

Topic: 11. CRUD using Viewsets

Speaker: Personal / Notebook: API Development using Django Framework



According to Django Documentation:

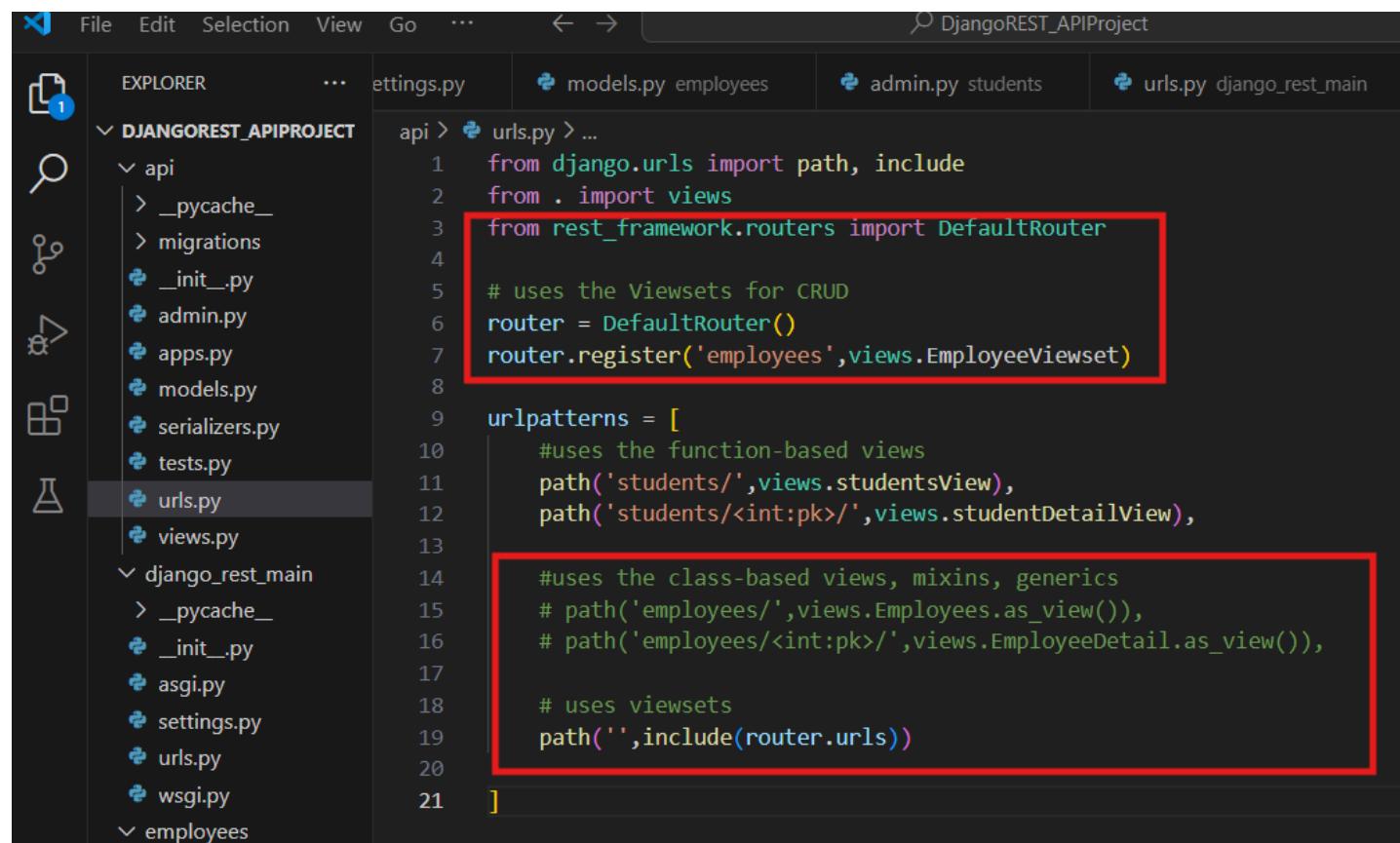
Django REST framework allows you to combine the logic for a set of related views in a single class, called a `ViewSet`. In other frameworks, you may also find conceptually similar implementations named 'Resources' or 'Controllers'.

A `ViewSet` class is simply a type of class-based View, that does not provide any method handlers such as `.get()` or `.post()`, and instead provides actions such as `.list()` and `.create()`.

The method handlers for a `ViewSet` are only bound to the corresponding actions at the point of finalizing the view, using the `.as_view()` method.

1. We use `viewsets.ViewSet` class for `list()`, `retrieve()`, `update()`, `create`, `delete()`. The `viewsets.ModelViewSet` uses only `queryset` and `serializer_class` and provides for pk and non-pk-based operations.

2. To use Viewsets, we use ROUTERS with the paths that we normally use in our Django project. So we comment on the paths we used with function-based, class-based, mixins, and generics in our URLs.PY.

A screenshot of the Visual Studio Code interface showing a Django REST API project. The Explorer sidebar on the left shows the project structure: 'DJANGOREST_APIPROJECT' with 'api' and 'django_rest_main' subfolders, and 'employees' and 'views' files. The 'urls.py' file in the 'api' folder is open in the editor. The code uses a `DefaultRouter` to register a `EmployeeViewSet`. It then defines a list of URL patterns, including paths for 'students' and 'students/', and a path for 'employees' using a `EmployeeViewSet`. The code is highlighted with syntax coloring, and two sections of the 'urlpatterns' list are enclosed in red boxes.

```
from django.urls import path, include
from . import views
from rest_framework.routers import DefaultRouter

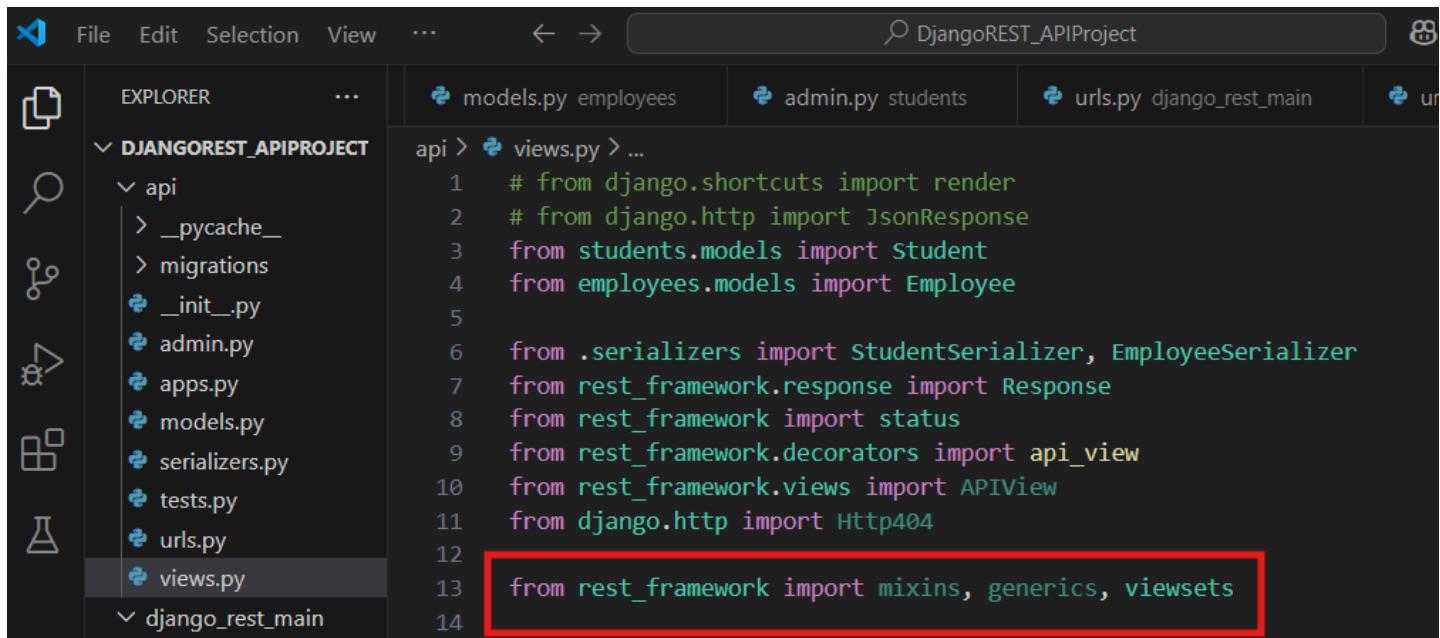
# uses the Viewsets for CRUD
router = DefaultRouter()
router.register('employees',views.EmployeeViewSet)

urlpatterns = [
    #uses the function-based views
    path('students/',views.studentsView),
    path('students/<int:pk>',views.studentDetailView),

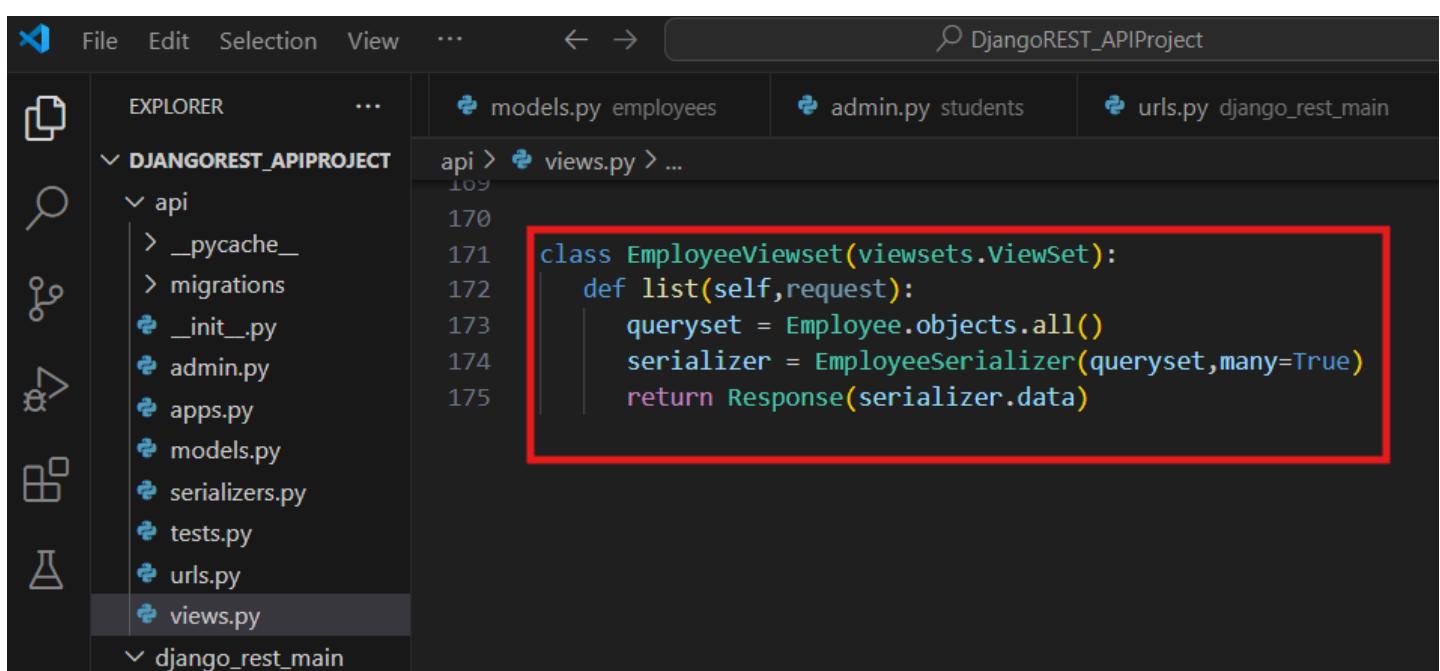
    #uses the class-based views, mixins, generics
    # path('employees/',views.Employees.as_view()),
    # path('employees/<int:pk>',views.EmployeeDetail.as_view()),

    # uses viewsets
    path('',include(router.urls))
]
```

3. Update the VIEWS.PY as:

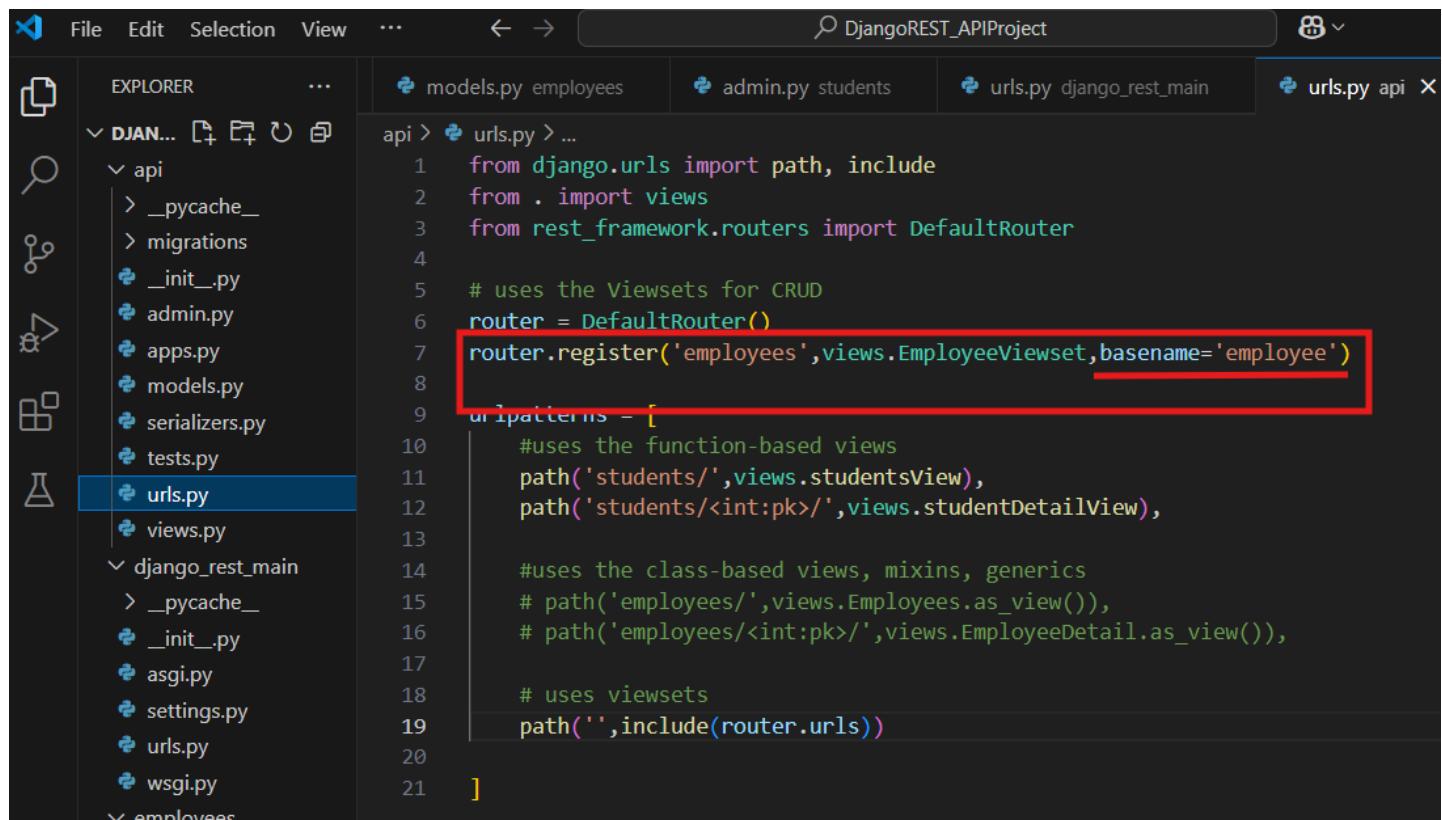


```
1 # from django.shortcuts import render
2 # from django.http import JsonResponse
3 from students.models import Student
4 from employees.models import Employee
5
6 from .serializers import StudentSerializer, EmployeeSerializer
7 from rest_framework.response import Response
8 from rest_framework import status
9 from rest_framework.decorators import api_view
10 from rest_framework.views import APIView
11 from django.http import Http404
12
13 from rest_framework import mixins, generics, viewsets
14
```



```
169
170
171 class EmployeeViewSet(viewsets.ViewSet):
172     def list(self, request):
173         queryset = Employee.objects.all()
174         serializer = EmployeeSerializer(queryset, many=True)
175         return Response(serializer.data)
```

We further update our URLs.PY to include the basename as the name of our model.



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` directory and its subfiles: `__init__.py`, `admin.py`, `apps.py`, `models.py`, `serializers.py`, `tests.py`, `urls.py`, and `views.py`. The `urls.py` file is selected and shown in the main editor area. The code in `urls.py` is as follows:

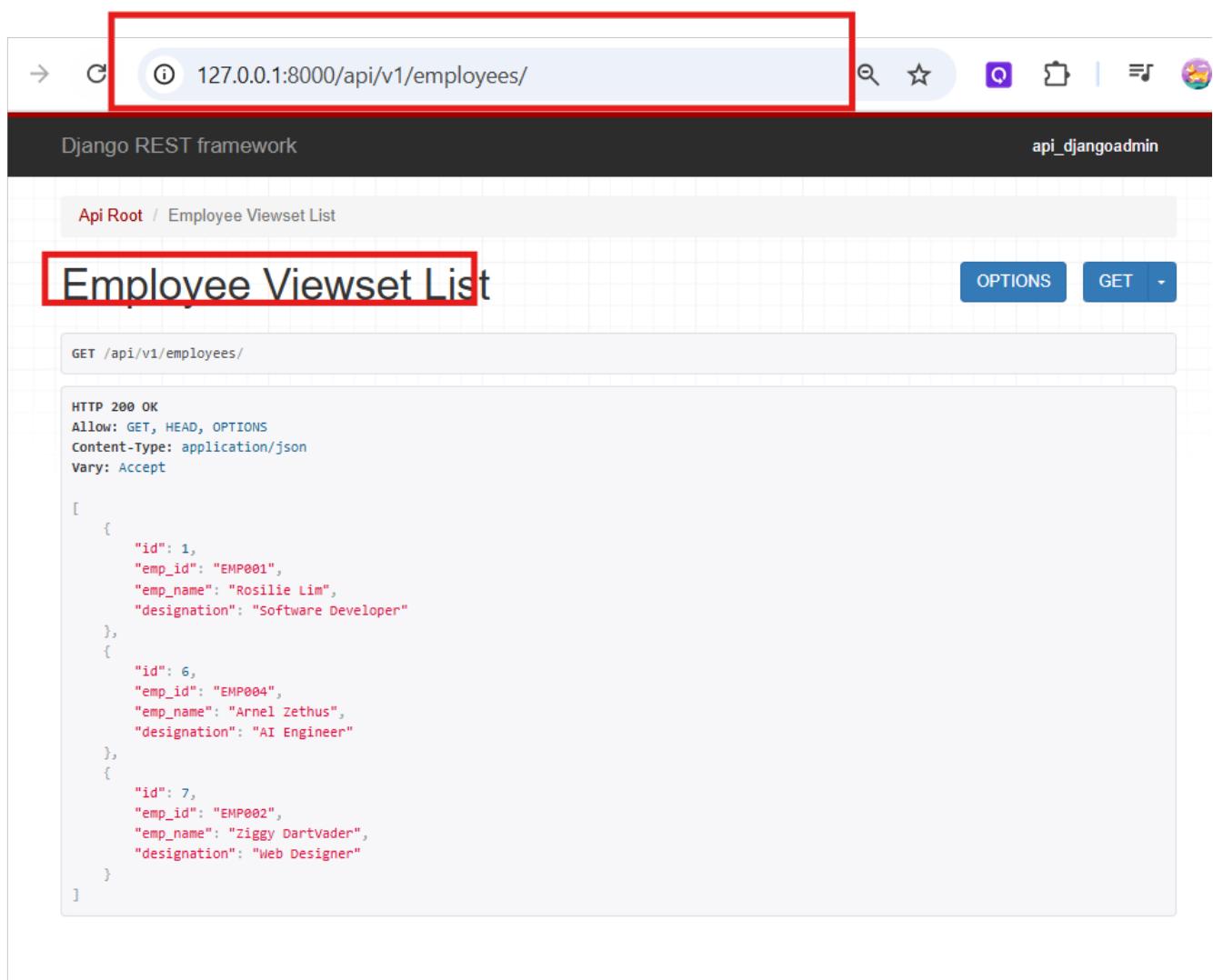
```

1  from django.urls import path, include
2  from . import views
3  from rest_framework.routers import DefaultRouter
4
5  # uses the Viewsets for CRUD
6  router = DefaultRouter()
7  router.register('employees', views.EmployeeViewSet, basename='employee')
8
9  urlpatterns = [
10     #uses the function-based views
11     path('students/', views.studentsView),
12     path('students/<int:pk>', views.studentDetailView),
13
14     #uses the class-based views, mixins, generics
15     # path('employees/', views.Employees.as_view()),
16     # path('employees/<int:pk>', views.EmployeeDetail.as_view()),
17
18     # uses viewsets
19     path('', include(router.urls))
20 ]
21

```

A red box highlights the line `router.register('employees', views.EmployeeViewSet, basename='employee')`.

So, when we run our path for Employees:



The screenshot shows a browser window with the URL `127.0.0.1:8000/api/v1/employees/` highlighted with a red box. The page title is "Django REST framework" and the user is logged in as "api_djangoadmin". The main content is titled "Employee Viewset List". Below the title, there is a "GET /api/v1/employees/" button and an "OPTIONS" button. The response body shows the JSON data for the 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": 6,
        "emp_id": "EMP004",
        "emp_name": "Arnel Zethus",
        "designation": "AI Engineer"
    },
    {
        "id": 7,
        "emp_id": "EMP002",
        "emp_name": "Ziggy DartVader",
        "designation": "Web Designer"
    }
]

```

4. For other CRUD operations, we create a new method, CREATE, to add a new record:

This results to:

→ G 127.0.0.1:8000/api/v1/employees/ ⚙️ ⚙️ ⚙️ ⚙️

Django REST framework api_djangoadmin

Api Root / Employee Viewset List

Employee Viewset List

OPTIONS GET

POST /api/v1/employees/

HTTP 201 Created
Allow: GET, POST, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

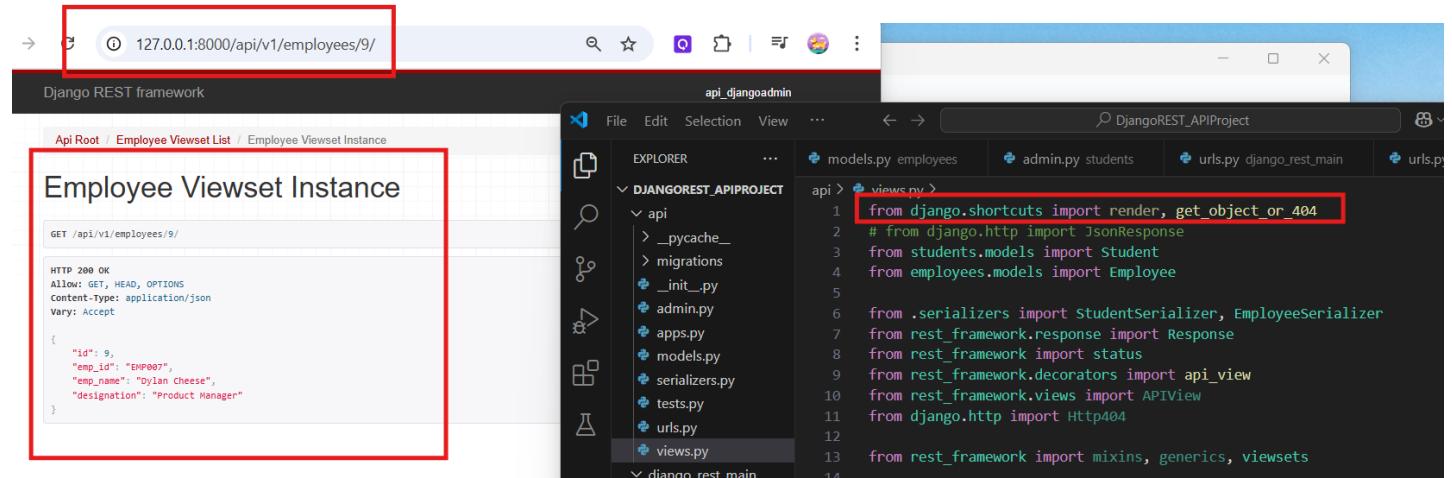
```
{  
    "id": 9,  
    "emp_id": "EMP007",  
    "emp_name": "Dylan Cheese",  
    "designation": "Product Manager"  
}
```

Media type: application/json

Content:

POST

5. To retrieve a single record, we don't have to create a separate path for our employee details, by using the retrieve method, we can get this record immediately.



Employee Viewset Instance

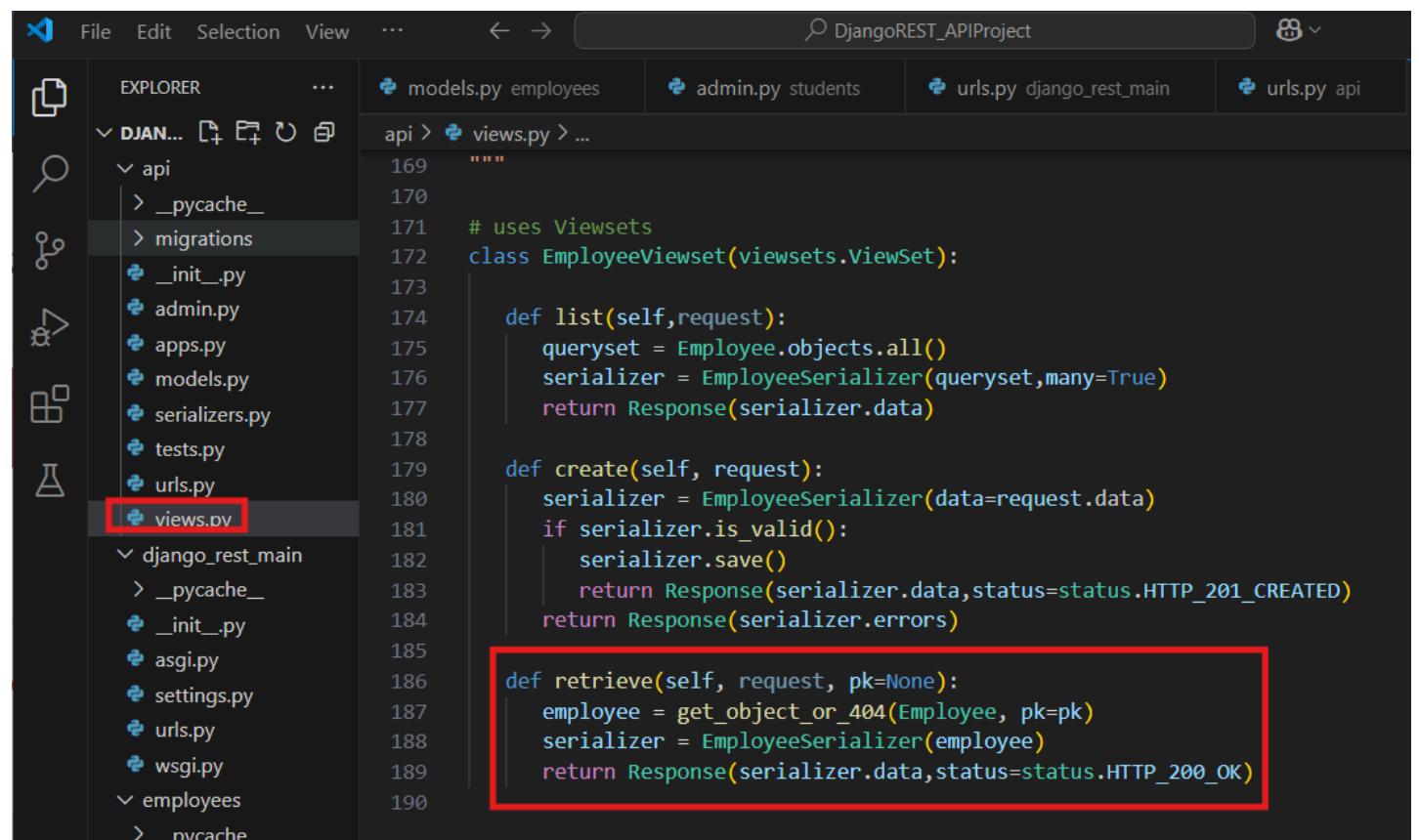
```
GET /api/v1/employees/9/
```

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

```
{"id": 9,  
"emp_id": "EMP007",  
"emp_name": "Dylan Cheese",  
"designation": "Product Manager"}
```

views.py

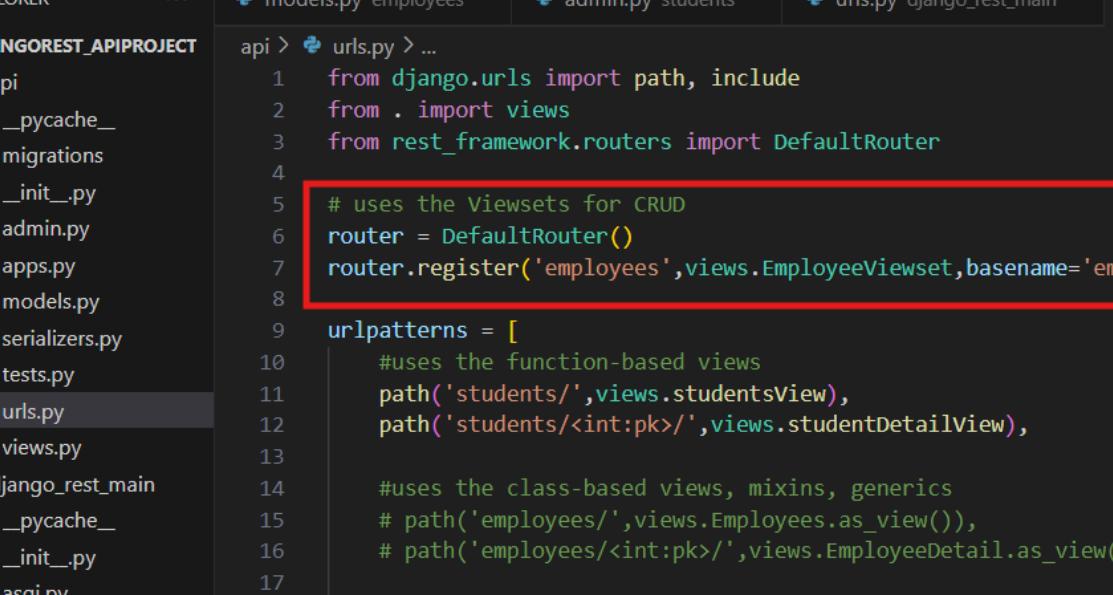
```
from django.shortcuts import render, get_object_or_404  
from django.http import JsonResponse  
from students.models import Student  
from employees.models import Employee  
from .serializers import StudentSerializer, EmployeeSerializer  
from rest_framework.response import Response  
from rest_framework import status  
from rest_framework.decorators import api_view  
from rest_framework.views import APIView  
from django.http import Http404  
from rest_framework import mixins, generics, viewsets
```



Employee Viewset Instance

```
class EmployeeViewSet(viewsets.ViewSet):  
    def list(self, request):  
        queryset = Employee.objects.all()  
        serializer = EmployeeSerializer(queryset, many=True)  
        return Response(serializer.data)  
  
    def create(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)  
  
    def retrieve(self, request, pk=None):  
        employee = get_object_or_404(Employee, pk=pk)  
        serializer = EmployeeSerializer(employee)  
        return Response(serializer.data, status=status.HTTP_200_OK)
```

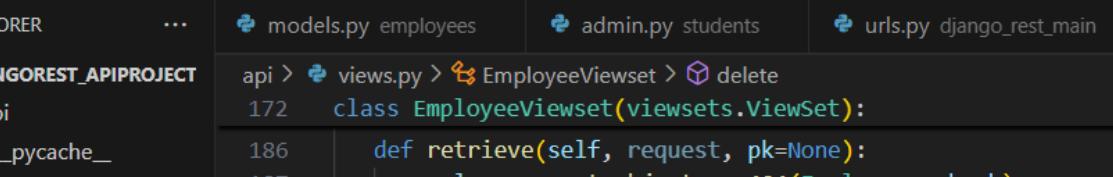
Our URLs.PY is still this:



The screenshot shows the VS Code interface with the Django REST API Project open. The Explorer sidebar on the left lists the project structure, including the `api` directory which contains `__init__.py`, `migrations`, `admin.py`, `apps.py`, `models.py`, `serializers.py`, `tests.py`, `urls.py` (selected), and `views.py`. The `urls.py` file in the `api` directory is displayed in the main editor. The code defines a `DefaultRouter` and registers the `EmployeeViewSet` with the name `'employee'`. It also defines patterns for `'students'` and `'employees'` using function-based and class-based views. A red box highlights the registration of the `EmployeeViewSet` and the inclusion of the router's URLs.

```
1  from django.urls import path, include
2  from . import views
3  from rest_framework.routers import DefaultRouter
4
5  # uses the Viewsets for CRUD
6  router = DefaultRouter()
7  router.register('employees',views.EmployeeViewSet, basename='employee')
8
9  urlpatterns = [
10     #uses the function-based views
11     path('students/',views.studentsView),
12     path('students/<int:pk>',views.studentDetailView),
13
14     #uses the class-based views, mixins, generics
15     # path('employees/',views.Employees.as_view()),
16     # path('employees/<int:pk>',views.EmployeeDetail.as_view()),
17
18     # uses viewsets
19     path('',include(router.urls))
20 ]
```

6. To delete a record, we create a new method:



```
api > views.py > EmployeeViewSet > delete
172 class EmployeeViewSet(viewsets.ViewSet):
186     def retrieve(self, request, pk=None):
187         employee = get_object_or_404(Employee, pk=pk)
188         serializer = EmployeeSerializer(employee)
189         return Response(serializer.data, status=status.HTTP_200_OK)
191
192     def delete(self, request, pk=None):
193         employee = get_object_or_404(Employee, pk=pk)
194         employee.delete()
195         return Response(status=status.HTTP_204_NO_CONTENT)
```

7. To update the record,

The screenshot shows a browser window at 127.0.0.1:8000/api/v1/employees/1 and a code editor for a Django REST API project named `DJANGOREST_APIPROJECT`.

Browser Response (Employee Viewset Instance):

```

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

{
  "id": 1,
  "emp_id": "EMP001",
  "emp_name": "Rosilie Lim",
  "designation": "Software Developer"
}
  
```

Code Editor (views.py):

```

class EmployeeViewSet(viewsets.ViewSet):
    ...
    def delete(self, request, pk=None):
        employee = get_object_or_404(Employee, pk=pk)
        employee.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)

    def update(self, request, pk=None):
        employee = get_object_or_404(Employee, pk=pk)
        serializer = EmployeeSerializer(employee, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_200_OK)
        else:
            return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
  
```

The JSON response in the browser and the `update` method in the code editor are highlighted with red boxes.

8. Our updated Employee model shall look like this:

Django REST framework api_djangoadmin

Api Root / Employee Viewset List

Employee Viewset List

DELETE OPTIONS GET ▾

`GET /api/v1/employees/`

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

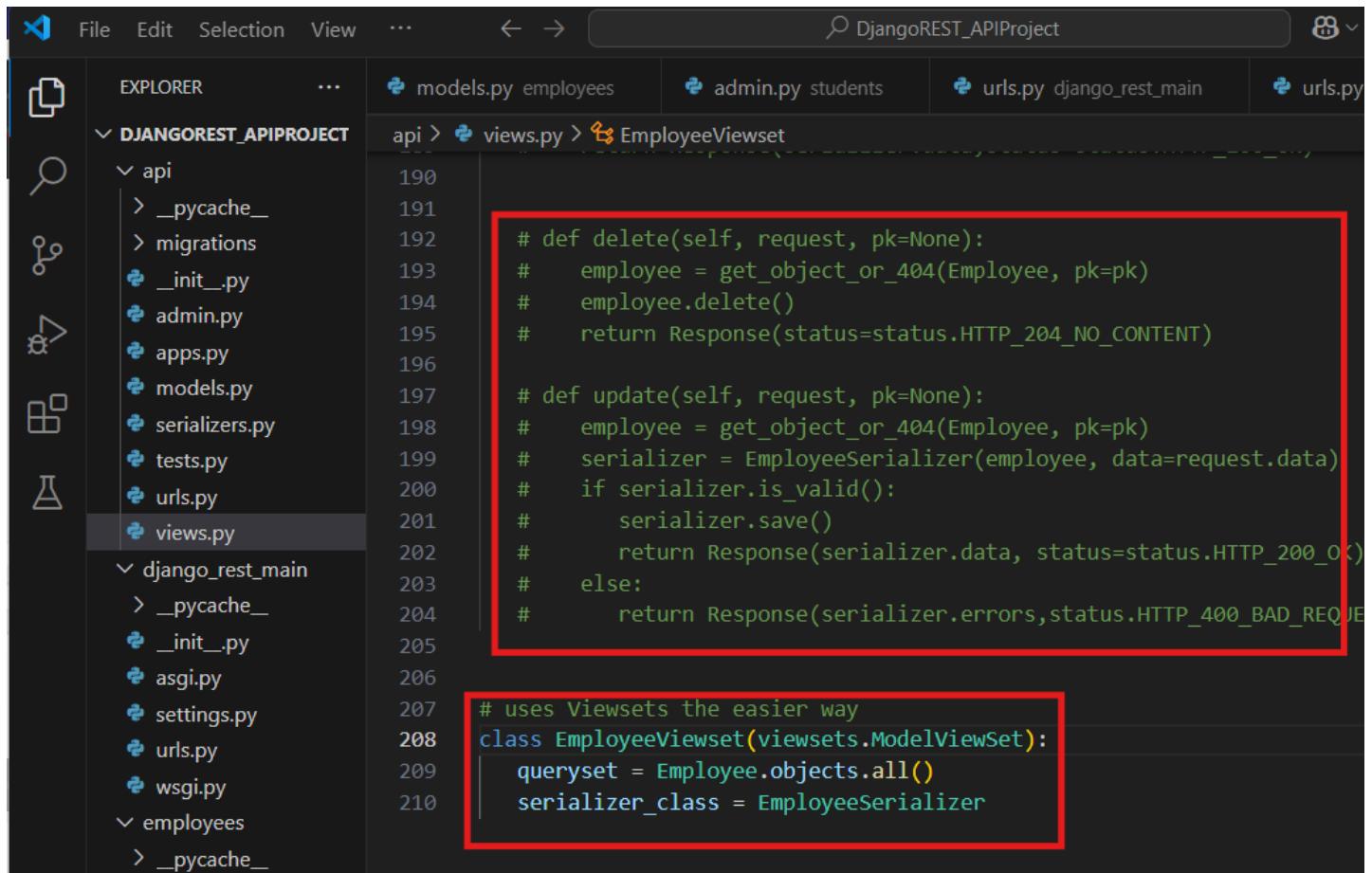
[{"id": 1, "emp_id": "EMP001", "emp_name": "Rosilie", "designation": "Software Developer"}, {"id": 6, "emp_id": "EMP004", "emp_name": "Arnel Zethus", "designation": "AI Engineer"}, {"id": 7, "emp_id": "EMP002", "emp_name": "Ziggy DartVader", "designation": "Web Designer"}]
```

Media type: application/json

Content:

POST

Using the steps above, we were able to create one class with several methods to perform CRUD operations. However, there is an easier alternative, the use of `viewsets.ModelViewSet`, which will handle all the CRUD operations an easier and more efficient way.



The screenshot shows a code editor interface with a sidebar and a main editor area. The sidebar on the left lists the project structure under 'DJANGOREST_APIPROJECT':

- api
- migrations
- __init__.py
- admin.py
- apps.py
- models.py
- serializers.py
- tests.py
- urls.py
- views.py** (highlighted)

The main editor area shows the content of 'views.py' for the 'EmployeeViewSet' class. The code is divided into two sections, each enclosed in a red box:

```
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210

# def delete(self, request, pk=None):
#     employee = get_object_or_404(Employee, pk=pk)
#     employee.delete()
#     return Response(status=status.HTTP_204_NO_CONTENT)

# def update(self, request, pk=None):
#     employee = get_object_or_404(Employee, pk=pk)
#     serializer = EmployeeSerializer(employee, data=request.data)
#     if serializer.is_valid():
#         serializer.save()
#         return Response(serializer.data, status=status.HTTP_200_OK)
#     else:
#         return Response(serializer.errors, status.HTTP_400_BAD_REQUEST)

# uses Viewsets the easier way
class EmployeeViewSet(viewsets.ModelViewSet):
    queryset = Employee.objects.all()
    serializer_class = EmployeeSerializer
```

To add a new record:

Employee Viewset List

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",
        "designation": "Software Developer"
    },
    {
        "id": 6,
        "emp_id": "EMP004",
        "emp_name": "Arnel Zethus",
        "designation": "AI Engineer"
    },
    {
        "id": 7,
        "emp_id": "EMP002",
        "emp_name": "Ziggy DartVader",
        "designation": "Web Designer"
    }
]
```

Raw data

HTML form

Emp id Emp name Designation

POST

To update a record:

[Api Root](#) / [Employee Viewset List](#) / Employee Viewset Instance

Employee Viewset Instance

[DELETE](#)[OPTIONS](#)[GET](#) ▾[GET /api/v1/employees/10/](#)

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

```
{  
    "id": 10,  
    "emp_id": "EMP003",  
    "emp_name": "Russell",  
    "designation": "Security"  
}
```

[Raw data](#)[HTML form](#)

Emp id

EMP003

Emp name

Russell

Designation

Security Officer

[PUT](#)

To delete a record:

→ C

① 127.0.0.1:8000/api/v1/employees/10/

🔍 ⭐

🔗

🖨️

≡



Django REST framework

api_djangoadmin

Api Root / Employee Viewset

Are you sure you want to delete this Employee Viewset Instance?

Cancel

Delete

OPTIONS

GET

Employee

PUT /api/v1/employees/10/

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

```
{  
    "id": 10,  
    "emp_id": "EMP003",  
    "emp_name": "Russell",  
    "designation": "Security Officer"  
}
```

Raw data

HTML form

Emp id

EMP003

Emp name

Russell

Designation

Security Officer

PUT

Viewing our updated list:

Api Root / Employee Viewset List

Employee Viewset List

[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",  
    "designation": "Software Developer"  
  },  
  {  
    "id": 6,  
    "emp_id": "EMP004",  
    "emp_name": "Arnel Zethus",  
    "designation": "AI Engineer"  
  },  
  {  
    "id": 7,  
    "emp_id": "EMP002",  
    "emp_name": "Ziggy DartVader",  
    "designation": "Web Designer"  
  }  
]
```

[Raw data](#)[HTML form](#)

Emp id

Emp name

Designation

[POST](#)

To view a non-existent record, it also shows validations:

The screenshot shows a browser window with the URL `127.0.0.1:8000/api/v1/employees/50/` highlighted with a red box. The page title is "Django REST framework" and the sub-page title is "Employee Viewset Instance". There are three buttons at the top right: "DELETE", "OPTIONS", and "GET". Below these are two sections: "Raw data" and "HTML form". The "Raw data" section contains the following JSON response:

```
{ "detail": "No Employee matches the given query." }
```

The "HTML form" section contains three input fields: "Emp id", "Emp name", and "Designation", each with a corresponding text input box. A "PUT" button is located to the right of the input fields. The "Raw data" and "HTML form" buttons are also visible above the input fields.

9. In summary, the `viewsets.ModelViewSet` is much faster than other methods.

Copyright © Personal Digital Notebooks | By Rosilie | Date Printed: Feb. 4, 2026, 5:53 a.m.