Module 10) Rest Framework

1. Introduction to APIs

Q-1]What is an API (Application Programming Interface)?

➢ An API allows two software applications to communicate and exchange data.

Examples:-

```
import requests
response = requests.get("https://api.github.com")
print(response.status_code)  # Example API call
```

Q-2] Types of APIs: REST, SOAP.

- **REST** uses HTTP methods and usually returns JSON.
- **SOAP** uses XML for structured communication.

Example REST API call:-

```
response = requests.get("https://jsonplaceholder.typicode.com/posts")
print(response.json()[0])
```

Q-3] Why are APIs important in web development?

➢ APIs enable data sharing and integration between systems (e.g., connecting your app to Google Maps or weather data).

2. Requirements for Web Development Projects

Q-4] Understanding project requirements.

Q-5] Setting up the environment and installing necessary packages.

➢ pip install django djangorestframework
➢ # settings.py example snippet

```
INSTALLED_APPS = [
    'rest_framework',
    'myapp',
]
```

3. Serialization in Django REST Framework

Q-6] What is Serialization?

Serialization converts Python/Django objects into JSON for easy transmission via APIs.

```
from rest_framework import serializers
from .models import Student

class StudentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Student
        fields = '__all__'
```

Q-7] Converting Django QuerySets to JSON.

```
from django.core import serializers
from .models import Student
data = serializers.serialize('json', Student.objects.all())
print(data)
```

Q-8] Using serializers in Django REST Framework (DRF).

```
serializer = StudentSerializer(Student.objects.all(), many=True)
print(serializer.data)
```

4. Requests and Responses in Django REST Framework

Q-9] HTTP request methods (GET, POST, PUT, DELETE).

```
# Example DRF view handling different methods
@api_view(['GET', 'POST', 'PUT', 'DELETE'])
def student_api(request):
    if request.method == 'GET':
        return Response({"message": "GET request"})
```

Q-10] Sending and receiving responses in DRF.

```
from rest_framework.response import Response
return Response({"message": "Hello, world!"})
```

5. Views in Django REST Framework

Q-11] Understanding views in DRF: Function-based views vs Class-based views.

Example:-

```
# Function-based view
@api_view(['GET'])
def student_list(request):
    return Response({"students": []})
# Class-based view
from rest_framework.views import APIView
class StudentList(APIView):
    def get(self, request):
        return Response({"students": []})
```

6. URL Routing in Django REST Framework

Q-12] Defining URLs and linking them to views.

```
from django.urls import path
from .views import student_list
urlpatterns = [
    path('students/', student_list),
]
```

7. Pagination in Django REST Framework

Q-13] Adding pagination to APIs to handle large data sets.

```
# settings.py
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 5
}
```

8. Settings Configuration in Django

Q-14] Configuring Django settings for database, static files, and API keys.

Example:-

```
# settings.py
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / "db.sqlite3",
    }
}
STATIC_URL = '/static/'
```

9. Project Setup

Q-15] Setting up a Django REST Framework project.

Steps:-

```
django-admin startproject myproject
cd myproject
python manage.py startapp api
# settings.py
INSTALLED_APPS = ['rest_framework', 'api']
```

10. Social Authentication, Email, and OTP Sending API

Q-16] Implementing social authentication (e.g., Google, Facebook) in Django.

- pip install social-auth-app-django
- AUTHENTICATION_BACKENDS = (

```
    'social_core.backends.google.GoogleOAuth2',
    'django.contrib.auth.backends.ModelBackend',
)
```

Q-17] Sending emails and OTPs using third-party APIs like Twilio, SendGrid.

```
from django.core.mail import send_mail

send_mail("Subject", "Your OTP is 1234", "from@example.com", ["to@example.com"])
```

11. RESTful API Design

Q-18] REST principles: statelessness, resource-based URLs, and using HTTP methods for CRUD operations.

```
# Example endpoints for CRUD

GET /api/users/

POST /api/users/

PUT /api/users/1/

DELETE /api/users/1/
```

12. CRUD API (Create, Read, Update, Delete)

Q-19] What is CRUD, and why is it fundamental to backend development?

- **CRUD** stands for **Create, Read, Update, and Delete** — the four basic operations that can be performed on persistent data (like in a database).

Operation	**Description**	**Example (User Table)**
Create	Add new data to the database	Add a new user
Read	Retrieve existing data	View user details
Update	Modify existing data	Change user email
Delete	Remove data	Delete user account

- **Why it's fundamental:**
  - **Core of data management:** Every backend system interacts with data, and CRUD defines how that data is managed.
  - **Database integration:** CRUD maps directly to SQL operations (INSERT, SELECT, UPDATE, DELETE).
  - **RESTful APIs:** Most APIs follow CRUD principles through HTTP methods —
    - POST → Create
    - GET → Read
    - PUT/PATCH → Update

    - DELETE → Delete

- **Consistency & scalability:** CRUD ensures uniformity in data handling, making systems easier to scale, test, and maintain.

```
@api_view(['GET', 'POST', 'PUT', 'DELETE'])
def student_crud(request, id=None):
    if request.method == 'POST':
        return Response({"message": "Data created"})
```

13. Authentication and Authorization API

Q-20] Difference between authentication and authorization.

- **Authentication:** Verifies *who* is making the request. (e.g., login, token check). Ie: It verifies the identity. For Example in Django REST Framework , token verification via TokenAuthentication

- **Authorization:** Determines *what* the authenticated user can do (e.g., only admin can delete data). Ie: verifies permission . For Example in Django REST Framework, IsAdminUser, IsAuthenticated permissions

Q-21] Implementing authentication using Django REST Framework's token-based system.

- Tokens verify user identity for secured API access.
- pip install djangorestframework-simplejwt
- from rest_framework_simplejwt.views import TokenObtainPairView
- urlpatterns = [path('api/token/', TokenObtainPairView.as_view())]

14. OpenWeatherMap API Integration

Q-22] Introduction to OpenWeatherMap API and how to retrieve weather data.

- **OpenWeatherMap API** is a free (and paid-tier) web service that provides real-time and forecasted weather data for any location worldwide.

Steps to retrive weather data:

1. **Get an API key:**
   Sign up at https://openweathermap.org/api to get your unique API key.
2. **Choose an endpoint:**
   Common endpoints include:
    - **Current weather data:**
      https://api.openweathermap.org/data/2.5/weather

- **5-day forecast:** https://api.openweathermap.org/data/2.5/forecast

Make a request (example):

https://api.openweathermap.org/data/2.5/weather?q=London&appid=YOUR_API_KEY&units=metric

In Python:-

```
import requests

city = "London"

api_key = "YOUR_API_KEY"

url = f"https://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

response = requests.get(url)

data = response.json()

print(f"Temperature: {data['main']['temp']}°C")

print(f"Weather: {data['weather'][0]['description']}")
```

- Fetches live weather details by city.
- import requests

```
api_key = "your_api_key"

city = "London"

url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}"

print(requests.get(url).json())
```

15. Google Maps Geocoding API

Q-23] Using Google Maps Geocoding API to convert addresses into coordinates.

- Turns text addresses into latitude & longitude.
- url = "https://maps.googleapis.com/maps/api/geocode/json"
- params = {"address": "New York", "key": "your_api_key"}
- print(requests.get(url, params=params).json())

16. GitHub API Integration

Q-24] Introduction to GitHub API and how to interact with repositories, pull requests, and issues.

- ➢ **GitHub API** allows developers to programmatically interact with GitHub — managing repositories, issues, pull requests, users, and more.
- ➢ Base URL:
  https://api.github.com
- ➢ Authentication:

  - Use a **Personal Access Token (PAT)** or **OAuth**.
    Example:
  - curl -H "Authorization: token YOUR_TOKEN" https://api.github.com/user

Common operations:

Feature	**HTTP Method**	**Endpoint**	**Description**
List repositories	GET	/users/{username}/repos	Fetch all public repos of a user
Get repo details	GET	/repos/{owner}/{repo}	Get info about a repo
Create an issue	POST	/repos/{owner}/{repo}/issues	Create a new issue
List pull requests	GET	/repos/{owner}/{repo}/pulls	Get all pull requests
Merge pull request	PUT	/repos/{owner}/{repo}/pulls/{number}/merge	Merge a PR

- ➢ Example in Python:

```
import requests

token = "YOUR_GITHUB_TOKEN"

headers = {"Authorization": f"token {token}"}

# Example: List repositories

user = "octocat"

url = f"https://api.github.com/users/{user}/repos"

response = requests.get(url, headers=headers)

for repo in response.json():

   print(repo["name"])
```

- Use cases:
    - Automate repository management.
    - Create bots that handle issues or PRs.
    - Integrate GitHub data into dashboards or CI/CD pipelines.

17. Twitter API Integration

Q-25] Using Twitter API to fetch and post tweets, and retrieve user data.

```
headers = {"Authorization": "Bearer YOUR_ACCESS_TOKEN"}

r = requests.get("https://api.twitter.com/2/users/by/username/TwitterDev", headers=headers)

print(r.json())
```

18. REST Countries API Integration

Q-26] Introduction to REST Countries API and how to retrieve country-specific data.

- r = requests.get("https://restcountries.com/v3.1/name/india")
- print(r.json())

19. Email Sending APIs (SendGrid, Mailchimp)

Q-27] Using email sending APIs like SendGrid and Mailchimp to send transactional emails.

```
import sendgrid

from sendgrid.helpers.mail import Mail

sg = sendgrid.SendGridAPIClient("YOUR_API_KEY")

message = Mail(from_email='from@example.com', to_emails='to@example.com',
subject='Test', html_content='Hello!')

sg.send(message)
```

20. SMS Sending APIs (Twilio)

Q-28] Introduction to Twilio API for sending SMS and OTPs.

- Twilio lets you send SMS messages globally.
- Sending SMSs and OPTs using Twilio API:-

```
from twilio.rest import Client

client = Client("ACCOUNT_SID", "AUTH_TOKEN")
```

```
client.messages.create(to="+1234567890", from_="+0987654321", body="Your OTP is 1234")
```

21. Payment Integration (PayPal, Stripe)

Q-29] Introduction to integrating payment gateways like PayPal and Stripe.

- Handles online payments securely.

```
import stripe
stripe.api_key = "your_secret_key"
payment = stripe.PaymentIntent.create(amount=5000, currency="usd")
print(payment)
```

22. Google Maps API Integration

Q-30] Using Google Maps API to display maps and calculate distances between locations.

- Calculates distances and displays maps between two points.

```
url = "https://maps.googleapis.com/maps/api/distancematrix/json"
params = {"origins": "New York", "destinations": "Los Angeles", "key": "your_api_key"}
print(requests.get(url, params=params).json())
```
