

## Topic: 15. DRF Filtering

Speaker: Personal / Notebook: API Development using Django Framework



### Django Filtering:

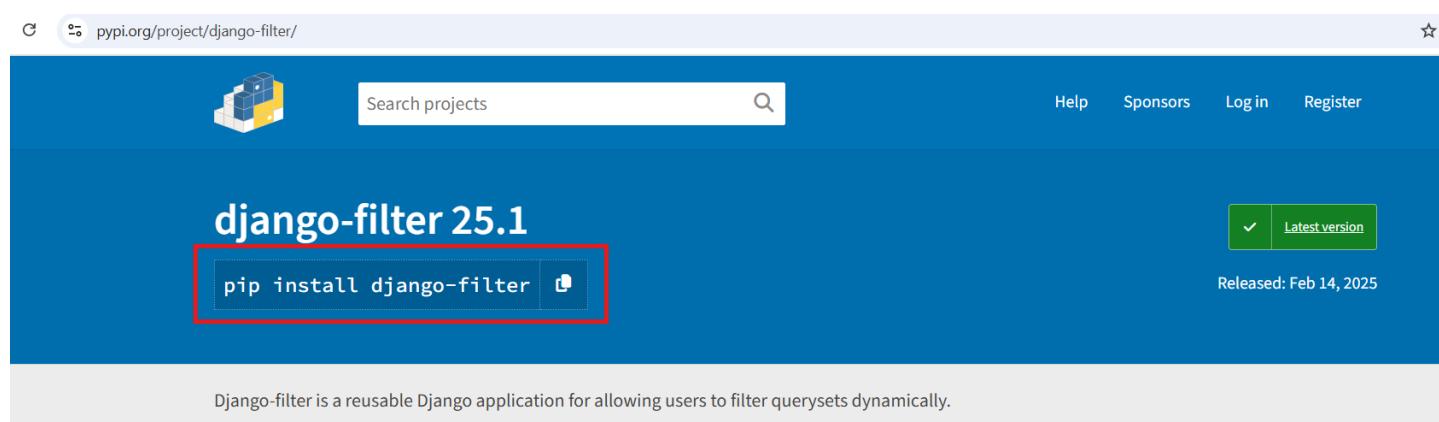
1. Filtering allows searching for certain record(s) based on given criteria. According to [Django Rest API Framework documentation](#):

*The default behavior of REST framework's generic list views is to return the entire queryset for a model manager. Often you will want your API to restrict the items that are returned by the queryset.*

*The simplest way to filter the queryset of any view that subclasses GenericAPIView is to override the .get\_queryset() method.*

*Overriding this method allows you to customize the queryset returned by the view in a number of different ways.*

2. Install the library for Django Filter, go [here](#). Go for the the latest version.

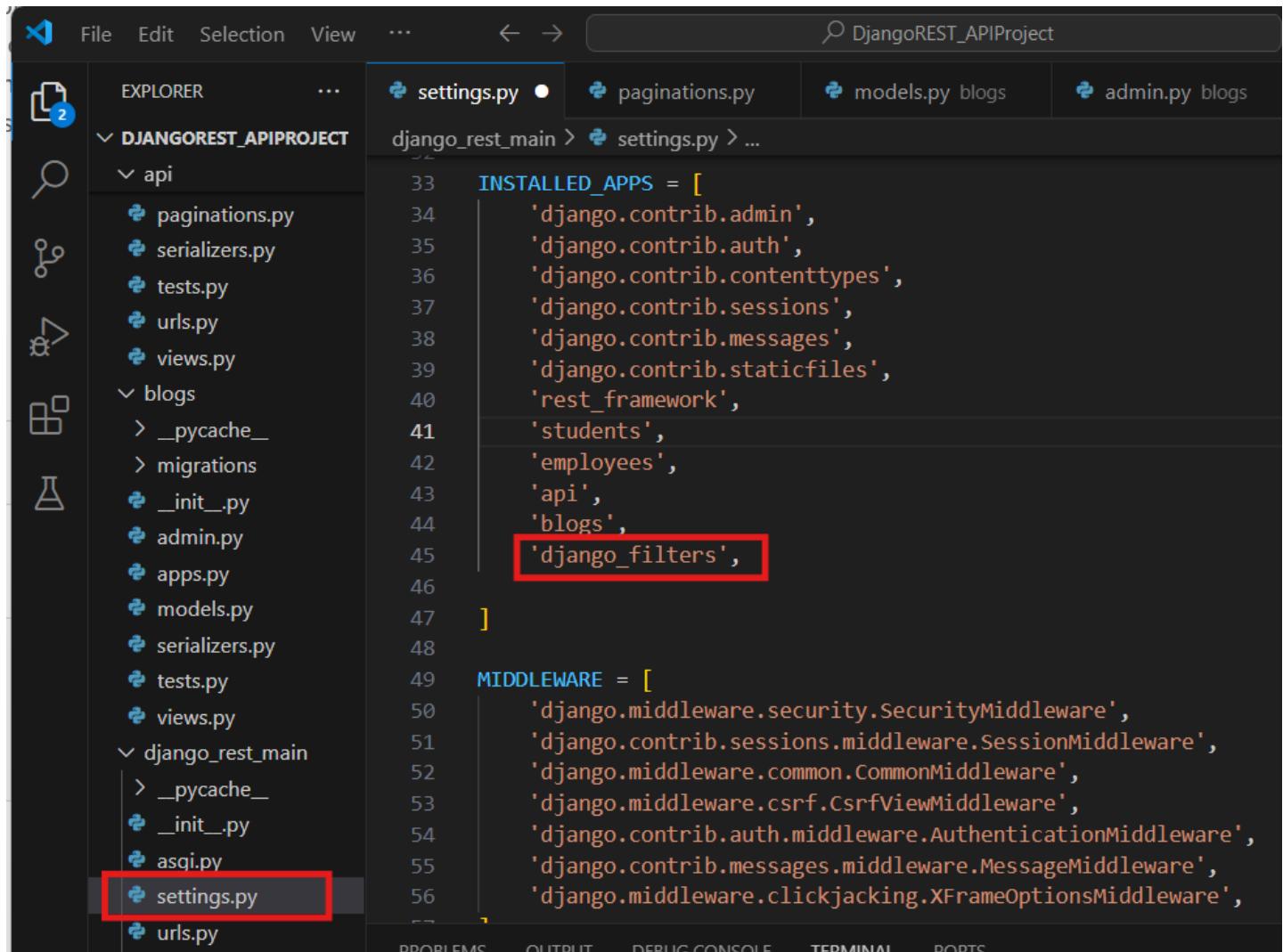


A screenshot of the PyPI project page for 'django-filter 25.1'. The page has a blue header with the PyPI logo, a search bar, and navigation links for Help, Sponsors, Log in, and Register. The main title is 'django-filter 25.1'. Below it is a button with the text 'pip install django-filter' with a copy icon. To the right is a green button labeled 'Latest version' with a checkmark. Below the title, it says 'Released: Feb 14, 2025'. A description box states: 'Django-filter is a reusable Django application for allowing users to filter querysets dynamically.' The page is divided into 'Navigation' and 'Project description'. The 'Project description' section contains the following text: 'Django-filter is a reusable Django application allowing users to declaratively add dynamic `QuerySet` filtering from URL parameters.' It also links to 'read the docs' and shows 'Coverage 99%' and 'PyPI package 25.1'. At the bottom, a terminal window shows the command '\$ pip install django-filter' being run, followed by the output of the package being downloaded and installed.

```
rosil@LearnCodeRepeat MINGW64 /c/Users/rosil/OneDrive/Documents/MyCodingCareer/Django Projects/API Dev REST_APIProject
● $ pip install django-filter
Collecting django-filter
  Downloading django_filter-25.1-py3-none-any.whl.metadata (5.1 kB)
  Requirement already satisfied: Django>=4.2 in c:\users\rosil\onedrive\documents\mycodingcareer\django\venv\lib\site-packages (from django-filter)

```

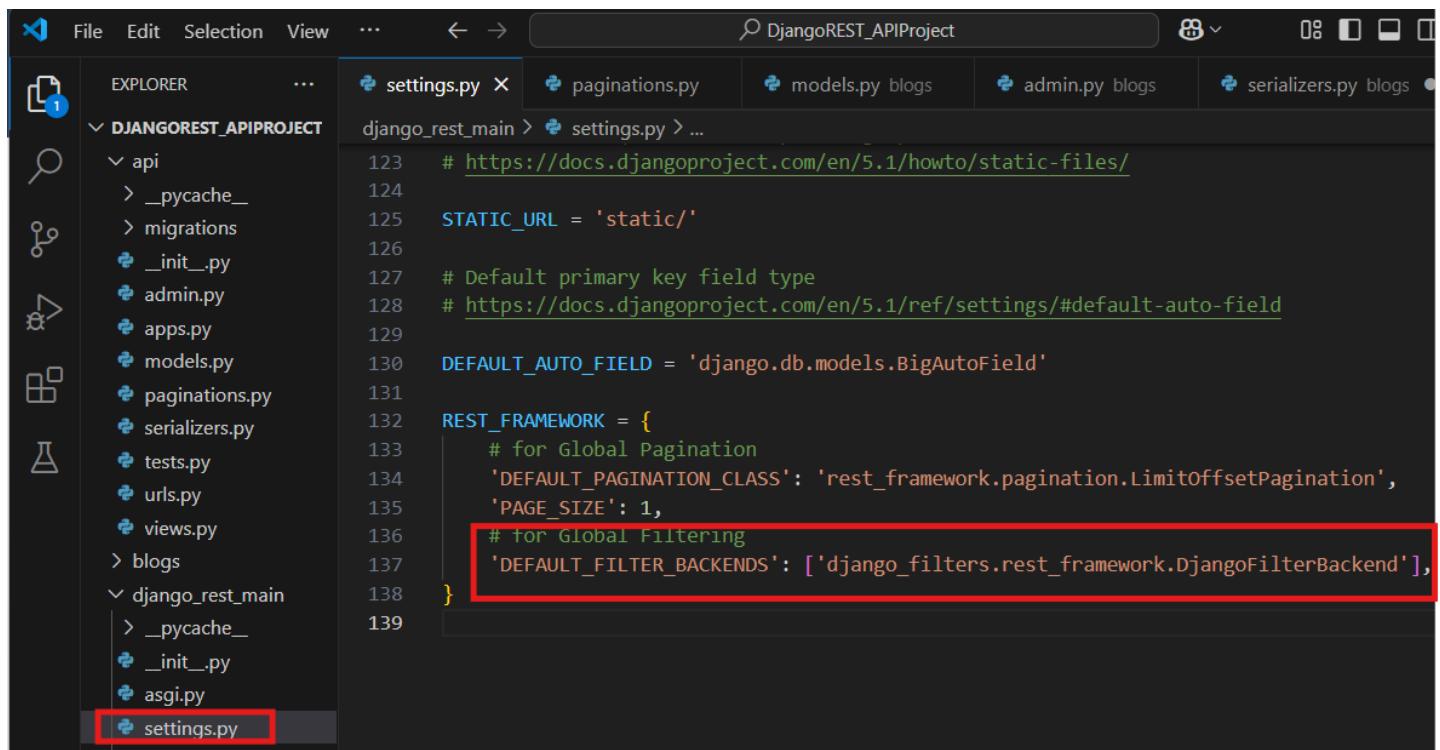
3. Register your Django package in SETTINGS.PY



The screenshot shows the VS Code interface with the Django REST API Project open. The Explorer sidebar on the left shows the project structure, including the `api`, `blogs`, and `django_rest_main` directories. The `settings.py` file in the `django_rest_main` directory is the active editor. A red box highlights the `settings.py` file in the Explorer and the `django_filters` entry in the code editor. The code editor displays the following `INSTALLED_APPS` configuration:

```
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     'employees',
43     'api',
44     'blogs',
45     'django_filters',
46 ]
47
48 MIDDLEWARE = [
49     'django.middleware.security.SecurityMiddleware',
50     'django.contrib.sessions.middleware.SessionMiddleware',
51     'django.middleware.common.CommonMiddleware',
52     'django.middleware.csrf.CsrfViewMiddleware',
53     'django.contrib.auth.middleware.AuthenticationMiddleware',
54     'django.contrib.messages.middleware.MessageMiddleware',
55     'django.middleware.clickjacking.XFrameOptionsMiddleware',
56 ]
```

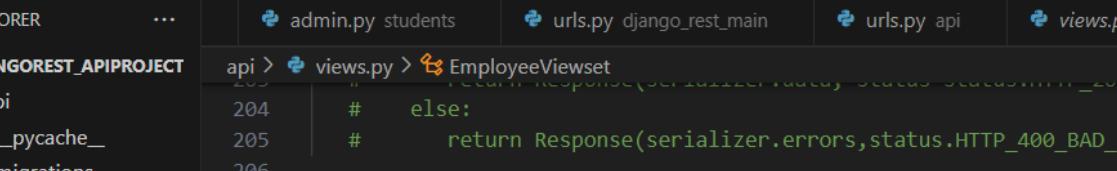
4. To set the GLOBAL FILTER, do this just like how you did the GLOBAL PAGINATION.



The screenshot shows the VS Code interface with the Django REST API Project open. The Explorer sidebar on the left shows the project structure, including the `api`, `blogs`, and `django_rest_main` directories. The `settings.py` file in the `django_rest_main` directory is the active editor. A red box highlights the `settings.py` file in the Explorer and the `DEFAULT_FILTER_BACKENDS` entry in the code editor. The code editor displays the following `REST_FRAMEWORK` configuration:

```
123 # https://docs.djangoproject.com/en/5.1/howto/static-files/
124
125 STATIC_URL = 'static/'
126
127 # Default primary key field type
128 # https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field
129
130 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
131
132 REST_FRAMEWORK = {
133     # for Global Pagination
134     'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.LimitOffsetPagination',
135     'PAGE_SIZE': 1,
136     # for Global Filtering
137     'DEFAULT_FILTER_BACKENDS': ['django_filters.rest_framework.DjangoFilterBackend'],
138 }
139
```

5. Update our API\VIEWS.PY:



```
File Edit Selection View ... < > DjangoREST_APIProject 88 08

EXPLORER ...
Djangorest_apiProject
  api
    > __pycache__
    > migrations
    __init__.py
    admin.py
    apps.py
    models.py
    paginations.py
    serializers.py
    tests.py
    urls.py
  views.py
    admin.py students
    urls.py django_rest_main
    urls.py api
    views.py blogs

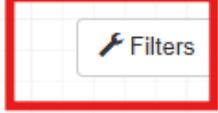
api > views.py > EmployeeViewSet
203
204     #     else:
205     #         return Response(serializer.errors, status.HTTP_400_BAD_REQUEST)
206
207
208     # uses Viewsets the easier way
209     class EmployeeViewSet(viewsets.ModelViewSet):
210         queryset = Employee.objects.all()
211         serializer_class = EmployeeSerializer
212         # uses the customized pagination instead of the default class
213         pagination_class = CustomPagination
214
215         # uses the filtering by employee's designation
216         filterset_fields = ['designation']
217
```

→ ⌂ ⓘ 127.0.0.1:8000/api/v1/employees/?designation... 🔍 ⭐ ⌂ ⌂ ⌂ ⌂ ⌂

Django REST framework api\_djangoadmin

Api Root / Employee Viewset List

# Employee Viewset List

 [OPTIONS](#) [GET](#) ▾

« 1 2 3 »

GET /api/v1/employees/?designation=

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

```
{  
    "next": "http://127.0.0.1:8000/api/v1/employees/?designation=&page-num=2",  
    "previous": null,  
    "count": 3,  
    "page_size": 1,  
    "results": [  
        {  
            "id": 1,  
            "emp_id": "EMP001",  
            "emp_name": "Rosilie",  
            "designation": "Software Developer"  
        }  
    ]  
}
```

Raw data [HTML form](#)

Emp id

Emp name

Designation

[POST](#)

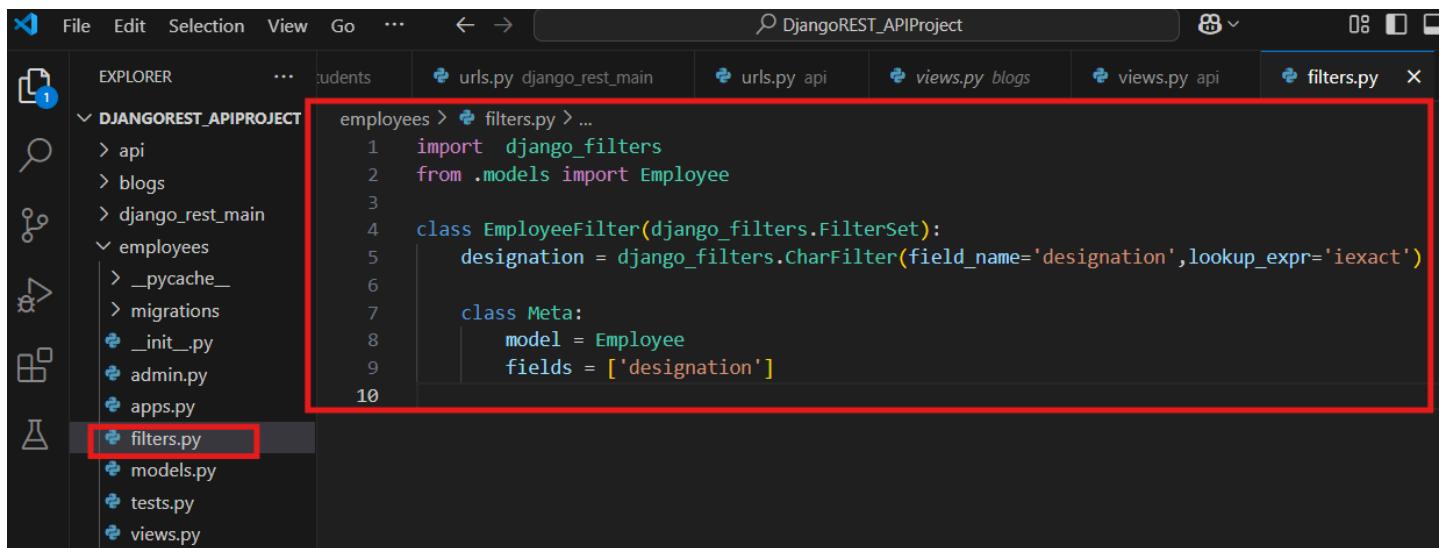
The screenshot shows a browser window displaying the Django REST framework API browser. The URL is 127.0.0.1:8000/api/v1/employees/?designation=AI%20Engineer. The page title is "Django REST framework" and the sub-page title is "Filters". The main content is titled "Field filters" and shows a "Designation" input field with "AI Engineer" typed in, and a "Submit" button. Below the input field, the response status is "HTTP 200", and the response headers are "Allow: GET, POST, HEAD, OPTIONS", "Content-Type: application/json", and "Vary: Accept". The response body is a JSON object:

```
{  
    "next": null,  
    "previous": null,  
    "count": 1,  
    "page_size": 1,  
    "results": [  
        {  
            "id": 6,  
            "emp_id": "EMP004",  
            "emp_name": "Arnel Zethus",  
            "designation": "AI Engineer"  
        }  
    ]  
}
```

At the bottom, there are "Raw data" and "HTML form" tabs. The "HTML form" tab is active, showing three input fields: "Emp id", "Emp name", and "Designation", each with a corresponding text input box. A "POST" button is located to the right of the input fields.

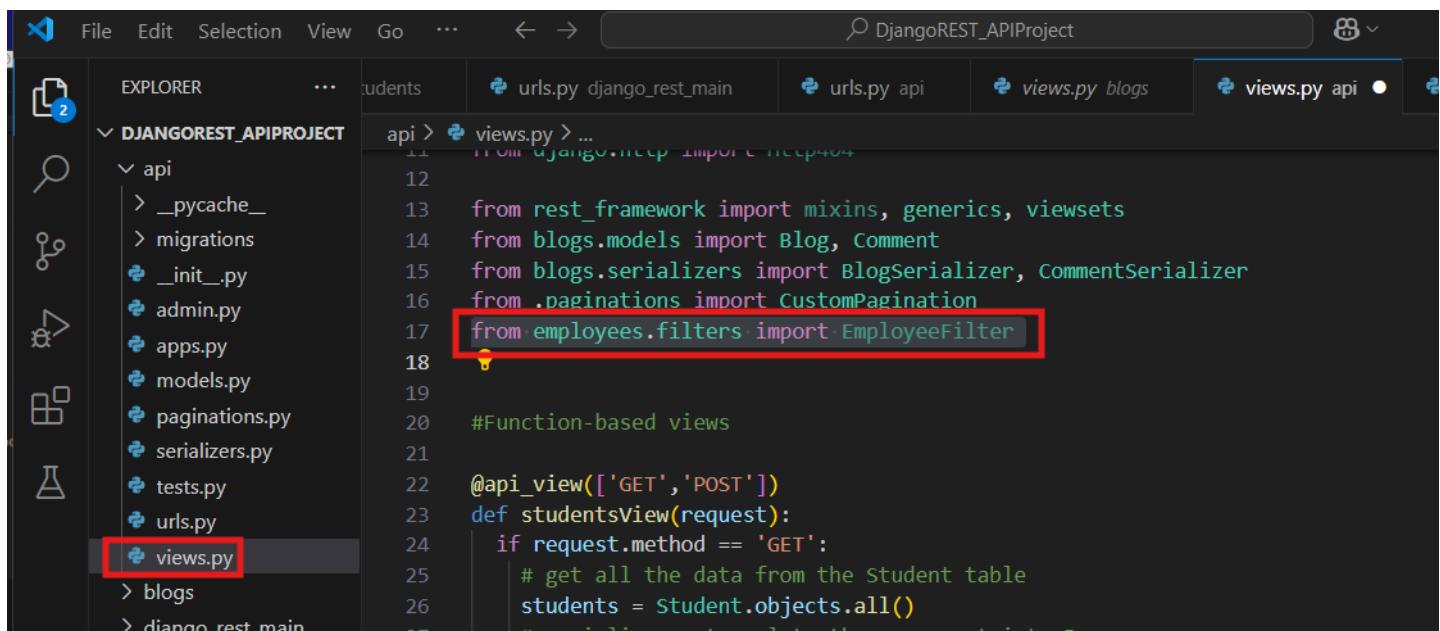
\* You need to type the exact keyword to find an exact match, so this is a case-sensitive filter. To solve this, we use custom filters instead.

6. To allow for case-insensitive filter, create a new file FILTERS.PY in the EMPLOYEES app or you can have this in API folder.



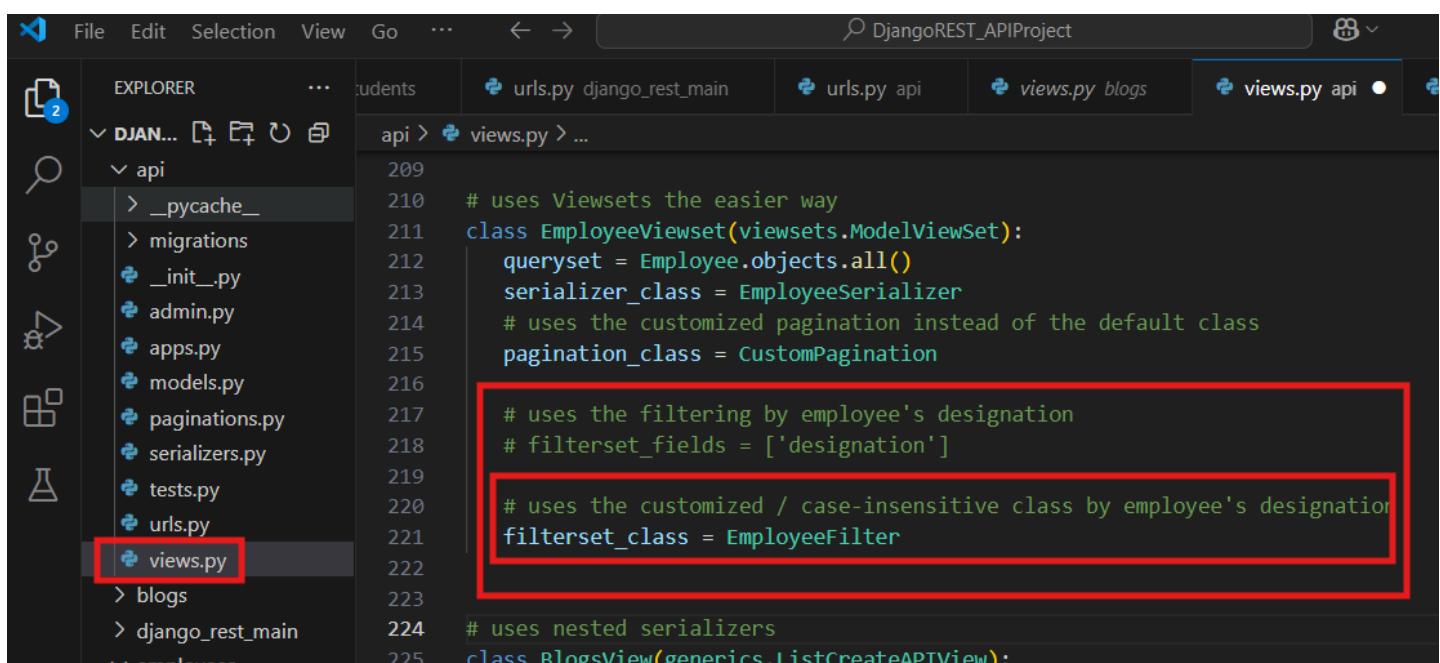
```
employees > filters.py > ...
1  import django_filters
2  from .models import Employee
3
4  class EmployeeFilter(django_filters.FilterSet):
5      designation = django_filters.CharFilter(field_name='designation', lookup_expr='iexact')
6
7  class Meta:
8      model = Employee
9      fields = ['designation']
```

7. Then update our VIEWS.PY. Import the class EmployeeFilter from the Employees\Filters.py



```
api > views.py > ...
11
12  from django.http import JsonResponse
13  from rest_framework import mixins, generics, viewsets
14  from blogs.models import Blog, Comment
15  from blogs.serializers import BlogSerializer, CommentSerializer
16  from .pagination import CustomPagination
17  from employees.filters import EmployeeFilter
18
19
20  #Function-based views
21
22  @api_view(['GET', 'POST'])
23  def studentsView(request):
24      if request.method == 'GET':
25          # get all the data from the Student table
26          students = Student.objects.all()
27          # serialize or translate the query set into json
```

To use the new filter class:



```
api > views.py > ...
209
210  # uses Viewsets the easier way
211  class EmployeeViewSet(viewsets.ModelViewSet):
212      queryset = Employee.objects.all()
213      serializer_class = EmployeeSerializer
214      # uses the customized pagination instead of the default class
215      pagination_class = CustomPagination
216
217      # uses the filtering by employee's designation
218      # filterset_fields = ['designation']
219
220      # uses the customized / case-insensitive class by employee's designation
221      filterset_class = EmployeeFilter
222
223
224      # uses nested serializers
225      class BlogsView(generics.ListCreateAPIView):
```

8. Now even if we search without minding the lowercase or uppercase, we can still find the record:

The screenshot shows the Django REST framework's EmployeeViewSet List page. At the top, a red box highlights the URL: 127.0.0.1:8000/api/v1/employees/?designation=AI+engineer. Below the URL, a 'Filters' modal is open, with a red box highlighting the 'Designation' input field containing 'AI engineer'. A 'Submit' button is below the input field. The main content area shows a JSON response with a red box highlighting a specific result: { "id": 6, "emp\_id": "EMP004", "emp\_name": "Anel Zethus", "designation": "AI Engineer" }. At the bottom, there are 'Raw data' and 'HTML form' buttons, and a 'POST' button for creating new data.

9. To add filters like by name and ID, we update our FILTERS.PY :

If we add the 2 filter keywords, we search the record for these 2 criteria.

The screenshot shows a code editor with the 'filters.py' file open. The file contains the following code:

```
1  import django_filters
2  from .models import Employee
3
4  class EmployeeFilter(django_filters.FilterSet):
5      designation = django_filters.CharFilter(field_name='designation', lookup_expr='iexact')
6      emp_name = django_filters.CharFilter(field_name='emp_name', lookup_expr='icontains')
7
8  class Meta:
9      model = Employee
10     fields = ['designation', 'emp_name']
```

Two specific sections of the code are highlighted with red boxes: the 'designation' filter definition and the 'fields' list in the 'Meta' class.

① 127.0.0.1:8000/api/v1/employees/?designation=Web+designer&emp\_name=Tammy

Django REST framework

Employee Views

GET /api/v1/employees/?designation=Web+designer&emp\_name=Tammy

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{ "next": null, "previous": null, "count": 1, "page_size": 2, "results": [ { "id": 13, "emp_id": "EMP005", "emp_name": "Tammy Chow", "designation": "Web Designer" } ] }
```

Filters

Field filters

Designation:

Emp name contains:

Submit

Raw data

HTML form

Emp id

Emp name

Designation

POST

This screenshot shows the Django REST framework's Employee Views page. A modal dialog titled 'Filters' is open, containing 'Field filters' for 'Designation' (Web designer) and 'Emp name contains' (Tammy). Below the dialog, a JSON response is displayed, showing a single result with id 13, emp\_id EMP005, emp\_name Tammy Chow, and designation Web Designer. At the bottom, there are 'Raw data' and 'HTML form' buttons, and a form with fields for Emp id, Emp name, and Designation, with a POST button.

② 127.0.0.1:8000/api/v1/employees/?designation=&emp\_name=Ros

Django REST framework

Employee Views

GET /api/v1/employees/?designation=&emp\_name=Ros

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{ "next": null, "previous": null, "count": 1, "page_size": 2, "results": [ { "id": 1, "emp_id": "EMP001", "emp_name": "Rosolie", "designation": "Software Developer" } ] }
```

Filters

Field filters

Designation:

Emp name contains:

Submit

Raw data

HTML form

Emp id

Emp name

This screenshot shows the Django REST framework's Employee Views page. A modal dialog titled 'Filters' is open, containing 'Field filters' for 'Emp name contains' (Ros). Below the dialog, a JSON response is displayed, showing a single result with id 1, emp\_id EMP001, emp\_name Rosolie, and designation Software Developer. At the bottom, there are 'Raw data' and 'HTML form' buttons, and a form with fields for Emp id and Emp name.

10. To retrieve the employee records within the given ID range for example: PK or ID 5 - 10:

① 127.0.0.1:8000/api/v1/employees/?designation=&emp\_name=&id\_min=5&id\_max=10

The screenshot shows the Django REST framework's Employee Viewset List. A 'Filters' modal is open with 'ID:' set to 5 and 10. The response JSON shows two employees: one with ID 6 and another with ID 7, both within the specified range.

```

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

{
    "next": null,
    "previous": null,
    "count": 2,
    "page_size": 5,
    "results": [
        {
            "id": 6,
            "emp_id": "EMP004",
            "emp_name": "Arnel Zethus",
            "designation": "AI Engineer"
        },
        {
            "id": 7,
            "emp_id": "EMP002",
            "emp_name": "Ziggy DartVader",
            "designation": "Web Designer"
        }
    ]
}
  
```

We update our FILTERS.PY as:

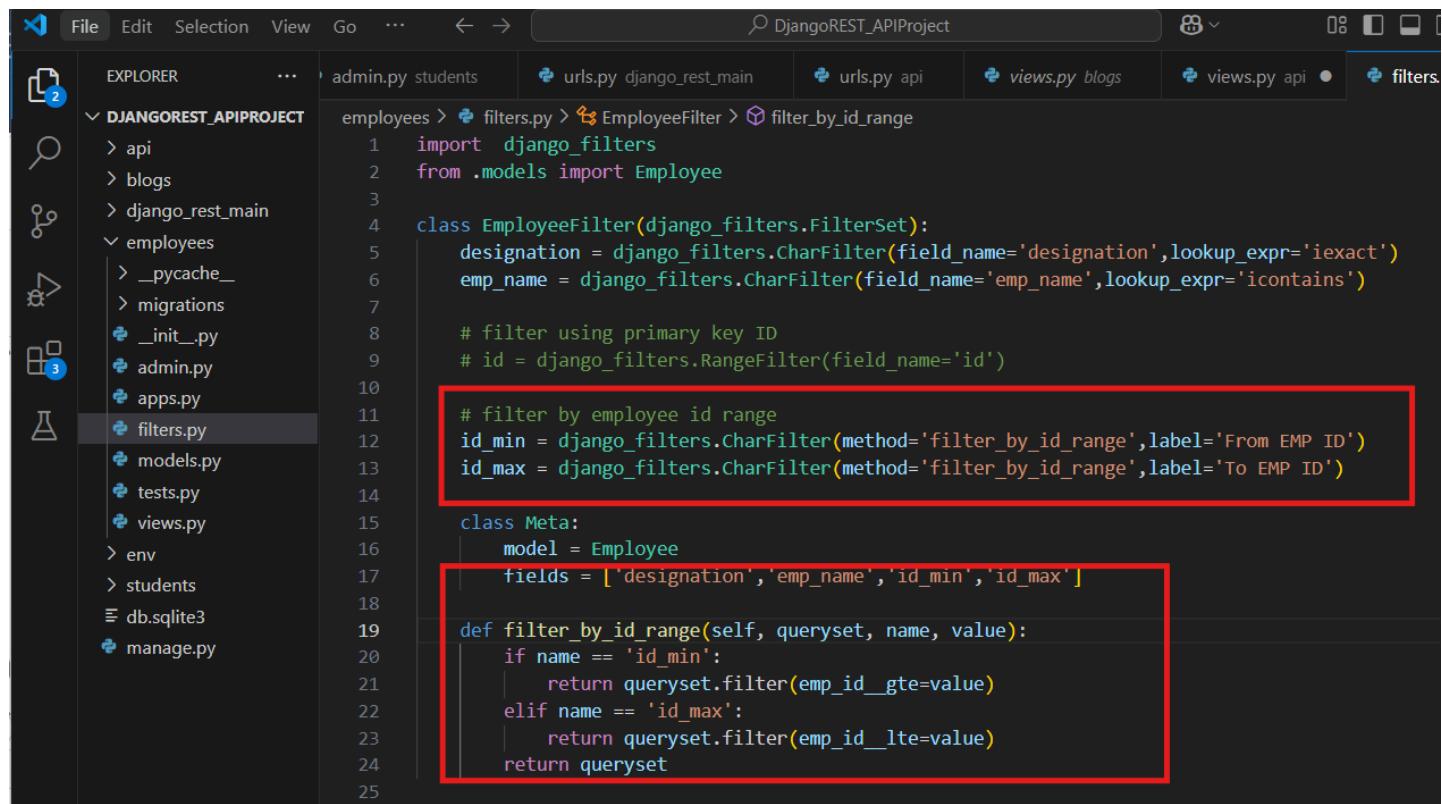
```

EXPLORER      ...
DJANGOREST_APIPROJECT
  > api
  > blogs
  > django_rest_main
  > employees
    > __pycache__
    > migrations
    > __init__.py
    > admin.py
    > apps.py
    > filters.py
    > models.py

admin.py students      urls.py django_rest_main      urls.py api      views.py blogs      views.py api      filters.py

employees > filters.py > ...
1  import django_filters
2  from .models import Employee
3
4  class EmployeeFilter(django_filters.FilterSet):
5      designation = django_filters.CharFilter(field_name='designation', lookup_expr='iexact')
6      emp_name = django_filters.CharFilter(field_name='emp_name', lookup_expr='icontains')
7      id = django_filters.RangeFilter(field_name='id')
8
9      class Meta:
10         model = Employee
11         fields = ['designation', 'emp_name', 'id']
12
  
```

11. But to use the EMP\_ID (ex. EMP005-EMP008) with CharField as our range filter, we update the FILTERS.PY as:



File Edit Selection View Go ... ← → ○ DjangoREST\_APIProject

EXPLORER ... admin.py students urls.py django\_rest\_main urls.py api views.py blogs views.py api filters.py

DJANGOREST\_APIPROJECT

- api
- blogs
- django\_rest\_main
- employees
  - \_\_pycache\_\_
  - migrations
  - \_\_init\_\_.py
  - admin.py
  - apps.py
  - filters.py**
  - models.py
  - tests.py
  - views.py
- env
- students
- db.sqlite3
- manage.py

```
employees > filters.py > EmployeeFilter > filter_by_id_range
1 import django_filters
2 from .models import Employee
3
4 class EmployeeFilter(django_filters.FilterSet):
5     designation = django_filters.CharFilter(field_name='designation', lookup_expr='iexact')
6     emp_name = django_filters.CharFilter(field_name='emp_name', lookup_expr='icontains')
7
8     # filter using primary key ID
9     # id = django_filters.RangeFilter(field_name='id')
10
11     # filter by employee id range
12     id_min = django_filters.CharFilter(method='filter_by_id_range', label='From EMP ID')
13     id_max = django_filters.CharFilter(method='filter_by_id_range', label='To EMP ID')
14
15     class Meta:
16         model = Employee
17         fields = ['designation', 'emp_name', 'id_min', 'id_max']
18
19     def filter_by_id_range(self, queryset, name, value):
20         if name == 'id_min':
21             return queryset.filter(emp_id__gte=value)
22         elif name == 'id_max':
23             return queryset.filter(emp_id__lte=value)
24         return queryset
25
```

Before the Filter by Emp\_ID:

## Django REST framework

Api Root / Employee Viewset List

# Employee Viewset List

[Filters](#) [OPTIONS](#) [GET](#) ▾

« 1 2 »

`GET /api/v1/employees/`

HTTP 200 OK

Allow: GET, POST, HEAD, OPTIONS

Content-Type: application/json

Vary: Accept

```
{  
    "next": "http://127.0.0.1:8000/api/v1/employees/?page-num=2",  
    "previous": null,  
    "count": 6,  
    "page_size": 5,  
    "results": [  
        {  
            "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": "EMP005",  
            "emp_name": "Ziggy DartVader",  
            "designation": "Web Designer"  
        },  
        {  
            "id": 11,  
            "emp_id": "EMP008",  
            "emp_name": "Dylan",  
            "designation": "Network Engineer"  
        },  
        {  
            "id": 12,  
            "emp_id": "EMP009",  
            "emp_name": "Lexine",  
            "designation": "Graphics Designer"  
        }  
    ]  
}
```

After:

Django REST framework

Filters

Field filters

Designation:

Emp name contains:

From EMP ID:  
EMP003

To EMP ID:  
EMP009

Submit

HTTP 200 OK

Allow: GET

Content-Type: application/json

Vary: Accept

{

  "next": null,

  "previous": null,

  "count": 4,

  "page": 1,

  "results": [

    {

      "id": 6,

      "emp\_id": "EMP004",

      "emp\_name": "Arnel Zethus",

      "designation": "AI Engineer"

    },

    {

      "id": 7,

      "emp\_id": "EMP005",

      "emp\_name": "Ziggy DartVader",

      "designation": "Web Designer"

    },

    {

      "id": 11,

      "emp\_id": "EMP008",

      "emp\_name": "Dylan",

      "designation": "Network Engineer"

    },

    {

      "id": 12,

      "emp\_id": "EMP009",

      "emp\_name": "Lexine",

      "designation": "Graphics Designer"

    }

  ]

}

The screenshot shows a Django REST framework API endpoint for employees. A modal dialog titled 'Field filters' is open, allowing users to filter results by designation, emp name, and emp ID. The 'From EMP ID' field contains 'EMP003' and the 'To EMP ID' field contains 'EMP009', both highlighted with a red box. A large red box also highlights the JSON response data, which shows a list of employees with IDs 6, 7, 11, and 12, and emp IDs EMP004, EMP005, EMP008, and EMP009 respectively. The modal has a 'Submit' button at the bottom.

```
HTTP 200 OK
Allow: GET
Content-Type: application/json
Vary: Accept
{
  "next": null,
  "previous": null,
  "count": 4,
  "page": 1,
  "results": [
    {
      "id": 6,
      "emp_id": "EMP004",
      "emp_name": "Arnel Zethus",
      "designation": "AI Engineer"
    },
    {
      "id": 7,
      "emp_id": "EMP005",
      "emp_name": "Ziggy DartVader",
      "designation": "Web Designer"
    },
    {
      "id": 11,
      "emp_id": "EMP008",
      "emp_name": "Dylan",
      "designation": "Network Engineer"
    },
    {
      "id": 12,
      "emp_id": "EMP009",
      "emp_name": "Lexine",
      "designation": "Graphics Designer"
    }
  ]
}
```