# Django REST API

## Create new project

* Cmd activate environment

1. *(python36) C:\Users\abhijith.m\0 zerone\Django>django-admin startproject withoutrest*
2. *(python36) C:\Users\abhijith.m\0 zerone\Django>cd withoutrest*
3. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>python manage.py startapp testapp*

* (testapp/views.py)>

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def emp\_data\_view(request):

emp\_data = {

'eno':100,

'ename':'Sunny',

'esal': 10000,

'eaddr': 'Mumbai',

}

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)

* (withoutrest/urls.py)>

from django.contrib import admin

from django.urls import path

from testapp import views

urlpatterns = [

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

path('api/', views.emp\_data\_view),

]

* Runserver
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>python manage.py runserver*
  + <http://127.0.0.1:8000/api>
* (testapp/views.py)> add

import json

def emp\_data\_jsonview(request):

emp\_data = {

'eno':100,

'ename':'Sunny',

'esal': 10000,

'eaddr': 'Mumbai',

}

resp = json.dumps(emp\_data)

return HttpResponse(resp, content\_type='application/json')

* (withoutrest/urls.py)> add

path('apijs/', views.emp\_data\_jsonview),

* <http://127.0.0.1:8000/apijs/>
* Cmd http client
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>pip install httpie*
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrest>http* [*http://127.0.0.1:8000/apijs/*](http://127.0.0.1:8000/apijs/)

HTTP/1.1 200 OK

Content-Length: 64

Content-Type: application/json

Date: Thu, 18 Jul 2019 12:42:52 GMT

Server: WSGIServer/0.2 CPython/3.6.8

X-Frame-Options: SAMEORIGIN

{

"eaddr": "Mumbai",

"ename": "Sunny",

"eno": 100,

"esal": 10000

}

* Json response without converting dict to json
* (testapp/views.py)> add

from django.http import JsonResponse

def emp\_data\_jsonview2(request):

emp\_data = {

'eno':100,

'ename':'Sunny',

'esal': 10000,

'eaddr': 'Mumbai',

}

return JsonResponse(emp\_data)

* (withoutrest/urls.py)> add

path('apijs2/', views.emp\_data\_jsonview2),

* Create file ‘test.py’ in any other location other than project directory

import requests

import json

BASE\_URL = 'http://127.0.0.1:8000/'

ENDPOINT = 'apijs'

resp = requests.get(BASE\_URL+ENDPOINT)

print(resp.json())

* Run test.py file

## Generic view

* (testapp/views.py)> add

from django.views.generic import View

class jsonCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp\_data={

'eno':100,

'ename': 'Sunny',

'esal':1000,

'eaddr':'Mumbai',

}

return JsonResponse(emp\_data)

* (withoutrest/urls.py)> add

path('apijscbv/', views.jsonCBV.as\_view()),

* In browser <http://127.0.0.1:8000/apijscbv/>
* (testapp/views.py)> update

from django.views.generic import View

class jsonCBV(View):

def get(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from get method'})

return HttpResponse(json\_data, content\_type='application/json')

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

return HttpResponse(json\_data, content\_type='application/json')

def put(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from put method'})

return HttpResponse(json\_data, content\_type='application/json')

def delete(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from delete method'})

return HttpResponse(json\_data, content\_type='application/json')

* Test.py

import requests

import json

BASE\_URL = 'http://127.0.0.1:8000/'

ENDPOINT = 'apijscbv'

resp = requests.post(BASE\_URL+ENDPOINT)

print(resp.json())

* It should return an error msg. because the csrf token is not defined.
* The middleware setting reject this request
* (withoutrest/settings.py)> inside ‘MIDDLEWARE’ >
* Comment the following line

# 'django.middleware.csrf.CsrfViewMiddleware',

* Test.py request ‘’resp = requests.delete(BASE\_URL+ENDPOINT)’’

### Basics of arguments

def f1(\*\*kwargs):

print(kwargs)

f1(name='abhi', rollno=100, mark=90)

{'name': 'abhi', 'rollno': 100, 'mark': 90}

def f1(\*args):

print(args)

f1()

f1(90)

f1(90,20,30)

()

(90,)

(90, 20, 30)

### Mixin

* (testapp/)> create file mixins.py

from django.http import HttpResponse

class HttpResponseMixin(object):

def render\_to\_http\_response(self, json\_data):

#1000 lines of code

return HttpResponse(json\_data, content\_type='application/json')

* (testapp/views.py)> update

from django.views.generic import View

from testapp.mixins import HttpResponseMixin

class jsonCBV(HttpResponseMixin, View):

def get(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from get method'})

return self.render\_to\_http\_response(json\_data)

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

return self.render\_to\_http\_response(json\_data)

def put(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from put method'})

return self.render\_to\_http\_response(json\_data)

def delete(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from delete method'})

return self.render\_to\_http\_response(json\_data)

## CRUD without REST framework

* In cmd

1. *(python36) C:\Users\abhijith.m\0 zerone\Django>django-admin startproject withoutrestm*
2. *(python36) C:\Users\abhijith.m\0 zerone\Django>cd withoutrestm*
3. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py startapp testapp*

* (withoutrestm/settings.py)> inside ‘INSTALLED\_APPS’ add

'testapp',

* (testapp/models.py)>

class Employee(models.Model):

id = models.AutoField(primary\_key=True)

eno = models.IntegerField()

ename = models.CharField(max\_length=64)

esal = models.FloatField()

eaddr = models.CharField(max\_length=64)

* (testapp/admin.py).

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)

* In cmd
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py makemigrations*
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py migrate*
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py createsuperuser*

*Username (leave blank to use 'abhijith.m'): admin*

*Email address: admin@example.com*

*Password:*

*Password (again):*

*Superuser created successfully.*

* Start server
  + *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py runserver*

### Read

#### Based on id

* (testapp/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

# Create your views here.

class EmployeeDetailCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp=Employee.objects.get(id=1)

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')

* (withoutrestm/urls.py)> add

from testapp import views

* (withoutrestm/urls.py)> inside ‘urlpatterns’

path('api/', views.EmployeeDetailCBV.as\_view())

* In browser <http://127.0.0.1:8000/api/>
* Provide id explicitly
* (testapp/views.py)> update

class EmployeeDetailCBV(View):

def get(self, request, id, \*args, \*\*kwargs):

emp=Employee.objects.get(id=id)

* (withoutrestm/urls)> update

path('api/<int:id>', views.EmployeeDetailCBV.as\_view())

#### Serializer

* (testapp/views.py)> add

from django.core.serializers import serialize

* (testapp/views.py)> update

# Create your views here.

class EmployeeDetailCBV(View):

def get(self, request, id, \*args, \*\*kwargs):

emp=Employee.objects.get(id=id)

json\_data = serialize('json', [emp,], fields=('eno', 'ename','eaddr'))

return HttpResponse(json\_data, content\_type='application/json')

* Get all
* (testapp/views.py)> add

class EmployeeListCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp=Employee.objects.all()

json\_data = serialize('json', emp)

return HttpResponse(json\_data, content\_type='application/json')

* (withoutrestm/urls)> add inside ‘urlpatterns’

path('api/', views.EmployeeListCBV.as\_view())

* Run test.py

import requests

import json

BASE\_URL = 'http://127.0.0.1:8000/'

ENDPOINT = 'api/'

def get\_resource(id):

resp = requests.get(BASE\_URL+ENDPOINT+id+'/')

print(resp.status\_code)

print(resp.json())

# id = input("Enter id:")

# get\_resource(id)

def get\_all():

resp = requests.get(BASE\_URL+ENDPOINT)

print(resp.status\_code)

print(resp.json())

get\_all()

* To exclude ‘’model and pk‘’ from out put
* (testapp/views.py)> add

class EmployeeListCBV(View):

def get(self, request, \*args, \*\*kwargs):

emp=Employee.objects.all()

json\_data = serialize('json', emp)

pdict = json.loads(json\_data)

final\_list = []

for obj in pdict:

emp\_data = obj['fields']

final\_list.append(emp\_data)

json\_data = json.dumps(final\_list)

return HttpResponse(json\_data, content\_type='application/json')

* (testapp)> create file mixins.py

from django.core.serializers import serialize

import json

class SerializerMixin(object):

def serialize(self, qs):

json\_data = serialize('json', qs)

p\_data = json.loads(json\_data)

final\_list = []

for obj in p\_data:

emp\_data = obj['fields']

final\_list.append(emp\_data)

json\_data = json.dumps(final\_list)

return json\_data

* (testapp/views.py)> update

from testapp.mixins import SerializerMixin

# Create your views here.

class EmployeeDetailCBV(SerializerMixin, View):

def get(self, request, id, \*args, \*\*kwargs):

qs=Employee.objects.get(id=id)

json\_data = self.serialize([qs,])

return HttpResponse(json\_data, content\_type='application/json')

# Create your views here.

class EmployeeListCBV(SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return HttpResponse(json\_data, content\_type='application/json')

#### Exception: Avoid DoesNotExist error

* (testapp/views.py)> update

class EmployeeDetailCBV(SerializerMixin, View):

def get(self, request, id, \*args, \*\*kwargs):

try:

qs=Employee.objects.get(id=id)

except :

json\_data = json.dumps({'Error': "The requested resource is not available"})

else:

json\_data = self.serialize([qs,])

return HttpResponse(json\_data, content\_type='application/json')

* Add httpResponse in mixin
* (testapp/mixin.py)> add

from django.http import HttpResponse

class HttpresponseMixin(object):

def render\_to\_http\_response(self, json\_data, status=200):

return HttpResponse(json\_data, content\_type='application/json', status=status)

* (testapp/views.py)> update

class EmployeeDetailCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, id, \*args, \*\*kwargs):

try:

qs=Employee.objects.get(id=id)

except :

json\_data = json.dumps({'Error': "The requested resource is not available"})

return self.render\_to\_http\_response(json\_data, status=404)

else:

json\_data = self.serialize([qs,])

return self.render\_to\_http\_response(json\_data)

#### Save or Display all data in table by console code

* In cmd
  1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee*

**indented format**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --indent 4*

**indented format with json**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format json --indent 4*

**XML format**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format xml --indent 4*

**Save file to emp.xml**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format xml >emp.xml --indent 4*

**Save in yaml format**

* 1. *(python36) C:\Users\abhijith.m\0 zerone\Django\withoutrestm>python manage.py dumpdata testapp.Employee --format yaml >emp.yaml --indent 4*

### Create

* (testapp/views.py)> update

class EmployeeListCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return HttpResponse(json\_data, content\_type='application/json')

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

self.render\_to\_http\_response(json\_data)

* Test.py > add

def create\_resource():

new\_emp = {

'eno':500,

'ename':'Shiva',

'esal': 5000,

'eaddr': 'Chennai',

}

new\_emp = json.dumps(new\_emp)

resp = requests.post(BASE\_URL+ENDPOINT, data=new\_emp)

print(resp.status\_code)

print(resp.json())

create\_resource()

### csrf token disabling

1. Method Level
2. Class Level

from django.views.decorators.csrf import csrf\_exempt

from django.utils.decorators import method\_decorator

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeListCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

qs = Employee.objects.all()

json\_data = self.serialize(qs)

return HttpResponse(json\_data, content\_type='application/json')

def post(self, request, \*args, \*\*kwargs):

json\_data = json.dumps({'msg':'This is from post method'})

return self.render\_to\_http\_response(json\_data)

1. Project Level

* Run test.py
* **To check client was send a valid json data or not**
* (testapp)> create new file ‘utils.py’

import json

def is\_json(data):

try:

p\_data = json.loads(data)

valid = True

except ValueError:

valid = False

return valid

* (testapp/views.py)> add

from testapp.utils import is\_json

* (testapp/views.py)> update

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeListCBV(HttpresponseMixin, SerializerMixin, View):

def get(self, request, \*args, \*\*kwargs):

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

valid\_jason = is\_json(data)

if not valid\_jason:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

json\_data = json.dumps({'msg':'You provided valid json data'})

return self.render\_to\_http\_response(json\_data, status=400)

### store database

* (testapp)> create new file forms.py

from django import forms

from testapp.models import Employee

class EmployeeForm(forms.ModelForm):

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\_\_'

* (testapp/views.py)> update inside ‘class EmployeeListCBV’

def post(self, request, \*args, \*\*kwargs):

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

empdata = json.loads(data)

form = EmployeeForm(empdata)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource created successfully'})

return self.render\_to\_http\_response(json\_data)

if form.errors:

json\_data = json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data, status=400)

* test.py

def create\_resource():

new\_emp = {

'eno':500,

'ename':'Shiva',

'esal': '50000',

'eaddr': 'Chennai',

}

new\_emp = json.dumps(new\_emp)

resp = requests.post(BASE\_URL+ENDPOINT, data=new\_emp)

print(resp.status\_code)

print(resp.json())

create\_resource()

* Run test.py

## Update

* (testapp/views.py)> add

from django.views.decorators.csrf import csrf\_exempt

from django.utils.decorators import method\_decorator

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeDetailCBV(HttpresponseMixin, SerializerMixin, View):

* (testapp/views.py)> add inside ‘class EmployeeDetailCBV’

def get\_object\_by\_id(self,id):

try:

emp = Employee.objects.get(id=id)

except Employee.DoesNotExist:

emp = None

return emp

def put(self,request, id, \*args, \*\*kwargs):

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found for updation"})

return self.render\_to\_http\_response(json\_data, status=404)

data = request.body

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

provided\_data = json.loads(data)

original\_data = {

'eno':emp.eno,

'ename':emp.ename,

'esal':emp.esal,

'eaddr': emp.eaddr

}

original\_data.update(provided\_data)

form = EmployeeForm(original\_data, instance=emp)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource Updated successfully'})

return self.render\_to\_http\_response(json\_data)

if form.errors:

json\_data = json.dumps(form.errors)

return self.render\_to\_http\_response(json\_data, status=400)

* Test.py > add and run

def update\_resource(id):

new\_emp = {

'esal': '70000',

'eaddr': 'Delhi',

}

new\_emp = json.dumps(new\_emp)

resp = requests.put(BASE\_URL+ENDPOINT+str(id)+'/', data=new\_emp)

print(resp.status\_code)

print(resp.json())

update\_resource(6)

## Delete

* (testapp/views.py)> add

def delete(self, request,id, \*args, \*\*kwargs):

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found for deletion"})

return self.render\_to\_http\_response(json\_data, status=404)

status,deleted\_item = emp.delete()

if status == 1:

json\_data = json.dumps({'Msg': "Resource deleted successfully"})

return self.render\_to\_http\_response(json\_data)

json\_data = json.dumps({'Error': "Unable to delete. Please try again"})

return self.render\_to\_http\_response(json\_data, status=404)

* Test.py > add and run

def delete\_resource(id):

resp = requests.delete(BASE\_URL+ENDPOINT+str(id)+'/')

print(resp.status\_code)

print(resp.json())

delete\_resource(6)

## By using single baseurl

* (testapp/views.py)> add

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeCRUDCBV(HttpresponseMixin, SerializerMixin, 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

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is not None:

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found."})

return self.render\_to\_http\_response(json\_data, status=404)

json\_data = self.serialize([emp,])

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)

* Test.py > add and run

def get\_resource(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())

get\_resource()

get\_resource(2)

* (testapp/views.py)> add

@method\_decorator(csrf\_exempt, name='dispatch')

class EmployeeCRUDCBV(HttpresponseMixin, SerializerMixin, 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

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is not None:

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found."})

return self.render\_to\_http\_response(json\_data, status=404)

json\_data = self.serialize([emp,])

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

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

empdata = json.loads(data)

form = EmployeeForm(empdata)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource created successfully'})

return self.render\_to\_http\_response(json\_data)

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

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is None:

json\_data = json.dumps({'msg':'please provide valid id for updation'})

return self.render\_to\_http\_response(json\_data, status=400)

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found. Not possible to update"})

return self.render\_to\_http\_response(json\_data, status=404)

provided\_data = json.loads(data)

original\_data = {

'eno':emp.eno,

'ename':emp.ename,

'esal':emp.esal,

'eaddr': emp.eaddr

}

original\_data.update(provided\_data)

form = EmployeeForm(original\_data, instance=emp)

if form.is\_valid():

form.save(commit=True)

json\_data = json.dumps({'msg':'Resource Updated successfully'})

return self.render\_to\_http\_response(json\_data)

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

valid\_json = is\_json(data)

if not valid\_json:

json\_data = json.dumps({'msg':'please send valid json data only'})

return self.render\_to\_http\_response(json\_data, status=400)

pdata = json.loads(data)

id = pdata.get('id', None)

if id is not None:

emp = self.get\_object\_by\_id(id)

if emp is None:

json\_data = json.dumps({'Error': "No matched resources are found."})

return self.render\_to\_http\_response(json\_data, status=404)

status,deleted\_item = emp.delete()

if status == 1:

json\_data = json.dumps({'Msg': "Resource deleted successfully"})

return self.render\_to\_http\_response(json\_data)

json\_data = json.dumps({'Error': "Unable to delete. Please try again"})

return self.render\_to\_http\_response(json\_data, status=404)

json\_data = json.dumps({'Error': "Unable to delete. Please provide valid id"})

return self.render\_to\_http\_response(json\_data, status=404)

* Test.py > add

# create\_resource()

def update\_resource\_b(id):

new\_emp = {

'id': id,

'esal': 10000,

'eaddr': 'Delhi',

}

new\_emp = json.dumps(new\_emp)

resp = requests.put(BASE\_URL+ENDPOINT, data=new\_emp)

print(resp.status\_code)

print(resp.json())

# update\_resource\_b(7)

def delete\_resource\_b(id):

data= {

'id': id

}

resp = requests.delete(BASE\_URL+ENDPOINT, data=json.dumps(data))

print(resp.status\_code)

print(resp.json())

delete\_resource\_b(7)

# With REST Frame work

* Create project > in cmd
  1. *(python36) E:\django projects>django-admin startproject withrestc4*
  2. *(python36) E:\django projects>cd withrestc4*
  3. *(python36) E:\django projects\withrestc4>python manage.py startapp testapp*
* (withrestc4/settings.py)> add inside ’ INSTALLED\_APPS’

'rest\_framework',

'testapp'

* (testapp/models.py)> add

class Employee(models.Model):

id = models.AutoField(primary\_key=True)

eno = models.IntegerField()

ename = models.CharField(max\_length=64)

esal = models.FloatField()

eaddr = models.CharField(max\_length=64)

* (testapp/admin.py)> add

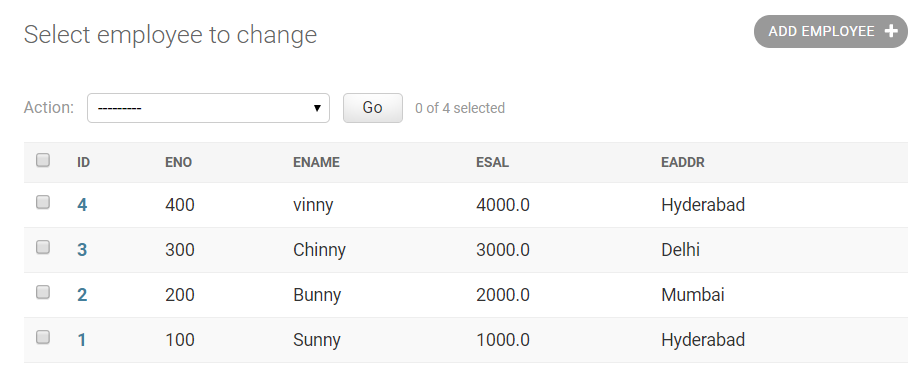
from testapp.models import Employee

# Register your models here.

class EmployeeAdmin(admin.ModelAdmin):

list\_display = ['id','eno', 'ename', 'esal', 'eaddr']

admin.site.register(Employee, EmployeeAdmin)

* Install package
  1. *(python36) E:\django projects\withrestc4>pip install djangorestframework*
* In cmd>
  1. *(python36) E:\django projects\withrestc4>python manage.py makemigrations*
  2. *(python36) E:\django projects\withrestc4>python manage.py migrate*
  3. *(python36) E:\django projects\withrestc4>python manage.py createsuperuser*
     + *Username (leave blank to use 'user'): admin*
     + *Email address: admin@example.com*
     + *Password:*
     + *Password (again):*
     + *Superuser created successfully.*
  4. *(python36) E:\django projects\withrestc4>python manage.py runserver*
* In browser <http://127.0.0.1:8000/admin/> add the following employees**
* (testapp/)> create new file ‘serializers.py’

from testapp.models import Employee

from rest\_framework.serializers import ModelSerializer

class EmployeeSerializer(ModelSerializer):

class Meta:

model = Employee

fields = '\_\_all\_\_'

* (withrestc4/urls.py)> add

from django.contrib import admin

from django.urls import path, include

from rest\_framework import routers

from testapp import views

router = routers.DefaultRouter()

#router.register('api', views.EmployeeCRUDCBV, base\_name='api')

router.register('api', views.EmployeeCRUDCBV)

urlpatterns = [

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

path("", include(router.urls))

]

* (testapp/views.py)> add

from django.shortcuts import render

from rest\_framework.viewsets import ModelViewSet

from testapp.models import Employee

from testapp.serializers import EmployeeSerializer

# Create your views here.

class EmployeeCRUDCBV(ModelViewSet):

queryset = Employee.objects.all()

serializer\_class = EmployeeSerializer

* <http://127.0.0.1:8000/api/> , <http://127.0.0.1:8000/api/1/> and add an employee

# model\_to\_dict

from django.forms.models import model\_to\_dict

model\_to\_dict(instance, fields=[field.name for field in instance.\_meta.fields])

# Sort Django template list

{% for key, val in dict.items|sort %}

key: {{key}} / {{value}}

{% endfor %}

# Query Dict to dict 🡨 not working

New in Django >= 1.4.

QueryDict.dict()

# Django re install db

import os

import shutil

root = os.getcwd()

parent\_folders = os.listdir(root)

if "db.sqlite3" in parent\_folders:

    os.remove("db.sqlite3")

else:

    ValueError("This is not django parent folder")

app\_names = []

for r\_f in parent\_folders:

    try:

        sub1\_f = os.listdir(r\_f)

    except:

        continue

    if "migrations" in sub1\_f:

        # delete migrations folder

        path\_migrations = os.path.join(r\_f, "migrations")

        try:

            shutil.rmtree(path\_migrations)

            app\_names.append(r\_f)

        except:

            ValueError("Cant remove folder %s" %(str(path\_migrations)))

# create migrations

apps = " ".join(app\_names)

os.system("python manage.py makemigrations %s"%(apps))

os.system("python manage.py migrate")

os.system("python manage.py createsuperuser --username admin --email admin@example.org")

os.system("python manage.py runserver")

# Errors

## Sqlite 3.8.\* above required

As this was about Centos7, you can use the Fedora package to upgrade the Centos sqlite package:

wget https://kojipkgs.fedoraproject.org//packages/sqlite/3.8.11/1.fc21/x86\_64/sqlite-3.8.11-1.fc21.x86\_64.rpm

sudo yum install sqlite-3.8.11-1.fc21.x86\_64.rpm

# Streamlit

## Documentation

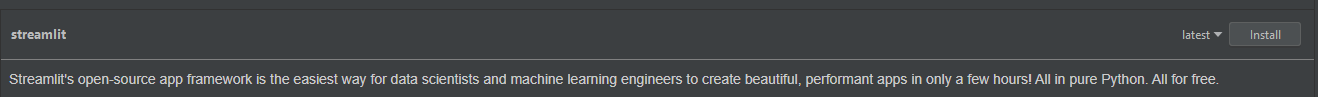
<https://docs.streamlit.io/library/get-started>

Github [code](https://github.com/Harsh1347/Streamlit)

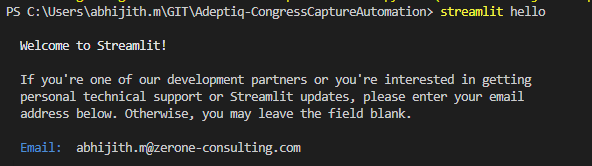
YouTube [Video](https://www.youtube.com/watch?v=UN4DaSAZel4&list=PLuU3eVwK0I9PT48ZBYAHdKPFazhXg76h5)

## Installation

* + - * + Install streamlit



* + - * Streamlit hello



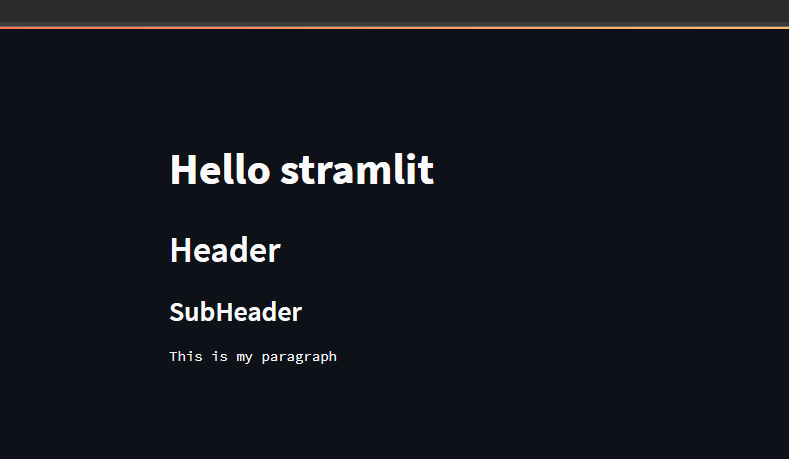
## Hello world

import streamlit as st  
  
st.title("Hello stramlit")

* + - * streamlit run test.py

## text

import streamlit as st  
  
st.title("Hello stramlit")  
st.header("Header")  
st.subheader("SubHeader")  
st.text("This is my paragraph")



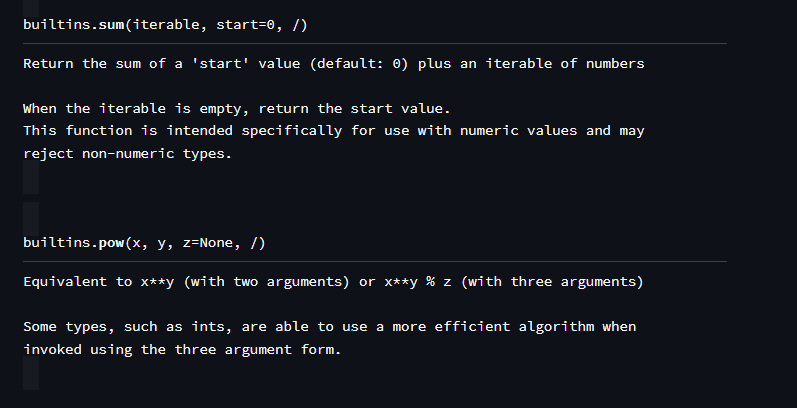
## markdown

<https://docs.streamlit.io/library/api-reference/text/st.markdown>

<https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet>

### write – writes module

st.write(sum, pow)



## Display Data

Github [code](https://github.com/Harsh1347/Streamlit)

import streamlit as st  
import pandas as pd  
import numpy as np  
import time  
  
a = [1,2,3,4,5,6,7,8]  
n = np.array(a)  
nd = n.reshape((2,4))  
dic = {  
 "name":["harsh","Gupta"],  
 "age":[21,32],  
 "city":["noida","delhi"]  
}  
  
data = pd.read\_csv("data//Salary\_Data.csv")  
  
st.dataframe(data,width=500,height= 500)  
st.table(dic)  
st.json(dic)  
st.write(dic)  
  
@st.cache  
def ret\_time(a):  
 time.sleep(5)  
 return time.time()  
  
if st.checkbox("1"):  
 st.write(ret\_time(1))  
  
if st.checkbox("2"):  
 st.write(ret\_time(2))

## Grid

from st\_aggrid import GridOptionsBuilder, AgGrid, GridUpdateMode, DataReturnMode

# cors issue in django

<https://www.stackhawk.com/blog/django-cors-guide/>

solution:- <https://www.geeksforgeeks.org/how-to-enable-cors-headers-in-your-django-project/>

# Autostart Django service on server

To automatically start a Django service on server start, you need to follow these general steps:

1. Create a systemd service file for your Django project
2. Enable the systemd service
3. Test the systemd service

Here's how you can do it:

1. Create a systemd service file for your Django project

Create a file named **yourproject.service** in **/etc/systemd/system/** directory with the following content:

[Unit]  
Description=Your Project Service  
After=network.target  
  
[Service]  
User=yourusername  
Group=www-data  
WorkingDirectory=/path/to/your/project  
ExecStart=/path/to/your/virtualenv/bin/gunicorn yourproject.wsgi:application -w 4 -b 127.0.0.1:8000  
  
[Install]  
WantedBy=multi-user.target

Make sure to replace **yourusername**, **/path/to/your/project**, **/path/to/your/virtualenv/**, and **yourproject.wsgi:application** with the actual values for your project.

1. Enable the systemd service

Run the following command to enable the service:

sudo systemctl enable yourproject.service

This will create a symlink from the service file to the systemd target directory, which will enable the service to start automatically at boot time.

1. Test the systemd service

Start the service with the following command:

sudo systemctl start yourproject.service

Check the status of the service with the following command:

sudo systemctl status yourproject.service

If you need to make changes to the service unit file, you can edit it and then reload the systemd configuration and restart the service:

sudo systemctl daemon-reload  
sudo systemctl restart myproject.service

# Create docker service

To create a Docker service for your Django project, you can follow these steps:

1. Create a Dockerfile in the root directory of your Django project. Open a text editor and create a file named "Dockerfile" with the following content:

FROM python:3.9  
  
ENV PYTHONUNBUFFERED 1  
  
WORKDIR /code  
  
COPY requirements.txt /code/  
RUN pip install -r requirements.txt  
  
COPY . /code/

Note: Replace "python:3.9" with the version of Python you are using for your Django project.

1. Create a requirements.txt file in the root directory of your Django project. Open a text editor and create a file named "requirements.txt" with the list of packages required by your Django project, one package per line.
2. Build the Docker image by running the following command in the terminal, from the root directory of your Django project:

docker build -t myproject:latest .

Note: Replace "myproject" with the name you want to give to your Docker image.

1. Create a Docker Compose file. Open a text editor and create a file named "docker-compose.yml" with the following content:

version: '3'  
  
services:  
 web:  
 build: .  
 command: python manage.py runserver 0.0.0.0:8000  
 ports:  
 - "8000:8000"

1. Start the Docker service by running the following command in the terminal, from the root directory of your Django project:

docker-compose up

Your Django project should now be running in a Docker container and accessible at [http://localhost:8000](http://localhost:8000/). If you need to make changes to your Django project, you can edit the code files and the Docker service will automatically reload the changes.

Note: You may need to make additional configuration changes to the Docker Compose file depending on your project's needs, such as adding environment variables or linking to a database service.