Datensätze filtern

in Django RestFramework

Filtering in Django REST Framework

Was ist Filtering?

- Filtering ermöglicht es, Datensätze basierend auf bestimmten Kriterien zu filtern.
- Wichtig für große Datenmengen, um relevante Daten effizient abzurufen.
- DRF bietet mehrere Methoden, um Filtering in API-Views zu integrieren.

Arten des Filterings in DRF

1. Einfaches Query Parameter Filtering

- Filtert Daten basierend auf Parametern in der URL.
- Beispiel: GET /api/items?category=Books

2. OrderingFilter

- Sortiert die Ergebnisse basierend auf angegebenen Feldern.
- Beispiel: GET /api/items?ordering=price,-name

3. SearchFilter

- Erweiterte Filtermöglichkeiten
- Beispiel: GET /api/items?search=Suchwort

1.) Einfaches Query Parameter Filtering

Beispiel: Filtern von Items nach Kategorie (GET /api/items?category=Books)

```
class ItemListView(generics.ListAPIView):
    serializer_class = ItemSerializer

    def get_queryset(self):
        """
        Optionally restricts the returned items to a given category,
        by filtering against a `category` query parameter in the URL.
        """
        queryset = Item.objects.all()
        category = self.request.query_params.get('category', None)
        if category is not None:
            queryset = queryset.filter(category=category)
        return queryset
```

2.) OrderingFilter

Beispiel: Sortieren von Ergebnissen (GET /api/items?ordering=-date)

```
from rest_framework.filters import OrderingFilter


class ItemListView(generics.ListAPIView):
    queryset = Item.objects.all()
    serializer_class = ItemSerializer
    filter_backends = [OrderingFilter]
    ordering_fields = ['name', 'date']  # Felder, nach denen sortiert werden kann
    ordering = ['name']  # Standard-Sortierung, falls im Model nicht festgelegt
```

3.) SearchFilter

Beispiel: Filtern von Ergebnissen (GET /api/items?search=Bob)

```
from rest_framework.filters import SearchFilter

class ItemListView(generics.ListAPIView):
    queryset = Item.objects.all()
    serializer_class = ItemSerializer
    filter_backends = [SearchFilter]
    search_fields = ['name', 'description']  # Felder, die durchsucht werden können
```

Erweiterte Suchparameter

'^' Starts-with Search:

Dieser Suchmodus ermöglicht es Ihnen, nach Einträgen zu suchen, deren Werte in den spezifizierten Feldern mit dem Suchbegriff beginnen. Wenn Sie beispielsweise search_fields = ['^name'] setzen, werden für eine Suchanfrage mit dem Begriff "Example" alle Einträge zurückgegeben, deren name-Feld mit "Example" beginnt.

'=' Exact Matches:

Dieser Modus wird verwendet, um nach genauen Übereinstimmungen des Suchbegriffs zu suchen. Wenn Ihr search_fields so eingestellt ist: search_fields = ['=name'], und Sie nach "Example" suchen, werden nur Einträge zurückgegeben, deren name-Feld genau "Example" entspricht.

'@' Full-text Search:

Volltextsuche ist besonders leistungsfähig, da sie es ermöglicht, ganze Textfelder nach dem Vorkommen des Suchbegriffs zu durchsuchen. Beachten Sie jedoch, dass diese Funktionalität derzeit nur von Django's PostgreSQL-Backend unterstützt wird. Bei der Verwendung in search_fields, z.B. search_fields = ['@description'], werden alle Einträge zurückgegeben, in denen der Suchbegriff im description-Feld vorkommt.

'$' Regex Search:

Mit der Regex-Suche können Sie komplexe Suchmuster definieren. Wenn search_fields so eingestellt ist: search_fields = ['$name'], können Sie reguläre Ausdrücke verwenden, um die Suchanfrage in den name-Feldern zu definieren. Dies ist eine leistungsstarke, aber auch potenziell aufwendige Suche, da reguläre Ausdrücke komplex sein können.

GET /api/items/?search=^Ex