**Python Rest Framework**

**Q1. What is an API (Application Programming Interface)?**

Ans. An API is a set of rules and protocols that allows different software applications to communicate with each other. It defines how requests for specific functions should be made, what data should be sent, and what format the responses will follow.

Interface: It provides a defined way for one program to interact with another.

Request/Response Model: Clients (like web apps) send requests to servers via the API and receive responses.

Data Formats: APIs usually exchange data in formats like JSON or XML.

Type Of APIs:

Web API, Library API, Operating System API

**Q2. Type of APIs : REST, SOAP**

**Ans. REST is an architectural style for designing networked applications. It uses HTTP methods like GET, POST, PUT, DELETE and works with resources, usually represented in JSON or XML.**

Stateless: Each request from the client to the server must contain all necessary information.

Resource-based: Uses URLs to access resources (like /users/123).

Lightweight: Often uses JSON, making it fast and simple.

Flexible: Can be used with many formats (JSON, XML, HTML, etc.).

SOAP is a **protocol** with strict standards. It uses **XML** for messages and typically runs over HTTP, but can also use SMTP and other protocols.

**Strict structure**: Uses WSDL (Web Services Description Language) for describing services.

**Built-in security**: Supports things like WS-Security, ACID transactions.

**More overhead**: Heavier than REST due to verbose XML and complex standards.

**Q3. Why are APIs important in web development?**

Ans. APIs are crucial in web development because they enable different systems, applications, and services to communicate and work together efficiently. Here's why they're so important:

* A web app can retrieve weather data from a weather API.
* An e-commerce site can use a payment gateway API (like PayPal or Stripe).
* Frontend sends requests to the API.
* Backend processes data and sends back responses (usually in JSON format).
* Reused across multiple apps (mobile, web, desktop).
* Scaled to serve more clients.
* Only expose what’s necessary.
* Use authentication and authorization to protect resources.

**Q4. Understanding project requirements.**

**Q5. Setting up the environment and installing necessary packages**

**Ans.** Project goals – What is the purpose of the project?

User needs – Who will use the application and what are their expectations?

Functional requirements – What features should the application include? (e.g., login, CRUD operations, payment system)

Non-functional requirements – How should the system perform? (e.g., speed, security, scalability)

Technology stack – What programming languages, frameworks, and tools will be used?

To build a web application using Django REST Framework, you need to prepare a development environment:

Create an isolated environment to manage dependencies:

python -m venv env

.myvenv\scripts\activate

Install Required Packages:

pip install django djangorestframework requests

Start the Project:

django-admin startproject myproject

cd myproject

python manage.py startapp myapp

**Q6. What is Serialization?**

**Q7. Converting Django QuerySets to JSON.**

**Q8. Using serializers in Django REST Framework (DRF).**

Ans. Serialization is the process of converting complex data types — like Django QuerySets or model instances — into native Python datatypes (like dictionaries), which can then be easily converted into JSON, XML, or other formats for:

APIs, Web responses, Data storage or transmission

Django QuerySets are not directly JSON serializable. Using serializers, you can convert them like this:

class Doctor(models.Model):

name = models.CharField(max\_length=100)

specialty = models.CharField(max\_length=100)

Django REST Framework provides powerful tools for serialization through the serializers module.

From restframework import serializers

From .models import doctor

class DoctorSerializer(serializers.ModelSerializer):

class Meta:

model = Doctor

fields = ['id', 'name', 'specialty']

**Q9. HTTP request methods (GET, POST, PUT, DELETE).**

**Q10. Sending and receiving responsesin DRF.**

Ans. In Django REST Framework (DRF), these methods define how clients (like browsers or mobile apps) interact with your API.

Get: Retrive data from the server Get a list of Doctor

Post: Send new data to the server Add new doctor profile

Put : Update Existing data Update Doctors full profile

Delete: Remove data from the server delete a doctor record

Sending and Receiving Responses in DRF

Django REST Framework provides the Request and Response classes to handle API data exchange.

Receiving Requests

In DRF views, request data is accessed using request.data

Sending Responses

Use DRF’s Response object to send data back to the client in JSON format by default.

**Q11. Understanding views in DRF: Function-based views vs Class-based views.**

Ans. In Django REST Framework (DRF), **views** handle the logic of your API — how data is processed and returned based on HTTP requests.

DRF supports two types of views:

Function-Based Views (FBVs)

These are regular Python functions that take a request and return a response. They use decorators like @api\_view to handle specific HTTP methods.

Class-Based Views (CBVs)

These use Python classes to organize logic by methods (get, post, etc.). DRF provides powerful generic views and viewsets to reduce code.

**Q12. Defining URLs and linking them to views.**

Ans. Defining URLs and Linking Them to Views in Django REST Framework (DRF)

In Django REST Framework, URL routing connects client requests to specific view functions or classes that handle the logic.

1. Define a View 2. Create a URL Pattern 3. Include App URLs in Project URLs

Now, accessing:

http://localhost:8000/api/fbv/ calls the function-based view

http://localhost:8000/api/cbv/ calls the class-based view

**Q13. Adding pagination to APIs to handle large data sets**

Ans. When your API returns a large number of records (e.g., 1,000+), it's inefficient to return all of them at once.

Pagination breaks the data into smaller chunks (pages), improving performance and usability.

Set Pagination in settings.py

REST\_FRAMEWORK = {

'DEFAULT\_PAGINATION\_CLASS': 'rest\_framework.pagination.PageNumberPagination',

'PAGE\_SIZE': 10 # Number of records per page

}

Create a View (List API)

class DoctorListView(ListAPIView):

queryset = Doctor.objects.all()

serializer\_class = DoctorSerializer

Add URL Mapping

urlpatterns = [

path('doctors/', DoctorListView.as\_view(), name='doctor-list'),

]

**Q14. Configuring Django settings for database,static files, and API keys.**

Ans. Database Configuration

In settings.py, configure your database in the DATABASES dictionary.

Static Files Configuration

Static files (CSS, JS, images) are managed using these settings:

STATIC\_URL = '/static/'

STATIC\_ROOT = BASE\_DIR / 'staticfiles'

STATICFILES\_DIRS = [

BASE\_DIR / 'static',

]

API Keys and Secrets

Avoid hardcoding secrets in settings.py. Use environment variables instead.

**In settings.py:**

SECRET\_KEY = os.environ.get

STRIPE\_API\_KEY = os.environ.get

**Q15. Setting up a Django REST Framework project.**

Ans. First, set up a virtual environment and install Django.

python -m venv env

pip install django djangorestframework

django-admin startproject myproject

Create an App

python manage.py startapp api

Add your app and REST framework to INSTALLED\_APPS in myproject/settings.py:

INSTALLED\_APPS = [

'rest\_framework',

'api',

]

Create a Model, Run migrations:python manage.py makemigrations

python manage.py migrate, Create a Serializer, Create API Views, Set Up URLs, Run the Server.

**Q16. What is CRUD, and why is it fundamental to backend development**

Ans. CRUD stands for:

Create — Add new data to the database

Read — Retrieve data

Update — Modify existing data

Delete — Remove data

CRUD operations are the **core functions** of almost any backend system that manages data.

For Data Management, RESTful APIs Are Built Around CRUD, Foundation for All Features, Scalability and Maintenance, Testing and Validation.