Advanced Django REST Framework (DRF) Exercise: Building a Feature-Rich Bookstore API with Custom Login Using Phone Number and OTP Authentication

In this advanced exercise, students will enhance the **Bookstore API** with more complex features, including advanced model fields, custom validation in serializers, and custom authentication using **phone numbers and OTP (One-Time Password)** along with **JWT**.

Learning Objectives

By the end of this exercise, students will learn how to:

- Use **advanced model fields** and constraints.
- Implement **custom validation** in serializers.
- Use **Serializer fields** like **ReadOnlyField**, **SlugRelatedField**, **Nested Serializers**, and **Custom Fields**.
- Create a **custom login system** using **phone numbers and OTP**.
- Handle **JWT authentication** with a custom payload.
- Implement **rate limiting** and **throttling** for specific API endpoints.

Step-by-Step Guide

Step 1: Project Setup and App Creation

1. **Create a Django project** called `bookstore_api` and a new app called `books`:

   ```
   bash
   Copy code
   django-admin startproject bookstore_api
   cd bookstore_api
   python manage.py startapp books
   ```

2. **Install Django REST Framework and OTP libraries**:

   ```
   bash
   Copy code
   pip install djangorestframework djangorestframework-simplejwt django-otp django-phonenumber-field
   ```

3. **Add installed apps** in `bookstore_api/settings.py`:

   ```
   python
   Copy code
   INSTALLED_APPS = [
       'rest_framework',
       'rest_framework_simplejwt',
       'books',
       'django_otp',
       'phonenumber_field',
   ]
   ```

Step 2: Enhance Models with Advanced Features

1. **Add phone number field and unique constraints** in `books/models.py`:

```
python
Copy code
from django.db import models
from django.contrib.auth.models import AbstractBaseUser, BaseUserManager
from phonenumber_field.modelfields import PhoneNumberField

class UserManager(BaseUserManager):
    def create_user(self, phone_number, password=None, **extra_fields):
        if not phone_number:
            raise ValueError("The Phone Number must be set")
        user = self.model(phone_number=phone_number, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)
        return user

class User(AbstractBaseUser):
    phone_number = PhoneNumberField(unique=True)
    is_active = models.BooleanField(default=True)

    USERNAME_FIELD = 'phone_number'
    REQUIRED_FIELDS = []

    objects = UserManager()

    def __str__(self):
        return str(self.phone_number)

class Author(models.Model):
    name = models.CharField(max_length=100)
    biography = models.TextField(blank=True)

class Book(models.Model):
    title = models.CharField(max_length=150)
    slug = models.SlugField(unique=True)
    published_date = models.DateField()
    price = models.DecimalField(max_digits=10, decimal_places=2)
    author = models.ForeignKey(Author, related_name='books',
on_delete=models.CASCADE)
```

Step 3: Create Serializers with Advanced Features

1. **Customize serializers with validation and nested relationships** in `books/serializers.py`:

```
python
Copy code
from rest_framework import serializers
from .models import Author, Book, User

class BookSerializer(serializers.ModelSerializer):
    author_name = serializers.ReadOnlyField(source='author.name')
    slug = serializers.SlugField(read_only=True)

    class Meta:
        model = Book
        fields = ['id', 'title', 'slug', 'author_name', 'published_date',
'price']

    def validate_price(self, value):
        if value <= 0:
            raise serializers.ValidationError("Price must be positive.")
        return value

class AuthorSerializer(serializers.ModelSerializer):
    books = BookSerializer(many=True, read_only=True)

    class Meta:
        model = Author
        fields = ['id', 'name', 'biography', 'books']

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['phone_number', 'is_active']
```

Step 4: Implement OTP-Based Login with JWT

1. **Create a custom view for OTP generation** in `books/views.py`:

```
python
Copy code
import random
from django_otp.oath import totp
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework_simplejwt.tokens import RefreshToken
from .models import User

class GenerateOTPView(APIView):
    def post(self, request):
        phone_number = request.data.get('phone_number')
        user, created = 
User.objects.get_or_create(phone_number=phone_number)
        otp = random.randint(100000, 999999)  # Replace with a real OTP 
generator in production
        # Send OTP via SMS (pseudo-code)
        print(f"OTP for {phone_number}: {otp}")
        request.session[f'otp_{phone_number}'] = otp  # Store OTP in 
session for demo
        return Response({"message": "OTP sent to phone number."})

class VerifyOTPView(APIView):
    def post(self, request):
        phone_number = request.data.get('phone_number')
        otp = request.data.get('otp')
        saved_otp = request.session.get(f'otp_{phone_number}')
        if str(otp) == str(saved_otp):
            user = User.objects.get(phone_number=phone_number)
            refresh = RefreshToken.for_user(user)
            return Response({"access": str(refresh.access_token), 
"refresh": str(refresh)})
        return Response({"error": "Invalid OTP."}, status=400)
```

Step 5: Configure JWT and Permissions

1. **Set up JWT and custom authentication classes** in `bookstore_api/settings.py`:

```
python
Copy code
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
}
```

Step 6: Configure URLs

1. **Add OTP endpoints** in `books/urls.py`:

```
python
Copy code
from django.urls import path
from .views import GenerateOTPView, VerifyOTPView

urlpatterns = [
    path('otp/generate/', GenerateOTPView.as_view(), name='otp-generate'),
    path('otp/verify/', VerifyOTPView.as_view(), name='otp-verify'),
]
```

Bonus Challenges

1. **Rate Limiting and Throttling**: Add rate limiting for OTP endpoints to prevent abuse.
2. **Email Notifications**: Implement email-based notifications for user registration.

Learning Outcomes

By completing this advanced exercise, students will:

- Gain experience with **custom authentication** using **phone numbers and OTP**.
- Implement **advanced validation** in serializers.
- Learn how to handle **nested serializers** and **related fields**.
- Enhance security with **JWT authentication** and **throttling**.

This exercise equips students with practical knowledge of real-world API development using Django REST Framework.