REST API

18/3/19

What is an API ?

API stands for application programming Interface.

An API is an intermediate software which allows two applications to Communicate each other.

Ex1: If we click on any application in our mobile phone then first the application will check for the internet and then it will go to corresponding server with request from the user.

→ The server takes data from the application and it will perform operations as per the request and then sends the response to the application which is on our mobile screen and it will display to the user in a specified manner.

Ex2: If we go to any restaurant then we sit at the table.

→ The kitchen room is little bit far away from us

→ A waiter will come to us and takes order and goes to kitchen room and bring food items for us.

→ So he acts as an API between our table and kitchen

→ Here he can takes the orders by using pen & paper or by using tab.

Scanned by CamScanner

→ If he uses pen & papers then he doesn't required Internet but he should go to kitchen room to inform our orders to the chef, so he is called normal API

→ If he uses tab to take orders then he required Internet so he need not go to kitchen room because when ever the submit our orders, it will go to chef through Internet, so here he is called web API.

What is a REST ?

REST stands for Representational State Transfer.

→ It is a architectural style which allows systems to take wide range of data and it gives the data to the client in the required format.

→ REST is a framework & Django framework that means if we want to use REST we should have Django framework

Django REST Framework (DRF)

→ Django Rest framework is a powerful and flexible toolkit that makes it easy to build web APIs

→ It provides class based generic API views, serializers viewsets. - - - -

Scanned by CamScanner

what is RESTful Web services

- An API provides an interface for developers to interact with applications database
- Instead of giving full access to database, an API sets set of rules and permissions and endpoints of various functionality like login, logout, reading a list of blogs, individual blog details, ----
- The traditional way to Construct a webAPI is via REST, it is a well-established architecture for how websites Can Communicate with each other.
- Since, Computers Communicates via through the web, this means by using http protocol which supports a no of Common methods Such as GET, PUT, POST and DELETE.
- If any request Comes then django handles based on the if statements, but REST API handle based on handler methods

Django (Function based views):

```
def display(request):
    if request.method="PUT"
    elif request.method="POST"
    elif request.method="Delete"
    else:
```

REST API (class based viewp)

```
class display():
    def PUT()
    def GET()
    def POST()
    def DELETE()
```

Scanned by CamScanner

→ A Restfull webservice is also called Restfull web API.

→ It is a webservice implemented by using http methods and principles of REST. Here we use http methods like GET and POST

20/3/19

→ It is a collection of resources with 4 defined aspects.

1. The base URI for the webservices. such as http://example.com/resources
2. The internet media such type of the data supported by the webservices this is often JSON, XML, or YAML but can be any other valid Internet media player.
3. The set of operations supported by the webservice using HTTP methods (eg. POST, GET, PUT or DELETE)
4. The API must be hypertext driven.

Note URI → Universal Resource Identifier
URL → Universal Resource Locator
JSON → JavaScript Object Notation
XML → Extended Markup language
YAML → Yet Another Markup language

Constraints of Restfull webservices

1. client - server
2. Stateless
3. Cache
4. uniform Interface
5. Layered system.
6. Code on Demmand (optional)

Scanned by CamScanner

Client - Server :

21/3/19

→ The client-server constraints works on the concept that the client and the server should be seperate from each other and allowed them to develop individually and independently.

→ In other words, we should be able to make changes to our mobile application without impacting either the data structure or database design on the server.

→ At the same time we should be able to modify the database or we should be able to make changes to our server without impacting the mobile application

→ It allows each application grow and scale independently of other and also allows our organization to grow quickly and efficiently

② Stateless

→ Http makes all client server interactions stateless.

→ Server will not store anything about latest Http request which is made by client.

→ It will treat each request as a new request

→ Server will not track any sessions and also history of the client. that way the server will add cookie to the response.

Scanned by CamScanner

③ Cache :

→ A stateless API can increase request overhead by handling large loads of incoming & outgoing calls.

→ A REST API should be designed to store the cache table data.

→ we can create cache table by using "Create Cache table" Command.

→ Cache table or cache memory will store the dynamic response from the server and if we make same request then the request will go to cache memory to get the stored response.

④ Uniform Interface :

Uniform interface allows the client to talk the server in a single language independent of architectural backend & either.

→ This interface should provide an unchanging, standardized means of communicating b/w the client and server, such as using http with URI resources, using CURD (Create, Update, Read, Delete) and using JSON

⑤ Layered

As the name implies, a layer system is a system comprised of layers, are each layer has a specific functionality and responsibility.

→ If we click of MVT framework, each layer has its own responsibilities.

Scanned by CamScanner

→ The models focusing on data that how it should be formed.

→ The Templates focusing on presentation

→ The views focusing on incomming actions.

→ Each layer is a seperate ~~but~~ but also interacts with eachother.

⑥ Code on demand

→ It is an optional, most of the time the data should be sending as resourses in the form of xml/JSON

Installing django rest framework (DRF)

To install django rest framework, we should install Python and django

→ If we want to install django rest framework then we go to Command prompt and run following Command

pip install djangorestframework

→ Generally we get documentation about Rest framework "www.restframework.com"

Note:→ whenever we create a project and application, then app name will configure under installed apps in settings.py file

→ we should add 'rest_framework' under installed apps manually for every project

Scanned by CamScanner

create a small application without using RestAPI (By using JSon Response) method.

22/3/19

steps : 1. foldername = API folder

2. project name = empapipro

3. app name = empapiapp

4. db name = empapidb

5. Configure database in settings.py file

6 Go to models.py file and create model.

```
from django.db import models

class Emp(models.Model):
    ename : models.CharField(max_length=50)
    sal : models.IntegerField()
    email : models.EmailField(max_length=50)

    def __init__(self):
        return self.ename
```

7. Go to admin file

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

class AdminEmp=(admin.ModelAdmin):
    list_display=['ename','sal','email']

admin.site.register(Emp,AdminEmp)
```

Scanned by CamScanner

8. Go to views file

```
from django.shortcuts import render
from django.http.response import HttpResponse
from .models import Emp

def emplist(request):
    data = Emp.object.all()
    s = {'result': list(data.values('ename', 'sal', 'email'))}
    return JsonResponse(s)
```

9. Go to urls.py file

```
from django.conf.urls import url
from django.contrib import admin
from empapiapp import views
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url(r'^emp/', views.emplist),
]
```

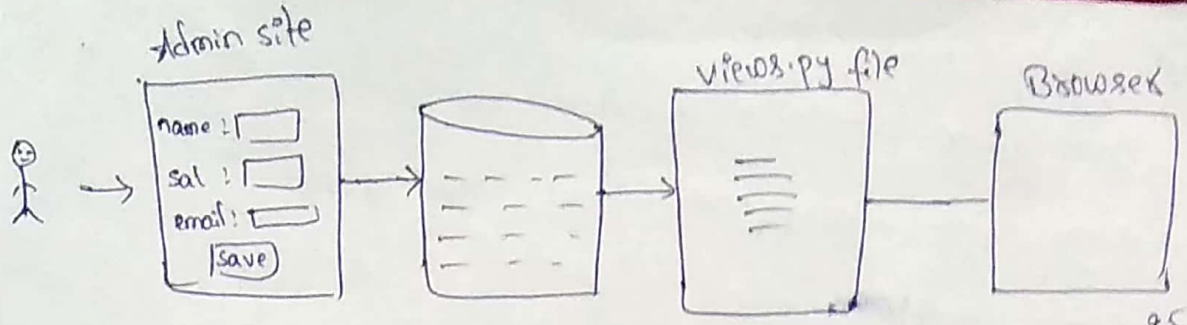
10. run makemigrations, migrate, create superuser and runserver commands

11. Go to adminsite and add some employee data

12. open new tab with same url and add '/emp' and click enter. it displays Json format data

{'results': [{"ename": "kesav", "sal": 10000, "email": "kesave@gmail.com"},
{"ename": "sai", "sal": 20000, "email": "sai@gmail.com"}]

Scanned by CamScanner



25/3/19

Note: whenever we send the request from the browser as http request object, the server application will send back http response object in the form of html response.

→ All ways the browser displays html format data only

→ Browser can easily understand html response. so that end user can easily read the data

→ If django application wants to communicate with any other application then internally we use JSON language, because it is universal language.

Types of webservices :

There are two types of web services.

1. SOAP based webservices
2. RESTful webservices

1). SOAP based

It stands for Simple Object Access protocal.

→ SOAP is always use XML based data

→ The biggest problem of XML is heavy weight.

Syntax :

```
<root>
<child1> sai <//child1>
<child2> Nani <//child2>
<//root>
```

Scanned by CamScanner

→ heavy bond width needed for XML SOAP based webservices for example, if any data is transfered from provider application to ~~transfe~~ consumer application heavy bond width data is required.

→ Passing the XML data will take more time, so it is not efficient

Note: passing means reading data from XML object to another object

→ Low performance

→ Implementing SOAP based API's will be more complex. because of this common problems, we are not using

* SOAP based webservices also some advantages

1. SOAP supports multiple types of protocols like, HTTP, SMTP, FTP, ...
2. SOAP uses WSDL
   If we want to send data from one application to another application then we need special language called WSDL
   WSDL stands for Web Service Description Language.
3. Due to this WSDL, data has more security while transfering between two application over the network.
4. No limit on size of the data

(2) RESTful webservices

→ REST stands for Representational State Transfer, when each uri is representation of an object.

→ By using http protocols or verbs, we can perform operations like GET, POST, PUT and DELETE.

Scanned by CamScanner

Advantages

1. The restful webservises are using JSON type based data.
2. JSON is a light weight Component
3. when transfering of JSON data ~~towards~~ over the network it will require less bond width.
4. NO WSDL language required that means we can expose directly our JSON data over the networks by using urls.
5. High performance.
6. Easy development.
7. It is human understandable message format like a plain-text. (it looks like python dictionary)

Disadvantages

1. less security to the data
2. It supports only HTTP protocols.
3. limited amount of data is transfering b/w applications

SOAP	REST
1. XML Based data	1. JSON based data
2. Heavy weight	2. less weight
3. More bandwidth required	3. less bandwidth required.
4. low performance	4. High performance
5. More complex to develop	5. Easy to develop
6. More secure	6. less secure
7. Uses WSDL	7. doesn't use WSDL
8. It supports more protocols (HTTP, SMTP, FTP, ---)	8. It supports only HTTP protocol
9. unlimited data can be transfered.	9. limited data can be transfered

Scanned by CamScanner

Class Based views

26/3/19

We have two types of views

1. Function based views
2. class based views

→ we can also write our API views by using class based views along with function based views

→ This powerful feature allows us to reuse the common functionality and also it helps us to keep our code "DRY" (DRY - Don't Repeat yourself)

1. The organization of code related to specific http methods like GET, PUT, POST, ... , can be implemented by using seperate methods instead of conditional branching.
2. Object oriented techniques such as inheritence can be used here
3. A class based views allows us to respond to different http request methods with different class methods instead of conditionally branching.
4. Django Rest framework provides an API view class.
5. API view classes are different from regular views classes in the following ways.

   i) Request which is passed to the handler methods will be REST framework's request, not Django's HTTP request.

   ii) Handlers methods may return REST framework's response not Django http response.

Scanned by CamScanner

Normal view

Django HTTP request

```
def home(request):
    if request.method == 'POST':
        ==
    elif r.m == 'GET':
        ==
    elif r.m == 'PUT':
        ==
    else:
        ==
```

Django HTTP response

API view

Rest framework's request

```
class home(APIview):
    def GET():
        ==
    def PUT():
        ==
    def POST();
        ==
    def DELETE();
        ==
```

handles methods

REST Framework's response

Create a project by using API views.

steps 1. foldername : API11am

2. Projectname : classbasedviewpro

3. app name : cbvapp

4. Go to settings.py file and add rest_framework under Installed apps.

```
Installed_APPS = (
    - - - - -
    - - - - -
    - - - - -
    'rest_framework',
    'cbvapp',
)
```

Scanned by CamScanner

5. Go to views file

```
from django.shortcuts import render
from rest_framework.views import APIView
from rest_framework.response import Response
class InstituteClass(APIView):
    def get(self, request):
        database = [
            {
                'oracle': 1000,
                'mysql': 1500,
                'sql Server': 1000,
            }
        ]
        language = [
            {
                'c': 1000,
                'Python': 2000,
                'Django': 2000,
            }
        ]
        frameworks = [
            {
                'Django': 2000,
                'REST API': 2000,
                'Flask': 2000,
            }
        ]
```

Scanned by CamScanner

```
return Response(
        {
            'databases' : database ,
            'languages' : language ,
            'frameworks' : frameworks .
        }
    )
```

6. Go to <u>urls.py</u> file

```
from cbvapp import views
urlpatterns = [
    url(r'^admin/', admin.site.urls),
    url (r'^institute/', views.InstituteClasses.as_view()),
]
```

7. run server.

8. Add /institute to the existing url press enter.

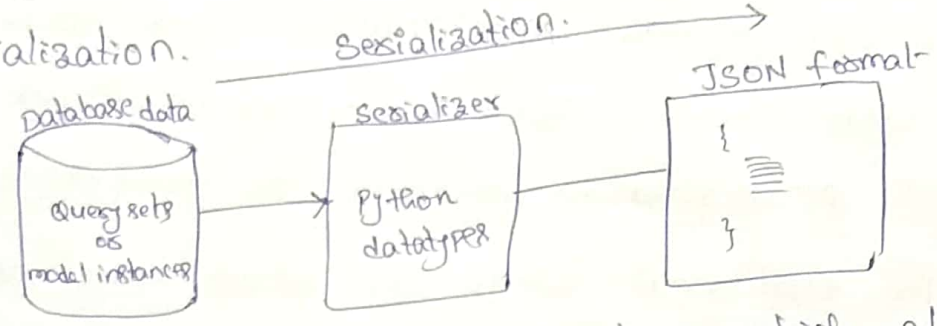
27/3/19

```
{
    "databases" : [
        {
            "oracle": 1000
            "mysql": 2000
            "sql-server": 2000
        }
    ]
    "languages" : [
        {
            =
        }
    ]
    "frameworks" : [
        {
            =
        }
    ]
}
```

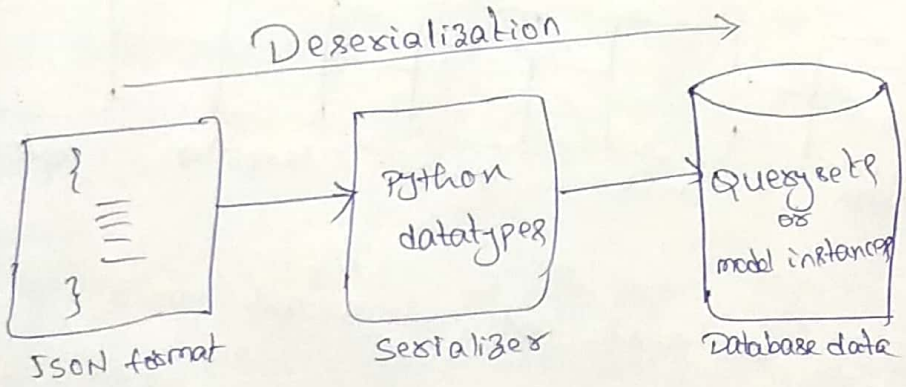
Scanned by CamScanner

Serializer

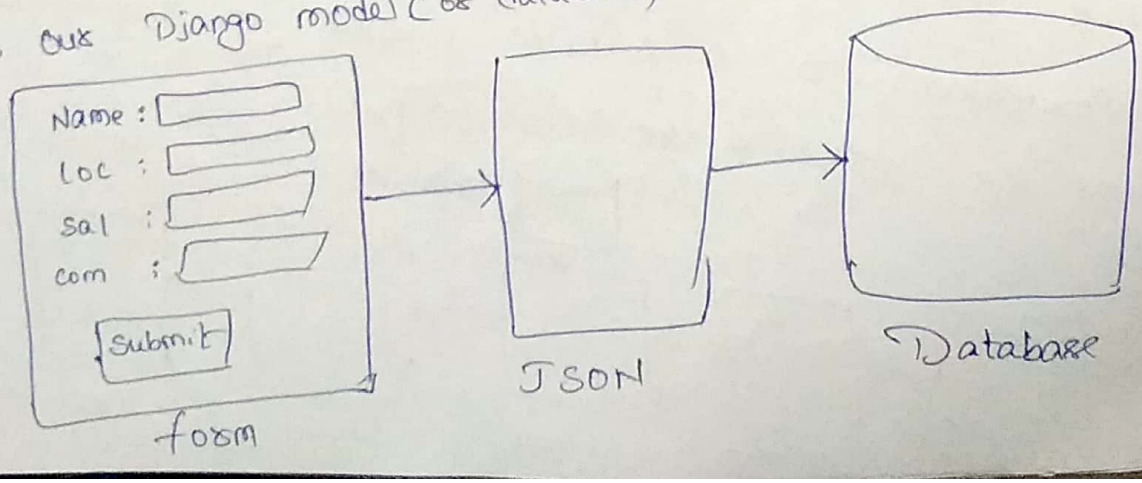
A serializers are used to convert the query sets or model instances into native python datatypes and then it can be easily render into JSON, this process is called serialization.



→ Serializers also provide deserialization which allows JSON data into native python datatypes and then it can be converted into querysets or model instances.



→ API accepts user input for example, submitting a form which is translated from html into JSON then again converted into our Django model (or database)



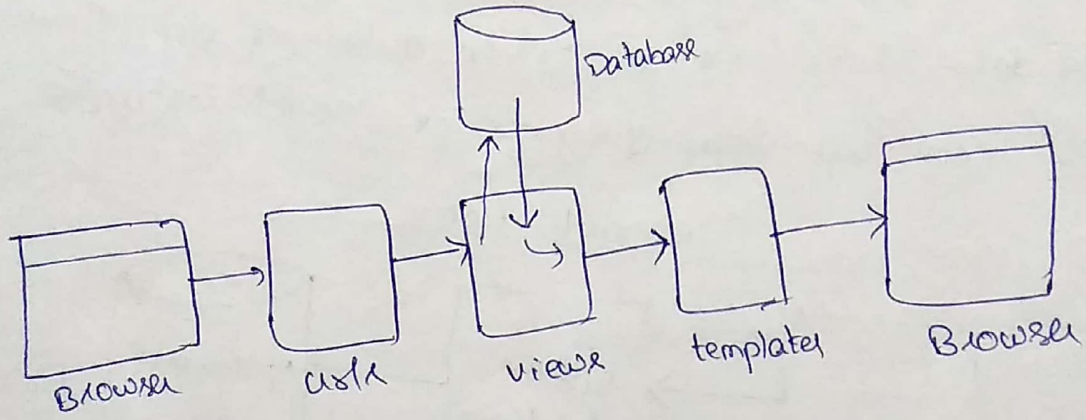
Scanned by CamScanner

The serializers in rest-framework works very - Similarly to Djangos form and model form classes.

→ we provide Serializer class which gives us a powerful a generic way to control the output of our responses.

→ we also provide ModelSerializer class which provides a useful shortcut for creating the serializers that deal with Model ~~Serializer~~ instances or querysets.

→ The traditional django application needs url, views, template to translate information from database to webpage



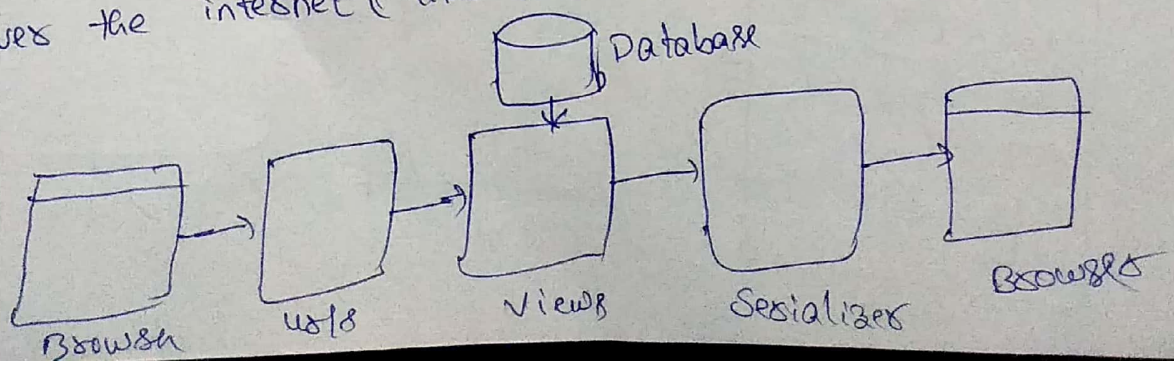
→ In DRF we need url, views, and serializer

→ The url controls the access to the API end points

→ view controls the logic of data

→ The Serializer performs the magic of converting our information into a suitable format for transmission over the internet (like JSON format)

→



Scanned by CamScanner

→ A normal webpage requires html, css, bootstrap and Javascript (usually) But our API is only sending data in the JSON format so we don't need html, CSS, bootstrap and Javascript, ---

→ we just need only data

Steps

1. folder name : apiNam
2. aproject name : emppro
3. appname : empapp
4. db name : rapi
5. Go to settings file and Configure database
6. Go to models.py file and create model

```
from django.db import models

class Emp (models.Model):
    ename = models.CharField (max_length = 20)
    email = models.EmailField (max_length = 20)
    def __str__(self):
        return self.ename.
```

7. Go to admin file

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

class AdminEmp(admin.ModelAdmin)
    list_display = ['ename', 'email']
admin.site.register(Emp, AdminEmp)
```

Scanned by CamScanner

* Create serializers.py file in the application

```
from rest_framework import Serializers

class EmpSerializer(serializers.Serializer):
    ename = serializers.CharField(max_length=20)
    email = serializers.EmailField(max_length=20)
```

* Go to views.py file

```
from django.shortcuts import render
from rest_framework.response import Response
from rest_framework.views import APIView
from .models import Emp
from .serializers import EmpSerializer

class EmpView(APIView):
    def get(self, request):
        emp = Emp.objects.get_all()
        serializer = EmpSerializer(emp, many=True)
        return Response(serializer.data)

    def post(self, request):
        serializer = EmpSerializer(data=request.data)
        if serializer.is_valid():
            ename = serializer.data.get('ename')
            email = serializer.data.get('email')
            msg = "Hello { }, your mail Id is { }".format(ename, email)
```

Scanned by CamScanner

```
        emp = Emp.objects.create(ename = ename, email = email)
        return Response({'message': msg})
    else:
        return Response(serializer.errors, status = status.HTTP_400_
                                                BAD_REQUEST)
```

10. Go to urls file

```
from django.conf.urls import url
from empapp import views
urlpatterns = [
    url(r'^admin/', admin.site.urls)
    url(r'^emp/', views.Empview.as_view()),
]
```

1/4/19

11. run makemigrations, migrate, superuser and runserver commands.

login to admin site ~~and~~

click on add form of emp model and enter data

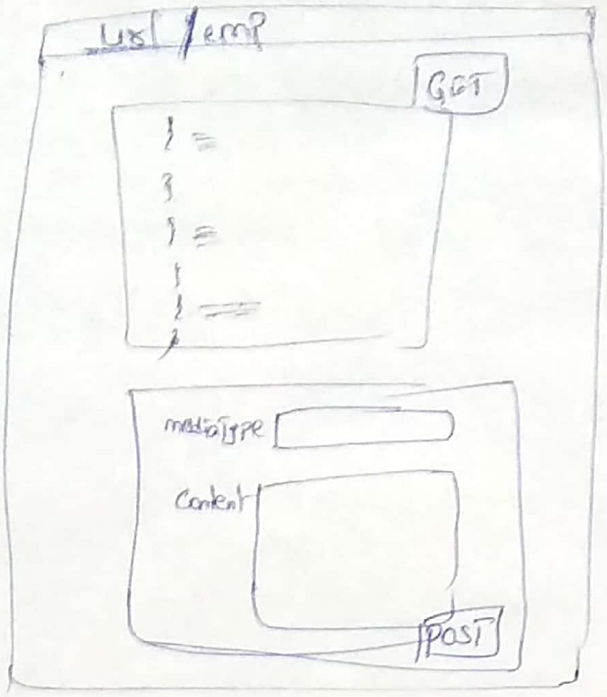
ename : sai	Nani	satya
email = sai@gmail.com	nani@gmail.com	satya@gmail.com
Save & another	Save & another	save

12. open new tab and copy and paste same IP address

13. add/emp to the url and click on enter.

~~click~~ o It displays the database data in ~~dat~~ JSON format, this process is called Serialization.

Scanned by CamScanner



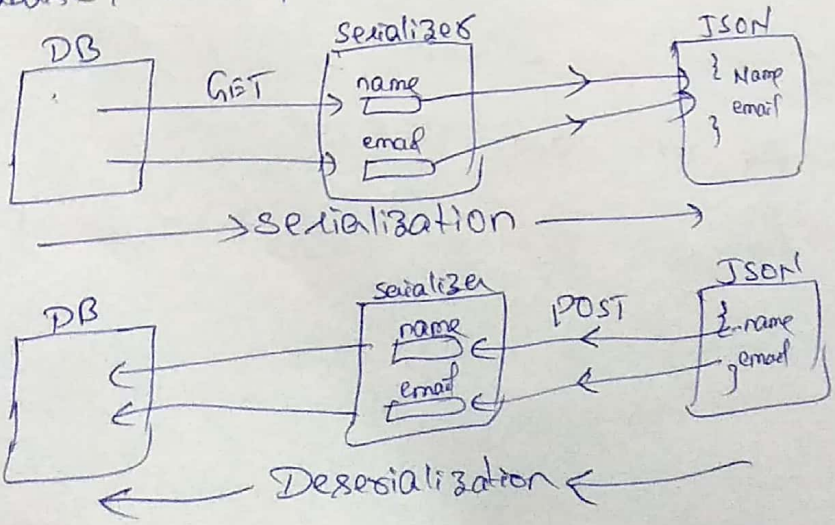
Here Three resources are displayed, Copy any one resource and paste in the Content box and modify the values like below.

Content :

```
{
  'name' : "komal"
  'email" : "komal@gmail.com
```

POST

→ If we click on POST button then JSON format resource will convert as a queryset and stores in the database, this process is called de-serialization.



Scanned by CamScanner

→ In Rest API, we perform the CURD operations in 5 ways.

→ we can divide the CURD operations as key based and Non key based operations.

operations	key based	Non key based
Create	Update	Create
update	Retrieve-One	Retrieve-all
Retrieve-all	Delete	
Retrieve-one		
Delete		

2/4/19

we create two class based views one is for key based operations and the other is for non-key based operations.

MIXINS

The mixins provide the actions that are used to provide the basic behaviour.

→ The mixin classes provide action methods rather than handler methods such as GET, POST, .

→ These mixin classes allows more flexible composition & behaviour

→ mixin classes can be imported from rest_framework.mixins.

→ The mixin classes are

List Model mixin
Create Model Mixin
Retrieve Model Mixin
Update Model Mixin
Destroy Model Mixin.

Scanned by CamScanner

→ ListModelMixin is used to get all the data

→ It provides .list() action method

CreateModelMix is used to insert the date

→ It provide .~~setrev~~ .create() action method

→ RetreiveModelMixin is used to retrieve a specific data

It provide .retrieve() action method

→ UpdateModelMixin is used to update the specific data

It provide .update() action method.

→ DestroyModelMixin is used to delete the data.

It provides .destroy() action method

<u>Steps</u> 1. foldersname = apiilam

2. Project name : mixinpro

3. app name : mixinsapp

4. db name : mixinr_db

5. settings.py file under installed apps add rest_framework and in databases Configure the database.

Generic (Handlemethods)		Mixins (actionmethod)		
get()	—	.list()	→ ~~List~~ ListModelMixin	→ getting all
post()	—	.create()	— CreateModelMixin	→ Inserting
get()	—	.retrieve()	— RetrieveModelMixin	→ getting one
put()	—	.update()	→ UpdateModelMixin	→ update one
delete()	→	.destroy()	→ DestroyModelMixin	→ Delete one

Scanned by CamScanner

6. Create a model in models.py file.

```
from django.db import model
class product(models.Model):
    product_Id = models.IntegerField(primary_key=True)
    product_name = models.CharField(max_length=50)
    product_cost = models.FloatField( )
    product_color = models.CharField(max_length=50)
```

7. Create serializers.py file

```
from rest_framework import serializers
from .models import product
class productSerializer(serializer.ModelSerializer):
    class Meta:
        model = product
        fields = ('__all__')
```

8. Go to admin file

```
from django.contrib import admin
from .models import product
class Adminproduct(admin.ModelAdmin):
    list_display = ('product_Id',
                    'product_name',
                    'product_cost',
                    'product_color')
admin.site.register(product, Adminproduct)
```

Scanned by CamScanner

9. Go to views file

```
from django.shortcuts import render
from rest_framework import mixins
from rest_framework import generics
from .serializers import ProductSerializer
from .models import product.

class ProductList(mixins.ListModelMixin,
                  mixins.CreateModelMixin,
                  generics.GenericAPIView):

    queryset = product.objects.all()
    serializer_class = ProductSerializer.

    def get (self, request, *args, **kwargs):
        return self.list(request, *args, **kwargs)

    def post (self, request, *args, **kwargs):
        return self.~~list~~create (request, *args, **kwargs)

class ProductDetail (mixins.UpdateModelMixin,
                     mixins.RetrieveModelMixin,
                     Mixins.DestroyModelMixin,
                     generics.GenericAPIView):

    queryset = product.objects.all()
    serializer_class = ProductSerializer

    def get (self, request, *args, **kwargs):
        return self.retrieve (request, *args, **kwargs)

    def put (self, request, *args, **kwargs):
        return self.update (request, *args, **kwargs)
```

Scanned by CamScanner

```
def delete (self, request, *args, **kwargs):
    return self.destroy (request, *args, **kwargs)
```

10. Create urls.py file in the application.

```
from django.conf.urls import url
from . import views

urlpatterns = [
    url(r'^product/$', views.ProductList.as_view()),
    url(r'^product/(?P<pk>[0-9]+)', views.ProductDetail.as_view()),
]
```

11. Go to project level urls file

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

urlpatterns = [
    url('admin/', admin.site.urls),
    url(r'^api/', include('mixins_APP.urls'))
```

12 run makemigrations, migrate and create superuser and runserver commands

→ login to adminsite and enter data like below.

Add product +

product ID: 101	102
product name: tap	mouse
product cost: 50000	1000
product color: black	red
save & another	save

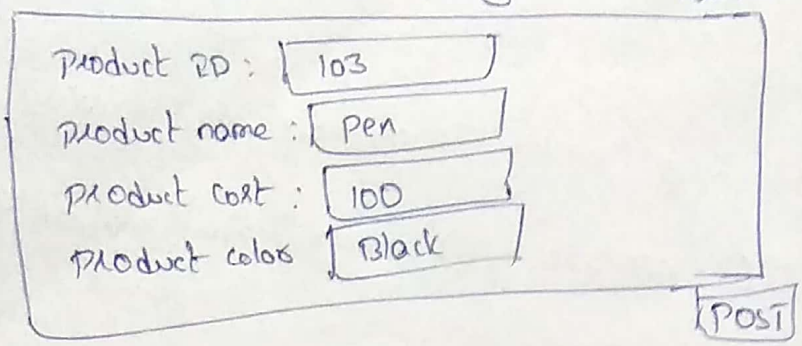
Scanned by CamScanner

→ open new tab with same url and add /api/product click enter

Then it converts all querysets from database into JSON format. (It is getting all operation)

→ In the same page we can find HTML form to insert new data from JSON to queryset.

RawData | HTML form



If we click on POST button this JSON Data will go to serializer, the serializer will convert from JSON to querysets and then queryset will save in the database. (It is insert operation)

Note: The above two operations are called non-key operations

Because we didn't use any specific id number in the url to perform above two operations. 127.0.0.1:8000/api/product

Here didn't use any specific id so those two operations are called non-key based operations.

→ If we want to get any specific queryset in the JSON then we have to specify the corresponding id number in url

127.0.0.1:8000/api/product/102

If we click enter then it display the corresponding product and also display html form with product data.

Scanned by CamScanner

→ If we want to modify the data of 102 product then modify data in html form and click PUT button.

→ If we want to delete any product then specify the product id in the url like 127.0.0.1:8000/api/product/103, if we click enter then displays corresponding product data and also displays html form to update and also there is Delete button.

→ If we click on delete button then the product will be deleted from the database.

4/4/19

Generic views

Generic views also called Concrete view classes

→ Generic views provides lot of default class names to perform any required crud operations.

→ we use the generic views as a parent models to our normal models.

1. CreateAPIview
2. ListAPIview
3. UpdateAPIview
4. DestroyAPIview
5. ListCreateAPIview
6. RetrieveAPI view
7. RetrieveUpdateAPIview
8. RetrieveDestroyAPIview
9. RetrieveUpdateDestroyAPIview

Scanned by CamScanner

① CreateAPIView class is used for create-only endpoints
→ It provides POST handler method

② ListAPIView is used for read-only endpoints.
→ It provides GET handler method.

③ RetrieveAPIView is used for read-only endpoints which represents a single model instance.
→ It provides GET handler method

④ DestroyAPIView is used for delete-only endpoints for a single model instance.
→ It provides DELETE handler method.

⑤ UpdateAPIView is used for update-only endpoints for a single model instance.
→ It provides PUT handler method.

⑥ ListCreateAPIView is used for read-only and create-only endpoints.
→ It provides GET & POST handler methods

⑦ RetrieveUpdateAPIView is used for read-only and update-only endpoints which represents single model instances
→ It provides GET and PUT handler methods.

⑧ RetrieveDestroyAPIView is used for read-only & delete-only endpoints which represents single model instances.
→ It provides GET and DELETE handler methods.

Scanned by CamScanner

⑨. RetrieveupdateDestroyAPIView is used for read-only, update-only and Delete-only endpoints which represents single model instances.

→ It provides GET, PUT and DELETE handler methods.

Step 1. folder name = API_Iam

2. project name = genericviewsPro

3. app name = genericviewsapp

4. DB name = generic_db;

5. Go to settings file and configure db and specify rest_framework under installed_apps

6. Go to models.py file and create Emp model eid, ename, esal created and modified fields

```
created = models.DateTimeField(auto_now_add = True)
modified = models.DateTimeField(auto_now = True)
```

⑦ Create serializers.py file, create Emp serializer.

```
from ———
from Generic_views_APP.models import Emp

class Empserializer(Serializers.ModelSerializer):
    class Meta:
        model = Emp
        fields = ['eid','ename','esal']
```

⑧ Go to ~~view~~ admin.py file

```
from ———
from .models import Emp
class AdminEmp(admin.ModelAdmin):
    List_display = ['eid','ename','esal','created','modified']

admin.site.register(Emp, AdminEmp)
```

Scanned by CamScanner

(9) Go to views.py file.

```
from rest_framework import generics
from Generic_views-App.serializers import EmpSerializer
from Generic_views_App.models import Emp

class GenericView_List(generics.ListCreateAPIView):
    queryset = Emp.objects.all()
    Serializer_class = EmpSerializer

class GenericView_Details(generics.RetrieveUpdateDestroyAPIView):
    queryset = Emp.objects.all()
    serializer_class = EmpSerializer.
```

(10) Create urls.py file in the application. like previous mixin example

(11) Go to project level urls.py file and configure like previous example

(12) run makemigrations, migrate, createsuperuser and runserver commands

(13) Login to admin site and give the data.

(14) open new tab with same url and perform crud operations like mixins examples

Viewsets:

viewsets allow us to write set of views in a single class

i.e we can write the views of 5 crud operations under a single class

→ REST API provides → (1) viewsets.ViewSet
→ (2) viewsets.ModelViewSet

Scanned by CamScanner

① Viewsets.ViewSet :

viewset is the default class in viewsets module.

→ In viewset class we have to write the manual coding for all views

→ we have to use viewset class as the base class for normal class.

steps 1. folder name : apillam

2. project name : viewsetspro

3. app name : viewsetsapp

4. db name : viewset_db

5 Configure db in settings file and specify rest_framework under the installed_apps.

6. Go to models.py file.

```
from django.db import models

class Feedback(models.Model):
    user_id = models.IF()
    name = models.CF(m.l=200, help_text='Name of the sender')
    email = models.EF(m.l=200, unique=True)
    subject = models.CF(m.l=200)
    message = models.TextField()
    created = models.DateTimeField(auto_now_add=True)
    modified = models.DateTimeField(auto_now=True)
    def __str__(self):
        return self.name + "_" + self.email
```

Eighteen

Scanned by CamScanner

⑦ Go to admin file

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

class AdminFeedback(admin.ModelAdmin)
    list_display = ['user_id',
                    'name',
                    'email',
                    'subject',
                    'message',
                    'created',
                    'modified']

admin.site.register(Feedback, AdminFeedback)
```

⑧ Create Serializer

```
from rest_framework import serializers
from .models import Feedback

class FeedbackSerializer(serializers.ModelSerializer):
    class Meta:
        model = Feedback
        fields = ('user_id',
                  'name',
                  'email',
                  'subject',
                  'message',
                  'created',
                  'modified')
```

Scanned by CamScanner

⑨ Go to views

```
from django.shortcuts import render
from rest_framework.response import response
from rest_framework import status
from rest_framework import viewsets
from .models import Feedback
from .serializers import FeedbackSerializer

class viewset_feedback(viewsets.ViewSet):
    def list(self, request):
        queryset = Feedback.objects.all()
        serializer = FeedbackSerializer(queryset, many=True)
        return Response(serializer.data)

    def create(self, request):
        serializer = FeedbackSerializer(data= request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
                            status = status.HTTP_201_CREATED)

        else:
            return Response(serializer.errors,
                            status = status.HTTP_400_BAD_REQUEST)

    def retrieve(self, request, pk):
        queryset = Feedback.objects.get(pk=pk)
        serializer = FeedbackSerializer(queryset)
        return Response(serializer.data)
```

Scanned by CamScanner

```
def update( self, request, pk):
    queryset = Feedback.objects.get(pk=pk)
    serializer = FeedbackSerialize(queyset, data=request.data)

    if serializer.is_valid():
        Seridizer.save()
        return Response(serializer.data,
                        status = status.HTTP_201_CREATED)

    else:
        return Response( Serializer.errors,
                         status = status.HTTP_400_BAD_REQUEST)

def destroy(self, request, pk):
    queryset = Feedback.objects.get(pk=pk)

    queryset.delete()

    return Response(status= status.HTTP_204_NO_CONTENT)
```

8/4/19

ROUTER

A router is used to generate an urls automatically as per the request method.

→ It also provides an endpoint url before convert into JSON format.

→ We use this endpoint url to provide a communation b/w two applications.

(10) viewset_App urls.py

```
from django.conf.urls import url
from django.conf.urls import include
from .import views
from rest_framework.routers import DefaultRouter
```

Scanned by CamScanner

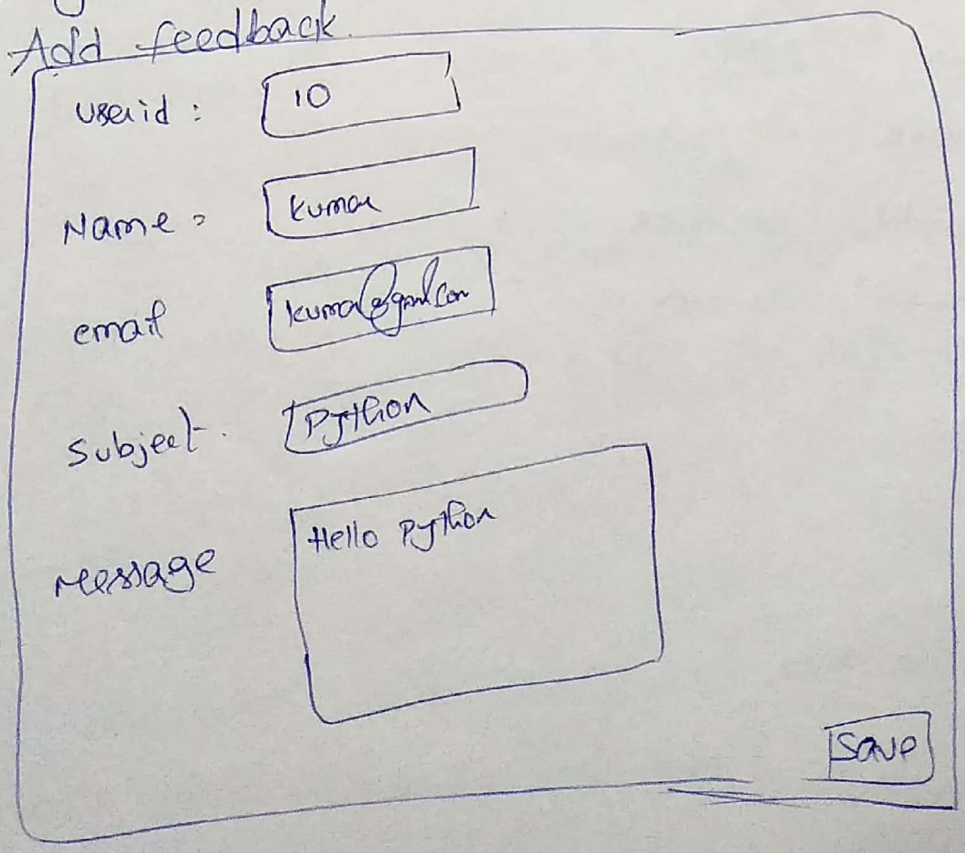
```
router = DefaultRouter()
router.register('viewset', views.ViewSet_feedback, base_name='viewset')

urlpatterns = [
    url(r'^', include(router.urls))
]
```

(11) Go to project level urls file

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

urlpatterns = [
    url(r'admin/', admin.site.urls)
    url(r'^feedback/', include('viewSet_App.urls'))
]
```

(12) run makemigrations, migrate, create superuser and runserver commands.

(13) login to admin site and give the data like below.



Scanned by CamScanner

→ open new tab with same url

→ Add /feedback to the existing url and click Enter.

127.0.0.1:2200/feedback/

```
{
"viewset": "http://127.0.0.1:2200/feedback/viewset/"
}
```

→ If we click on the endpoint url then the request will go to app level urls file then it converts data into JSON format and displays

127.0.0.1:2200/feedback/viewset/

```
[ {
    "usr_id": "10",
    "name": "kumar",
    "email": "kumar@gmail.com",
    "subject": "Python",
    "message": "Hello Python",
    "created": "2019-04-08 ——",
    "modified": "2019-04-08 ——"
  }
]
```

Scanned by CamScanner

ModelViewSets : (viewsets.ModelViewSet)

ModelViewSet provides shortcuts to write set of views in a single class where as in viewset class we write lot of manual coding to write set of views in a single class

steps:
1. foldername : apillam
2. pro name :
3. app name :
4. db name :
5. Configure db in settings file and specify rest_framework under installed apps
6. Go to models.py file create Emp model with empid, empname, email and salary.
7. Go to admin.py file and Configure as per the model.
8. Create serializers.py file and Create EmpSerializer(serializers.ModelSerializer)
9. Go to views.py file

```
from rest_framework.viewsets import ModelViewSet
from .serializers import EmpSerializer
from .models import Emp

class SubscriberViewSet (ModelViewSet):
    serializer_class = EmpSerializer
    queryset = Emp.objects.all()
```

Note: ModelViewSet class takes serializer_class and queryset variables as inputs and it will handle any kind of request methods like

Scanned by CamScanner

(10) Create app level urls and Configure routers like previous ex

```
router = DefaultRouter()
routers.register("subscribers", ModelViewSet)
urlpatterns = router.urls
urlpatterns = [
        url("r'", include(router.urls))
]
```

(11) Go to project level urls and Configure

(12) run makemigrations, migrate, create superuser and runserver Commands

(13) login to adminsite and enter the data

(14) open new tab with same url and add /subscribers and perform the operations like previous example.

Perform CURD operations by using Function based views

steps:

1. Foldername: API11AM, pro name = Function_based_project

3. APP: Function_based_APP, 4. db name: Function_based_db

5. Go to settings and Configure db and specify rest_framework in installed apps

(6.) Go to models.py file

```
from django.db import models.
class product (models.Model):
        product_id = models.IF( primary_key =True)
        product_name = models.CF ( ml = 20)
        product_price = models.DecimalField ( max_digits = 10, decimal_places=2)
        product_color = models.CF (m_l = 20)
```

Scanned by CamScanner

⑦ Go to admin.py file

```
from ———— admin
from .models import product
class AdminProduct(admin.ModelAdmin):
    list_display = [same fields in model------]
admin.site.register(product, AdminProduct)
```

⑧ In APP create Serializers.py file

```
from .rest_framework import Serializers
from .models import Product
class productSerializer(Serializers.ModelSerializer):
    class Meta:
        model = Product
        fields = ('product_id',
                  'product_name',
                  'product_price',
                  'product_color')
```

⑨ Go to views.py file

```
from rest_framework import status
from rest_framework.decorators import api_view
from rest_framework.response import Response
from .serializers import ProductSerializer
from .models import product
@api_view(['GET','POST'])
def productList(request):
    if request.method == 'GET':
        products = product.objects.all()
        Serializer = ProductSerializa(products, many=True)
        return Response(Serializer.data)
    elif request.method == 'POST':
        serializer = ProductSerializa(data = request.data)
        if Serializa.is_valid():
            Serializer.save()
            return Response(Serializa.data, status = status.HTTP_201_Created)
    return Response(Serializer.errors, status = status.HTTP_400_BAD_request)
```

Scanned by CamScanner

(a) api_view('GET', 'PUT', 'DELETE')

```
def ProductDetail (request, pk):
    try:
        products = Product.objects.get (pk = pk)
    except Product.DoesNotExist:
        return Response( status = status.HTTP_404_NOT_FOUND)
    if request.method == 'GET':
        serializer = ProductSerializer (Products)
        return Response (serializer.data)
    elif request.method == 'PUT':
        serializer = ProductSerializer (products, data = request.data)
        if serializer.is_valid():
            serializer.save()
            return Response (serializer.data)
        return Response (serializer.errors, status = status.HTTP_402_BAD_request)

    elif request.method = 'DELETE':
        products.delete()
        return Response (status = status.HTTP_206_NO_CONTENT)
```

(10) Create APP level urls.py file

```
from django.conf.urls import url

from .app import views

urlpatterns = [
        url (r'^product/$', views.productList),

        url (r'^product/(?P<pk>[0-9]+)', views.ProductDetail)

    ]
```

(11) Go to project level urls file

```
from django.contrib import admin

from django.conf.urls import url, include
```

Scanned by CamScanner

```
urlpatterns = [
    url (r'^admin/', admin.site.urls),
    url (r'^api/', include ('Function_Based_APP.urls'))

]
```

(12) Run makemigrations, migrate, createsuperuser, runserver Commands.

Login to the admin site & enter data on admin site.

open new tab with same url with adding like below

ip : post/api/product/ ↵ Enter

It will display all data in the JSON format

{
≡
}

POST ↵

→ Copy any one resource and paste in the Content box and change the values click on POST button, this JSON data is Converte and save in db.

→ If we want to perform GET, PUT, DELETE operations on specific resource then add that id number in url like below

ip : post/api/product/101 ↵ Enter

Then it will get product details which belongs to 101 product

→ If we want to modify or update any value of product_id 101 then copy the resource and paste in Content box and change the value and click on PUT button so the value will be updated in database also

→ If you want to delete product id = 101 then click on delete button so that Corresponding queryset also delete from database.

Scanned by CamScanner

Different ways of Converting Python dictionary to JSON.

16/4/19

1. By .format function
2. By dumps() function
3. By JsonResponse ()

Ex: 1. foldersname : API11AM
2. pro name : dumpspro
3. appname : dumpsapp
4. Go to views file

```
from django.shortcuts import render
from django.http.response import HttpResponse
```

By using format()

```
def empdata_view (request):
    emp_data = {
        'eno' : 100,
        'ename' : 'Ramu',
        'esal' : 10,000,
        'eaddr' : 'Hyd',
    }
    resp = 'Employee Number : { } ' \
           'employee name : { } ' \
           'Employee salary : { } ' \
           'Employee address : { } ' \
           .format (emp_data['eno'],
                    emp_data['ename'],
                    emp_data['esal'],
                    emp_data['eaddr'])
    return HttpResponse (resp)
```

Scanned by CamScanner

By using dump()

```
import json
def empdata_json_view(request):
    emp_data = {
        'eno' : 200
        'ename' : 'kiran'
        'esal' : 20000
        'eaddr' : 'Bangalore'
    }
    json_resp = json.dumps(emp_data)
    return HttpResponse(json_resp, content_type='application/json')
```

By using JsonResponse()

```
from django.http import JsonResponse
def empdata_json2_view(request):
    emp_data = {
        'eno' : 300,
        'ename' : 'kumar',
        'esal' : 30000,
        'eaddr' : 'Chennai',
    }
    return JsonResponse(~~json_resp~~ emp_data)
```

(5) Go to urls file

admin urls

```
from .appname import views
urlpatterns = [
    url('admin/', admin.site.urls),
    url(r'^empdata/', views.empdata_view),
    url(r'^jsondata/', views.empdata_json_view),
    url(r'^jsondata2/', views.empdata_json2_view),
]
```

(6) Runserver

Scanned by CamScanner

Q write a program to do CURD operations on database without REST API Concepts.

17/4/19

Serialization: Serialization is a Concept of Converting one form data into another form.

The process of Converting querysets or database instances into Python native datatype (dictionary) and then Converted into Json format is called serialization.

Deserialization: The process of Converting the Json format into Python native datatype (dictionary) and then Converted into querysets or database instances is called deserialization.

Q. How many ways we can perform serialization.

Three ways are available,

1. By using Python inbuilt module "json."

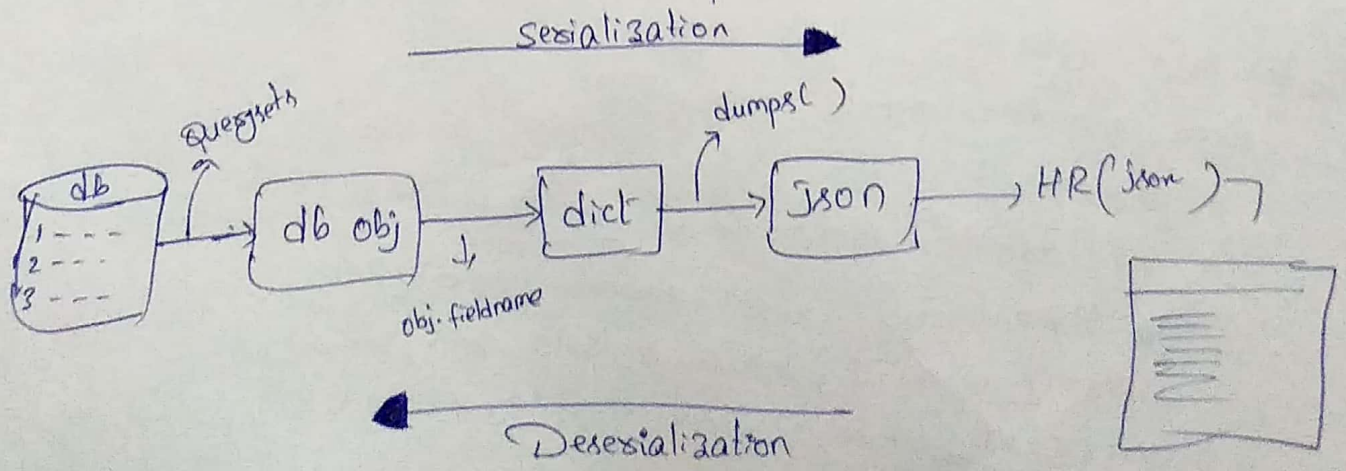
   json.dumps(data)

2. By using djangos serialize() function.

   from django.core.serializers import serialize

3. By using rest_framework Serializers module.

   from rest_framework import serializers.



Scanned by CamScanner

1. Projectname : Serialize_project
2. Appname : serialize_App
3. dbname : serialize_db
4. open settings.py file add our app name under installed apps
   Configure the database

Note: Comment CsrfViewMiddleware class inside middleware section.

5. open model.py file

```
from django.db import models
class Employee(models.Model):
    eno = models.IFC()
    ename = models.CF(m_l = 100)
    esal =
    eaddr =
```

6. open views.py file

```
from serialize_App.models import Employee
from django.views.generic import View
from django.http.response import HttpResponse.
import json
from django.core.serializers import serialize

# To get single record (Id based)
class EmployeeDetailView(View):
    def get(self, request, id):
        try:
            emp = Employee.objects.get(id = id)
        Except Employee.DoesNotExist:
            json_data = json.dumps({'msg': 'Request resource Not available'})
            return HttpResponse(json_data, Content_type='application/json')
```

Scanned by CamScanner

```
else:
    emp_data = {
        'eno' : emp.eno,
        'ename' : emp.ename,
        'esal' : emp.esal,
        'eaddr' : emp.eaddr
    }
    json_data = json.dumps(emp_data)
    return HttpResponse(json_data, Content_type='application/json')
```

```
json_data = Serialize('json', [emp], 'fields' = ['eno', 'ename')
p_data = json.loads(json_data)
```

what is the Concept of django Serialization.

- Serializers is a django inbuilt module
- Serializers module Contain one method ie Serialize() function.
- By using serialize function we can Convert queryset object into json type very easy.

for ex: `json_data = serializers.serialize('json', queryset)`

Here we are not write a big code but Converting queryset records into json type, This single line will give everything. thats why serializers Concept got more popularity in django.

Note: serialize() taking two mandatory parameters
1st parameter is a type of data to Convert, and 2nd parameter is a database object reference variable.

```
serialize('json', queryset)
```

Scanned by CamScanner

→ Serialize() also accepted some optional parameters like fields

Q: How do you customizing the required fields to display on browser

Ans: By using fields attribute, we can customize required fields

serialize('json', qs, fields=['eno','ename'])

Output:

{'model': 'serialize_App.Emp', 'pk': 1, 'fields': {'eno': 100, 'ename': 'kumar', 'esal': 10000, 'eaddr': 'Hyd'}}

Here, lot of meta data came as output, I want only required employee fields data, I don't want to ~~the~~ display all meta data.

Q: How do you overcome this problem?

Step 1: After getting the json_data by using serialize() function then convert json_data into dictionary type data by using loads() method of json module.

```
Python_data = json.loads(json_data)
```

It contains list of dictionary objects

Step 2: we will write for loop for getting exact dictionary data

```
for obj in Python_data:
    emp_data = obj["fields"]
```

Here, obj → contains meta data like (model, pk, fields)

obj["fields"] → will get required data of emp. No meta data comes here

emp_data → here each employee data is coming each time. so we need to add these data to one empty list

final_list = []

Scanned by CamScanner

step3: Assign each emp_data into this empty list by using append()

```
final_list.append(emp_data)
```

→ finally our list contains all required employee fields data only.
No meta data here. It is a dictionary inside list.

step4: Now convert this python dictionary data into json data type by using dumps() method.

```
json_data = json.dumps(final_list)
```

step5: Send this json_data into HttpResponse() method to display on the browser

```
json_data = serialize('json', [emp], fields=['eno', 'ename'])

python_data = json.loads(json_data)

final_list = [ ]

for obj in python_data:

        emp_data = obj['fields']

        final_list.append(emp_data)

json_data = json.dumps(emp_list)

return HttpResponse(json_data, content_type='application/json')
```

Scanned by CamScanner

Scanned by CamScanner

Updating the existing record

22/4/19

```
def get_objects_by_id(self,id):
    try:
        emp = Employee.objects.get(id=id)
    except Employee.DoesNotExist:
        emp = None
    return emp
```

this above method is common for get single record, update record and delete record by using id based.

```
def put(self, request, id):
    emp = self.get_objects_by_id(id)
    if emp is None:
        json_data = json.dumps({'msg':'The requested resource not available'})
        return HttpResponse(json_data, content_type='application/json', status=404)
    else:
        data = request.body
        valid_json = is_json(data)
        if not valid_json:
            json_data = json.dumps({'msg':'Please send valid json data'})
            return HttpResponse(json_data, content_type='application/json', status=400)
        else:
            provided_data = json.loads(data)
            original_data = {
                'eno': emp.eno,
                'ename': emp.ename,
```

Scanned by CamScanner

```
        'esal' : emp.esal,

        'eaddr' : emp.eaddr
    }
    original_data.update(provided_data)
    form = EmployeeForm(original_data, instance=emp)
```

it is updating the existing record.

```
    if form.is_valid():
        form.save(commit=True)
        json_data = json.dumps({'msg':'Data updated successfully'})
        return HttpResponse(json_data, content_type='application/json')

    if form.errors:
        json_data = json.dumps(form.errors)
        return HttpResponse(json_data, content_type='app/json', status=400)
```

→ open text.py and write the updated code.

```
import json
def update_resource(id, sal):
    emp = {
        'eno' : 60
        'ename' : 'kumar'
        'esal' : sal,
        'eaddr' : 'Ameerpet'
    }
    resp = requests.put(FINAL_URL + id + '/', data=json.dumps(emp))
# resp = requests.put(FINAL_URL + id + '/', data=emp)
```

Scanned by CamScanner

```
    print(resp.status_code)
    print(resp.json())

id = input("enter some id: ")
sal = input('enter some salary: ')
update_resource(id, sal)
```

Code for delete the record

```
def delete_resource(id):
    resp = requests.delete(FINAL_URL + id + '/')
    if resp.status_code == 204:
        print(resp.status_code)
        print('Record deleted successfully')
    else:
        print(resp.json())

id = input("Enter some id to delete: ")
delete_resource(id)
```

Open views.py and write code for delete

```
def delete(self, request, id):
    emp = self.get_object_by_id(id)
    if emp is None:
        json_data = json.dumps({"msg":"requested resource not available"})
        return HttpResponse(json_data, content_type="application/json", status=404)
```

Scanned by CamScanner

else :

emp.delete()

Q: How do you get the data from database directly without openening the mysql client Command prompt.

Ans: we have dumpdata Command we can get data from database through Command prompt.

for ex: Python manage.py dumpdata serialize_APP.Employee --Indent 7

(serialize_APP → app name, Employee → Model name, 7 → Any integer)

Authentication and Authorization

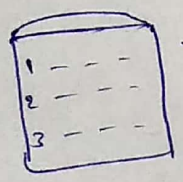
26-04-19

Security:

Q. If we want to get all records from db How do you get?

A. By using getall resources url like http://127.0.0.1:8000/api/emp/

then we will get easily data by using this url, any person can get your data, may delete, or may modify your data, means any thing they may do unnecessarly.



→ get data from db

→ Create data ~~into~~ db

→ delete/update data into db

Till now we are developing the APIs without having security. As a REST API developer we need to provide security to our APIs to Control the thirdparty persons (unautherised users)

Scanned by CamScanner

→ To provide security to our db we are using the authentication and authorization mechanisms

Authentication

The process of validating the user credentials is called as authentication

for ex: If we want to open gmail account it requires username and password.

ex: If you are a employee in any software company then security guard will ask you access card without proper Id he may not allow you into organization.

→ Most of the times we are checking the authentication by using username and password only. Some times by using tokens to check valid person or not.

→ Django REST Framework (DRF) provides several inbuilt authentication classes they are.

1. BasicAuthentication
2. SessionAuthentication
3. TokenAuthentication
4. JWT (JSON Window Token) Authentication.

Scanned by CamScanner

Authorization :

The process of validating the access permissions of the user is called authorization.

Ex: If you have one bank account, by showing your account number and some details we can get your account balance details but can't get others bank details.

Q: Is every authentication user is an authorized user or not?

→ NO,

Q. Is every Authorization user is an authentication user or not?

→ Yes.

Note: After authentication we have to perform authorization if we have that permissions.

→ DRF provides several permission classes they are

1. AllowAny → Any person can access the data, without username and password also, No restriction on data, only can Read data but not write, update, or delete.

2. IsAuthenticated → Only authenticated persons can access the data (CURD - operations all)

3. IsAdminUser → If the user is Admin user then only he is allowed to do operations (CURD), If not admin user then returns exceptions

Scanned by CamScanner

4. IsAuthenticatedOrReadOnly :→ Here any person can read the data is possible, if user is authenticated then he can do all CURD - operation.

Different ways are there to provide Security to our project.

1. Project level security
2. class level security.

27/4/19

If we want to provide project level security then inside settings.py file we are providing the security details in the form of dictionary object.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES' : ('rest_framework.
                authentication.BasicAuthentication', ),

    'DEFAULT_PERMISSION_CLASSES' : ('rest_framework.
                permissions.IsAuthenticated', )
}
```

Scanned by CamScanner

Basic Authentication :

→ This authentication model only allows already logged in users to access the REST API.

→ To make a user as a logged user, the application has to send a login page on the browser to accept username and password.

→ The application has to verify the credentials, if they are valid then it should make that user as a logged in user as well as it should redirect the user to access the REST API.

→ If credentials are not valid then the application has to send login form again to access the username and password.

steps 1. project name : Basic_Authentication_Project

2. App name : Basic_Authentication_app

3. db name : Basic_Authentication_db

4. Go to settings.py file and add our app name, rest_framework under installed apps.

5. Configure the db details

6. Create one dictionary variable for creating the security details at the end of ~~seco~~ settings.py file.

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES' : ('rest_framework.
                authentication.BasicAuthentication',),
    'DEFAULT_PERMISSION_CLASSES': ('rest_framework.
                ~~authent~~ permissions.IsAuthenticated',)
}
```

Scanned by CamScanner

⑦ Open models.py file.

```
from django.db import models

class Emp(models.Model):
    empid = models.IF(primary_key=True)
    empname = models.CF(max_length=50)
    salary = models.DecimalField(max_digit=10, decimal_places=2)
```

⑧ Create serializers.py file

```
from rest_framework import serializers
from .models import Emp

class EmpSerializer(serializers.ModelSerializer):
    class Meta:
        model = Emp
        fields = ('empid', 'empname', 'salary')
                (or)
        # fields = '__all__()'
```

⑨ open views.py file

```
from .serializers import EmpSerializer
from .models import Emp
from rest_framework import viewsets

class EmpViewSet(viewsets.ModelViewSet)
    queryset = Emp.objects.all()
    serializer_class = EmpSerializer
```

Scanned by CamScanner

10 open project level urls.py

```
from django.contrib import admin
from django.conf.urls import urls, include
urlpatterns = [
    url('admin/', admin.site.urls),
    url ('^api/', include('Basic_Authentication_APP.urls')),
]
```

11 Create app level urls.py file

```
from django.conf.urls import url, include
from .views import EmpViewSet

from rest_framework.routers import DefaultRouter
router = DefaultRouter()
router.register('emp_viewset', EmpViewSet)

urlpatterns = [
    url(r'', include(router.urls))
]
```

12. run makemigrations, migrate and create superuser and run server.

status code	Description	status code	Description
200	getting successfully	405	Method not allowed
201	Created successfully	400	Bad request.
204	deleted successfully	401	unauthorized user
404	Not found		

Scanned by CamScanner

POSTMAN

→ Postman is a third party tool we need to download this one and install it

download from → https://www.getpostman.com/downloads/

After installing we can test our APIs.

→ we can test the JAVA API, .Net API, Django APIs, ----

→ postman is always going to check endpoint urls only. But not checking this endpoint url is developed by which language or not.

By using BasicAuthentication 2 drawbacks are there

1. Username & password we send to the server in base encoding format, which is not secured.

→ Our credential are not going to store in the form of cookies, in the form of request only going.

→ we cant customize look and feel of authentication form

SessionAuthentication:

If we are using SessionAuthentication then we need to develop user defined login page

→ After providing the credentials, we need to get those values and validating those values

(Remaining is same as BasicAuthentication)

Scanned by CamScanner

30/4/19

1 Projectname : sessionauthenticationproject
2 app name : sessionauthenticationapp
3 db name : session_db
4. open settings.py file and add the app name, rest_framework and Configure the database under database section.

5. open models.py

```
from django.db import models

class Emp(models.Model):
    empid = models.IF(primary_key=True)
    empname = models.CF(m_l = 50)
    empsalary = models.DecimalField(max_digits=10, decimal_places=2)
```

6. Create Serializers.py file.

```
from .models import Emp
from rest_framework import Serializer

class EmpSerializer(Serializers.ModelSerializer):
    class Meta:
        model = Emp
        fields = ['empid', 'empname', 'empsalary']
```

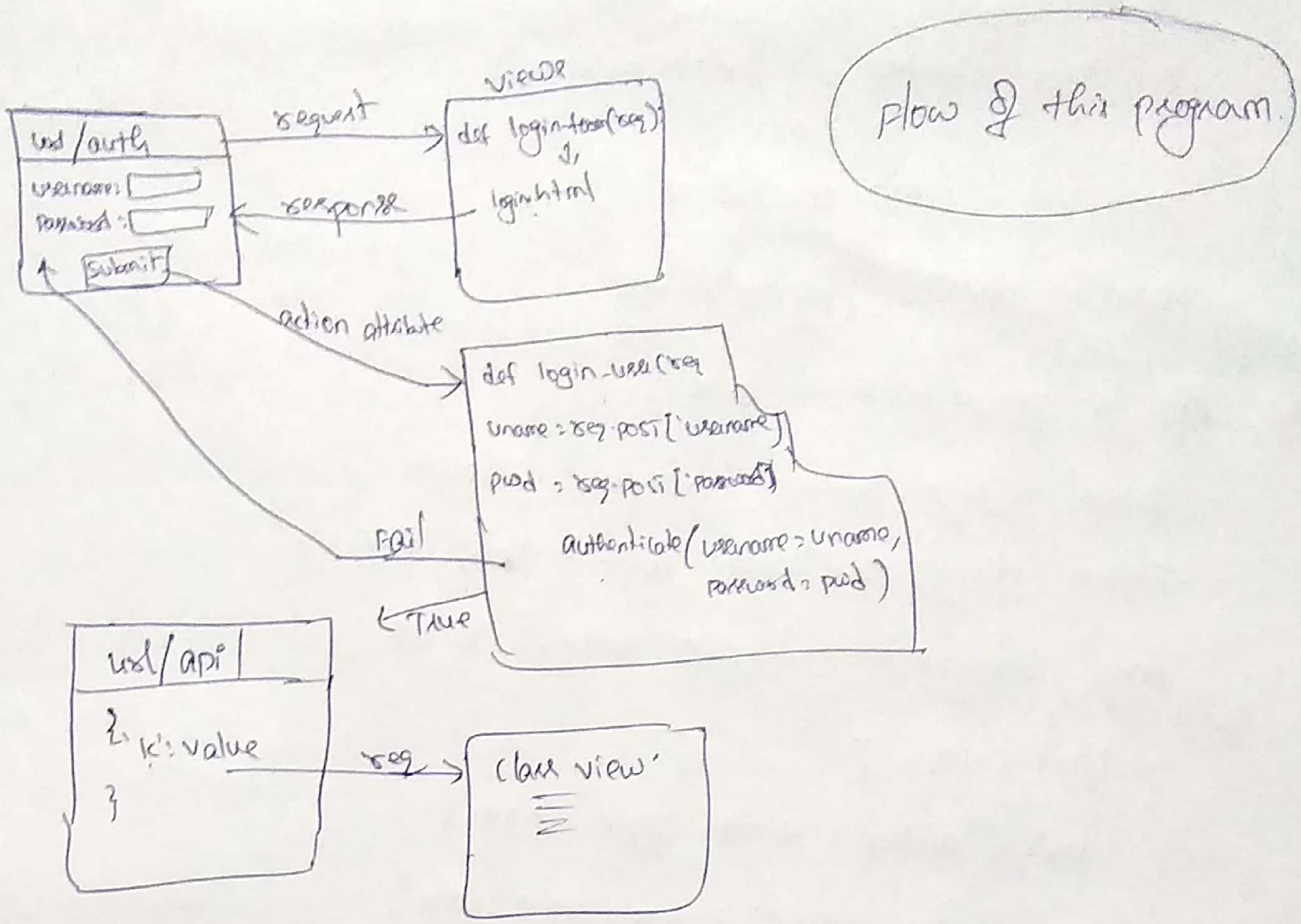
7 Open views.py

```
from .models import Emp
from rest_framework import Serializer
from rest_framework import viewsets

from rest_framework.authentication import SessionAuthentication

from rest_framework.permissions import IsAdminUser
```

Scanned by CamScanner

from django.~~for~~shortcuts import render, redirect

from django.contrib.auth import authenticate, login




```
def login_form(request):
    return render(request, 'login.html')

def login_user(request):
    username1 = request.POST['uname']
    password1 = request.POST['pwd']
    user = authenticate(username = username1, password = password1)
    if user is not None:
        login(request, user)
        return redirect('/API')
    else:
        return redirect('/auth')
```

Scanned by CamScanner

```
class EmpViewSet2(viewsets.ModelViewSet):
    authentication_classes=(SessionAuthentication,)
    Permission_classes = (IsAdminUser,)

    queryset = Emp.objects.all()
    Serializer_class = EmpSerializer
```

(8) Go to project level urls

```
from django.contrib import admin
from django.conf.urls import urls, include
from SessionAuthenticationapp import views

urlpatterns = [
    url('auth/', views.login_form),
    url('loginuser/', views.login_user),
    url(r'^api/', include('SessionAuthenticationAPP.urls')),
    url('admin/', admin.site.urls
]
```

(9) Create application level urls.py file.

```
from django.conf.url import url, include
from SessionAuthenticationAPP import views
from rest_framework.routers import defaultrouter

router = DefaultRouter()
router.register('emp_viewset', views.EmpViewset2)

urlpatterns = [
    url(r'', include(router.urls))
]
```

Scanned by CamScanner

(10) run makemigrations, migrate, create superuser, runserver commands.

(11) open templates folder and create login.htm file and write

```
<html>
<head>
    <title> Session Based project </title>
</head>
<style>
    body {
        background-color : bisque;
    }
    form {
        Padding : 10PX;
        margin : 10 PX;
    }
    table {
        border : 2PX solid red;
    }
    th {
        padding : 10 PX;
</style>
<body>
<form action = "/loginuser/" method = " POST ">
    {% csrf_token %}
    <h1> Please provide your credentials here.... </h1>
    <table>
        <tr>
            <th> Username </th>
            <td> <input type = "text" name = "uname" > </td>
        </tr>
```

Scanned by CamScanner

```
<tr>
    <th> password </th>
    <td><input type="password" name="pwd" /></td>
</tr>
<tr>
    <th></th>
    <td><input type="submit" name="sbmt" value="submit"></td>
</tr>
</table>
</form>
</body>
</html>
```

→ Now run the runserver command and check output

1/5/19

Token Based Authentication :

→ In this authentication model users or clients are authenticated by using a user "token value".

→ For every user admin generates a "token value" ~~for every~~ and it will stored in Database inside token table

→ Before accessing the RESTAPI urls a user has to get a token value from this database, by sending username & password.

→ After obtaining the token value, user has to send their "token value" with "authorization header" to access a Rest API

Q. who is responsible to create a tokenvalues by default?

Admin person is responsible to create token values.

Scanned by CamScanner

Steps 1. pro name : TokenAuthentication_Project

2. App name : TokenAuthentication_App

3 db name : Token_db

4. Open settings.py file add app name, rest_framework and rest_framework.authtoken inside Installed-Apps section.

5. Configure the db in db section.

Note: rest_framework.authtoken is a default application, this is responsible to create the tokens table

6 Add security related concepts at end of settings file

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

7. Open models.py file

```
from django.db import models

class Emp (models.Model):
    empid = models.IF (Primary_key = True)
    empname = models.CF ( m_l = 50)
    emp sal = models.DecimalField (max_digits = 10, decimal_places = 2)
    created = models.DateTimeField (auto_now_add = True)
    modified = models.DateTimeField (auto_now = True)
```

Scanned by CamScanner

8. create serializer.py file

```
from .model import Emp
from rest_framework import serializer

class EmpSerializer (Serializers.ModelSerializer):
    class Meta:
        model = Emp
        fields = '__all__'
```

9. Open views.py

```
from ~~django.db~~ .model import ~~model~~ Emp
from Serializer import EmpSerializer
from rest_framework import viewsets.
from django.shortcuts import render, redirect

class EmpViewset (viewsets.ModelViewset):
    queryset = Emp.objects.all()
    serialize_class = EmpSerializer.
```

10. Open project urls.py

```
from django.conf.urls import urls, include
from django.contrib import admin.
from ~~rest~~_framework.authtoken import views

urlpatterns = [
    url ('admin/', admin.site.urls),
    url (r'api/', include('TokenAuthentication_APP.urls'))
    url (r'gettoken/$', views.obtain_auth_token)
]
```

Emp

Scanned by CamScanner

Note: obtain_auth_token is a built in view which is available in (from rest_framework.authtoken import views)

It is the responsible to get the token value from the tokens table by taking our username and password.

→ If admin didn't generate token value then dynamically this default view will generate token value.

11 Open app level urls.py

```
from django.conf.urls import url, include
from .views import EmpViewSet
from rest_framework.routers import DefaultRouter

router = DefaultRouter()
router.register('emp-viewset', EmpViewSet)

urlpatterns = [
    url(r'', include(router.urls))
]
```

12. Do makemigrations, migrate, create superuser and runserver Commands

2/5/19

Generating the token value

step1 Create the superuser credentials, login into admin site. Admin can create 'new user' if required. All user names are stored in 'users' table section in admin site.

step2 : You can see 'Token' app section on admin site. If you click of that, initially no tokens are available. If you want to add Token value to any particular user

Scanned by CamScanner

Then click on "Add Token" button => select the require user in drop down box → click on save button => Then you will get "Token value" to that user.

Scanned by CamScanner