

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

Speaker: Personal / Notebook: API Development using Django Framework



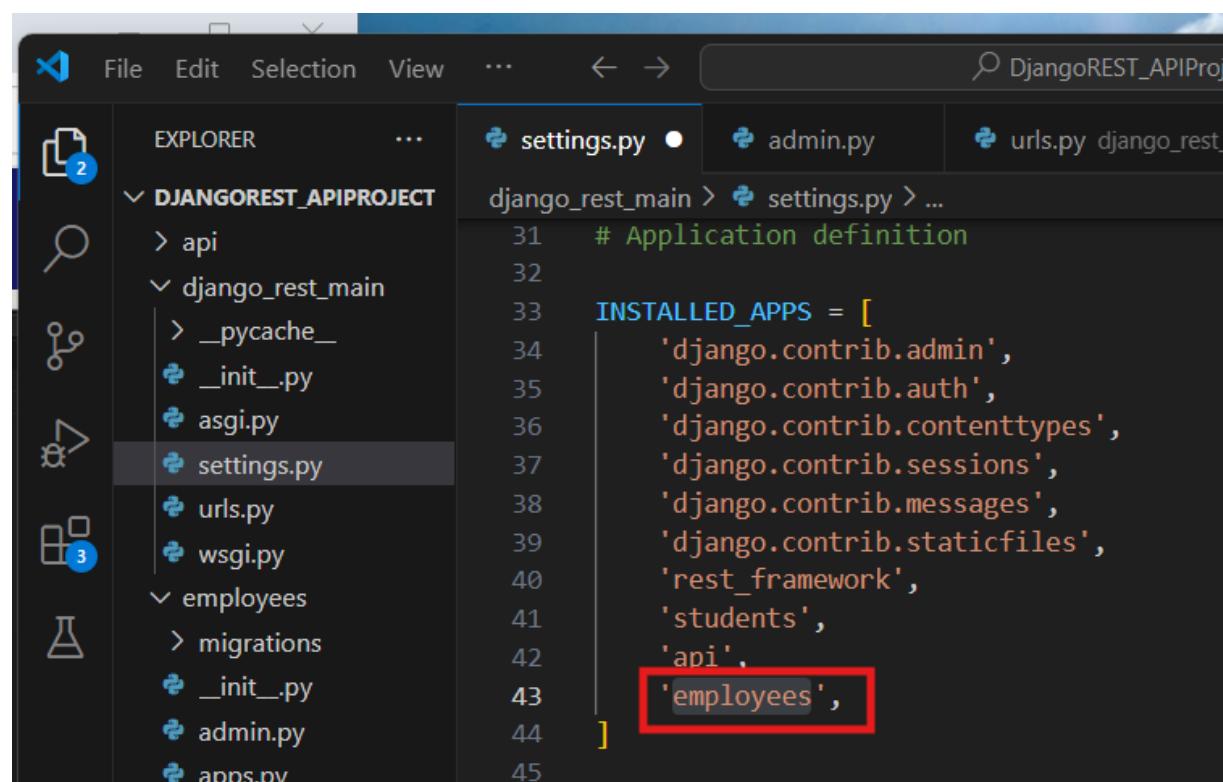
Class-based views follow the principles of object-oriented programming. These views are used for reusability and code efficiency.

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
```

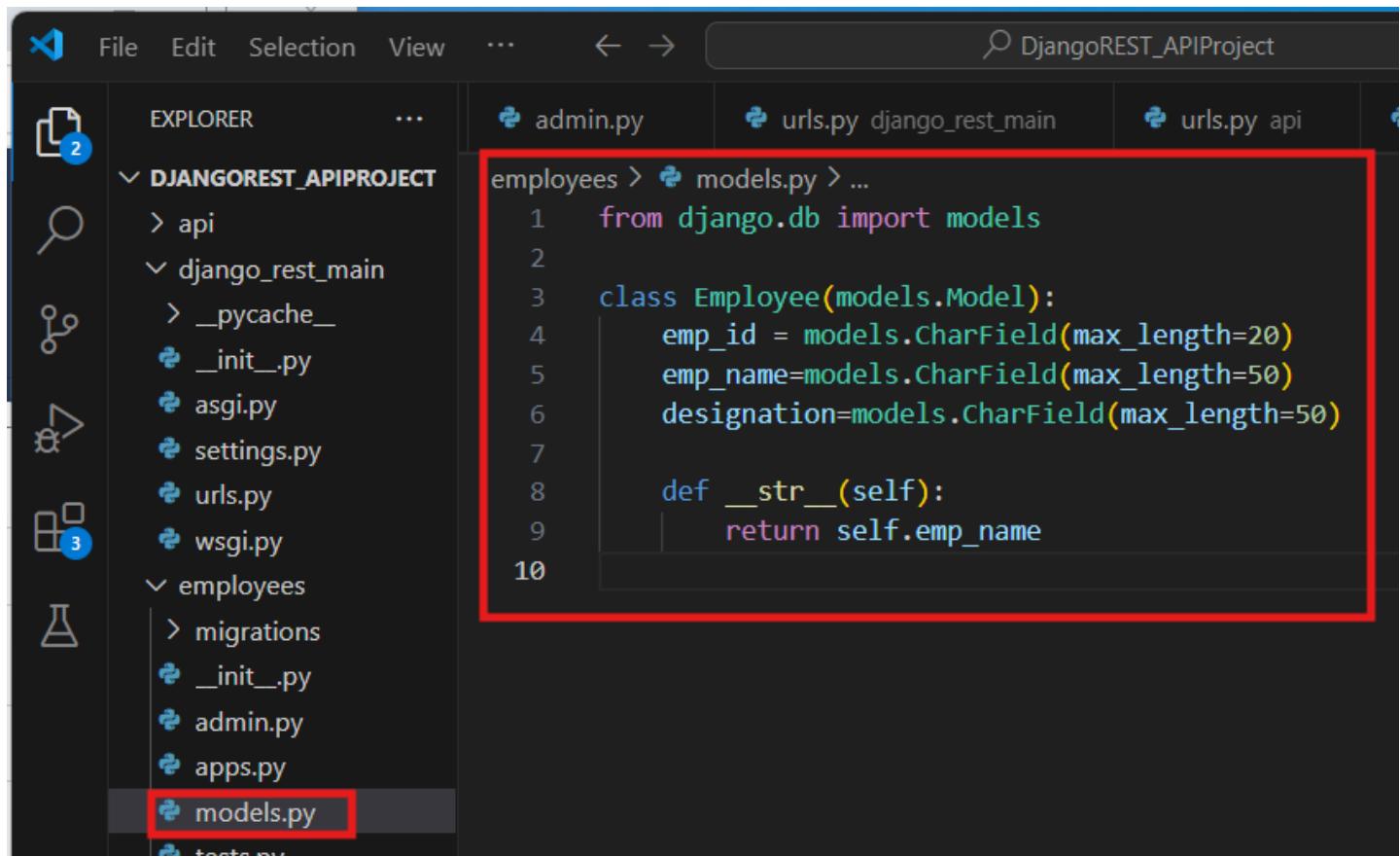
2. Register this new app in the SETTINGS.PY

A screenshot of the Visual Studio Code (VS Code) interface. The left sidebar shows the project structure under 'EXPLORER' with a tree view of files and folders. The 'DJANGOREST_APIPROJECT' folder contains 'api', 'django_rest_main', 'employees', and 'models.py'. The 'django_rest_main' folder contains 'migrations', 'settings.py', 'urls.py', and 'wsgi.py'. The 'employees' folder contains 'migrations', 'settings.py', 'admin.py', and 'apps.py'. The 'models.py' file is currently selected and shown in the main editor area. The code in 'models.py' is as follows:

```
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 ]
```

The line 'employees', is highlighted with a red box.

3. Create the new model in MODELS.PY



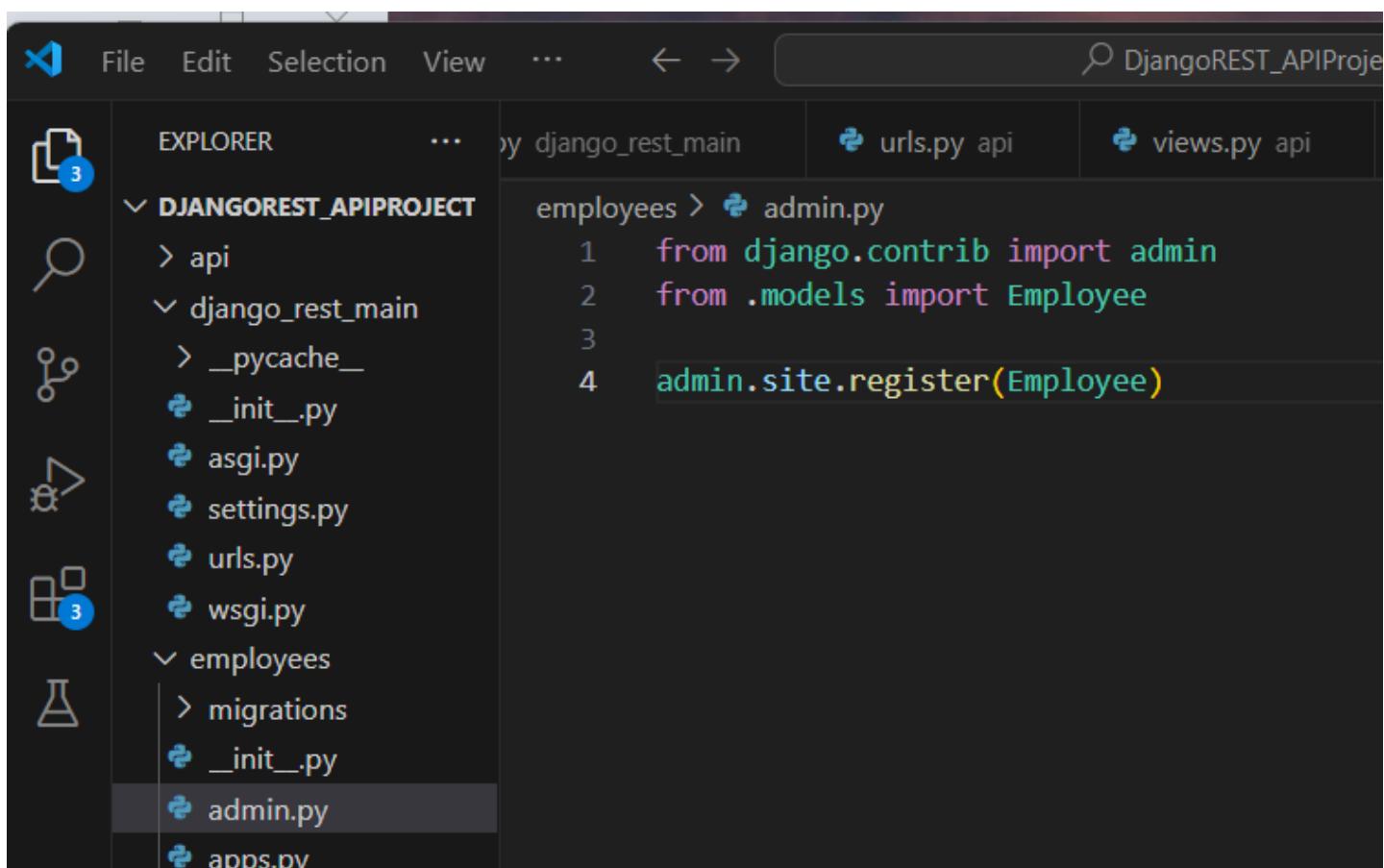
File Edit Selection View ... ← → 🔍 DjangoREST_APIProject

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
 - models.py
- tests.py

4. Update the ADMIN.PY



File Edit Selection View ... ← → 🔍 DjangoREST_APIProject

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
- tests.py

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
Devt/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
Devt/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
Devt/Resources/DjangoREST_APIProject
○ $ []

```

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

127.0.0.1:8000/admin/employees/employee/2/change/

Django administration

Home > Employees > Employees > Jane Doe

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

- Groups
- Users

EMPLOYEES

- Employees

STUDENTS

- Students

Change employee

Jane Doe

Emp id: EMP002

Emp name: Jane Doe

Designation: Web Designer

SAVE **Save and add another** **Save and continue editing**

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

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

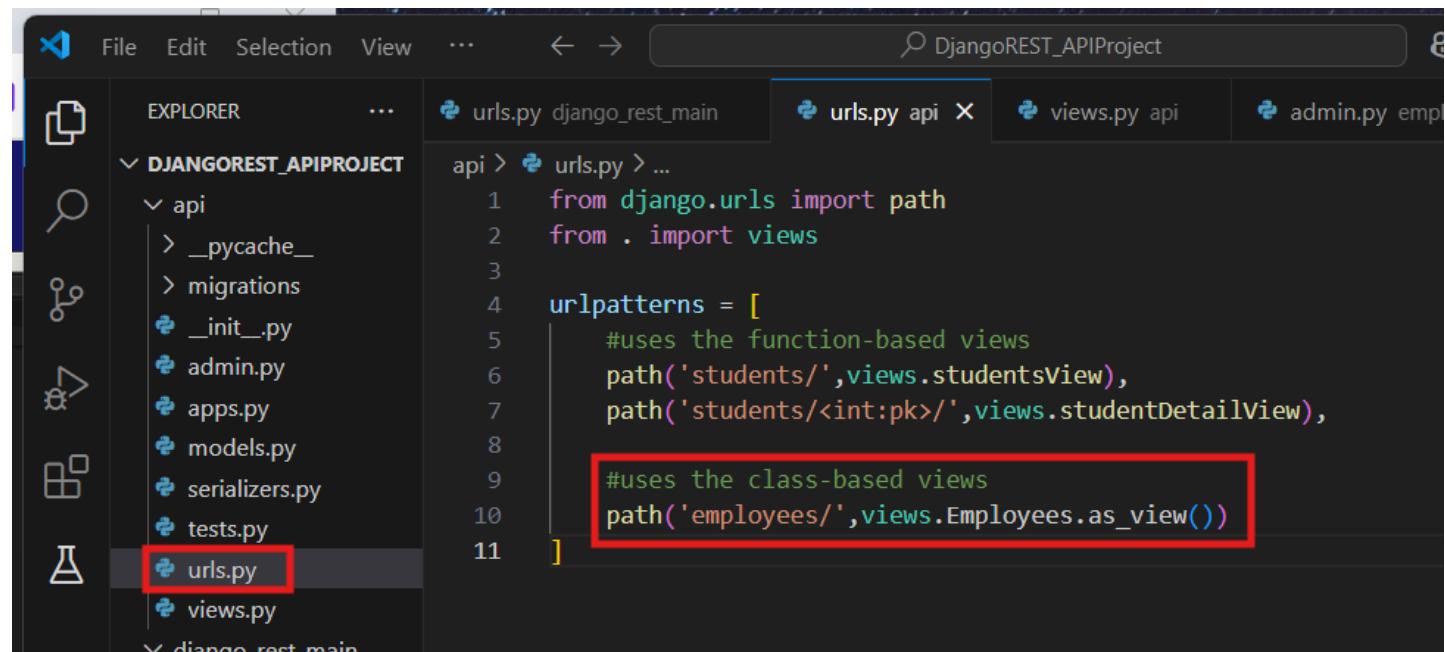
urls.py django_rest_main urls.py api views.py api admin.py employees

```

api > serializers.py > EmployeeSerializer > Meta
1   from rest_framework import serializers
2   from students.models import Student
3   from employees.models import Employee
4
5   class StudentSerializer(serializers.ModelSerializer):
6       class Meta:
7           model = Student
8           fields = "__all__"
9
10  class EmployeeSerializer(serializers.ModelSerializer):
11      class Meta:
12          model = Employee
13          fields = "__all__"

```

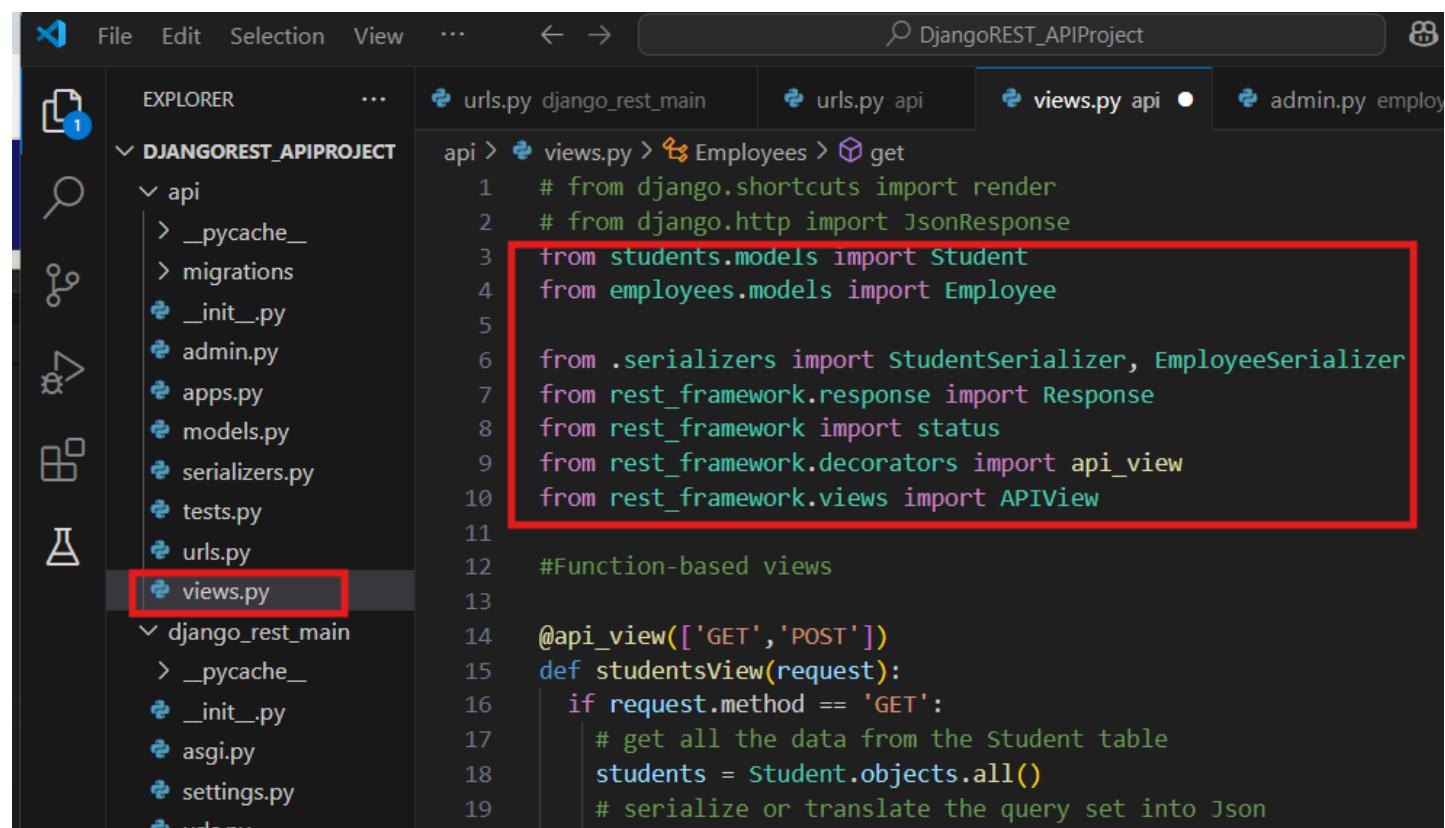
8. To retrieve all the records, go to the API\URLS.PY and update:



```
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 ]
```

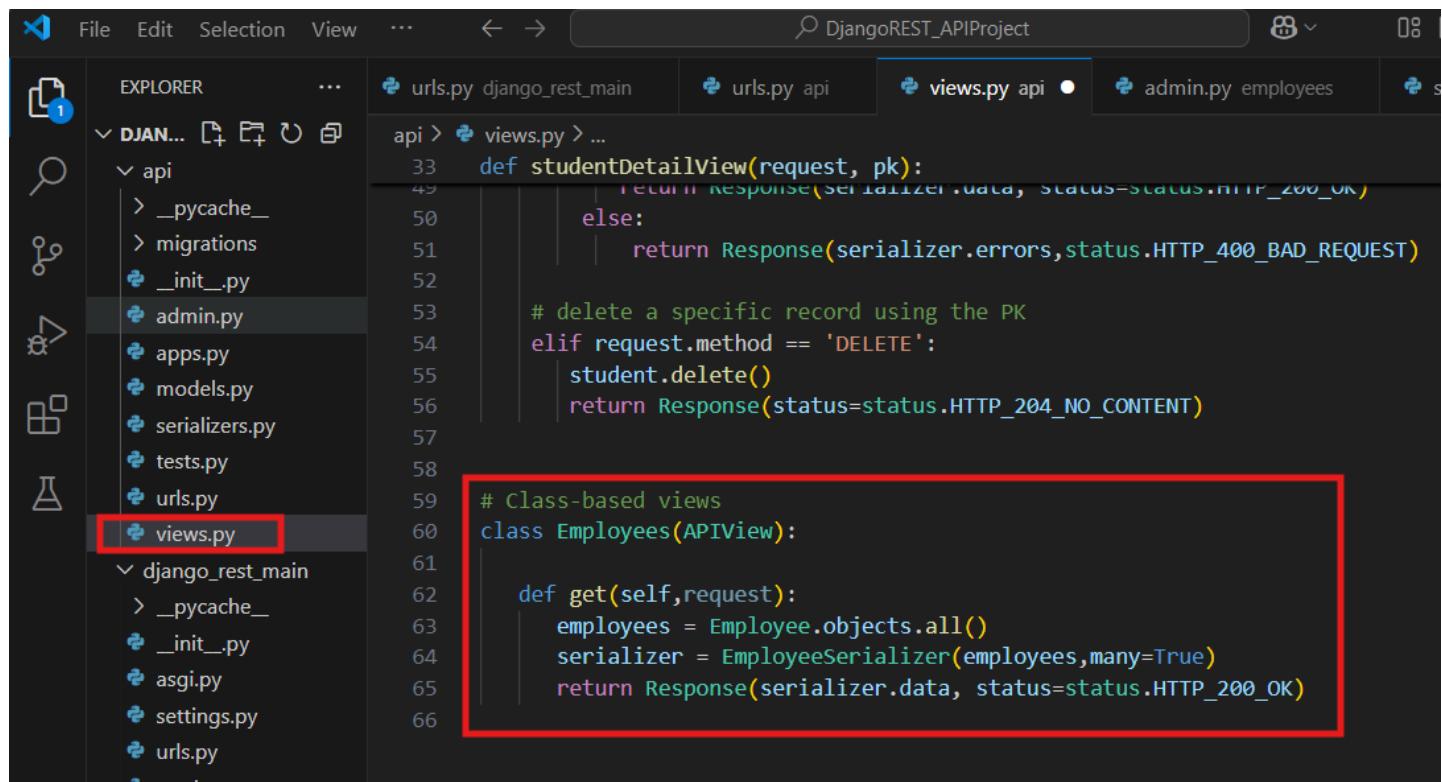
9. Create the class-based views. Go to the API\VIEWS.PY:

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
```

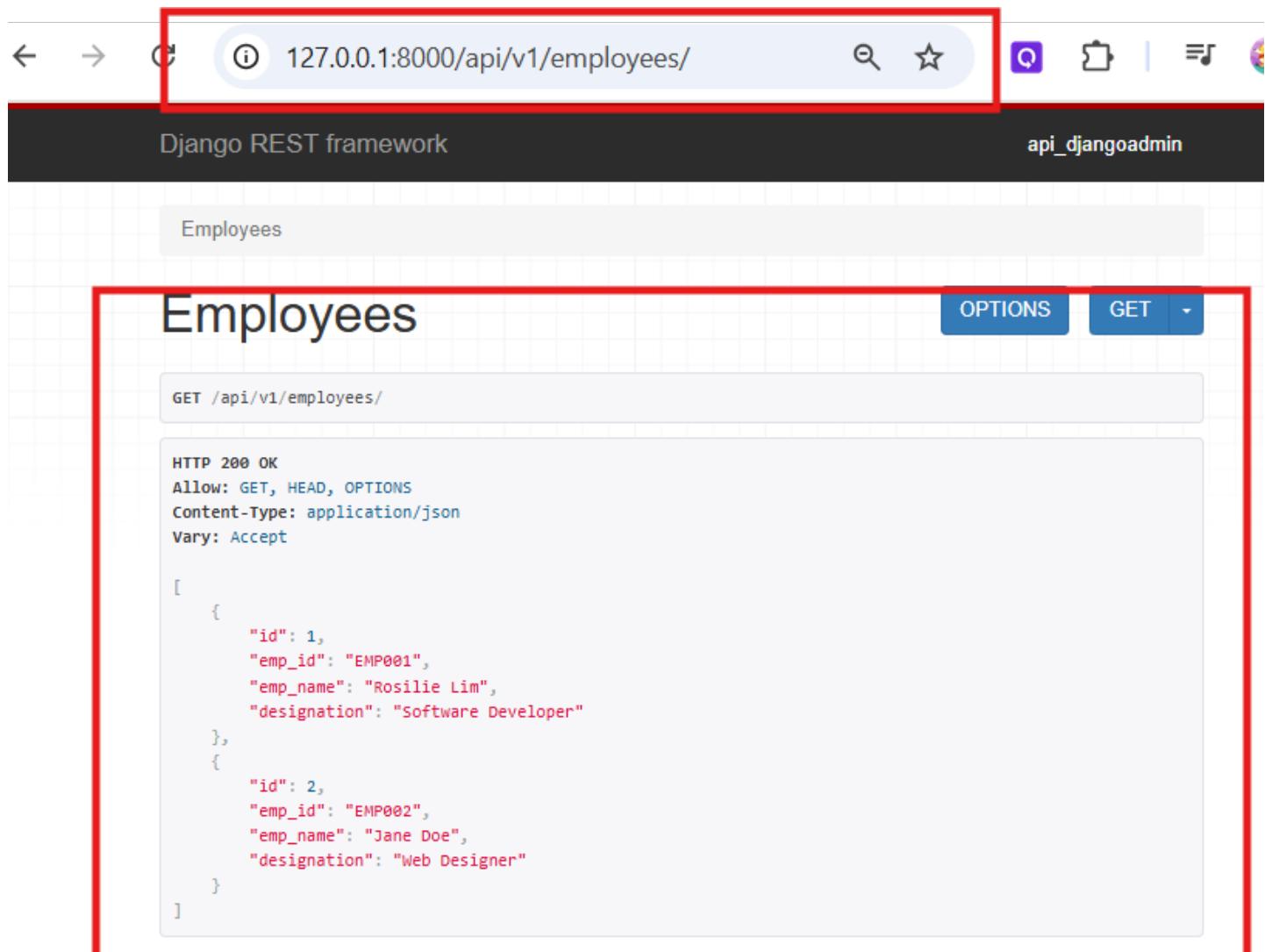
Create the class Employee and its methods:



The screenshot shows a code editor with the project structure on the left and the `views.py` file content on the right. The `views.py` file contains a class-based view `Employees` that returns a list of employees. A red box highlights the code for the `Employees` class and its `get` method.

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

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



The screenshot shows a browser window displaying the Django REST framework API endpoint for employees. The URL in the address bar is `127.0.0.1:8000/api/v1/employees/`. The page title is "Django REST framework" and the sub-page title is "Employees". The main content shows the API response for a GET request, which includes the `OPTIONS` and `GET` buttons. A red box highlights the JSON response data, which shows two employees with their details.

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.

