

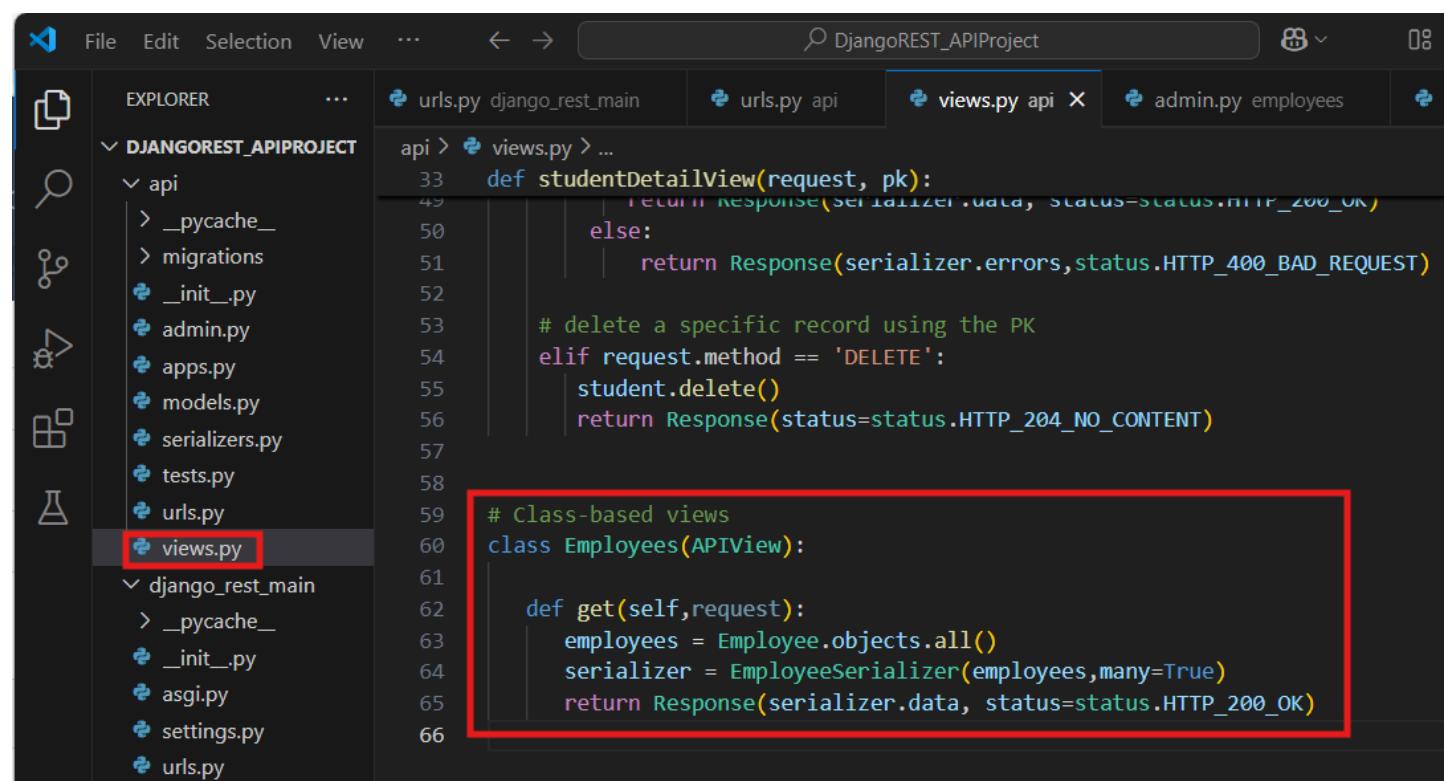
Topic: 8. Updating the Models using Class-Based Views

Speaker: / Notebook: API Development using Django Framework



We previously created a new app called EMPLOYEES and a new Employee serializer for our Employee class.

1. Previously, we updated the API\VIEWS.PY to create an EMPLOYEE class and its methods:



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 DJANGOREST_APIPROJECT folder containing api, django_rest_main, and other files. The views.py file in the api folder is the active tab, highlighted with a red box around its content. The code in views.py defines a class-based view for the Employee model, specifically the Employees class, which handles the GET method to return all employees as a JSON response.

```
api > views.py > ...
33     def studentDetailView(request, pk):
34         serializer = StudentSerializer(data)
35         if serializer.is_valid():
36             return Response(serializer.data, status=status.HTTP_200_OK)
37         else:
38             return Response(serializer.errors, status.HTTP_400_BAD_REQUEST)
39
40     # delete a specific record using the PK
41     elif request.method == 'DELETE':
42         student.delete()
43         return Response(status=status.HTTP_204_NO_CONTENT)
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59     # Class-based views
60     class Employees(APIView):
61
62         def get(self, request):
63             employees = Employee.objects.all()
64             serializer = EmployeeSerializer(employees, many=True)
65             return Response(serializer.data, status=status.HTTP_200_OK)
66
```

We run the new path:

Employees

OPTIONS GET

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

[
    {
        "id": 1,
        "emp_id": "EMP001",
        "emp_name": "Rosilie Lim",
        "designation": "Software Developer"
    },
    {
        "id": 2,
        "emp_id": "EMP002",
        "emp_name": "Jane Doe",
        "designation": "Web Designer"
    }
]
```

2. To post or add a new record to our model:

```
def studentDetailView(request, pk):
    return Response(status=status.HTTP_204_NO_CONTENT)

class Employees(APIView):
    def get(self, request):
        employees = Employee.objects.all()
        serializer = EmployeeSerializer(employees, many=True)
        return Response(serializer.data, status=status.HTTP_200_OK)

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

Add the new record. Follow the correct format to be able to save successfully.

127.0.0.1:8000/api/v1/employees/

Django REST framework

Employees

OPTIONS GET

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

[
    {
        "id": 1,
        "emp_id": "EMP001",
        "emp_name": "Rosilie Lim",
        "designation": "Software Developer"
    },
    {
        "id": 2,
        "emp_id": "EMP002",
        "emp_name": "Jane Doe",
        "designation": "Web Designer"
    }
]
```

Media type: application/json

Content:

```
{
    "emp_id": "EMP003",
    "emp_name": "Russell Lim",
    "designation": "Security"
}
```

POST

This will result to:

→ ⌂ ⓘ 127.0.0.1:8000/api/v1/employees/ 🔎 ⭐ ⚙️ ⌂ ⌂ ⌂

Django REST framework api_djangoadmin

Employees

OPTIONS GET ▾

POST /api/v1/employees/

HTTP 201 Created

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

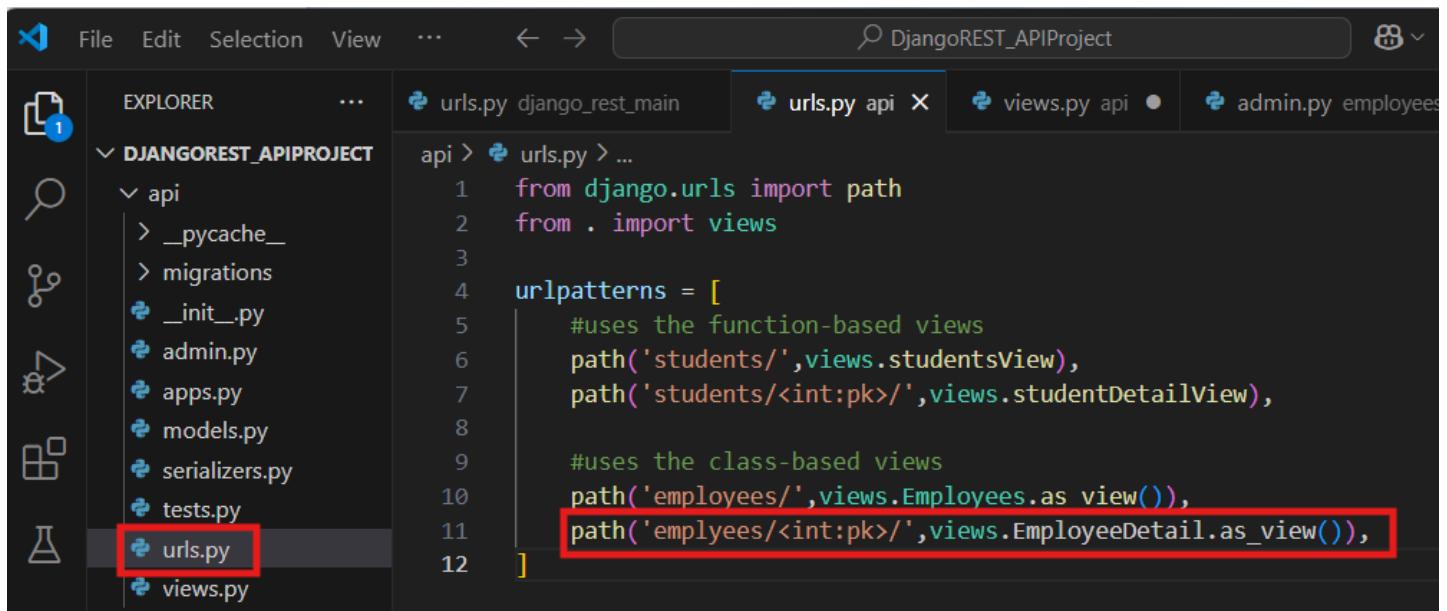
```
{  
    "id": 3,  
    "emp_id": "EMP003",  
    "emp_name": "Russell Lim",  
    "designation": "Security"  
}
```

Media type: application/json

Content:

POST

3. To allow a single record operation like CRUD on a specific record, we update our API\URLS.PY as:



The screenshot shows a code editor with the following structure:

- File: File, Edit, Selection, View, ...
- Toolbar: Back, Forward, Search, etc.
- Search bar: DjangoREST_APIProject
- Explorer sidebar (left):
 - EXPLORER
 - ...
 - DJANGOREST_APIPROJECT
 - api
 - __pycache__
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - models.py
 - serializers.py
 - tests.py
 - urls.py
 - views.py- Code editor (right):

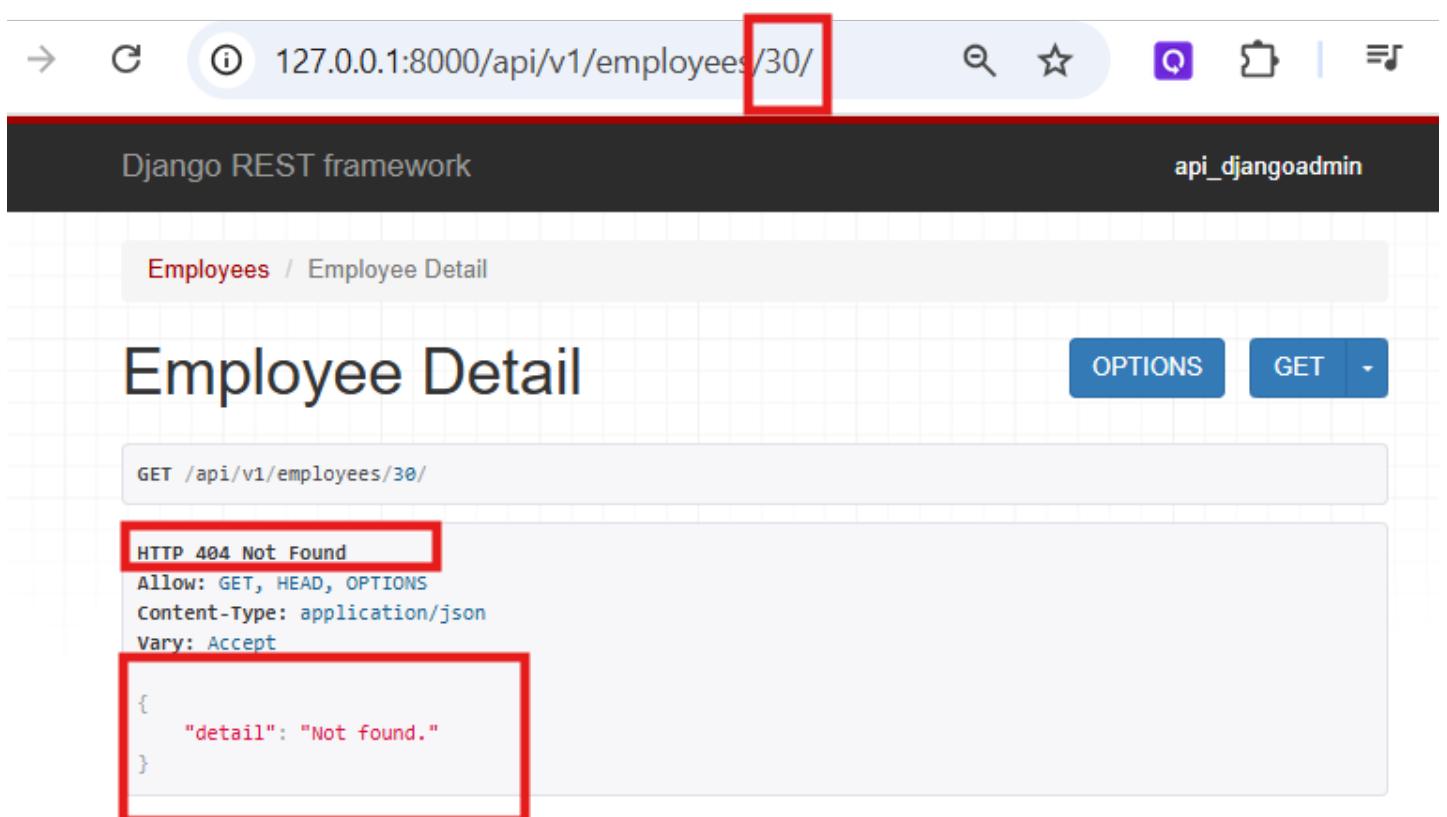
```
api > urls.py > ...
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      #uses the function-based views
6      path('students/',views.studentsView),
7      path('students/<int:pk>',views.studentDetailView),
8
9      #uses the class-based views
10     path('employees/',views.Employees.as_view()),
11     path('employees/<int:pk>',views.EmployeeDetail.as_view()),
12 ]
```

4. Update the API\VIEWS.PY to include EmployeeDetail class:

To test, add the path:

<http://127.0.0.1:8000/api/v1/employees/30/>

And if the record is not in the table model:



The browser screenshot shows the following details:

- Address bar: 127.0.0.1:8000/api/v1/employees/30/ (highlighted with a red box)
- Header: Django REST framework
- Header: api_djangoadmin
- Page title: Employees / Employee Detail
- Content:
 - OPTIONS
 - GET

GET /api/v1/employees/30/

HTTP 404 Not Found

Allow: GET, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{ "detail": "Not found." }
```

5. To update a single record, update the API\VIEWS.PY as:

The screenshot shows a browser window on the left and a VS Code interface on the right. The browser window displays a 'Employee Detail' page for an employee with ID 3, showing a JSON response with fields: emp_id, emp_name, and designation. The VS Code interface shows the 'views.py' file for the 'EmployeeDetail' view. The 'PUT' method is defined as follows:

```

# adds a new record
def put(self, request, pk):
    employee = self.get_object(pk)
    serializer = EmployeeSerializer(employee, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_200_OK)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

```

This creates the form immediately but this causes an error when you submit the PUT button. So update it as:

The screenshot shows the VS Code interface with the 'views.py' file open. The 'PUT' method has been updated to handle existing records:

```

# adds a new record
def put(self, request, pk):
    employee = self.get_object(pk)
    serializer = EmployeeSerializer(employee, data=request.data)
    if serializer.is_valid():
        serializer.save()
        return Response(serializer.data, status=status.HTTP_200_OK)
    return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

```

Add the new record:

← → C 127.0.0.1:8000/api/v1/employees/ 🔍 ⭐ Q 🗂️ 🎵 🎨

Django REST framework

api_djangoadmin

Employees

OPTIONS GET

GET /api/v1/employees/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

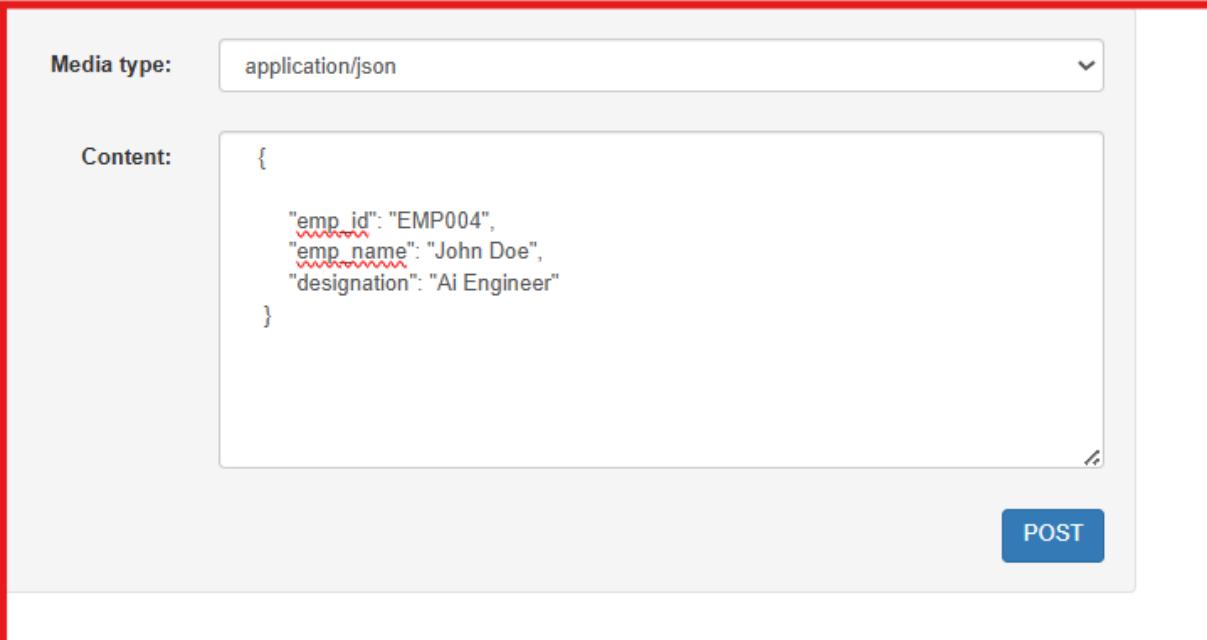
Vary: Accept

```
[  
  {  
    "id": 1,  
    "emp_id": "EMP001",  
    "emp_name": "Rosilie Lim",  
    "designation": "Software Developer"  
  },  
  {  
    "id": 2,  
    "emp_id": "EMP002",  
    "emp_name": "Jane Doe",  
    "designation": "Web Designer"  
  },  
  {  
    "id": 3,  
    "emp_id": "EMP003",  
    "emp_name": "Russell Lim",  
    "designation": "Security"  
  }  
]
```

Media type: application/json

Content: {
 "emp_id": "EMP004",
 "emp_name": "John Doe",
 "designation": "Ai Engineer"
}

POST



The screenshot shows the Postman application interface. On the left, a sidebar titled 'My Workspace' contains sections for 'Collections', 'Environments', 'Flows', and 'History'. A central dialog box is titled 'Create a collection for your requests' and includes a 'Create Collection' button. The main workspace shows an 'Overview' tab and a selected 'HTTP' request tab with the URL `http://127.0.0.1:8000/api/v1/employees/3`. The 'Params' tab of the request configuration is highlighted with a red box. The 'Body' tab of the response view is also highlighted with a red box, showing a JSON response with the following data:

```
1 {  
2   "id": 3,  
3   "emp_id": "EMP003",  
4   "emp_name": "Russell Lim",  
5   "designation": "Security"  
6 }
```

Using POSTMAN to store, choose POST, then select BODY, then RAW, then JSON. Add your records then select the SEND method.

The screenshot shows the Postman application interface. On the left, the sidebar includes 'My Workspace' with 'Collections' (empty), 'Environments' (empty), 'Flows' (empty), and 'History' (empty). Below this is a section to 'Create a collection for your requests' with a 'Create Collection' button. The main workspace shows an 'Overview' tab and a 'POST http://127.0.0.1:8001/api/v1/employees/' request. The 'Body' tab is selected, showing the raw JSON input:

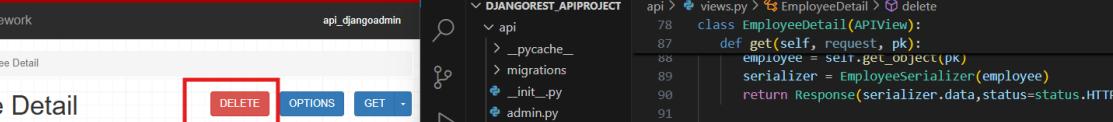
```
{ "emp_id": "EMP005", "emp_name": "Tam Delilah", "designation": "Marketing Analyst"}
```

. The 'Body' tab is highlighted with a red box. The 'Headers' tab shows 8 items. The 'Send' button is at the top right. The bottom right shows the response: **201 Created**, 28 ms, 419 B, with a preview of the JSON response:

```
{ "id": 5, "emp_id": "EMP005", "emp_name": "Tam Delilah", "designation": "Marketing Analyst"}
```

. The 'Send' button and the JSON response preview are also highlighted with red boxes.

7. To delete a record, create a **DELETE** method and update as:



Django REST framework api_djangoadmin

Employees / Employee Detail

Employee Detail

DELETE OPTIONS GET

GET /api/v1/employees/5/

HTTP 200 OK

Allow: GET, PUT, DELETE, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

{

 "Id": 5,

 "emp_id": "EMP005",

 "emp_name": "Tariq Delilah",

 "designation": "Marketing Analyst"

}

Media type: application/json

Content:

EXPLORER

urls.py django_rest_main urls.py api views.py api x admin.py employees

...

▼ DJANGOREST_APIPROJECT

 ▼ api

 > __pycache__

 > migrations

 > __init__.py

 > admin.py

 > apps.py

 > models.py

 > serializers.py

 > tests.py

 > urls.py

 > views.py

 ▼ django_rest_main

 > __pycache__

 > __init__.py

 > asgi.py

 > settings.py

 > urls.py

 > wsgi.py

 > urls.py

 > views.py

api > views.py > EmployeeDetail > delete

78 class EmployeeDetail(APIView):

79 def get(self, request, pk):

80 employee = self.get_object(pk)

81 serializer = EmployeeSerializer(employee)

82 return Response(serializer.data, status=status.HTTP_200_OK)

83 # adds a new record

84 def put(self, request, pk):

85 employee = self.get_object(pk)

86 serializer = EmployeeSerializer(employee, data=request.data)

87 if serializer.is_valid():

88 serializer.save()

89 return Response(serializer.data, status=status.HTTP_200_OK)

90 # deletes a record

91 def delete(self, request, pk):

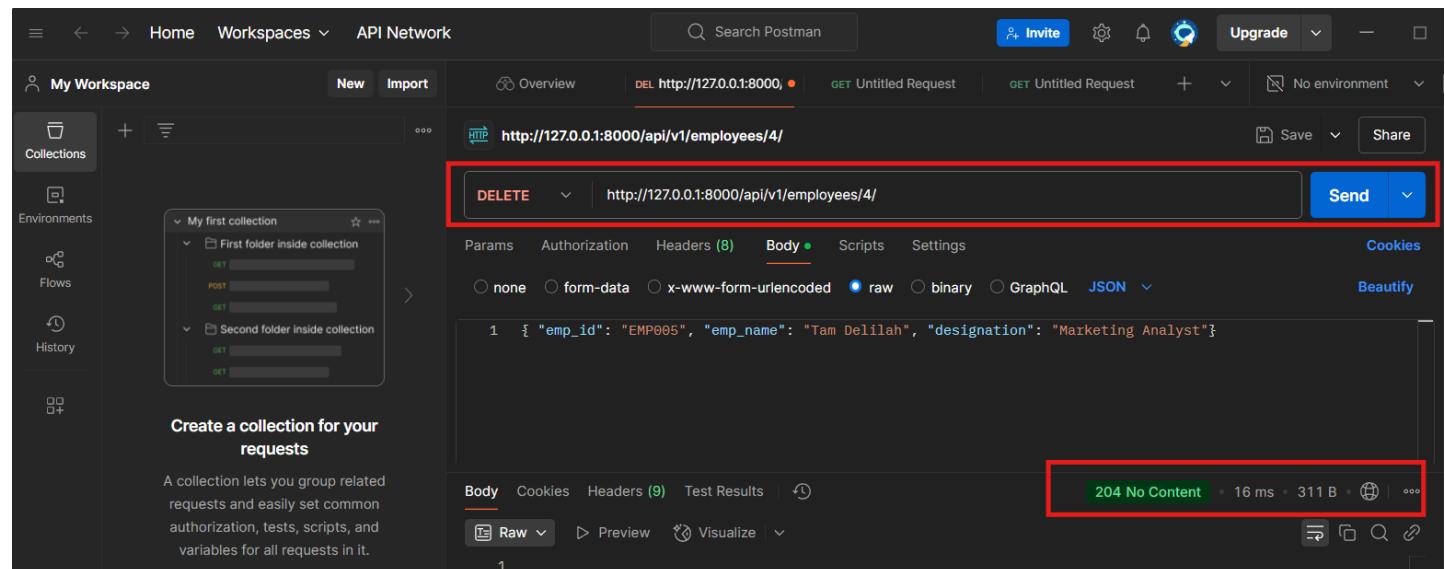
92 employee = self.get_object(pk)

93 employee.delete()

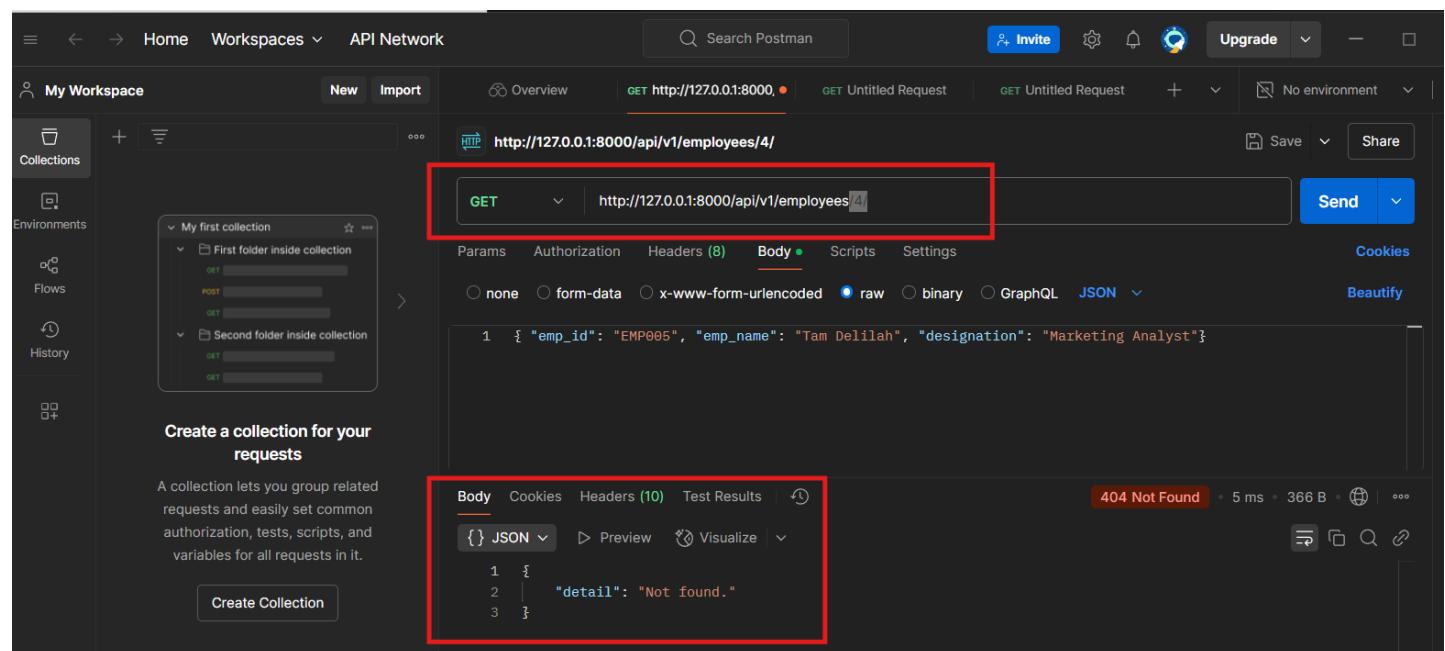
94 return Response(status=status.HTTP_204_NO_CONTENT)

We deleted record ID = 5.

8. In POSTMAN, we can also issue a delete option:



The screenshot shows the Postman interface with a DELETE request to `http://127.0.0.1:8000/api/v1/employees/4/`. The request body is a JSON object with fields `emp_id`, `emp_name`, and `designation`. The response status is **204 No Content**, indicating the record was successfully deleted.



The screenshot shows the Postman interface with a GET request to `http://127.0.0.1:8000/api/v1/employees/4/`. The request body is a JSON object with fields `emp_id`, `emp_name`, and `designation`. The response status is **404 Not Found**, indicating the record was not found.