

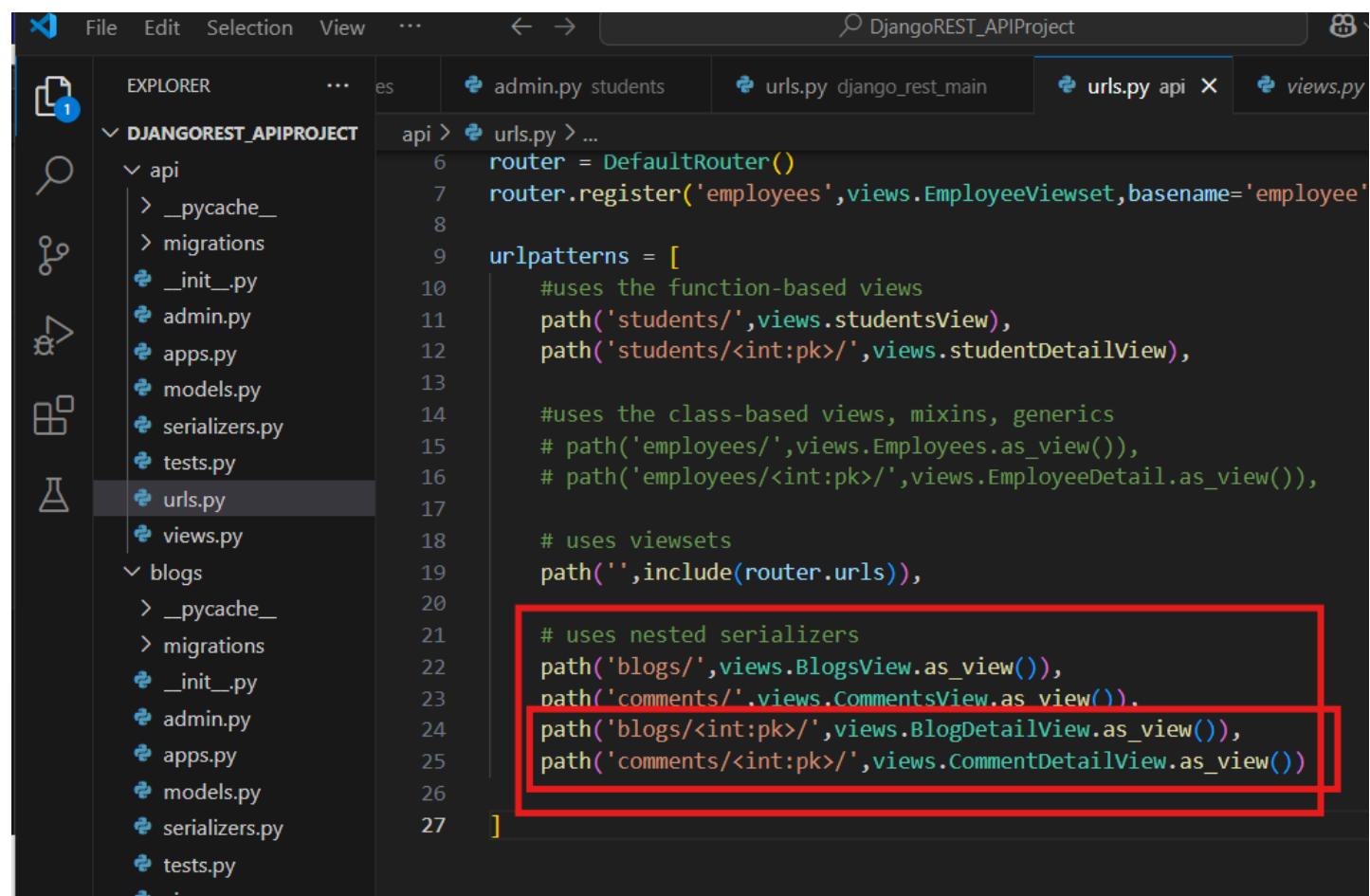
Topic: 13. Nested Serializers for Related Models Part 2

Speaker: Personal / Notebook: API Development using Django Framework



In this post, we use the primary key to allow other CRUD operations.

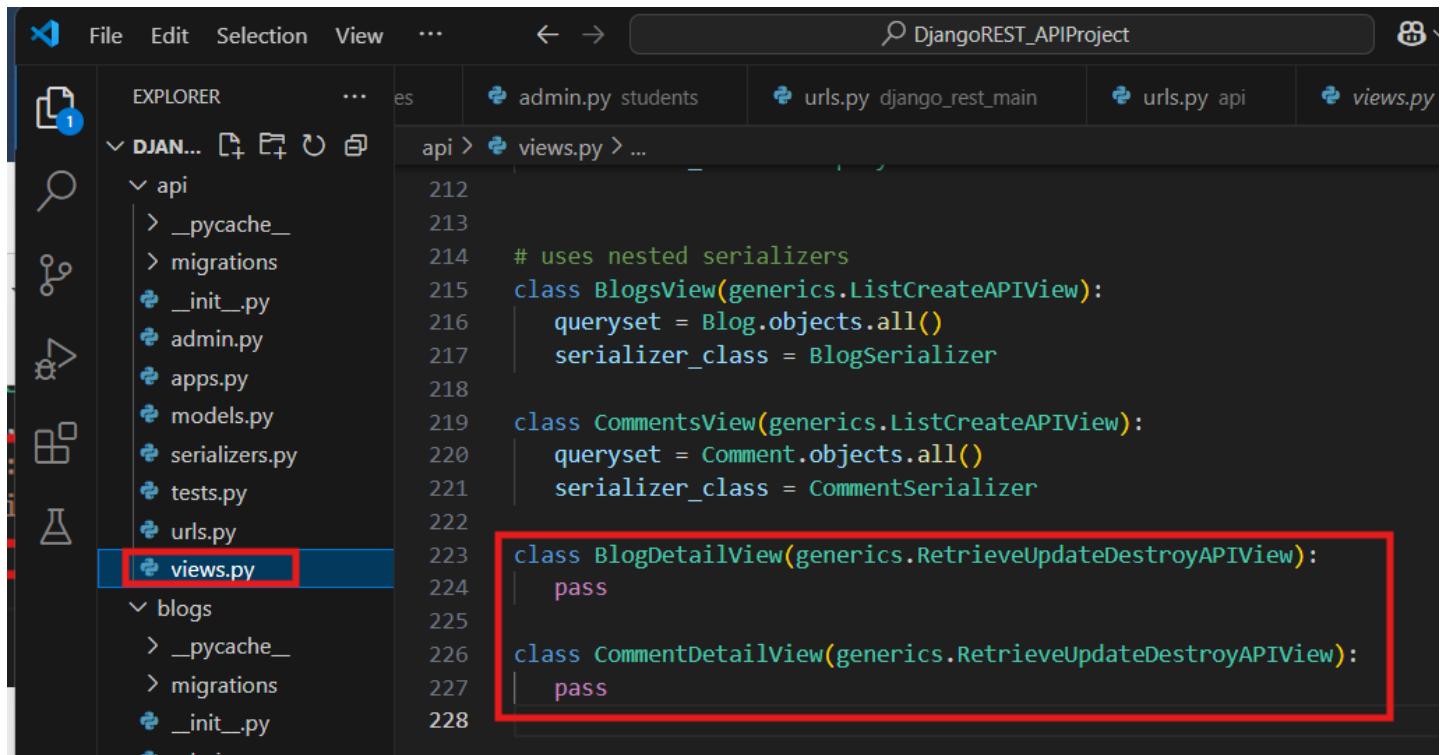
1. So we update the API\URLS.PY to include the PK-based operations.



```
File Edit Selection View ... ← → DjangoREST_APIProject
EXPLORER ... admin.py students urls.py django_rest_main urls.py api X views.py
DJANGOREST_APIPROJECT
  api
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    serializers.py
    tests.py
    urls.py
    views.py
  blogs
    __pycache__
    migrations
    __init__.py
    admin.py
    apps.py
    models.py
    serializers.py
    tests.py
    views.py

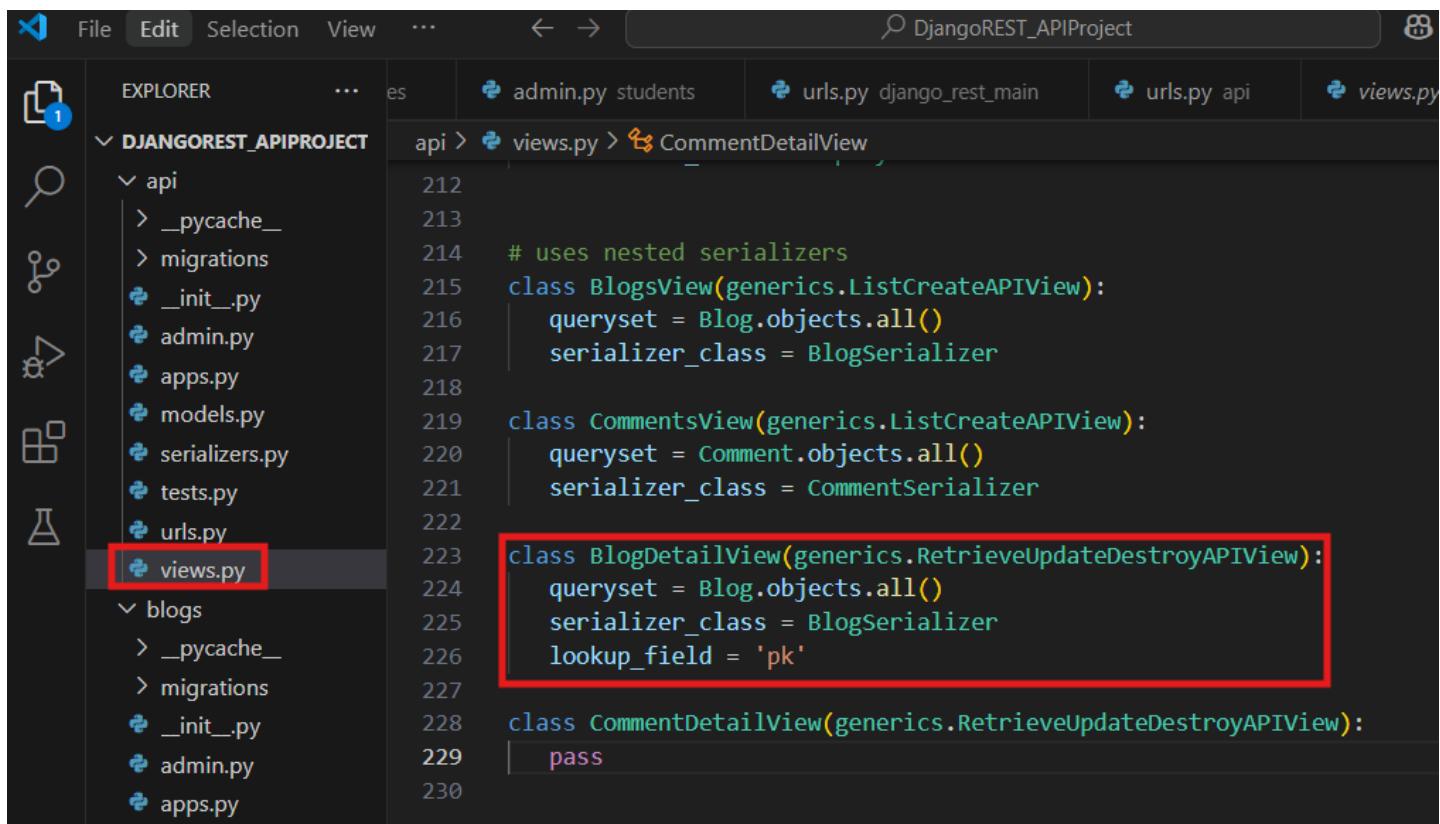
api > urls.py > ...
6   router = DefaultRouter()
7   router.register('employees',views.EmployeeViewSet, basename='employee')
8
9   urlpatterns = [
10      #uses the function-based views
11      path('students/',views.studentsView),
12      path('students/<int:pk>',views.studentDetailView),
13
14      #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      # uses nested serializers
22      path('blogs/',views.BlogsView.as_view()),
23      path('comments/.views.CommentsView.as_view()'),
24      path('blogs/<int:pk>',views.BlogDetailView.as_view()),
25      path('comments/<int:pk>',views.CommentDetailView.as_view())
26
27 ]
```

2. Update the API\VIEWS.PY



```
File Edit Selection View ... ← → ⚙ DjangoREST_APIProject
EXPLORER ... es admin.py students urls.py django_rest_main urls.py api views.py
api > views.py > ...
212
213
214 # uses nested serializers
215 class BlogsView(generics.ListCreateAPIView):
216     queryset = Blog.objects.all()
217     serializer_class = BlogSerializer
218
219 class CommentsView(generics.ListCreateAPIView):
220     queryset = Comment.objects.all()
221     serializer_class = CommentSerializer
222
223 class BlogDetailView(generics.RetrieveUpdateDestroyAPIView):
224     pass
225
226 class CommentDetailView(generics.RetrieveUpdateDestroyAPIView):
227     pass
228
```

Then update as:



```
File Edit Selection View ... ← → ⚙ DjangoREST_APIProject
EXPLORER ... es admin.py students urls.py django_rest_main urls.py api views.py
api > views.py > CommentDetailView
212
213
214 # uses nested serializers
215 class BlogsView(generics.ListCreateAPIView):
216     queryset = Blog.objects.all()
217     serializer_class = BlogSerializer
218
219 class CommentsView(generics.ListCreateAPIView):
220     queryset = Comment.objects.all()
221     serializer_class = CommentSerializer
222
223 class BlogDetailView(generics.RetrieveUpdateDestroyAPIView):
224     queryset = Blog.objects.all()
225     serializer_class = BlogSerializer
226     lookup_field = 'pk'
227
228 class CommentDetailView(generics.RetrieveUpdateDestroyAPIView):
229     pass
230
```

3. Now view the specific path of blog post # 1:

The screenshot shows a browser window with a red border around the main content area. The address bar contains the URL `127.0.0.1:8000/api/v1/blogs/1/`. The page title is `Django REST Framework` and the sub-page title is `api_djangoadmin`. The main content is titled `Blog Detail` and includes buttons for `DELETE`, `OPTIONS`, and `GET`. A red box highlights the `GET` button and the entire content area below it.

`GET /api/v1/blogs/1/`

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

```
{  
    "id": 1,  
    "comments": [  
        {  
            "id": 1,  
            "comment": "this is comment #1",  
            "blog": 1  
        },  
        {  
            "id": 2,  
            "comment": "this is comment #2",  
            "blog": 1  
        },  
        {  
            "id": 3,  
            "comment": "this is comment #3",  
            "blog": 1  
        }  
}
```

Raw data HTML form

Blog title: this is a blog 1

Blog body: this is a blog body 1

PUT

Now, you can update and delete this record.

→ C 127.0.0.1:8000/api/v1/blogs/1/      

Django REST framework api_djangoadmin

Api Root / Blogs / Blog Detail

Blog Detail

[DELETE /api/v1/blogs/1/](#) [OPTIONS](#) [GET](#) ▾

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

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

Raw data [HTML form](#)

Blog title

Blog body

[PUT](#)

You can also add a new post here:

Api Root / Blogs

Blogs

OPTIONS

GET

GET /api/v1/blogs/

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

[
    {
        "id": 2,
        "comments": [
            {
                "id": 4,
                "comment": "this is comment #1",
                "blog": 2
            },
            {
                "id": 5,
                "comment": "this is comment #2",
                "blog": 2
            }
        ],
        "blog_title": "this is a blog 2",
        "blog_body": "this is a blog body 2"
    }
]
```

Raw data

HTML form

Blog title Blog body 

POST

Then you can reload your page:

127.0.0.1:8000/api/v1/blogs/

Django REST framework

api_djangoadmin

Api Root / Blogs

Blogs

OPTIONS GET

GET /api/v1/blogs/

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
[{"id": 2, "comments": [{"id": 4, "comment": "this is comment #1", "blog": 2}, {"id": 5, "comment": "this is comment #2", "blog": 2}], "blog_title": "this is a blog 2", "blog_body": "this is a blog body 2"}, {"id": 3, "comments": [], "blog_title": "this is a blog 3", "blog_body": "This is a blog description"}]
```

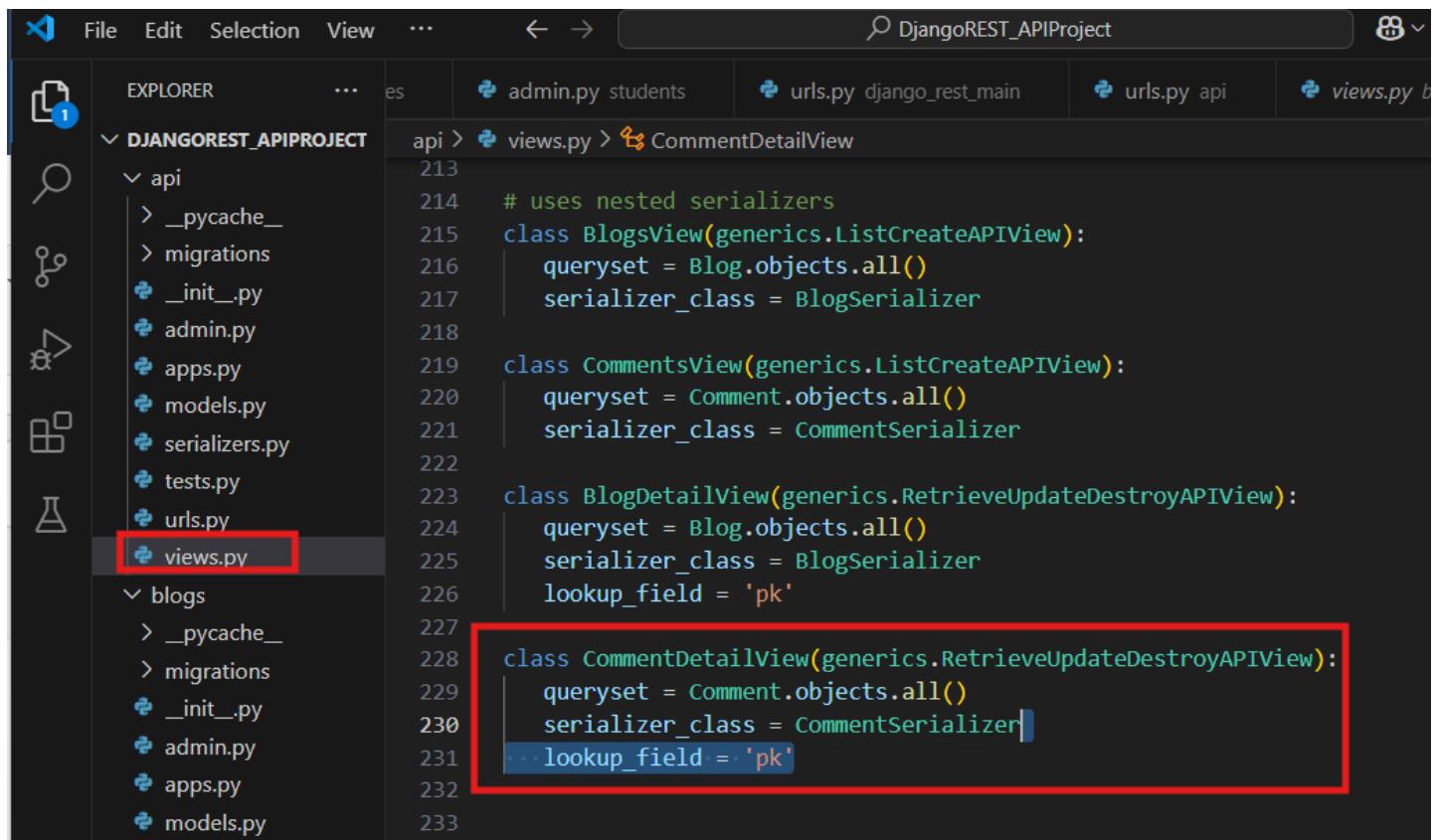
Raw data HTML form

Blog title

Blog body

POST

4. To allow CRUD operations on `Comment` model, then we update the `class CommentDetailView`:



The screenshot shows a code editor interface with the following details:

- File Bar:** File Edit Selection View ...
- Search Bar:** DjangoREST_APIProject
- Explorer:** Shows the project structure under DJANGOREST_APIPROJECT, including the api directory and its subfiles: __pycache__, migrations, __init__.py, admin.py, apps.py, models.py, serializers.py, tests.py, urls.py, and views.py. The views.py file is selected and highlighted with a red box.
- Code Editor:** Displays the code for views.py. The code defines several generic API views:
 - BlogsView (ListCreateAPIView): queryset = Blog.objects.all(), serializer_class = BlogSerializer
 - CommentsView (ListCreateAPIView): queryset = Comment.objects.all(), serializer_class = CommentSerializer
 - BlogDetailView (RetrieveUpdateDestroyAPIView): queryset = Blog.objects.all(), serializer_class = BlogSerializer, lookup_field = 'pk'
 - CommentDetailView (RetrieveUpdateDestroyAPIView): queryset = Comment.objects.all(), serializer_class = CommentSerializer, lookup_field = 'pk'

5. To view the COMMENTS path:

Api Root / Comments

Comments

[OPTIONS](#)[GET](#) ▾

GET /api/v1/comments/

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

```
[  
  {  
    "id": 4,  
    "comment": "this is comment #1",  
    "blog": 2  
  },  
  {  
    "id": 5,  
    "comment": "this is comment #2",  
    "blog": 2  
  }  
]
```

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

Comment

This is a comment #1

Blog

this is a blog 3

[POST](#)

6. To delete the specific comment:

The screenshot shows a browser window displaying the Django REST framework API. The URL in the address bar is `127.0.0.1:8000/api/v1/comments/7/`. The page title is "Django REST framework" and the sub-page title is "Comment Detail".

On the right side of the page, there are three buttons: "DELETE" (red), "OPTIONS" (blue), and "GET" (blue). A red box highlights the "DELETE" button.

On the left side, there is a "Raw data" section containing the following JSON response:

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

{
    "id": 7,
    "comment": "test comment",
    "blog": 3
}
```

Below this, there is a "HTML form" section with two fields:

- Comment:** A text input field containing "test comment".
- Blog:** A dropdown menu containing "this is a blog 3".

At the bottom right of the "HTML form" section is a "PUT" button.

7. View the deleted record:

The screenshot shows a browser window with the URL `127.0.0.1:8000/api/v1/comments/7/` highlighted with a red box. The page title is "Django REST framework" and the user is logged in as "api_djangoadmin". The main content area shows a "Comment Detail" page for a comment that does not exist. The response code is "HTTP 404 Not Found" with the following headers: "Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS", "Content-Type: application/json", and "Vary: Accept". The error message is:

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

. Below this, there is a form with fields for "Comment" and "Blog", and a "PUT" button. The "Raw data" tab is selected.

8.