

Topic: 4. Setting API EndPoints

Speaker: Personal / Notebook: API Development using Django Framework

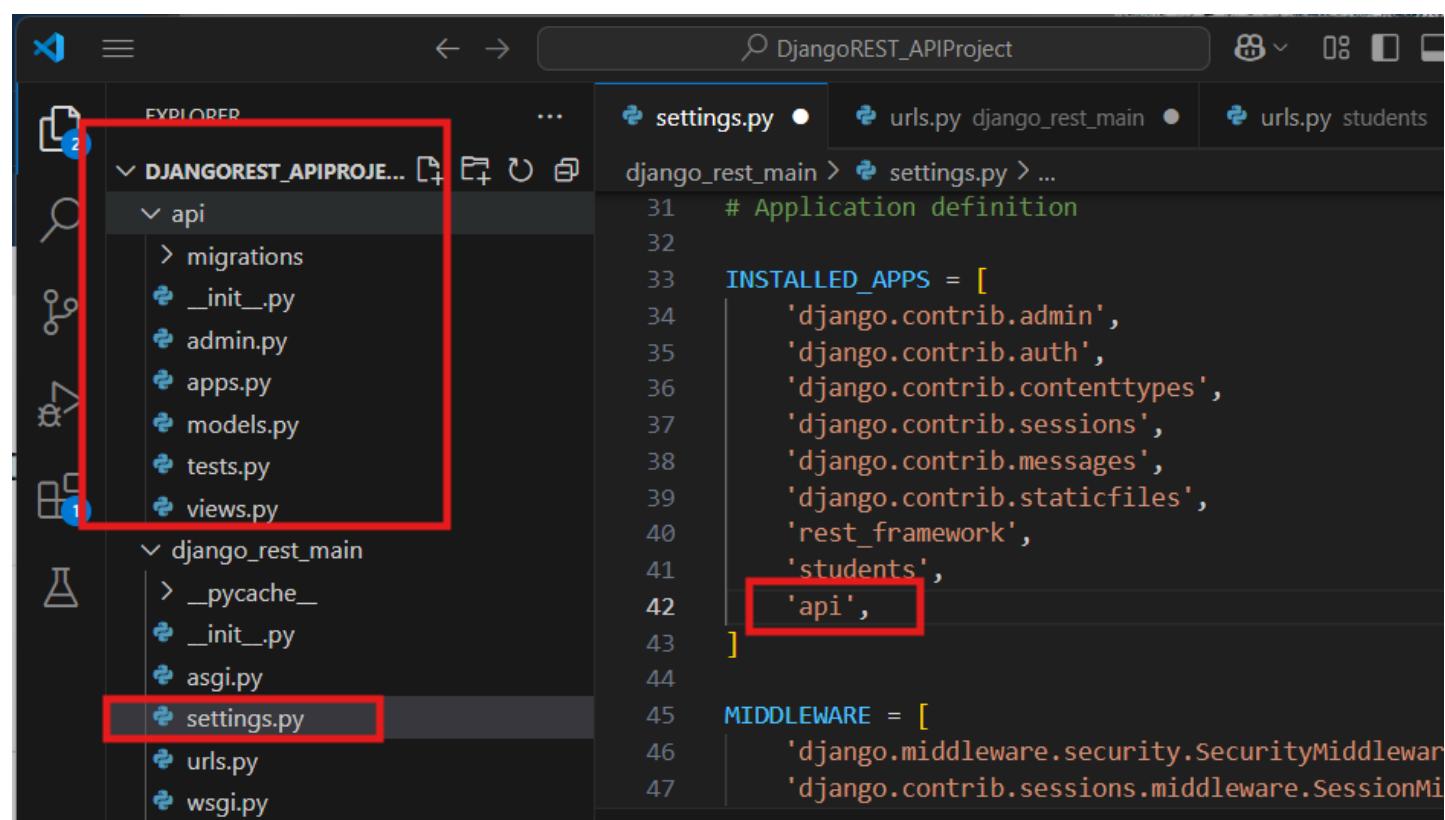


For other resources on how to create simple API endpoints using Django, use this [reference in Medium](#).

1. Create an API app in your Django project:

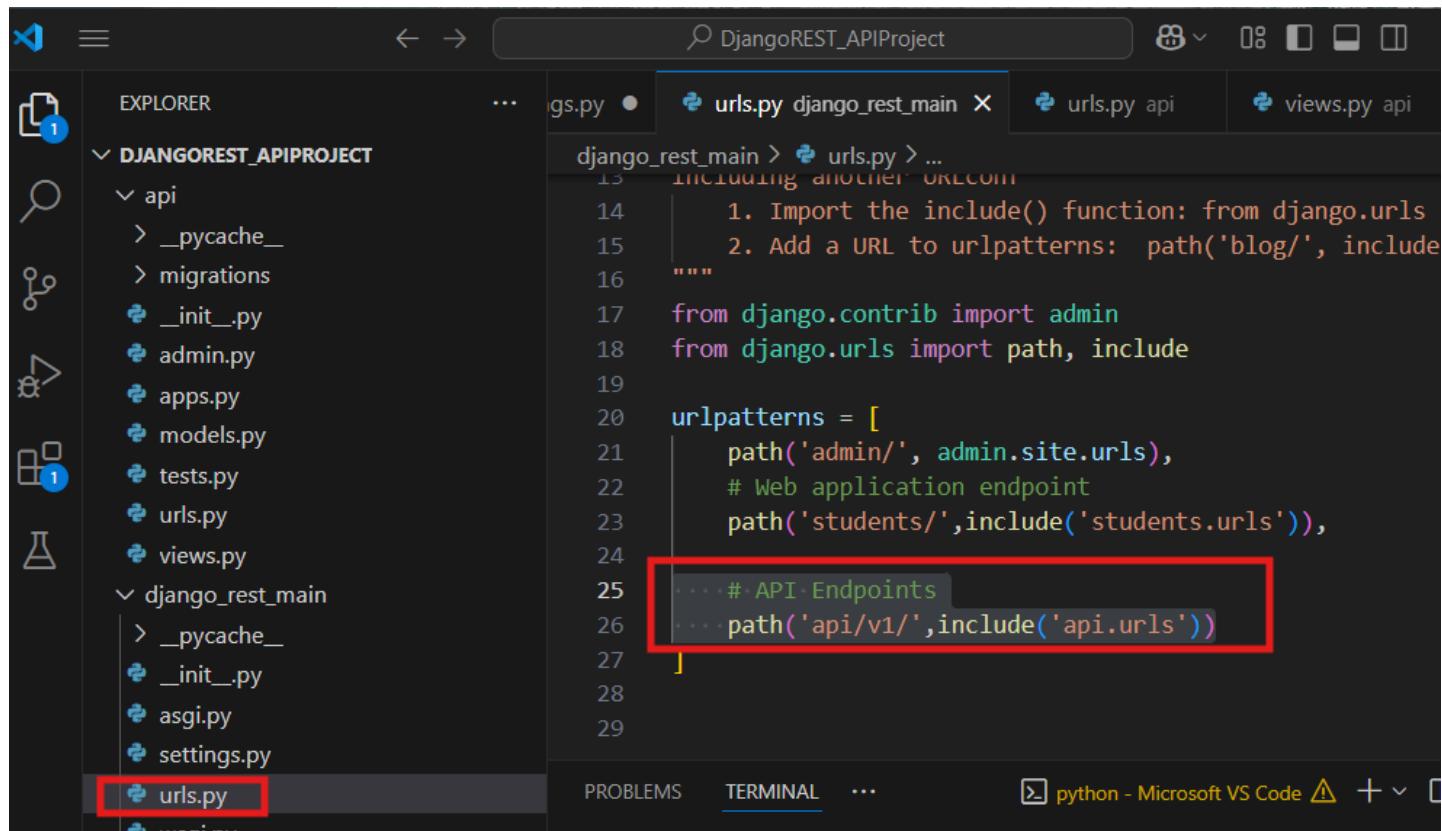
```
$ python manage.py startapp api
```

2. Register this new app in your SETTINGS.PY

A screenshot of the Visual Studio Code (VS Code) interface. The title bar says 'DjangoREST_APIProject'. The left sidebar shows the 'EXPLORER' view with a tree structure of the project. A red box highlights the 'api' application folder, which contains 'migrations', '__init__.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', and 'views.py'. Below it is the 'django_rest_main' folder with files like '__pycache__', '__init__.py', 'asgi.py', 'settings.py' (which is also highlighted with a red box), 'urls.py', and 'wsgi.py'. The main editor area shows the 'settings.py' file for the project. A red box highlights the line of code where the 'api' application is registered in the 'INSTALLED_APPS' list. The code 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 ]
44
45 MIDDLEWARE = [
46     'django.middleware.security.SecurityMiddleware',
47     'django.contrib.sessions.middleware.SessionMi
```

3. Now, update the main project's URLs.PY to include the URLs.PY of the newly created app.



EXPLORER

DJANGOREST_APIPROJECT

- api
- migrations
- __init__.py
- admin.py
- apps.py
- models.py
- tests.py
- urls.py
- views.py

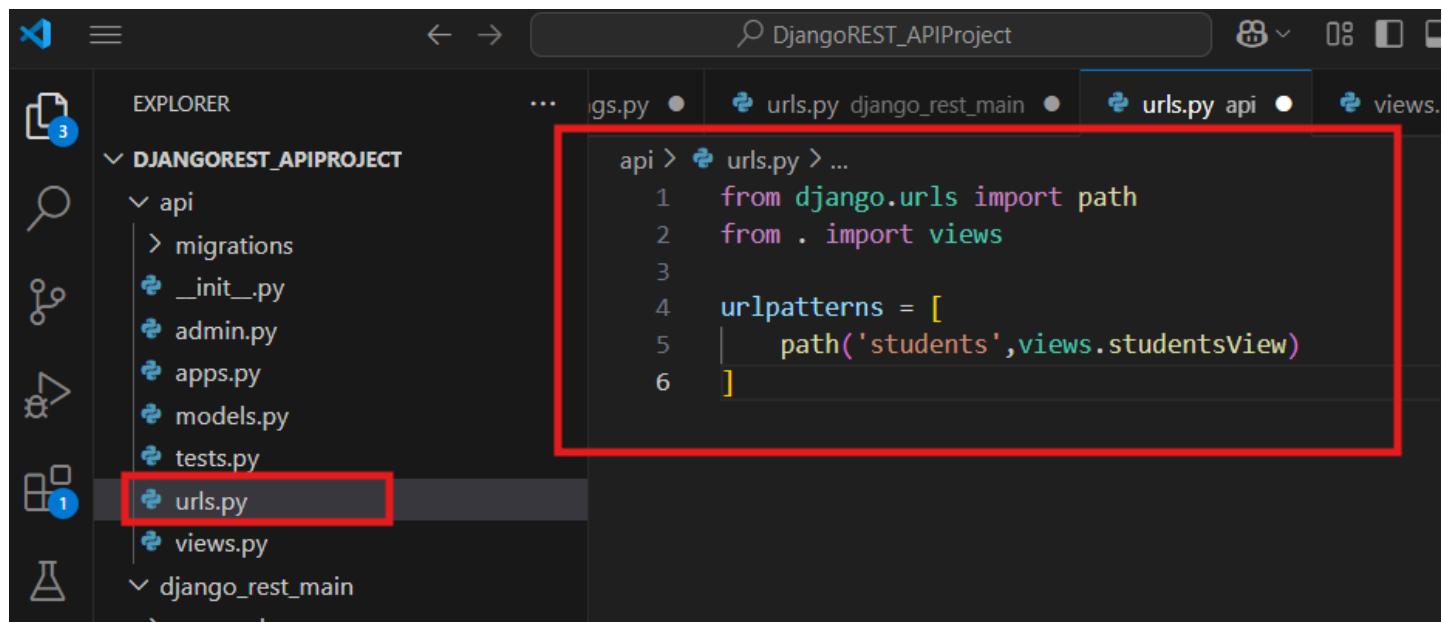
urls.py

```
14     1. Import the include() function: from django.urls
15     2. Add a URL to urlpatterns: path('blog/', include
16     """
17     from django.contrib import admin
18     from django.urls import path, include
19
20     urlpatterns = [
21         path('admin/', admin.site.urls),
22         # Web application endpoint
23         path('students/', include('students.urls')),
24
25         # API Endpoints
26         path('api/v1/', include('api.urls'))
27
28
29
```

PROBLEMS TERMINAL ...

python - Microsoft VS Code

4. We then create a URLs.py in our API app and create the path.



EXPLORER

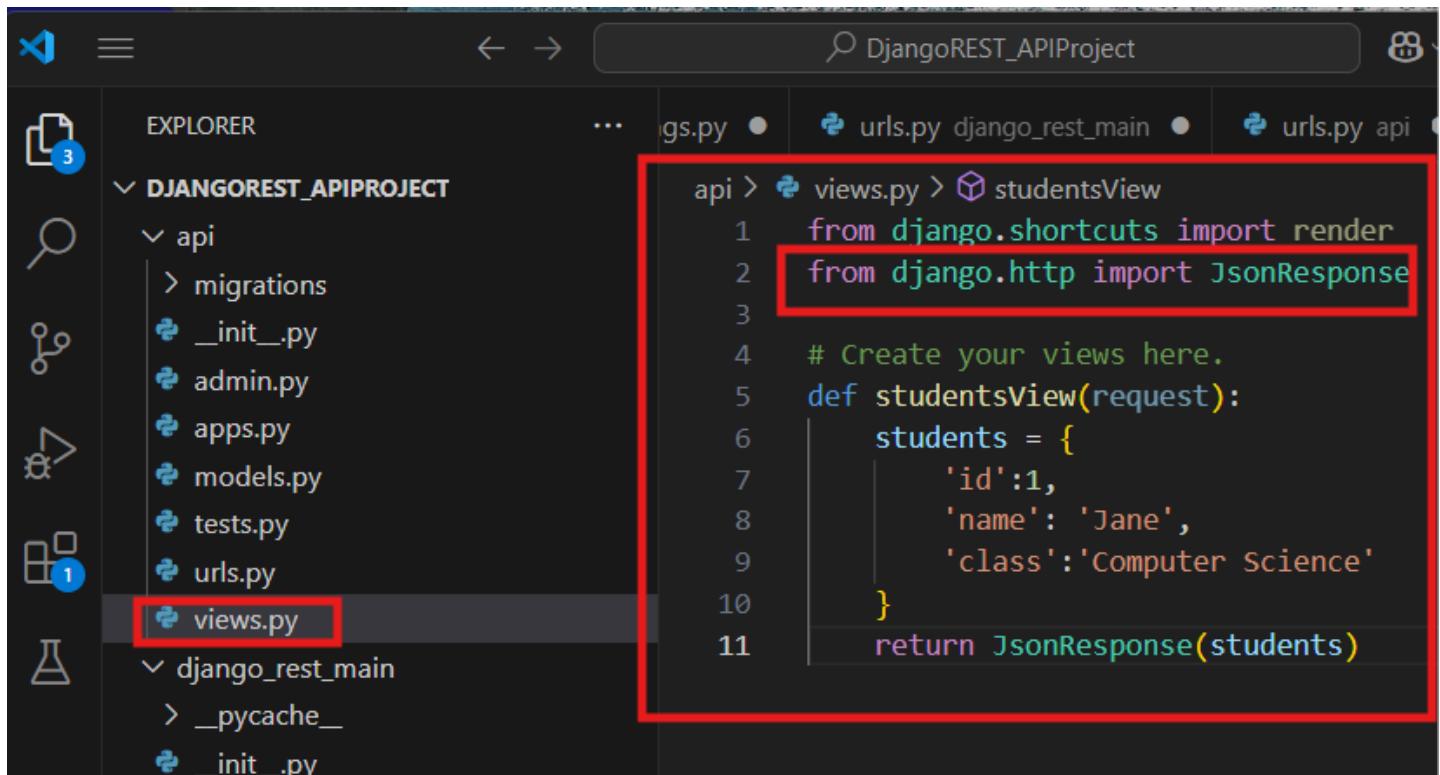
DJANGOREST_APIPROJECT

- api
- migrations
- __init__.py
- admin.py
- apps.py
- models.py
- tests.py
- urls.py
- views.py

urls.py

```
1 from django.urls import path
2 from . import views
3
4 urlpatterns = [
5     path('students',views.studentsView)
6 ]
```

5. Update the API APP'S VIEWS.PY . Unlike regular views of web-based endpoints, APIs return JSON data.



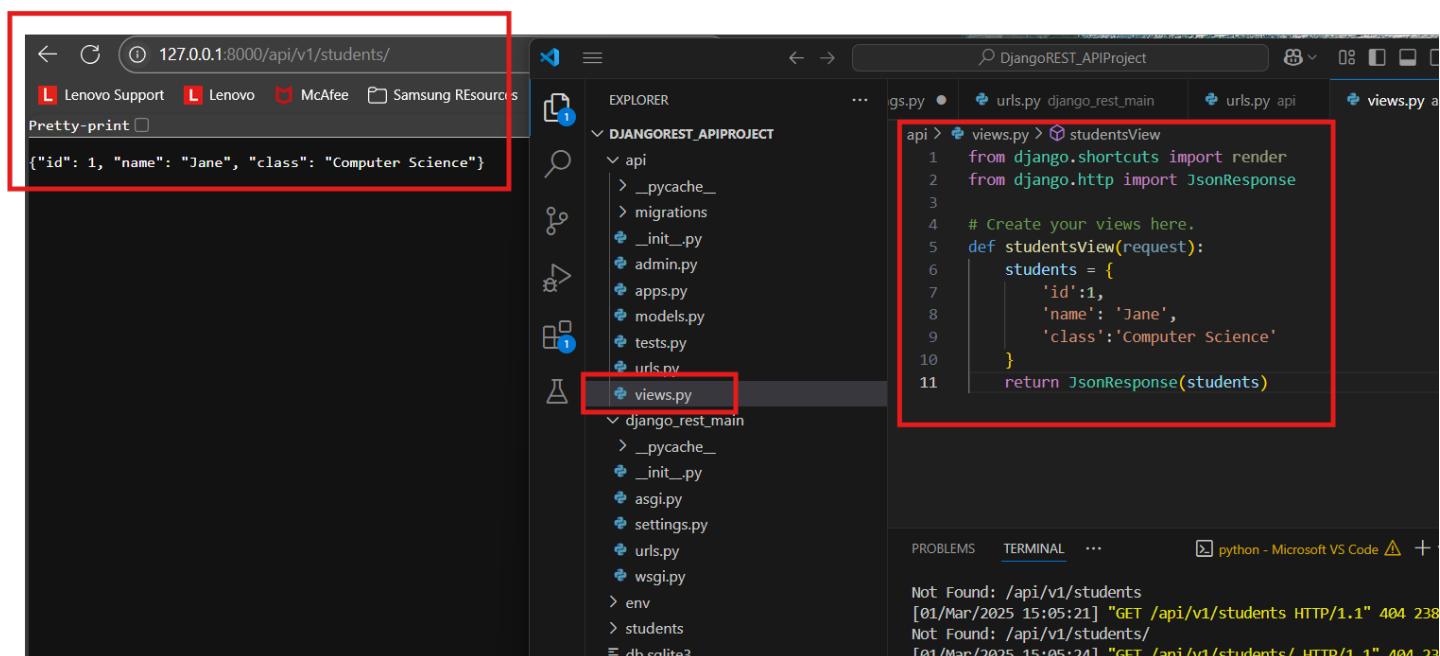
The screenshot shows the Microsoft VS Code interface with the project 'DjangoREST_APIProject' open. The Explorer sidebar on the left shows the project structure, including the 'api' directory with 'migrations', '_init_.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'views.py' file is selected and highlighted with a red box. The code editor on the right contains the following Python code:

```
api > views.py > studentsView
1  from django.shortcuts import render
2  from django.http import JsonResponse
3
4  # Create your views here.
5  def studentsView(request):
6      students = {
7          'id':1,
8          'name': 'Jane',
9          'class':'Computer Science'
10     }
11     return JsonResponse(students)
```

6. Run the server and add the API endpoint:

```
$ python manage.py runserver
```

In your browser: 127.0.0.1:8000/api/v1/students/



The screenshot shows the Microsoft VS Code interface with the project 'DjangoREST_APIProject' open. The Explorer sidebar on the left shows the project structure, including the 'api' directory with 'migrations', '_init_.py', 'admin.py', 'apps.py', 'models.py', 'tests.py', 'urls.py', and 'views.py'. The 'views.py' file is selected and highlighted with a red box. The code editor on the right contains the same Python code as before. To the left, a browser window is open at the URL '127.0.0.1:8000/api/v1/students/'. The browser's status bar shows 'Pretty-print' and the response content: {"id": 1, "name": "Jane", "class": "Computer Science"}. The browser's address bar shows the full URL.

7. Run the migrations command and create superuser.

```
$ python manage.py migrate
```

This will create `user.auths` default table.

The screenshot shows the VS Code interface with the following details:

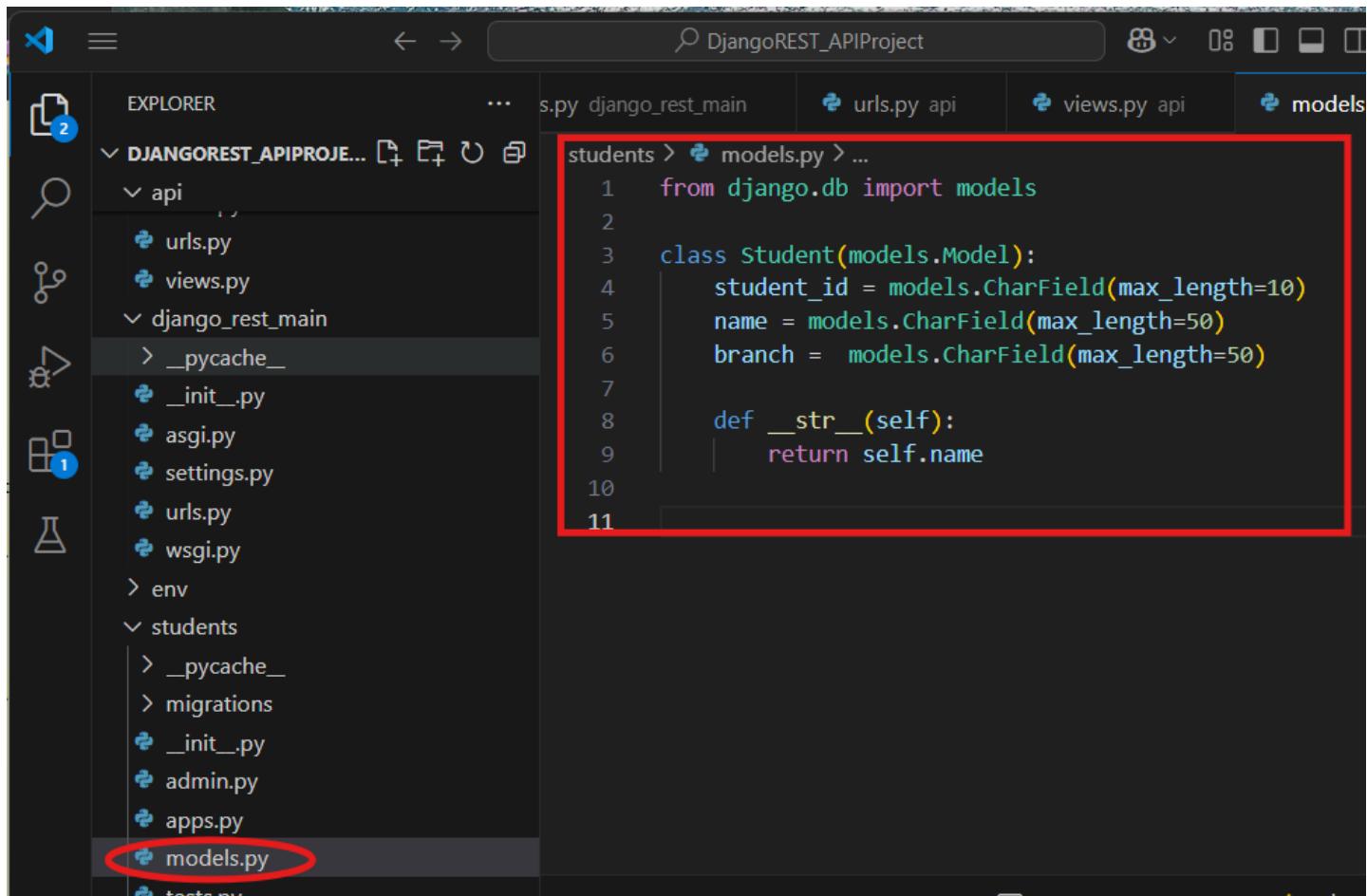
- Explorer View:** Shows the project structure for "DJANGOREST_APIPROJECT". The "api" directory is expanded, showing files like `__init__.py`, `admin.py`, `apps.py`, `models.py`, `tests.py`, `urls.py`, and `views.py`. The `views.py` file is currently selected.
- Code Editor:** Displays the content of `views.py` under the `api` directory. The code defines a `studentsView` function that returns a JSON response with a single student record.
- Terminal View:** Shows the output of running the `migrate` command. The terminal window title is "DjangoREST_APIProject". The output shows the following migrations being applied:

```
rosil@LearnCodeRepeat MINGW64 C:/Users/rosil/AppData/Local/Programs/Microsoft VS Code
● $ python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying auth.0012_alter_user_first_name_max_length... OK
  Applying sessions.0001_initial... OK
(env)
rosil@LearnCodeRepeat MINGW64 C:/Users/rosil/AppData/Local/Programs/Microsoft VS Code
○ $
```

8. Access the admin panel by running the server. Access it using:

```
http://127.0.0.1:8000/admin/
```

9. Create a student model in the STUDENTS app to create a new table.



```
1  from django.db import models
2
3  class Student(models.Model):
4      student_id = models.CharField(max_length=10)
5      name = models.CharField(max_length=50)
6      branch = models.CharField(max_length=50)
7
8      def __str__(self):
9          return self.name
10
11
```

10. Run the migrations.

```
$ python manage.py makemigrations
$ python manage.py migrate
```

```
● $ python manage.py makemigrations
  Migrations for 'students':
    students\migrations\0001_initial.py
      + Create model Student
  (env)
```

```
(env)
rosil@LearnCodeRepeat MINGW64 C:/Users/rosil/AppData/Local/Programs/Mi
Code
$ python manage.py migrate
  Operations to perform:
    Apply all migrations: admin, auth, contenttypes, sessions, students
  Running migrations:
    Applying students.0001_initial... OK
  (env)
```

11. To make this table appear on your admin dashboard, register this in the ADMIN.PY of STUDENTS app.

Django administration

WELCOME, API_DJANGOADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups + Add Change

Users + Add Change

STUDENTS

Students + Add Change

Recent actions

My actions

None available

EXPLORER

DJANGOREST_APIPROJECT

- api
- urls.py
- views.py
- django_rest_main
- __pycache__
- __init__.py
- asgi.py
- settings.py
- urls.py
- wsgi.py
- env
- students
- __pycache__
- migrations
- __init__.py
- admin.py
- apps.py

admin.py

```
students > admin.py
1 from django.contrib import admin
2 from .models import Student
3
4 # Register your models here.
5 admin.site.register(Student)
6
```

PROBLEMS TERMINAL ...

C:\Users\rrosil\OneDrive\Documents\MyCodingCourses\ DjangoREST_APITProject\students\admin.py

12. Create a sample record in the table.

127.0.0.1:8000/admin/students/student/1/change/

Django administration

WELCOME, API_DJANGOADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Students > Students > Student object (1)

Start typing to filter...

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

STUDENTS

Students + Add

Change student

Student object (1)

Student id: S001

Name: Rosilie

Branch: CS

SAVE Save and add another Save and continue editing Delete

13. To display the model student records using the admin dashboard:

127.0.0.1:8000/admin/students/student/

Django administration

WELCOME, API_DJANGOADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Students > Students

Select student to change

ADD STUDENT

Action: ----- Go 0 of 3 selected

STUDENT ID	NAME	BRANCH
S003	Xeria	Hotel Mgt
S002	Yuri	Engineering
S001	Rosilie	Computer Science

3 students

EXPLORER

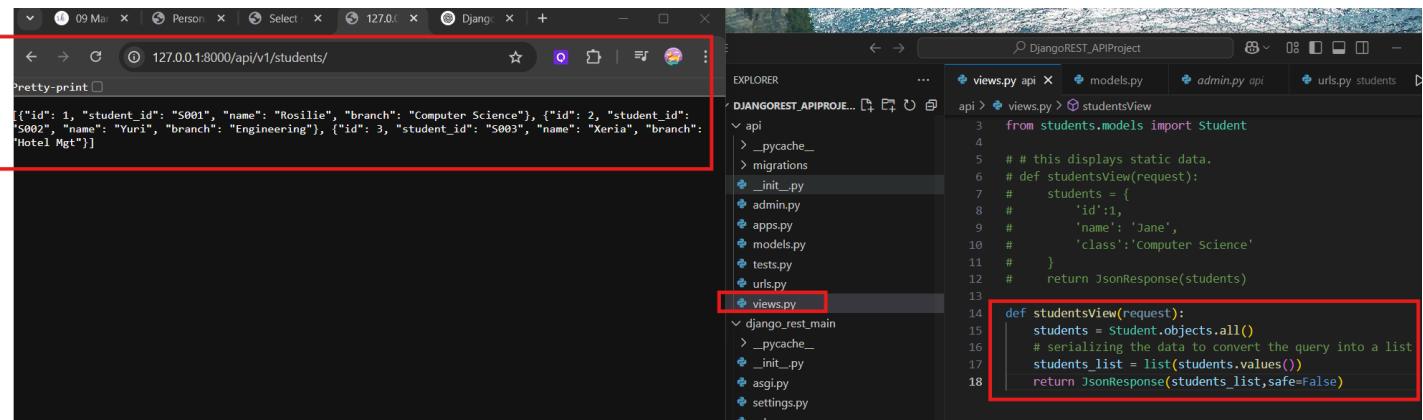
DJANGOREST_APIPROJECT

- api
- urls.py
- views.py
- django_rest_main
- __pycache__
- __init__.py
- asgi.py
- settings.py
- urls.py
- wsgi.py
- env
- students
- __pycache__
- migrations
- __init__.py
- admin.py
- apps.py

admin.py

```
students > admin.py > ...
1 from django.contrib import admin
2 from .models import Student
3
4 class StudentAdmin(admin.ModelAdmin):
5     # list the records under these field names
6     list_display = ["student_id", "name", "branch"]
7
8 admin.site.register(Student, StudentAdmin)
9
```

14. To display the dynamic data from the database. We need to **SERIALIZE** to convert the returned query set into a list.



The screenshot shows a browser window on the left displaying the URL `127.0.0.1:8000/api/v1/students/`. The response is a JSON list of student records:

```
[{"id": 1, "student_id": "S001", "name": "Rosilie", "branch": "Computer Science"}, {"id": 2, "student_id": "S002", "name": "Yuri", "branch": "Engineering"}, {"id": 3, "student_id": "S003", "name": "Xeria", "branch": "Hotel Mgt"}]
```

The browser window has a red box around the JSON response. To the right is a code editor in a Django REST API project. The `views.py` file is open, showing the `studentsView` function:

```
from students.models import Student
# this displays static data.
def studentsView(request):
    students = [
        {'id': 1, 'name': 'Jane', 'class': 'Computer Science'}
    ]
    return JsonResponse(students)

def studentsView(request):
    students = Student.objects.all()
    # serializing the data to convert the query into a list
    students_list = list(students.values())
    return JsonResponse(students_list, safe=False)
```

The code editor has a red box around the line `students_list = list(students.values())`.

15. **SERIALIZERS** are like **TRANSLATORS** that convert certain data i.e **QUERYSET** from your database into other types of data like **JSON** data that can be used on **HTML**. While **DESERIALIZERS** will reverse the translation i.e from **JSON** file into **Query set** (database records).