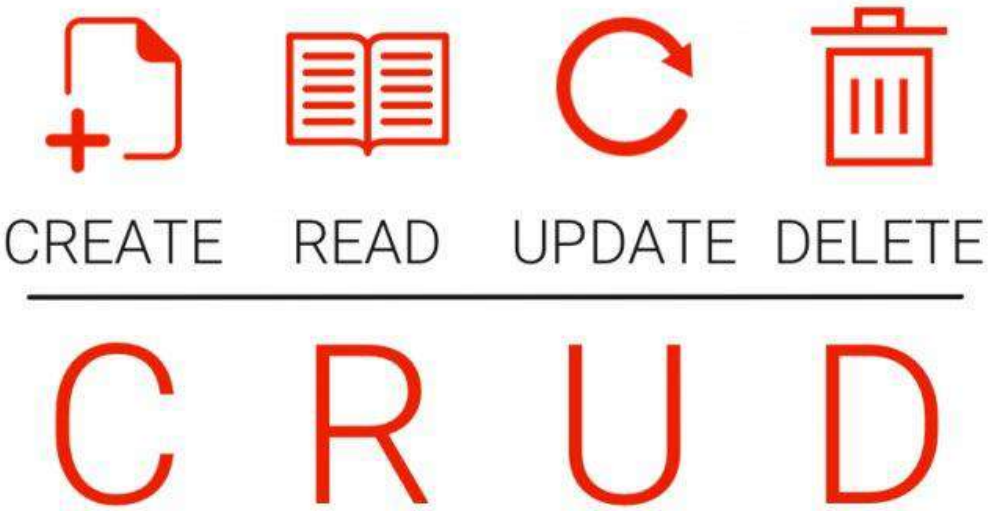
Sathya Technologies

Django Rest Framework

Notes By

Naveen

Database Operations as JSON response without using Rest Framework



Preparing a model

Note: All CURD operations we perform on the same model.

models.py

```
from django.db import models
class ProductModel(models.Model):
    no = models.IntegerField(primary_key=True)
    name = models.CharField(unique=True, max_length=30)
    price = models.FloatField()
    quantity = models.IntegerField()
```

Note : Once the model class is ready do makemigrations and migrate.

My Database Table (In My Table Some data is pre inserted)

	no	name	price	quantity
1	101	Canon	25000	10
2	102	Kids Camera	5890	10
3	103	Mi RedMi 8	12500	15
4	104	Dell Laptop	45000	5
5	105	Mac book	89000	2

Read

1) In this example the Django application is reading all the products from database and giving as JSON response to Postman using Class based view.

Note : In this we are using python built-in package "json"

urls.py

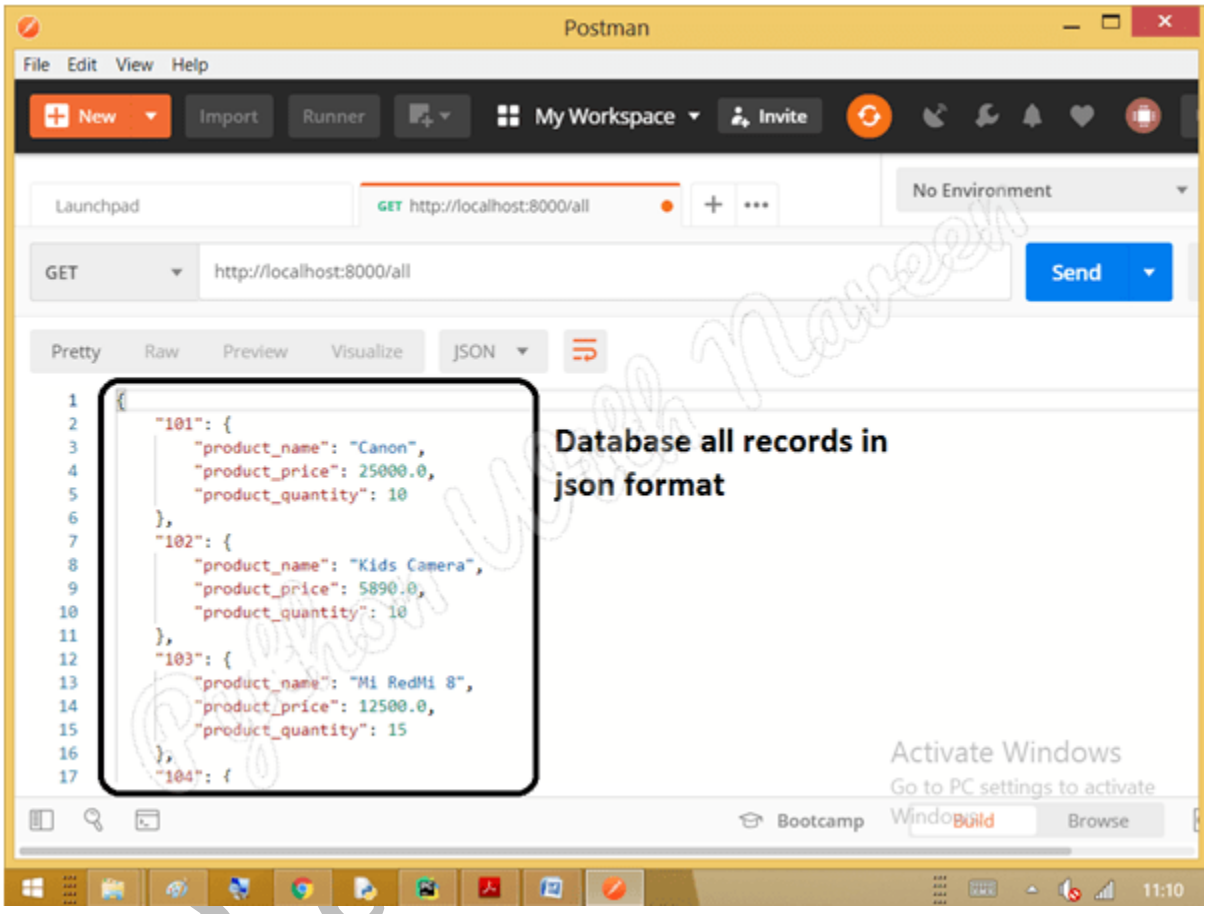
path('all/',views.ViewAllProducts.as_view(),name="index_class")

views.py

```
from django.http import HttpResponse
from django.views.generic import View
from app1.models import ProductModel
import json
class ViewAllProducts(View):
    def get(self,request):
        qs = ProductModel.objects.all()
        # reading data from Queryset and making a dictionary
        data = {} # Empty dictionary
        for row in qs:
            d1 = {
                row.no:{
                  "product_name":row.name,
                  "product_price":row.price,
                  "product_quantity":row.quantity}}
            data.update(d1)
        # Converting dictionary to JSON
        json_data = json.dumps(data)
```

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

Output



Serializing Django objects

Django's serialization framework provides a mechanism for "translating" Django models into other formats.

Usually these other formats will be text-based and used for sending Django data over a wire, but it's possible for a serializer to handle any format (text-based or not).

Example :

def serialize(format, queryset)

def serialize(format, queryset, fileds=(tuple))

2) In this example the Django application is reading all the products from database and giving as JSON response to Postman using Class based view.

Note: In this we are using django built in " serializers"

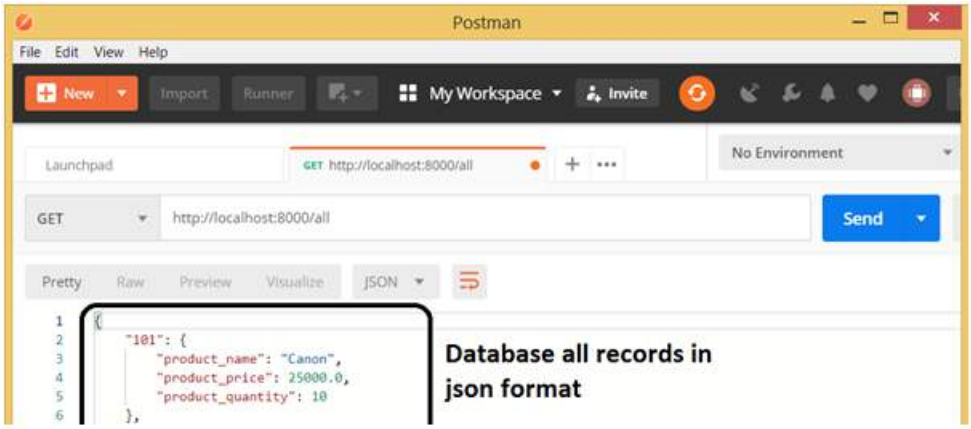
urls.py

```
path('all/',views.ViewAllProducts.as_view(),name="index_class")
```

views.py

```
from django.core.serializers import serialize
from django.http import HttpResponse
from django.views.generic import View
from app1.models import ProductModel
class ViewAllProducts(View):
    def get(self,request):
        qs = ProductModel.objects.all()
        json_data = serialize('json',qs)
        return
HttpResponse(json_data,content_type="application/json")
```

Output



3) In this example the Django application is reading 1 product from database and giving as JSON response to Postman using Class based view.

Note : In this we are using python built-in package "json"

urls.py

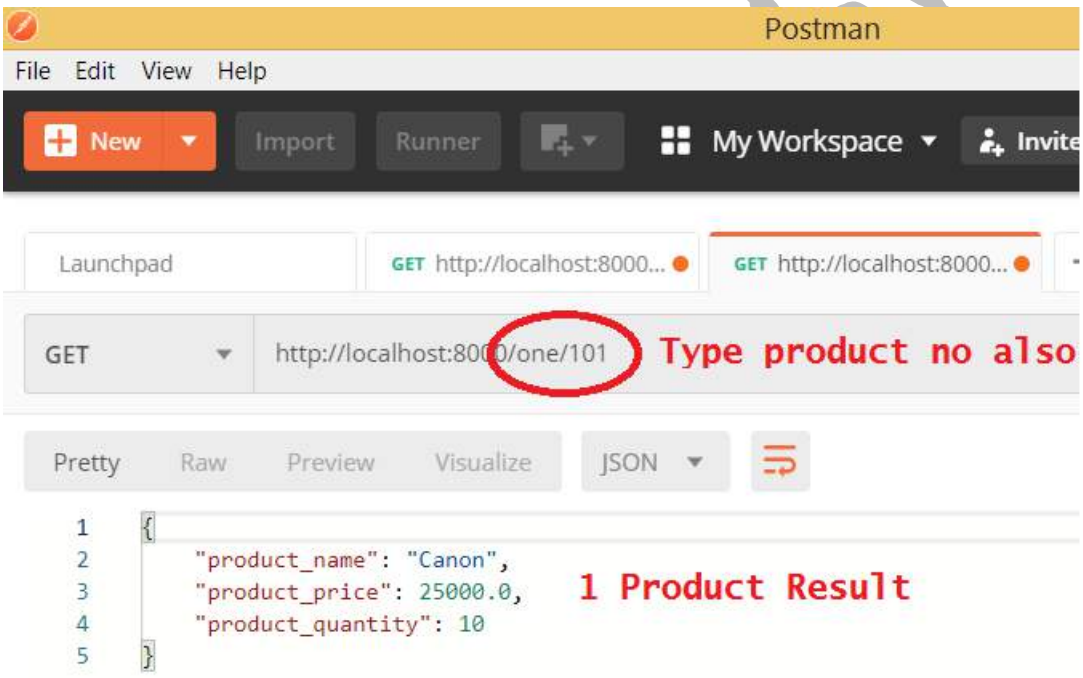
```
path('one/<int:product>',views.ViewOneProduct.as_view())
```

views.py

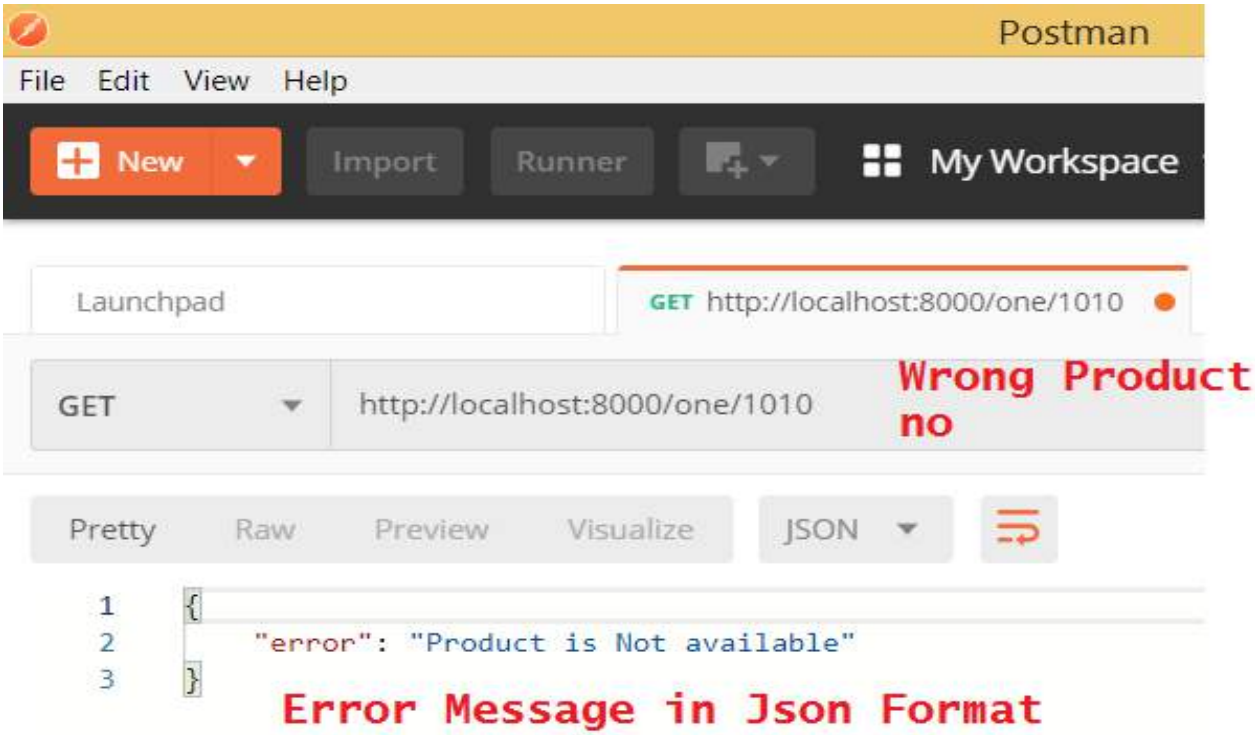
```
from django.http import HttpResponse
from django.views.generic import View
from app1.models import ProductModel
import json
class ViewOneProduct(View):
    def get(self,request,product):
        try:
            qs = ProductModel.objects.get(no=product)
            data = {"product_name":qs.name,
                    "product_price":qs.price,
                    "product_quantity":qs.quantity}
            json_data = json.dumps(data)
```

```
    except ProductModel.DoesNotExist:
        error_mess = {"error":"Product No is Not available"}
        json_data = json.dumps(error_mess)
    return
HttpResponse(json_data,content_type="application/json")
```

Output



----> In the above example we are handling the exception and sending a error message in json.



4) In this example the Django application is reading 1 product from database and giving as JSON response to Postman using Class based view.

Note: In this we are using django built in " serializers"

urls.py

path('**one/<int:product>**',views.ViewOneProduct.as_view())

views.py

```
class ViewOneProduct(View):
    def get(self,request,product):
        try:
            res = ProductModel.objects.get(no=product)
            json_data = serialize('json',[res])
        except ProductModel.DoesNotExist:
            error_mess = {"error":"Product No is Not available"}
```

```
        json_data = json.dumps(error_mess)
    return
HttpResponse(json_data,content_type="application/json")
```

Output

GET http://localhost:8000/one/101

Pretty Raw Preview Visualize JSON

```
1  [
2      {
3          "model": "app1.productmodel",
4          "pk": 101,
5          "fields": {
6              "name": "Canon",
7              "price": 25000.0,
8              "quantity": 10
9          }
10     }
11 ]
```

Diffrent Output for serializers

Create

1) In this example the Django application is inserting 1 product into database and Django application is taking input from postman.

Note: In this example we are doing post operations

urls.py

path('insert_one/',views.InsertOneProduct.as_view(),name="insert_one")

views.py

```
from django.views.generic import View

class InsertOneProduct(View):
    def post(self,request):
        print(request.body)
```

Note: To read data from postman, in views class we use **"request.body"**

The above example will give error message because the post method required CSRF-TOKEN verification.

Error Message :

```
Run: N1
Forbidden (CSRF cookie not set.): /insert_one/
[2                    9] "POST /insert_one/ HTTP/1.1" 403
Forbidden (CSRF cookie not set.): /insert_one/
[2                    5] "POST /insert_one/ HTTP/1.1" 403
```

CSRF verification failed. Request aborted.

Solution to the above error is CSRF TOKEN Disable

Solution-1

1) Open settings.py file from project package.

2) In settings.py file move to "MIDDLEWARE" list

3) In "MIDDLEWARE" list remove or comment the " **'django.middleware.csrf.CsrfViewMiddleware'** ".

4) If you remove or comment this will affect the complete project.

Solution-2

1) Open **"views.py"** and do import.

2) **from** django.views.decorators.csrf **import** csrf_exempt

3) **from** django.utils.decorators **import** method_decorator

4) In **"view.py"** move to the class where u need to disable

5) @method_decorator(csrf_exempt,name=**'dispatch'**)
   class InsertOneProduct(View):

6) If you fallow the solution 2 it will effect only 1 class.

urls.py

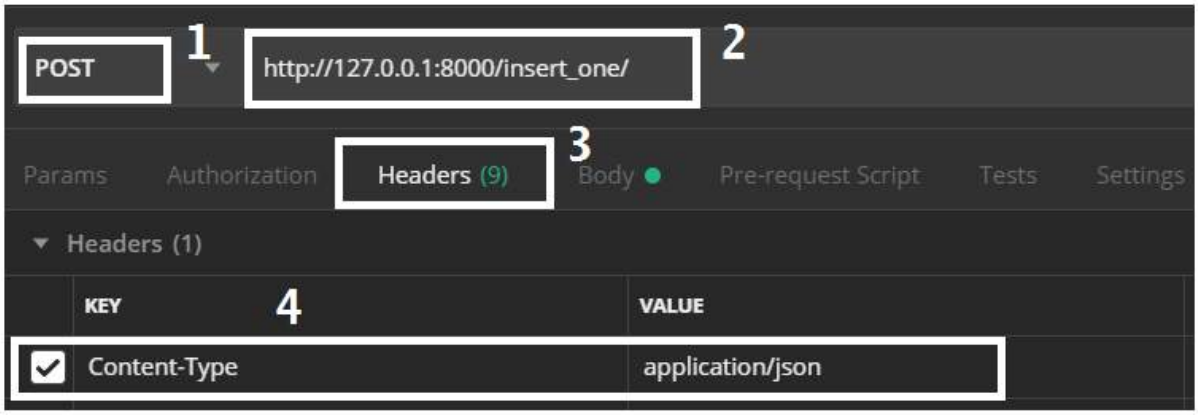
```
path('insert_one/',views.InsertOneProduct.as_view(),name="insert_one")
```

views.py

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

@method_decorator(csrf_exempt,name='dispatch')
class InsertOneProduct(View):
    def post(self,request):
        print(request.body)
```

To send data from Postman



Output in Server

```
Run: N1
b'{\n\t"idno":106,\n\t"name":"OPPO",\n\t"price":15000.00,\n\t"quant
Internal Server Error: /insert_one/
Traceback (most recent call last):
```

We can see the output in server Note : data is not in json format

To Convert the data into "json" format we use loads function.

Example

```
def post(self,request):
    data = request.body
    json_data =  json.loads(data)
    print(json_data)
```

```
Run: N1
{'idno': 106, 'name': 'OPPO', 'price': 15000.0, 'quantity': 10}
Internal Server Error: /insert_one/
Traceback (most recent call last):
```

After converting the data in json format

To Save the data into database we need to create a form.

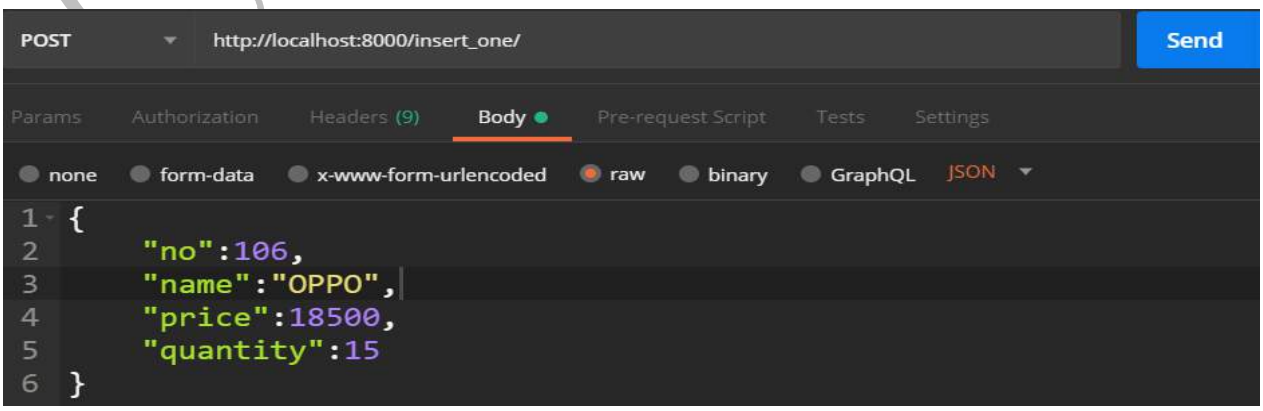
```
from django import forms
from app1.models import ProductModel
class ProductForm(forms.ModelForm):
    # Built in validations
    no = forms.IntegerField(min_value=101)
    class Meta:
        model = ProductModel
        fields = '__all__'
```

Changes in views.py

```
from django.http import HttpResponse
from django.views.generic import View
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from app1.forms import ProductForm
import json

@method_decorator(csrf_exempt,name='dispatch')
class InsertOneProduct(View):
    def post(self,request):
        data = request.body
        json_data =  json.loads(data)
        pf = ProductForm(json_data)
        if pf.is_valid():
            pf.save()
            json_data = json.dumps({"success":"Product is saved"})
        else:
            json_data = json.dumps(pf.errors)

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



On send Button Click the output is

```
POST    http://localhost:8000/insert_one/
Pretty  Raw  Preview  Visualize  JSON
1 {
2     "success": "Product is saved"
3 }
```

If you provide product no less than 101 it will return error message which is built in forms.

Input

```
POST    http://localhost:8000/insert_one/    Send
Params  Authorization  Headers (9)  Body  Pre-request Script  Tests  Settings
none  form-data  x-www-form-urlencoded  raw  binary  GraphQL  JSON
1 {
2     "no":10,
3     "name":"OPPO",
4     "price":18500,
5     "quantity":15
6 }
```

1) Product no is less than 101

2) Oppo Name is available in DB

You will get 2 Error Messages.

Output

```
1 {
2     "no": [
3         "Ensure this value is greater than or equal to 101."
4     ],
5     "name": [
6         "Product model with this Name already exists."
7     ]
8 }
```

forms.py (With custom validations)

```
from django import forms
from app1.models import ProductModel
class ProductForm(forms.ModelForm):
    class Meta:
        model = ProductModel
        fields = '__all__'
  # Custom Validation
    def clean_price(self):
        price = self.cleaned_data["price"]
        if price >= 1000:
            return price
        else:
            raise forms.ValidationError("Price Minimum of RS:1000 are allowed")
```

Note: No changes in views.py class, changes only in forms.py class

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
    "no":107,
    "name":"Samsung",
    "price":500,
    "quantity":15
}
```

Price is less than 1000

Output

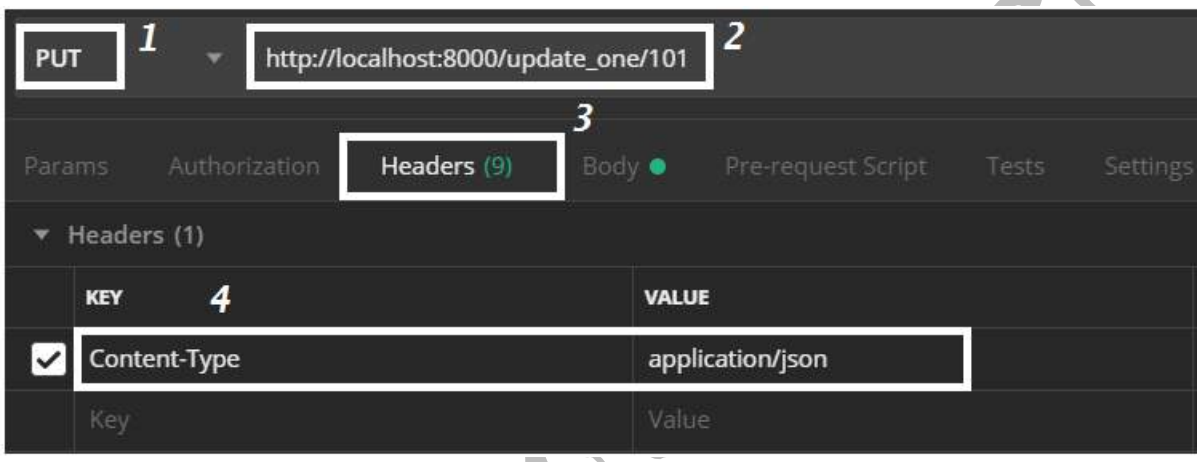
```
{
    "price": [
        "Price Minimum of RS:1000 are allowed"
    ]
}
```

Update

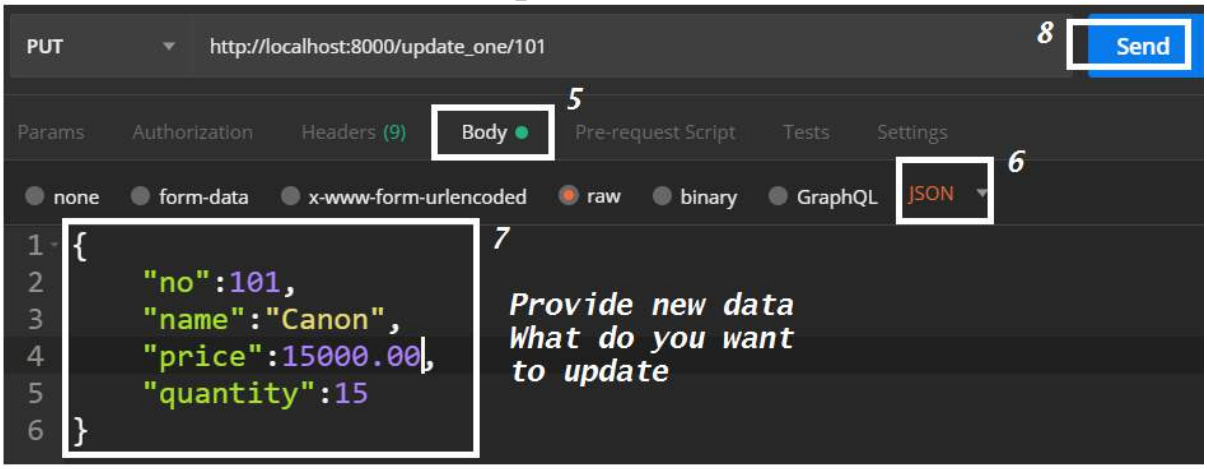
1) In this example the Django application is updating 1 product into database and Django application is taking input from postman.

Note: In this example we are doing put operations.

Sending Data from Postman (Step - 1)



Sending Data from Postman (Step - 2)



urls.py

path('**update_one/<product>**',views.UpdateOneProduct.as_view())

views.py

```
from django.http import HttpResponse
from django.views.generic import View
from app1.models import ProductModel
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from app1.forms import ProductForm
import json

@method_decorator(csrf_exempt,name='dispatch')
class UpdateOneProduct(View):
    def put(self,request,product):
        try:
            old_product = ProductModel.objects.get(no=product)
            new_product = json.loads(request.body)
            pf = ProductForm(new_product,instance=old_product)
            if pf.is_valid():
                pf.save()
                json_data = json.dumps({"success": "Product is Updated"})
            else:
                json_data = json.dumps(pf.errors)
            return HttpResponse(json_data,
content_type="application/json")
        except ProductModel.DoesNotExist:
            json_data = json.dumps({"error":"Invalid Product No Dude"})
            return
HttpResponse(json_data,content_type="application/json")
```

Output : 1

PUT 1 http://localhost:8000/update/one/1011 2 Wrong Product No 3 Send

Pretty Raw Preview Visualize JSON

```
1 {
2     "error": "Invalid Product No Dude"
3 }
```

4

PUT 1 http://localhost:8000/update/one/101 2 Valid Product No 3 Send

Pretty Raw Preview Visualize JSON

```
1 {
2     "success": "Product is Updated"
3 }
```

In This above example we need to provide complete details of the product else you will get error

none form-data x-www-form-urlencoded raw

```
1 {
2     "price":15000.00
3 }
```

Trying to Update only Price

If we are not providing complete product details we get error like

```
2     "no": [
3         "This field is required."
4     ],
5     "name": [
6         "This field is required."
7     ],
8     "quantity": [
9         "This field is required."
```

Solution to the above program.

views.py

```
from django.http import HttpResponse
from django.views.generic import View
from app1.models import ProductModel
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from app1.forms import ProductForm
import json

@method_decorator(csrf_exempt,name='dispatch')
class UpdateOneProduct(View):
    def put(self,request,product):
        try:
            old_product = ProductModel.objects.get(no=product)
            new_product = json.loads(request.body)

            data = {
                "no":old_product.no,
                "name":old_product.name,
                "price":old_product.price,
                "quantity":old_product.quantity
            }

            for key,value in new_product.items():
                data[key] = value

            pf = ProductForm(data,instance=old_product)
            if pf.is_valid():
                pf.save()
```

```
            json_data = json.dumps({"success": "Product is Updated"})
        else:
            json_data = json.dumps(pf.errors)
        return HttpResponse(json_data,
content_type="application/json")
    except ProductModel.DoesNotExist:
        json_data = json.dumps({"error":"Invalid Product No Dude"})
        return
HttpResponse(json_data,content_type="application/json")
```

Output

Trying to Update only Price

Note:

1) ProductForm(json_data) # To save a new Record

2) ProductForm(data,instance=old_product) # To Update an old record.

Delete

1) In this example the Django application is deleting 1 product from database and Django application is taking input from postman.

Note: In this example we are doing delete operations.

urls.py

```
path('delete_one/<product>',views.DeleteOneProduct.as_view())
```

views.py

```
from django.http import HttpResponse
from django.views.generic import View
from app1.models import ProductModel
from django.views.decorators.csrf import csrf_exempt
from django.utils.decorators import method_decorator
from app1.forms import ProductForm
import json

@method_decorator(csrf_exempt,name="dispatch")
class DeleteOneProduct(View):
  def delete(self,request,product):
    try:
      result = ProductModel.objects.get(no=product).delete()
      if result[0] == 1:
        json_data = json.dumps({"message": "Product is Deleted "})
        return HttpResponse(json_data,
content_type="application/json")

    except ProductModel.DoesNotExist:
      json_data = json.dumps({"error":"Invalid Product No Dude"})
```

return HttpResponse(json_data,content_type="**application/json**")

Output