BASIC PROJECT DEMONSTRATION

(Django with REST Framework Integration for Managing Auction)

This project demonstrates the seamless integration of Django, a powerful web framework for building web applications, with Django REST Framework, a toolkit for building Web APIs. The primary objective is to showcase how Django's Object-Relational Mapping (ORM) capabilities can be utilised alongside the RESTful architecture provided by Django REST Framework to efficiently manage data related to coins. Through this integration, developers can leverage Django's robust features for database interaction while also exposing CRUD operations via RESTful API endpoints, thereby enabling seamless data management for coin-related information. This documentation provides a step-by-step guide on setting up the project, defining models, creating API endpoints, and deploying the application for real-world use cases.

1. *Object Relational Mapping*

We defined your database structure, and we can use Django's ORM (Object-Relational Mapping) to interact with your database and then display the data in the Django admin panel.

Table: coins

Field	Type	Description
coin_id	Primary Key	Unique identifier for the coin
coin_image	String	URL or file path to the coin image
coin_name	String	Name or title of the coin
coin_desc	Text	Description or details about the coin
coin_year	Integer	Year the coin was minted
coin_country	String	Country where the coin was minted
coin_material	String	Material of the coin (e.g., gold, silver, copper)

coin_weight	Float	Weight of the coin in grams
starting_bid	Float	Starting bid price for the coin
coin_status	String	Status of the coin (e.g., available, sold)

NOTE : WHILE FOLLOWING BELOW CODES, MAKE SURE THE INDENTATION IS VERY IMPORTANT IN PYTHON

First, you'll need to create a Django model for the "coins" table. Here's how you can do it:

- Activate the virtual environment:

  source myenv/bin/activate (Replace 'myenv' with actual environment)

- Open your Django app's models.py file. If not, create the app within the main directory by command :

  python manage.py startapp coins

  Remember to also add 'coins' to the INSTALLED_APPS list in your Django project's settings.py file:

```
INSTALLED_APPS = [
    ...
    'coins',
    ...
]
```

- Define a Python class for the "coins" table, representing each field as a class attribute.
- You should install Pillow using pip, either globally or within your virtual environment. Here's how to do it within your virtual environment:

  Install Pillow using pip:

  Pillow is being installed to provide Python developers with a powerful library for image processing tasks, offering functionalities such as image loading, manipulation, and saving. It simplifies the implementation of image-related features in Python applications, making it a popular choice for projects requiring image handling capabilities.

python -m pip install Pillow

- The coin_images/ directory should be created within your Django project's media directory. By default, Django looks for media files (including user-uploaded files like images) in the media/ directory of your project.

  Create a media/ directory in your Django project (BasicApp) if you haven't already:

```
BasicApp/
├── coins/
│   ├── __init__.py
│   ├── admin.py
│   ├── apps.py
│   ├── migrations/
│   │   └── __init__.py
│   ├── models.py
│   ├── serializers.py
│   ├── tests.py
│   └── views.py
├── manage.py
└── BasicProject/
    ├── __init__.py
    ├── asgi.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
└── media/                    ->>Here, you can place the folder
    └── coin_images/
```

  mkdir media

  mkdir media/coin_images

- Use Django model fields to map each database field to a corresponding Python data type.

  Update your **models.py** in your app folder :

```
from django.db import models

class Coin(models.Model):
```

```
    # Define choices for coin status
    STATUS_CHOICES = (
        ('select', 'Select'),  # Placeholder option
        ('available', 'Available'),
        ('sold', 'Sold'),
        ('pending', 'Pending'),
    )

    coin_id = models.AutoField(primary_key=True)  # Auto-incrementing primary key
    coin_image = models.ImageField(upload_to='coin_images/', null=True, blank=True)  # Image field to store coin image
    coin_name = models.CharField(max_length=100)  # Char field for coin name
    coin_desc = models.TextField()  # Text field for coin description
    coin_year = models.IntegerField()  # Integer field for coin year
    coin_country = models.CharField(max_length=50)  # Char field for coin country
    coin_material = models.CharField(max_length=50)  # Char field for coin material
    coin_weight = models.FloatField()  # Float field for coin weight
    starting_bid = models.FloatField()  # Float field for starting bid
    coin_status = models.CharField(max_length=50,
choices=STATUS_CHOICES)  # Char field with choices for coin status
created_by_id = models.IntegerField(null=True, blank=True)
    def __str__(self):
        return self.coin_name  # Return the coin name as its string representation
```

- Ensure that your Django project's settings.py file is configured to serve media files during development. Add or modify the MEDIA_URL and MEDIA_ROOT settings accordingly:

```
# settings.py
```

```
# Add this at the top of the file
import os
# Add this to the bottom of the file
MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
```

- Make sure you have configured the necessary URL patterns to serve media files during development. In your project's urls.py, add the following import and URL configuration:

```
from django.contrib import admin
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static
urlpatterns = [
    path('admin/', admin.site.urls),
]
# Add this at the end of the file
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL,
document_root=settings.MEDIA_ROOT)
```

- Once done with the above installations and folder creations, next you need to run Django migrations to create the corresponding table in the database: (Do within your project directory)

```
python manage.py makemigrations
python manage.py migrate
```

If it shows "No changes detected" in terminal after migration, this could be because of below reasons :

- ***No Model Changes***: *If you haven't made any changes to your models since the last migration, Django won't detect any new changes.*
- ***Migration Already Applied***: *If the migration has already been applied to the database, Django won't detect any new changes since it keeps track of applied migrations.*
- ***Incorrect Directory***: *Ensure that you're running the makemigrations command in the correct directory containing your Django app.*
- ***Migration History***: *If you've manually modified migration files or deleted migration history, Django may not detect changes properly.*
- ***Verify the "INSTALLED APPS" in settings.py*** *:*
    - *"your_app_name"*

- Once the table is created, you can register the model with the Django admin site to view and manage the data. Here's how you can do it:
    - ❖ Open your app's admin.py file.
    - ❖ Import your Coin model and register it with the admin site.

```
from django.contrib import admin

from django.utils.html import format_html

from .models import Coin  # Import the Coin model

from django.conf import settings  # Import Django settings module

@admin.register(Coin)  # Register the Coin model with the admin site

class CoinAdmin(admin.ModelAdmin):  # Define the admin class for Coin model

# Specify the fields to display in the list view of the admin site

list_display = ('coin_name', 'display_coin_image', 'coin_desc',

'coin_year', 'coin_country', 'coin_material', 'coin_weight',

'starting_bid', 'coin_status')

# Define a method to display the coin image in the list view

def display_coin_image(self, obj):

# Generate HTML code to display the coin image with specified width and height

return format_html('<img src="{}" style="max-width:100px;

max-height:100px;">'.format(obj.coin_image.url))
```

Now, you can run your Django development server (python manage.py runserver) and navigate to the Django admin panel (http://localhost:8000/admin) to view and manage your "coins" data. You'll be able to add, edit, and delete coin records through the admin interface.

If admin panel is non accessible, create admin superuser by the command in root directory :

```
python manage.py createsuperuser
```

Follow the prompts to sign up with email, username and password. Then you'll be able to log on to the django admin and do the CRUD

==========