Administrative Division

The administrative division module is a common API module used for remote data collection that provides information about geographical boundaries, such as countries, states, and municipalities. This module is particularly important for the output of the Remote Data Collection (e.g web-app that involve maps visualization and data filtering). By using the administrative division module, we can easily filter data by location and create interactive maps that highlight specific regions of interest.

One common use case is for creating a cascading dropdown menu or hierarchical form, where the options available in one dropdown depend on the selection made in the previous dropdown. For example, if a user selects "Indonesia" in the first dropdown, the second dropdown would populate with the province in the "Indonesia", and so on.

In short, the administrative division module is a crucial building block for many different types of remote data collection applications, and its accurate and up-to-date information can improve the quality and usefulness of many other API modules that rely on location-based data.

Requirements

When working with geographical data, it is often desirable for the administrative data to match with a shapefile. A shapefile is a popular file format for representing geographic data, and it consists of several files that together define the boundaries of the geographic features.

Ideally, the administrative data used in your API should match with the administrative boundaries defined in a shapefile. This is because shapefiles provide a standard format for representing geographic data and can be used by many different mapping and visualization.

Matching administrative data with a shapefile can also help to ensure the accuracy and consistency of your geographic data. Shapefiles are typically created and maintained by government agencies or other authoritative sources, and they are often updated regularly to reflect changes in administrative boundaries.

> In terms of requirements, including administrative data that matches with a shapefile may be necessary if we plan to use our API with mapping or visualization library that require geojson.

Here are the general steps you would need to follow to create a seeder for a shapefile:

1. Convert the shapefile to a format to geojson. There are several tools available for this, such as QGIS or mapshaper.
2. Write a script to read the geojson and populate your Administration table with the relevant data. This script should read the geojson and insert new rows into the Administration and Administration Level table for each administrative division, including its ID, parent ID (if applicable), and name.
3. Run the seeder script to populate your Administration table with the data from the geojson.

Models

The database schema you provided has four fields: `id`, `parent_id`, `name`, and `level`. The id field represents the unique identifier for each administrative division, while the `parent_id` field represents the ID of the parent division (if applicable). Finally, the name field stores the name of the division.

Here's an example of how you might define a **Django** model to represent this schema:

```
from django.db import models


class AdministrationLevel(models.Model):
    id = models.IntegerField(primary_key=True)
    name = models.CharField(max_length=255)

    def __str__(self):
        return self.name

class Administration(models.Model):
    id = models.IntegerField(primary_key=True)
    parent_id = models.IntegerField(null=True, blank=True)
    name = models.CharField(max_length=255)
    level = models.ForeignKey(AdministrationLevel, on_delete=models.CASCADE)

    def __str__(self):
        return self.name
```

> The **parent_id** field is defined with **null=True** and **blank=True**, which means that it can be empty or null.

By defining this model, we can easily query and manipulate administrative divisions in our Django application. For example, we might use the following code to retrieve all administrative divisions with a parent ID of 42:

```
parent_division = Administration.objects.get(id=42)
child_divisions = Administration.objects.filter(parent_id=parent_division.id)
```

This would retrieve all administrative divisions that have 42 as their parent ID and allow us to perform further processing.

API Endpoint

Here's an example of how we might define our API endpoints using the Django REST Framework:

Serializer

```
from rest_framework import serializers
from .models import AdministrationLevel, Administration

class AdministrationLevelSerializer(serializers.ModelSerializer):
    class Meta:
        model = AdministrationLevel
        fields = ('id', 'name')

class AdministrationSerializer(serializers.ModelSerializer):
    level = AdministrationLevelSerializer()

    class Meta:
        model = Administration
        fields = ('id', 'parent_id', 'name', 'level')
```

Views

```
from rest_framework import viewsets, routers
from .models import Administration
from .serializer import AdministrationSerializer

class AdministrationViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = Administration.objects.all()
    serializer_class = AdministrationSerializer
```

```
    def list(self, request, *args, **kwargs):
        # Optional filtering by administrative level
        level_id = self.request.query_params.get('level_id', None)
        if level_id is not None:
            self.queryset = self.queryset.filter(level__id=level_id)

        return super().list(request, *args, **kwargs)

router = routers.DefaultRouter()
router.register(r'administrations', AdministrationViewSet)
```

Results

```
[
    {
        "id": 2,
        "parent_id": 1,
        "name": "Indonesia",
        "level": {
            "id": 2,
            "name": "Country"
        }
    },
    {
        "id": 4,
        "parent_id": 1,
        "name": "Province",
        "level": {
            "id": 2,
            "name": "West Java"
        }
    }
]
```

Revision #7
Created 2023-05-10 03:29:27 UTC by Deden Bangkit
Updated 2023-05-10 06:54:38 UTC by Deden Bangkit