Filter Results in django

Filter result

* To filter result based on query parameter I first need to access that parameter.
* Then filter based on query parameter.
  + What if there is no query parameter?
    - In this case scenario I need an if statement to check if there is value coming from the request.query\_params dictionary.
  + And I need to use get method because if key does not exist it return None unlike the traditional way (using brackets will always assume that I have that keyword in query\_param dict thus app will crash).
* And return a response.

Django filters:

* Ready-made class that helps me to filter without writing the logic by myself.
  + I need it because what if I want to create a filter by more than one query parameter?
    - The logic will grow larger and more complex.
  + So, in this case scenario the Django-filters will help me do it.
* to use it I need to download it using pip:
  + package name: Django-filter.
* Then add it to apps in setting folder to apps like this: ‘django\_filters’.
* To use it I need to import Django\_filters.rest\_framework import DefaultFilterBackend
* Then I need to define two methods: filter\_backends and filter\_fields = [ ‘field\_name’]
  + Filter-backends = DefaultFilterBackend
  + Filter\_fields take array of fields
    - filter\_fields = [ ‘field\_name’, ‘field\_2\_name’]

Why create my custom filter class?

* Some filters take exact value and some others need different kind of implementation like price for example or quantity.
* The filter\_filers way has a problem, that I can’t do more complex logic in them like:
  + query string with contains value instead of exact.
  + Also, if I want to specify ranged price list the filter\_fields won’t help me.
* To solve this problem, I create my own class filter.
* How?
  + Create a filters.y file and inside it import following:
    - from Django\_filters.rest\_framework import FilterSet.
  + Inherit my custom class from FilterSet.
    - Class ProductFilter (FilterSet):
  + Then create meta class and give it following parameters:
    - Model = Product
    - Fields = {

‘field\_name’: [‘option’],

‘field\_2\_name’: [‘lt’, ‘gt’]

}

* + Fields takes dictionary.
    - Each item in dict key is the name of the field that I want to filter.
    - The value is list of options that I can use to specify how I can use it.
  + Then in the ViewSet class I add following property and give it value my class
    - instead of filter\_fields now I use filter\_class and pass to it the filter class that I created.
    - Filterset\_class = ProductFilter

Search in Django.

* Search allow me to search for an object or item based on provided keyword.
  + It’s insensitive case which means if I type small letters, I will also get capital letters.
  + How to implement?
    - Import SearchField.

From rest\_framework.filters import SearchField.

* + - Provide the SearchField to the filter\_backends property array

Filter\_backends = [SearchField]

* + - Create search\_fields = [] property and give it the fields I want to use for search.
      * Search\_fields = [‘title’, ‘description’]

Sorting

* Sorting allows me to sort data depends on sorting option, for example I can sort in ascendant order and descendant order based on field provided.
* To do this I first need to import OrderingFilter:

From rest\_framework.filters import OrderingFilter

* Add OrderingFilter to filter\_backends array.

Filter\_backends = [OrderingFilter]

* Add ordering\_fields = [filter fields array]
  + Ordering\_fields = [‘field\_1’, ‘field\_2’]
* Also, I can select related model fields.

Pagination

- Pagination is when I want to allow only n (I specify how many) items to be sent to user when requesting data by splitting them into pages

- then provide link in each page refer to next page and it will send another 10 items.

- previous link to allow user to get previous result.

Implementation:

* There are 3 ways to implement this:
  + One: Apply it to specific class:
    - From rest\_framework.pagination import PageNumberPagination
    - Inside the class add following property: pagination\_class = PageNumberPagination
    - Inside app setting inside REST\_FRAMEWORK dictionary add following:
      * ‘PAGE\_SIZE’: 10
  + Two: if I want to apply pagination to all models I use default pagination, to do that I add it inside app\_setting REST\_FRAMEWORK dictionary.
    - “DEFAULT\_PAGINATION\_CLASS”: ‘rest\_framework.pagination.PageNumberPagination’
    - ‘PAGE\_SIZE’: 10
  + Three: I create my own default pagination to avoid warning message inside terminal about providing a default pagination.
    - Create file and add pagination to it.
    - From rest\_framework.pagination import PageNumberPagination
      * Inherit my custom class from PageNumberPagination.
      * Add page\_size: n (the number of items in each link)
    - Inside the ViewSet model I add pagination\_class = CustomDefaultPagination.