Project Requirement Document

Project Overview

The project involves developing a spatial data platform that supports CRUD operations (Create, Read, Update, Delete) for spatial data such as points and polygons. The backend will handle the storage, retrieval, and management of this data as JSON objects, while the frontend will provide an interface for interacting with the spatial data on a map.

Key Features:

1. Backend API for managing spatial point data.
2. Backend API for managing spatial polygon data.
3. Frontend interface for displaying and interacting with the spatial data.

Backend Requirements

Technology Stack:

- **Backend Language**: Python with Django
- **Database**: JSON-based storage for spatial data (no PostGIS required)
- **API Framework**: Django REST Framework

Models:

1. **`PointData` Model:**
   - **Fields**:
     - `name` (CharField): The name of the point.
     - `location` (JSONField): The coordinates of the point as JSON.
   - **Example `location` JSON**:

```json
{
  "type": "Point",
  "coordinates": [-0.09, 51.505]
}
```

2. **`PolygonData` Model:**
   - **Fields**:
     - `name` (CharField): The name of the polygon.
     - `area` (JSONField): The coordinates of the polygon as JSON.
   - **Example `location` JSON**:

```json
{
  "type": "Polygon",
  "coordinates": [
    [[-0.09, 51.505], [-0.08, 51.505], [-0.08, 51.506], [-0.09, 51.506], [-0.09, 51.505]]
  ]
}
```

API Endpoints:

1. **Point Data Endpoints:**
   - `POST /api/points/`: Create a new point.
   - `GET /api/points/`: Retrieve all points.
   - `PUT /api/points/{id}/`: Update a specific point.
   - `DELETE /api/points/{id}/`: Delete a specific point.

2. **Polygon Data Endpoints:**

- `POST /api/polygons/`: Create a new polygon.
- `GET /api/polygons/`: Retrieve all polygons.
- `PUT /api/polygons/{id}/`: Update a specific polygon.
- `DELETE /api/polygons/{id}/`: Delete a specific polygon.

Changes Required in Backend:

1. ** Store spatial data as JSON `**:

- Store and process spatial data as plain JSON.
- Modify serializers to handle JSON data for both points and polygons.

2. **API Responses**:

- Ensure that all responses, especially for `GET` endpoints, return data as JSON objects in the appropriate format.

3. **Tests**:

- Example test data should be in plain JSON format, aligning with the new models.

Example API Responses:

- **GET /api/points/**:

```json
[
  {
    "id": 1,
    "name": "Point 1",
```

```
      "location": {
        "type": "Point",
        "coordinates": [-0.09, 51.505]
      }
    }
  ]
  ```
```

- **GET /api/polygons/**:
  ```json
  [
    {
      "id": 1,
      "name": "Polygon 1",
      "area": {
        "type": "Polygon",
        "coordinates": [
          [[-0.09, 51.505], [-0.08, 51.505], [-0.08, 51.506], [-0.09, 51.506], [-0.09, 51.505]]
        ]
      }
    }
  ]
  ```

  ```

Frontend Requirements

Technology Stack:

- **Mapping Library**: OpenLayers

Frontend Features:

1. **Display Points on the Map**:
   - Fetch all point data & polygon data from the `getSpatialData' endpoint.
   - Display each point as a marker and polygons on the map.

3. **Interactivity**:
   - Display additional information (e.g., `name`) when clicking on a point or polygon.
   - Update map dynamically when new data is created or existing data is modified.

Example Code Snippets:

- **Fetching and Displaying Points & Polygons**:

This will be triggered when user clicks on ***"Get Data"*** button in the dashboard page, which will fetch all the points and polygon from database and display it in the map.

```javascript
$.ajax({
    url: '  getSpatialData',
    method: 'GET',
    success: function(data) {
        var data = response["data"];
        addPoints(data["points_data"]);
```

```
            addPolygon(data["polygon_data"]);


        }
    });
    ```
```

**## **Assumptions****

- The spatial data platform will operate entirely with JSON data for both points and polygons.

- The frontend and backend communicate via ajax calls, with all spatial data passed as JSON.

- The frontend application will use OpenLayers for rendering map data.

---

**## **Conclusion****

This document outlines the necessary changes to implement a spatial data platform that handles point and polygon data as JSON. The backend will be adjusted to remove dependencies on GIS libraries, and the frontend will be updated to interact with the new API endpoints and render the spatial data using OpenLayers.

Note :-

One point with name Point 0 will be loaded in the map when the application is launched. Once "***Get Data"*** button is clicked in the UI it will fetch all the points and polygons stored in the database.
```

If the points or polygons data are updated it will also reflect in the UI since both these apps are integrated, and the UI points and polygons are fetched form the same database.

This project also have tests.py content which can be used to test the CRUD operations of spatial data.

I’ve also attached the README.md, request_payload.json file and the Postman api test collection with sample data in it for testing purpose.