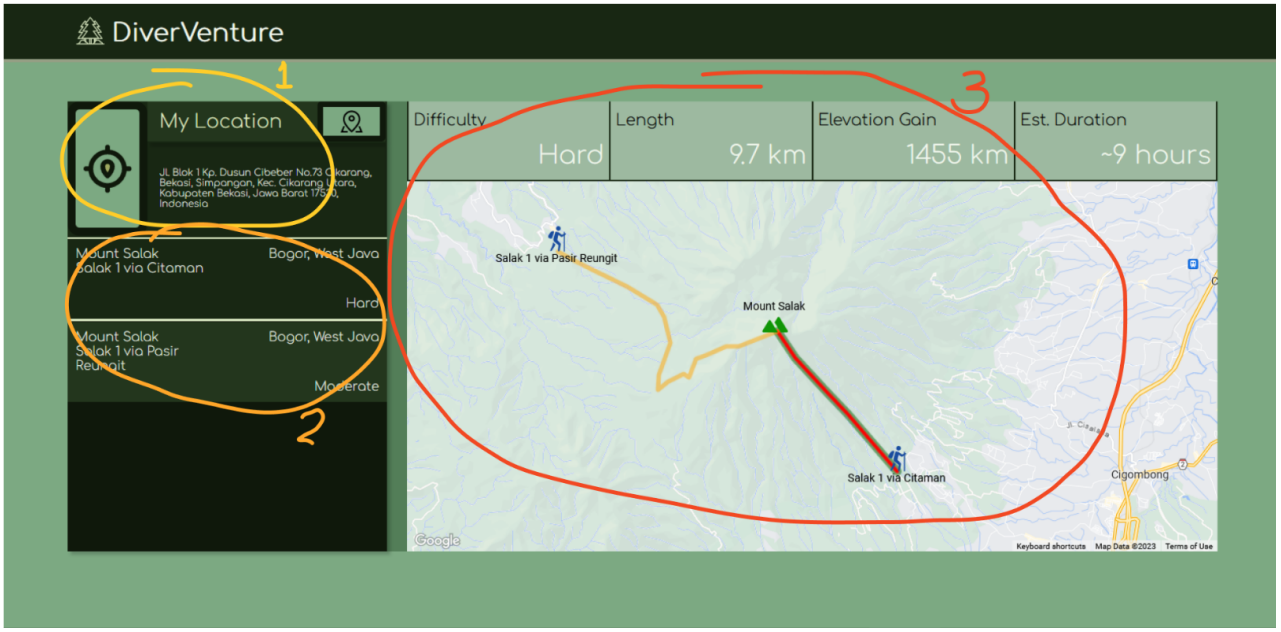
PostgreSQL: Diverventure utilizes PostgreSQL as the relational database management system (RDBMS) for storing and managing data. PostgreSQL provides robust data integrity, reliability, and scalability for the application.

- GIS Functionality:

Google Maps API: The integration of the Google Maps API enables Diverventure to display and interact with the map interface. It allows for the dynamic mapping of hiking trails, markers, and points of interest on the mountain area.

By leveraging the PERN stack and integrating the Google Maps API, Diverventure benefits from the flexibility and efficiency of the chosen technologies. The stack provides a seamless development experience, while the Google Maps API enriches the application with powerful mapping and geolocation capabilities. Together, these technologies create a solid foundation for building a feature-rich and user-friendly hiking trail mapping web application.

❖ How our application looks



In the Diverventure application, the user interface is designed to provide a seamless and intuitive experience for exploring hiking trails and points of interest in the mountainous areas. Here is a description of the various components and their functionalities:

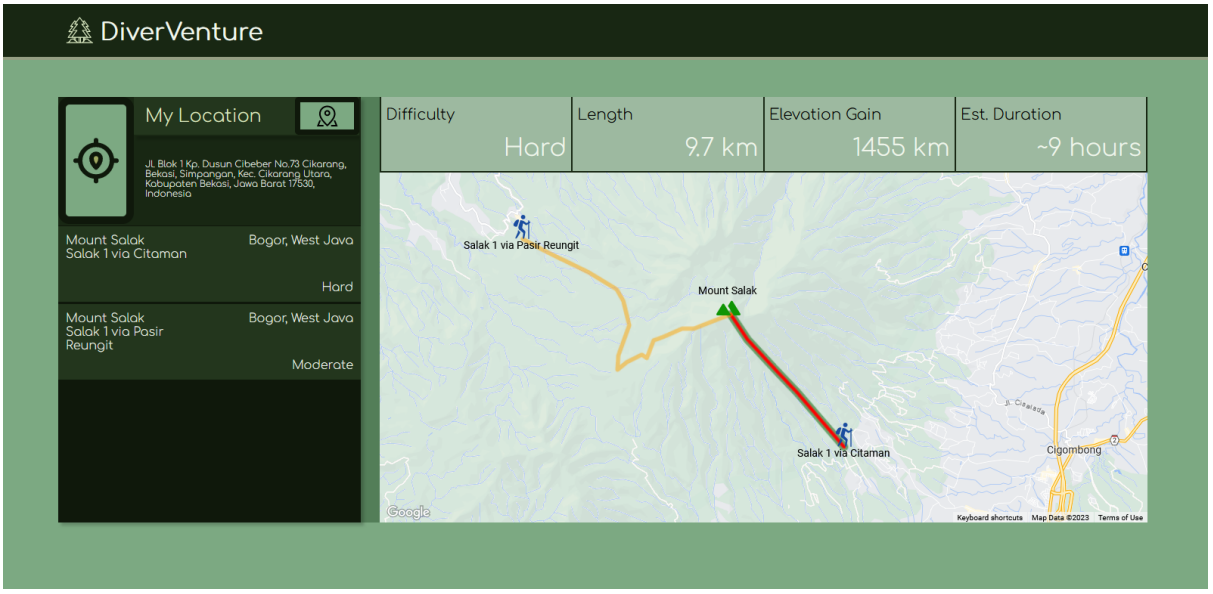
1. Current Location Component:
    - Positioned in the center of the screen, there is a component that displays your current location.
    - On the left side of this component, there is a button that allows you to obtain your current location.
    - On the top right corner, there is a button that enables you to input a custom location on the map.

2. Routes List Component:
    - Located on the side of the screen, there is a component that lists all available routes.
    - Each route in the list is clickable and selecting a route will pan the map to that specific trail, making it active.
    - This component allows you to preview how each route looks and select the one you want to explore.

3. Map Component:
    - The main part of the application consists of a map component that occupies the majority of the screen.
    - At the top of the map, when a route is selected, detailed information about the route is displayed, including its difficulty, length, and other relevant details.
    - The map itself is positioned in the lower section of the screen, occupying the bottom portion.

- On the map, there are three markers:
  - The first marker, represented by a mountain icon, indicates the location of the mountain. When clicked, it displays all the trail lines and geometry within that mountainous area.
  - The other two markers represent the starting points of different routes. When clicked, they reveal the trail lines and geometry leading to the summit or endpoint of that specific route.

This layout provides users with a clear and organized view of the hiking trails and relevant information. The combination of the map, route selection, and current location functionality enables users to easily explore and plan their hiking adventures.



❖ Database

Table places

	id [PK] integer	name character varying (64)	province character varying (64)	city character varying (64)	location geometry
1	1	Mount Salak	West Java	Bogor	0101000020E61000001511D1BBF1AE5A4097ACAC83D9DC1AC0

Table routes

	id [PK] integer	name character varying (100)	distance numeric	difficulty text	description text	route_geom geometry	place_id integer	duration interval	elevation numeric
1	1	Salak 1 via Citaman	9.7	Hard	A scenic hiking trail with challenging route	0105000000010000...	1	09:00:00	1455
2	2	Salak 1 via Pasir Reungit	12.3	Moderate	Thrilling hike through Gunung Salak's Pasir Reungit tra...	0105000020E61000...	1	10:00:00	1210

❖ Codes

./Client

App.js

```
import React, {Fragment} from 'react';
import './App.css';

import Header from "./components/header.js"
import Body from "./components/Parent.js"

function App() {
  return (
    <Fragment>
      <div className='main-container'>
        <Header />
        <Body/>
      </div>
    </Fragment>
  );
};

export default App;
```

Header.js

```
import React, { Fragment } from "react";
import './header.css';

//Render the header with title and logo
const Header = () => {
    return (
        <Fragment>
            <div className="top-bar">
                <svg
                xmlns="http://www.w3.org/2000/svg"
                className="title-logo"
```

```
                version="1.1"
                viewBox="0 0 128 128"
                xmlSpace="preserve"
                >
                <path d="M122.4
112.5h-4.1c-1.3-4.3-5.1-7.4-9.6-7.6-2.2-5-7-8.2-12.5-8.2-2.1 0-4 .5-5.8
1.3v-4.3h24.2c.7 0 1.4-.4 1.7-1 .3-.6.3-1.4-.2-2l-16.4-22h11.8c.7 0
1.4-.4 1.7-1.1.3-.6.2-1.4-.2-2L97.6 46.2h6.4c.7 0 1.4-.4 1.7-1
.3-.6.2-1.4-.2-2l-18-23.9-.1-.1-.1-.1-.2-.2H87c-.1 0-.1-.1-.2-.1 0 0-.1
0-.1-.1-.1 0-.1-.1-.2-.1h-1.2c-.1 0-.1 0-.2.1 0 0-.1 0-.1.1-.1
0-.1.1-.2.1h-.1l-.2.2-.1.1-.1.1-2.8 3.7L68
5.1l-.1-.1-.1-.1-.2-.2h-.1c-.1-.1-.1-.1-.2-.1 0 0-.1 0-.1-.1-.1
0-.1-.1-.2-.1H66.1 66c-.1 0-.1 0-.2.1h-.1c-.1
0-.2.1-.2.1h-.1l-.2.2-.1.1-.1.1-14.4
19.1-3.7-4.8-.1-.1-.1-.1-.2-.2h-.1c-.1 0-.1-.1-.2-.1 0 0-.1 0-.1-.1-.1
0-.1-.1-.2-.1h-1.2c-.1 0-.1 0-.2.1 0 0-.1 0-.1.1-.1
0-.1.1-.2.1h-.1l-.2.2-.1.1-.1.1-17.8 23.9c-.4.6-.5 1.3-.2 2 .3.6 1 1
1.7 1h6.4L18.4 65.8c-.4.6-.5 1.3-.2 2 .3.6 1 1.1 1.7 1.1h11.8l-16.4
22c-.4.6-.5 1.3-.2 2 .3.6 1 1 1.7 1H41V98c-1.7-.7-3.5-1.1-5.4-1.1-6.4
0-11.9 4.5-13.3 10.6-2.4 1-4.4 2.8-5.5 5.2h-6.1c-1 0-1.9.8-1.9 1.9 0 1
.8 1.9 1.9 1.9h111.8c1 0 1.9-.8 1.9-1.9-.1-1.2-.9-2.1-2-2.1zm-35.7
0h-4.9c.8-.9 1.9-1.6 3.2-1.9.7-.2 1.2-.8 1.4-1.5.7-4.9 4.9-8.5 9.8-8.5
4.4 0 8.2 2.8 9.5 6.9.3.8 1.1 1.4 1.9 1.3H108.2c2.7 0 5 1.6 6.1
3.9H86.7zm-15.2 0V93.8H81.6V108c-1.8 1-3.3 2.6-4.2 4.5h-5.9zm-6.1
0v-19h2.4v19h-2.4zm-7.7
0c-1.1-3.7-4.1-6.5-7.8-7.3V93.8h11.7v18.7h-3.9zm-11.6 0H21.2c.8-.9
1.9-1.6 3.2-1.9.7-.2 1.2-.8 1.4-1.5.7-4.9 4.9-8.5 9.8-8.5 4.4 0 8.2 2.8
9.5 6.9.3.8 1.1 1.4 1.9 1.3H47.6c2.7 0 5 1.6 6.1
3.9h-7.6zm-9.2-44.4c.4-.6.5-1.3.2-2-.3-.6-1-1-1.7-1H23.7l15.4-19.6c.4-.
6.5-1.3.2-2-.3-.6-1-1.1-1.7-1.1h-6.5l14.3-18.9 2.8 3.7-4.8 6.3c-.4.6-.5
1.3-.2 2 .3.6 1 1 1.7 1h8.4L34.5 60.6c-.4.6-.5 1.3-.2 2 .3.6 1 1.1 1.7
1.1h14.8L31.1 90.1H20.5l16.4-22zM66.5 9.3l17.8 23.5h-8.5c-.7
0-1.4.4-1.7 1.1-.3.6-.2 1.4.2 2L93.2 60H78.6c-.7 0-1.4.4-1.7 1-.3.6-.3
1.4.2 2l20 26.8H36L56
63c.4-.6.5-1.3.2-2-.3-.6-1-1-1.7-1H39.8l18.9-24.1c.4-.6.5-1.3.2-2-.3-.6
-1-1.1-1.7-1.1h-8.5L66.5 9.3zM86 23.6l14.3 18.9h-6.5c-.7 0-1.4.4-1.7
1.1-.3.6-.2 1.4.2 2l15.4 19.6H96.1c-.7 0-1.4.4-1.7 1-.3.6-.3 1.4.2
2l16.4 22h-9L82.3 63.7h14.8c.7 0 1.4-.4
1.7-1.1.3-.6.2-1.4-.2-2l-18.9-24h8.4c.7 0 1.4-.4 1.7-1
.3-.6.2-1.4-.2-2L84 26.2l2-2.6zm.7 77.1c-.5.5-.9 1-1.4
1.5v-8.4h1.4v6.9zm-41.9-6.9h1.4v7.9c-.4-.5-.9-1-1.4-1.4v-6.5z"></path>
                </svg>
```

```
                <h1 className ="title-text">DiverVenture</h1>
            </div>
        </Fragment>
    );
};

export default Header;
```

Parent.js (Body component)

```
import React, { useState } from 'react';
import Maps from './Maps';
import Details from './Details';
import CurrentLocation from './CurrentLoc';
import Cards from './Cards';

const ParentComponent = () => {
    const [currentSelectedRoute, setcurrentSelectedRoute] = useState(null);
    const [currentPutMarkerState, setCurrentPutMarkerState] = useState(null);
    const [currentLocationMarkerCoor, setCurrentLocationMarkerCoor] =
useState(null);
    const [currentSelectedCard, setcurrentSelectedCard] = useState(null);

    //when a route marker is clicked we store data passed from the marker in
currentSelectedRoute
    //then the data will be send to other components to show it
    const handleMarkerClick = (markerId) => {
        setcurrentSelectedRoute(markerId);
    };

    //when put marker button is selected we pass the data to Maps component
    //to enable/disable custom location marker placement
    const handlePutMarkerButton = (state) => {
        setCurrentPutMarkerState(state);
    };

    //when a card is clicked we pass the id of that card
    //and pass it to Maps to enable the marker
    const handleClickedCard = (cardId) => {
        setcurrentSelectedCard(cardId);
    };


    return (
```

```
        <div className='content-container-col'>
            <div className='content-container-row'>
                <div className='left-container-col'>
                    <CurrentLocation
handlePutMarkerState={handlePutMarkerButton}
currentLocationCoor={currentLocationMarkerCoor} />
                    {/* <div className='search-bar-container'>
                        <input type="text"
className="search-input-text-field" placeholder="Search..."></input>
                    </div> */}
                    <div className='cards-container'>
                        <Cards clickedCard={handleClickedCard} />
                    </div>
                </div>
                <div className='right-container-col'>
                    {currentSelectedRoute && (
                        <div className='details-container'>
                            <Details detailName={`Difficulty`}
details={`${currentSelectedRoute[0].difficulty}`} />
                            <Details detailName={`Length`}
details={`${currentSelectedRoute[0].distance} km`} />
                            <Details detailName={`Elevation Gain`}
details={`${currentSelectedRoute[0].elevation} km`} />
                            <Details detailName={`Est. Duration`}
details={`~${currentSelectedRoute[0].duration.hours} hours`} />
                        </div>
                    )}
                    <Maps onMarkerClick={handleMarkerClick}
putMarkerState={currentPutMarkerState}
locationMarkerCoor={setCurrentLocationMarkerCoor}
currentSelectedCard={currentSelectedCard}/>

                </div>
            </div>
            <div className='bottom-container'>
            </div>
        </div>
    );
};

export default ParentComponent;
```

CurrentLoc.js

```
import React, { useEffect, useState } from 'react';
```

```
// Function to get place data from coordinates using the Geocoder API
function getPlaceDataFromCoordinates(lat, lng, callback) {
  const geocoder = new window.google.maps.Geocoder();
  const latLng = new window.google.maps.LatLng(lat, lng);

  geocoder.geocode({ location: latLng }, (results, status) => {
    if (status === window.google.maps.GeocoderStatus.OK) {
      if (results.length > 0) {
        const placeData = results[0];
        callback(placeData);
      }
    } else {
      console.error('Geocoder failed due to: ' + status);
    }
  });
}

// CurrentLocation component
const CurrentLocation = ({handlePutMarkerState , currentLocationCoor})
=> {
  const [currentLocation, setCurrentLocation] = useState(null)
  const [currentCoor, setCurrentCoor] = useState(null)
  const [putMarkerState, setPutMarkerState] = useState(false)

  // Fetch place data from coordinates when currentLocationCoor prop
changes
  useEffect(() => {
    if (currentLocationCoor !== null && window.google) {
      getPlaceDataFromCoordinates(
        currentLocationCoor.lat,
        currentLocationCoor.lng,
        setCurrentLocation
      );
      setCurrentCoor([currentLocationCoor.lat,
currentLocationCoor.lng]);
    }
  }, [currentLocationCoor]);

  // Fetch place data from coordinates when currentCoor state changes
  useEffect(() => {
    if (currentCoor !== null && window.google) {
```

```
      getPlaceDataFromCoordinates(currentCoor[0], currentCoor[1],
setCurrentLocation);
    }
  }, [currentCoor]);

  // Call the handlePutMarkerState prop with putMarkerState value on
every render
  useEffect(() =>{
    handlePutMarkerState(putMarkerState);
  })

  // Get the current location using the Geolocation API
  const getLocation = () => {
    if (navigator.geolocation) {
      navigator.geolocation.getCurrentPosition(showPosition);
    } else {
      console.log("Geolocation is not supported by this browser.");
    }
  };

  // Callback function to handle the retrieved current position
  const showPosition = (position) => {
    const { latitude, longitude } = position.coords;
    setCurrentCoor([latitude, longitude]);
  };

  return (
    <div className="current-loc-container">
      <button className="current-loc-button" onClick={getLocation}>
        <svg
          xmlns="http://www.w3.org/2000/svg"
          width="800"
          height="800"
          fill="#000"
          className="gpsIcon"
          data-name="Flat Line"
          viewBox="0 0 24 24"
        >
          <path
            fill="var(--color-3)"
            strokeWidth="2"
            d="M14 11a2 2 0 00-4 0c0 2 2 4 2 4s2-2 2-4z"
          ></path>
```

```
          <path
            fill="none"
            stroke="var(--color-1-dark)"
            strokeLinecap="round"
            strokeLinejoin="round"
            strokeWidth="2"
            d="M12 3v2m9 7h-2m-7 9v-2m-9-7h2m9-1a2 2 0 00-4 0c0 2 2 4 2
4s2-2 2-4zm-2-6a7 7 0 107 7 7 7 0 00-7-7z"
          ></path>
        </svg>
      </button>
      <div className="current-loc-texts-container">
        <div className="current-loc-text-label-container" style={{
backgroundColor: 'var(--color-1-light)' }}>
          <div className="current-loc-label-text-container">
            <span className="current-loc-label-text">My Location</span>
          </div>
          <button className="current-loc-put-marker-button"
          style={putMarkerState ? {
            backgroundColor: 'var(--color-3)',
            borderColor: 'var(--color-4)',
          } : null}
          onClick={() =>setPutMarkerState(!putMarkerState)}>
            <svg
              xmlns="http://www.w3.org/2000/svg"
              width="800"
              height="800"
              viewBox="0 0 1024 1024"
              className="mapSelectIcon"
            >
              <path d="M800 416a288 288 0 10-576 0c0 118.144 94.528
272.128 288 456.576C705.472 688.128 800 534.144 800 416zM512
960C277.312 746.688 160 565.312 160 416a352 352 0 01704 0c0
149.312-117.312 330.688-352 544z"></path>
              <path d="M512 448a64 64 0 100-128 64 64 0 000 128zm0
64a128 128 0 110-256 128 128 0 010 256zm345.6 192L960
960H672v-64H352v64H64l102.4-256h691.2zm-68.928 0H235.328l-76.8
192h706.944l-76.8-192z"></path>
            </svg>

          </button>
        </div>
```

```
        <div className="current-loc-text-container"><span
className="current-loc-text">{currentLocation ? (
          <p>{currentLocation.formatted_address}</p>
        ) : (
          <p>Loading...</p>
        )}</span></div>
      </div>
    </div>
  );
};

export default CurrentLocation;
```

Cards.js

```
import React, { useState, useEffect } from 'react';

const Cards = ({ clickedCard }) => {
    const [routes, setRoutes] = useState(null);

     // Fetch routes data from the server
    const getRoutes = async () => {
        try {
            const response = await
fetch("http://localhost:5000/routes");
            const jsonData = await response.json();
            setRoutes(jsonData);
        } catch (err) {
            console.error(err.message);
        }
    }

    // Handle click event on a card and pass the id to the parent
component
    const handleOnClick = (id) => {
        clickedCard(id)
    }

     // Fetch routes data when the component mounts/start
```

```
    useEffect(() => {
        getRoutes();
    }, []);

    return (<div>
        {routes && routes.map(route => {
            return (<div key={route.route_id} className='card'
onClick={() => handleOnClick(route.route_id)}>
                <div className='card-details-left'>
                    <div className='card-details-top-left'>
                        <span
className='card-text'>{route.place_name}</span>
                    </div>
                    <div className='card-details-bot-left'>
                        <span
className='card-text'>{route.route_name}</span>
                    </div>
                </div>
                <div className='card-details-right'>
                    <div className='card-details-top-right'>
                        <span className='card-text'>{`${route.city},
${route.province}`}</span>
                    </div>
                    <div className='card-details-bot-right'>
                        <span
className='card-text'>{route.difficulty}</span>
                    </div>
                </div>
            </div>

            );
        })}
    </div>
    )

}

export default Cards;
```

Details.js

```
import React from 'react';

// The details component represents a single detail item.
// it displays a detail name and its corresponding details.
const Details = ({ detailName, details }) => {
    return (
        <div className='detail-container'>
            <div className='detail-name'>
                <span className='detail-name-text'>{detailName}</span>
            </div>
            <div className='detail'>
                <span className='detail-text'>{details}</span>
            </div>
        </div>
    );
};

export default Details;
```

Map.js

```
import { useState, useEffect, useMemo, useCallback } from 'react';
import { GoogleMap, LoadScript, MarkerF, PolylineF, DirectionsService,
DirectionsRenderer } from '@react-google-maps/api';
import mountainIcon from '../icons/mountains.png';
import hikeIcon from '../icons/hike.png';

const containerStyle = {
    width: '100%',
    height: '100%',
};

const Maps = ({ onMarkerClick, putMarkerState, locationMarkerCoor,
currentSelectedCard }) => {

    // Define the center of the map using the useMemo hook
    const center = useMemo(() => ({ lat: -6.742540, lng: 106.757082 }), []);
```

```
    // Define the options for the map using the useMemo hook
    const options = useMemo(() => ({
        mapId: 'ae4b64d137d842b3',
        disableDefaultUI: true,
        clickableIcons: false,
    }), []);

    // State variables for routes, places, route, location marker,
directions, directions status,
    // selected marker ID, selected place marker, place name, distance, and
duration
    const [routes, setRoutes] = useState([]);
    const [places, setPlaces] = useState([]);
    const [route, setRoute] = useState(null);
    const [locationMarker, setLocationMarker] = useState(null);
    const [directions, setDirections] = useState(null);
    const [directionsStatus, setDirectionsStatus] = useState(null);
    const [selectedMarkerId, setSelectedMarkerId] = useState(null);
    const [selectedPlaceMarker, setSelectedPlaceMarker] = useState(null);
    const [placeName, setPlaceName] = useState('');
    const [distance, setDistance] = useState(null);
    const [duration, setDuration] = useState(null);

    // Function to fetch all routes from the server
    const getRoutes = async () => {
        try {
            const response = await fetch("http://localhost:5000/routes");
            const jsonData = await response.json();
            setRoutes(jsonData);
        } catch (err) {
            console.error(err.message);
        }
    }

    // Function to fetch all places from the server
    const getPlaces = async () => {
        try {
            const response = await fetch("http://localhost:5000/place");
            const jsonData = await response.json();
            setPlaces(jsonData);
        } catch (err) {
            console.error(err.message);
        }
    }

    // Function to fetch a specific route by ID from the server
    const getRoutesId = async (id) => {
```

```
        try {
            const response = await
fetch(`http://localhost:5000/routes/${id}`);
            const jsonData = await response.json();
            setRoute(jsonData);
        } catch (err) {
            console.error(err.message);
        }
    }

    // Fetch places and routes at the start
    useEffect(() => {
        getPlaces();
        getRoutes();
    }, []);

    // Set the selected marker ID when the currentSelectedCard changes
    useEffect(() => {
        if (currentSelectedCard !== null) {
            setSelectedMarkerId(currentSelectedCard);
        }
    }, [currentSelectedCard])

    // Fetch the route data when the selectedMarkerId changes
    useEffect(() => {
        if (selectedMarkerId !== null) {
            getRoutesId(selectedMarkerId);
        }
    }, [selectedMarkerId]);

    // Perform actions when putMarkerState, locationMarker, route change
    // locationMarker and route will be passed to the parent component
    useEffect(() => {
        console.log(putMarkerState);
        locationMarkerCoor(locationMarker);
        onMarkerClick(route);
    })

    // Update distance, duration, and place name when directions and
directionsStatus change
    useEffect(() => {
        if (directions && directionsStatus === 'OK') {
            // Extract distance and duration from directions response
            const leg = directions.routes[0].legs[0];
            const endAddress = directions.routes[0].legs[0].end_address;
            // Set the extracted place name, distance, and duration
            setPlaceName(endAddress);
            setDistance(leg.distance.text);
```

```
            setDuration(leg.duration.text);
        }
    }, [directions, directionsStatus]);

    // Function to handle map click event
    const handleMapClick = (event) => {
        // Extract the latitude and longitude coordinates of the clicked
location
        const clickedLat = event.latLng.lat();
        const clickedLng = event.latLng.lng();

         // Check if the putMarkerState is true (when trying to make a custom
location)
        if (putMarkerState) {
            // Clear the existing location marker if it exists
            if (locationMarker) {
                setLocationMarker(null);
            }
            // Create a new marker object with the clicked coordinates
            const newMarker = {
                lat: clickedLat,
                lng: clickedLng,
            };
            // Set the new location marker with the clicked coordinates
            setLocationMarker(newMarker);
            console.log(locationMarker);
        } else {
            // Clicking on the map will
            // Reset the route, selected marker ID, selected place marker,
location marker, and directions status
            setRoute(null)
            setSelectedMarkerId(null)
            setSelectedPlaceMarker(null);
            setLocationMarker(null)
            setDirectionsStatus(null)
        }
        console.log('Clicked coordinates:', clickedLat, clickedLng);
    };

    // Function to handle marker click events
    const markerOnClick = useCallback(
        (data) => {
            setSelectedMarkerId(data);
            setDirections(null);
        },
        [setSelectedMarkerId]
    );
```

```
    // Function to handle place(mountain) marker click events
    const placeMarkerOnClick = useCallback(
        (data) => {
            setSelectedPlaceMarker(data);
        },
        [setSelectedPlaceMarker]
    );

        console.log(placeName);
    return (<>
        <div className='map-container'>
            <LoadScript googleMapsApiKey="API_KEY">
                <GoogleMap
                    mapContainerStyle={containerStyle}
                    center={center}
                    zoom={12}
                    options={options}
                    onClick={handleMapClick}
                >
                    {directions && directionsStatus === 'OK' && (
                        <DirectionsRenderer directions={directions} />
                    )}

                    {locationMarker && (
                        <MarkerF
                            position={{ lat: locationMarker.lat, lng:
locationMarker.lng }}
                        />
                    )}
                    {places.map(place => {
                        // console.log(place.location_coor[0]);
                        return (
                            <MarkerF key={`${place.id}-p`}
position={place.location_coor[0]} icon={{
                                url: mountainIcon,
                                labelOrigin: new window.google.maps.Point(15,
-1)
                            }}
                                label={place.name} onClick={() =>
placeMarkerOnClick(place.id)} />
                        );
                    })}
                    {routes.map(route => {
                        const firstLocation = route.route_geometry[0];
                        console.log('test')

                        return (
```

```
                            <MarkerF key={route.route_id}
position={firstLocation} icon={{
                                url: hikeIcon,
                                labelOrigin: new window.google.maps.Point(15,
40)
                            }}
                                label={route.route_name}
                                onClick={() => markerOnClick(route.route_id)}
/>
                        );
                    })}
                    {route && route.map(route => {

                        let mainPolylineOptions = {
                            strokeColor: 'white',
                            zIndex: -1,
                            strokeOpacity: 1,
                            strokeWeight: 3,
                        };
                        let borderPolylineOptions = {
                            strokeColor: 'darkgreen',
                            zIndex: -2,
                            strokeOpacity: 0.5,
                            strokeWeight: 10,
                        };

                        if (route.difficulty === 'Easy') {
                            mainPolylineOptions.strokeColor = 'green';
                        } else if (route.difficulty === 'Moderate') {
                            mainPolylineOptions.strokeColor = 'orange';
                        } else if (route.difficulty === 'Hard') {
                            mainPolylineOptions.strokeColor = 'red';
                        }
                        return (<>
                            <PolylineF key={`${route.route_id}-b`}
path={route.route_geometry} options={borderPolylineOptions} />
                            <PolylineF key={route.route_id}
path={route.route_geometry} options={mainPolylineOptions} />
                        </>
                        )
                    })}

                    {selectedPlaceMarker && routes
                        .filter(route => route.place_id ===
selectedPlaceMarker)
                        .map(route => {
                            let polylineOptions = {
                                strokeColor: 'white',
```

```
                                zIndex: -1,
                                strokeOpacity: 0.5,
                                strokeWeight: 5,
                            };
                            if (route.difficulty === 'Easy') {
                                polylineOptions.strokeColor = 'green';
                            } else if (route.difficulty === 'Moderate') {
                                polylineOptions.strokeColor = 'orange';
                            } else if (route.difficulty === 'Hard') {
                                polylineOptions.strokeColor = 'red';
                            }
                            return <PolylineF key={route.id}
path={route.route_geometry} options={polylineOptions} />
                        }
                        )}
                    {directionsStatus !== 'OK' && route && locationMarker &&
(
                        <DirectionsService
                            options={{
                                destination: route[0].route_geometry[0],
                                origin: locationMarker,
                                travelMode: 'DRIVING',
                            }}
                            callback={(response, status) => {
                                if (status === 'OK') {
                                    setDirections(response);
                                } else {
                                    console.log('Directions request failed:',
status);
                                }
                                setDirectionsStatus(status);
                            }}
                        />
                    )}

                </GoogleMap>
            </LoadScript>
        </div>
    </>
    );
};

export default Maps;
```

./server

---

db.js

```
const Pool = require("pg").Pool;

//set up the database
const pool = new Pool({
    user: "postgres",
    password: "1234",
    host: "localhost",
    port: "5432",
    database: "GIS"
})

module.exports = pool;
```

---

index.js

```
const express = require("express");
const app = express();
const cors = require("cors");
const pool = require("./db");

//middleware
app.use(cors());
app.use(express.json());

//get routes
app.get("/routes", async (req, res) => {
    try {
        const allRoutes = await pool.query(`
        SELECT
        route_id,
        route_name,
        distance,
        difficulty,
```

```
        duration,
        elevation,
        description,
        place_id,
        place_name,
        city,
        province,
        json_agg(json_build_object('lat',ST_X(coordinate) , 'lng',
ST_Y(coordinate))) AS route_geometry
        FROM
        (
            SELECT r.id AS route_id,
            p.id AS place_id,
            r.name AS route_name,
            r.distance AS distance,
            r.difficulty AS difficulty,
            r.description AS description,
            r.duration AS duration,
            r.elevation AS elevation,
            (ST_DumpPoints(r.route_geom)).geom AS coordinate,
            p.name AS place_name,
            p.province AS province,
            p.city AS city,
            (ST_DumpPoints(p.location)).geom AS location_coor
            FROM routes r
            JOIN place p ON r.place_id = p.id
        ) AS subquery
        GROUP BY
        route_id,
        route_name,
        distance,
        difficulty,
        duration,
        elevation,
        description,
        place_id,
        place_name,
        city,
        province;
        `);
        res.json(allRoutes.rows);
    } catch (err) {
        console.error(err.message);
```

```
    }
});

//get a route
app.get("/routes/:id", async (req, res) => {
    try {
        const {id} = req.params;
        const route = await pool.query(`
        SELECT
        route_id,
        route_name,
        distance,
        difficulty,
        duration,
        elevation,
        description,
        place_id,
        place_name,
        city,
        province,
        json_agg(json_build_object('lat',ST_X(coordinate) , 'lng',
ST_Y(coordinate))) AS route_geometry
        FROM
        (
            SELECT r.id AS route_id,
            p.id AS place_id,
            r.name AS route_name,
            r.distance AS distance,
            r.difficulty AS difficulty,
            r.duration AS duration,
            r.elevation AS elevation,
            r.description AS description,
            (ST_DumpPoints(r.route_geom)).geom AS coordinate,
            p.name AS place_name,
            p.province AS province,
            p.city AS city,
            (ST_DumpPoints(p.location)).geom AS location_coor
            FROM routes r
            JOIN place p ON r.place_id = p.id
            WHERE r.id = $1
        ) AS subquery
        GROUP BY
        route_id,
```

```
        route_name,
        distance,
        difficulty,
        duration,
        elevation,
        description,
        place_id,
        place_name,
        city,
        province;
        `, [id]);
        res.json(route.rows);
    } catch (err) {
        console.error(err.message);
    }
});

//get places
app.get("/place", async (req, res) => {
    try {
        const place = await pool.query(`
        SELECT
        id,
        name,
        province,
        city,
        json_agg(json_build_object('lat',ST_Y(coordinate) , 'lng',
ST_X(coordinate))) AS location_coor
        FROM
        (
            SELECT
            id,
            name,
            province,
            city,
            (ST_DumpPoints(location)).geom AS coordinate
            FROM
            place
        ) AS subquery
        GROUP BY
        id,
        name,
        province,
```

```
        city;
        `);
        res.json(place.rows);
    } catch (err) {
        console.error(err.message);
    }
});

//start the server on port 5000
app.listen(5000, () => {
    console.log("server has started on port 5000");
});
```

❖ Refs

- Similar website
  https://www.alltrails.com/
- Javascript, HTML, CSS tutorials on youtube

- PERN Stack Course
  PERN Stack Course - Postgres, Express, React, and Node
- React Documentations

https://react.dev/learn

- Google Maps Platform

https://console.cloud.google.com/google/maps-apis/discover?project=striking-ruler-384904

- React Google Maps API Documentations

https://react-google-maps-api-docs.netlify.app/

- Data taken from

https://www.radarbogor.id/2023/02/04/6-jalur-pendakian-gunung-salak-pasir-reungit-banyak-panorama/

- Free SVGs

https://www.svgrepo.com/