Filtering, Searching & Pagination in DRF: Building Smart and Scalable APIs

🔍 Filtering, Searching & Pagination in DRF: Building Smart and Scalable APIs

When your API starts getting a lot of data, returning everything in one response becomes inefficient and messy. Think about an e-commerce website with 10,000 products — we can't just send all 10,000 in one go! That's where **pagination**, **searching**, and **filtering** come in.
These features help you:
Improve performance
Make APIs more user-friendly
Support frontend functionality like search bars and infinite scroll
In this module, you'll learn how to use Django REST Framework's built-in support for: Django consulting services
Pagination – Split large data into pages
Searching – Allow keyword-based search
Filtering – Return only matching results

📦 Step 1: Add Pagination to Your API

Pagination divides your results into smaller chunks. You can use page numbers, limit-offset, or cursor-based pagination. Let's start with the default **PageNumberPagination**.

```
# settings.py
REST_FRAMEWORK = {
    ...
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 5,
}
```

Now when you make a GET request to your API (e.g., `/api/tasks/`), the response will look like this:

```
{
  "count": 12,
  "next": "http://localhost:8000/api/tasks/?page=2",
  "previous": null,
  "results": [...]
}
```

You can also customize pagination:

```
# core/pagination.py
from rest_framework.pagination import PageNumberPagination

class CustomPagination(PageNumberPagination):
    page_size = 10
    page_size_query_param = 'limit'
    max_page_size = 100
# settings.py
'DEFAULT_PAGINATION_CLASS': 'core.pagination.CustomPagination',
```

🔎 Step 2: Add Search Functionality

Searching allows users to find resources using keywords. DRF supports this via `SearchFilter`.

```
# settings.py
REST_FRAMEWORK = {
    ...
    'DEFAULT_FILTER_BACKENDS': [
        'rest_framework.filters.SearchFilter',
    ]
}
# core/views.py
from rest_framework import viewsets, filters
from .models import Task
from .serializers import TaskSerializer

class TaskViewSet(viewsets.ModelViewSet):
    queryset = Task.objects.all()
    serializer_class = TaskSerializer
    filter_backends = [filters.SearchFilter]
    search_fields = ['title', 'description']
```

Now your endpoint supports searches like:

```
GET /api/tasks/?search=meeting
```

This will return tasks where the word "meeting" is found in the title or description.

🎯 Step 3: Add Filtering by Fields

Filtering allows users to specify exact field matches. For this, DRF recommends using **django-filter**, a powerful filtering library.

📥 Install the Package

```
pip install django-filter
# settings.py
INSTALLED_APPS = [
    ...
    'django_filters',
]

REST_FRAMEWORK = {
    ...
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
    ]
}
```

Then, define the fields you want to filter:

```
# core/views.py
from django_filters.rest_framework import DjangoFilterBackend

class TaskViewSet(viewsets.ModelViewSet):
    ...
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ['completed']
```

Now you can make requests like:

```
GET /api/tasks/?completed=true
```

You can also create custom filter sets:

```
# core/filters.py
import django_filters
from .models import Task

class TaskFilter(django_filters.FilterSet):
    class Meta:
        model = Task
        fields = {
            'completed': ['exact'],
            'title': ['icontains'],
        }
# core/views.py
from .filters import TaskFilter

class TaskViewSet(viewsets.ModelViewSet):
    ...
    filterset_class = TaskFilter
```

📌 Summary of Key Concepts

Pagination splits results into pages and improves performance
SearchFilter allows keyword searches across multiple fields
DjangoFilterBackend supports exact field-level filtering
You can combine all three for a rich, user-friendly API experience
These features are critical when building real-world apps like job boards, marketplaces, inventory systems, and social feeds.

🧪 Challenge for Interns

Create a new model called `Event` with fields: `title`, `date`, and `location`
Implement full CRUD using ViewSet + Router
Add search support on `title` and `location`
Add filters for `date`
Enable pagination with 5 items per page

In the next module, we'll dive into handling file uploads, media storage, and building APIs for profile pictures, resumes, and documents.