

Topic: 5. Serializing / Deserializing JSON Data (GET/POST)

Speaker: Personal / Notebook: API Development using Django Framework



To see more details about serializers, view this [Youtube clip](#)

1. To view the JSON file in a formatted style, we added the [Google Chrome extension, JSON FORMATTER](#):

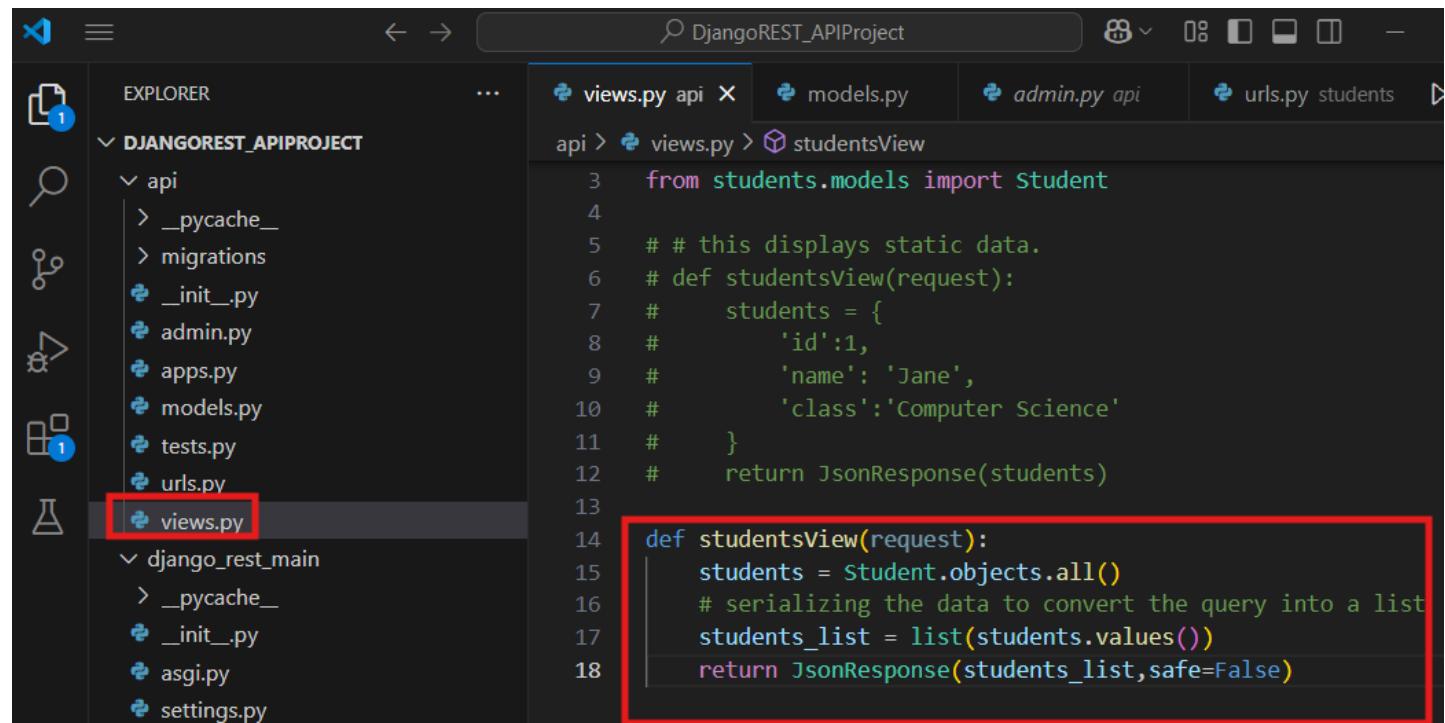
BEFORE:

A screenshot of a browser window showing a JSON response. The URL is 127.0.0.1:8000/api/v1/students/. The response is a single line of JSON: [{"id":1,"student_id":"S001","name":"Rosilie","branch":"Computer Science"}, {"id":2,"student_id":"S002","name":"Yuri","branch":"Engineering"}, {"id":3,"student_id":"S003","name":"Xeria","branch":"Hotel Mgt"}]. A red box highlights this line of text.

AFTER:

A screenshot of a browser window showing the same JSON response, but with pretty-printing enabled. The URL is 127.0.0.1:8000/api/v1/students/. The response is now formatted with indentation: [{ "id": 1, "student_id": "S001", "name": "Rosilie", "branch": "Computer Science" }, { "id": 2, "student_id": "S002", "name": "Yuri", "branch": "Engineering" }, { "id": 3, "student_id": "S003", "name": "Xeria", "branch": "Hotel Mgt" }]. A red box highlights the opening bracket [and the entire list of objects. The 'Pretty-print' checkbox in the browser's address bar is checked.

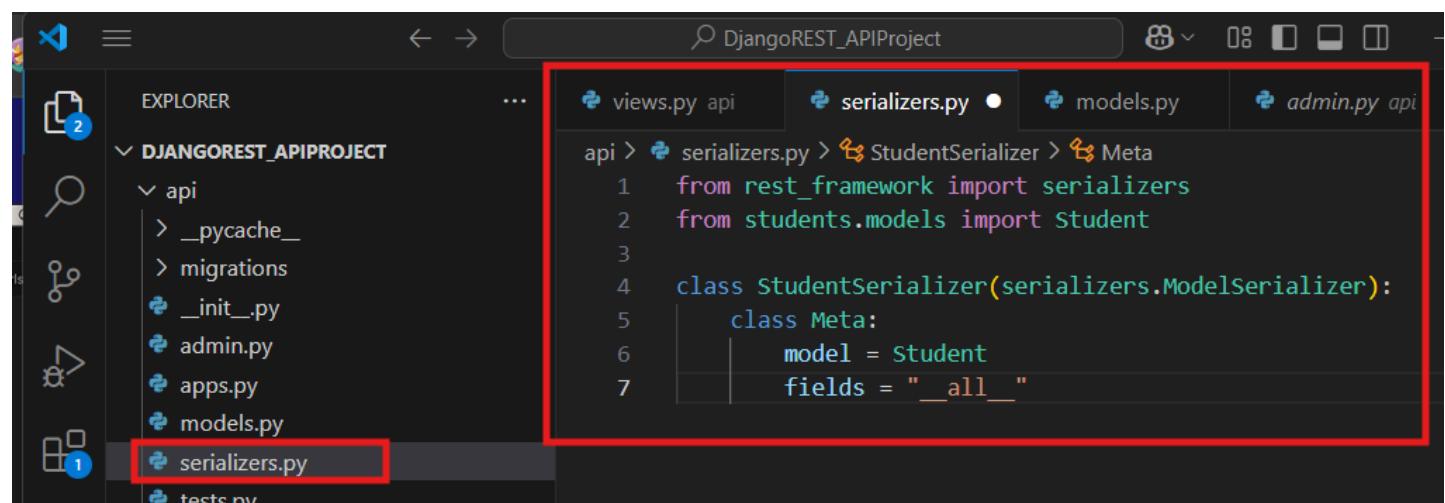
2. Previously, we manually used serializers to convert our query set into a list. The code is below to show the output in Step 1.



The screenshot shows the VS Code interface with the Django REST API Project open. The Explorer sidebar on the left shows the project structure, including the API app with files like `views.py`, `models.py`, and `admin.py`. The `views.py` file is selected and highlighted with a red box around its code block. The code in `views.py` is as follows:

```
3  from students.models import Student
4
5  # # this displays static data.
6  # def studentsView(request):
7  #     students = {
8  #         'id':1,
9  #         'name': 'Jane',
10 #         'class':'Computer Science'
11 #     }
12 #     return JsonResponse(students)
13
14 def studentsView(request):
15     students = Student.objects.all()
16     # serializing the data to convert the query into a list
17     students_list = list(students.values())
18     return JsonResponse(students_list,safe=False)
```

3. In Django, we can use serializer tools. In the API app folder, create a new file SERIALIZERS.PY:

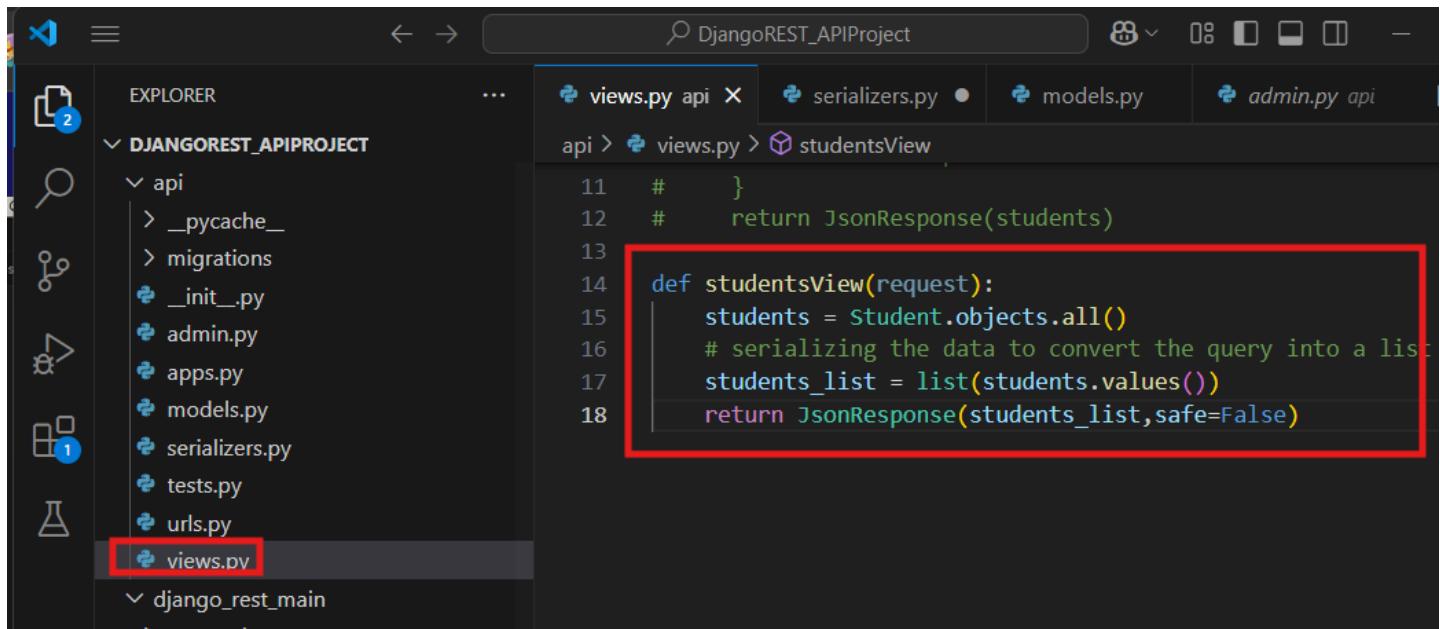


The screenshot shows the VS Code interface with the Django REST API Project open. The Explorer sidebar on the left shows the project structure, including the API app with files like `views.py`, `serializers.py`, `models.py`, and `admin.py`. The `serializers.py` file is selected and highlighted with a red box around its code block. The code in `serializers.py` is as follows:

```
1  from rest_framework import serializers
2  from students.models import Student
3
4  class StudentSerializer(serializers.ModelSerializer):
5      class Meta:
6          model = Student
7          fields = "__all__"
```

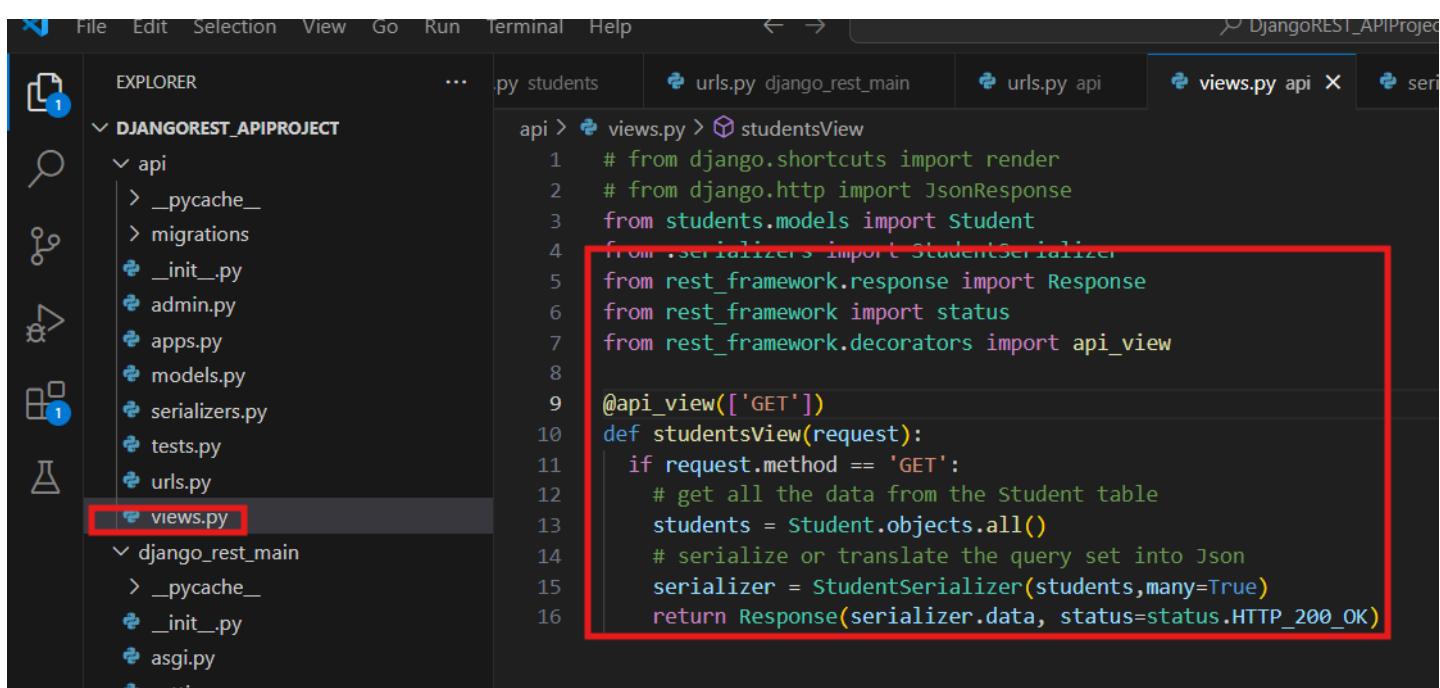
4. Update our API\VIEWS.PY:

FROM manual serialization:



```
views.py api X serializers.py models.py admin.py api
api > views.py > studentsView
11     }
12     # return JsonResponse(students)
13
14 def studentsView(request):
15     students = Student.objects.all()
16     # serializing the data to convert the query into a list
17     students_list = list(students.values())
18     return JsonResponse(students_list,safe=False)
```

TO:



```
views.py students urls.py django_rest_main urls.py api views.py api X serializers.py
api > views.py > studentsView
1 # from django.shortcuts import render
2 # from django.http import JsonResponse
3 from students.models import Student
4 from .serializers import StudentSerializer
5 from rest_framework.response import Response
6 from rest_framework import status
7 from rest_framework.decorators import api_view
8
9@api_view(['GET'])
10def studentsView(request):
11    if request.method == 'GET':
12        # get all the data from the Student table
13        students = Student.objects.all()
14        # serialize or translate the query set into Json
15        serializer = StudentSerializer(students,many=True)
16        return Response(serializer.data, status=status.HTTP_200_OK)
```

So, when you run the URL path again:

<http://127.0.0.1:8000/api/v1/students/>

The screenshot shows a browser window with the URL `127.0.0.1:8000/api/v1/students/` in the address bar. The page title is "Django REST framework" and the sub-page title is "api_djangoadmin". The main content area is titled "Students". On the right, there are two buttons: "OPTIONS" and "GET ▾". Below these buttons, a "GET /api/v1/students/" request is shown with the following response:

```
HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

[{"id": 1, "student_id": "S001", "name": "Rosilie", "branch": "Computer Science"}, {"id": 2, "student_id": "S002", "name": "Yuri", "branch": "Engineering"}, {"id": 3, "student_id": "S003", "name": "Xeria", "branch": "Hotel Mgt"}]
```

5. Now, if you update your database model for a new record and use the GET button from Step 4, you will be able to use GET button to get the latest added records or you can simply refresh your page and that will be considered as a GET method.

← → ⌂ 127.0.0.1:8000/api/v1/students/ ☆ ⌂ ⌂ ⌂ ⌂ ⌂

Django REST framework

api_djangoadmin

Students

Students

[OPTIONS](#) [GET](#) ▾

[GET /api/v1/students/](#)

HTTP 200 OK
Allow: GET, OPTIONS
Content-Type: application/json
Vary: Accept

```
[  
  {  
    "id": 1,  
    "student_id": "S001",  
    "name": "Rosilie",  
    "branch": "Computer Science"  
  },  
  {  
    "id": 2,  
    "student_id": "S002",  
    "name": "Yuri",  
    "branch": "Engineering"  
  },  
  {  
    "id": 3,  
    "student_id": "S003",  
    "name": "Xeria",  
    "branch": "Hotel Mgt"  
  },  
  {  
    "id": 4,  
    "student_id": "S004",  
    "name": "Russell",  
    "branch": "Veterinary"  
  }]
```

6. Now using **POSTMAN**, you can copy the same API link paste it into the POSTMAN search bar and use GET method. It should return all records from the database. Simply click on + and add the same path we used from the browser. To use POSTMAN, this must be installed in your device.

HTTP 200 OK

Allow: GET, OPTIONS

Content-Type: application/json

Vary: Accept

```
[{"id": 1, "student_id": "S001", "name": "Rosilie", "branch": "Computer Science"}, {"id": 2, "student_id": "S002", "name": "Yuri", "branch": "Engineering"}, {"id": 3, "student_id": "S003", "name": "Xeria", "branch": "Hotel Mgt"}, {"id": 4, "student_id": "S004", "name": "Russell", "branch": "Veterinary"}]
```

200 OK 11 ms 592 B

7. To store data using the Django Rest Framework, update the VIEWS.PY to allow for POST method.

```
# from django.http import JsonResponse
from students.models import Student
from .serializers import StudentSerializer
from rest_framework.response import Response
from rest_framework import status
from rest_framework.decorators import api_view

@api_view(['GET', 'POST'])
def studentsView(request):
    if request.method == 'GET':
        # get all the data from the Student table
        students = Student.objects.all()
        # serialize or translate the query set into Json
        serializer = StudentSerializer(students, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)
    elif request.method == 'POST':
        # saves data into our database
        serializer = StudentSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        # if data are not valid
        print(serializer.errors)
        return Response(serializer.errors, status.HTTP_400_BAD_REQUEST)
```

When you reload your page, then you can insert a new post:

12 Std | Person | Select | 127.0 | S | G | how c | Settings | - | □

127.0.0.1:8000/api/v1/students/

Django REST framework

Students

OPTIONS GET

GET /api/v1/students/

```
HTTP 200 OK
Allow: POST, OPTIONS, GET
Content-Type: application/json
Vary: Accept

[{"id": 1, "student_id": "S001", "name": "Rosilie", "branch": "Computer Science"}, {"id": 2, "student_id": "S002", "name": "Yuri", "branch": "Engineering"}, {"id": 3, "student_id": "S003", "name": "Xeria", "branch": "Hotel Mgt"}, {"id": 4, "student_id": "S004", "name": "Russell", "branch": "Veterinary"}]
```

Media type: application/json

Content:

```
{"student_id": "S005", "name": "Mary Ann", "branch": "Engineering"}
```

POST

127.0.0.1:8000/api/v1/students/

HTTP 201 Created

Allow: POST, OPTIONS, GET

Content-Type: application/json

Vary: Accept

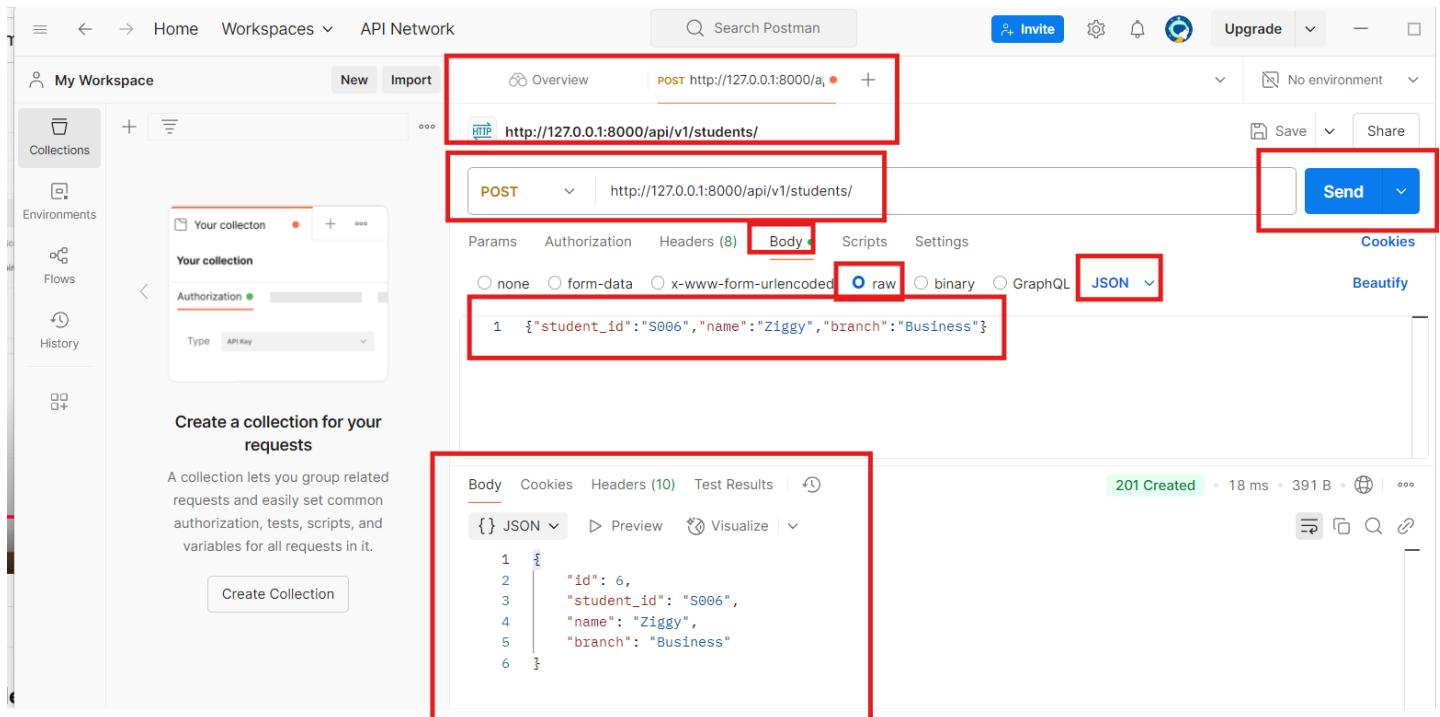
```
{  
    "id": 5,  
    "student_id": "S005",  
    "name": "Mary Ann",  
    "branch": "Engineering"  
}
```

Media type: application/json

Content:

POST

8. To use POSTMAN, add the path again. Select BODY, then RAW, then JSON. Add your records then select select the SEND method.

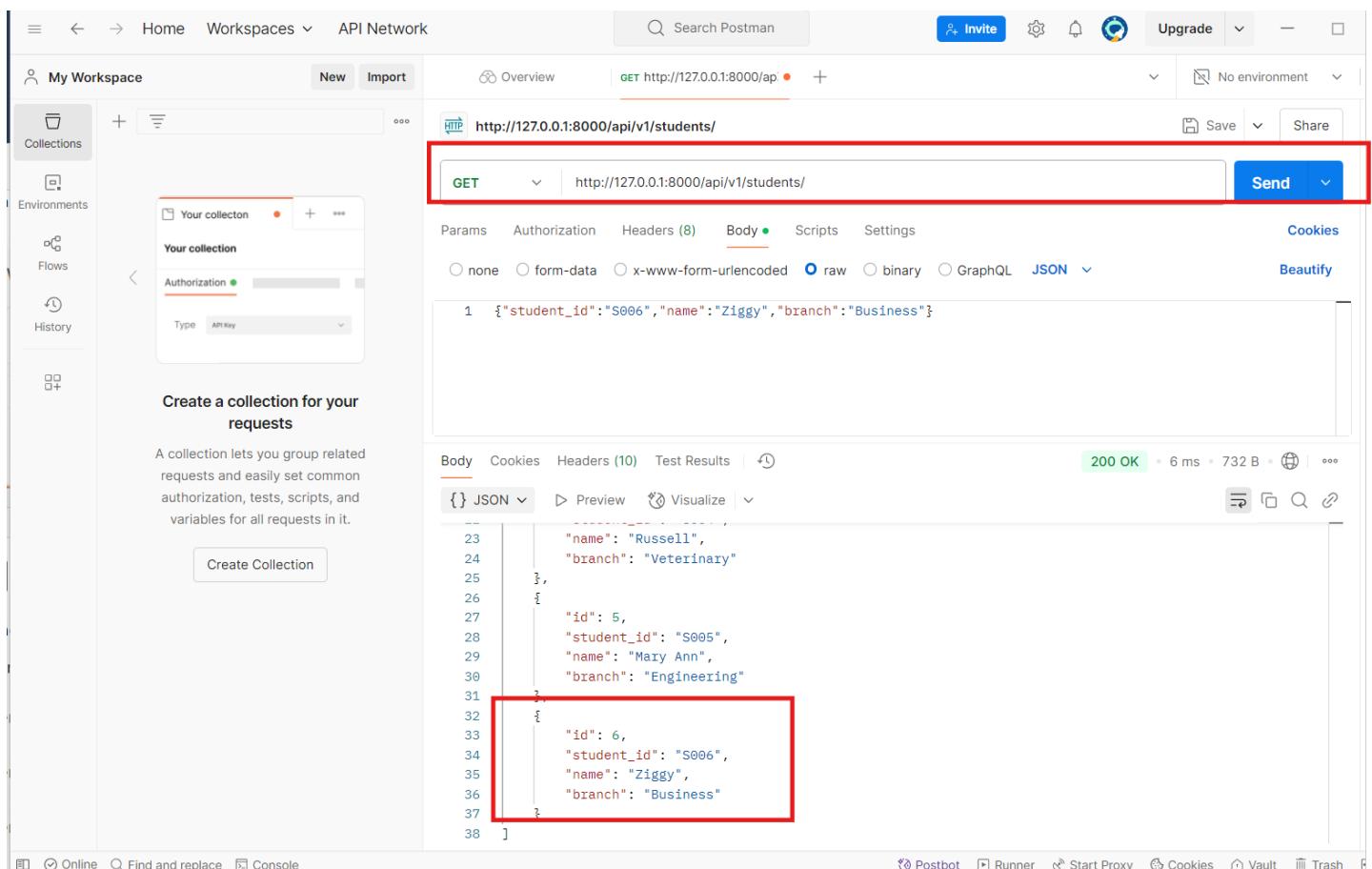


The screenshot shows the Postman interface with a collection named 'Your collection'. A POST request is made to `http://127.0.0.1:8000/api/v1/students/`. The request body is set to 'raw' and 'JSON' format, containing the following data:

```
1 {"student_id": "S006", "name": "Ziggy", "branch": "Business"}
```

The response status is `201 Created` with a response time of 18 ms and a response size of 391 B. The response body shows the newly created student record with `id: 6`.

9. Now, to see the newly inserted record, use the GET method. You will then see the newly added record.



The screenshot shows the Postman interface with the same collection. A GET request is made to `http://127.0.0.1:8000/api/v1/students/`. The request body is set to 'raw' and 'JSON' format, containing the same data as the previous POST request.

The response status is `200 OK` with a response time of 6 ms and a response size of 732 B. The response body shows a list of students, including the newly added record with `id: 6`.