DJANGO REST FRAMEWORK

BY MOHAMMAD HASAN KHODDAMI

WHAT WE GONNA LEARN

- 1- DRF for sure !
- 2- creating document by swagger
- 3- using Postman

HOW TO INSTALL DRF ?

- By using following command :
  - Pip install djangorestfremwork
- Also you have to introduce it into installed apps on setting.py

The correct is framework

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_freamwork'
]
```

REQUEST AND RESPONSE STRUCTURE IN DRF

- Request : extends httprequest
- Now what httprequest ? → the request that we claim from the client are instances from httprequest which give us some option like : the method of the incoming request, figuring out who is the client, authentication and etc…

REQUEST.DATA AND REQUEST.QUERY_PARAMS

- In DRF instead of having request.files and request.post we have request.data

- Query params are the query which used in url and we can have access to that
- Also we know about request.user

RESPONSE

- As we know every request has a response
- Our response are the instances from response class
- The built in function for sending response is :
- Response (data, …) , just data is necessary
- Data param in response Is native python`s data types like dictionary or list or tuple or etc…
- Response won`t serialize for us. Response take the native python`s data type and change it to Json and send it to request

CREATE VIEW

- As you know we know we have 2 type of function FBV and CBV in DRF also we have 2 types CBV and FBV same as Django but it`s recommended to use CBV.
  - But for start we use function base view

@API_VIEW

Now we want to create our first view py API_view decorator

Mistyping correct is →framework

```
from rest_freamwork.decorators import api_view
from rest_freamwork.response import Response
```

Step 1 : import api_view and Response (as we said our responses are instances from Response class)

```
@api_view()
def hello_world(request)
return Response('message' : hello world)
```

Step 2 : create our function , note : we didn`t define our method so by default it`s GET

```
@api_view(['GET' , 'POST'])
def hello_world(request)
return Response('message' : hello world)
```

Step 3 : Define the methods

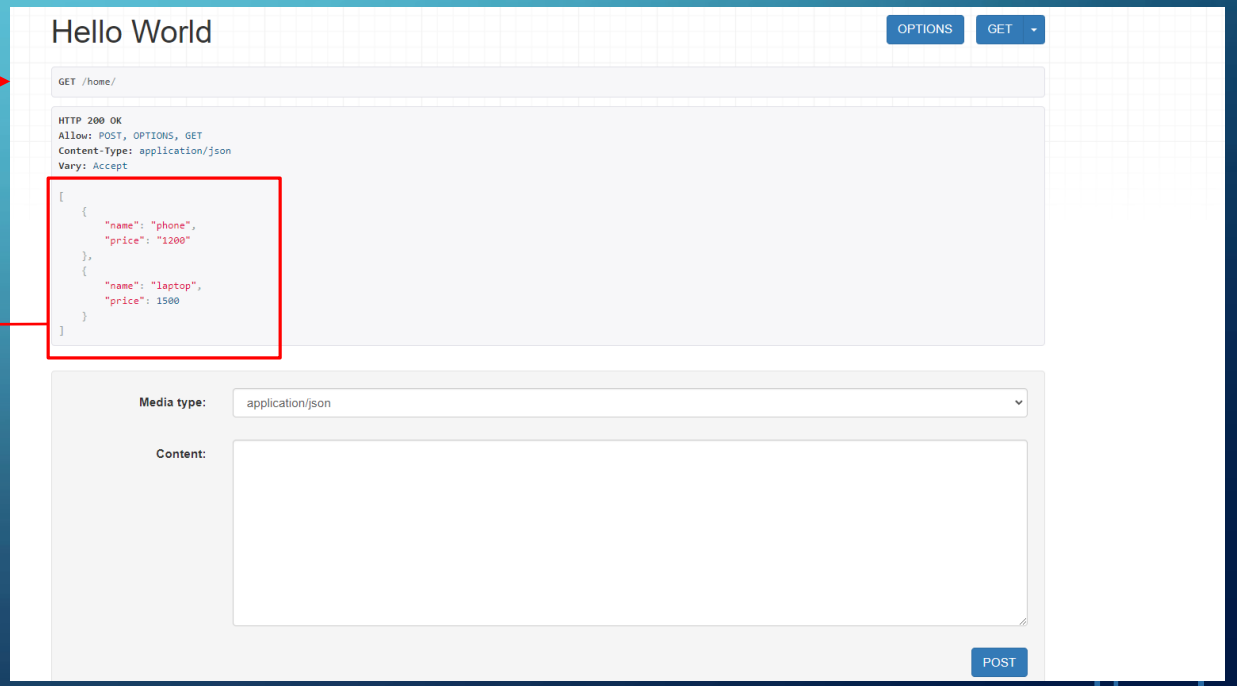
Step 4,5,6,7… : is creating urls just like Django.

EXAMPLE AND ELABORATION

I wrote this code and sent it to Response. (note : Response gets native python data`s)

```
home > views.py > hello_world
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
# Create your views here.


@api_view(['GET', 'POST'])
def hello_world(request):

    production = [
        {'name': 'phone',
         'price': '1200'},
        {'name': 'laptop',
         'price': 1500}
    ]
    return Response(production)
```

This is the result
DRF has default template



This is the data`s which can use as API !
Others will use it on their website !
Done !!

CBV IN DRF

The other settings just like Django.
And I turn hello world FBV into CBV.

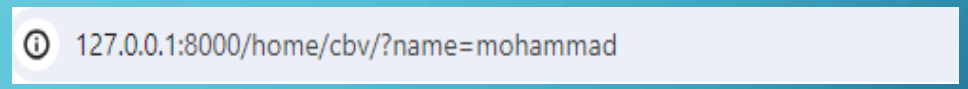
```python
from rest_framework.views import APIView
```

```python
class HelloWorld(APIView):
    production = [
        {'name': 'phone',
         'price': '1200'},
         {'name': 'laptop',
          'price': 1500}
    ]

    def get(self, request):

        return Response(self.production)

    def post(self, request):

        return Response(self.production)
```

GETTING QUERY PARAMS

This is query params (after ?) and we want to create a view for it

- A. I used get built in for if there is no name return me none !
- B. I just changed Get method.

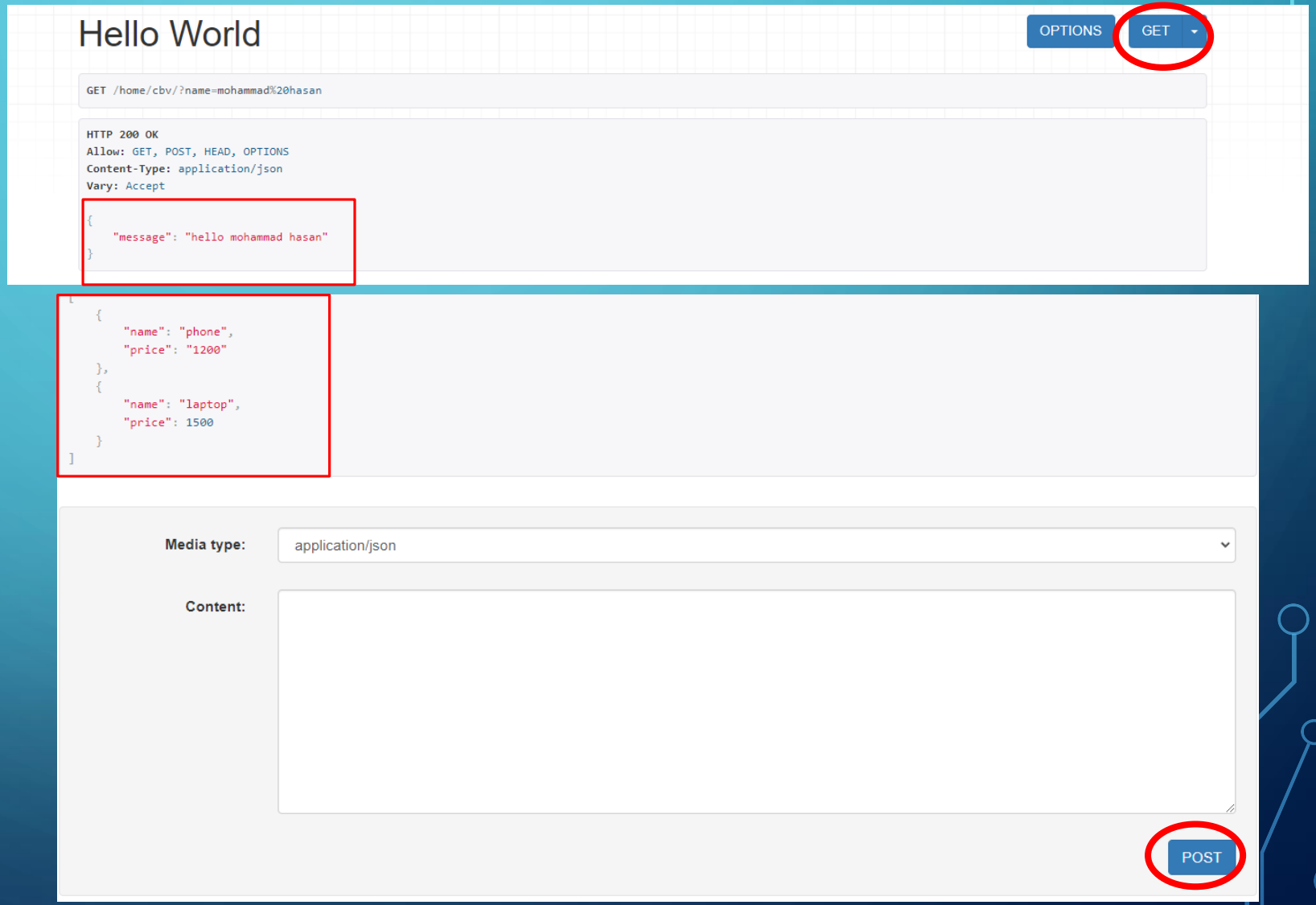



```python
class HelloWorld(APIView):
    production = [
        {'name': 'phone',
         'price': '1200'},
        {'name': 'laptop',
         'price': 1500}
    ]

    def get(self, request):
        name = request.GET.get('name')
        return Response({'message' : f'hello {name}'})

    def post(self, request):

        return Response(self.production)
```

RESULT !

By using GET method

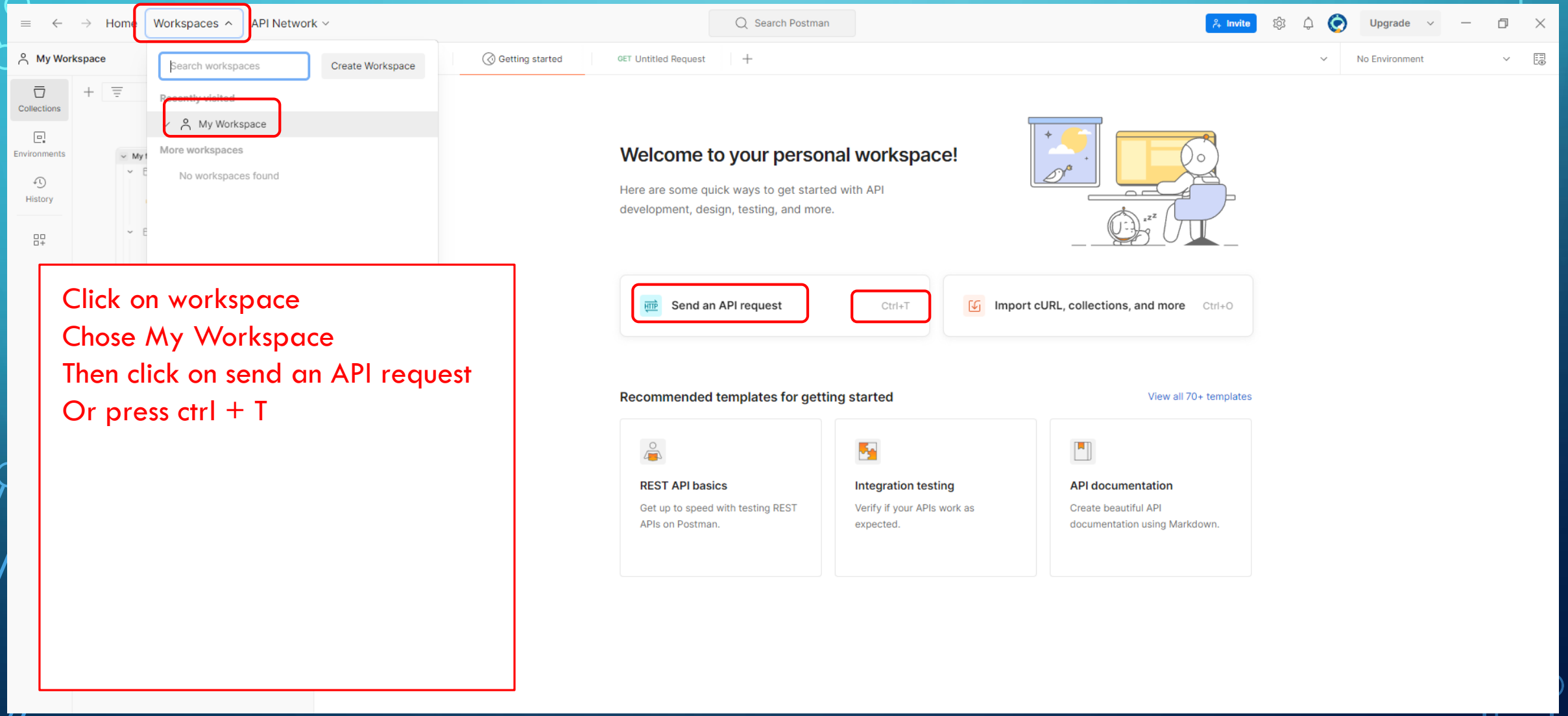
By using POST method give us the production content because I've changed the POST method



INSTALLING THE POSTMAN

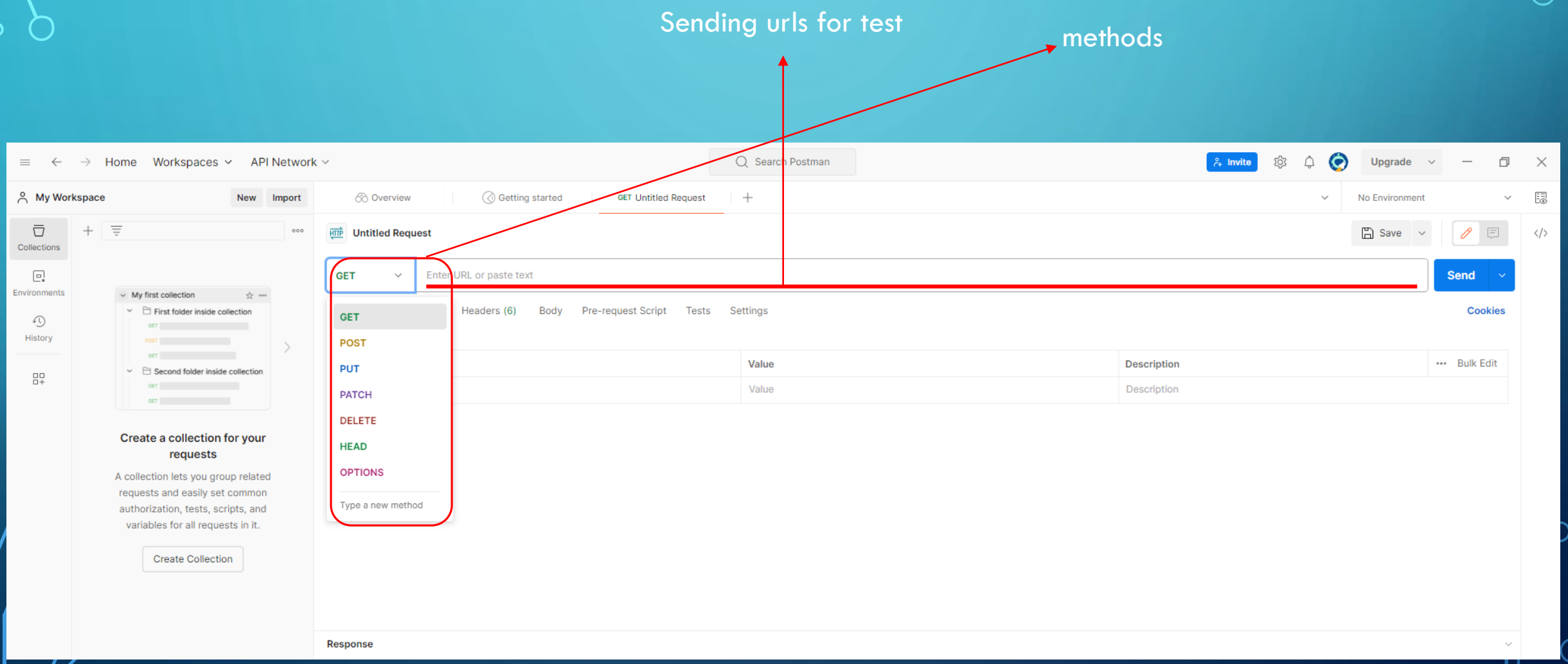
- Download PostMan from following link :
  - https://www.postman.com/downloads/
- Create an account and launch the app
- By using postman we can test our API

POSTMAN

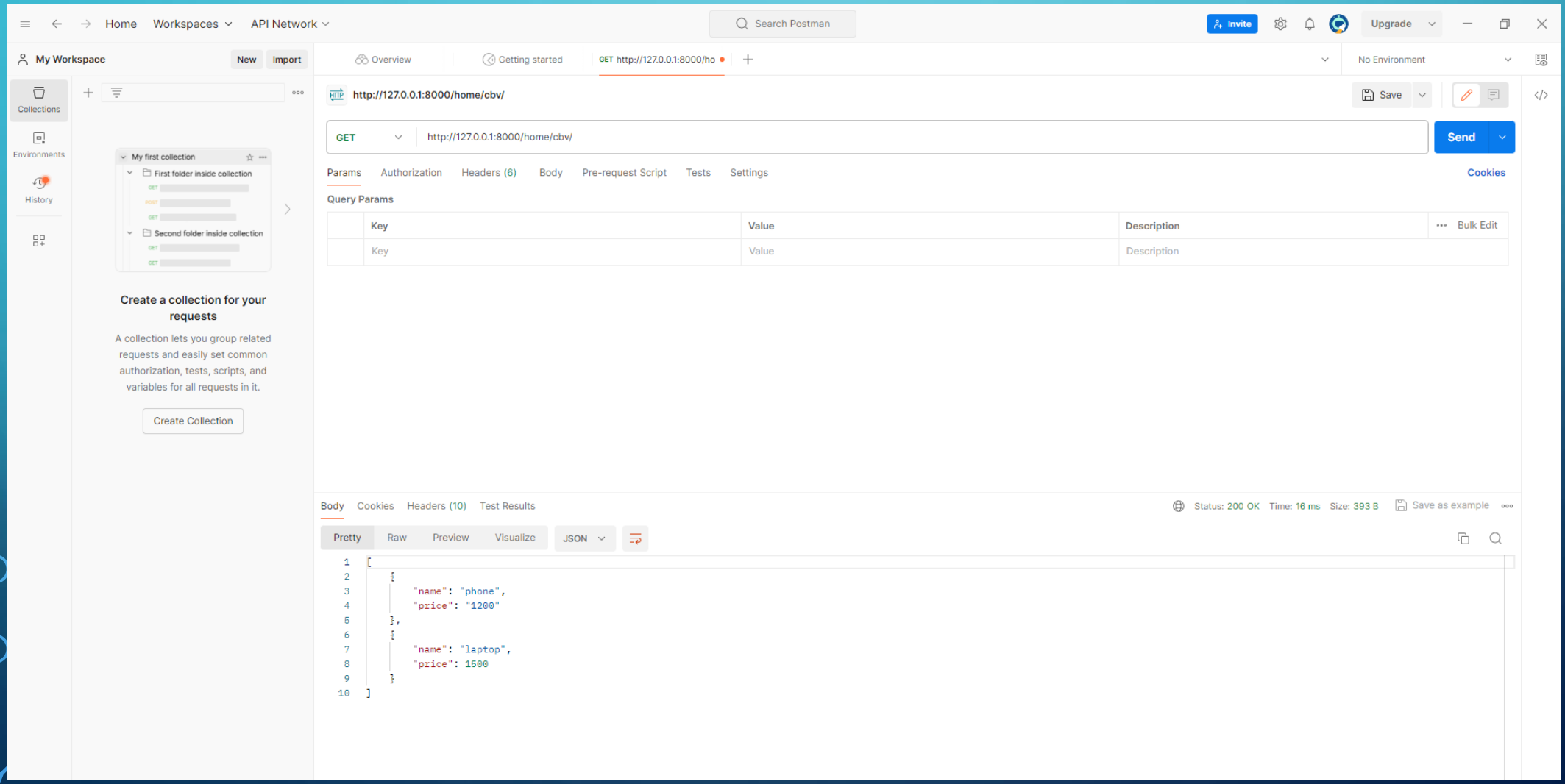


Click on workspace
Chose My Workspace
Then click on send an API request
Or press ctrl + T

POSTMAN


Sending urls for test
methods
Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
GET Untitled Request
No Environment
Collections
Environments
History
Untitled Request
Save
GET
Enter URL or paste text
Send
GET
POST
PUT
PATCH
DELETE
HEAD
OPTIONS
Type a new method
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Value
Description
Bulk Edit
Value
Description
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Response

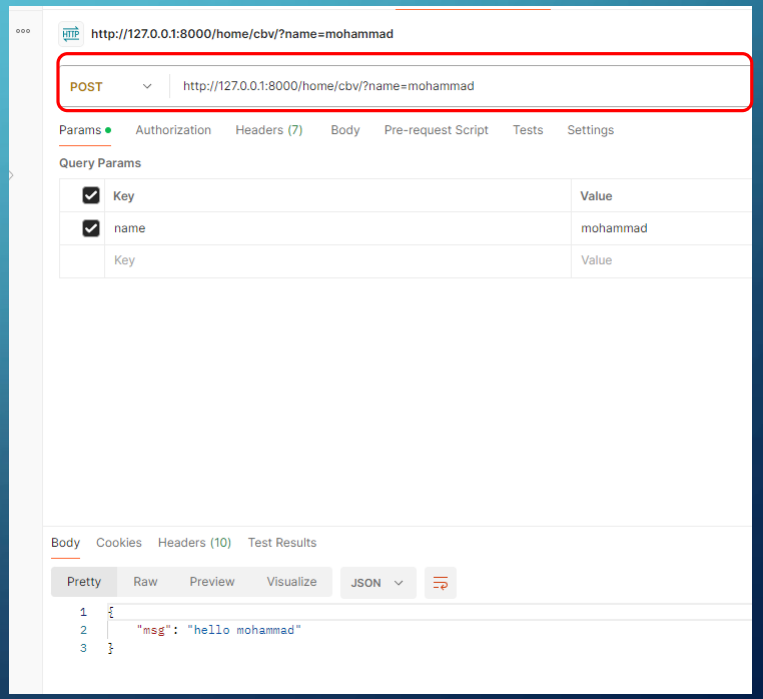
POSTMAN-GET


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
My Workspace
New
Import
Overview
Getting started
GET http://127.0.0.1:8000/ho
No Environment
Collections
Environments
History
http://127.0.0.1:8000/home/cbv/
Save
GET
http://127.0.0.1:8000/home/cbv/
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
My first collection
First folder inside collection
Second folder inside collection
Create a collection for your requests
A collection lets you group related requests and easily set common authorization, tests, scripts, and variables for all requests in it.
Create Collection
Body
Cookies
Headers (10)
Test Results
Status: 200 OK
Time: 16 ms
Size: 393 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
[
{
"name": "phone",
"price": "1200"
},
{
"name": "laptop",
"price": 1500
}
]

POSTMAN-POST

If you want to send query_params as post you have to use request.query_params

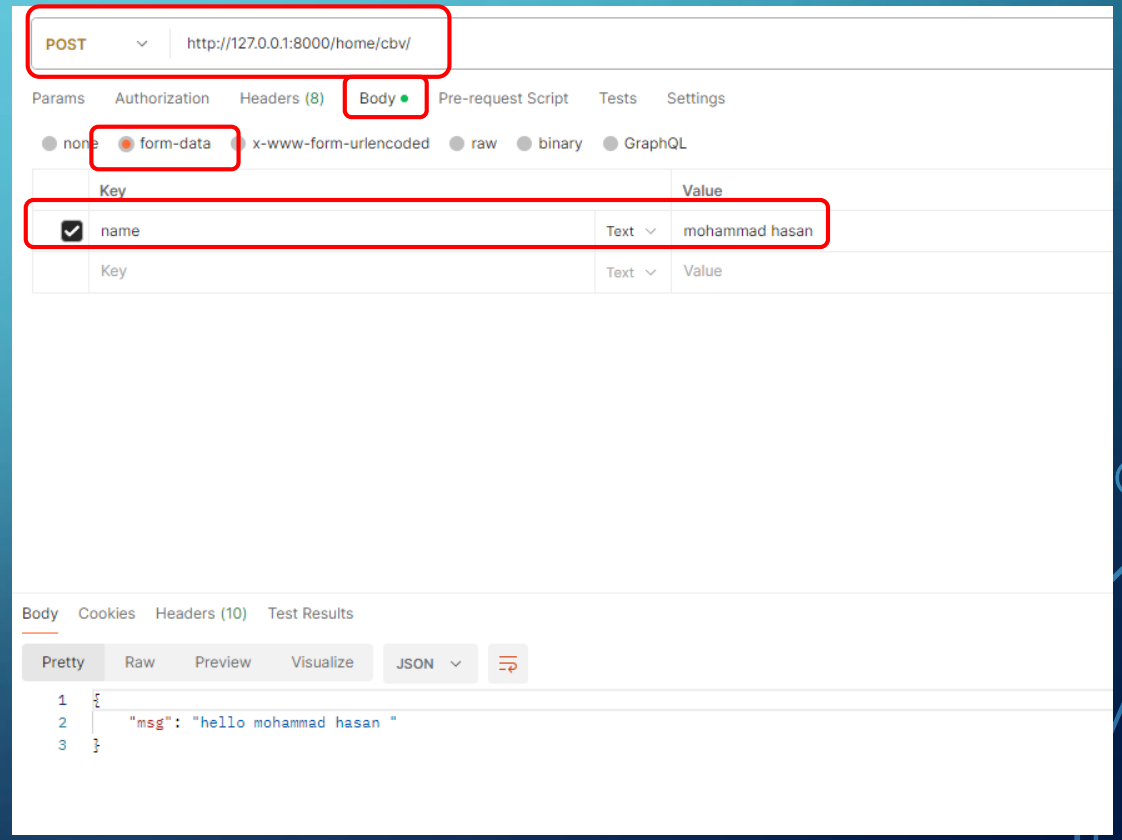
```python
def post(self, request):
    name = request.query_params.get('name')
    return Response({'msg' : f'hello {name}'})
```



POSTMAN-POST JSON

If we want to send form-data or json content we have to use request.data in body part of post man.

```python
def post(self, request):
    if request.query_params:
        name = request.query_params.get('name')
        return Response({'msg' : f'hello {name}'})
    elif request.data:
        name = request.data.get('name')
        return Response({'msg':f'hello {name} '})
```



FINAL CODE

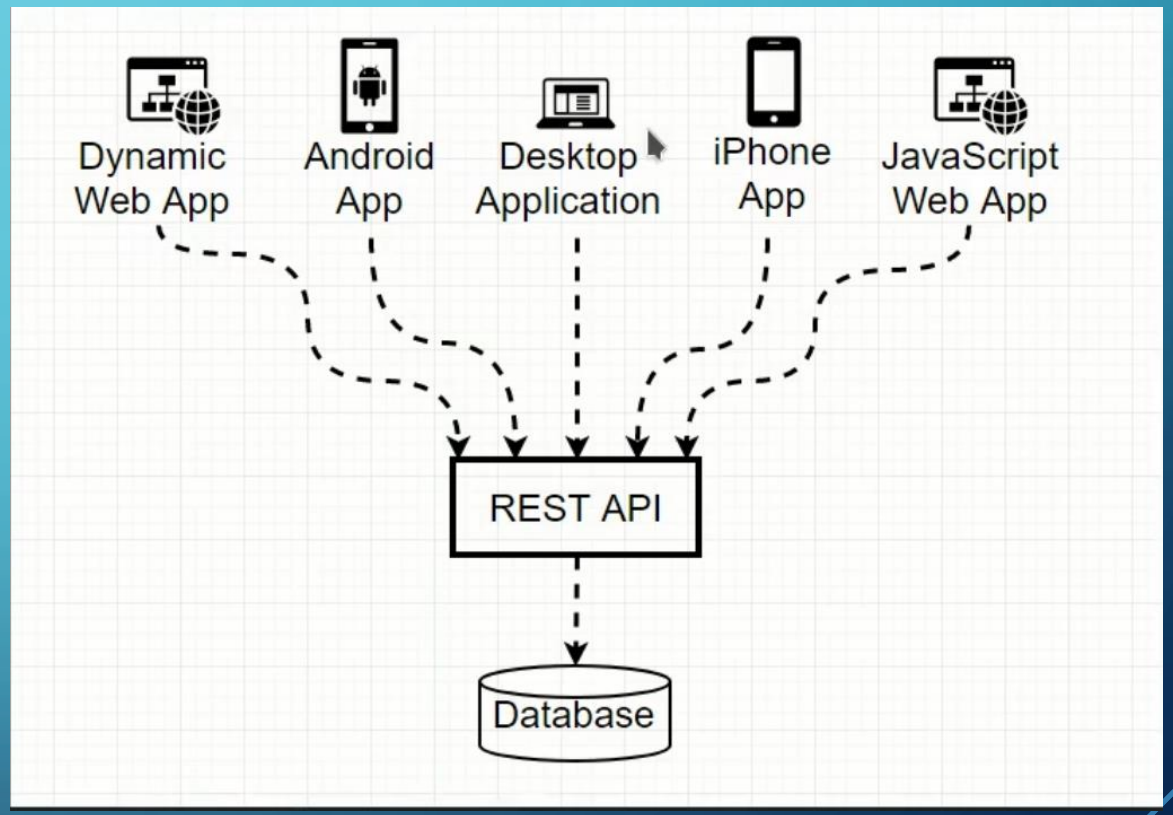
```python
from django.shortcuts import render
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework.views import APIView
# Create your views here.
class HelloWorld(APIView):
    production = [
        {'name': 'phone',
         'price': '1200'},
        {'name': 'laptop',
         'price': 1500}
    ]

    def get(self, request):

        return Response(self.production)

    def post(self, request):
        if request.query_params:
            name = request.query_params.get('name')
            return Response({'msg' : f'hello {name}'})
        elif request.data:
            name = request.data.get('name')
            return Response({'msg':f'hello {name} '})
```

CONCEPTS

- Now we want to say what is Django rest framework and how does it works?
- First of all what is Rest? Rest is an architect sample for programing
- Second of all why do we use Django rest framework?
- i`ll explain in next slide

THAT'S WHY – REASON 1

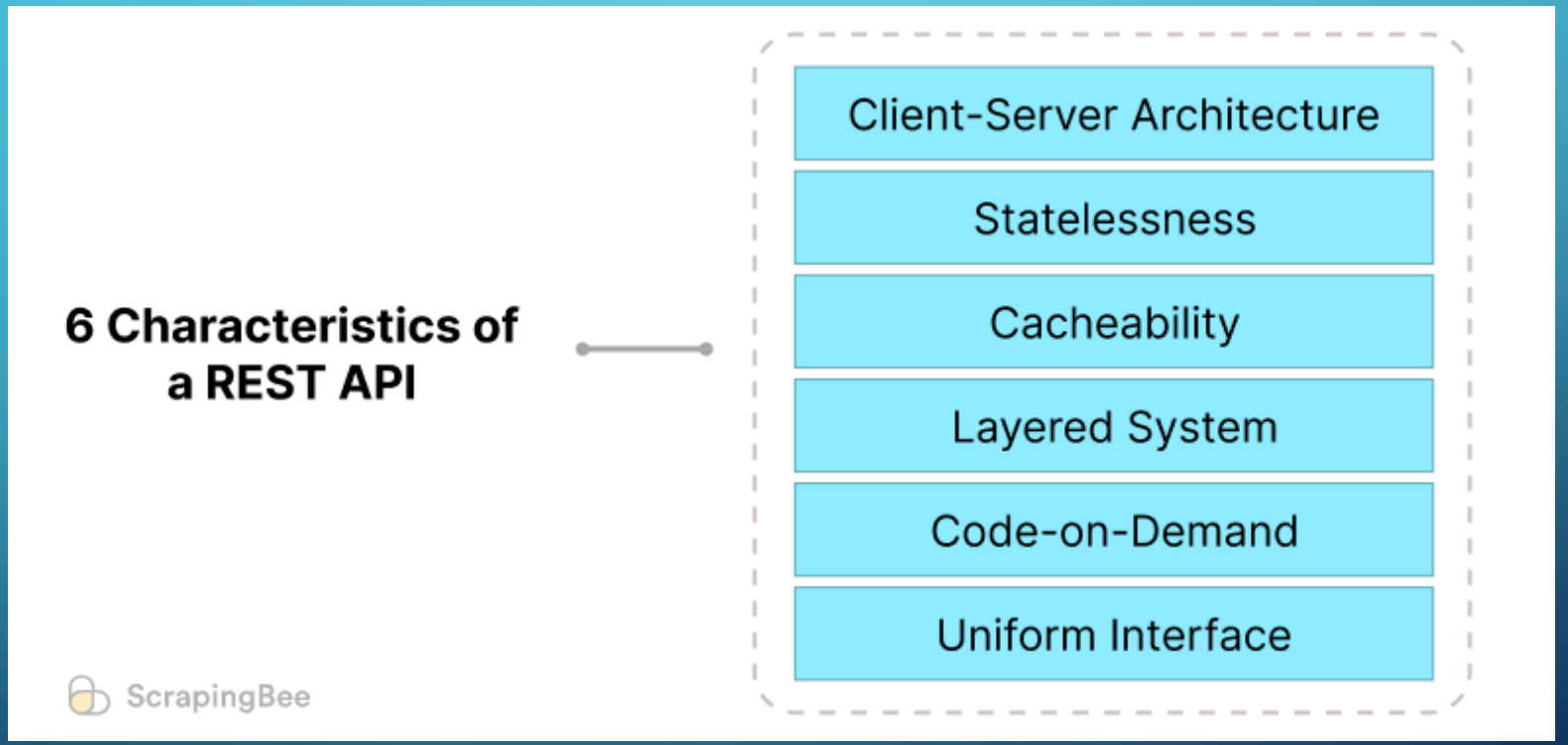
We can create and API and connect it to database then we don`t need to create so many database for each server or app



THAT'S WHY – REASON 2

- REACT !
- Sometimes the front end is react ! And we can`t connect the Django into it, therefore we have to create an API and use it as a connector.

REST PRINCIPLES



CONCEPTS

- What is request.data ? It`s not quite accurate question better question is what is .data
- .data are the data`s which client send it to server in body part of request

- Now request.user… it`s the same as Django we can have access to user information by request.user

GETTING QUERY PARAMS AS ARGUMENT

Example :
http://127.0.0.1:8000/home/ali/ → sending query params as argument
http://127.0.0.1:8000/home/?name=ali → sending query params as query params

Note : query params are not define as argument

If we want to claim the argument we should use following settings below:

```python
#urls.py
path('home/<str:name>/', views.GetName.as_view(), name = 'getname')
#views.py
class GetName(APIView):
    def get(self, request, name):
        return Response({'name' : name})
```

DIFFERENT BETWEEN REQUEST.DATA AND REQUEST.QUERY_PARAMS

- Simplest way to say ? Request.data often use for POST ! Because can`t send secret data`s in url!

SERIALIZERS

- In last slides we create dictionary in views but in real world is differents
  - we have to create models and models turn data`s to dictionary then client could read the information for this process we can use serializers
  - Serializers allow complex data such as queryset and model instances to be converted to native python datatypes that can easily rendered to json or xml
  - Serializers also provide deserilizers

SERIALIZERS

- Step 1 : create a model
- Step 2 : create new file and name it serializers.py
- Step 3 : from rest_framework import serializers on serializers.py
- Step 3.5 : create a class which inheritance serilaziers.Serializer
- Step 4 : create serializer !
- Also you can create custom field in serializer

MODEL TO SERIALIZER

Some field are not available in serializer therefore we have to use alternatives or create custom field in drf documentation the alternative fields are defined please read that !

```python
#models.py
class Person(models.Model):
    name = models.CharField(max_length=50)
    age = models.PositiveIntegerField()
    email = models.EmailField(max_length=254)

#serializer.py
from rest_framework import serializers


class PersonSerializer(serializers.Serializer):
    name = serializers.CharField()
    age = serializers.IntegerField()
    email = serializers.EmailField()
```

USE SERIALIZERS

We create 2 user from our model in admin panel then we want to make their data readable by serializers

Writing instance is optional but I wrote that for better undestanding

```python
#views.py
from .models import Person
from .serializers import PersonSerializer
class GetName(APIView):
    def get(self, request):
        person = Person.objects.all()
        personserialize = PersonSerializer(instance= person, many = True)
        return Response({'data' : personserialize.data})
```

By default serializers except 1 argument when we send it more than 2 we have write : many = True

We can send it as context
Or send it as native python data type

We want data so we have to write it !

RESULT

Note : model data`s are query set and we can use it as response

As context

As python native datatype

Note 2 : we can define id in serializers too !(but usually we don`t do it)

USER REGISTER

- Do the same thing as Django !
- Now we reach at restframework !
  - First : we don`t need GET method cause information going to send therefore we need POST method
- We are going to use default Django registration model

USER REGISTER

- Step 1 : create serializer.py
- Step 2 : create serializers
- … then we are going into views

```python
from rest_framework import serializers

class UserRegisterSerializer(serializers.Serializer):
    username = serializers.CharField(required = True)
    email = serializers.EmailField(required = True)
    password = serializers.CharField(required = True)
```

USER REGISTER

Import !

```
from rest_framework.views import APIView
from rest_framework.response import Response
from .serializers import UserRegisterSerializer
from django.contrib.auth.models import User
```

USER REGISTER VIEW EXPLANATION

We shouldn`t send serializer data ! Because client send it ! (actually we don`t have any data)

```python
class RegisterView(APIView):
    def post(self, request):
        user = UserRegisterSerializer(data = request.POST)
        if user.is_valid():
            User.objects.create_user(
                username= user.validated_data['username'],
                email = user.validated_data['email'],
                password = user.validated_data['password'],
            )
            return Response({'user': user.data})
        return Response(user.errors)
```

Create user

In rest framework we have validated_data instead of cleaned_data

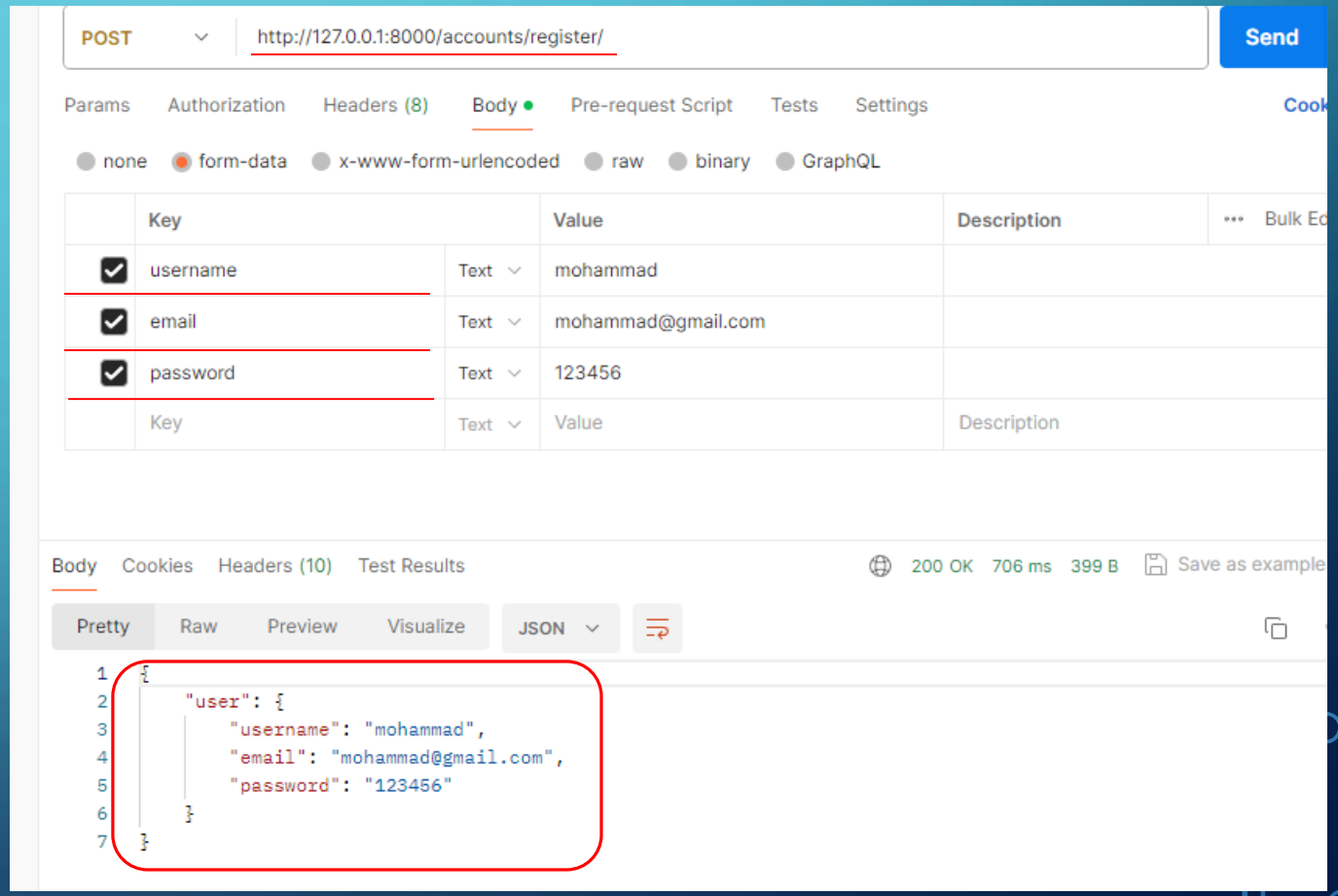
There is default error of serialazors

USER REGISTER TEST

- For testing it we using post man

And now if check admin panel we have new user

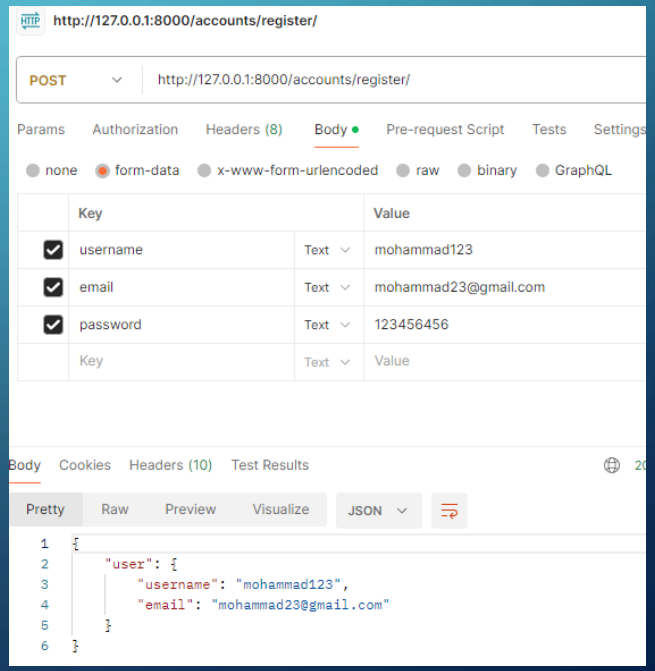
But reading password could be dangerous , therefore we adding write_only into serializer field



USER REGISTER PASSWORD

```python
#serializer.py
password = serializers.CharField(required = True, write_only = True)
```

Result



CUSTOM VALIDATION

- Custom validation separate in 2 method
  - 1- field level validation
  - 2- object level validation
  - 3- validators (not recommended)

FIELD LEVEL VALIDATION

- For using field level validation we have to create a function which start by validate then writing the field that we want to validate it. Then raise validation error

```python
class UserRegisterSerializer(serializers.Serializer):
    username = serializers.CharField(required = True)
    email = serializers.EmailField(required = True)
    password = serializers.CharField(required = True, write_only = True)


    def validate_username(self, value):
        if value == 'admin':
            raise serializers.ValidationError('username cannot be admin')
        return value
```

We have to return the value , value is the value that we want validate it!

CONFIRM PASSWORD

- For validation confirm password we have to use object level validation
  - Step 1 : I create serializer field and name it to password 2
  - Step 2 : create function which name is validate
  - Step 3 : give him data parameter
  - Step 4 : validate It as we want
  - Step 5 : return data

CONFIRM PASSWORD

Data contains all fields in serialaizer so for validate a field we have to slice It !

Different between field level and object level is object level is the data that client send it

```python
class UserRegisterSerializer(serializers.Serializer):
    username = serializers.CharField(required = True)
    email = serializers.EmailField(required = True)
    password = serializers.CharField(required = True, write_only = True)
    password2 = serializers.CharField(required = True, write_only = True)



    def validate(self, data):
        if data['password'] != data['password2']:
            raise serializers.ValidationError('passwords must be match')
        return data
```

VALIDATORS

- A. create a function out of the class
- B. send the function to target serializer field as a tuple

VALIDATORS

```python
def clean_email(value):
    if 'admin' in value:
        raise serializers.ValidationError('admin can`t be in email')
    return value

class UserRegisterSerializer(serializers.Serializer):
    username = serializers.CharField(required = True)
    email = serializers.EmailField(required = True, validators = [clean_email])
    password = serializers.CharField(required = True, write_only = True)
    password2 = serializers.CharField(required = True, write_only = True)
```

MODEL SERIALIZERS

- Model serializers is like MODELFORM in Django
- As we can remember model form in Django used for turning model into from for updating or creating the content in the form
- Therefore we don`t need to create a manual serializer

MODEL SERIALIZER

```python
class UserRegisterSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('username', 'email', 'password')
```

Instead of naming the fields we can use “__all__”
And if we want all fields except one field we can use excludes = (“name_of_field”)

EXTRA KWARGS

- So we use modelserializer therefore we don`t have direct access to fields. So now how we can set write_only for password or give the email field our function ? Answer : → By using extra kwargs

```
class UserRegisterSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ('username', 'email', 'password')
        extra_kwargs = {
            'password': {'write_only' : True},
            'email' : {'validators': [clean_email]}
        }
```

Extra_kwargs is nested dictionary

ADDING FIELD

- For password confirm we need password 2 field but we don`t have it … so what should we do ? We have to create it ! Then have to add it into fields

```python
class UserRegisterSerializer(serializers.ModelSerializer):
    password_2 = serializers.CharField(write_only = True)

    class Meta:
        model = User
        fields = ('username', 'email', 'password', 'password2')
        extra_kwargs = {
            'password': {'write_only' : True},
            'email' : {'validators': [clean_email]}
        }
```

OVERRIDE CLEAN

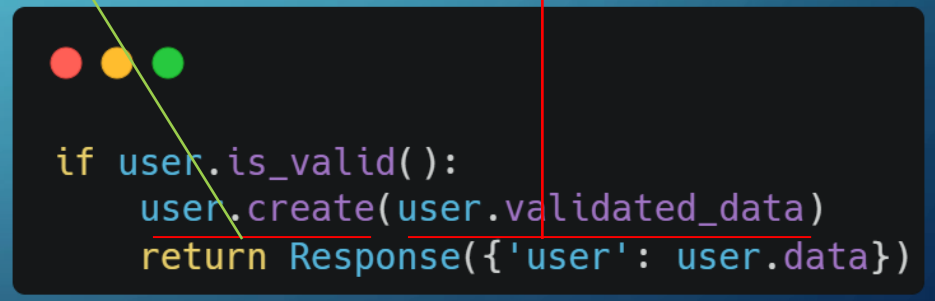
- In this session we want to use .create and .update from model serializers
- In this circumstance there is no need to create user in views ! We just use it in model serializers ! Therefore follow the below commands:
- 1- create a function in seriliazer class which call it create
- 2- Copy views.py create part into serializer
- 3- call create in views.py

.CREATE

This is the change that we do it on Views.py

Giving create method our validated data from serializer

Our serializer name + calling create method



.CREATE

This 2 start create connection between validated_data and User , which means gives validated_data information into User class

Create method

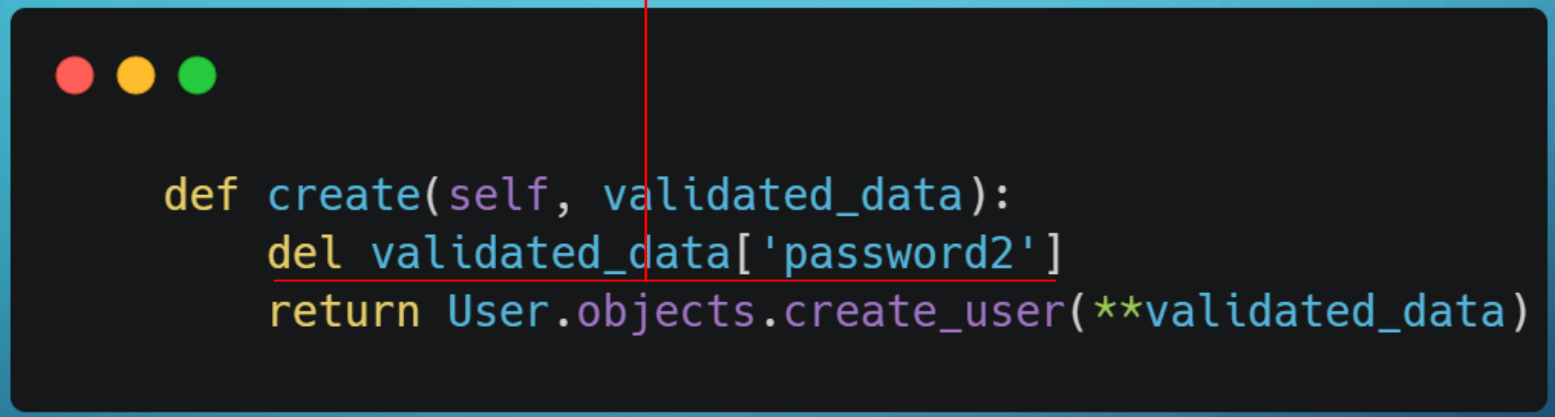
The validated data which user sent us by Request.POST

```
def create(self, validated_data):
    return User.objects.create_user(**validated_data)
```

But we have a problem here ! User don`t get password 2 from us but validated data have the password 2 field so what should we do ? → answer : we have to delete password 2 field

.CREATE

Therefore we delete password 2 in create user but it`s going to be validated so don`t worry!



```
def create(self, validated_data):
    del validated_data['password2']
    return User.objects.create_user(**validated_data)
```

STATUS CODES

- Requirement : for knowing status codes you can read my article
  - https://virgool.io/@Niklaus/http-status-codes-fo0o0tvji674
- Now here we want to use the status codes in response method
- There is 2 way to make status codes readable :
- A. Writing manual Status code (not recommended)
- B. using rest framework to send status code

STATUS CODE

- For sending manual status code just do like this !

```python
class RegisterView(APIView):
    def post(self, request):
        user = UserRegisterSerializer(data = request.POST)
        if user.is_valid():
            user.create(user.validated_data)
            return Response({'user': user.data})
        return Response(user.errors, status=500)
```

STATUS CODE

- Now by using rest framework
- A. Import status from rest_framework
- B. Write it in response function and chose the status code that you want

```python
from rest_framework import status


class RegisterView(APIView):
    def post(self, request):
        user = UserRegisterSerializer(data = request.POST)
        if user.is_valid():
            user.create(user.validated_data, status = status.HTTP_201_CREATED)
            return Response({'user': user.data})
        return Response(user.errors, status= status.HTTP_400_BAD_REQUEST)
```

Note : use the proper status code !

AUTHENTICATION

- Authentication uses for identify the user for changing user access have to use changing permissions.
- When we authenticate user 2 method going to set for us
  - 1- request.user → which save user information
  - 2- request.auth → which save user authentication information

AUTHENTICATION

- We have 3 ways to introduce authentication in our Django project
- 1- in setting.py → in this case the authentication introduce to all views, and all of the project
- 2- in views if we use CBV → in this case we introduce authentication in just one class by using authentication_classes property
- 3- in views if we use FBV → in this case we use authentication_classes as decorator

UNAUTHORIZED AND FORBIDDEN

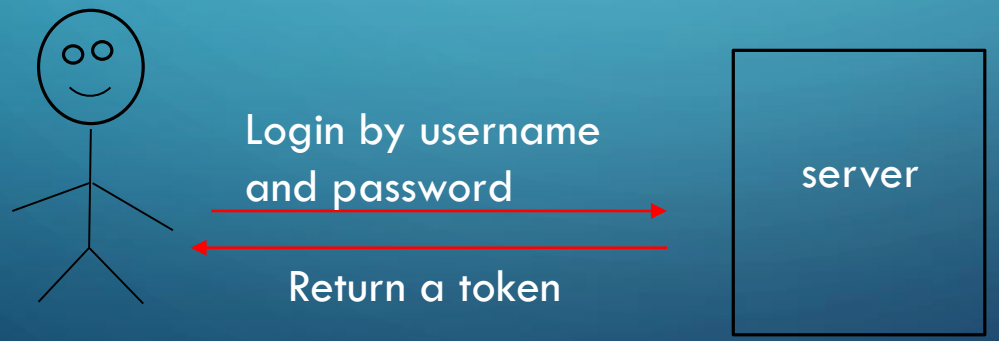
- If the user is unauthorized we have to use 401 status code
- And if it`s forbidden which means permission denied we have to use 403 status code

TYPES OF AUTHENTICATION

- 1- Basic Authentication → kind of useless ! Just for test !
- 2- Token authentication → which authenticate users by token
- 3- session authentication
- 4- Remote use authentication
- 5- third party packages:
  - JWT, Django Guardian and etc...

TOKEN AUTHENTICATION

- For authenticating user , user send us the token !
- Many of views which we created is login required therefore we can`t ask user to login again and again. Thus we use token authentication which means the browser send us the token and prove that the user is authenticated.



Now user use the token when he is connected in our server

TOKEN AUTHENTICATION

- Step A. introduce authtoken in installapps in setting.py
- Step B. Migrate !

```
#setting.py on installapps
'rest_framework.authtoken',

#on terminal
python manage.py migrate
```

TOKEN AUTHENTICATION

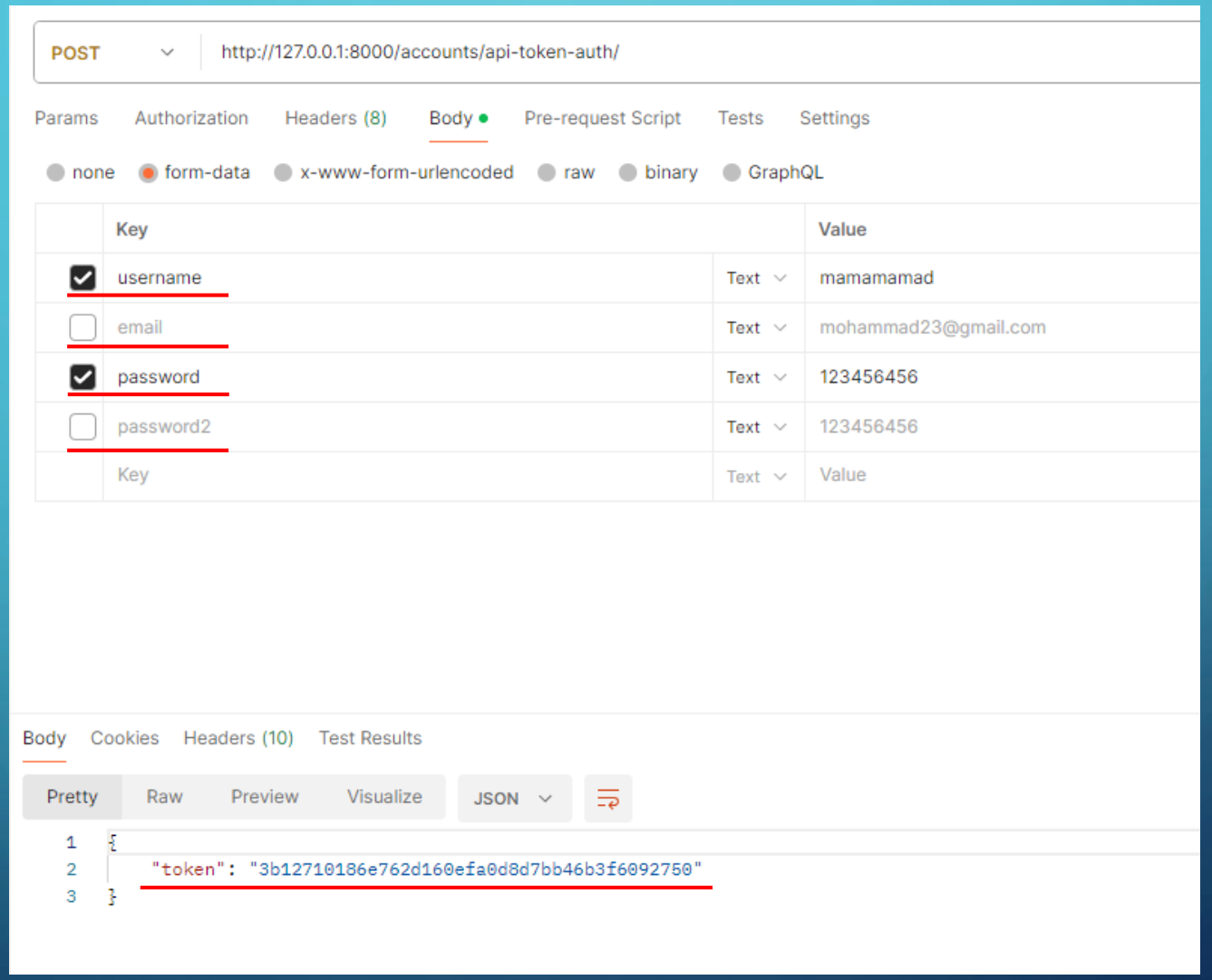
- There is some different ways to create token for users
- 1- manual
- 2- by using signals
- 3- by using for loop ! (by using get or create method)
- 4- use authtoken views
- 5- use obtainauthtoken
- 6- in Django admin
- 7- by using python manage.py
- We are going to use number 4

BY USING VIEWS IN URLS

- Step A. import views from .authtoken as auth_token
- Step B. Create a new url and use auth_token to call obtain_auth_token

```
from django.urls import path
from . import views
from rest_framework.authtoken import views as auth_token

app_name = 'accounts'

urlpatterns = [
    path('register/', views.RegisterView.as_view(), name = 'register'),
    path('api-token-auth/', auth_token.obtain_auth_token)

]
```

RESULT


POST
http://127.0.0.1:8000/accounts/api-token-auth/
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
username
Text
mamamamad
email
Text
mohammad23@gmail.com
password
Text
123456456
password2
Text
123456456
Key
Text
Value
Body
Cookies
Headers (10)
Test Results
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "token": "3b12710186e762d160efa0d8d7bb46b3f6092750"
3 }

DEFINE THE AUTHENTICATION

- Now we want to introduce authentication by using first way which means , define it in setting.py

```
#setting.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ]
}
```

Our authentication type

Define the authentication

PERMISSIONS

- Permission with authentications and throttling can limit users access.
- If a user don`t have permission he/she got 2 errors
  - 1- permission denied
  - 2- not authenticated

DEFINING PERMISSION

- Same as authentication there is 3 ways to define permission (introduce it) into Django
  - A. by define permission_classes in setting.py which effect all on views
  - B. by define permission_classes in views (CBV)
  - C. by define perimission_classes as decorator (FBV)

TYPE OF PERMISSIONS

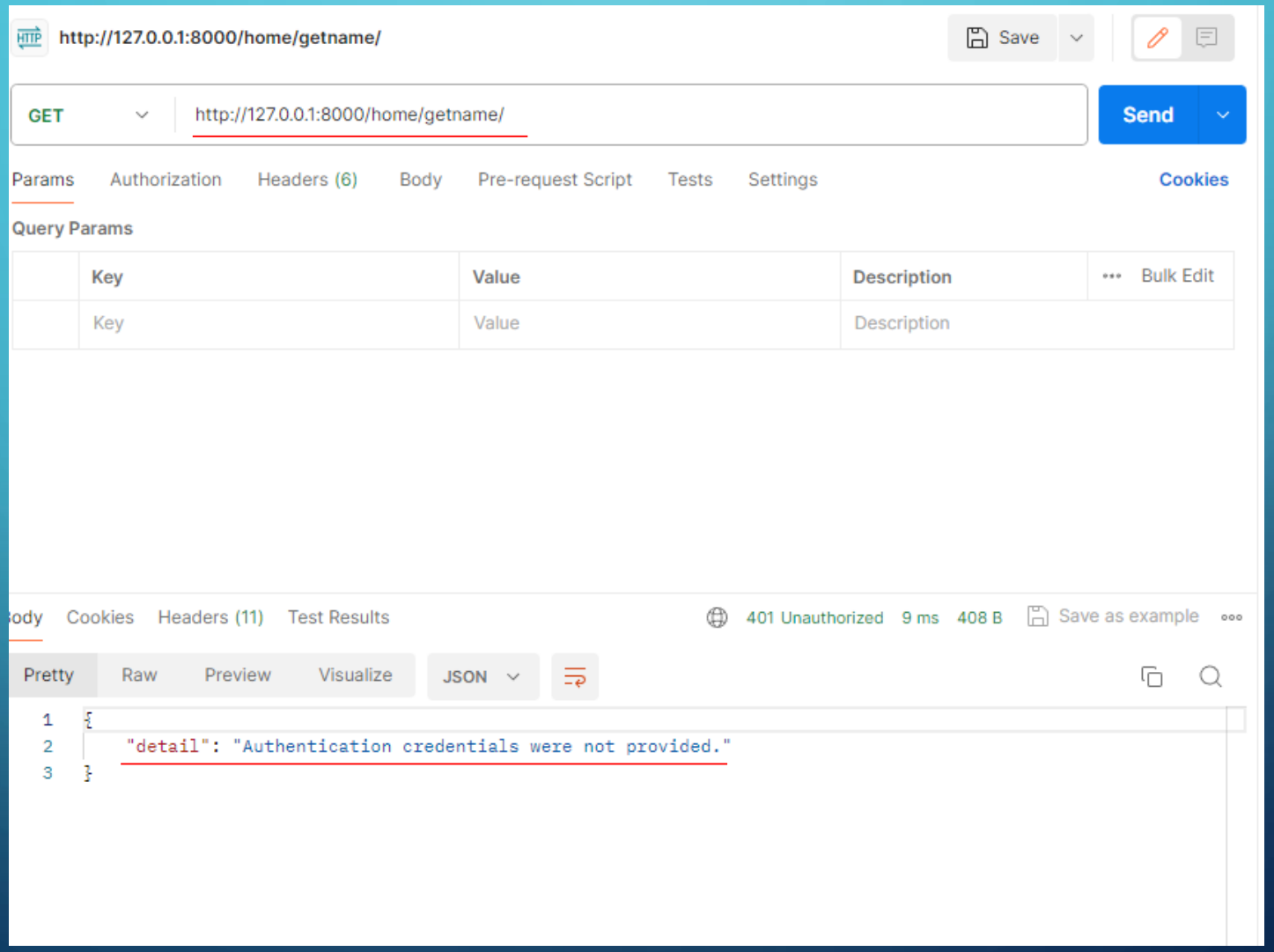
- A. allow any: which let all users have access to this view rather user authenticated or not (this is default permission)
- B. Is authenticated: if user is authenticated can have access
- C. is AdminUser : user has to authenticate and staff (admin)
- D. is AuthenticatedOrReadOnly: if user authenticated can POST but if user didn`t authenticate just Read Only which means safe methods → HEAD, GET, OPTION
- F. Custom permission : which we can create the our designed permission

DEFINING PERMISSION IN VIEWS

- Step A: import our method from rest_framework.permission
- Step B: create class variable
- Step C: give the class variable the method

The authentication method

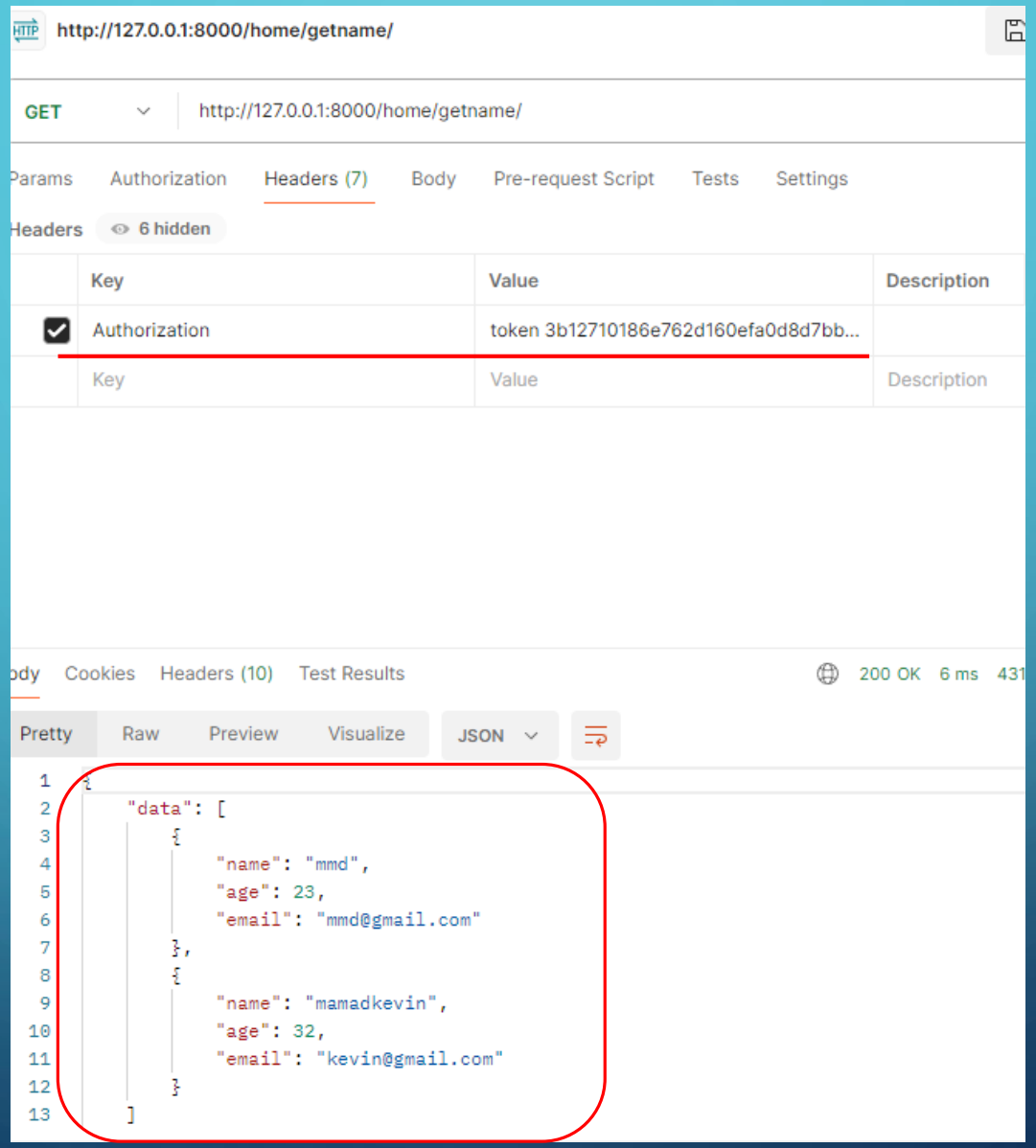
RESULT


http://127.0.0.1:8000/home/getname/
Save
GET
http://127.0.0.1:8000/home/getname/
Send
Params
Authorization
Headers (6)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
ody
Cookies
Headers (11)
Test Results
401 Unauthorized
9 ms
408 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
1 {
2 "detail": "Authentication credentials were not provided."
3 }

AUTHORIZING

- For authentication by token and read the data we have to send token in header
- So we extract our token from admin panel
- Then put it in header part with this order
- Key = authorization value : token token_number

RESULT


http://127.0.0.1:8000/home/getname/
GET
http://127.0.0.1:8000/home/getname/
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Headers
6 hidden
Key
Value
Description
Authorization
token 3b12710186e762d160efa0d8d7bb...
Key
Value
Description
ody
Cookies
Headers (10)
Test Results
200 OK
6 ms
431
Pretty
Raw
Preview
Visualize
JSON
{
"data": [
{
"name": "mmd",
"age": 23,
"email": "mmd@gmail.com"
},
{
"name": "mamadkevin",
"age": 32,
"email": "kevin@gmail.com"
}
]

QUESTION- ANSWER

- Want to create a view which get the question and return answer (by users)
- Step A : create new model in models.py and name it question
- Step B: create new model in models.py and name it answer
- Step C : register it on admin panel, makemigrations and migrate
- Step D : Craete Serializer for it cause we want to create API
- Step E : create a class to do CRUD → (create , read, update , delete)

STEP A

Import User model too !
Create related name for user too (questions)

```
class Question(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=150)
    slug = models.SlugField(max_lenght = 150)
    body = models.TextField()
    created = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.user} - {self.title}'
```

STEP B

```
class Answer(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE, related_name = 'answers')
    question = models.ForeignKey(Question, on_delete=models.CASCADE, related_name = 'answers')
    body = models.TextField()
    created = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f'{self.user} - {self.question.title}'
```

STEP C

```
#on admin.py
admin.site.register(Question)
admin.site.register(Answer)


#on terminal
python manage.py makemigrations
python manage.py migrate
```

STEP D

```python
#serializer.py
class QuestionSerializer(serializers.ModelSerializer):
    class Meta:
        model = Question
        fields = '__all__'


class AnswerSerializer(serializers.ModelSerializer):
    class Meta:
        model = Answer
        fields = '__all__'
```

STEP E

Note : we can create one view for all of operations in CRUD

R Read method

C Create method

U Update method

D Delete method

CRUD

```python
class QAView(APIView):
    def get(self, request):
        pass

    def post(self, request):
        pass

    def put (self, request, id):
        pass

    def delete(self, request, id):
        pass
```

POST PUT DELETE

- POST → for create
- PUT → put for update
- DELETE → for delete !
- We start from POST

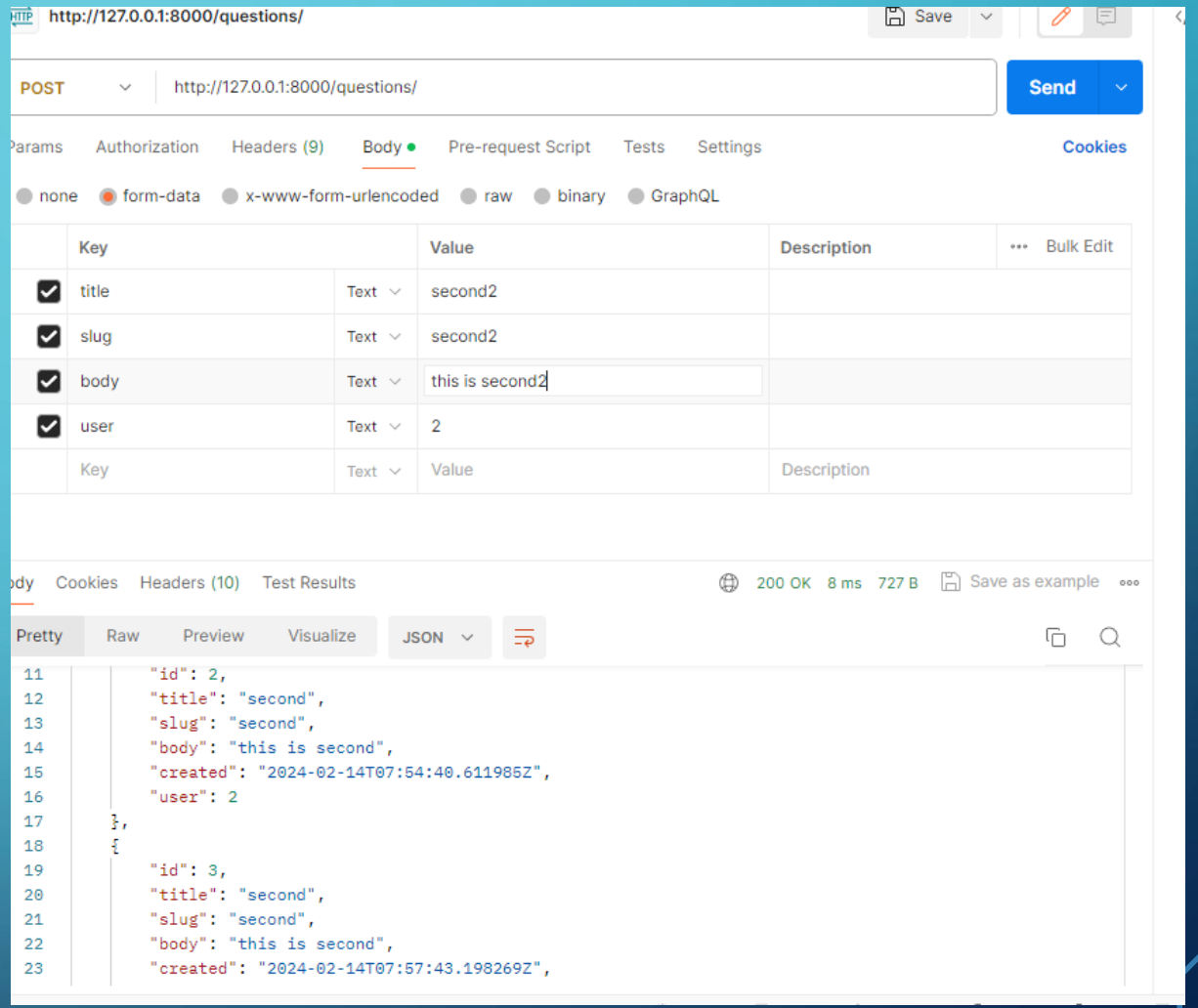
POST

Sending the data to our serializer

```
    def post(self, request):
        question_serializer = QuestionSerializer(data = request.POST)
        if question_serializer.is_valid():
            question_serializer.save()
            return Response(data = question_serializer.data , status= status.HTTP_201_CREATED)
        return Response(question_serializer.errors, status= status.HTTP_400_BAD_REQUEST)
```

RESULT

But I create before taking screenshot! My bad ☺



PUT

- Here we have different task !
- Attention: for updating a question we have to get the id from database
  - But there is no id in urls ! So we need to create 2 urls with same view

```
    path('questions/', views.QAView.as_view(), name = 'questions'),
    path('questions/<int:id>', views.QAView.as_view(), name = 'questionsid'),
```

PUT

Partial true means may update one field not the all fields

Giving the question information to serializer

That's why we needs 2 urls

```
    def put (self, request, id):
        question = Question.objects.get(id = id)
        question_serializer = QuestionSerializer(instance= question, data = request.POST, partial = True)
        if question_serializer.is_valid():
            question_serializer.save()
            return Response(question_serializer.data, status= status.HTTP_200_OK)
        return Response(question_serializer.errors, status= status.HTTP_400_BAD_REQUEST)
```

DELETE

Everything is clear !

```
    def delete(self, request, id):
        question = Question.objects.get(id = id)
        question.delete()
        return Response({'message': 'question deleted'}, status= status.HTTP_200_OK)
```

METHOD FIELD

- Now we want to separate our method in different classes and then we want to show the answers through the questions.
- In documentation we are going to use miscellaneous fields which is very useful!
- In miscellaneous fields we have a part which call it serializer method field which let you connect a serializer field to method and whatever you return on that method will connect to that field

METHOD FIELD

Create a field to connect it to method

```python
class QuestionSerializer(serializers.ModelSerializer):
    answers = serializers.SerializerMethodField

    class Meta:
        model = Question
        fields = '__all__'

    def answers_all(self, obj):
        result = obj.answers.all()
        return AnswerSerializer(instance= result, many = True).data
```

Because we are in QuestionSerializer , obj are all questions that user can read

Using related name in our models

Cause we got all answers by using related name now we can call answer serialzier

HOW TO USE METHOD FIELD?

- 2 Ways
  - A. use signature , give the field the method name
  - B. use get_field
- We are going to use method B

USING METHOD FIELDS

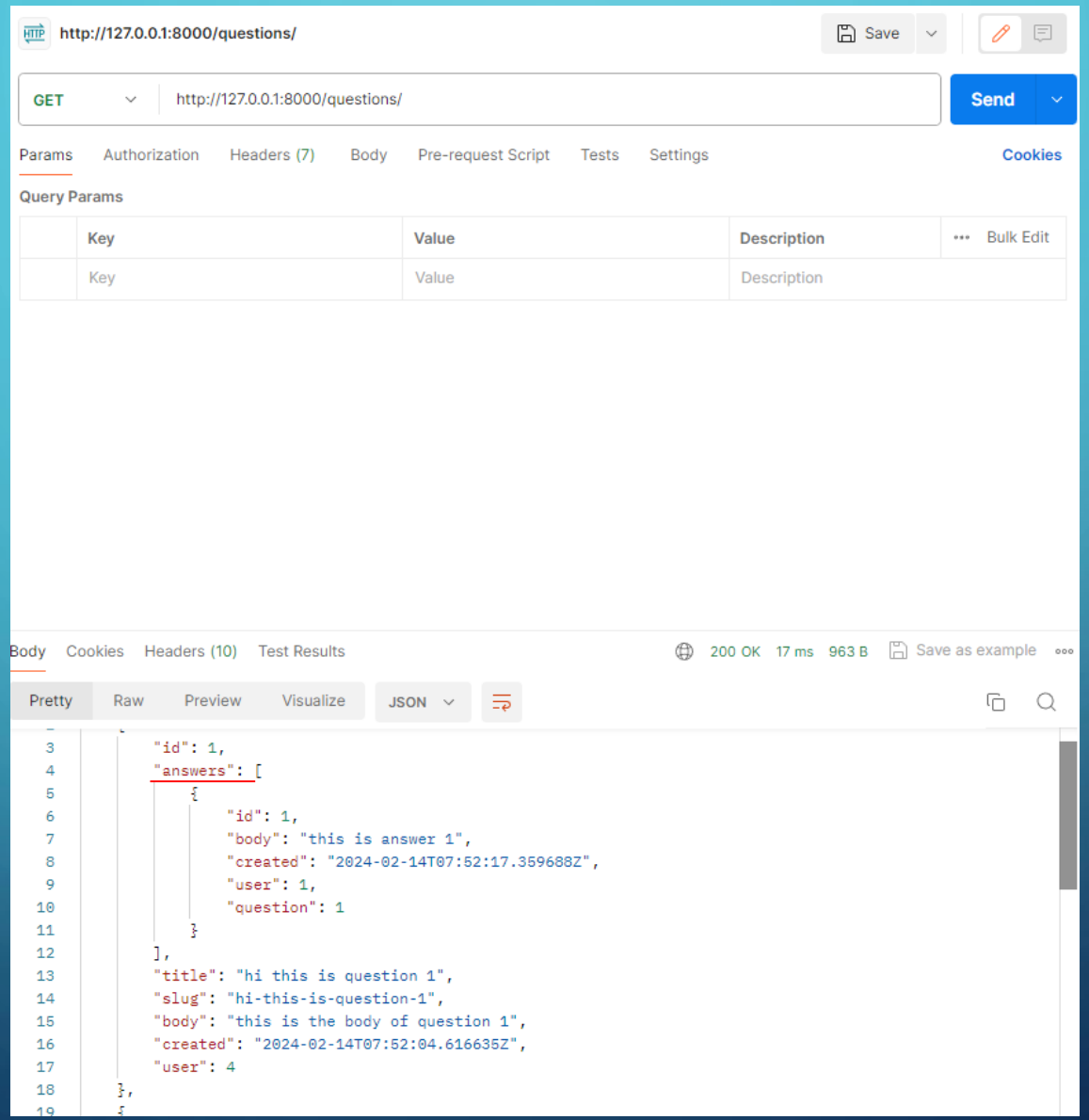
I changed my method name to get_<field_name> therefore automatically this method sends data to my field

Thus we connect a field to method

Forgot the parentheses

```python
class QuestionSerializer(serializers.ModelSerializer):
    answers = serializers.SerializerMethodField

    class Meta:
        model = Question
        fields = '__all__'

    def get_answers(self, obj):
        result = obj.answers.all()
        return AnswerSerializer(instance= result, many = True).data
```

RESULT


http://127.0.0.1:8000/questions/
Save
GET
http://127.0.0.1:8000/questions/
Send
Params
Authorization
Headers (7)
Body
Pre-request Script
Tests
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Key
Value
Description
Body
Cookies
Headers (10)
Test Results
200 OK
17 ms
963 B
Save as example
Pretty
Raw
Preview
Visualize
JSON
"id": 1,
"answers": [
{
"id": 1,
"body": "this is answer 1",
"created": "2024-02-14T07:52:17.359688Z",
"user": 1,
"question": 1
}
],
"title": "hi this is question 1",
"slug": "hi-this-is-question-1",
"body": "this is the body of question 1",
"created": "2024-02-14T07:52:04.616635Z",
"user": 4
},

SEPARATING VIEWS

- Now we want to separating our views (because in future may we have problem with permissions or etc…)

```python
class QuestionReadView(APIView):
    def get(self, request):
        question = Question.objects.all()
        question_serializer = QuestionSerializer(instance= question , many = True)
        return Response(question_serializer.data, status= status.HTTP_200_OK)



class QusetionCreateView(APIView):

    def post(self, request):
        question_serializer = QuestionSerializer(data = request.POST)
        if question_serializer.is_valid():
            question_serializer.save()
            return Response(data = question_serializer.data , status= status.HTTP_201_CREATED)
        return Response(question_serializer.errors, status= status.HTTP_400_BAD_REQUEST)

class QuestionUpdateView(APIView):
    def put (self, request, id):
        question = Question.objects.get(id = id)
        question_serializer = QuestionSerializer(instance= question, data = request.POST, partial = True)
        if question_serializer.is_valid():
            question_serializer.save()
            return Response(question_serializer.data, status= status.HTTP_200_OK)
        return Response(question_serializer.errors, status= status.HTTP_400_BAD_REQUEST)

class QuestionDeleteView(APIView):
    def delete(self, request, id):
        question = Question.objects.get(id = id)
        question.delete()
        return Response({'message': 'question deleted'}, status= status.HTTP_200_OK)
```

CUSTOM PERMISSION

- Sometimes we need to create a permission ! That's why we use custom permission
- For creating custom permission we have to create a class which in heritage from BasePermission on that class we can have 2 methods :
  - Has_permission(self, request, view) → which check that user can come into the view or not
  - Has_object_permission(self, request, view, obj) → which check that if user come into the view can change anything or not

CUSTOM PERMISSION

- Has_permission(self, request, view) → before user come into view
- Has_object_permission(self, request, view, obj) → after user come into view

REMINDER

- As you remember we had a permission_classes[] now we have to use it here on our views on POST PUT DELETE view.

```python
class QuestionUpdateView(APIView):
    permission_classes = [IsAuthenticated,]
    def put (self, request, id):
        question = Question.objects.get(id = id)
        question_serializer = QuestionSerializer(instance= question, data = request.POST, partial = True)
        if question_serializer.is_valid():
            question_serializer.save()
            return Response(question_serializer.data, status= status.HTTP_200_OK)
        return Response(question_serializer.errors, status= status.HTTP_400_BAD_REQUEST)
```

But There is one problem!

PROBLEM

- The problem is the authenticated user can change every question not just his mine therefore we have to use custom permission.
- Therefore we have to create a permission which just allow the owner (author) of question to let change the question

CREATE CUSTOM PERMISSION

- Step A : create a file which call it permissions in base DIR
- Step B : from rest_freamwork.permission import BasePermission
- Step C : Craete a class which inheritance from BasePermission
- Step D : If you remember we had 2 methods for basepermission in this place user authenticated thus he have access to view so we need the second method

CREATE CUSTOM PERMISSION

The object that we want to change it in our example object is our question

```python
from rest_framework.permissions import BasePermission



class ChangeOrReadOnly(BasePermission):
    def has_object_permission(self, request, view, obj):
        pass
```

SAFE METHODS

- Safe methods are the method which they are safe !
  - Safe methods are : HEAD , OPTION , GET
- Now we have to check if it is safe method or not

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


class ChangeOrReadOnly(BasePermission):


    def has_object_permission(self, request, view, obj):
        if request.method in SAFE_METHODS:
            return True
        return obj.user == request.user
```

This is our question class

USING CUSTOM PERMISSION

- Step A: I have to import it on my views
- Step B : I have to define it permission_classes
- Step C: I have to use check_object_permission
- Step A+ : also we can override a message

```python
class ChangeOrReadOnly(BasePermission):
    message = 'You are not the author'

    def has_object_permission(self, request, view, obj):
        if request.method in SAFE_METHODS:
            return True
        return obj.user == request.user
```

USING CUSTOM PERMISSIOM

```python
#step A
from permission import ChangeOrReadOnly
#step B
class QuestionUpdateView(APIView):
    permission_classes = [IsAuthenticated, ChangeOrReadOnly]
```

But It won`t work because APIView class don`t check the permission automatically therefore we need to use a method which call it check_object_permission(request, obj)

USING CUSTOM PERMISSION

- STEP C:

```
class QuestionUpdateView(APIView):
    permission_classes = [IsAuthenticated, ChangeOrReadOnly]
    def put (self, request, id):
        question = Question.objects.get(id = id)
        self.check_object_permissions(request, question)
```

This method make the APIView to check the custom permissions

This our obj (in permission class and here this is our obj that we want to give it a permission)

CUSTOM PERMISSION

- Also we can create IsAuthenticated in custom permission

```python
class ChangeOrReadOnly(BasePermission):
    message = 'You are not the author'

    def has_permission(self, request, view):
        return request.user.is_authenticated and request.user

    def has_object_permission(self, request, view, obj):
        if request.method in SAFE_METHODS:
            return True
        return obj.user == request.user
```

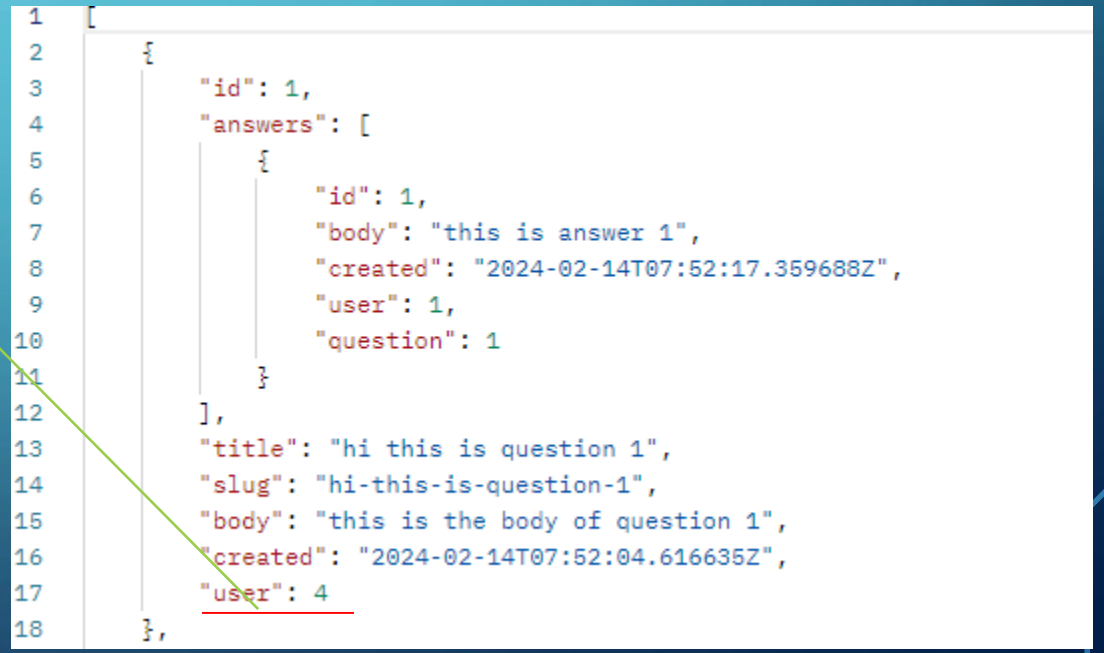
User has not be a None

Then give this permission to view (PUT and delete).

SERIALIZER RELATIONS

- Serializer relations define that how we can show the relation in a result

This User is author of question
We create it on model.py
It return us the primary key which is id
Now we want to change it to email , username or etc



TYPES OF SERALIZER RELATION

- 1-string related field → which return us everything in __str__ (we used User class for authentication this class return username in __str__) therefore it shows us username
- 2- primary key related field → which return us primary key (the default behavior is like that) [this is just like string field don`t need to learn]
- 3- hyper link related field → which return us a specific url
- 4- slug related field → which we can chose what field do we want
- 5- custom relational field → it`s custom !

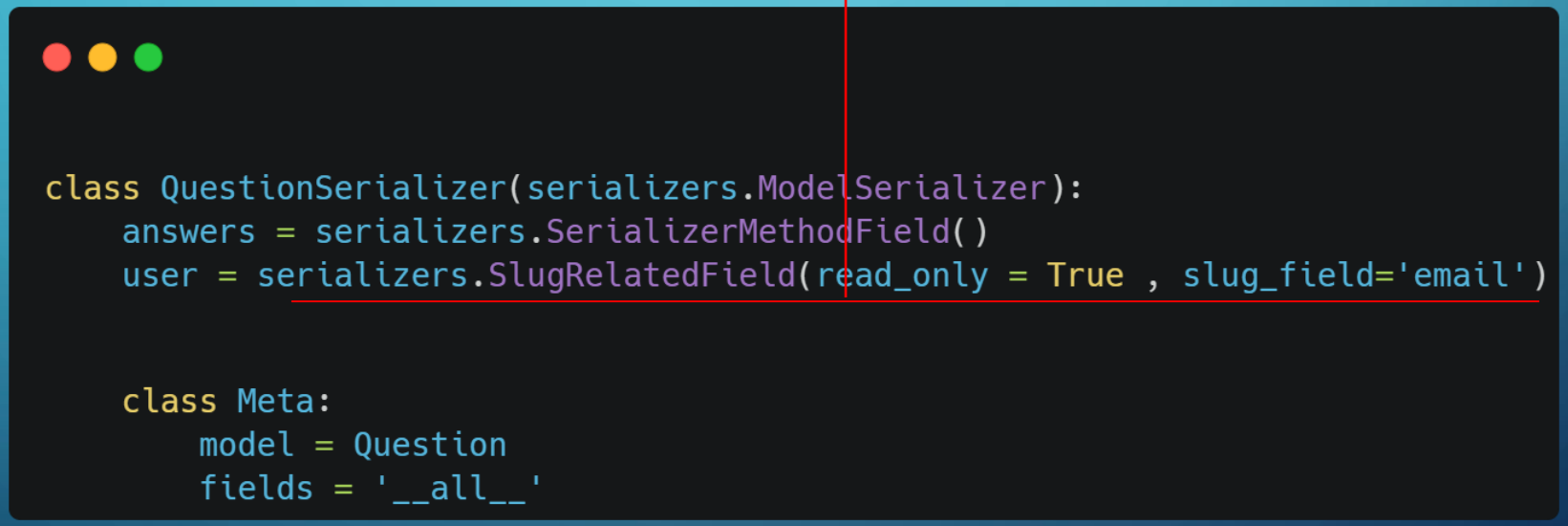
STRING RELATED FIELD

```python
class QuestionSerializer(serializers.ModelSerializer):
    answers = serializers.SerializerMethodField()
    user = serializers.StringRelatedField(read_only = True)


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

Read only on true which limit the user access , and user can`t change the relation

SLUG FIELD

Which I say that to shows email from user informations



```python
class QuestionSerializer(serializers.ModelSerializer):
    answers = serializers.SerializerMethodField()
    user = serializers.SlugRelatedField(read_only = True , slug_field='email')


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

CUSTOM RELATION FIELD

- Step A : in your app create a .py file (u can name it as u want)
- Step B: import serializer
- Step C: inheritance .relationfield
- Step D: to _reperensetive method

CRF

In crf.py file , value is our User model

```
from rest_framework import serializers


class UserNameEmailRelationField(serializers.RelatedField):
    def to_representation(self, value):
        return f'{value.username} - {value.email}'
```

USE IT IN SERIALIZER

```
from .crf import UserNameEmailRelationField

class QuestionSerializer(serializers.ModelSerializer):
    answers = serializers.SerializerMethodField()
    user = UserNameEmailRelationField(read_only = True)


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

VIEWSETS

- Viewsets allows us to put many operation (process) in one class
- For each class we create a url but when project develop so much and get bigger it`s gonna be difficult, Thus we use Viewsets
- Viewset allow us to do 5-6 process in 1 class therefore we need 1 urls for 5-6 process
- Viewset is a type from CBV but it doesn`t have .get , .post and etc… instead provides process as method like .list , .create and etc…

VIEWSETS

- Each viewset has 6 actions
  - 1- list : for listing all data`s (reading all data`s)
  - 2- create: for creating new object
  - 3- retrieve: for reading 1 object
  - 4- update : for updating an object but if it doesn`t exist update create it for us
  - 5-partial_update : for updating a part of object
  - 6-destroy : for deleteing

VIEWSETS

- Now we want to use viewset in accounts app for managing users
  - A. classes have to in heritage from viewset.viewset
  - B. we have to import viewset !

```
from rest_framework import viewsets

class UserViewSet(viewsets.ViewSet):
    pass
```

WHEN WE USE VIEWSET

- We use viewset when code Is not complicated. If it`s complicated use APIVIEW

For show all user

For show 1 user

For update a part of user data

For delete user

```python
class UserViewSet(viewsets.ViewSet):
    def list(self, request):
        pass

    def retrieve(self, request, pk = None):
        pass

    def partial_update(self, request, pk = None):
        pass

    def destroy(self, request, pk = None):
        pass
```

WHY DO WE USE VIEWSETS?

- 2 reason
  - A. can put many methods on 1 class
  - B. using routers that makes need less to urls
- Negative point:
  - Some views created by routers automatically which make the program hard to read

CREATE A VIEWSET

- For createing viewset
  - 1- create a serializer for it
  - 2- set the permission you want (I set IsAuthenticated)
  - 3- create the list view

CREATE SERIALIZER FOR VIEWSET

```python
class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = '__all__'
```

I think everything is clear !

CREATE VIEWSET

I create a class variable from User class cause it`s going to repeat many times

Creating a variable and giving that our serialzier which using our queryset(User class instance)

```python
from .serializers import UserRegisterSerializer, UserSerializer
from rest_framework.permissions import IsAuthenticated
from django.contrib.auth.models import User


class UserViewSet(viewsets.ViewSet):
    permission_classes = [IsAuthenticated,]
    queryset = User.objects.all()

    def list(self, request):
        user_serializer = UserSerializer(instance= self.queryset, many = True)
        return Response(data = user_serializer.data)
```

CREATE URL FOR VIEWSET

- Default urls can`t connect to viewset cause viewsets has different methods
- For viewset we use routers
- Routers create url automatically for us and suitable for viewsets

HOW ROUTERS CREATE URL ?

URL Style	HTTP Method	Action	URL Name
{prefix}/	GET	list	{basename}-list
	POST	create	
{prefix}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=False)` decorated method	{basename}-{url_name}
{prefix}/{lookup}/	GET	retrieve	{basename}-detail
	PUT	update	
	PATCH	partial_update	
	DELETE	destroy	
{prefix}/{lookup}/{url_path}/	GET, or as specified by `methods` argument	`@action(detail=True)` decorated method	{basename}-{url_name}

Lookup means primary key

CREATE ROUTERS

- A. import routers
- B. create instance from simple router
- E. generate urls by the instance

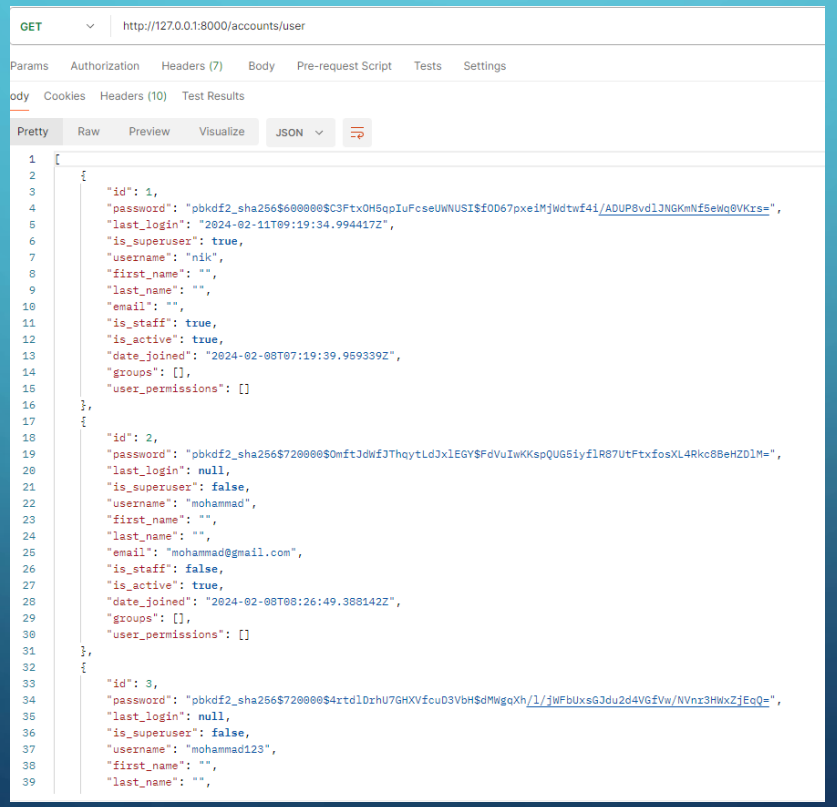
```python
from django.urls import path
from . import views
from rest_framework.authtoken import views as auth_token
from rest_framework import routers

app_name = 'accounts'

urlpatterns = [
    path('register/', views.RegisterView.as_view(), name = 'register'),
    path('api-token-auth/', auth_token.obtain_auth_token)

]

router = routers.SimpleRouter()
router.register('user', views.UserViewSet)

urlpatterns += router.urls
```

Create instance and give It the url + our viewset

Add it to urls patterns our urls from router

ROUTERS

- Now according to slide 117th if we call our path by GET method it has to show us our member



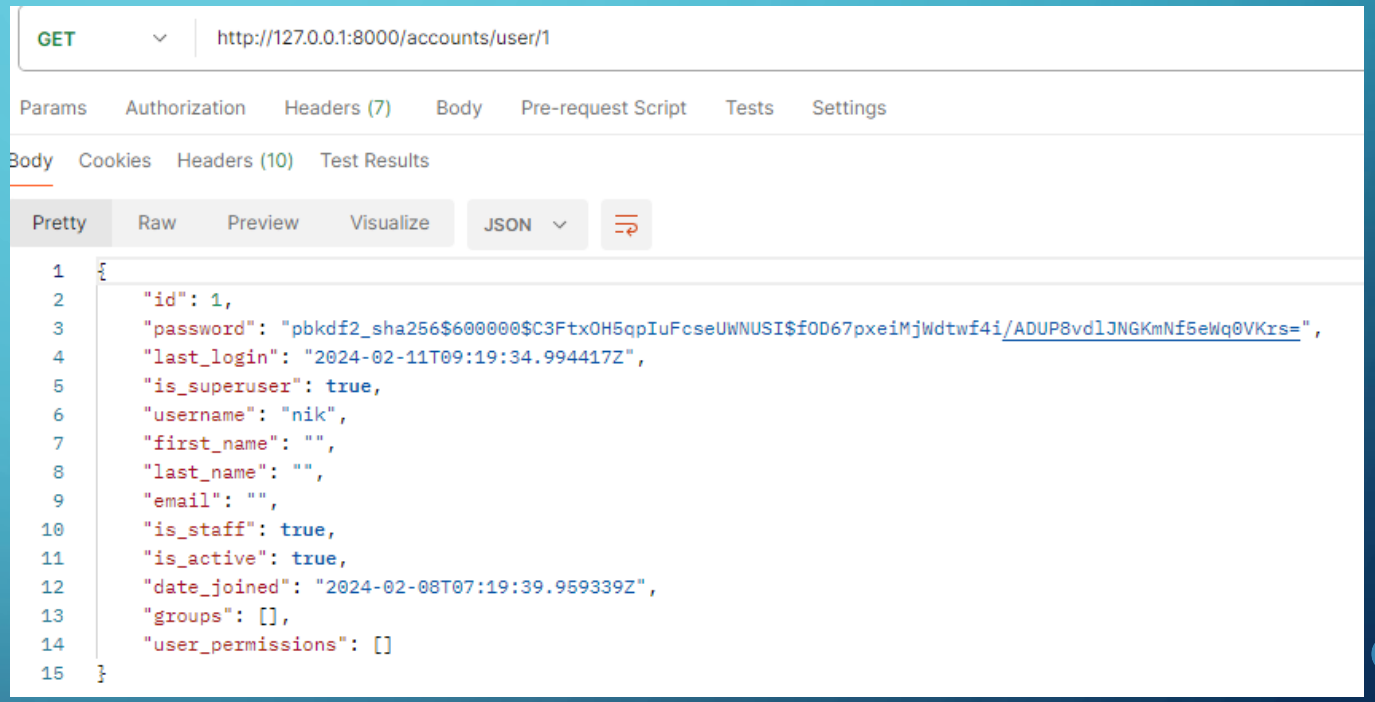
CREATE MORE VIEWSETS

I had queryset and it was all of my users but I didn`t need all therefore I filter it by pk

```python
from django.shortcuts import get_object_or_404

    def retrieve(self, request, pk = None):
        user = get_object_or_404(self.queryset, pk = pk)
        user_serializer = UserSerializer(instance = user )
        return Response(data = user_serializer.data)
```

RESULT

According to table at 117th slide if we user prefix + lookup the retrieve method is going to activate and we don`t need to create specific url for it



PARTIAL_UPDATE VIEWSET

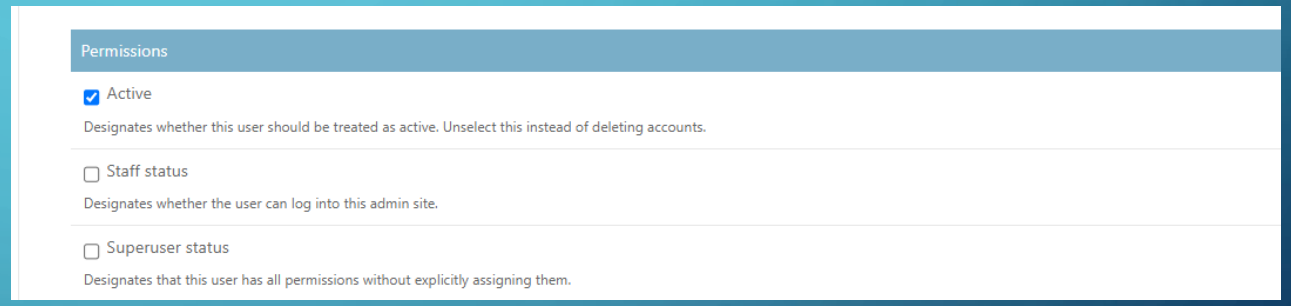
```python
def partial_update(self, request, pk = None):
    user = get_object_or_404(self.queryset, pk = pk)
    user_serializer = UserSerializer(instance = user, data = request.POST , partial = True)
    if user_serializer.is_valid():
        user_serializer.save()
        return Response(data = user_serializer.data)
    return Response(user_serializer.errors)
```

I give it user as instance I tell my serializer that user want to send data

DESTROY USER VIEWSET

- In real world ! We don`t delete the user just deactive it

On admin panel



Using code

```
    def destroy(self, request, pk = None):
        user = get_object_or_404(self.queryset, pk = pk)
        user.is_active = False
        user.save()
        return Response({'message': 'user deactivated'})
```

USING CUSTOM PERMISSION ON VIEWSET

- It`s different a little bit for performance there is no has_object_permission() on viewsets
- If we want to use custom permission we have to use perform_create() or validate it by serializers
- (it`s a simple if !)

```
        if request.user != user:
            return Response({'message': 'you are not the owner'})
```

THROTTLING

- Throttling just like permission but the different is throttling is using for limitation in sending request by user for example each user can send 100 request in 1 hour or the users who didn`t authenticate can send 10 request in 1 hour
- Note : don`t use throttling for security isn`t enough if you want to handle brute force or denial of service throttling isn`t good enough because hackers can change their ip and ditch throttling

THROTTLING

- Throttling using x_forwarded_for Http header and REMOTE_ADDR for identify users if user change those 2 throttling will not know the user . Usually professional users or hackers just change that

SETTING THE THROTTLING POLICY

- A. by using setting which effect on all views
- B. by using class variable in CBV
- C. by using decorators in FBV
- Note : we have to define the request per day/hour/minute/second

DEFINE THROTTLING IN SETTING

Where we define rest frame work we define throttling too

For anonymous users

For authenticated users

Define it the anon and user how much request can send in 1 hour

```
#rest framework
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
    'DEFAULT_THROTTLE_CLASSES' : [
        'rest_framework.throttling.AnonRateThrottle',
        'rest_framework.throttling.UserRateThrottle'
    ],
    'DEFAULT_THROTTLE_RATES' : {
        'anon' : '3/hour',
        'user' : '10/hour'
    }
}
```

DEFAULT_THROTTLE_CLASSES is a list
DEFAULT_THROTTLE_RATES is a dictionary

DEFINE THROTTLING IN CLASSES

A. Delete this part
B. But don't touch THROTTLE RATES

```
#rest framework
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
    'DEFAULT_THROTTLE_CLASSES' : [
        'rest_framework.throttling.AnonrateThrottle',
        'rest_framework.throttling.UserRateThrottle'
    ],
    'DEFAULT_THROTTLE_RATES' : {
        'anon' : '3/hour',
        'user' : '10/hour'
    }
}
```

Import it

Use class variable

```
from rest_framework.throttling import UserRateThrottle, AnonRateThrottle
class QuestionReadView(APIView):
    throttle_classes = [UserRateThrottle, AnonRateThrottle]

    def get(self, request):
        question = Question.objects.all()
        question_serializer = QuestionSerializer(instance= question , many = True)
        return Response(question_serializer.data, status= status.HTTP_200_OK)
```

SCOPED RATE THROTTLE

- Allow us to name the limitation and create the limitation by that name
- A. have to define it in setting.py
- B. use throttle scope as class variable

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework.authentication.TokenAuthentication',
    ],
    'DEFAULT_THROTTLE_CLASSES' : [
        'rest_framework.throttling.ScopedRateThrottle'
    ],
    'DEFAULT_THROTTLE_RATES' : {
        'question' : '5/minutes'
    }
}
```

This is my limitation and I can use it now as class variable

USE IT SCOPED RATED THROTTLE IN CLASS

In using scoped anon or authenticated user is no different .

```
class QuestionReadView(APIView):
    throttle_scope = 'question'

    def get(self, request):
        question = Question.objects.all()
        question_serializer = QuestionSerializer(instance= question , many =
True)
        return Response(question_serializer.data, status= status.HTTP_200_OK)
```

JWT

- Till now we were using token for authentication but there is new standard which call it JWT is stand for of JSON Web Token Authentication
- What`s different ?
  - A. the token has expire and should refresh it
  - B. we don`t save tokens in database instead save it on local storage of user browser
  - C. better performance of website
- The recommended method is using simple JWT

JWT TOKEN

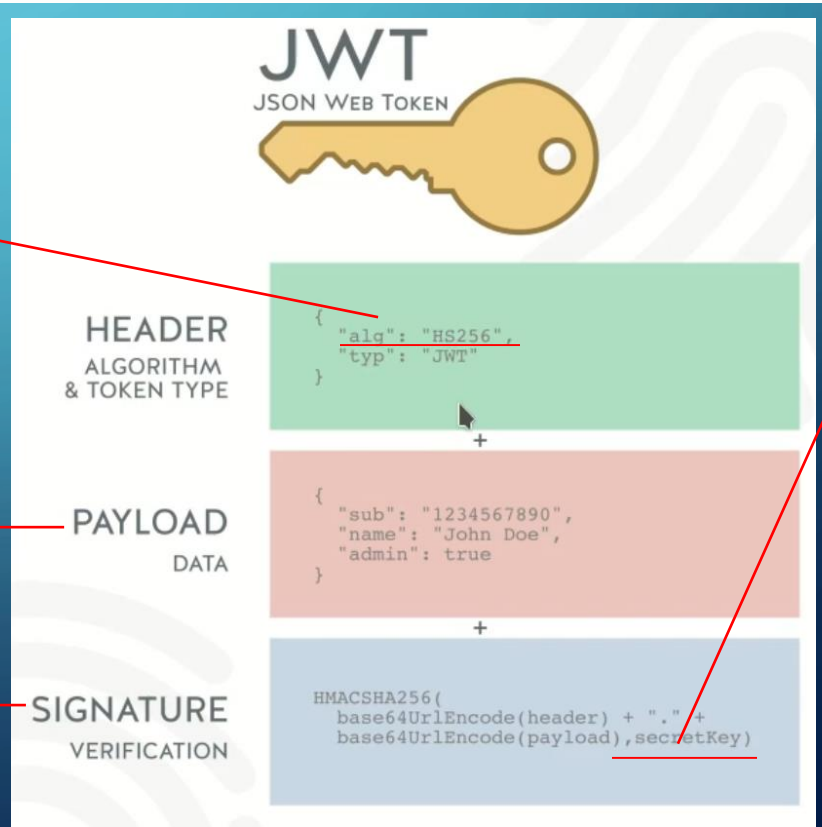
JWT Token dived in 3 part and separate by DOT (.)
Each part doing has specific function

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Using hash 256 for header and algorithm

DATA !

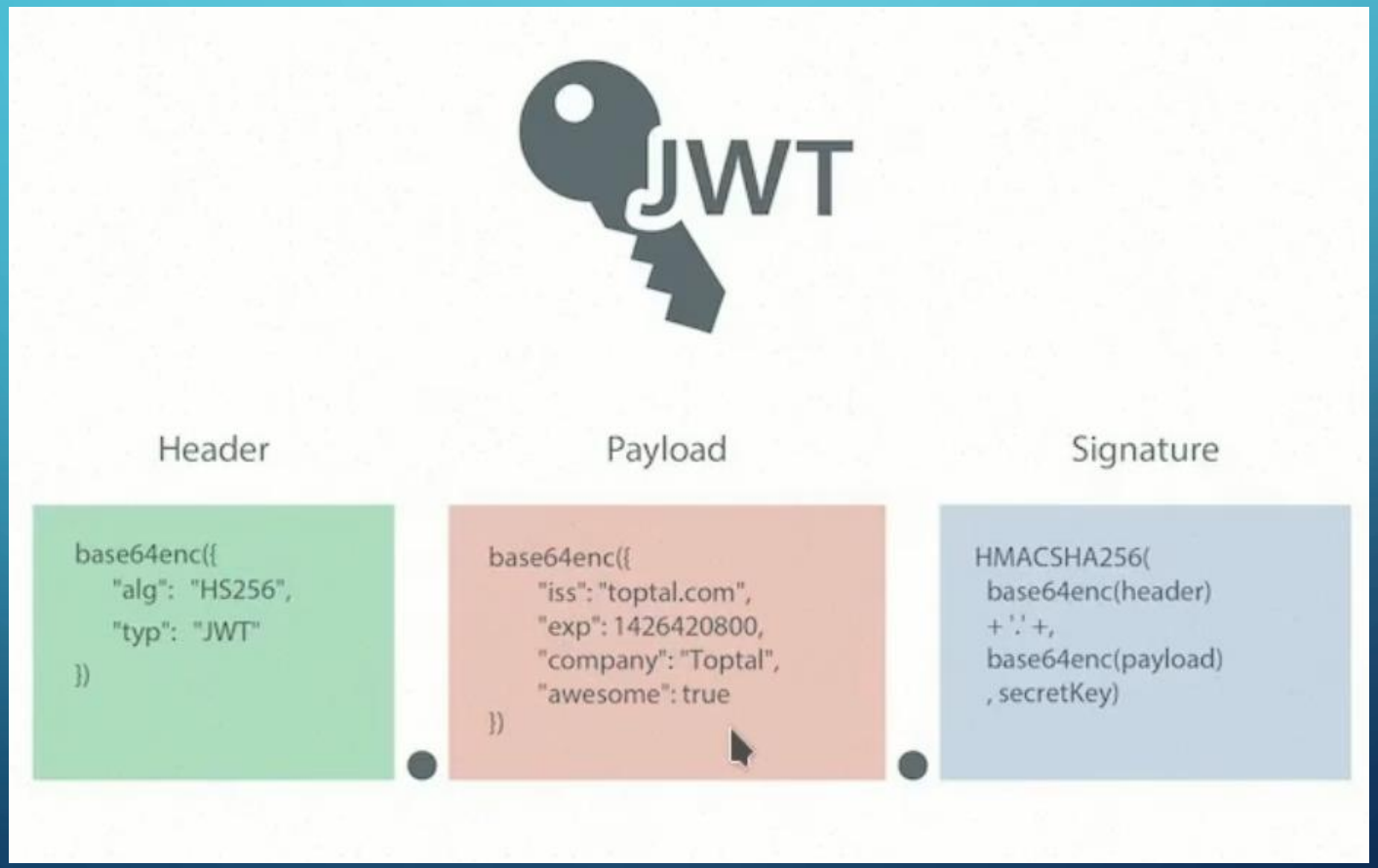
For combination data



Secret key Is available in setting.py

JWT TOKEN

Another example.



USING SIMPLE JWT

- A. pip install djangorestframework-simplejwt
- B. add it to setting.py in DEFAULT_AUTHENTICATION_CLASSES

```
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
```

- C. add views to urls.py
  - It can be root urls.py or apps urls.py

```
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView
```

- D. add the urls and views
  - Into urlpatterns , Note : we don`t need the old url for token authentication therefore delete it ! And delete it rest_framework_auth in setting.py

```
urlpatterns = [
    path('register/', views.RegisterView.as_view(), name = 'register'),
    path('token/', TokenObtainPairView.as_view(), name = 'token_obtain_pair'),
    path('token/refresh', TokenRefreshView.as_view(), name = 'token_refresh')
]
```

SETTING OF JWT

Access token exp
And refresh token exp
These two are important

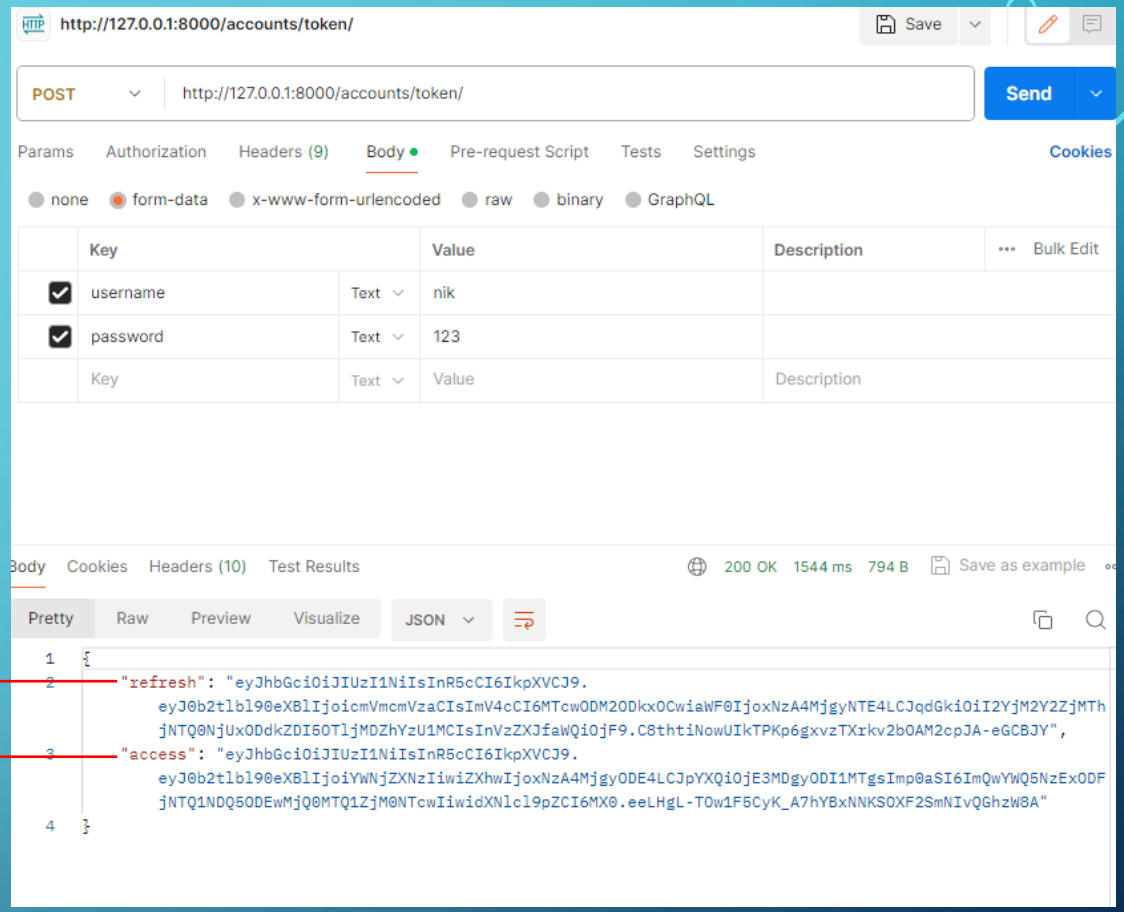
Using Bearer instead of token in sending request

Settings

Some of Simple JWT's behavior can be customized through settings variables in `settings.py`:

```
# Django project settings.py

from datetime import timedelta
...

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(minutes=5),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": False,
    "UPDATE_LAST_LOGIN": False,

    "ALGORITHM": "HS256",
    "SIGNING_KEY": settings.SECRET_KEY,
    "VERIFYING_KEY": "",
    "AUDIENCE": None,
    "ISSUER": None,
    "JSON_ENCODER": None,
    "JWK_URL": None,
    "LEEWAY": 0,

    "AUTH_HEADER_TYPES": ("Bearer",),
    "AUTH_HEADER_NAME": "HTTP_AUTHORIZATION",
    "USER_ID_FIELD": "id",
    "USER_ID_CLAIM": "user_id",
    "USER_AUTHENTICATION_RULE": "rest_framework_simplejwt.authentication.default_user_authentication_ru

    "AUTH_TOKEN_CLASSES": ("rest_framework_simplejwt.tokens.AccessToken",),
    "TOKEN_TYPE_CLAIM": "token_type",
    "TOKEN_USER_CLASS": "rest_framework_simplejwt.models.TokenUser",

    "JTI_CLAIM": "jti",

    "SLIDING_TOKEN_REFRESH_EXP_CLAIM": "refresh_exp",
    "SLIDING_TOKEN_LIFETIME": timedelta(minutes=5),
    "SLIDING_TOKEN_REFRESH_LIFETIME": timedelta(days=1),

    "TOKEN_OBTAIN_SERIALIZER": "rest_framework_simplejwt.serializers.TokenObtainPairSerializer",
    "TOKEN_REFRESH_SERIALIZER": "rest_framework_simplejwt.serializers.TokenRefreshSerializer",
    "TOKEN_VERIFY_SERIALIZER": "rest_framework_simplejwt.serializers.TokenVerifySerializer",
    "TOKEN_BLACKLIST_SERIALIZER": "rest_framework_simplejwt.serializers.TokenBlacklistSerializer",
    "SLIDING_TOKEN_OBTAIN_SERIALIZER": "rest_framework_simplejwt.serializers.TokenObtainSlidingSerializ
    "SLIDING_TOKEN_REFRESH_SERIALIZER": "rest_framework_simplejwt.serializers.TokenRefreshSlidingSerial
}
```

USING JWT

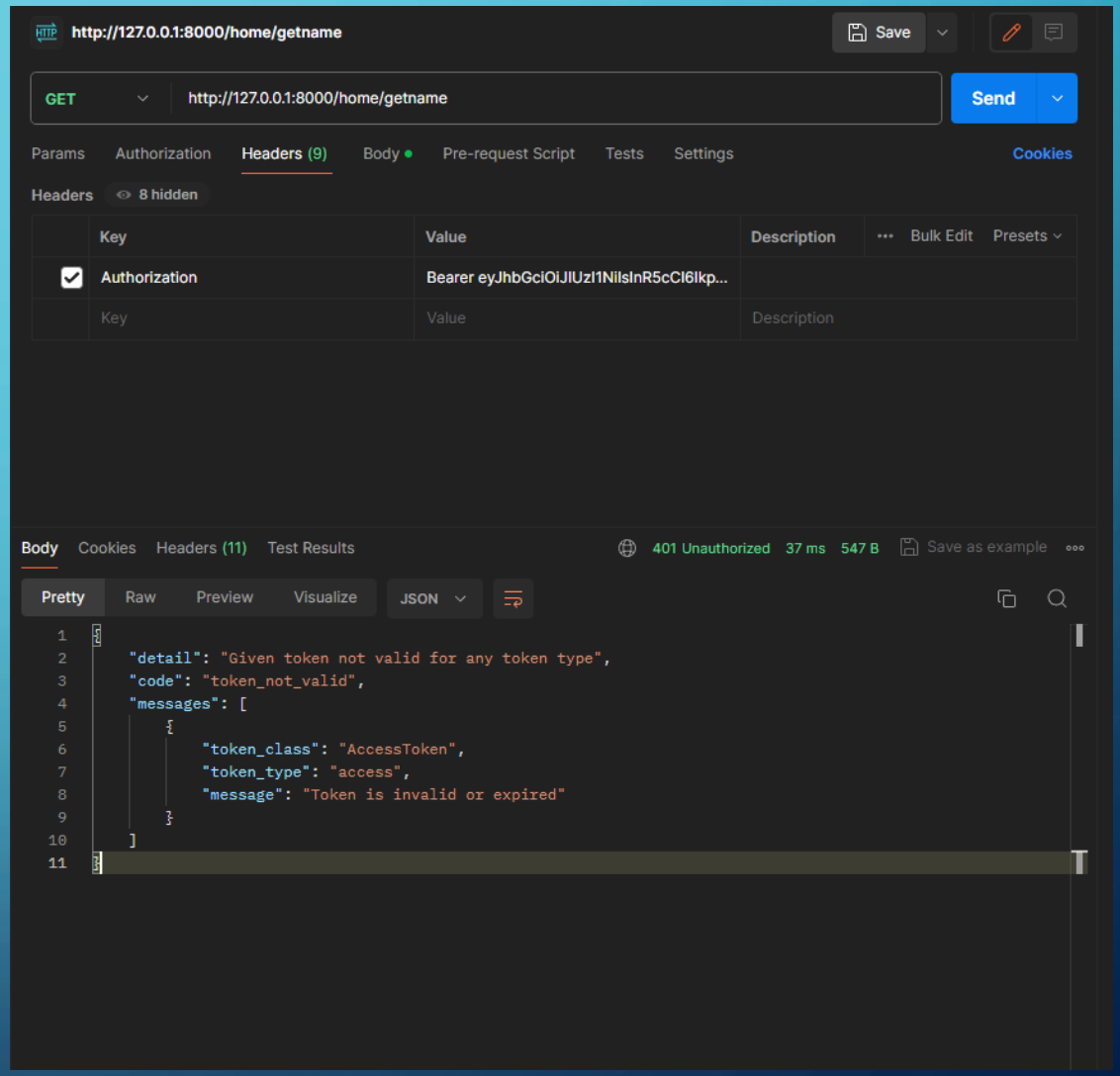


Give us 2 token
Refresh and access

When access token expires we use refresh token
For access to limit part we have to send access token
Now we want to use in GetNameView in home app

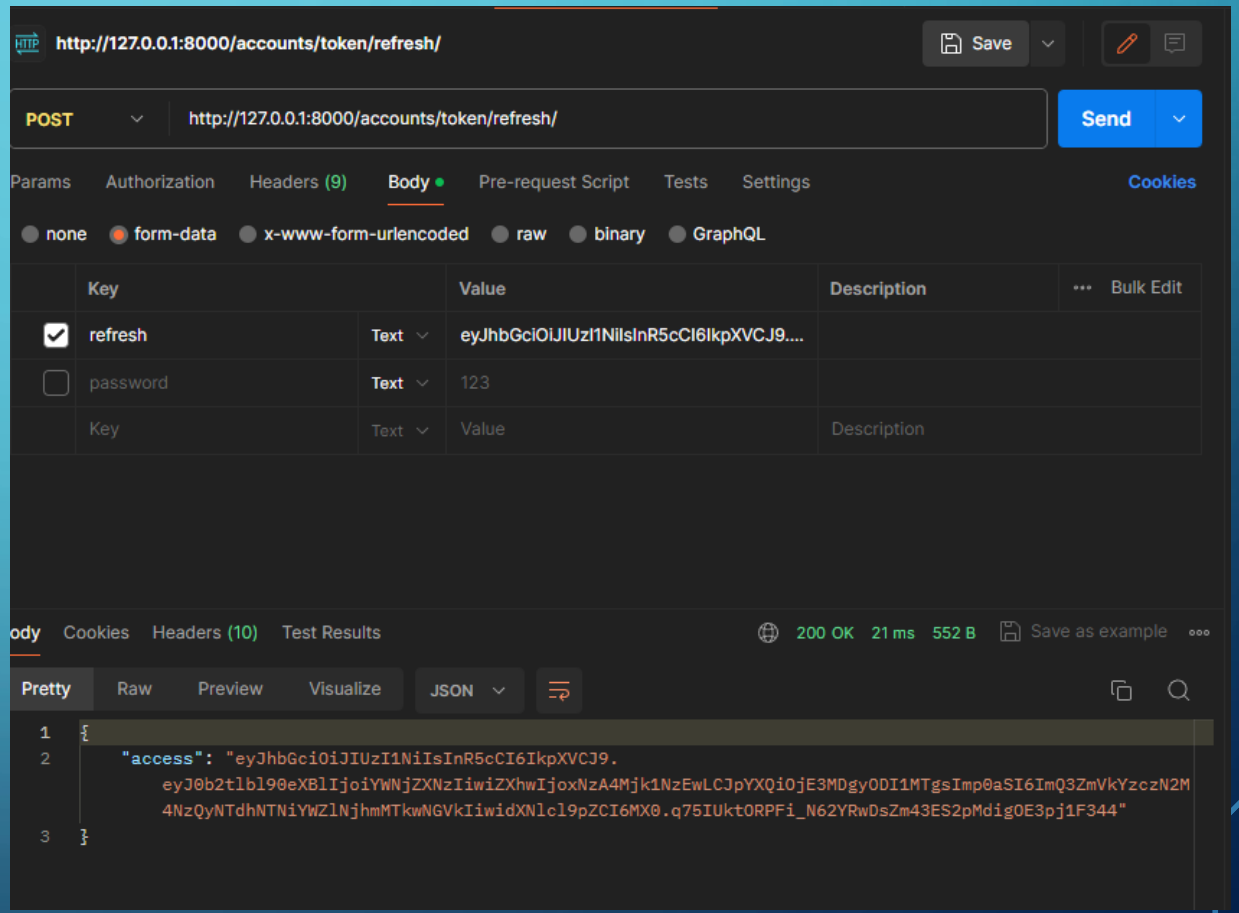
USING JWT

My token is expired but this is how this authentication works



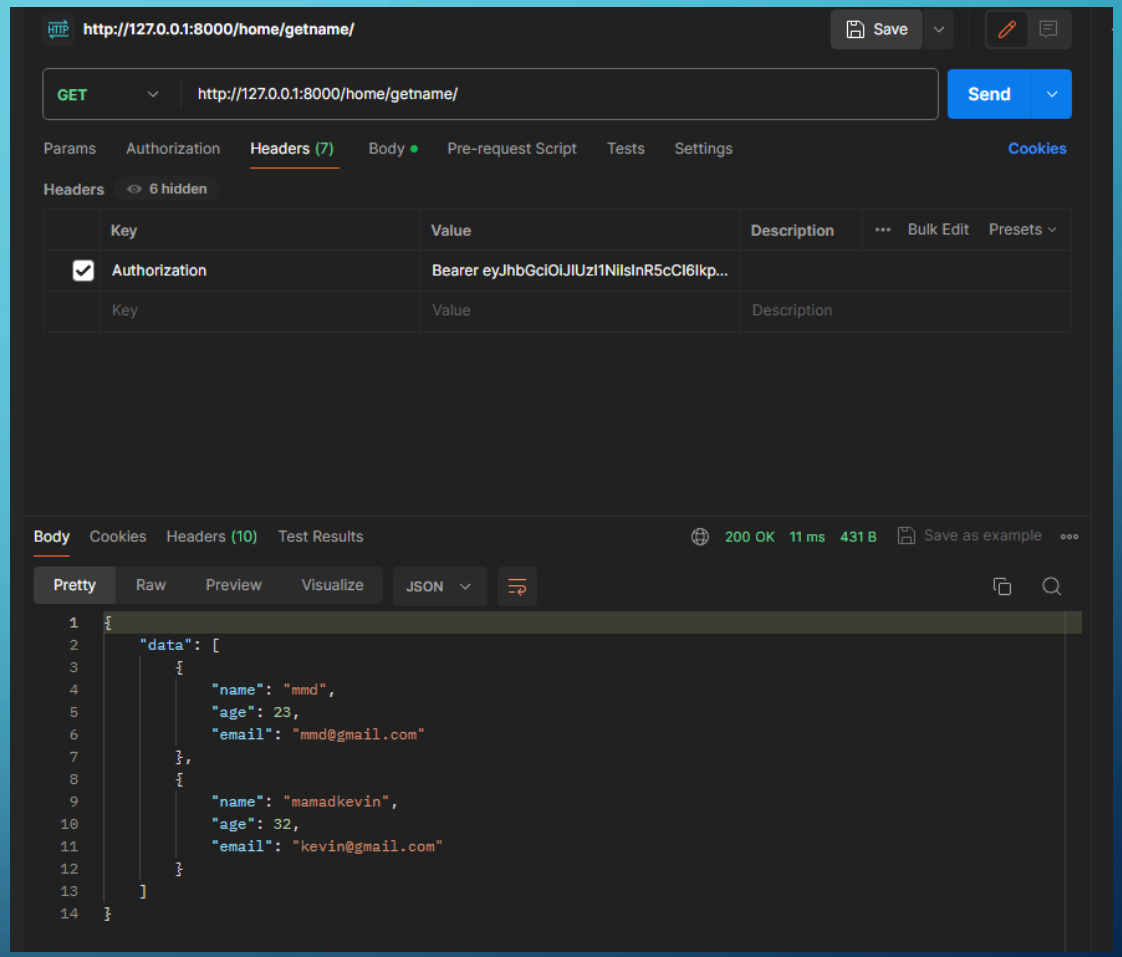
REFRESH JWT TOKEN

I create a key which call it refresh then by POST method I connect to my refresh url then I send my refresh token for it



RESULT

Now I can see the data till next 5 minutes



SWAGGER

- Now we want to create a document from our API because people doesn`t have enough time to read our source code
- For create a document we have to use schema , so what schema ?
- According to Django rest framework documentation :
- *A machine-readable [schema] describes what resources are available via the API, what their URLs are, how they are represented and what operations they support.*

SCHEMA

- Schema as we told is machine readable which means human can`t use it !
- Therefore we have to use some tools to turn this schema to a graphical suitable document
- First we want to create schema
- A. pip install pyyaml → is used to generate schema into YAML-based
- B. pip install uritemplate → for identify the urls

CREATE SCHEMA

- Import get_schema_view and set it into main urls.py
- Give it the documentation path → you have to copy paste it from documentation

I changed path to schema

You have to just copy paste it

```python
from rest_framework.schemas import get_schema_view
    path(
        "schema/",
        get_schema_view(
            title="Your Project", description="API for all things …", version="1.0.0"
        ),
        name="openapi-schema",
    ),
]
```

Now if you read the result you can see the data`s are YAML based but it`s hard to read
Therefore we want to use swagger to make it readable

DRF-SPECTACULAR

- Old timers ! Uses drf-yasg but it`s to old now ! Thus we use drf-spectacular
- https://drf-spectacular.readthedocs.io/en/latest/
- How to use it ?
  - A. pip install drf-spectacular
  - B. add it into installed apps in setting.py
  - C. set the default schema class
  - D. Add it spectacular to setting
  - F. using the documented path and replace it to schema in main urls.py

DRF-SPECTACULAR

The 144th slide steps

I delet api in path too you can do it if you want!

```
A:
  pip install drf-spectacular
B:

INSTALLED_APPS = [
'drf_spectacular',
]
C:
REST_FRAMEWORK = {
'DEFAULT_SCHEMA_CLASS': 'drf_spectacular.openapi.AutoSchema'
}
D:
  SPECTACULAR_SETTINGS = {
    'TITLE': 'Django Rest Project',
    'DESCRIPTION': 'Your project description',
    'VERSION': '1.0.0',
}
  F:
    from drf_spectacular.views import SpectacularAPIView, SpectacularRedocView, SpectacularSwaggerView
urlpatterns = [
    # YOUR PATTERNS
    path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
    # Optional UI:
    path('api/schema/swagger-ui/', SpectacularSwaggerView.as_view(url_name='schema'), name='swagger-ui'),
    path('api/schema/redoc/', SpectacularRedocView.as_view(url_name='schema'), name='redoc'),
```

Downloading schema

Graphical instructor

DRF-SPECTACULAR

- Now if we run it we can see it works!
- But it don`t give us parameters and docstring therefore we have to override it in views and write description by writing docstring and for parameters
- For parameters :
  - Because drf-spectacular didn`t know the serializer couldn`t give us the needed parameters so we have to define serializer to it

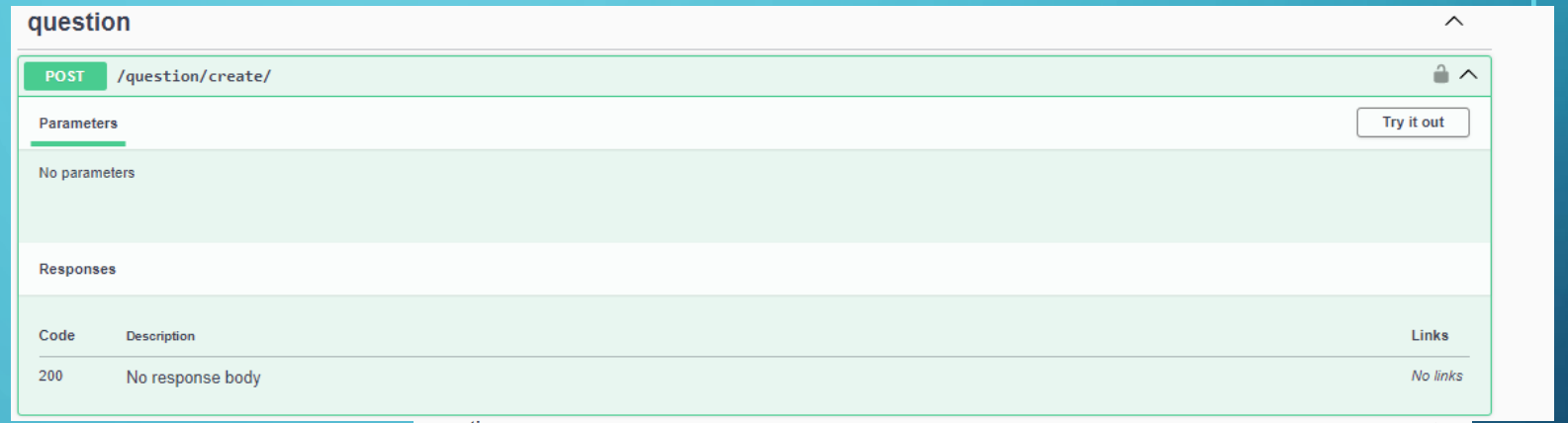
DEFINE SERIALIZER

Adding docstring

Define the serializer

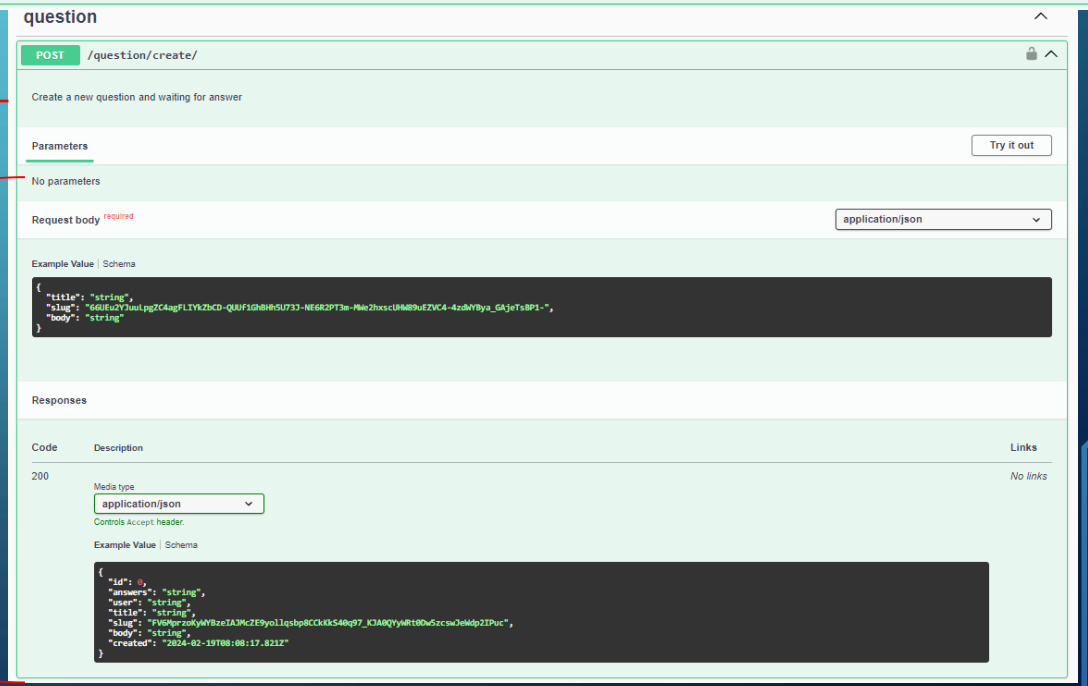
```
class QusetionCreateView(APIView):
    """
    Create a new question and waiting for answer

    """
    permission_classes = [IsAuthenticated,]
    serializer_class = QuestionSerializer
```

BEFORE/AFTER !



docstring

Parameters which shows from serialzier class



REDOC

- Redoc is like swagger but have different ui ! Just that !

- Note : we used api view in our classes therefore it`s need serializer class variable but list view or etc… has it by default . Api view couldn`t identify serializer because we were writing it into our method

RENDERS AND PARSERS

- Render and parser define how data transformation
- For example if we send request it`s json and response is json too but we can change request by parsers and can change response by renders
- It could be YAML , XML, Json or etc.. By default it`s json
- In browser there is another readable data type which call browseable api

SETTING THE RENDER

- Just add it into setting.py in rest_framework part

Render gonna be json

Render gonna be browsableapi

```
'DEFAULT_RENDERER_CLASSES': [
    'rest_framework.renderers.JSONRenderer',
    'rest_framework.renderers.BrowsableAPIRenderer',
]
```

TYPES OF RENDER

- Json render
- Template html render → if you have html page you can show it on html
- Static html render → you can show html code in static
- Browsable api render
- Admin render → just like admin panel in Django
- https://www.django-rest-framework.org/api-guide/renderers/

SET A RENDER

- For setting the render there is an easy way in your default_render_classes
  - just type rest_framework.renders.name_of_render

```
'DEFAULT_RENDERER_CLASSES': [
 'rest_framework.renderers.JSONRenderer',
 'rest_framework.renderers.AdminRenderer',
 'rest_framework.renderers.BrowsableAPIRenderer',
```

CUSTOM RENDER AND THIRD PACKAGE PARTY

- You can create your render or use TPP which allow us to use :
  - YAML
  - XML
  - …
- In real world we just use Jsonrender
- Even we don`t use browsable api render in production time

PARSER

Just like renders !

```
REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES': [
        'rest_framework.parsers.JSONParser',
    ]
}
```

PAGINATION

- Pagination use for partition of data`s
- If you want to make your data dived to some parts you can use pagination
- For pagination we have 3 classes
  - A. page number pagination
  - B. limit offset pagination
  - C. cursor pagination (which it`s useless… often!)
- If we use APIView or viewset we have to pagination manual but if we use generic pagination will automatically set on class

ADDING PAGINATION

On setting.py!

```
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
    'PAGE_SIZE': 100
}
```

First we change it to pagenumberpagination

Also we change it in 2

In each page how much value you want to return

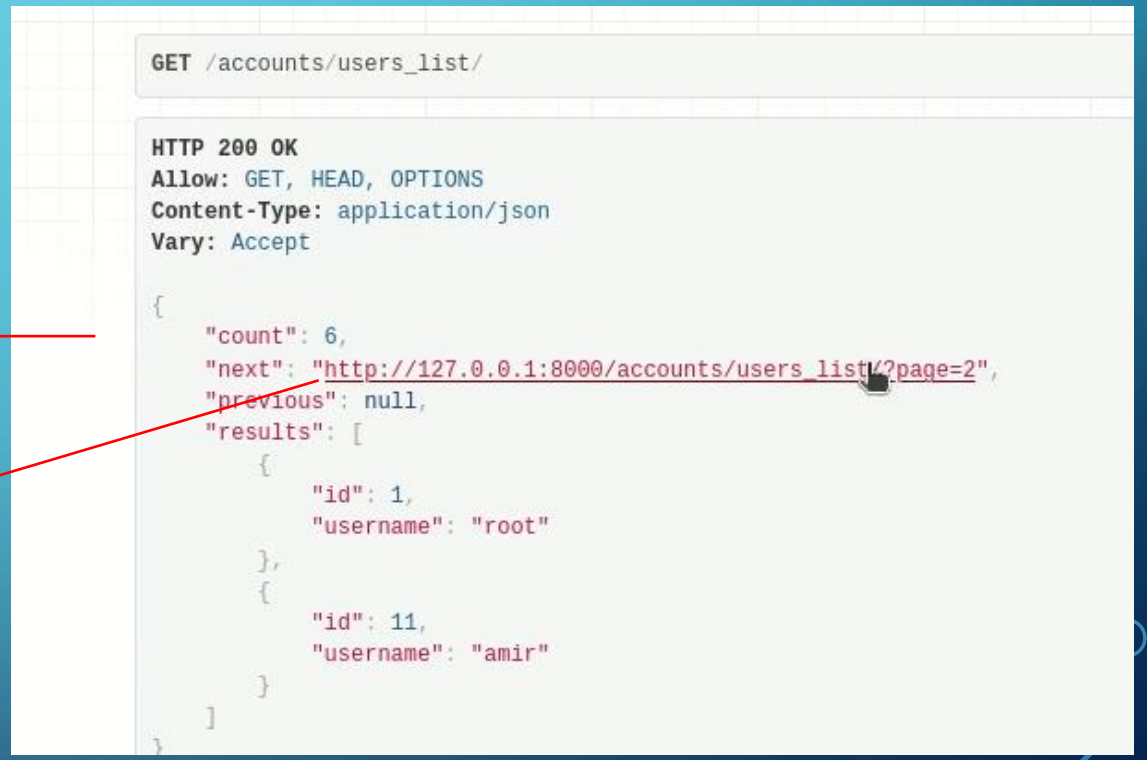
PAGENUMBER PAGINATION

- Pagenumber pagination slice data by page number
- We add pagination on setting therefore it`s not gonna effect on viewset and apiview just on generic view
- Just generic and model viewset handle it automatically

GENERIC

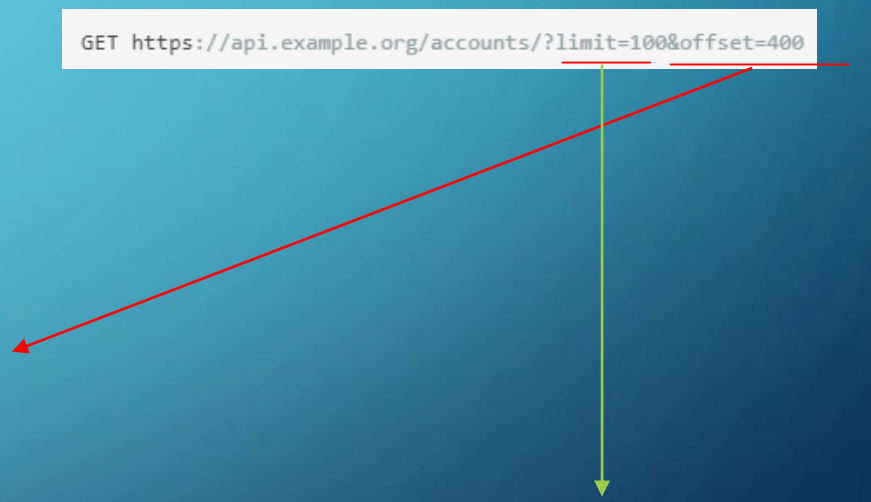
This view writed by generic and it can handle pagination by adding pagination to settings

By page query params (?page = x) categorize data



LIMIT OFFSET PAGINATION

Limit offset pagination getting 2 parameters offset and limit
And we can change limit by changing it



It means howmuch values in database getting ignored. For example I have 500 records , offset = 400 menas start showing me
401 - 500 records

CUSTOM PAGINATION STYLE

- We can create our pagination class
- A. from rest_framework.pagination import PageNumberPagination

Pagination class

Note : you have delete pagination settings in setting.py

Using pagination class

```
class Large(PageNumberPagination):
    page_size = 2

class UserListApi(generics.ListAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    pagination_class = Large
```

USING PAGINATION ON API VIEW

- For api view we can use Django rest framework or Django !
- And we are going to use Django !!

```
A:
  From django.core.paginator import Paginator
```

Saying that get the page number from query params or set it 1 by default

```
class UserApi(APIView):
    def get(self, request):
        queryset = User.objects.all()
        page_number = self.request.query_params.get('page', 1)
        srz_data = UserSerializer(instance=queryset, many=True)
        return Response(data=srz_data.data)
```

USING PAGINATION ON API VIEW

Getting limit from query params or by default set it as 2

```python
class UserApi(APIView):
    def get(self, request):
        queryset = User.objects.all()
        page_number = self.request.query_params.get('page', 1)
        page_size = self.request.query_params.get('limit', 2)
        srz_data = UserSerializer(instance=queryset, many=True)
        return Response(data=srz_data.data)
```

Using paginator class

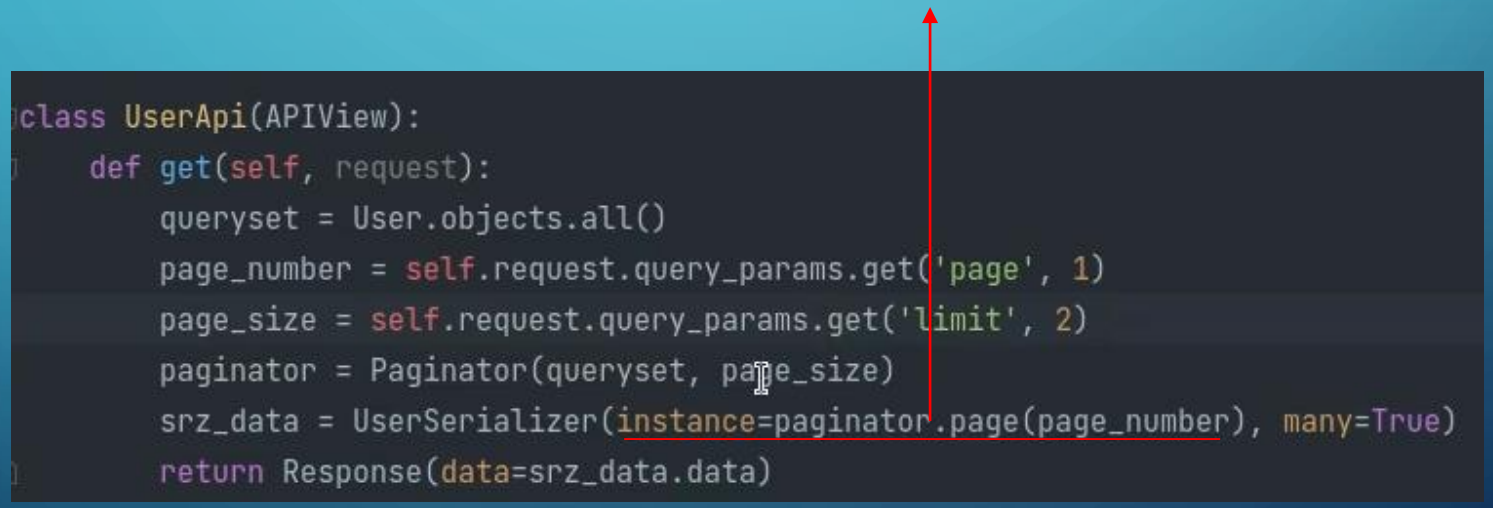
Which variable going to be paginated?

What is pagination size?

```python
class UserApi(APIView):
    def get(self, request):
        queryset = User.objects.all()
        page_number = self.request.query_params.get('page', 1)
        page_size = self.request.query_params.get('limit', 2)
        paginator = Paginator(queryset, page_size)
        srz_data = UserSerializer(instance=queryset, many=True)
        return Response(data=srz_data.data)
```

USING PAGINATION ON API VIEW

Now we send our paginator as instance and say that getting page from user or by default set it 1 (we set this settings on page number)



META DATA

- We use meta data when we want to Control on sent data`s by OPTION method
- What is OPTION method ?
  - The OPTIONS method requests that the target resource transfer the HTTP methods that it supports. This can be used to check the functionality of a web server by requesting '*' instead of a specific resource.
- Sometime docstrings or etc… have important information and we don`t want to show that on OPTION

META DATA

- Meta data controls option to what data`s have to return and what not !

CREATE CUSTOM METADATA CLASS

I create my metadata on permission.py
You can create it anywhere

A:

```python
from rest_framework.metadata import BaseMetadata
```

Name parameter : go to my view and get my view name

For renderer in my render class on view give me the media type

```python
class CustomMetadata(BaseMetadata):
    def determine_metadata(self, request, view):
        return {
            'name': view.get_view_name(),
            'renderers': [renderer.media_type for renderer in view.renderer_classes],
            'parsers': [parser.media_type for parser in view.parser_classes]
        }
```

Same as renderer

ADDING YOUR METADATA CLASS TO SETTINGS

```
REST_FRAMEWORK = {
    'DEFAULT_METADATA_CLASS': 'permissions.CustomMetadata'
}
```

File and class name ! (address it !)

FINITE

- Finite !

THANKS FOR YOU ATTENTION

CREATED BY MOHAMMAD HASAN KHODDAMI

MOHMMADH.KHODDAMI@GMAIL.COM