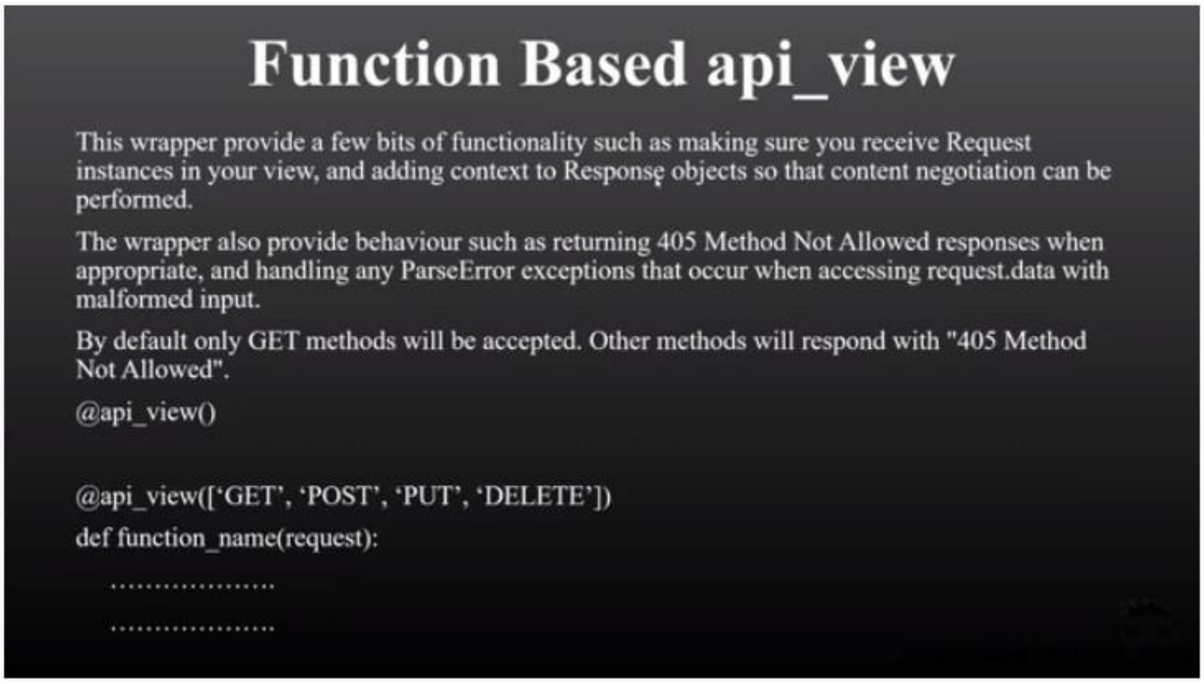


Function Based api_view

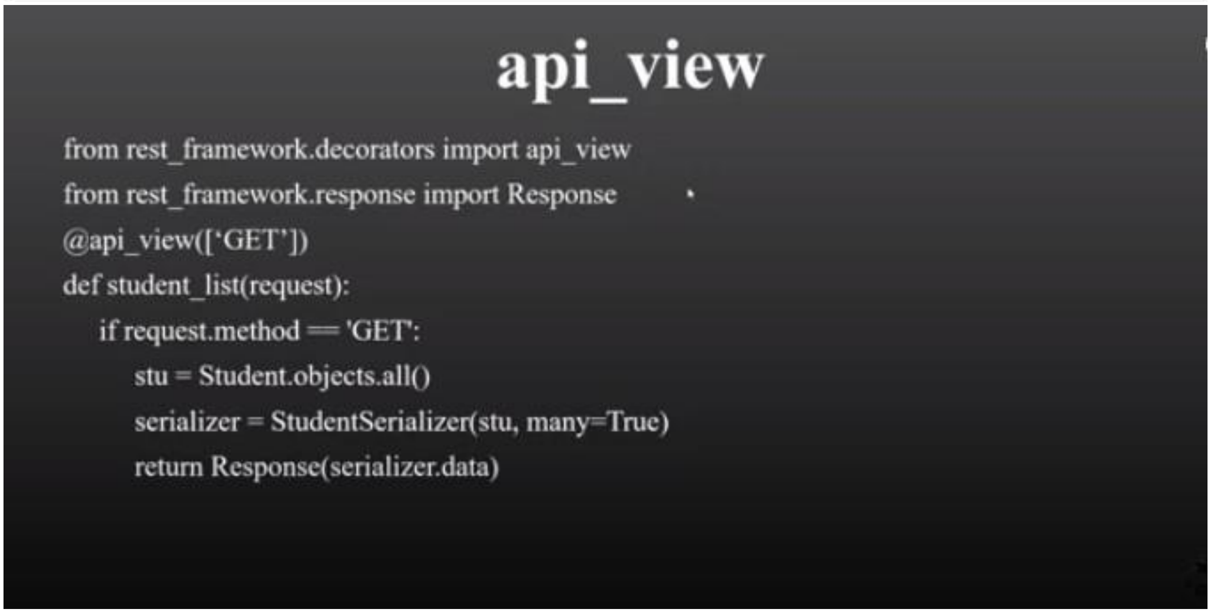
This wrapper provide a few bits of functionality such as making sure you receive Request instances in your view, and adding context to Response objects so that content negotiation can be performed.

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".

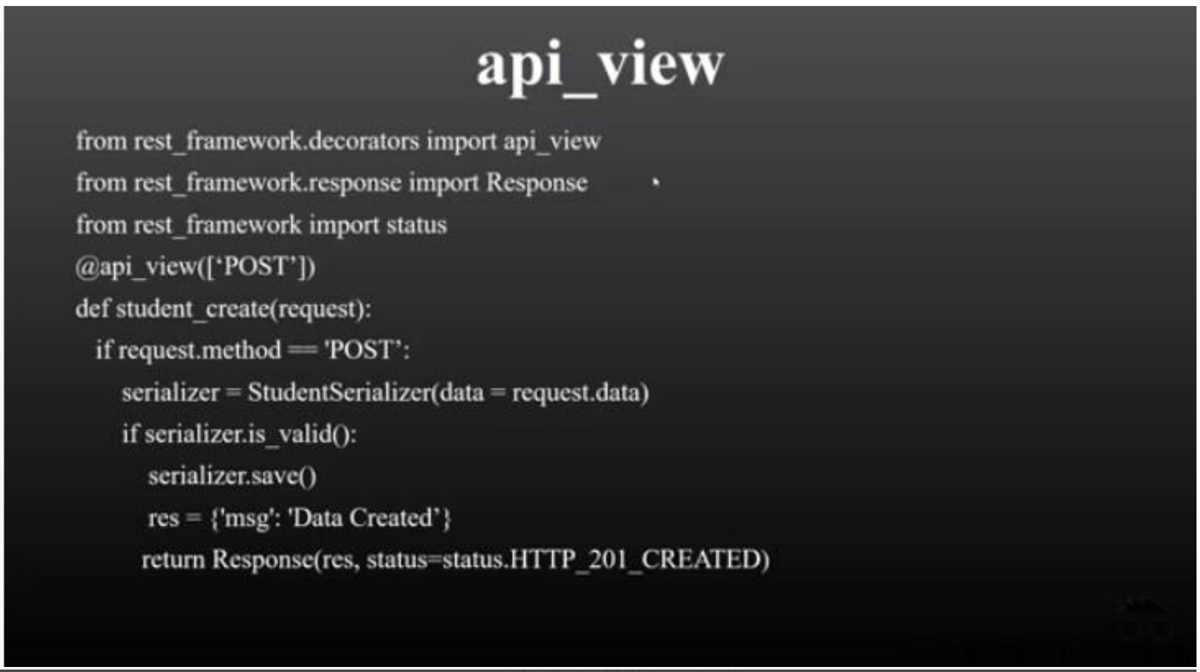
```
@api_view()


@api_view([‘GET’, ‘POST’, ‘PUT’, ‘DELETE’])
def function_name(request):
    ………………..
    ………………..
```



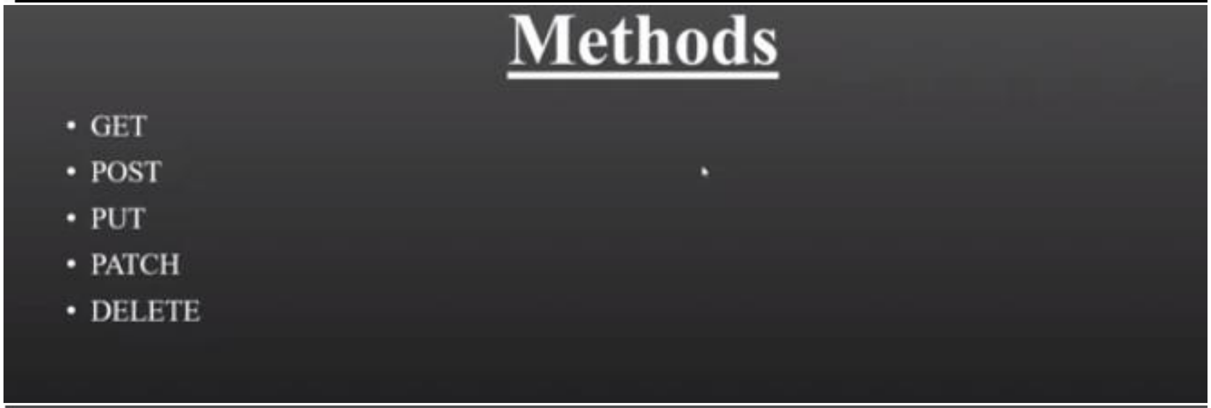
api_view

```
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)
```



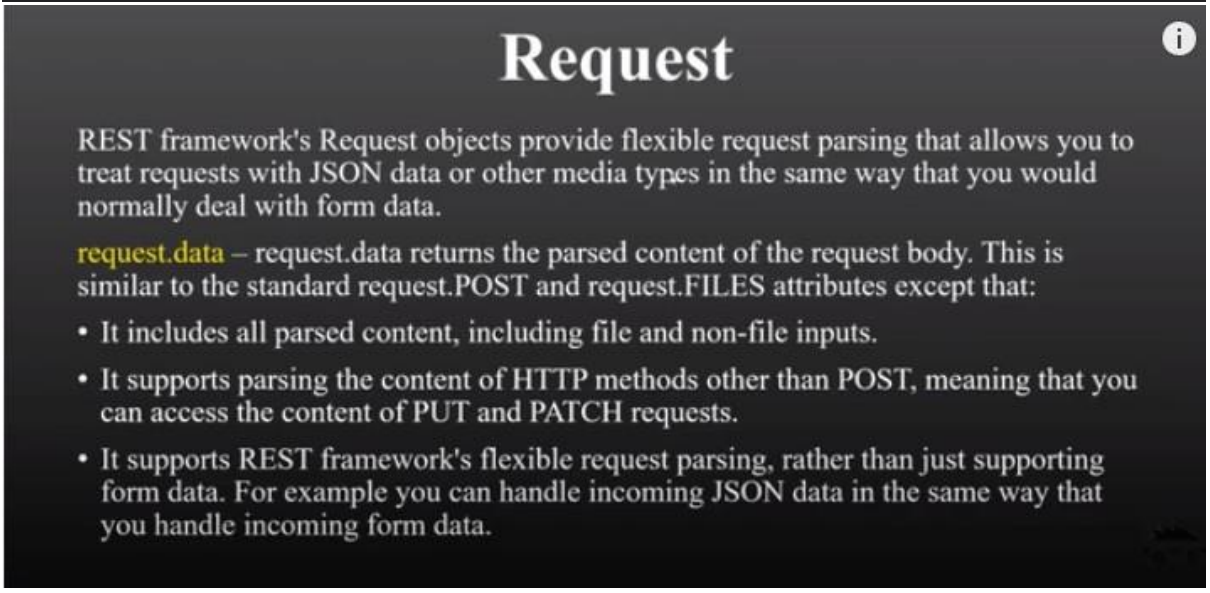
api_view

```
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)
```



Methods

- GET
- POST
- PUT
- PATCH
- DELETE

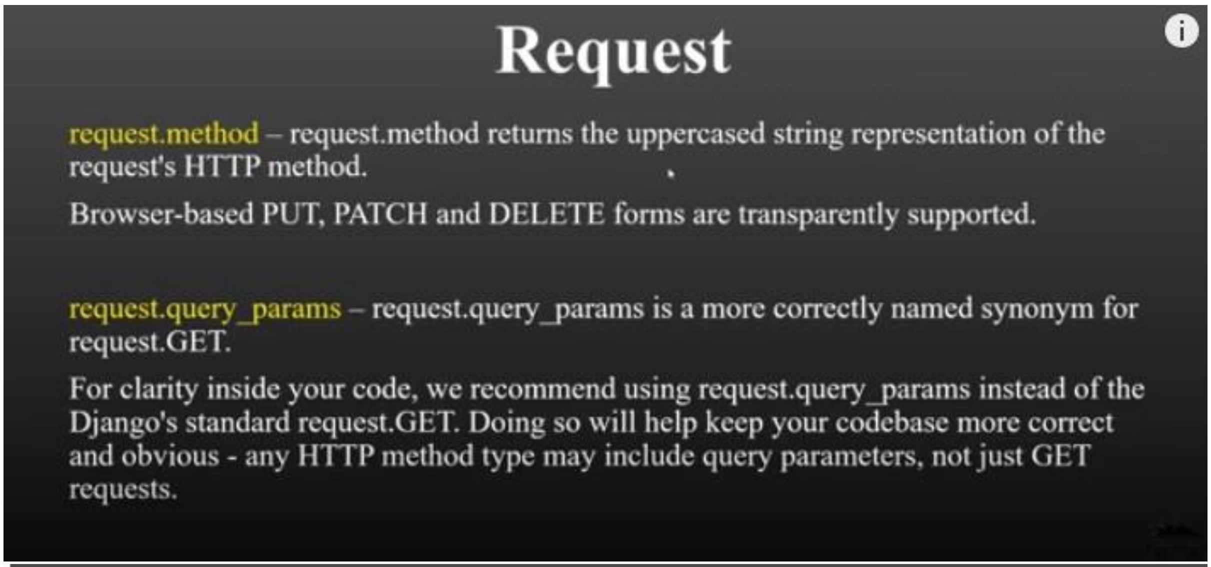


Request

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.



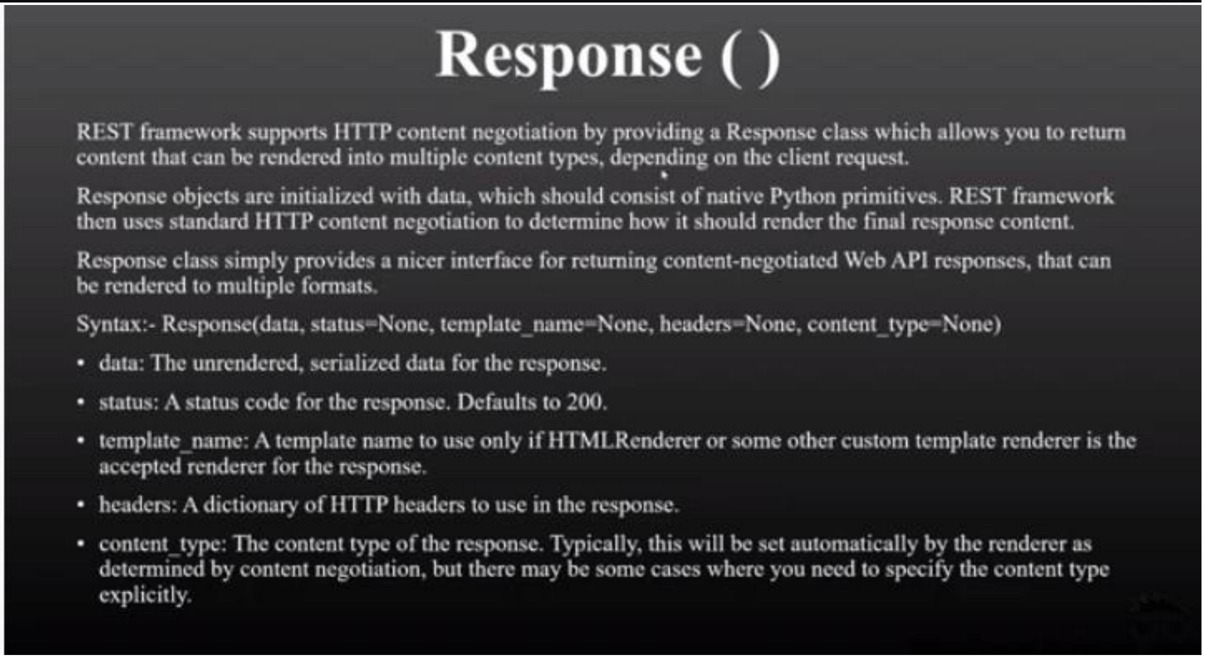
Request

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 ( )

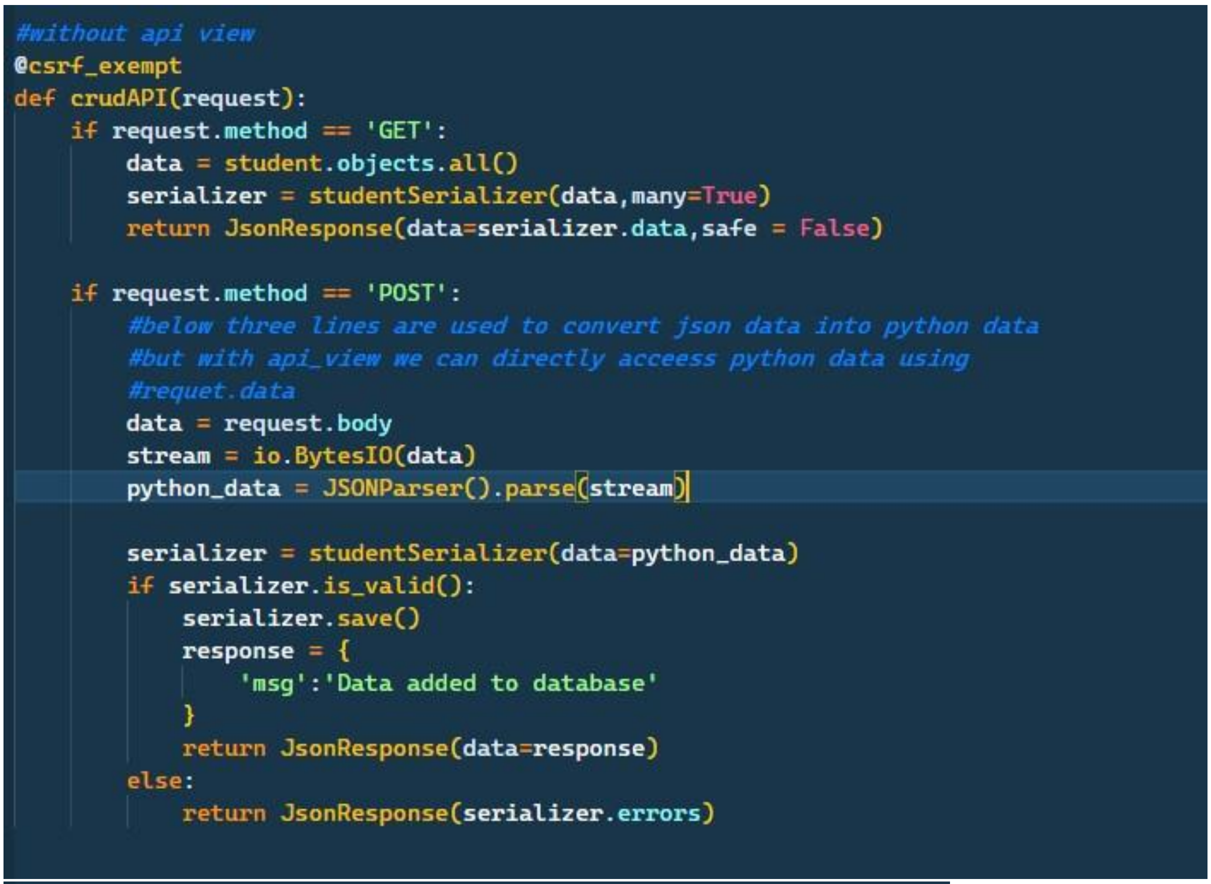
REST framework supports HTTP content negotiation by providing a Response class which allows you to return content that can be rendered into multiple content types, depending on the client request.

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)

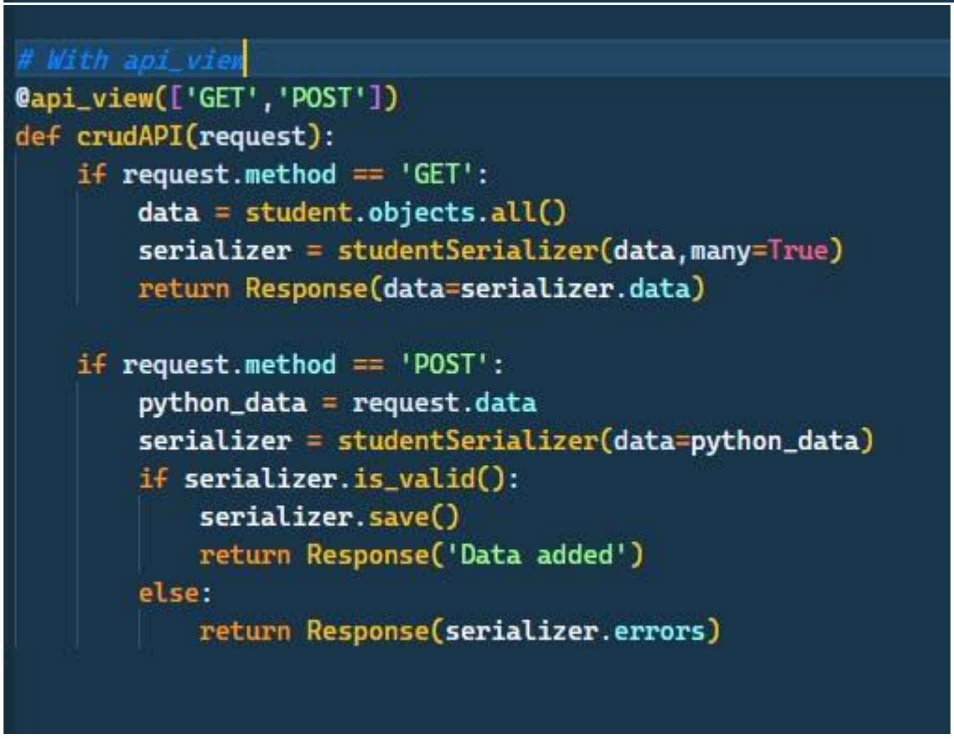
- data: The unrendered, serialized data for the response.
- status: A status code for the response. Defaults to 200.
- template_name: A template name to use only if HTMLRenderer or some other custom template renderer is the accepted renderer for the response.
- headers: A dictionary of HTTP headers to 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.



```python
#without api view
@csrf_exempt
def crudAPI(request):
    if request.method == 'GET':
        data = student.objects.all()
        serializer = studentSerializer(data,many=True)
        return JsonResponse(data=serializer.data,safe = False)

    if request.method == 'POST':
        #below three lines are used to convert json data into python data
        #but with api_view we can directly acceess python data using
        #requet.data
        data = request.body
        stream = io.BytesIO(data)
        python_data = JSONParser().parse(stream)

        serializer = studentSerializer(data=python_data)
        if serializer.is_valid():
            serializer.save()
            response = {
                'msg':'Data added to database'
            }
            return JsonResponse(data=response)
        else:
            return JsonResponse(serializer.errors)
```



```python
# With api_view
@api_view(['GET','POST'])
def crudAPI(request):
    if request.method == 'GET':
        data = student.objects.all()
        serializer = studentSerializer(data,many=True)
        return Response(data=serializer.data)

    if request.method == 'POST':
        python_data = request.data
        serializer = studentSerializer(data=python_data)
        if serializer.is_valid():
            serializer.save()
            return Response('Data added')
        else:
            return Response(serializer.errors)
```