Learning Objectives

- **Define throttling**
- **Differentiate between burst and sustained rates**
- **Identify the periods used for throttling**
- **Add different throttling for anonymous and authenticated users**
- **Create different burst and sustained rates for anonymous and authenticated users**
- **Define how DRF defines anonymous users**
- **Scope throttling to different views and viewsets**

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.

Throttling Overview

What is Throttling?

No matter how we choose to host our Django application, we're always going to be resource-limited at some point. Even if you design amazing, automatically-scaling infrastructure in the world's best cloud platform, you probably only have a finite amount of money.

To prevent resource usage from ballooning out of control, we can implement throttling on our API. That is, limiting the amount of requests that clients can make.

Often, throttling is determined by two different rates, a *burst* rate and a *sustained* rate. The burst rate applies over a short period, and the sustained rate over a longer period.

For example, a client might have a burst rate of 60 requests per minute, and 1,000 requests per day. Let's say the client starts making two requests per second. They would exhaust their burst quota after 30 seconds, and then have to wait 30 seconds before they could start making requests again.

If they continued on this pattern of 60 requests a minute for a while, they would exhaust their sustained quota after about 17 minutes (1,000 requests / 60 requests per minute ≈ 16.67 minutes), and would not be able to make any more requests until 24 hours have passed.

We usually also want to throttle differently for anonymous users and logged-in users. Generally, logged-in users will have higher rates and can be billed accordingly.

Let's see how Django Rest Framework implements these methods.

Django Rest Framework Throttling Overview

DRF provides extensible methods of setting up throttling. It stores the rate limit information in the Django default cache, so we need to make sure this is set up in `settings.py` (this is already done in our Blango project).

DRF throttles are implemented using throttle classes. Some base classes are provided to take care of common use cases, and can be used as base classes for custom throttling. DRF provides the `rest_framework.throttling.AnonRateThrottle` class to define limits for anonymous users, and `rest_framework.throttling.UserRateThrottle` for logged-in users.

The throttling classes to use, as well as their limits, are defined using the `REST_FRAMEWORK` dictionary in the Django settings file. The `DEFAULT_THROTTLE_CLASSES` define a list of throttle classes to apply, and the `DEFAULT_THROTTLE_RATES` define their limits.

Here's how we would set up `AnonRateThrottle` and `UserRateThrottle` with the rates of 500 requests per day and 2,000 requests per day, respectively:

```
REST_FRAMEWORK = {
    # existing settings omitted
    "DEFAULT_THROTTLE_CLASSES": [
        "rest_framework.throttling.AnonRateThrottle",
        "rest_framework.throttling.UserRateThrottle"
    ],
    "DEFAULT_THROTTLE_RATES": {
        "anon": "500/day",
        "user": "2000/day"
    }
}
```

Notice that the rates are set as a string, in the form of `<n>/<period>`. The period can be `second`, `minute`, `hour` or `day`.

How do we know what keys are valid for the `DEFAULT_THROTTLE_RATES`? These are actually the scopes of the throttle classes, which are defined by the `scope` attribute on the class. For example, the `AnonRateThrottle` has a `scope` defined like this:

```
class AnonRateThrottle(SimpleRateThrottle):
    scope = 'anon'
```

The scopes are arbitrary, you can come up with your own ones when writing throttle classes. Let's look at that now, as we need custom classes to implement burst/sustained rate limiting.

Burst and Sustained Rates

Burst and Sustained Rates

To configure throttling for burst and sustained rates, we need to write custom classes that define scopes for each particular limit. If we want burst and sustained rates for both signed-in and anonymous users, that means four classes with four scopes. For example:

```
from rest_framework.throttling import AnonRateThrottle,
        UserRateThrottle


class AnonSustainedThrottle(AnonRateThrottle):
    scope = "anon_sustained"


class AnonBurstThrottle(AnonRateThrottle):
    scope = "anon_burst"


class UserSustainedThrottle(UserRateThrottle):
    scope = "user_sustained"


class UserBurstThrottle(UserRateThrottle):
    scope = "user_burst"
```

▼ **Throttling file**
In our case, these classes will all be in the `blog/api/throttling.py` file.

Then the settings to use these classes and scopes are like this:

```
REST_FRAMEWORK = {
    # existing settings omitted
    "DEFAULT_THROTTLE_CLASSES": [
        "blog.api.throttling.AnonSustainedThrottle",
        "blog.api.throttling.AnonBurstThrottle",
        "blog.api.throttling.UserSustainedThrottle",
        "blog.api.throttling.UserBurstThrottle",
    ],
    "DEFAULT_THROTTLE_RATES": {
        "anon_sustained": "500/day",
        "anon_burst": "10/minute",
        "user_sustained": "5000/day",
        "user_burst": "100/minute",
    },
}
```

Here, anonymous users are allowed 500 requests per day (sustained) or 10 per minute (burst). Logged-in users can do 5,000 requests per day (sustained) or 100 per minute (burst).

▼ **Anonymous users**
DRF anonymous users are determined by IP address, so multiple clients at the same IP address would be considered the same “user”.

Throttles Per View and Scoped Throttles

Throttles Per View and Scoped Throttles

DRF provides the `ScopedRateThrottle` class which works in conjunction with the `throttle_scope` attribute on a view or viewset. Using this class lets us easily apply specific limits to views.

Here's how we could scope our Blango API views. Let's limit our `PostViewSet` to 50 requests per minute, and `UserDetail` view to 2000 requests per day.

Start by adding the scoped settings to `DEFAULT_THROTTLE_RATES`, we'll call ours `post_api` and `user_api`.

```
    REST_FRAMEWORK = {
        # existing settings omitted
        "DEFAULT_THROTTLE_RATES": {
            # existing settings omitted
            "post_api": "50/minute",
            "user_api": "2000/day"
        },
    }
```

DRF also lets us apply throttle classes directly to views or viewsets, by setting the `throttle_classes` attribute. In this case, we want to retain the `DEFAULT_THROTTLE_CLASSES` we've already set up, and just apply the `ScopedRateThrottle` to the `PostViewSet` and `UserDetail` view.

```
from rest_framework.throttling import ScopedRateThrottle


class PostViewSet(viewsets.ModelViewSet):
    throttle_classes = [ScopedRateThrottle]
    throttle_scope = "post_api"
    # existing attributes/methods omitted


class UserDetail(generics.RetrieveAPIView):
    throttle_classes = [ScopedRateThrottle]
    throttle_scope = "user_api"
    # existing attributes/methods omitted
```

With this configuration, clients can make up to 50 requests per minute to the `Post` API, before being blocked from it. They can continue to make requests to the `User` API (up to 2,000 per day), since it's under a different scope. Requests to these views aren't also limited by the classes in the `DEFAULT_THROTTLE_CLASSES` setting as the `throttle_classes` overrides those.

We won't actually be using `ScopedRateThrottle`, but will use the others, so let's go ahead and get them set up now.

Try It Out

Try It Out

Let's add some throttling to the Blango API. We'll set up sustained and burst limits for both authorized and anonymous users.

Start by creating a file called `throttling.py` in the `blog/api` directory. We'll be writing classes that inherit from both `AnonRateThrottle` and `UserRateThrottle`, so start by importing these classes:

```
from rest_framework.throttling import AnonRateThrottle,
        UserRateThrottle
```

Then, implement the throttle classes for all four scenarios:

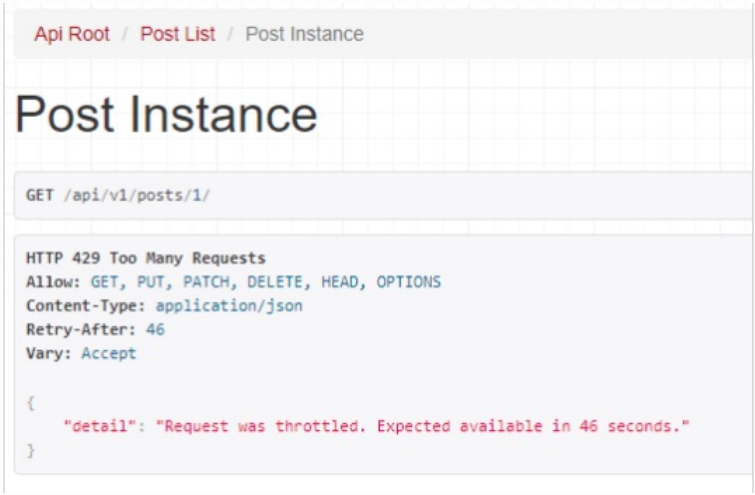
```
class AnonSustainedThrottle(AnonRateThrottle):
    scope = "anon_sustained"


class AnonBurstThrottle(AnonRateThrottle):
    scope = "anon_burst"


class UserSustainedThrottle(UserRateThrottle):
    scope = "user_sustained"


class UserBurstThrottle(UserRateThrottle):
    scope = "user_burst"
```

Now save the file and head over to `settings.py`. Add the `DEFAULT_THROTTLE_CLASSES` and `DEFAULT_THROTTLE_RATES` keys to the `REST_FRAMEWORK` dictionary:

Open settings.py

```
    "DEFAULT_THROTTLE_CLASSES": [
        "blog.api.throttling.AnonSustainedThrottle",
        "blog.api.throttling.AnonBurstThrottle",
        "blog.api.throttling.UserSustainedThrottle",
        "blog.api.throttling.UserBurstThrottle",
    ],
    "DEFAULT_THROTTLE_RATES": {
        "anon_sustained": "500/day",
        "anon_burst": "10/minute",
        "user_sustained": "5000/day",
        "user_burst": "100/minute",
    },
```

The quickest way to check if throttling is working is with the DRF GUI, as an anonymous user. Load up any DRF URL and refresh the page repeatedly. On the 10th attempt, you'll get a response like:



throttled response

View Blog

The response has a `429 Too Many Requests` status, and sets the header `Retry-After`, which is a number of seconds after which you can expect the response to be successful. This is only an approximation, but it's good etiquette to not try the request again until after this amount of time has elapsed. Of course, there's nothing stopping you from attempting the request again if you really want!

Custom Throttling

What if you want to write your own throttling rules? You can write your own throttling classes and override the `allow_request()` method: return `True` or `False` to allow or deny the request. Here's an example class that the

DRF documentation provides, which will randomly deny one in every ten requests:

```
import random

class RandomRateThrottle(throttling.BaseThrottle):
    def allow_request(self, request, view):
        return random.randint(1, 10) != 1
```

You can read the offical Custom throttles documentation to learn more.

That's all for throttling in Django Rest Framework. Next we'll look at adding filtering to our API.

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 throttling"
```

- Push to GitHub:

```
git push
```