<u>Task:</u>

Create a simple RESTful API using Django and Django REST Framework. This API should allow users to perform CRUD operations on a resource representing books. You will need to create endpoints for creating, reading, updating, and deleting books. After implementing the API, you will use Postman to test the endpoints and demonstrate the functionality.

<u>Answer:</u>

Procedure:

Make sure you have already python has installed

Check:

- **python --version**

Python 3.9.13

1. Creating a virtual Environment By using Command on Console
   - **mkvirtualenv books_store**
   - **workon books_store (But if create it first time it will by default active)**
   - **deactivate (for deactive virtual environment)**

2. Installing Django Framework
   - **pip install Django**

3. Installing Django Rest Framework
   - **pip install djangorestframework**

4. Installing simple jwt token package
   - **pip install djangorestframework-simpljwt**

5. Check installed Package:
   - **pip freeze**

   ```
   asgiref==3.7.2
   Django==4.2.11
   djangorestframework==3.14.0
   djangorestframework-simplejwt==5.3.1
   PyJWT==2.8.0
   pytz==2024.1
   sqlparse==0.4.4
   typing_extensions==4.10.0
   tzdata==2024.1
   ```

6. Creating Project
   - **Project_location>django-admin startproject bookstore**
     Eg: D:\Personal_Workspace> django-admin startproject bookstore

7. Creating App:
   - **Python manage.py startapp books**

   Eg:

   > cd bookstore

   D:\Personal_Workspace\bookstore> python manage.py startapp books

8. Install this app inside in settings.py file

```
9. INSTALLED_APPS = [
10.     'django.contrib.admin',
11.     'django.contrib.auth',
12.     'django.contrib.contenttypes',
13.     'django.contrib.sessions',
14.     'django.contrib.messages',
15.     'django.contrib.staticfiles',
16.     'books',
17.     'rest_framework',
18.     'rest_framework_simplejwt',
19. ]
```

At this point my project setup has completed.

File/Folder Requirements:

1. Creating folder named with templates\books
2. Create file base.html and home.html Inside templates\books
3. Create a file named with serializers.py inside books app
4. Create a file name with urls.py inside books app

models.py:

```python
from django.db import models

# Create your models here.
class BookModel(models.Model):
    author = models.CharField(max_length=60)
    genre = models.CharField(max_length=50)
    published_date = models.DateField()

    class Meta:
        db_table = 'books_table'
```

Use Command to generate SQL code and execute that code

- python manage.py makemigrations

- python manage.py migrate

Create Super user by using commang

- python manage.py createsuperuser

admin.py

```
from django.contrib import admin
from .models import BookModel

# Register your models here.

@admin.register(BookModel)
class BooksAdmin(admin.ModelAdmin):
    list_display = ['id','author','genre','published_date']
```

serializers.py

```
from rest_framework.serializers import ModelSerializer

from .models import BookModel

class BooksModelSerializer(ModelSerializer):
    class Meta:
        model = BookModel
        # fields = ['author','genre','published_date']
        fields = '__all__'
```

views.py

```
from django.shortcuts import render
from .models import BookModel
from .serializers import BooksModelSerializer
from rest_framework.response import Response
from rest_framework import status
from rest_framework.generics import ListCreateAPIView,
RetrieveUpdateDestroyAPIView
from rest_framework.viewsets import ModelViewSet
from rest_framework.permissions import IsAuthenticated
from rest_framework_simplejwt.authentication import JWTAuthentication
from rest_framework.decorators import api_view
from rest_framework.views import APIView
# We can also import APIView from decorators
# from rest_framework.decorators import APIView


# Creating a home function to rendering home.html file
```

```python
def home(request):
    return render(request, 'books/home.html')


# Creating BooksModelViewSet class for CRUD Opreations.
# First Way By Using ModelViewSet Module
class BooksModelViewSet(ModelViewSet):
    queryset = BookModel.objects.all()
    serializer_class = BooksModelSerializer
    #if you want to aunthentication and authorization kind of thing, then
remove comment below or line 22 and 23
    # authentication_classes = [JWTAuthentication]
    # permission_classes = [IsAuthenticated]


#Second Way By using ListCreateAPIView and RetrieveUpdateDestroyAPIView
class BooksListCreateAPIView(ListCreateAPIView):
    queryset = BookModel.objects.all()
    serializer_class = BooksModelSerializer
    #if you want to aunthentication and authorization kind of thing, then
remove comment below or line 30 and 31
    # authentication_classes = [JWTAuthentication]
    # permission_classes = [IsAuthenticated]

class BooksRetrieveUpdateDestroyAPIView(RetrieveUpdateDestroyAPIView):
    queryset = BookModel.objects.all()
    serializer_class = BooksModelSerializer
    #if you want to aunthentication and authorization kind of thing, then
remove comment below or line 37 and 38
    # authentication_classes = [JWTAuthentication]
    # permission_classes = [IsAuthenticated]


#Third Way by Using APIView Class
class BooksAPIView(APIView):
    def get_by_object(self, id):
        try:
            book = BookModel.objects.get(id=id)
        except BookModel.DoesNotExist:
            book = None
        return book

    def get(self, request, pk=None, format=None):
        id = pk
        if id is not None:
            book = self.get_by_object(id)
            if book is None:
                return Response({'msg':'Provided Book Id is not available'})
            serializer = BooksModelSerializer(book)
            return Response(serializer.data)
        books = BookModel.objects.all()
        serializer = BooksModelSerializer(books, many=True)
        return Response(serializer.data)

    def post(self, request, format=None):
        serializer = BooksModelSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg':'Data Created'},
status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def put(self, request, pk, format=None):
        id = pk
        book = self.get_by_object(id=id)
        if book is None:
            return Response({'msg':'Provided Book Id is not available'})
        serializer = BooksModelSerializer(book, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg':'Complete Updated Success'})
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def patch(self, request, pk, format=None):
        id = pk
        book = self.get_by_object(id)
        if book is None:
            return Response({'msg':'Provided Book Id is not available'})
        serializer = BooksModelSerializer(book, data=request.data,
partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg':'Partial Updated Success'})
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    def delete(self, request, pk, format=None):
        id = pk
        book = self.get_by_object(id)
        if book is None:
            return Response({'msg':'Provided Book Id is not available'})
        book.delete()
        return Response({'msg':'Data Deleted'})


# Fourth Way Function Based APIView
def get_by_object(id):
        try:
            book = BookModel.objects.get(id=id)
        except BookModel.DoesNotExist:
            book = None
        return book

@api_view(['GET','POST','PUT','PATCH','DELETE'])
def BooksFBAPIView(request, pk=None):
    if request.method == 'GET':
        if pk is not None:
            book = get_by_object(pk)
            if book is None:
                return Response({'msg':'Provided Book Id is not available'})
            serializer = BooksModelSerializer(book)
            return Response(serializer.data)
        books = BookModel.objects.all()
        serializer = BooksModelSerializer(books, many=True)
        return Response(serializer.data)


    elif request.method == 'POST':
        serializer = BooksModelSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg':'Data Created'},
status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    elif request.method == 'PUT':
        if pk is not None:
            book = get_by_object(pk)
            if book is None:
                return Response({'msg':'Provided Book Id is not available'})
        serializer = BooksModelSerializer(book, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg':'Complete Updated Success'})
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    elif request.method == 'PATCH':
        if pk is not None:
            book = get_by_object(pk)
            if book is None:
                return Response({'msg':'Provided Book Id is not available'})
        serializer = BooksModelSerializer(book, data=request.data,
partial=True)
        if serializer.is_valid():
            serializer.save()
            return Response({'msg':'Partial Updated Success'})
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
    elif request.method == 'DELETE':
        if pk is not None:
            book = get_by_object(pk)
            if book is None:
                return Response({'msg':'Provided Book Id is not available'})
        book.delete()
        return Response({'msg':'Data Deleted'})
```

Urls.py

```
from django.urls import path,include
from .views import home
from .views import (BooksModelViewSet,
                    BooksListCreateAPIView,
                    BooksRetrieveUpdateDestroyAPIView,
                    BooksAPIView,
                    BooksFBAPIView)
from rest_framework_simplejwt.views import TokenObtainPairView,
TokenRefreshView, TokenVerifyView
from rest_framework.routers import DefaultRouter
router = DefaultRouter()
router.register('api', BooksModelViewSet, basename='BooksAPI' )

urlpatterns = [
    path('', home, name='home'),
    path('', include(router.urls)),
    # ListCreateAPIView and RetrieveUpdateDestroyAPIView
    path('api/v1', BooksListCreateAPIView.as_view(),
name='BooksListCreateAPIView'),
    path('api/v1/<int:pk>/', BooksRetrieveUpdateDestroyAPIView.as_view(),
name='BooksRetrieveUpdateDestroyAPIView'),
    # Class Based APIView
    path('apivewcbv/v1/',BooksAPIView.as_view(), name='Class_basedAPIView'),
    path('apivewcbv/v1/<int:pk>/',BooksAPIView.as_view(),
name='Class_basedAPIView_pk'),
    # Fucntion Based APIView
    path('apifb/v1/', BooksFBAPIView, name='BooksFBAPIView_FB'),
    path('apifb/v1/<int:pk>/', BooksFBAPIView, name='BooksFBAPIView_FB'),
    # For Jwt Authentication purpose, getting jwt token kind of thing
    path('get_token/', TokenObtainPairView.as_view(), name='get_token'),
    path('ref_token/',TokenRefreshView.as_view(), name='ref_token'),
    path('ver_token/',TokenVerifyView.as_view()),
]
```