**Practical Task:**

**Building a simplified event management API for an admin panel. It should support role-based**

**access (Admin and User), event creation, and basic ticket purchases.**

**Solution :**

**Step 1: Set up the Django Project**

1. **Install Django and Dependencies**:

Python, Django installed, and other necessary libraries.

bash

Copy code

python -m venv env

env\Scripts\activate

pip install django djangorestframework psycopg2-binary djangorestframework-simplejwt

1. **Create the Django Project and App**:

bash

Copy code

django-admin startproject EventAPI

cd EventAPI

django-admin startapp events

1. **Configure the Database (PostgreSQL)**:

In EventAPI/settings.py, configure the PostgreSQL database:

python

Copy code

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'eventapi\_db',

'USER': 'your\_db\_user',

'PASSWORD': 'your\_db\_password',

'HOST': 'localhost',

'PORT': '5432',

}

}

PostgreSQL database created:

bash

Copy code

psql -U postgres

CREATE DATABASE eventapi\_db;

CREATE USER your\_db\_user WITH PASSWORD 'your\_db\_password';

ALTER ROLE your\_db\_user SET client\_encoding TO 'utf8';

ALTER ROLE your\_db\_user SET default\_transaction\_isolation TO 'read committed';

ALTER ROLE your\_db\_user SET timezone TO 'UTC';

GRANT ALL PRIVILEGES ON DATABASE eventapi\_db TO your\_db\_user;

\q

1. **Create a Superuser for Django**:

bash

Copy code

python manage.py createsuperuser

**Step 2: Models**

Now define the models for User, Event, and Ticket.

1. **User Model**: We'll use Django's AbstractBaseUser for custom user roles.

In events/models.py:

python

Copy code

from django.db import models

from django.contrib.auth.models import AbstractBaseUser, BaseUserManager

class UserManager(BaseUserManager):

def create\_user(self, username, password=None, \*\*extra\_fields):

if not username:

raise ValueError('The Username field must be set')

user = self.model(username=username, \*\*extra\_fields)

user.set\_password(password)

user.save(using=self.\_db)

return user

def create\_superuser(self, username, password=None, \*\*extra\_fields):

extra\_fields.setdefault('role', 'Admin')

return self.create\_user(username, password, \*\*extra\_fields)

class User(AbstractBaseUser):

ROLES = (

('Admin', 'Admin'),

('User', 'User'),

)

username = models.CharField(max\_length=255, unique=True)

password = models.CharField(max\_length=255)

role = models.CharField(max\_length=5, choices=ROLES, default='User')

objects = UserManager()

USERNAME\_FIELD = 'username'

REQUIRED\_FIELDS = ['password']

def \_\_str\_\_(self):

return self.username

1. **Event Model**: To store event details like name, date, and tickets.

python

Copy code

class Event(models.Model):

name = models.CharField(max\_length=255)

date = models.DateField()

total\_tickets = models.IntegerField()

tickets\_sold = models.IntegerField(default=0)

def \_\_str\_\_(self):

return self.name

1. **Ticket Model**: To store ticket purchase details.

python

Copy code

class Ticket(models.Model):

user = models.ForeignKey(User, on\_delete=models.CASCADE)

event = models.ForeignKey(Event, on\_delete=models.CASCADE)

quantity = models.IntegerField()

purchase\_date = models.DateTimeField(auto\_now\_add=True)

def \_\_str\_\_(self):

return f'{self.quantity} tickets for {self.event.name} by {self.user.username}'

**Step 3: API Endpoints (with Role-based Access)**

1. **Installed DRF (Django REST Framework)**:

bash

Copy code

pip install djangorestframework

1. **Define Serializers**:

In events/serializers.py:

python

Copy code

from rest\_framework import serializers

from .models import User, Event, Ticket

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = ['username', 'password', 'role']

class EventSerializer(serializers.ModelSerializer):

class Meta:

model = Event

fields = ['id', 'name', 'date', 'total\_tickets', 'tickets\_sold']

class TicketSerializer(serializers.ModelSerializer):

class Meta:

model = Ticket

fields = ['user', 'event', 'quantity', 'purchase\_date']

1. **Defined Views**:

In events/views.py:

python

Copy code

from rest\_framework import status, permissions

from rest\_framework.decorators import api\_view, permission\_classes

from rest\_framework.response import Response

from .models import User, Event, Ticket

from .serializers import UserSerializer, EventSerializer, TicketSerializer

# User Registration Endpoint

@api\_view(['POST'])

@permission\_classes([permissions.AllowAny])

def register\_user(request):

if request.method == 'POST':

serializer = UserSerializer(data=request.data)

if serializer.is\_valid():

user = serializer.save()

return Response({"message": "User created successfully"}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

# Create Event (Admin Only)

@api\_view(['POST'])

@permission\_classes([permissions.IsAdminUser])

def create\_event(request):

if request.method == 'POST':

serializer = EventSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response({"message": "Event created successfully"}, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

# Get Events (Admin and User)

@api\_view(['GET'])

@permission\_classes([permissions.IsAuthenticated])

def get\_events(request):

if request.method == 'GET':

events = Event.objects.all()

serializer = EventSerializer(events, many=True)

return Response(serializer.data)

# Purchase Tickets (User Only)

@api\_view(['POST'])

@permission\_classes([permissions.IsAuthenticated])

def purchase\_tickets(request, event\_id):

event = Event.objects.get(id=event\_id)

if not event:

return Response({"error": "Event not found"}, status=status.HTTP\_404\_NOT\_FOUND)

quantity = request.data.get('quantity')

if quantity + event.tickets\_sold > event.total\_tickets:

return Response({"error": "Not enough tickets available"}, status=status.HTTP\_400\_BAD\_REQUEST)

user = request.user

ticket = Ticket.objects.create(user=user, event=event, quantity=quantity)

event.tickets\_sold += quantity

event.save()

return Response({"message": "Tickets purchased successfully"}, status=status.HTTP\_201\_CREATED)

1. **URL Routing**:

In events/urls.py:

python

Copy code

from django.urls import path

from . import views

urlpatterns = [

path('register/', views.register\_user),

path('events/', views.create\_event),

path('events/', views.get\_events),

path('events/<int:event\_id>/purchase/', views.purchase\_tickets),

]

In EventAPI/urls.py:

python

Copy code

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

path('admin/', admin.site.urls),

path('api/', include('events.urls')),

]

**Step 4: Custom SQL Query**

To fetch the top 3 events by the total tickets sold, we can write a custom SQL query in Django.

python

Copy code

from django.db import connection

def get\_top\_events():

with connection.cursor() as cursor:

cursor.execute("""

SELECT event\_id, name, SUM(quantity) as total\_sold

FROM events\_ticket

JOIN events\_event ON events\_ticket.event\_id = events\_event.id

GROUP BY event\_id

ORDER BY total\_sold DESC

LIMIT 3;

""")

return cursor.fetchall()

**Step 5: JWT Authentication (Bonus)**

For JWT-based authentication, add rest\_framework\_simplejwt in your settings:

python

Copy code

INSTALLED\_APPS = [

# other apps

'rest\_framework\_simplejwt',

]

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': (

'rest\_framework\_simplejwt.authentication.JWTAuthentication',

),

}

Then add views for token creation in events/views.py:

python

Copy code

from rest\_framework\_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns += [

path('token/', TokenObtainPairView.as\_view(), name='token\_obtain\_pair'),

path('token/refresh/', TokenRefreshView.as\_view(), name='token\_refresh'),

]