

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`.

```
File Edit Selection View Go ... DjangoREST_APIProject
EXPLORER
  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
      employees
  settings.py
  models.py employees
  admin.py students
  urls.py django_rest_main

api > urls.py > ...
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)
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 ]
```

3. Update the `VIEWS.PY` as:

The screenshot shows the Visual Studio Code editor interface for a Django REST API project. The Explorer sidebar on the left shows the project structure under 'DJANGOREST_APIPROJECT', with 'api' expanded to show files like 'views.py'. The main editor window displays the code in 'views.py', with line 13 highlighted in a red box: `from rest_framework import mixins, generics, viewsets`.

```
api > views.py > ...
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
```

The screenshot shows the Visual Studio Code editor interface for a Django REST API project. The Explorer sidebar on the left shows the project structure under 'DJANGOREST_APIPROJECT', with 'api' expanded to show files like 'views.py'. The main editor window displays the code in 'views.py', with lines 171-175 highlighted in a red box, showing the definition of the 'EmployeeViewSet' class and its 'list' method.

```
api > views.py > ...
109
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.

```
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 ]
```

So, when we run our path for Employees:

127.0.0.1:8000/api/v1/employees/

Django REST framework api_djangoadmin

Api Root / Employee Viewset List

Employee Viewset List

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": 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:

The image shows a web browser window at the top with the URL `127.0.0.1:8000/api/v1/employees/` highlighted in red. Below the browser is a Django REST framework interface for the 'Employee Viewset List'. The interface shows the response for a GET request to `/api/v1/employees/`, which is a JSON array of employee objects. Below the response, there is a form for the 'Media type' (set to `application/json`) and 'Content' (a JSON object for a new employee). To the right is a code editor showing the Python code for the `EmployeeViewSet` class, with the `create` method highlighted in red. The code includes a `lookup_field = 'pk'` and a `create` method that uses `serializer.save()` and returns a `Response` with `status.HTTP_201_CREATED`.

This results to:

The image shows the Django REST framework interface for the 'Employee Viewset List'. The interface shows the response for a POST request to `/api/v1/employees/`, which is a JSON object representing the newly created employee. The response is highlighted in red. Below the response, there is a form for the 'Media type' (set to `application/json`) and 'Content' (an empty text area). A 'POST' button is visible at the bottom right of the form.

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.

The browser window displays the URL `127.0.0.1:8000/api/v1/employees/9/` and the response for the `Employee Viewset Instance`. The response is a JSON object:

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

The code editor shows the `views.py` file with the following code:

```
1 from django.shortcuts import render, get_object_or_404
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
```

The code editor shows the `views.py` file with the following code:

```
169 """
170
171 # uses Viewsets
172 class EmployeeViewSet(viewsets.ViewSet):
173
174     def list(self, request):
175         queryset = Employee.objects.all()
176         serializer = EmployeeSerializer(queryset, many=True)
177         return Response(serializer.data)
178
179     def create(self, request):
180         serializer = EmployeeSerializer(data=request.data)
181         if serializer.is_valid():
182             serializer.save()
183             return Response(serializer.data, status=status.HTTP_201_CREATED)
184         return Response(serializer.errors)
185
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)
190
```

Our URLS.PY is still this:

The screenshot shows the Django REST API Project's `urls.py` file. The code 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 ]
```

Two red boxes highlight the Viewsets configuration: one around lines 5-7 and another around line 19.

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

The screenshot shows the `EmployeeViewSet` class in `views.py`. The code is as follows:

```
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)
190
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)
```

A red box highlights the `delete` method implementation.

7. To update the record,

The image shows a web browser on the left and a code editor on the right. The browser displays the Django REST framework interface for an 'Employee Viewset Instance'. The URL is `127.0.0.1:8000/api/v1/employees/1/`. The response is a JSON object: `{ "id": 1, "emp_id": "EMP001", "emp_name": "Rosilie Lim", "designation": "Software Developer" }`. The code editor shows the `EmployeeViewSet` class with `delete` and `update` methods. The `update` method uses `serializer.is_valid()` to check for errors and returns a `Response` with the updated data or errors.

8. Our updated Employee model shall look like this:

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.

```
File Edit Selection View ... DjangoREST_APIProject  
EXPLORER  
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  
  employees  
    > __pycache__  
models.py employees admin.py students urls.py django_rest_main urls.py  
api > views.py > EmployeeViewSet  
190  
191  
192  
193 # def delete(self, request, pk=None):  
194 #     employee = get_object_or_404(Employee, pk=pk)  
195 #     employee.delete()  
196 #     return Response(status=status.HTTP_204_NO_CONTENT)  
197  
198 # def update(self, request, pk=None):  
199 #     employee = get_object_or_404(Employee, pk=pk)  
200 #     serializer = EmployeeSerializer(employee, data=request.data)  
201 #     if serializer.is_valid():  
202 #         serializer.save()  
203 #         return Response(serializer.data, status=status.HTTP_200_OK)  
204 #     else:  
205 #         return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)  
206  
207 # uses Viewsets the easier way  
208 class EmployeeViewSet(viewsets.ModelViewSet):  
209     queryset = Employee.objects.all()  
210     serializer_class = EmployeeSerializer
```

To add a new record:

127.0.0.1:8000/api/v1/employees/

Django REST framework api_django admin

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	<input type="text" value="EMP003"/>
Emp name	<input type="text" value="Russell"/>
Designation	<input type="text" value="Security"/>

To update a record:

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

Emp name

Designation

PUT

To delete a record:

Django REST framework api_djangoadmin

Api Root / Employee Viewset Instance

Employee Viewset Instance

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	<input type="text" value="EMP003"/>
Emp name	<input type="text" value="Russell"/>
Designation	<input type="text" value="Security Officer"/>

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

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:

127.0.0.1:8000/api/v1/employees/50/

Django REST framework api_djangoadmin

Api Root / Employee Viewset List / Employee Viewset Instance

Employee Viewset Instance

DELETE OPTIONS GET

GET /api/v1/employees/50/

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

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

Raw data HTML form

Emp id

Emp name

Designation

PUT

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