

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 APIVIEWS.PY to create an EMPLOYEE class and its methods:

```
33 def studentDetailView(request, pk):
49     return Response(serializer.data, status=status.HTTP_200_OK)
50     else:
51         return Response(serializer.errors, status.HTTP_400_BAD_REQUEST)
52
53     # delete a specific record using the PK
54     elif request.method == 'DELETE':
55         student.delete()
56         return Response(status=status.HTTP_204_NO_CONTENT)
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:

127.0.0.1:8000/api/v1/employees/

Django REST framework api_djangoadmin

Employees

OPTIONS GET

GET /api/v1/employees/

```
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:

DjangoREST_APIProject

EXPLORER

- DJANGOREST_APIPROJECT
 - api
 - views.py

```
33 def studentDetailView(request, pk):
34     ...
35     return Response(status=status.HTTP_204_NO_CONTENT)
36
37
38 # Class-based views
39 class Employees(APIView):
40
41     # retrieves all records
42     def get(self, request):
43         employees = Employee.objects.all()
44         serializer = EmployeeSerializer(employees, many=True)
45         return Response(serializer.data, status=status.HTTP_200_OK)
46
47
48 # stores a new record
49 def post(self, request):
50     serializer = EmployeeSerializer(data=request.data)
51     if serializer.is_valid():
52         serializer.save()
53         return Response(serializer.data, status=status.HTTP_201_CREATED)
54     return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
55
```

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

The screenshot shows a web browser at the URL `127.0.0.1:8000/api/v1/employees/`. The page title is "Django REST framework" and the user is logged in as "api_djangoadmin". The main heading is "Employees".

On the right side, there are two buttons: "OPTIONS" and "GET". Below them, the HTTP method and URL are shown as "GET /api/v1/employees/".

The response body is displayed in a code block:

```
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"
  }
]
```

Below the response, there is a form for creating a new record. The "Media type" is set to "application/json". The "Content" field contains the following JSON:

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

A "POST" button is located at the bottom right of the form, highlighted with a red box.

This will result to:

Employees

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 APIURLS.PY as:

The screenshot shows the Visual Studio Code editor with a Django REST API project. The Explorer sidebar on the left shows the project structure, with `urls.py` highlighted. The main editor window displays the `urls.py` file for the `api` app. The code includes imports for `path` and `views`, and a list of URL patterns. The following line is highlighted with a red box:

```
path('employees/<int:pk>', views.EmployeeDetail.as_view()),
```

4. Update the APIVIEWS.PY to include EmployeeDetail class:

To test, add the path:

`http://127.0.0.1:8000/api/v1/employees/3/`

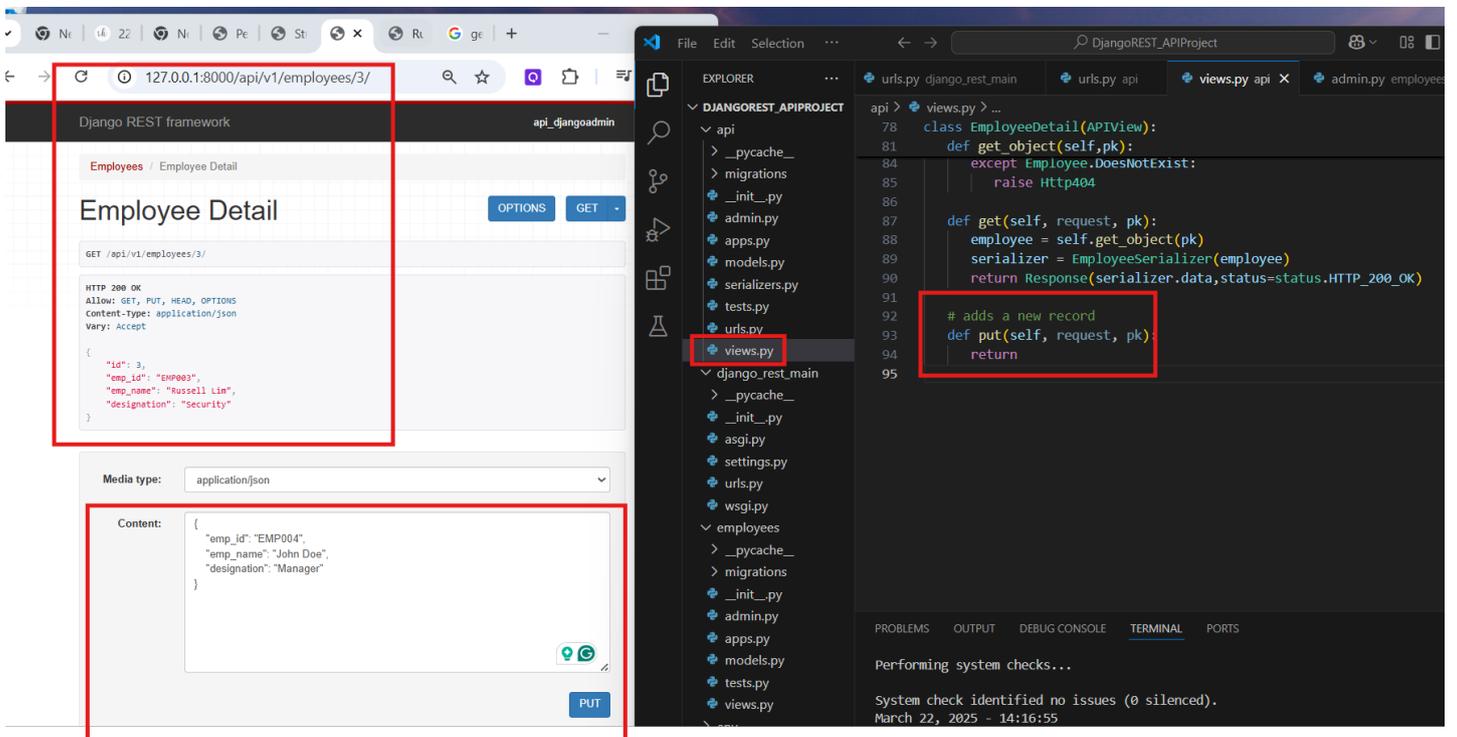
And if the record is not in the table model:

The screenshot shows a web browser window with the address bar containing `127.0.0.1:8000/api/v1/employees/30/`, which is highlighted with a red box. The page title is "Django REST framework" and the user is logged in as "api_djangoadmin". The breadcrumb navigation shows "Employees / Employee Detail". The main heading is "Employee Detail" with "OPTIONS" and "GET" buttons. Below the heading, the request details are shown: "GET /api/v1/employees/30/". The response is highlighted with a red box and shows:

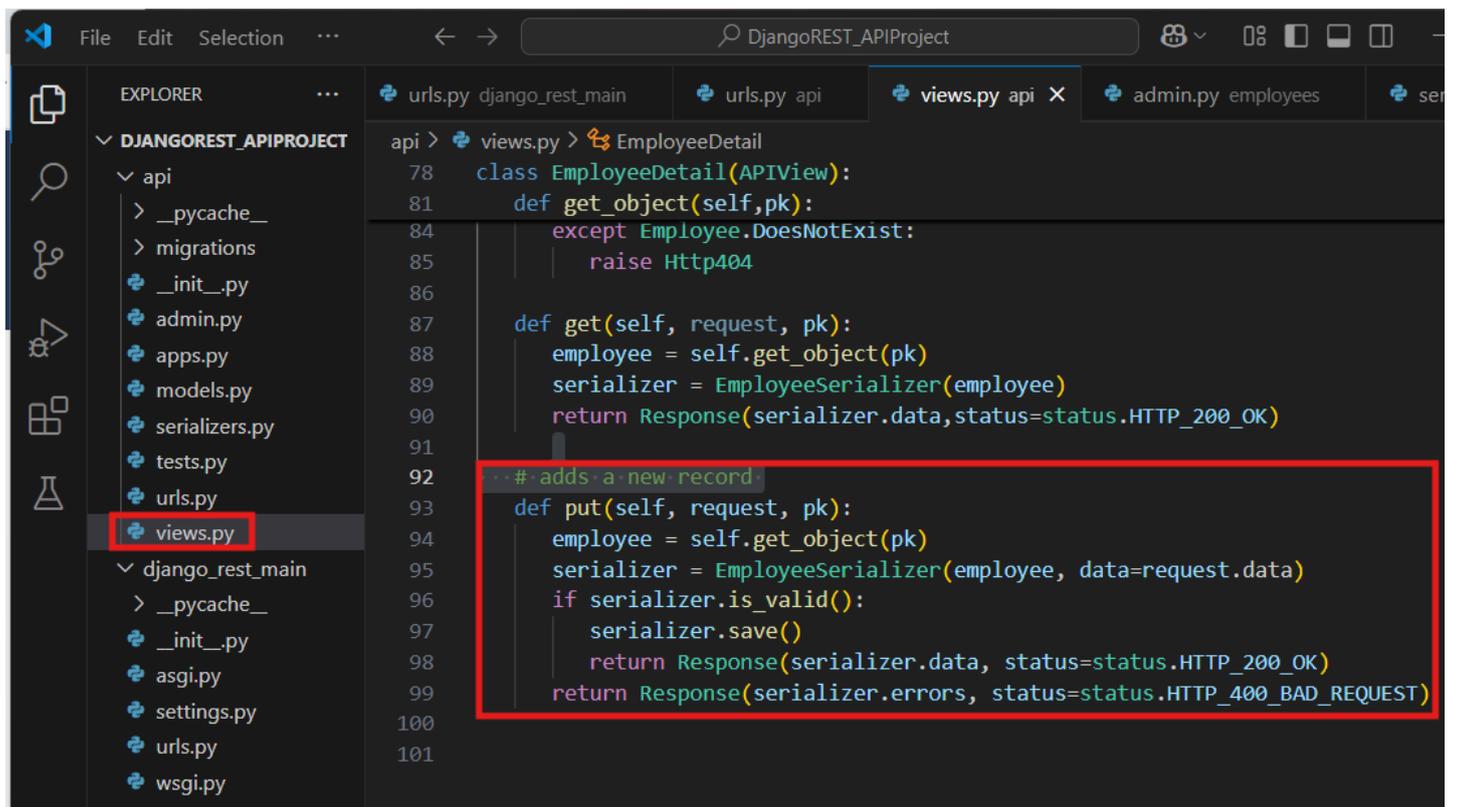
```
HTTP 404 Not Found
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

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

5. To update a single record, update the APIVIEWS.PY as:



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



Add the new record:

Employees

Employees

OPTIONS

GET

GET /api/v1/employees/

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

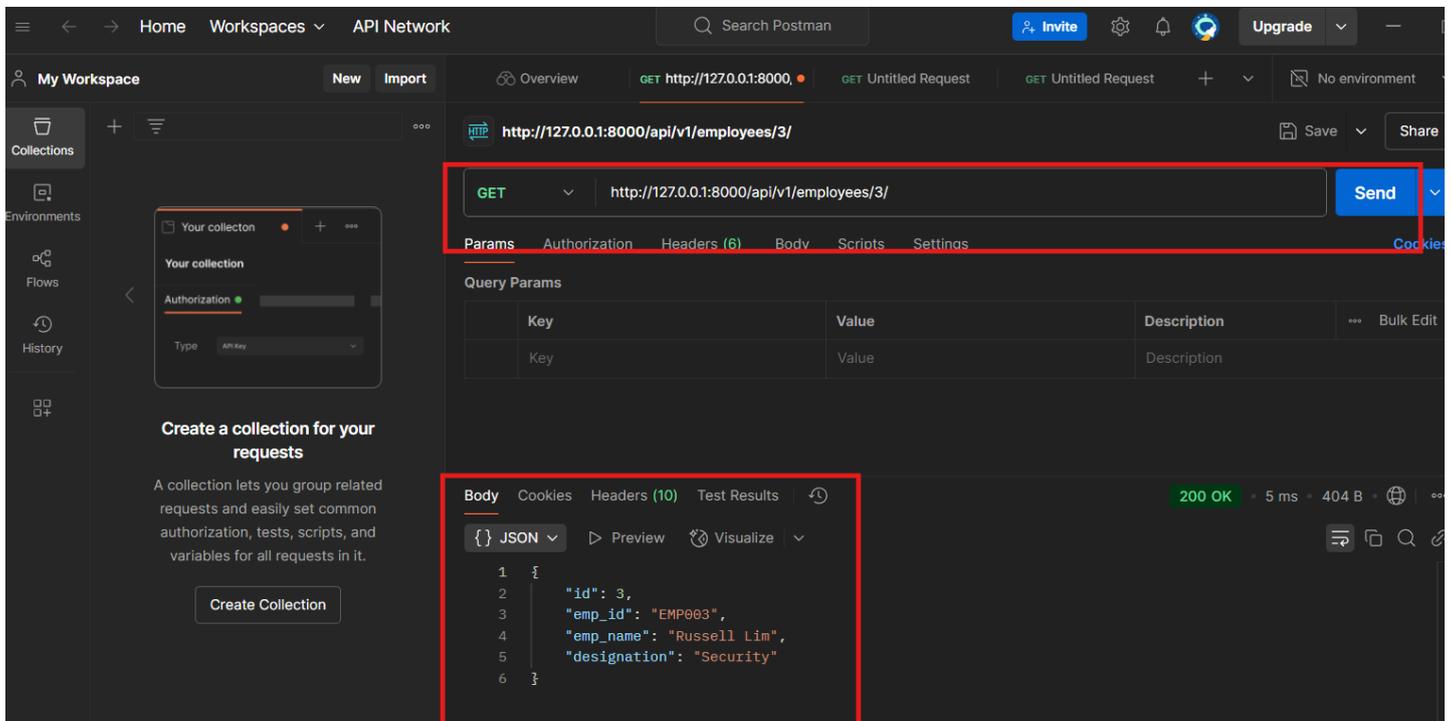
[
  {
    "id": 1,
    "emp_id": "EMP001",
    "emp_name": "Rosilie Lim",
    "designation": "Software Developer"
  },
  {
    "id": 2,
    "emp_id": "EMP002",
    "emp_name": "Jane Doe",
    "designation": "Web Designer"
  },
  {
    "id": 3,
    "emp_id": "EMP003",
    "emp_name": "Russell Lim",
 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

Media type:

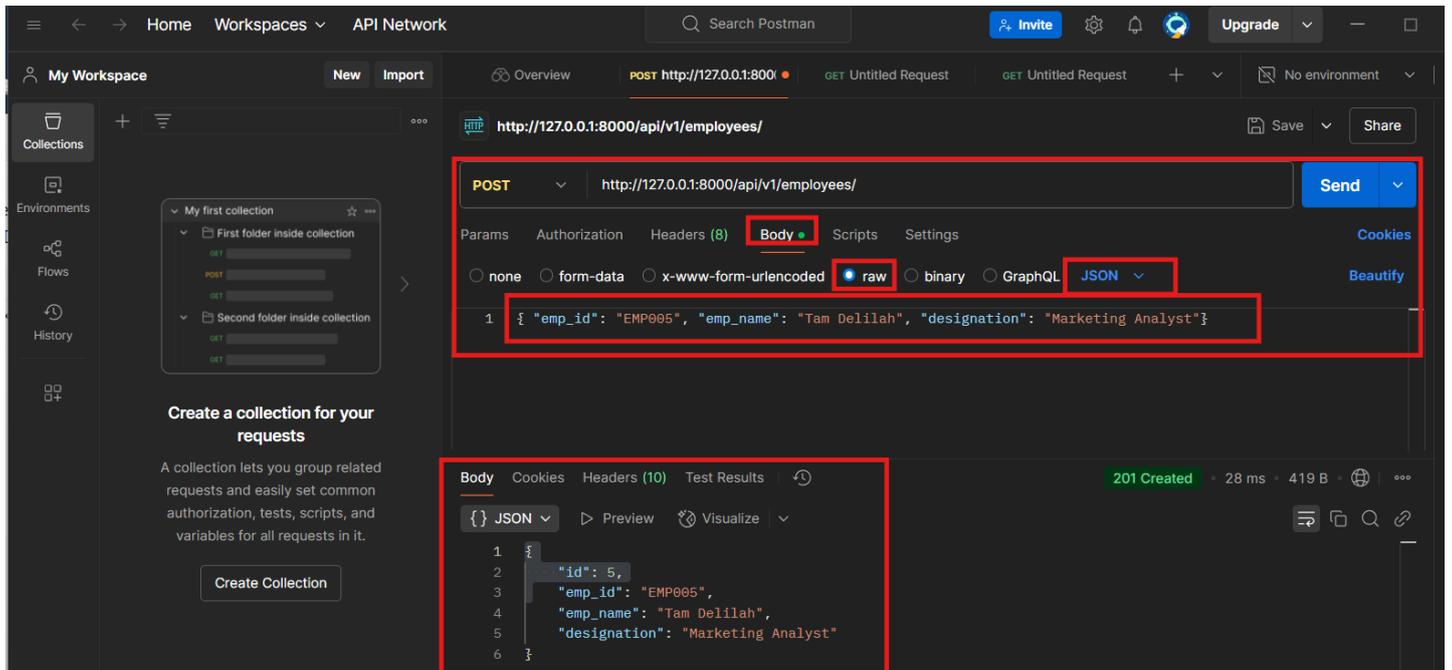
Content:

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

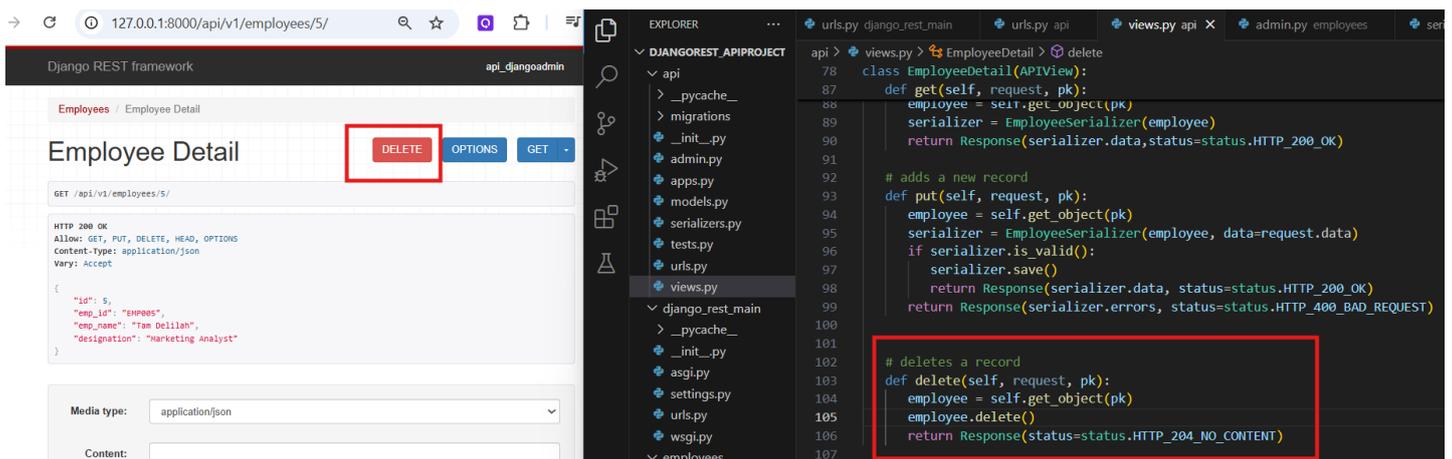
POST



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



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



We deleted record ID = 5.

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

