Learning Objectives

- Explain what happens to querysets when filtering
- Define user-based filter, url-based filtering, and query parameter filtering
- Add user-based filtering and url-based filtering to Blango

Clone Blango Repo

Clone Blango Repo

Before we continue, you need to clone the `blango` repo so you have all of your code. You will need the SSH information for your repo.

In the Terminal

- Clone the repo. Your command should look something like this:

```
git clone git@github.com:<your_github_username>/blango.git
```

- You should see a `blango` directory appear in the file tree.

You are now ready for the next assignment.

User-Based Filtering

Queryset

So far we've used static querysets when building APIs. That is, the `queryset` attribute is defined on the class and doesn't differ for each request.

Fundamentally, filtering data requires changing the queryset that is used to build a response. Django Rest Framework allows us to do just about anything we want to create a queryset, and we have opportunities to work with the `Request` object, and URL and query parameters.

To change the queryset that's used by the view, we can implement the `get_queryset()` method. In its default implementation, `get_queryset()` just returns the `queryset` attribute. But let's look at how we can use it to perform some filtering.

User-Based Filtering

The `get_queryset()` method takes no arguments, so in order to filter we need to use attributes/properties that are set on `self`. The first one we'll use is `request`. Of course, we can access any of the `request` attributes that we want, such as HTTP headers, but in this example we're just interested in filtering by the request's `user`.

We want to make our `PostViewSet` a little bit more "private", and restrict access to unpublished posts. Let's make it so:

- admin/staff users get all `Posts`.
- logged-in users get published `Posts` or those that they've authored.
- anonymous users get published `Posts` only.

We can implement the `get_queryset()` method like this:

```
from django.db.models import Q
from django.utils import timezone

class PostViewSet(viewsets.ModelViewSet):
    # we'll still refer to this in `get_queryset()`
    queryset = Post.objects.all()

    def get_queryset(self):
        if self.request.user.is_anonymous:
            # published only
            return
        self.queryset.filter(published_at__lte=timezone.now())

        if not self.request.user.is_staff:
            # allow all
            return self.queryset

        # filter for own or
        return self.queryset.filter(
            Q(published_at__lte=timezone.now()) |
        Q(author=self.request.user)
        )

    # other methods/attributes omitted
```

Also, remember that we are caching the response from the `list()` method. Since the list of `Posts` now changes with each user, we need to make sure we add the `vary_on_headers()` decorator to it, with `Authorization` and `Cookie` as arguments:

```
    @method_decorator(cache_page(120))
    @method_decorator(vary_on_headers("Authorization",
        "Cookie"))
    def list(self, *args, **kwargs):
        return super(PostViewSet, self).list(*args, **kwargs)
```

Since we've made this change to the `get_queryset()` method it applies to all the API action methods. The `retrieve()` method filters the existing queryset by the `Post`'s PK. That is, it performs `self.get_queryset().get(pk=pk)` rather than `Post.objects.get(pk=pk)`, so whatever filtering `get_queryset()` does is already taken into account.

Try It Out

Now you'll implement user-based filtering in your Blango app. In `blog/api/views.py` start by adding the imports we need.

```
from django.db.models import Q
from django.utils import timezone
```

Next, implement this `get_queryset()` method on `PostViewSet`.

```
    def get_queryset(self):
        if self.request.user.is_anonymous:
            # published only
            return
        self.queryset.filter(published_at__lte=timezone.now())

        if not self.request.user.is_staff:
            # allow all
            return self.queryset

        # filter for own or
        return self.queryset.filter(
            Q(published_at__lte=timezone.now()) |
        Q(author=self.request.user)
        )
```

Finally make sure we're varying the `list()` response on the `Cookie` and `Authorization` headers.

```
    @method_decorator(cache_page(120))
    @method_decorator(vary_on_headers("Authorization",
        "Cookie"))
    def list(self, *args, **kwargs):
        return super(PostViewSet, self).list(*args, **kwargs)
```

Then, try it out. Now if you're a logged-out user, you won't see any unpublished `Posts`. If you're logged in, you'll see published and the ones for which you're an author. And, if you're a staff (Django Admin) user you'll see them all. You'll need a couple of user accounts to test this out.

View Blog

Next, we'll look at filtering based on parts of the URL.

URL-Based Filtering

URL-Based Filtering

DRF also allows filtering based on parameters in the URL path. Note that this is not the same as filtering by query parameters, the values that come after a `?` in the URL (e.g. `?filter_val_1=foo&filter_val_2=bar`). We'll briefly look at this type of filtering in a little bit.

This type of URL filtering requires setting up a custom URL pattern to your view or viewset. If using viewsets, you might only need to set up a custom URL pattern if using action methods with the `action()` decorator doesn't do what you need.

Once you have a URL set up with a named parameter, it can be accessed in the view or viewset using the `kwargs` attribute. This is a dictionary containing all the named parameters in the URL.

Let's update our `PostViewSet` so we can get a list of `Posts` for a set of named periods:

- `new`: `Posts` published in the last hour.
- `today`: `Posts` were published today.
- `week`: `Posts` published in the last week.

To do this we'll add a URL pattern that maps to the `list()` method on `PostViewSet`:

```
    path(
        "posts/by-time/<str:period_name>/",
        PostViewSet.as_view({"get": "list"}),
        name="posts-by-time",
    ),
```

Here we're creating a URL pattern that maps `GET` requests to the `list()` method on our view set. This is what the normal `Posts` list URL pattern does too, however it's only if we access through this new URL we'll get the named parameter `period_name` set in `kwargs`.

To filter by this parameter in `PostViewSet`, we'll make some more changes to `get_queryset()`. Assume we have applied the user filtering rules to build the `queryset` variable. We can then access the parameter in the `self.kwargs` dictionary, and perform additional filtering on `queryset`:

```
class PostViewSet(viewsets.ModelViewSet):
    # existing attributes/methods omitted

    def get_queryset(self):
        # queryset has been set by applying user filtering rules

        # fetch the period_name URL parameter from self.kwargs
        time_period_name = self.kwargs.get("period_name")

        if not time_period_name:
            # no further filtering required
            return queryset

        if time_period_name == "new":
            return
        queryset.filter(published_at__gte=timezone.now() -
        timedelta(hours=1))
        elif time_period_name == "today":
            return queryset.filter(
                published_at__date=timezone.now().date(),
            )
        elif time_period_name == "week":
            return
        queryset.filter(published_at__gte=timezone.now() -
        timedelta(days=7))
        else:
            raise Http404(
                f"Time period {time_period_name} is not valid,
        should be "
                f"'new', 'today' or 'week'"
            )
```

As you can see, the filtering we're doing is not particularly difficult, we're just applying mostly standard Django filters. Of course you can make them as complex as you need.

Try It Out

Update your `get_queryset()` method to add the time period based filtering. You'll need to add these imports first:

```
from datetime import timedelta
from django.http import Http404
```

Then update the `get_queryset()` method like so:

```
    def get_queryset(self):
        if self.request.user.is_anonymous:
            # published only
            queryset =
        self.queryset.filter(published_at__lte=timezone.now())

        elif not self.request.user.is_staff:
            # allow all
            queryset = self.queryset
        else:
            queryset = self.queryset.filter(
                Q(published_at__lte=timezone.now()) |
        Q(author=self.request.user)
            )

        time_period_name = self.kwargs.get("period_name")

        if not time_period_name:
            # no further filtering required
            return queryset

        if time_period_name == "new":
            return queryset.filter(
                published_at__gte=timezone.now() -
        timedelta(hours=1)
            )
        elif time_period_name == "today":
            return queryset.filter(
                published_at__date=timezone.now().date(),
            )
        elif time_period_name == "week":
            return
        queryset.filter(published_at__gte=timezone.now() -
        timedelta(days=7))
        else:
            raise Http404(
                f"Time period {time_period_name} is not valid,
        should be "
                f"'new', 'today' or 'week'"
            )
```

Notice that instead of returning once we've determined the queryset based on the user, we assign it to the `queryset` variable, and apply further filtering only if `period_name` is provided.

You then need to set up the URL pattern to point to this view. Open your `blog/api/urls.py` file and add this pattern:

Open api/urls.py

```
    path(
        "posts/by-time/<str:period_name>/",
        PostViewSet.as_view({"get": "list"}),
        name="posts-by-time",
    ),
```

It should be added after the `router` include pattern (this one):

```
path("", include(router.urls)),
```

Load up the URL, you can use a browser or Postman. The paths will be `/api/v1/posts/by-time/new/`, `/api/v1/posts/by-time/today/` and `/api/v1/posts/by-time/week/`. You might need to use the Django admin to alter the publication date of some of the posts so that some data shows up.

View Blog

You should also note that since we have made the changes to the `get_queryset()` method the caching applied to `list()` will still work, we don't have to update it. When Django generates the cache key, it takes the URL into account, so even though the `list()` method will be used in both instances, the URL (and therefore the cache key), differs so we don't have to worry about any kind of clashes.

Query Parameter Filtering

Query Parameter Filtering

We won’t cover filtering on query parameters in this module, as we’ll look at them more extensively in the next module, with the third-party package *django-filter*.

However, to give a one sentence tutorial: you can access query parameters in `get_queryset()` using the `self.request.query_params` dictionary.

That brings us to the end of this module. In module three, we’re going to look at some third-party libraries, starting with *django-filter*.

Pushing to GitHub

Pushing to GitHub

Before continuing, you must push your work to GitHub. In the terminal:

- Commit your changes:

```
git add .
git commit -m "Finish filtering"
```

- Push to GitHub:

```
git push
```