General Interview Questions on RESTful Web Services

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Q1) Explain what is REST and RESTFUL?

- REST was first introduced by Roy Fielding in 2000.
- REST stands for REpresentational State Transfer. It means that each unique URL is a represention of some object. it focuses on how state of resource should be transported over HTTP protocol to different clients written in different languages.
- REST is an architectural style.
- It defines several Rules/Guide Lines to develop Web APIs/Web Services very easily in concise way. Hence REST is the most popular Architecture to develop Web Services.

RESTFul API:

The API which is developed by using REST Architecture is nothing but RESTFul API. i.e interface between the user and application where API implements REST Architecture. In RESTFUL web service HTTP methods like GET, POST, PUT, PATCH and DELETE can be used to perform CRUD operations

Note: REST is basically an architecture where as RESTFul API is an API that implements REST.

Q2) Explain the architectural style for creating any web API based on REST?

The architectural style for creating web api are

1) We can use HTTP for client server communication
2) We can use XML/JSON to send and receive messages. i.e XML/JSON acts as formatting language.
3) Each resource/service can be accessed by a unique URL. This URL acts as the address for the resource/service.
4) Stateless communication

Q3) Explain some Key characterics of REST?

The following are various important key characterics of REST

1) REST is a stateless and hence SERVER has no state(or session data)
2) With a well-applied REST API, the server can be restarted without any impact on the client
3) Web Services mostly allow
   - GET → To access resources
   - POST → To create resources
   - PUT/PATCH → To update resources
   - DELETE → To delete resources

Q4) Which protocol is used by RESTful Web services?

RESTful web services use the HTTP protocol as a medium of communication between the client and the server.

Q5) What is a "Resource" in REST?

REST architecture treats any content as a resource, which can be either database record, text files, HTML pages, images, videos or dynamic business information. Consumer application can send

- GET request to access a resource,
- POST request to create a resource,
- PUT/PATCH request to update a resource,
- DELETE request to delete a resource.

Q6) Explain Which markup languages used in the REST API?

- JSON and XML are the most commonly used markup languages in the rest api.
- But these days JSON is commonly used because of light weight, high performance and less bandwidth requirements.

Q7) What is the most popular way to represent a resource in REST?

XML and JSON are the most popular representations of resources.

Q8) What are various HTTP Methods supported by REST?

The following are various important HTTP Methods supported by REST

1) GET: To access all resources or a particular resource
2) POST: To create a new resource
3) PUT: To update an existing resource
4) PATCH: To perform partial updation of an existing resource
5) DELETE: To delete a particular resource
6) OPTIONS: It represents what are various HTTP Methods supported to access a resource
7) HEAD: It represents Header part of the GET Response. It provides Meta information.

Q9) What is idempotent request?

- By repeating the request multiple times, If we are getting the same response, such type of request is called Idempotent request.
- PUT is idempotent where as POST is not idempotent.

Q10) Explain the difference between POST and PUT?

- In general POST can be used to create a new resource where as PUT can be used to update an existing resource. PUT is idempotent where as POST is not idempotent.
- POST is not idempotent because it creates a new resource every time.
- PUT is idempotent as it won't create any new resource everytime and just updating existing resource.

Q11) Explain the difference between PUT and PATCH?

We can use PUT for full updation where as PATCH for partial updation of resources.

Q12) Explain the difference between GET and POST?

https://www.youtube.com/watch?v=E-Y5CeeYJxU

Q13) Explain the format of HTTP Request?

Explain the core components of HTTP Request?

Diagram

HTTP Request contains 3 parts

1) Request Line
2) Request Headers
3) Request Body

1) Request Line:

It contains 3 parts

1) Request Method like GET,POST etc
2) Resource Path (like /api/2/)
3) Protocol version used by browser(HTTP/1.1)

2) Request Headers:

It contains configuration information of the browser(like media types accepted by browser,encoding types supported by browser) and extra information about request body

3) Request Body:

It contains original information provided by the client.

Q14) Explain the format of HTTP Response?

Explain the core components of HTTP Response?

Diagram

HTTP Response contains 3 parts

1) Status Line
2) Response Headers
3) Response Body

1) Status Line:

It contains 3 parts

1) HTTP protocol version used by server (HTTP/1.1)
2) Status Code (200)

3) Description of the status code (OK)

2) **Response Headers:**
It contains extra information about response body like content type, content length etc

3) **Response Body:**
It contains original response (like JSON) which intended for client.

Q15) What is payload in RESTFul Web services?

The "payload" is the data what we are transporting from client application to server application.

Q16) What is the upper limit for a payload to pass in the POST method?

In GET request, the data will be appended to the service URL. There is a limit on length of URL. Hence limit is applicable for payload of GET request.

In POST request, the payload (data) will be encapsulated in request body, which is not having any size limit. Hence there is no limit for payload of POST request.

Q17) Explain the term 'Statelessness' with respect to RESTful WEB service?

Statelessness means complete isolation. Server won't maintain any information of the client. Every request to the server is treated as an independent new request. With every request client is responsible to send authentication information also like Tokens etc.

Q18) Explain advantages and disadvantages of 'Statelessness'?

Advantages:

1) Every request to the server is considered as independent request. i.e there are no dependencies to previous requests.
2) Any previous communication with the client and server is not maintained and hence the total process is simplified.
3) Every request contains complete information.
4) Without effecting client applications, we can restart server.

Disadvantages:

1) With every request, compulsory client should send extra information like authentication tokens etc. It is a burden to the client application.
2) It causes network traffic problems and require more bandwidth.

Q19) What is the caching mechanism?

Caching is the process in which server response is stored so that a cached copy can be used when required and there is no need of generating the same response again.

The main advantages of caching are:

1) **It reduces load on the server**
2) **It improves performance of the application**
3) **It improves the scalability of the application.**

Only the client is able to cache the response and that too for a limited period of time.

Mentioned below are the header of the resources and their brief description so that they can be identified for the caching process:

- **Time and Date of resource creation**
- **Time and date of resource modification that usually stores the last detail.**
- **Cache control header**
- **Time and date at which the cached resource will expire.**
- **The age which determines the time from when the resource has been fetched.**

Q20) What is status code and what are various possible HTTP status codes?

HTTP status code represent the status of the response like success or fail etc. The following are various possible HTTP status codes

1XX → Informational
2XX → Successful
3XX → Redirection
4XX → Client Error
5XX → Server Error

Q21) List out some common status codes experienced in your previous project?

Eg 1:

Code 200: This indicates success.
If we send GET request and the requested resource is available then we will get response with 200 status code.

Eg 2:

Code 201: This indicates resource has been successfully created.
If we send POST request and if the resource created successfully then we will get 201 status code response

Eg 3:

Code 204: This indicates that there is no content in the response body.

If we send DELETE request and if the resource deleted successfully then we will get 204 status code response.

Eg 4:

Code 404: This indicates that there is no method available.
If we send GET request and if the requested resource is not available then we will get 404 status code response.

Eg 5:

code 400
It means, BAD REQUEST, states that invalid input is provided e.g. validation error, missing data.

Eg 6:

401 Unauthorized
If authentication information is not provided when we will get 401 Unauthorized response.

Eg 7:

403 Forbidden
If csrf verification fails then we will get 403 status code response.

Eg 8: 500
It means, INTERNAL SERVER ERROR, states that server has thrown some exception while processing our request.

Q22) What are the best practices that are to be followed while designing RESTful web services?

The following are various best practices while desiging RESTful web services

1) **Every input on the server should be validated.**
2) **Input should be well formed.**
3) **Never pass any sensitive data through URL.**
4) **For any session, the user should be authenticated.**
5) **Only HTTP error messages should be used for indicating any fault.**
6) **Use message format that is easily understood and is required by the client.**
7) **end point(URL) should be descriptive and easily understandable.**

Django REST Framework (DRF) Interview Questions

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Q1. Explain the following?

1. API
2. Web API
3. REST
4. RESTful Web API

Q2. What are the types of web services?

Q3. Explain the differences between SOAP and RESTful web services?

Q4. Explain the advantages and Limitations of SOAP Web Services?

Q5. Explain the advantages and Limitations of RESTful Web Services?

Q6. Write sample Django FBV to send JSON Response?

1st way (By using Python json module):

```
import json
def employee_data_jsonview(request):
    employee_data={'eno':100,'ename':'Sunny Leone','esal':1000,'eaddr':'Hyderabad'}
    json_data=json.dumps(employee_data)
    return HttpResponse(json_data,content_type='application/json')
```

2nd way (By using Django JsonResponse class):

```
from django.http import JsonResponse
def employee_data_jsondirectview(request):
    employee_data={'eno':100,'ename':'Sunny Leone','esal':1000,'eaddr':'Hyderabad'}
    return JsonResponse(employee_data)
```

Q7. Write sample Django CBV to send JSON Response?

```
from django.views.generic import View
class JsonCBV(View):
    def get(self,request,*args,**kwargs):
        employee_data={'eno':100,'ename':'Sunny Leone','esal':1000,'eaddr':'Hyderabad'}
        return JsonResponse(employee_data)
```

Q8. From the python program, how we can send HTTP Request?

By using requests module, we can send HTTP request from python script

eg:

```
import requests
BASE_URL='http://127.0.0.1:8000/'
ENDPOINT='api'
r=requests.get(BASE_URL+ENDPOINT)
data=r.json()
print('Employee Number:',data['eno'])
print('Employee Name:',data['ename'])
print('Employee Salary:',data['esal'])
```

print('Employee Address:',data['eaddr'])

Q9. From command prompt , how to send HTTP request?
By using cURL and HTTPie, we can send HTTP request from the command prompt.

Q10. What are Mixins and explain usage?
Mixins are special type of inheritance in Python.
It is limited version of Multiple inheritance.

In multiple inheritance, we can create object for parent class and parent class can extend other classes. But in Mixins, for the parent class we cannot create object and it should be direct child class of object.i.e parent class cannot extend any other classes.

In Multiple inheritance, parent class can contain instance variables.But in Mixins, parent class cannot contain instance variable but can contain class level static variables.

Hence the main purpose of parent class in Mixins is to provide functions to the child classes.

Note: Mixins meant for code reusability.

Q11. Explain differences between Mixins and Multiple Inheritance?
Table: Mixins | Multiple Inheritance
1. Parent class instantiation is not possible
Parent class instantitation is possible

2. It contains only instance methods but not instance variables
It contains both instance methods and variables

3. The methods are useful only for child classes.
The methods are useful for both Parent and child classes

4. Parent class should be direct child class of object
Parent class can extend any class need not be object.

Note:
1. Mixins are reusable classes in django.
2. Mixins are available only in languages which provide support for multiple inheritance like Python,Ruby,Scala etc
Mixins are not applicable for Java and C#,because these languages won't support multiple inheritance.

Q12. Explain the usage of Mixin with an example?
Writing CBV by using Mixin class:

mixins.py:

```
----------
from django.http import JsonResponse
class JsonResponseMixin(object):
   def render_to_json_response(self,context,**kwargs):
      return JsonResponse(context,**kwargs)
```

CBV:

```
----
from testapp.mixins import JsonResponseMixin
class JsonCBV2(JsonResponseMixin,View):
   def get(self,request,*args,**kwargs):
      employee_data={'eno':100,'ename':'Sunny Leone','esal':1000,'eaddr':'Hyderabad'}
      return self.render_to_json_response(employee_data)
```

Q13. In Django REST Framework, what are various predefined Mixin classes available?

The following are various Mixin classes provided by DRF.

1. ListModelMixin
2. CreateModelMixin
3. RetrieveModelMixin
4. UpdateModelMixin
5. DestroyModelMixin

Q14. What is serialization?

Converting object from one form to another form is called serialization.

eg: converting python dict object to JSON

Q15. In how many ways, we can perform serialization?

1st way: By using python's json module:

```
---------------------------------------
import json
json_data=json.dumps(python_dict)
```

2nd way: By using django.core.serializers module:

```
-----------------------------------------------
from django.core.serializers import serialize
def get(self,request,id,*args,**kwargs):
      emp=Employee.objects.get(id=id)
      json_data=serialize('json',[emp,],fields=('eno','ename'))
      return HttpResponse(json_data,content_type='application/json')
```

3rd way: By using DRF Serializers:

---
models.py:

```
from django.db import models
class Employee(models.Model):
   eno=models.IntegerField()
   ename=models.CharField(max_length=64)
   esal=models.FloatField()
   eaddr=models.CharField(max_length=64)
```
serializers.py:

```
from rest_framework import serializers
class EmployeeSerializer(serializers.Serializer):
   eno=serializers.IntegerField()
   ename=serializers.CharField(max_length=64)
   esal=serializers.FloatField()
   eaddr=serializers.CharField(max_length=64)
```

Converting Employee Object to Python Native Data Type By using EmployeeSerializer(Serialization Process):

```
>>> from testapp.models import Employee
>>> from testapp.serializers import EmployeeSerializer
>>> emp=Employee(eno=100,ename='Durga',esal=1000,eaddr='Hyd')
>>> eserializer=EmployeeSerializer(emp)
>>> eserializer.data
{'eno': 100, 'ename': 'Durga', 'esal': 1000.0, 'eaddr': 'Hyd'}
```

Just we converted Employee object to python native data type(dict)

Converting Python native data type to JSON:

```
>>> from rest_framework.renderers import JSONRenderer
>>> json_data=JSONRenderer().render(eserializer.data)
>>> json_data
b'{"eno":100,"ename":"Durga","esal":1000.0,"eaddr":"Hyd"}'
```

How to perform serialization for QuerySet:

```
>>> qs=Employee.objects.all()
>>> qs
<QuerySet [<Employee: Employee object>, <Employee: Employee object>]>
>>> eserializer=EmployeeSerializer(qs,many=True)
>>> eserializer.data
```

```
[OrderedDict([('eno', 100), ('ename', 'Durga'), ('esal', 1000.0), ('eaddr', 'Hyderabad')]),
OrderedDict([('eno', 200), ('ename', 'Bunny'), ('esal', 2000.0), ('eaddr', 'Mumbai')])]
>>> json_data=JSONRenderer().render(eserializer.data)
>>> json_data
b'[{"eno":100,"ename":"Durga","esal":1000.0,"eaddr":"Hyderabad"},{"eno":200,"ename
":"Bunny","esal":2000.0,"ead
dr":"Mumbai"}]'
```

Q16. How to use dumpdata option, to display database data to the console?

We can dump our database data either to the console or to the file by using dumpdata option. This option provides support for json and xml formats. The default format is json. We can write this data to files also.

commands:

```
---------
py manage.py dumpdata testapp.Employee
  print data to the console in json format without identation
py manage.py dumpdata testapp.Employee --indent 4
  print data to the console in json format with identation
py manage.py dumpdata testapp.Employee >emp.json --indent 4
  write data to emp.json file instead of displaying to the console
py manage.py dumpdata testapp.Employee --format json >emp.json --indent 4
  we are specifying format as json explicitly
py manage.py dumpdata testapp.Employee --format xml  --indent 4
  print data to the console in xml format with identation
py manage.py dumpdata testapp.Employee --format xml > emp.xml --indent 4
  write data to emp.xml file instead of displaying to the console
```

Note: dumpdata option provides support only for 3 formats
1. json(default)
2. xml
3. YAML (YAML Ain't Markup Language)

YAML is a human readable data serialization language, which is supported by multiple languages.

Q17. How to disable CSRF verification?

Just for our testing purposes we can disable CSRF verification, but not recommended in production environment.

We can disable CSRF verification at Function level,class level or at project level

1. To disable at function/method level:

```
from django.views.decorators.csrf import csrf_exempt

@csrf_exempt
def my_view(request):
   body
```

This approach is helpful for Function Based Views(FBVs)

2. To disable at class level:

If we disable at class level then it is applicable for all methods present inside that class.
This approach is helpful for class based views(CBVs).

code:

```
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator

@method_decorator(csrf_exempt,name='dispatch')
class EmployeeListCBV(SerializeMixin,View):
                pass
```

3. To disable at project level globally:

---

inside settings.py file comment the following middleware

'django.middleware.csrf.CsrfViewMiddleware'

Q18. Without using any 3rd party REST Frameworks(like DRF), implement the following HTTP Methods functionality?

1. GET/List/Retrieve
2. POST/Create
3. PUT/Update
4. DELETE/destroy

Answer:

models.py:

```
from django.db import models

# Create your models here.
class Employee(models.Model):
```

```
    eno=models.IntegerField()
    ename=models.CharField(max_length=64)
    esal=models.FloatField()
    eaddr=models.CharField(max_length=64)
```

admin.py:

```
---------
from django.contrib import admin
from testapp.models import Employee
# Register your models here.
class EmployeeAdmin(admin.ModelAdmin):
    list_display=['id','eno','ename','esal','eaddr']

admin.site.register(Employee,EmployeeAdmin)
```

forms.py:

```
---------
from django import forms
from testapp.models import Employee
class EmployeeForm(forms.ModelForm):
    #validations
    def clean_esal(self):
        inputsal=self.cleaned_data['esal']
        if inputsal < 5000:
            raise forms.ValidationError('The minimum salary should be 5000')
        return inputsal

    class Meta:
        model=Employee
        fields='__all__'
```

mixins.py:

```
---------
from django.core.serializers import serialize
import json
class SerializeMixin(object):
    def serialize(self,qs):
        json_data=serialize('json',qs)
        pdict=json.loads(json_data)
        final_list=[]
        for obj in pdict:
            final_list.append(obj['fields'])
        json_data=json.dumps(final_list)
        return json_data
```

```
from django.http import HttpResponse
class HttpResponseMixin(object):
    def render_to_http_response(self,data,status=200):
        return HttpResponse(data,content_type='application/json',status=status)
utils.py:
---------
import json
def is_json(data):
    try:
        real_data=json.loads(data)
        valid=True
    except ValueError:
        valid=False
    return valid
urls.py:
--------
from django.conf.urls import url
from django.contrib import admin
from testapp import views

urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^api/$', views.EmployeeCRUDCBV.as_view()),
    # url(r'^api/$', views.EmployeeListCBV.as_view()),
]

views.py:
---------
from django.shortcuts import render
from django.views.generic import View
from testapp.models import Employee
import json
from django.http import HttpResponse
from django.core.serializers import serialize
from testapp.mixins import SerializeMixin,HttpResponseMixin
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from testapp.utils import is_json
from testapp.forms import EmployeeForm

@method_decorator(csrf_exempt,name='dispatch')
class EmployeeCRUDCBV(HttpResponseMixin,SerializeMixin,View):
    def get_object_by_id(self,id):
        try:
            emp=Employee.objects.get(id=id)
```

```
        except Employee.DoesNotExist:
            emp=None
        return emp

    def get(self,request,*args,**kwargs):
        data=request.body
        if not is_json(data):
            return self.render_to_http_response(json.dumps({'msg':'plz send valid json data
only'}),status=400)
        data=json.loads(request.body)
        id=data.get('id',None)
        if id is not None:
            obj=self.get_object_by_id(id)
            if obj is None:
                return self.render_to_http_response(json.dumps({'msg':'No Matched Record
Found with Specified Id'}),status=404)
            json_data=self.serialize([obj,])
            return self.render_to_http_response(json_data)
        qs=Employee.objects.all()
        json_data=self.serialize(qs)
        return self.render_to_http_response(json_data)

    def post(self,request,*args,**kwargs):
        data=request.body
        if not is_json(data):
            return self.render_to_http_response(json.dumps({'msg':'plz send valid json data
only'}),status=400)
        empdata=json.loads(request.body)
        form=EmployeeForm(empdata)
        if form.is_valid():
            obj = form.save(commit=True)
            return self.render_to_http_response(json.dumps({'msg':'resource created
successfully'}))
        if form.errors:
            json_data=json.dumps(form.errors)
            return self.render_to_http_response(json_data,status=400)
    def put(self,request,*args,**kwargs):
        data=request.body
        if not is_json(data):
            return self.render_to_http_response(json.dumps({'msg':'plz send valid json data
only'}),status=400)
        data=json.loads(request.body)
        id=data.get('id',None)
        if id is None:
```

```
            return self.render_to_http_response(json.dumps({'msg':'To perform updation id is
mandatory,you should provide'}),status=400)
        obj=self.get_object_by_id(id)
        if obj is None:
            json_data=json.dumps({'msg':'No matched record found, Not possible to perform
updataion'})
            return self.render_to_http_response(json_data,status=404)

        new_data=data
        old_data={
        'eno':obj.eno,
        'ename':obj.ename,
        'esal':obj.esal,
        'eaddr':obj.eaddr,
        }
        # for k,v in new_data.items():
        #     old_data[k]=v
        old_data.update(new_data)
        form=EmployeeForm(old_data,instance=obj)
        if form.is_valid():
            form.save(commit=True)
            json_data=json.dumps({'msg':'Updated successfully'})
            return self.render_to_http_response(json_data,status=201)
        if form.errors:
            json_data=json.dumps(form.errors)
            return self.render_to_http_response(json_data,status=400)
    def delete(self,request,*args,**kwargs):
        data=request.body
        if not is_json(data):
            return self.render_to_http_response(json.dumps({'msg':'plz send valid json data
only'}),status=400)
        data=json.loads(request.body)
        id=data.get('id',None)
        if id is None:
            return self.render_to_http_response(json.dumps({'msg':'To perform delete, id is
mandatory,you should provide'}),status=400)
        obj=self.get_object_by_id(id)
        if obj is None:
            json_data=json.dumps({'msg':'No matched record found, Not possible to perform
delete operation'})
            return self.render_to_http_response(json_data,status=404)
        status,deleted_item=obj.delete()
        if status==1:
            json_data=json.dumps({'msg':'Resource Deleted successfully'})
            return self.render_to_http_response(json_data,status=201)
```

```
        json_data=json.dumps({'msg':'unable to delete ...plz try again'})
        return self.render_to_http_response(json_data,status=500)
```

test.py:

```
--------
import requests
import json
BASE_URL='http://127.0.0.1:8000/'
ENDPOINT='api/'
def get_resources(id=None):
    data={}
    if id is not None:
        data={
        'id':id
        }
    resp=requests.get(BASE_URL+ENDPOINT,data=json.dumps(data))
    print(resp.status_code)
    print(resp.json())
def create_resource():
    new_emp={
    'eno':2000,
    'ename':'Katrina',
    'esal':20000,
    'eaddr':'Mumbai',
    }
    r=requests.post(BASE_URL+ENDPOINT,data=json.dumps(new_emp))
    print(r.status_code)
    # print(r.text)
    print(r.json())
create_resource()
def update_resource(id):
    new_data={
    'id':id,
    'eno':7777,
    'ename':'Kareena',
    'eaddr':'Lanka',
    'esal':15000
    }
    r=requests.put(BASE_URL+ENDPOINT,data=json.dumps(new_data))
    print(r.status_code)
    # print(r.text)
    print(r.json())
def delete_resource(id):
    data={
    'id':id,
```

```
}
r=requests.delete(BASE_URL+ENDPOINT,data=json.dumps(data))
print(r.status_code)
# print(r.text)
print(r.json())
```

Q19. What are various frameworks are available to develop WEB APIs for Django Application?

1. Tastify
2. Django REST Framework(DRF)

etc

Q20. What are various specialities of DRF?

Some reasons we might want to use Django REST framework:

1. The Web browsable API is a huge usability win for developers.
2. Authentication policies including packages for OAuth1a and OAuth2.
3. Serialization that supports both ORM and non-ORM data sources.
4. Customizable all the way down
5. Extensive documentation, and great community support.
6. Used and trusted by internationally recognised companies including Mozilla, Red Hat, Heroku, and Eventbrite.

Q21. In DRF, What are the jobs of Serializers?

DRF Serializers are responsible for the following activities

1. Serialization
2. Deserialization
3. Validation

Q22. How to implement validations by using DRF Serializers?

We can implement validations by using the following 3 ways

1. Field Level Validations
2. Object Level Validations
3. By using validators

1. Field Level Validations:

---

syntax:

validate_fieldname(self,value):

eg: To check esal should be minimum 5000

```
class EmployeeSerializer(serializers.Serializer):
   ....

   def validate_esal(self,value):
      if value<5000:
         raise serializers.ValidationError('Employee Salaray Should be Minimum 5000')
      return value
```

2. Object Level Validations:

If we want to perform validations for multiple fields simultaneously then we should go for object level validations.

eg: If ename is 'Sunny' then salary should be minimum 60000

```
def validate(self,data):
      ename=data.get('ename')
      esal=data.get('esal')
      if ename.lower()=='sunny':
         if esal<60000:
            raise serializers.ValidationError('Sunny Salary should be minimum 60K')
      return data
```

use cases:

1. first entered pwd and re-entered pwd must be same.

2. First entered account number and re-entered account number must be same

These validations we can implement at object level.

3. Validations by using validator field:

```
def multiples_of_1000(value):
   if value % 1000 != 0:
      raise serializers.ValidationError('Salary should be multiples of 1000s')

class EmployeeSerializer(serializers.Serializer):
   ...
   esal=serializers.FloatField(validators=[multiples_of_1000,])
   ..
```

Note: If we implement all 3 types of validations then the order of priority is

1. validations by using validator
2. validations at field level
3. validations at object level

Q23. In DRF, what are various types of serializers?
There are multiple types of serializers like

1. Simple/Normal Serializer
2. ModelSerializer
3. Nested Serializer
etc

Q24. What are various advantages of using ModelSerializer when compared with Normal Serializer?

If our serializable objects are Django model objects, then it is highly recommended to go for ModelSerializer.
ModelSerializer class is exactly same as regular serializer classe except the following differences

1. The fields will be considered automatically based on the model and we are not required to specify explicitly.

2. It provides default implementation for create() and update() methods.

Note: ModelSerializer won't provide any extra functionality and it is just for typing shortcut.

We can define ModelSerializer class as follows:

```
class EmployeeSerializer(serializers.ModelSerializer):
   class Meta:
      model=Employee
      fields='__all__'
```

Here we are not required to specify fields and these will be considered automatically based on Model class. We are not required to implement create() and update() methods, because ModelSerializer class will provide these methods.

Q25. In How many ways we can specify Fields in ModelSerializer?

3 ways

1. To include all fields
```
  fields='__all__'
```

2. To include only some fields
   fields=('eno','ename','eaddr')
   This approach is helpful if we want to include very less number of fields.

3. To exclude some fields
   exclude=('esal',)

   Except esal, all remaining fields will be considered.
   If we want to consider majarity of the fileds then this approach is helpful.

Q26. In DRF, in how many ways we can define View classes?

DRF provides 2 classes to define business logic for our API Views.

1. APIView
2. ViewSet

Q27. By using APIView class, how we can perform CRUD operations?
    How to implement GET,POST,PATCH,PUT,DELETE Methods?

views.py:

```
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework.response import Response
from testapp.serializers import NameSerializer
# Create your views here.
class TestApiView(APIView):
    def get(self,request,format=None):
        colors=['RED','BLUE','GREEN','YELLOW','INDIGO']
        return Response({'msg':'Welcome to Colorful Year','colors':colors})
    def post(self,request):
        serializer=NameSerializer(data=request.data)
        if serializer.is_valid():
            name=serializer.data.get('name')
            msg='Hello {} Wish You Happy New Year !!!'.format(name)
            return Response({'msg':msg})
        return Response(serializer.errors,status=400)
    def put(self,request,pk=None):
        return Response({'msg':'Response from put method'})
    def patch(self,request,pk=None):
        return Response({'msg':'Response from patch method'})
    def delete(self,request,pk=None):
        return Response({'msg':'Response from delete method'})
```

serializers.py:

```
---------------
from rest_framework import serializers
class NameSerializer(serializers.Serializer):
   name = serializers.CharField(max_length=7)
```

urls.py:

```
--------
from django.conf.urls import url,include
from django.contrib import admin
from testapp import views

urlpatterns = [
   url(r'^admin/', admin.site.urls),
   url(r'^api/', views.TestApiView.as_view()),
]
```

Q28. If we are using APIView class to define CBV, then which methods we have to provide implementation?

get(),post(),put(),patch(),delete()

Q29. How to implement CBV by using ViewSet?

Complete Application:

```
---------------------
views.py:
---------
from rest_framework.response import Response
from testapp.serializers import NameSerializer
from rest_framework import viewsets
class TestViewSet(viewsets.ViewSet):
   def list(self,request):
      colors=['RED','GREEN','YELLOW','ORANGE']
      return Response({'msg':'Wish YOu Colorful Life in 2019','colors':colors})
   def create(self,request):
      serializer=NameSerializer(data=request.data)
      if serializer.is_valid():
         name=serializer.data.get('name')
         msg='Hello {} Your Life will be settled in 2019'.format(name)
         return Response({'msg':msg})
      return Response(serializer.errors,status=400)
   def retrieve(self,request,pk=None):
      return Response({'msg':'Response from retrieve method'})
   def update(self,request,pk=None):
```

```
        return Response({'msg':'Response from update method'})
    def partial_update(self,request,pk=None):
        return Response({'msg':'Response from partial_update method'})
    def destroy(self,request,pk=None):
        return Response({'msg':'Response from destroy method'})
```

serializers.py:

```
----------
from rest_framework import serializers
class NameSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=7)
```

urls.py:

```
---------
from django.conf.urls import url,include
from django.contrib import admin
from testapp import views
from rest_framework import routers
router=routers.DefaultRouter()
router.register('test-viewset',views.TestViewSet,base_name='test-viewset')


urlpatterns = [
    url(r'^admin/', admin.site.urls),
    # url(r'^api/', views.TestApiView.as_view()),
    url(r'',include(router.urls))
]
```

Q30. If we are using ViewSet to develop CBV, then which methods we have to provide implementation?

list(),create(),retrieve(),update(),partial_update(),destroy()

Q31. Explain the differences between APIView and ViewSet?

Table: APIView | ViewSet

1. present in rest_framework.views module
present in rest_framework.viewsets modules

2. Method Names reflect HTTP Methods like get(),post(),put(),patch(),delete()
Method Names reflect Database Model class actions/operations like list(),retrieve(),create(),update(),partial_update() and destroy()

3. We have to map views to urls explicitly
We are not required to map views to urls explicitly. DefaultRouter will takes care url maapings automatically

4. Most of the business logic we have to write explicitly
Most of the business logic will be generated automatically

5. Length of the code is more

Length of the code is less
6. API Development time is more
API Development time is less
7. Developer has complete control over the logic
Developer won't have complete control over the logic.
8. Clear Execution Flow is possible
Clear Execution Flow is not possible
9. Best suitable for complex operations like using multiple data sources simultaneously, calling other APIs etc
Best suitable for developing simple APIs like developing CRUD interface for database models.

Q32. What is the role of Router in ViewSet?
In APIViews, we have to map views to urls manually. But in ViewSet,we are not required to do explicitly. DRF provides a 'DefaultRouter' class to map ViewSet to the urls,which are used by partner application.

Routers provide an easy way of automatically determining the URL configurations.
Routers are required only for views developed by ViewSet.

Q33. What are various generic APIViews available in DRF?
To perorm CRUD operations, DRF provides the following generic APIView classes

ListAPIView
CreateAPIView
RetrieveAPIView
UpdateAPIView
DestroyAPIView
ListCreateAPIView
RetrieveUpdateAPIView
RetrieveDestroyAPIView
RetrieveUpdateDestroyAPIView

Q34. How to develop API to perform CRUD operations by using generic APIView classes?

views.py:

```
class EmployeeListCreateAPIView(generics.ListCreateAPIView):
    queryset=Employee.objects.all()
    serializer_class=EmployeeSerializer
class EmployeeRetrieveUpdateDestroyAPIView(generics.RetrieveUpdateDestroyAPIView):
    queryset=Employee.objects.all()
    serializer_class=EmployeeSerializer
    lookup_field='id'
```

urls.py:

```
url(r'^api/$', views.EmployeeListCreateAPIView.as_view()),
url(r'^api/(?P<id>\d+)/$', views.EmployeeRetrieveUpdateDestroyAPIView.as_view()),
```

Q35. How to develop API to perform CRUD operations by using Mixin classes?

Demo Application:

```
from rest_framework import mixins
class EmployeeListModelMixin(mixins.CreateModelMixin,generics.ListAPIView):
    queryset=Employee.objects.all()
    serializer_class=EmployeeSerializer
    def post(self,request,*args,**kwargs):
        return self.create(request,*args,**kwargs)

class
EmployeeDetailAPIViewMixin(mixins.UpdateModelMixin,mixins.DestroyModelMixin,gen
erics.RetrieveAPIView):
    queryset=Employee.objects.all()
    serializer_class=EmployeeSerializer
    def put(self,request,*args,**kwargs):
        return self.update(request,*args,**kwargs)
    def patch(self,request,*args,**kwargs):
        return self.partial_update(request,*args,**kwargs)
    def delete(self,request,*args,**kwargs):
        return self.destroy(request,*args,**kwargs)

url(r'^api/$', views.EmployeeListModelMixin.as_view()),
url(r'^api/(?P<pk>\d+)/$', views.EmployeeDetailAPIViewMixin.as_view()),
```

Q36) By using which tools we can test functionality of our api?

There are multiple tools are available

1. POSTMAN
2. Swagger
3. DRF Browsable API
4. We can write our own client(like Python script) to test functionality
5. From the command prompt by using HTTPie or cURL

Q37) What is Authentication and what are various authentication mechanisms we can apply in DRF?

The process of validating user is called authentication. Most of the times we can perform authentication by using username and password combination or by using tokens etc

DRF provides several inbuilt authentication mechanisms
1. Basic Authentication
2. Session Authentication
3. Token Authentication
4. JWT(Json Web Token) Authentication
etc

Note: By using DRF, we can implement our own custom authentication mechanism also

Q38) What is Autherization and what are various predefined authorization mechanisms available in DRF?

The process of validating access permissions of user is called authorization.
DRF provides several permission classes for authorization

1. AllowAny
2. IsAuthenticated
3. IsAdminUser
4. IsAuthenticatedOrReadOnly
5. DjangoModelPermissions
6. DjangoModelPermissionsOrAnonReadOnly
etc

Q39) How to enable authentication and authorization for django rest api globally?

In the settings.py, we have to add the following configurations

```
REST_FRAMEWORK={
'DEFAULT_AUTHENTICATION_CLASSES':('rest_framework.authentication.TokenAuthentication',),
'DEFAULT_PERMISSION_CLASSES':('rest_framework.permissions.IsAuthenticated',)

}
```

Q40) How to enable authentication and authorization for django rest api locally for a particular View class?

our application may contain several view classes. If we want to enable authentication and authorization for a particular view class then we have to use this local approach.

```
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import IsAuthenticated
class EmployeeCRUDCBV(ModelViewSet):
```

```
...
authentication_classes=[TokenAuthentication,]
permission_classes=[IsAuthenticated,]
```

Q41) Explain the following Permission classes functionality?

1. AllowAny
2. IsAuthenticated
3. IsAdminUser
4. IsAuthenticatedOrReadOnly
5. DjangoModelPermissions
6. DjangoModelPermissionsOrAnonReadOnly

1. AllowAny:

The AllowAny permission class will allow unrestricted access irrespective of whether request is authenticated or not.

This is default value for permission-class. It is very helpful to allow unrestricted access for a particular view class if global settings are enabled.

2. IsAuthenticated:

The IsAuthenticated permission class will deny permissions to any unauthorized user. ie only authenticated users are allowed to access endpoint.

This permission is suitable, if we want our API to be accessible by only registered users.

Note:
We can send Token in postman inside Headers Section
Key: Authorization
Value: Token 3639020972202cc1d25114ab4a5f54e6078184a4

3. IsAdminUser:

If we use IsAdminUser permission class then only AdminUser is allowed to access.i.e the users where is_staff property is True.

This type of permission is best suitable if we want our API to be accessible by only trusted administrators.

If the user is not admin and if he is trying to access endpoint then we will get 403 status code error response saying:

```
{
   "detail": "You do not have permission to perform this action."
}
```

Note:

Normal User ---> can not access admin interface
Admin---> can login to admin interface but no permissions to modify anything
Superuser---> can login to admin interface and can modify anything

4. IsAuthenticatedOrReadOnly:

To perform read operation (safe methods:GET,HEAD,OPTIONS) authentication is not required. But for the remaining operations(POST,PUT,PATCH,DELETE) authentication must be required.

If any person is allowed to perform read operation and only registered users are allowed to perform write operation then we should go for this permission class.

eg: In IRCTC application, to get trains information(read operation) registration is not required. But to book tickets(write operation) login must be required.

5. DjangoModelPermissions :

This is the most powerful permission class. Authorization will be granted iff user is authenticated and has the relevant model permissions.

DjangoModelPermissions = Authentication + Model Permissions

If the user is not authenticated(we are not providing token) then we will get 401 Unauthorized error message saying

```
{
   "detail": "Authentication credentials were not provided."
}
```

If we are providing Token (authenticated) but not having model permissions then we can perform only GET operation. But to perform POST,PUT,PATCH,DELETE compulsory model permissions must be required,otherwise we will get 403 Forbidden error message saying

```
{
   "detail": "You do not have permission to perform this action."
}
```

How to provide model permissions:

To perform POST operation the required model permission is 'add'
To perform PUT,PATCH operations the required model permission is 'change'
To pderform DELETE operation the required model permission is 'delete'

We have to provide these model permissions in admin interface under User permissions:

testapp | employee | Can change employee
testapp | employee | Can add employee
testapp | employee | Can delete employee

Note:
1. DjangoModelPermissions class is more powerful and we have complete control on permissions.
*****2. For superuser we are not required to give model permissions explicitly and already avaialble**

6. DjangoModelPermissionsOrAnonReadOnly:

It is exactly same as DjangoModelPermissions class except that it allows unauthenticated users to have read-only access to the API.

eg:

```
from rest_framework.viewsets import ModelViewSet
from testapp.models import Employee
from testapp.serializers import EmployeeSerializer
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import
IsAuthenticated,AllowAny,IsAdminUser,IsAuthenticatedOrReadOnly,DjangoModelPermiss
ions,DjangoModelPermissionsOrAnonReadOnly
class EmployeeCRUDCBV(ModelViewSet):
   queryset=Employee.objects.all()
   serializer_class=EmployeeSerializer
   authentication_classes=[TokenAuthentication,]
   permission_classes=[DjangoModelPermissionsOrAnonReadOnly,]
```

Q42. Explain how to define Custom Permissions with an exampl ?

Based on our programming requirement, we can define our own permission classes also. We have to create child class for BasePermission class and we have to override has_permission() method.

eg1: Define our own Permission class which allows only SAFE_METHODS (GET,HEAD,OPTIONS)

permissions.py:

```
--------------
from rest_framework.permissions import BasePermission,SAFE_METHODS
class IsReadOnly(BasePermission):
   def has_permission(self,request,view):
      if request.method in SAFE_METHODS:
         return True
      else:
         return False
```

views.py:

```
---------
from rest_framework.viewsets import ModelViewSet
from testapp.models import Employee
from testapp.serializers import EmployeeSerializer
from rest_framework.authentication import TokenAuthentication
from rest_framework.permissions import
IsAuthenticated,AllowAny,IsAdminUser,IsAuthenticatedOrReadOnly,DjangoModelPermiss
ions,DjangoModelPermissionsOrAnonReadOnly
from testapp.permissions import IsReadOnly
class EmployeeCRUDCBV(ModelViewSet):
   queryset=Employee.objects.all()
   serializer_class=EmployeeSerializer
   authentication_classes=[TokenAuthentication,]
   permission_classes=[IsReadOnly,]
```

eg2: Defining our own permission class which allows only GET and PATCH methods.

```
from rest_framework.permissions import BasePermission
class IsGETOrPatch(BasePermission):
   def has_permission(self,request,view):
      allowed_methods=['GET','PATCH']
      if request.method in allowed_methods:
         return True
      else:
         return False
```

eg3: Define our own permission class with the following requirement:

If the name is sunny then allow all methods
If the name is not sunny and the name contains even number of characters then allow only safe methods otherwise not allowed to perform any operation

```
from rest_framework.permissions import BasePermission,SAFE_METHODS
```

```
class SunnyPermission(BasePermission):
   def has_permission(self,request,view):
      username=request.user.username
      if username.lower()=='sunny':
         return True
      elif username != '' and len(username) %2 == 0 and request.method in
SAFE_METHODS:
         return True
      else:
         return False
```

Q43. In your previous project, which type of authentication used?

Either TokenAuthentication or JWT Authentication

Q44. What are various limitations of Token Authentication?
Explain the differences between TokenAuthentication and JWT Authentication?

In TokenAuthentication, all tokens will be stored in database table(Tokens table). For every request DRF will communicate with the database to validate token and to identify corresponding user. This database interaction for every request creates performance problems. Hence scalability of api will be down.

To over come this problem, we should go for JWT Authentication. The main advantage of JWT Authentication over TokenAuthentication is database interaction is not required to identify user. From the token itself,DRF can identify user, which improves performance and scalability of the application.
Because of this advantage,JWTAuthentication is the most commonly used type of authentication in real time.

Q45. How to implement the following authentication mechanisms?
1. Basic Authentication
2. Session Authentication
3. Token Authentication
4. JWT(Json Web Token) Authentication

Q46. Explain with an example, the process of implementing Custom Authentication?

Based on our requirement,we can implement our own authentication.

process:
1. We have to write our Custom Authentication class by extending from BaseAuthentication.
2. We have to override authenticate() method.

3. Returns a tuple of (user,None) for successful authentication
4. Raise AuthenticationFailed exception for failed authentication.

authentications.py:

```
------------------
from rest_framework.authentication import BaseAuthentication
from rest_framework.exceptions import AuthenticationFailed
from django.contrib.auth.models import User
class CustomAuthentication(BaseAuthentication):
   def authenticate(self,request):
      username=request.GET.get('username')
      if username is None:
         return None
      try:
         user=User.objects.get(username=username)
      except User.DoesNotExist:
         raise AuthenticationFailed('No such type of user')
      return (user,None)
```

views.py:

```
--------
from testapp.authentications import CustomAuthentication
class EmployeeCRUDCBV(ModelViewSet):
   queryset=Employee.objects.all()
   serializer_class=EmployeeSerializer
   authentication_classes=[CustomAuthentication,]
   permission_classes=[IsAuthenticated,]
```

We can send request for the endpoint as follows
http://127.0.0.1:8000/api/?username=durga

Demo Application-2 for Custom Authentication:

Requirements:

1. Client required to send secret key and username as query parameters.
2. Length of key should be 7 characters
3. The first character should be lower case alphabet symbol which should be last character of username
4. The third character should be 'Z'
5. The 5 th character should be first character of username

eg:

username: durga
secrete key: a7ZXd98

http://127.0.0.1:8000/api/?username=durga&key=a7ZXd98

authentications.py:

```
from rest_framework.authentication import BaseAuthentication
from django.contrib.auth.models import User
from rest_framework.exceptions import AuthenticationFailed
class CustomAuthentication2(BaseAuthentication):
    def authenticate(self,request):
        username=request.GET.get('username')
        key=request.GET.get('key')
        if username is None or key is None:
            return None
        c1=len(key) == 7
        c2=key[0]==username[-1].lower()
        c3=key[2]=='Z'
        c4=key[4]==username[0]
        try:
            user=User.objects.get(username=username)
        except User.DoesNotExist:
            raise AuthenticationFailed('Your provided username is  invalid,plz provide valid
username to access endpoint')
        if c1 and c2 and c3 and c4:
            return (user,None)
        raise AuthenticationFailed('Your provided key is  invalid,plz provide valid key to
access endpoint')
```

views.py:

```
from rest_framework.viewsets import ModelViewSet
from testapp.models import Employee
from testapp.serializers import EmployeeSerializer
from rest_framework.permissions import IsAuthenticated
from testapp.authentications import CustomAuthentication2
class EmployeeCRUDCBV(ModelViewSet):
    queryset=Employee.objects.all()
    serializer_class=EmployeeSerializer
    authentication_classes=[CustomAuthentication2,]
    permission_classes=[IsAuthenticated,]
```

Q47. DRF provides how many Pagination classes?
DRF provides several pagination classes to implement pagination.

1. PageNumberPagination
2. LimitOffsetPagination
3. CursorPagination

1. If we want all resources page by page then we should go for PageNumberPagination. Here we can specify only page_size and we cannot specify offset and ordering.

2. If we want the resources based on specified limit and offset then we should go for LimitOffsetPagination. Here we have choice to specify offset value from where we have to consider resources. We cannot specify ordering.

3. If we want all resources based on some ordering then we should go for CursorPagination. Here we can specify page_size and orderning and we cannot specify offset value.

Q48. How to implement Filtering and Ordering by using DRF?
Q49. When we should go for Nested Serializers?

Sometimes we can use one serializer inside another serializer to serialize dependent Model fields, such type of serializers are called Nested Serializers.

If Model mappings are there(like OneToOne,ManyToOne,ManyToMany) then we should go for Nested Serializers.

eg1: Assume there are two Models named with Author and Book. Book model has ForiegnKey reference to Author. While listing Author information, the corresponding Books information also required to provide. Hence inside AuthorSerializer, we required use BookSerializer. This concept is nothing but Nested Serializers.

syntax:

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

eg2: Assume there are two Models named with Musician and Album. Album model has ForiegnKey reference to Musician. While listing Musician information, the corresponding Albums information also required to provide. Hence inside MusicianSerializer, we required use AlbumSerializer. This concept is nothing but Nested Serializers.

syntax:

```
class MusicianSerializer(serializers.ModelSerializer):
   albums_by_musician=AlbumSerializer(read_only=True,many=True)
```

....

Q50. Have you ever consumed 3rd party external APIs in your django project?
Yes, to get geographic information based on ip address, we consumed ipstack api.

Q. Differences between SOAP and REST web services?
Types of web services:

There are 2 types of web services
1. SOAP Based WebServices
2. RESTful WebServices

1. SOAP Based Web Services:

SOAP: Simple Object Access Protocol

SOAP is an XML based protocol for accessing web services.
To describe SOAP based web services we have to use a special language: WSDL(Web Service Description Language)

SOAP based web services are more secured. We can consume by using RPC Method calls.
These web services can provide support for multiple protocols like HTTP,SMTP,FTP etc

Limitations:
----
1. SOAP Based web services will always provide data only in XML format. Parsing of this XML data is very slow,which creates performance problems.

2. Transfter of XML data over network requires more bandwidth

3. Implementing SOAP Based Web Services is very difficult.

Note: Because of heavy weight,less performance and more bandwidth requirements, SOAP based web services are not commonly used these days.

RESTful Web Services:

REST stands for Representational State Transfer. It means that each unique URL is a represention of some object. We can get contents of this object by using HTTP GET,we can modify by using PUT/PATCH and we can delete by using DELETE.
We can create by using POST.

Most of the times RESTful web service will provide data in the form of JSON, parsing is not difficult. Hence this type of web services are faster when compared with SOAP based Web Services.

Transfer of JSON Data over the network requires less bandwidth.

Limitations:

1. It is less secured
2. It provide support only for the protocols which can provide URI, mostly HTTP.

Note: Because of ligth weight,high performance, less bandwidth requirements, easy development, human understandable message format, this type of web services are most commonly used type of web services.

Differences between SOAP and REST:

SOAP | REST

1. XML Based Message Protocol
An Architecture Syle but not protocol

2. uses WSDL for communication between consumer and provider
uses xml/json to send and receive data

3. Invokes services by using RPC Method calls
Invokes services by simply URL Path

4. Does not return human readable result
Returns readable results like plain xml or JSON

5. These are heavy weight
These are light weight

6. These require more bandwidth
These require less bandwidth

7. Can provide support for multiple protocols like HTTP,SMTP,FTP etc
Can provide support only for protocols that provide URI, mostly HTTP

8. Performance is Low
Perforamnce is High

9. More Secured
Less Secured

Note: Most of the Google web services are SOAP Based.
Yahoo -->RESTful
eBay and Amazon using both SOAP and Restful