API Modules for MIS

This guide provides an overview of common API modules used for Remote Data Collection, including descriptions of their functions, how to use them, and examples of when they might be useful. This article is a valuable resource for understanding the various API modules available and how to make the most of them.

- [Administrative Division](#)
- [Administrative Category](#)
- [Administrative Entity](#)

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 <u>geojson</u>. There are several tools available for this, such as QGIS or <u>mapshaper</u>.
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:

```python
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()
```

```python
    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

```json
[
  {
    "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"
    }
  }
]
```

Administrative Category

The Administrative Category module is a way to categorize administrative divisions based on their type and hierarchy. Each administrative division can belong to one or more categories, which might include categories like habitation (e.g rural, urban), land condition (dry, wet). These categories can be used to filter and group data within the API.

Models

```
class AdministrationTypeCategory(models.Model):
    name = models.CharField(max_length=255)
    parent_category = models.ForeignKey('self',
        null=True,
        blank=True,
        related_name='children',
        on_delete=models.CASCADE)
    description = models.TextField()


class AdministrationCategories(models.Model):
    administration = models.ForeignKey(
        Administration,
        on_delete=models.CASCADE)
    administration_type_category = models.ForeignKey(
        AdministrationTypeCategory,
        on_delete=models.CASCADE)
```

Administration Type Category

This model represents the type of administrative division type, such as Rural or Urban. It might include fields like name or description. Example:

id	parent_id	name	description
1	null	Habitation	A geographic area in which to live
2	1	Rural	A geographic area that is located outside towns and cities

| 3 | 1 | Urban | A geographic area that is located in cities |

Administration Categories

This model represents the administrative categories for a given administrative entity. Example:

id	administration_id	administration_type_category_id
1	42	1

Administrative Entity

The Administrative Entity module is an extension of the Administrative Division module, and it provides a way to represent additional entities within an administrative boundary. An administrative entity is typically a geographic area that is governed by a specific administrative body or organization. For example, an administrative entity might be a city, a town, a village, or a neighborhood.

In the context of an API, the Administrative Entity module can be used to provide additional information about the entities within each administrative boundary. This information might include the name of the entity, its type (e.g. city, town, village), and any relevant metadata (e.g. population, area, number of schools).

The Administrative Entity module is typically used in conjunction with the Administrative Division module, as each administrative entity should be associated with the administrative division that contains it. This allows the API to provide a hierarchical view of the administrative boundaries, from the highest level (e.g. country) down to the lowest level (e.g. neighborhood).

One example use case for the Administrative Entity module is for adding schools and public facilities within an administrative entity. By associating each school or facility with the appropriate administrative entity, the API can provide a comprehensive view of the facilities available within each area, and allow users to filter or search for facilities by location.

Overall, the Administrative Entity module provides a way to represent additional entities within an administrative boundary and to provide more detailed information about the administrative hierarchy. This can be useful for a wide range of applications, from urban planning to public health to social services.

Models

```
class AdministrationAttributeDataType:
    text = 1
    number = 2
    option = 3

    FieldStr = {
        text: 'Text',
        number: 'Number',
        option: 'Option'
    }
```

```
class AdministrationEntityType(models.Model):
    name = models.CharField(max_length=255)
    description = models.TextField()


class AdministrationEntityTypeAttribute(models.Model):
    administration_entity_type = models.ForeignKey(AdministrationEntityType, on_delete=models.CASCADE)
    name = models.CharField(max_length=255)
    data_type = models.IntegerField(
      choices=AdministrationAttributeDataType.FieldStr.items(),
      default=AdministrationAttributeDatatType.text)
    options = models.JSONField(default=None, null=True)


class Entity(models.Model):
    administration = models.ForeignKey(Administration, on_delete=models.CASCADE)
    administration_entity_type = models.ForeignKey(AdministrationEntityType, on_delete=models.CASCADE)
    address = models.CharField(max_length=255)
    coordinates = models.PointField()


class AdministrationEntityValue(models.Model):
    entity = models.ForeignKey(Entity, on_delete=models.CASCADE)
    attribute = models.ForeignKey(AdministrationEntityTypeAttribute, on_delete=models.CASCADE)
    value = models.JSONField()
```

Administration Entity Type

This model represents the type of administrative entity, such as schools, health facilities, or parks. It might include fields like name or description. Example:

id	name	description
1	Schools	Schools available in the division

Administration Entity Type Attribute

This model represents the attributes of a given administrative entity type, such as the name of the school, the number of students, or the type of school. It might include fields like name, description, or data type.

id	administration_entity _type_id	name	data_type	options
1	1	School Name	1	null

| 2 | 1 | School Type | 3 | ["Boys School","Girl School", "Public School", "Private School"] |
| 3 | 1 | Number of Students | 2 | null |

Entity

This model represents the relationship between an administrative entity and the division that contains it.

id	administration	administration_entity _type_id	address	coordinates
1	42	1	Jl. Kemang Raya No.36, RT.12/RW.5, Bangka, Kec. Mampang Prpt., Kota Jakarta Selatan, Daerah Khusus Ibukota Jakarta 12150	(-6.262371400000001, -6.262371400000001)

Administration Entity Value

This model represents the values of the attributes for a given administrative entity. It might include fields like entity_id (the ID of the administrative entity), attribute_id (the ID of the administrative entity attribute), and value (the value of the attribute for the entity).

id	entity_id	administration_entity_type_ id	value
1	1	1	SD Pelita Harapan Kemang
2	1	2	Private School
3	1	3	90