Function Based api_view

This wrapper provide a few bits of functionality such as making sure you receive Request instances in your view, and adding context to Response objects so that content negotiation can be performed.

The wrapper also provide behaviour such as returning 405 Method Not Allowed responses when appropriate, and handling any ParseError exceptions that occur when accessing request.data with malformed input.

By default only GET methods will be accepted. Other methods will respond with "405 Method Not Allowed".

@api_view()

@api_view([‘GET’, ‘POST’, ‘PUT’, ‘DELETE’])

def function_name(request):

    …….

    …….

api_view

```
from rest_framework.decorators import api_view
from rest_framework.response import Response
@api_view([‘GET’])
def student_list(request):
    if request.method == 'GET':
        stu = Student.objects.all()
        serializer = StudentSerializer(stu, many=True)
        return Response(serializer.data)
```

api_view

```
from rest_framework.decorators import api_view
from rest_framework.response import Response
from rest_framework import status
@api_view([‘POST’])
def student_create(request):
   if request.method == 'POST’:
       serializer = StudentSerializer(data = request.data)
       if serializer.is_valid():
           serializer.save()
           res = {'msg': 'Data Created’}
          return Response(res, status=status.HTTP_201_CREATED)
       return Response(serializer.error, status=status.HTTP_400_BAD_REQUEST)
```

Methods

- GET
- POST
- PUT
- PATCH
- DELETE

Request

REST framework's Request objects provide flexible request parsing that allows you to treat requests with JSON data or other media types in the same way that you would normally deal with form data.

request.data – request.data returns the parsed content of the request body. This is similar to the standard request.POST and request.FILES attributes except that:

- It includes all parsed content, including file and non-file inputs.
- It supports parsing the content of HTTP methods other than POST, meaning that you can access the content of PUT and PATCH requests.
- It supports REST framework's flexible request parsing, rather than just supporting form data. For example you can handle incoming JSON data in the same way that you handle incoming form data.

Request

request.method – request.method returns the uppercased string representation of the request's HTTP method.

Browser-based PUT, PATCH and DELETE forms are transparently supported.

request.query_params – request.query_params is a more correctly named synonym for request.GET.

For clarity inside your code, we recommend using request.query_params instead of the Django's standard request.GET. Doing so will help keep your codebase more correct and obvious - any HTTP method type may include query parameters, not just GET requests.

Response ( )

REST framework supports HTTP content negotiation by providing a Response class which allows you to return content that can be rendered into multiple content types, depending on the client request.

Response objects are initialized with data, which should consist of native Python primitives. REST framework then uses standard HTTP content negotiation to determine how it should render the final response content.

Response class simply provides a nicer interface for returning content-negotiated Web API responses, that can be rendered to multiple formats.

Syntax:- Response(data, status=None, template_name=None, headers=None, content_type=None)

- data: The unrendered, serialized data for the response.
- status: A status code for the response. Defaults to 200.
- template_name: A template name to use only if HTMLRenderer or some other custom template renderer is the accepted renderer for the response.
- headers: A dictionary of HTTP headers to use in the response.
- content_type: The content type of the response. Typically, this will be set automatically by the renderer as determined by content negotiation, but there may be some cases where you need to specify the content type explicitly.

DRF Status

REST framework includes a set of named constants that you can use to make your code more obvious and readable.

The full set of HTTP status codes included in the status module.

Example:-

```
from rest_framework import status

def student_create(request):
        res = {'msg': 'Data Created’}
        return Response(res, status=status.HTTP_201_CREATED)
```

Status Code – 1xx

Informational - 1xx

This class of status code indicates a provisional response. There are no 1xx status codes used in REST framework by default.

HTTP_100_CONTINUE

HTTP_101_SWITCHING_PROTOCOLS

Status Code – 2xx

Successful - 2xx

This class of status code indicates that the client's request was successfully received, understood, and accepted.

HTTP_200_OK

HTTP_201_CREATED

HTTP_202_ACCEPTED

HTTP_203_NON_AUTHORITATIVE_INFORMATION

HTTP_204_NO_CONTENT

HTTP_205_RESET_CONTENT

HTTP_206_PARTIAL_CONTENT

HTTP_207_MULTI_STATUS

HTTP_208_ALREADY_REPORTED

HTTP_226_IM_USED

Status Code – 3xx

Redirection - 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request.

HTTP_300_MULTIPLE_CHOICES

HTTP_301_MOVED_PERMANENTLY

HTTP_302_FOUND

HTTP_303_SEE_OTHER

HTTP_304_NOT_MODIFIED

HTTP_305_USE_PROXY

HTTP_306_RESERVED

HTTP_307_TEMPORARY_REDIRECT

HTTP_308_PERMANENT_REDIRECT

Status Code – 4xx

Client Error - 4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

HTTP_400_BAD_REQUEST

HTTP_401_UNAUTHORIZED

HTTP_402_PAYMENT_REQUIRED

HTTP_403_FORBIDDEN

HTTP_404_NOT_FOUND

HTTP_405_METHOD_NOT_ALLOWED

HTTP_406_NOT_ACCEPTABLE

HTTP_407_PROXY_AUTHENTICATION_REQUIRED

HTTP_408_REQUEST_TIMEOUT

HTTP_409_CONFLICT

Status Code – 4xx

HTTP_410_GONE

HTTP_411_LENGTH_REQUIRED

HTTP_412_PRECONDITION_FAILED

HTTP_413_REQUEST_ENTITY_TOO_LARGE

HTTP_414_REQUEST_URI_TOO_LONG

HTTP_415_UNSUPPORTED_MEDIA_TYPE

HTTP_416_REQUESTED_RANGE_NOT_SATISFIABLE

HTTP_417_EXPECTATION_FAILED

HTTP_422_UNPROCESSABLE_ENTITY

HTTP_423_LOCKED

HTTP_424_FAILED_DEPENDENCY

HTTP_426_UPGRADE_REQUIRED

HTTP_428_PRECONDITION_REQUIRED

HTTP_429_TOO_MANY_REQUESTS

Status Code – 4xx

HTTP_431_REQUEST_HEADER_FIELDS_TOO_LARGE

HTTP_451_UNAVAILABLE_FOR_LEGAL_REASONS

Status Code – 5xx

Server Error - 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition.

HTTP_500_INTERNAL_SERVER_ERROR

HTTP_501_NOT_IMPLEMENTED

HTTP_502_BAD_GATEWAY

HTTP_503_SERVICE_UNAVAILABLE

HTTP_504_GATEWAY_TIMEOUT

HTTP_505_HTTP_VERSION_NOT_SUPPORTED

HTTP_506_VARIANT_ALSO_NEGOTIATES

HTTP_507_INSUFFICIENT_STORAGE

HTTP_508_LOOP_DETECTED

Status Code – 5xx

HTTP_509_BANDWIDTH_LIMIT_EXCEEDED

HTTP_510_NOT_EXTENDED

HTTP_511_NETWORK_AUTHENTICATION_REQUIRED