

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.

Search projects

Help Sponsors Log in Register

django-filter 25.1

Latest version

```
pip install django-filter
```

Released: Feb 14, 2025

Django-filter is a reusable Django application for allowing users to filter querysets dynamically.

Navigation

Project description

Release history

Project description

Django-filter is a reusable Django application allowing users to declaratively add dynamic `QuerySet` filtering from URL parameters.

Full documentation on [read the docs](#).

Coverage: 99% | PyPI package: 25.1

```
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
```

3. Register your Django package in `SETTINGS.PY`

```
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
49 MIDDLEWARE = [
50     'django.middleware.security.SecurityMiddleware',
51     'django.contrib.sessions.middleware.SessionMiddleware',
52     'django.middleware.common.CommonMiddleware',
53     'django.middleware.csrf.CsrfViewMiddleware',
54     'django.contrib.auth.middleware.AuthenticationMiddleware',
55     'django.contrib.messages.middleware.MessageMiddleware',
56     'django.middleware.clickjacking.XFrameOptionsMiddleware',
```

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

```
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 APIVIEWS.PY:

The image shows a code editor window for a Django REST API project. The Explorer on the left displays the project structure, with the 'api' folder expanded and 'views.py' selected. The main editor shows the code for the 'EmployeeViewSet' class in 'views.py'. The code includes a comment about using Viewsets, a class definition, and several attributes: 'queryset', 'serializer_class', 'pagination_class', and 'filterset_fields'. The 'filterset_fields' attribute is highlighted with a red box.

```
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     # uses the filtering by employee's designation
215     filterset_fields = ['designation']
216
217
218 # uses the default serializer
```

Employee Viewset List

Filters

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 web browser window with the URL `127.0.0.1:8000/api/v1/employees/?designa...`. A modal titled "Filters" is open, displaying "Field filters" for the "Designation" field. The input field contains "AI Engineer" and a "Submit" button. Below the modal, a JSON response is shown, with a red box highlighting the "results" array:

```
{
  "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 of the page, there are input fields for "Emp id", "Emp name", and "Designation", along with a "POST" button.

* 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']
10
```

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

```
api > views.py > ...
11 from django.http import Http404
12
13 from rest_framework import mixins, generics, viewsets
14 from blogs.models import Blog, Comment
15 from blogs.serializers import BlogSerializer, CommentSerializer
16 from .paginations import CustomPagination
17 from employees.filters import EmployeeFilter
18
19 #Function-based views
20
21 @api_view(['GET', 'POST'])
22 def studentsView(request):
23     if request.method == 'GET':
24         # get all the data from the Student table
25         students = Student.objects.all()
26         # serialize or template the query set into JSON
```

To use the new filter class:

```
api > views.py > ...
209 # uses Viewsets the easier way
210 class EmployeeViewSet(viewsets.ModelViewSet):
211     queryset = Employee.objects.all()
212     serializer_class = EmployeeSerializer
213     # uses the customized pagination instead of the default class
214     pagination_class = CustomPagination
215
216     # uses the filtering by employee's designation
217     # filterset_fields = ['designation']
218
219     # uses the customized / case-insensitive class by employee's designation
220     filterset_class = EmployeeFilter
221
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 a web browser displaying a Django REST API interface. The URL bar shows `127.0.0.1:8000/api/v1/employees/?designation=AI+engineer`. The page title is "Employee Views". A "Filters" dialog box is open, showing a "Field filters" section with a "Designation:" label and an input field containing "AI engineer". A "Submit" button is visible below the input field. The main content area displays the JSON response for the GET request:

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

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

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.

```
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']
11
```

127.0.0.1:8000/api/v1/employees/?designation=Web+designer&emp_name=Tammy

The screenshot shows the Django REST framework interface for the 'Employee Viewset List'. A 'Filters' dialog box is open, showing 'Field filters' with 'Designation' set to 'Web designer' and 'Emp name contains' set to 'Tammy'. The 'Submit' button is highlighted. In the background, the JSON response for the query is displayed, with the first result highlighted in red:

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

Below the JSON response, there are input fields for 'Emp id', 'Emp name', and 'Designation', and a 'POST' button.

127.0.0.1:8000/api/v1/employees/?designation=&emp_name=Ros

The screenshot shows the Django REST framework interface for the 'Employee Viewset List'. A 'Filters' dialog box is open, showing 'Field filters' with 'Designation' empty and 'Emp name contains' set to 'Ros'. The 'Submit' button is highlighted. In the background, the JSON response for the query is displayed, with the first result highlighted in red:

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

Below the JSON response, there are input 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:

127.0.0.1:8000/api/v1/employees/?designation=&emp_name=&id_min=5&id_max=10

Django REST framework

Api Root / Employee Viewset List

Employee Views

GET /api/v1/employees/?designation=&emp_name=&id_min=5&id_max=10

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"
    }
  ]
}
```

Filters

Field filters

Designation:

Emp name contains:

ID: -

Submit

Raw data HTML form

Emp id

Emp name

Designation

POST

We update our 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

DJANGOREST_APIPROJECT

- api
- blogs
- django_rest_main
- employees
 - __pycache__
 - migrations
 - __init__.py
 - admin.py
 - apps.py
 - filters.py
 - models.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:

```
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:

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

Api Root

Emp

GET /api/

HTTP 200

Allow: GET

Content-Type

Vary: Accept

{

"next":

"prev":

"count":

"page":

"result":

Filters

Field filters

Designation:

Emp name contains:

From EMP ID:

To EMP ID:

Submit

```
{
  "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"
}
```