Objectives of Our Mini-Geoportal

- To provide users with an interactive platform for geospatial exploration using web maps powered by the Leaflet library.
- To offer users the ability to switch between multiple base maps such as OpenStreetMap, Satellite Imagery, and OpenCycleMap.
- To display real-time environmental data layers like soil moisture, cloud cover, wind direction, air quality, and weather information through APIs.
- To present detailed geospatial datasets of Nepal, including rivers, hydropower plants, transmission lines, airports, and administrative boundaries.
- To allow users to upload their own data, generate a buffer around it, and download the results as a .geojson file or shapefile for further analysis.

Front-End

1. HTML (HyperText Markup Language)

- HTML is the backbone of your website, providing the structure and content.

- It defines the layout of the webpage, including the navigation bar, welcome section, map containers, and footer.

- It Embeds external resources like CSS, JavaScript, and iframes.

- Uses semantic elements like <header>, <nav>, <section>, and <footer> for better readability and content management.

2. CSS (Cascading Style Sheets)

- CSS is used for styling and layout, ensuring the website is visually appealing and responsive.

- The Custom Styles define the appearance of elements like the navigation bar, buttons, map containers, and text.

- Responsive Design: Uses @media queries to ensure the website adapts to different screen sizes.

- Bootstrap Integration: Combines custom CSS with Bootstrap's pre-defined classes for consistent styling.
- Animations and Effects: Adds hover effects (e.g., button hover) and transitions for interactive elements.

3. JavaScript

- JavaScript adds interactivity and dynamic behavior to the website.

- Leaflet.js: Powers the interactive map with layers, markers, and popups.

- API Integration: Fetches real-time data (e.g., soil moisture, cloud cover, wind direction, air quality, and weather) from external APIs like Open-Meteo and OpenWeather.

- Event Handling: Listens for user interactions (e.g., map clicks) and updates the UI dynamically.

- DOM Manipulation: Dynamically updates the content of popups and layers based on user input or API responses.

4. Bootstrap

- Bootstrap is a front-end framework that simplifies responsive design and provides pre-built UI components.

- Navigation Bar: Uses Bootstrap's navbar component for a responsive and mobile-friendly navigation menu.

- Grid System: Organizes content into rows and columns for a structured layout.

- Utility Classes: Applies Bootstrap's utility classes (e.g., py-4, text-center, container-fluid) for spacing, alignment, and responsiveness.

- JavaScript Components: Utilizes Bootstrap's JavaScript plugins (e.g., navbar toggler) for interactive elements.

5. Leaflet.js

- Leaflet is a lightweight JavaScript library for interactive maps.

- Base Layers: Integrates multiple base layers (OpenStreetMap, Satellite Imagery, OpenCycleMap) for users to switch between.

- Overlay Layers: Displays dynamic data layers (e.g., soil moisture, cloud cover, wind direction) fetched from APIs.

- Markers and Popups: Adds custom markers with popups to display real-time data at specific locations.

- Layer Control: Provides a layer control panel for users to toggle layers on and off.

6. APIs (Application Programming Interfaces)

- APIs fetch real-time data from external sources and integrate it into the website.

- Open-Meteo API: Retrieves environmental data like soil moisture, cloud cover, and wind direction.

- OpenWeather API: Fetches weather and air quality data (e.g., temperature, humidity, PM2.5, PM10).

- Fetch API: Used in JavaScript to make HTTP requests to these APIs and process the responses.

7. Fonts and Icons

- Enhances the visual appeal and readability of the website.

- Google Fonts: Uses Arial, sans-serif and Georgia, 'Times New Roman', Times, serif for typography.

- Custom Icons: Uses custom icons (e.g., portal.png, soilmoist.png, cloud.png, wind.png, weathpollution.png) for markers and branding.

8. Favicon

- Provides a small icon that represents the website in browser tabs and bookmarks.

- The favicon is linked in the <head> section using <link rel="icon" href="favicon_name.png" type="image/png">.

9. Iframes

- Embeds external content or additional web pages within the website.

- Embeds another web map using an <iframe>.

10. Responsive Design

- Ensures the website works seamlessly across different devices and screen sizes.

- Uses Bootstrap's grid system and responsive utilities.

- Sets widths and heights using relative units like vw (viewport width) and vh (viewport height).

- Adjusts layout and font sizes for smaller screens.

11. Custom JavaScript Functions

- Adds custom functionality to the website.

- Refresh Function: Reloads the page when the "Home" link in the navbar is clicked.

- Dynamic Data Fetching: Fetches and displays real-time data based on user interactions (e.g., map clicks).

12. Turf Library

- Turf.js is a geospatial analysis library used in your application to perform spatial operations on GeoJSON data.

- It allows the users to Determine spatial relationships (e.g., point-in-polygon, intersection) between features, filter and display only the relevant features on the map and enhance user interaction by providing location-based filtering capabilities.

>Key Turf.js Functions Used

Some of the Turf.js functions used in our application are as:

a. turf.point

- Converts a set of coordinates (like longitude and latitude) into a point on the map.

- When given the coordinates and it creates a point feature that can be used for spatial analysis.

- Syntax:

turf.point([longitude, latitude], { properties });

b. turf.feature

- Converts a shape (like a polygon or line) into a GeoJSON feature.

- When given provide the shape (geometry) and optional properties, and it creates a feature that Turf.js can work with.

- Syntax:

turf.feature(geometry, { properties });

c. turf.booleanPointInPolygon

- Checks if a point is inside a polygon.

- When given a point and a polygon, and it tells you whether the point is inside the polygon (true or false).

- Syntax:

turf.booleanPointInPolygon(point, polygon);

d. turf.intersect

- Finds the overlapping area between two shapes.

- When provided two shapes (like polygons or lines), and it returns the area where they overlap. If they don’t overlap, it returns null.

- Syntax:

turf.intersect(shape1, shape2);

>Spatial Operations Performed Using Turf.js

Turf.js is used in out application to filter and analyze geographic data as:

a. Filtering Hydropower Projects by Province or District

- Filters hydropower projects based on whether they are inside a selected province or district.

How it works:

- Each hydropower project is treated as a point.
- The selected province or district is treated as a polygon.
- Turf.js checks if the hydropower point is inside the province or district polygon using turf.booleanPointInPolygon.
- Only the projects inside the selected area are shown on the map.

b. Filtering Index Sheets by District

- Filters index sheets based on whether they overlap with a selected district.

- How it works:

- Each index sheet is treated as a polygon.
- The selected district is also treated as a polygon.
- Turf.js checks if the index sheet polygon overlaps with the district polygon using turf.intersect.
- Only the index sheets that overlap with the district are shown on the map.

c. Filtering Transmission Lines by Province

- Filters transmission lines based on whether they overlap with a selected province.

- How it works:

- Each transmission line is treated as a line.
- The selected province is treated as a polygon.
- Turf.js checks if the transmission line overlaps with the province polygon using turf.intersect.
- Only the lines that overlap with the province are shown on the map.

d. Filtering Airports by Province

- Filters airports based on whether they are inside a selected province.

- How it works:

- Each airport is treated as a point.
- The selected province is treated as a polygon.
- Turf.js checks if the airport point is inside the province polygon using turf.booleanPointInPolygon.
- Only the airports inside the selected province are shown on the map.

Back-End

1. PostgreSQL

- While PostgreSQL is a backend database, it is mentioned in the context of providing geospatial datasets for download.
- The front end likely interacts with a backend service that queries PostgreSQL to fetch and serve geospatial data.
- Vector data was first uploaded and stored in PostgreSQL as tables, which were then fetched and displayed on the website using the Flask framework in Python.

2. Flask Framework (Python)

The structure and functionality of a Flask application is designed to:

- Download Shapefiles: Allow users to download shapefiles stored in a PostgreSQL database.
- Buffer Analysis: Enable users to upload shapefiles, perform buffer operations, and download the results.

Downloading Shapefiles

Functionality:

- Database Connection: The application connects to a PostgreSQL database using the pgsql2shp command-line tool.
- Shapefile Export: It exports shapefiles from specified tables in the database.
- Zipping and Downloading: The exported shapefiles are zipped and provided for download.

Key Components:

- Export Function:

```
def export_shapefile_from_db(output_dir, shapefile_name, table_name):
    # Exports shapefile from the database
```

- Download Route:

```
@app.route('/download-shapefile/<table_name>')
def download_shapefile(table_name):
    # Handles shapefile download requests
```

Buffer Analysis

Functionality

- File Upload: Users can upload zipped shapefiles.
- Buffer Operation: The application performs buffer analysis using geopandas.
- Result Download: The buffered geometries are saved as a GeoJSON file and made available for download.

Key Components:

- Buffer Analysis Function:

```
def perform_buffer_analysis(upload_folder, result_folder, shapefile, buffer_distance):
    # Performs buffer analysis and saves the result
```

- Upload and Buffer Route:

```
@app.route('/upload-buffer', methods=['POST'])
    def upload_buffer():
        # Handles buffer analysis requests
```

HTML Templates

index.html: Provides a user interface for downloading shapefiles.

buffer.html: Offers a form for users to upload shapefiles and specify buffer distances

Running the Application

To execute the application, run:

python app.py

The application will be accessible at http://localhost:5000.

Conclusion

This Flask application provides a straightforward interface for interacting with geospatial data, offering both download capabilities from a PostgreSQL database and buffer analysis on uploaded shapefiles. The modular design ensures ease of maintenance and potential expansion.

Limitations

When running the web map by directly opening the .html file in a browser using the file:// protocol, the map layers may not load due to **CORS (Cross-Origin Resource Sharing) restrictions**. Modern browsers enforce strict security policies to prevent web pages from accessing resources hosted on a different domain, protocol, or port unless explicitly permitted by the server. Since the map tiles and other resources are typically hosted on external servers (e.g., OpenStreetMap, Esri), these restrictions block the resources from loading when the file is accessed locally.

However, when the .html file is served through a local HTTP server (e.g., using VS Code's Live Server or Python's HTTP server), the browser uses the http:// protocol, which allows the external resources to load properly. This is because the server simulates a proper web environment where CORS policies are less restrictive. To resolve this issue, it is recommended to always run the web map through a local or

online HTTP server during development or deployment to ensure that all external resources load correctly.