

Topic: 7. Class-Based Views Basics & Retrieving All Records

Speaker: Personal | Notebook: API Development using Django Framework



Class-based views follow the principles of object-oriented programming. These views are used for reusability and code efficiency.

For this example, we need to create a new app called EMPLOYEES.

1. In the terminal, create a new app:

```
$ python manage.py startapp employees
```

2. Register this new app in the SETTINGS.PY

```
File Edit Selection View ... DjangoREST_APIProj
EXPLORER
  DJANGOREST_APIPROJECT
    > api
    > django_rest_main
      > __pycache__
      > __init__.py
      > asgi.py
      > settings.py
      > urls.py
      > wsgi.py
    > employees
      > migrations
      > __init__.py
      > admin.py
      > apps.py
  settings.py admin.py urls.py django_rest_

django_rest_main > settings.py > ...
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'rest_framework',
41     'students',
42     'api',
43     'employees',
44 ]
45
```

3. Create the new model in MODELS.PY

The screenshot shows the Visual Studio Code interface for a Django REST API project. The Explorer sidebar on the left shows the project structure: DJANGOREST_APIPROJECT, api, django_rest_main, and employees. The employees folder is expanded, showing migrations, __init__.py, admin.py, and apps.py. The models.py file is selected and highlighted with a red box. The code in models.py is as follows:

```
employees > models.py > ...
1  from django.db import models
2
3  class Employee(models.Model):
4      emp_id = models.CharField(max_length=20)
5      emp_name=models.CharField(max_length=50)
6      designation=models.CharField(max_length=50)
7
8      def __str__(self):
9          return self.emp_name
10
```

4. Update the ADMIN.PY

The screenshot shows the Visual Studio Code interface for the same Django REST API project. The Explorer sidebar on the left shows the project structure, with the employees folder expanded and admin.py selected. The code in admin.py is as follows:

```
employees > admin.py
1  from django.contrib import admin
2  from .models import Employee
3
4  admin.site.register(Employee)
```

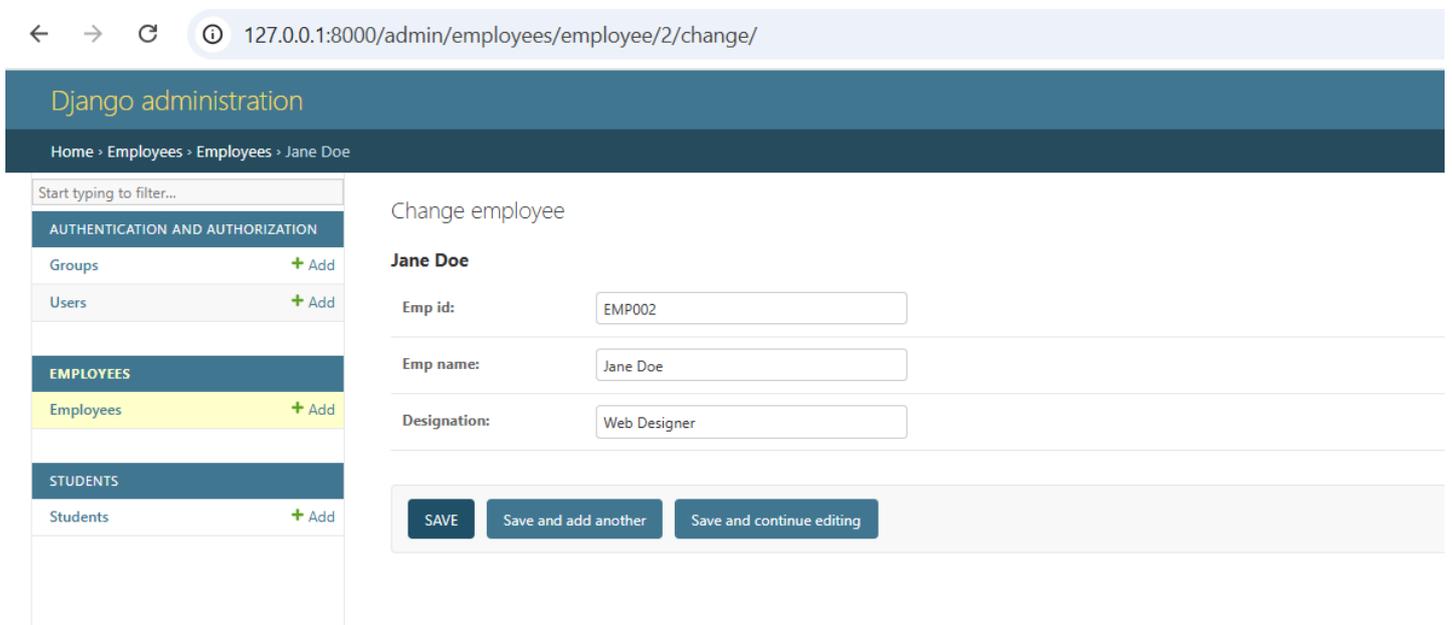
5. Make the migrations. Be sure you are in the correct folder to see the migrations happen.

```

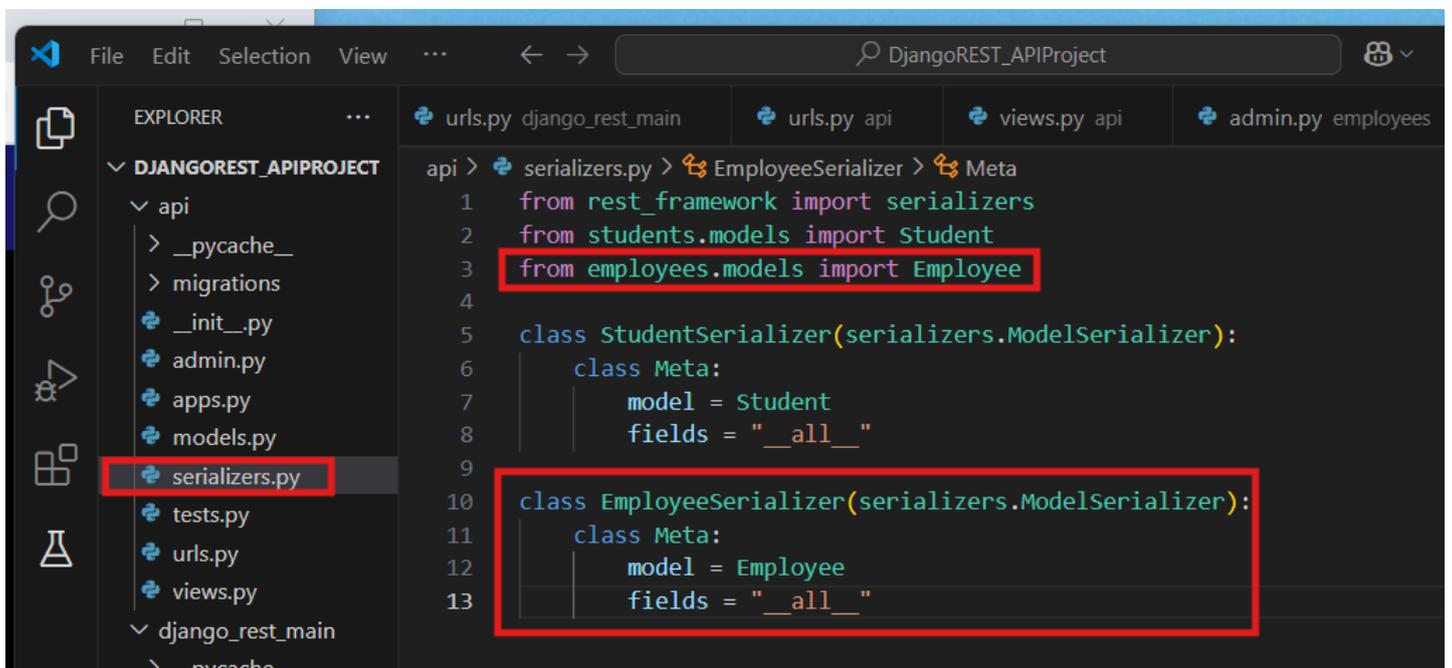
rosil@LearnCodeRepeat MINGW64 /c/Users/rosil/OneDrive/Documents/MyCodingCareer/Django Projects/API
Dev/Resources/DjangoREST_APIProject
• $ python manage.py makemigrations
Migrations for 'employees':
  employees\migrations\0001_initial.py
    + Create model Employee
(env)
rosil@LearnCodeRepeat MINGW64 /c/Users/rosil/OneDrive/Documents/MyCodingCareer/Django Projects/API
Dev/Resources/DjangoREST_APIProject
• $ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, employees, sessions, students
Running migrations:
  Applying employees.0001_initial... OK
(env)
rosil@LearnCodeRepeat MINGW64 /c/Users/rosil/OneDrive/Documents/MyCodingCareer/Django Projects/API
Dev/Resources/DjangoREST_APIProject
$

```

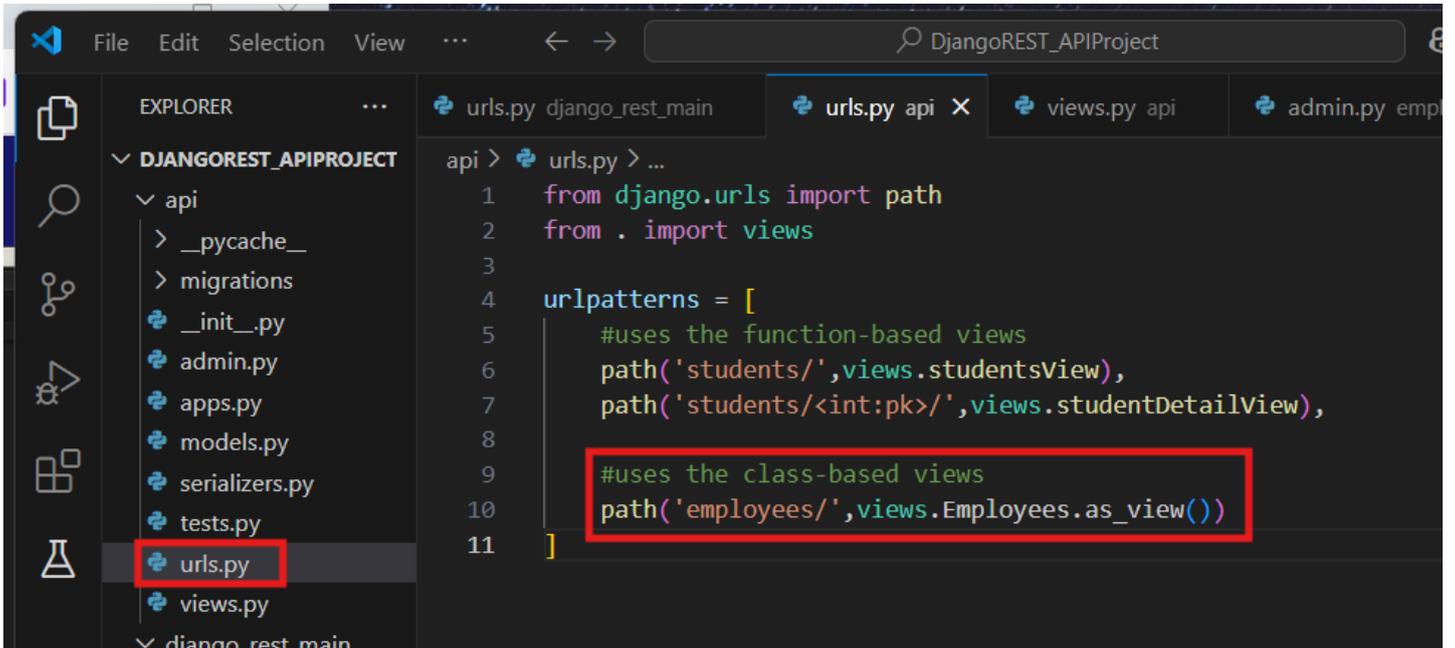
6. Check the admin panel and create new records for the Employees model:



7. Create a serializer for the Employees model. Go to API\SERIALIZER.PY



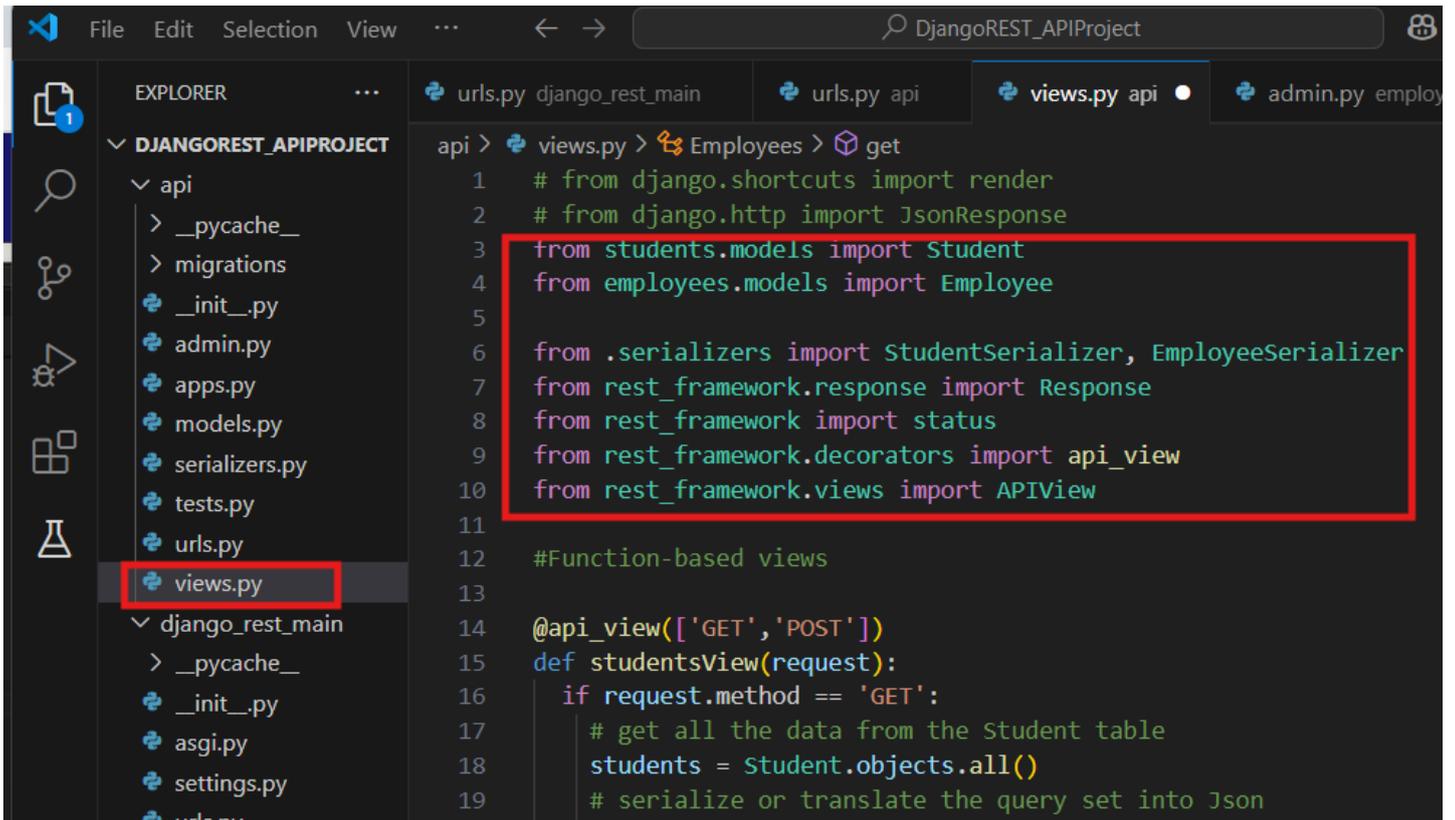
8. To retrieve all the records, go to the APIURLS.PY and update:



```
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 ]
```

9. Create the class-based views. Go to the APIVIEWS.PY:

Add the necessary libraries:



```
api > views.py > Employees > get
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
12 #Function-based views
13
14 @api_view(['GET', 'POST'])
15 def studentsView(request):
16     if request.method == 'GET':
17         # get all the data from the Student table
18         students = Student.objects.all()
19         # serialize or translate the query set into Json
```

Create the class Employee 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
```

10. To test, use the URL <http://127.0.0.1:8000/api/v1/employees/>

Django REST framework api_djangoadmin

Employees

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"
  }
]
```

11.

