Mastering CRUD with Django & Django REST Framework: Build Real APIs Step-by-Step

🔄 Mastering CRUD with Django & Django REST Framework: Build Real APIs Step-by-Step

One of the most important things every backend developer must know is how to build and manage **CRUD operations**. CRUD stands for **Create, Read, Update, and Delete**. These are the four basic operations needed to manage data in most applications. Django consulting services
In this module, you'll learn how to build a complete CRUD API using Django REST Framework (DRF). You'll define models, create serializers, write views, and test everything using Postman.

📦 Step 1: Define a Model

First, we define a simple model for a task management app. Each task will have a title, description, and status.

```
# core/models.py
from django.db import models

class Task(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField(blank=True)
    completed = models.BooleanField(default=False)

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

Run the following commands to create the database table:

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

🧩 Step 2: Create a Serializer

A serializer in DRF is used to convert complex Python objects (like Django models) into JSON so it can be sent over the web. It also helps with data validation.

```
# core/serializers.py
from rest_framework import serializers
from .models import Task

class TaskSerializer(serializers.ModelSerializer):
    class Meta:
        model = Task
        fields = '__all__'
```

🧠 Step 3: Write API Views

There are many ways to write views in DRF. We'll use class-based views with `APIView` for more control and better learning.

```
# core/views.py
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .models import Task
from .serializers import TaskSerializer

class TaskListCreate(APIView):
    def get(self, request):
        tasks = Task.objects.all()
        serializer = TaskSerializer(tasks, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = TaskSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

class TaskDetail(APIView):
    def get_object(self, pk):
        try:
            return Task.objects.get(pk=pk)
        except Task.DoesNotExist:
            return None
    def get(self, request, pk):
        task = self.get_object(pk)
        if not task:
            return Response({"error": "Task not found"}, status=404)
        serializer = TaskSerializer(task)
        return Response(serializer.data)

    def put(self, request, pk):
        task = self.get_object(pk)
        if not task:
            return Response({"error": "Task not found"}, status=404)
        serializer = TaskSerializer(task, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=400)

    def delete(self, request, pk):
        task = self.get_object(pk)
        if not task:
            return Response({"error": "Task not found"}, status=404)
        task.delete()
        return Response({"message": "Deleted successfully"}, status=204)
```

🔗 Step 4: Configure URLs

Create a `urls.py` file inside your `core` app and wire up your views.

```
# core/urls.py
from django.urls import path
from .views import TaskListCreate, TaskDetail

urlpatterns = [
    path('tasks/', TaskListCreate.as_view(), name='task-list-create'),
    path('tasks//', TaskDetail.as_view(), name='task-detail'),
]

# In myproject/urls.py
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('core.urls')),
]
```

🧪 Step 5: Test Your API with Postman

Open Postman or Thunder Client in VS Code
Send a **POST** request to `http://localhost:8000/api/tasks/` with JSON:

```
{
  "title": "Learn Django",
  "description": "Build a CRUD app",
  "completed": false
}
```

Use **GET** to list all tasks
Use **PUT** to update a task
Use **DELETE** to delete a task by ID

📌 Best Practices and Tips

Use consistent naming for serializers and views
Validate input before saving
Always return proper status codes (200, 201, 400, 404, 204)
Break views into smaller views if needed
Keep code DRY (Don't Repeat Yourself)

🚀 Summary

You just built a full CRUD API from scratch using Django and DRF! This is the most common backend task across companies – managing data using APIs.

In real-world apps, every feature – from user profiles to blog posts to orders – will use the same CRUD logic. That's why mastering this early on will give you a strong foundation.

In the next module, we'll learn about using **ModelViewSet** and **routers** to speed up CRUD operations with less code.