Neil Osborne – Django API

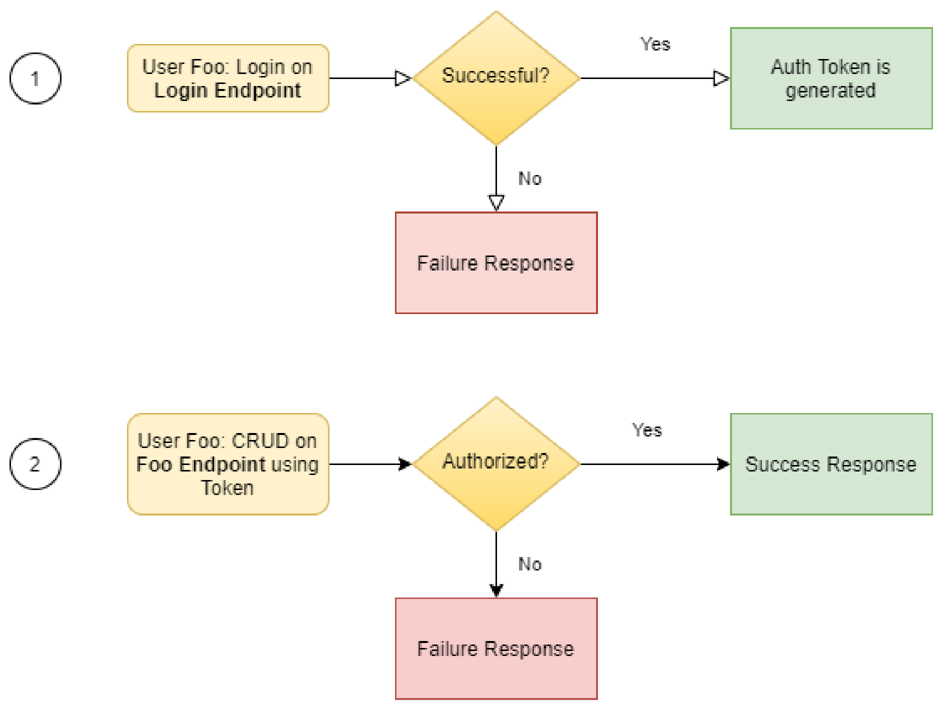
Project Requirements (as received)

1. REST API that exposes CRUD methods on classes Foo and FooBar
2. All methods on API require user to be securely authenticated and authorized (HTTPS)
3. Provide a login API through which a user can attempt to login.
Authentication errors must be handled gracefully.
4. Registered users: foo and foobar.
User foo has permission to invoke methods on Foo
User foobar has permission to invoke methods on FooBar
Permission exceptions are handled gracefully.

Deliverables:

- The project will consist of 3 apps: Users, Data and API.
- The Users will be based on Django's default user template.
- Users: Foo (foo@test.com) - FooBar (foobar@test.com)
- Data Tables (Classes) : Foo, FooBar
- EndPoints
    a. Login
    b. ModelViewSets (CRUD)
- Database: SQLite

Project Flowchart



Packages

Python version==3.9.0
asgiref==3.3.1
Django==3.1.6
djangorestframework==3.12.2
pytz==2021.1
sqlparse==0.4.1

Project Preparation

Defining a Virtual Environment

Start by creating a Virtual Environment for the project. Inside a New Folder, open Command Prompt and type: **python –m venv venv**

That will create a virtual environment (VE) called ‘venv’.

Activating the VE – type: **venv\scripts\activate**
You will see that the command line will display (venv) before the activate path.

Installing Packages

(venv) (project path)> **pip install Django djangorestframework**

Initiating the Django Project

(venv) (project path)> **django-admin startproject Project**

Creating the Project Apps

(venv) (project path)\Project> **python manage.py startapp Data**
(venv) (project path)\Project> **python manage.py startapp API**

Creating the Initial Migrations

(venv) (project path)\Project> **python manage.py makemigrations**
(venv) (project path)\Project> **python manage.py migrate**

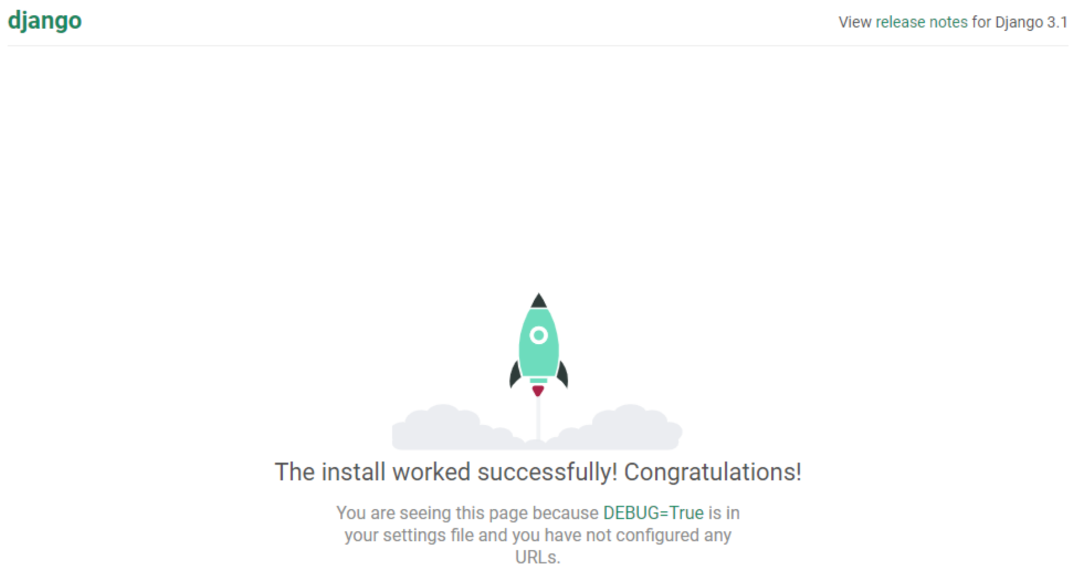
Creating the Super User (Project Admin)

(venv) (project path)\Project> **python manage.py createsuperuser**
username: neil
email: neil@test.com
password: Password321

Firing up the server

(venv) (project path)\Project> **python manage.py runserver**

The development server is now running, and can be accessed on the browser:
http://127.0.0.1:8000/


django

View release notes for Django 3.1

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

The Project Admin page will be accessible on http://127.0.0.1/admin/

Settings File

In the Project's settings file, append the following in INSTALLED_APPS:
'rest_framework',
'rest_framework.authtoken',
'Users',
'Data',
'API',

URLS

Create a ‘urls.py’ file in the API folder. Inside, you will need to create all the endpoints needed.

Serializers

Create a ‘serializers.py’ file in the API folder. A serializer is a feature from the Django REST Framework that allows us to convert data inputs into Python objects and vice versa.

Permissions

User Foo is a member of Group FooGroup, which has access to CRUD table (model) Foo.

User FooBar is a member of group FooBarGroup, which has access to CRUD table FooBar.

Create a ‘permissions.py’ file in the API folder. These will hold the specific permissions to each endpoint.

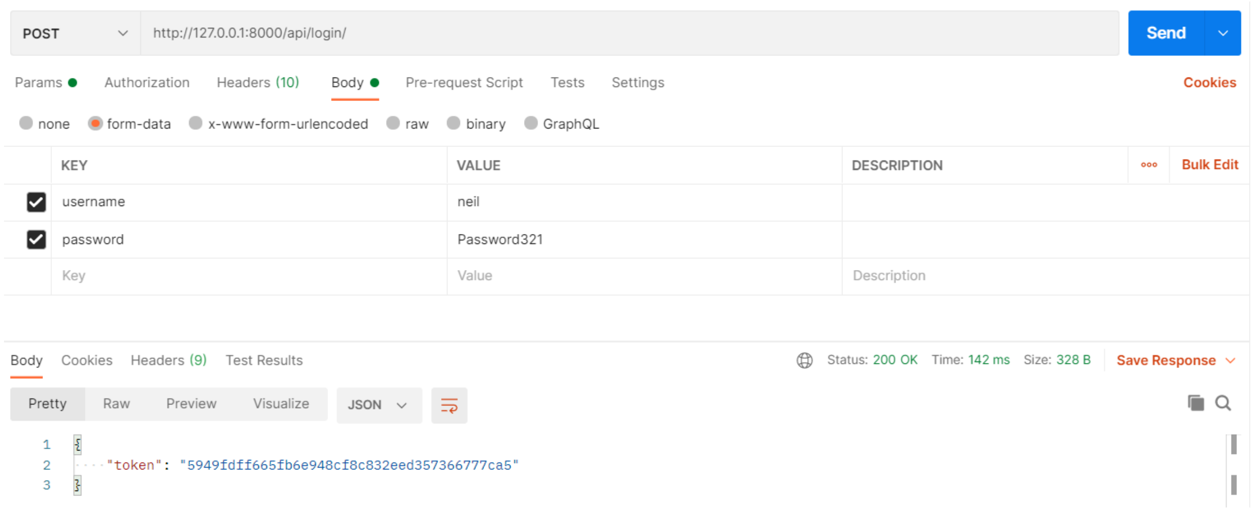
Exception Handling

Since the requirement clearly states that Authentication/Permissions have to be handled gracefully, an Exception Handler for these errors has been created, containing subsets of errors 400, 401 and 403.
It is found in the utils.py module, and referenced in the project’s settings.py

Accessing the API – Postman

Using Postman, we can make calls to the API endpoints.

Below is the Login API (server_address/api/login)


POST http://127.0.0.1:8000/api/login/

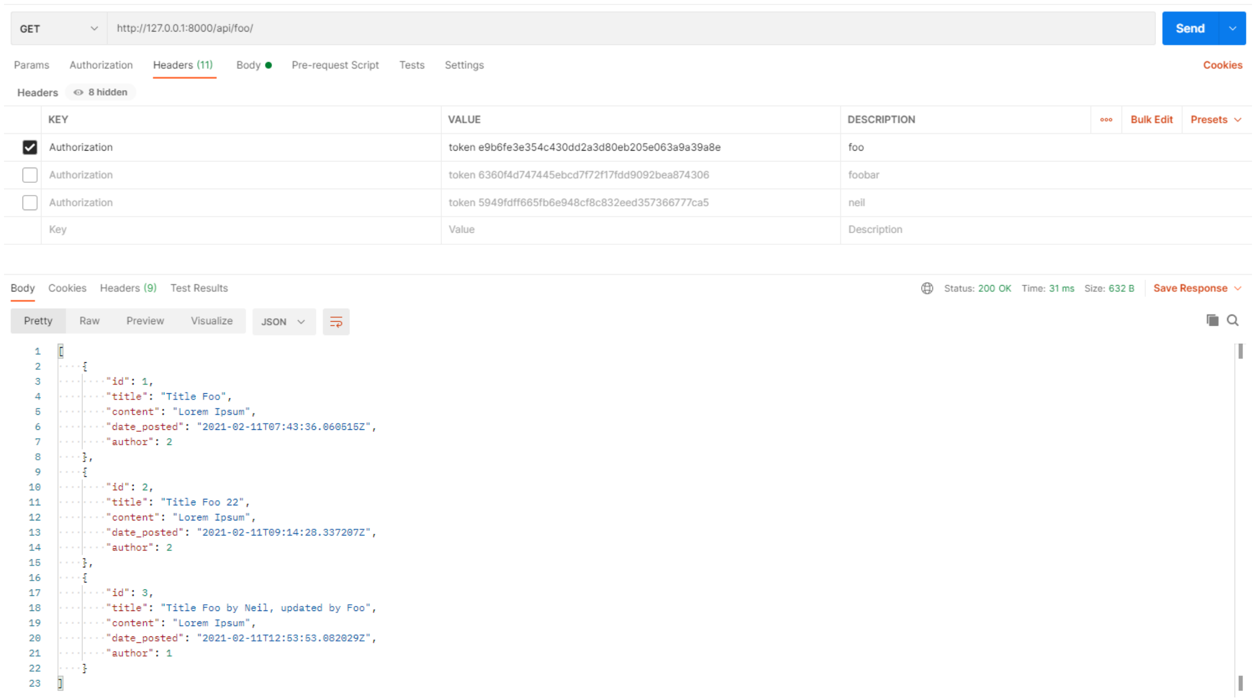
Body: form-data

KEY	VALUE	DESCRIPTION
username	neil	
password	Password321	

Status: 200 OK Time: 142 ms Size: 328 B

```
{
    "token": "5949fdff665fb6e948cf8c832eed357366777ca5"
}
```

Below is a **successful** GET call to the Foo Endpoint, using the user Foo's Token



GET http://127.0.0.1:8000/api/foo/

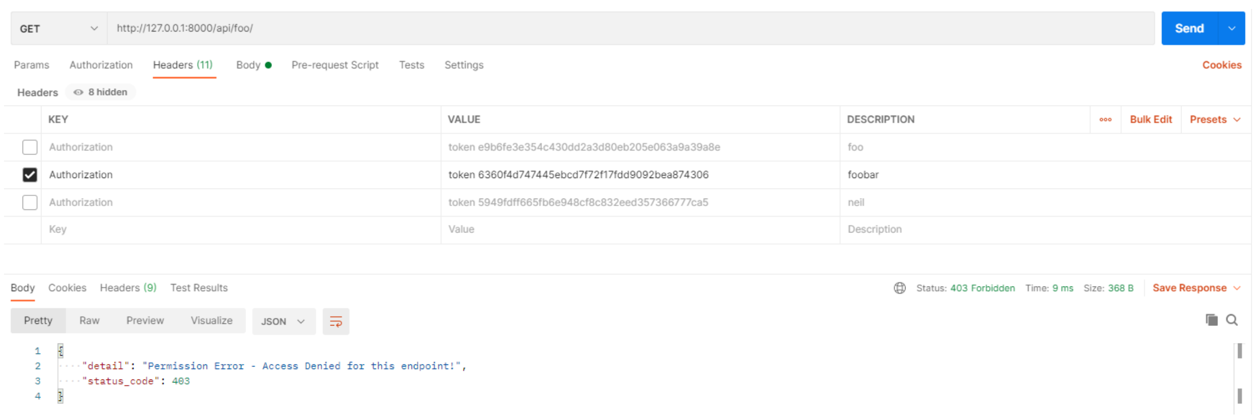
Headers (11) — 8 hidden

	KEY	VALUE	DESCRIPTION
✓	Authorization	token e9b6fe3e354c430dd2a3d80eb205e063a9a39a8e	foo
	Authorization	token 6360f4d747445ebcd7f72f17fdd9092bea874306	foobar
	Authorization	token 5949fdff665fb6e948cf8c832eed357366777ca5	neil

Status: 200 OK Time: 31 ms Size: 632 B

```
[
    {
        "id": 1,
        "title": "Title Foo",
        "content": "Lorem Ipsum",
        "date_posted": "2021-02-11T07:43:36.060515Z",
        "author": 2
    },
    {
        "id": 2,
        "title": "Title Foo 22",
        "content": "Lorem Ipsum",
        "date_posted": "2021-02-11T09:14:28.337207Z",
        "author": 2
    },
    {
        "id": 3,
        "title": "Title Foo by Neil, updated by Foo",
        "content": "Lorem Ipsum",
        "date_posted": "2021-02-11T12:53:53.082029Z",
        "author": 1
    }
]
```

Below is a **failed** GET call to the Foo Endpoint, using the user FooBar's Token:



GET http://127.0.0.1:8000/api/foo/

Headers (11) — 8 hidden

	KEY	VALUE	DESCRIPTION
	Authorization	token e9b6fe3e354c430dd2a3d80eb205e063a9a39a8e	foo
✓	Authorization	token 6360f4d747445ebcd7f72f17fdd9092bea874306	foobar
	Authorization	token 5949fdff665fb6e948cf8c832eed357366777ca5	neil

Status: 403 Forbidden Time: 9 ms Size: 368 B

```
{
    "detail": "Permission Error - Access Denied for this endpoint!",
    "status_code": 403
}
```

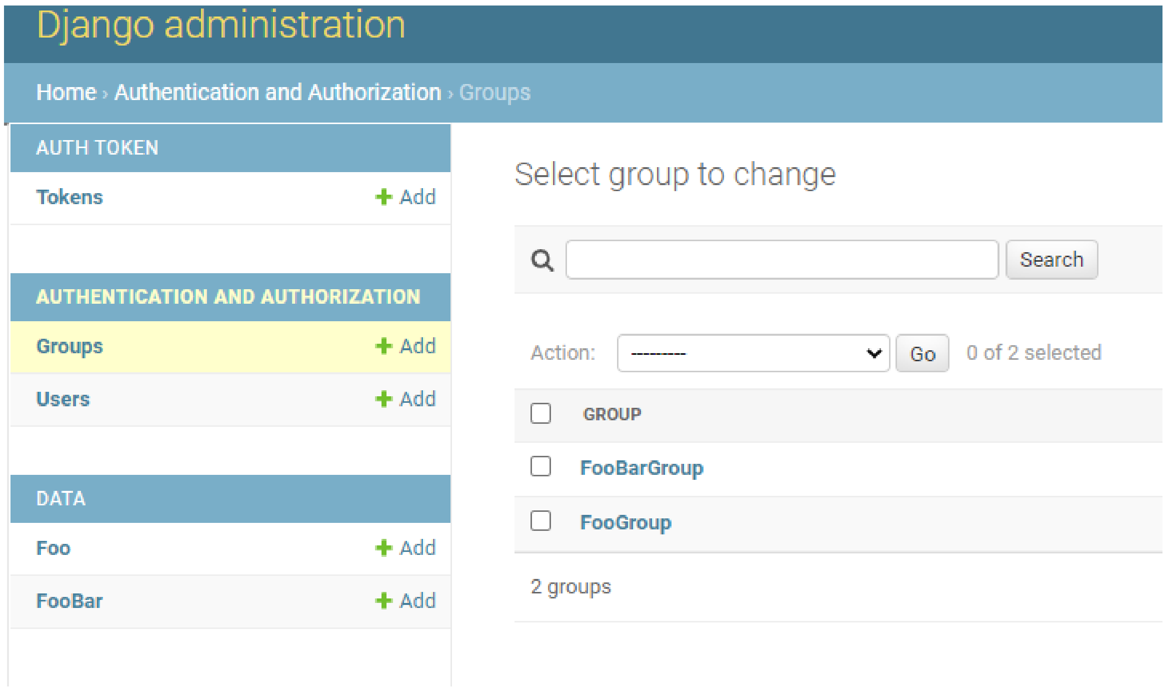
The user 'neil', being a superuser, has access to all endpoints.

Q & A

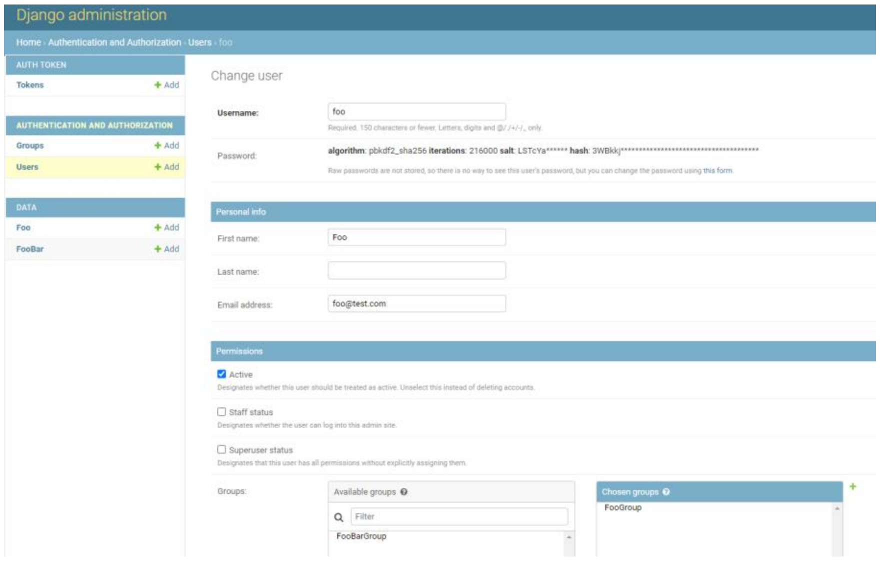
Q: It appears that you're not on a Unix platform so I had issues with CR in the file that had me stumped for a while - until I converted manage.py into the correct line endings for Linux.

A: That is really strange; I have always used Windows for development and Ubuntu/Debian for production, and never had this problem. There are definitely some minor differences between Windows and Linux that I have come across with Django, but the initial setup of the server is very Python-dependent, not really OS-dependent.

1. It is not clear where FooGroup and FooBarGroup are DEFINED (no models), so I don't actually understand how the permissions are being enforced - since the models don't seem to appear in the database.
A: FooGroup and FooBarGroup are actual groups defined in the Auth model. Permission-based views should rely on Groups, as it's the most efficient way of providing access to certain endpoints/models. Having said that, I had included the SQLite db file with the project so that everything is ready for you. Nonetheless, here is a screenshot of the current setup:



Screenshot of the groups.



Django administration

Home › Authentication and Authorization › Users › foo

AUTH TOKEN

Tokens + Add

AUTHENTICATION AND AUTHORIZATION

Groups + Add

Users + Add

DATA

Foo + Add

FooBar + Add

Change user

Username:	foo Required. 150 characters or fewer. Letters, digits and @/./+/-/_ only.
Password:	**algorithm**: pbkdf2_sha256 **iterations**: 216000 **salt**: LSTcYa****** **hash**: 3WBkkj************************************** Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name:	Foo
Last name:	
Email address:	foo@test.com

Permissions

☑ Active
Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

☐ Staff status
Designates whether the user can log into this admin site.

☐ Superuser status
Designates that this user has all permissions without explicitly assigning them.

Groups:

Available groups: FooBarGroup

Chosen groups: FooGroup

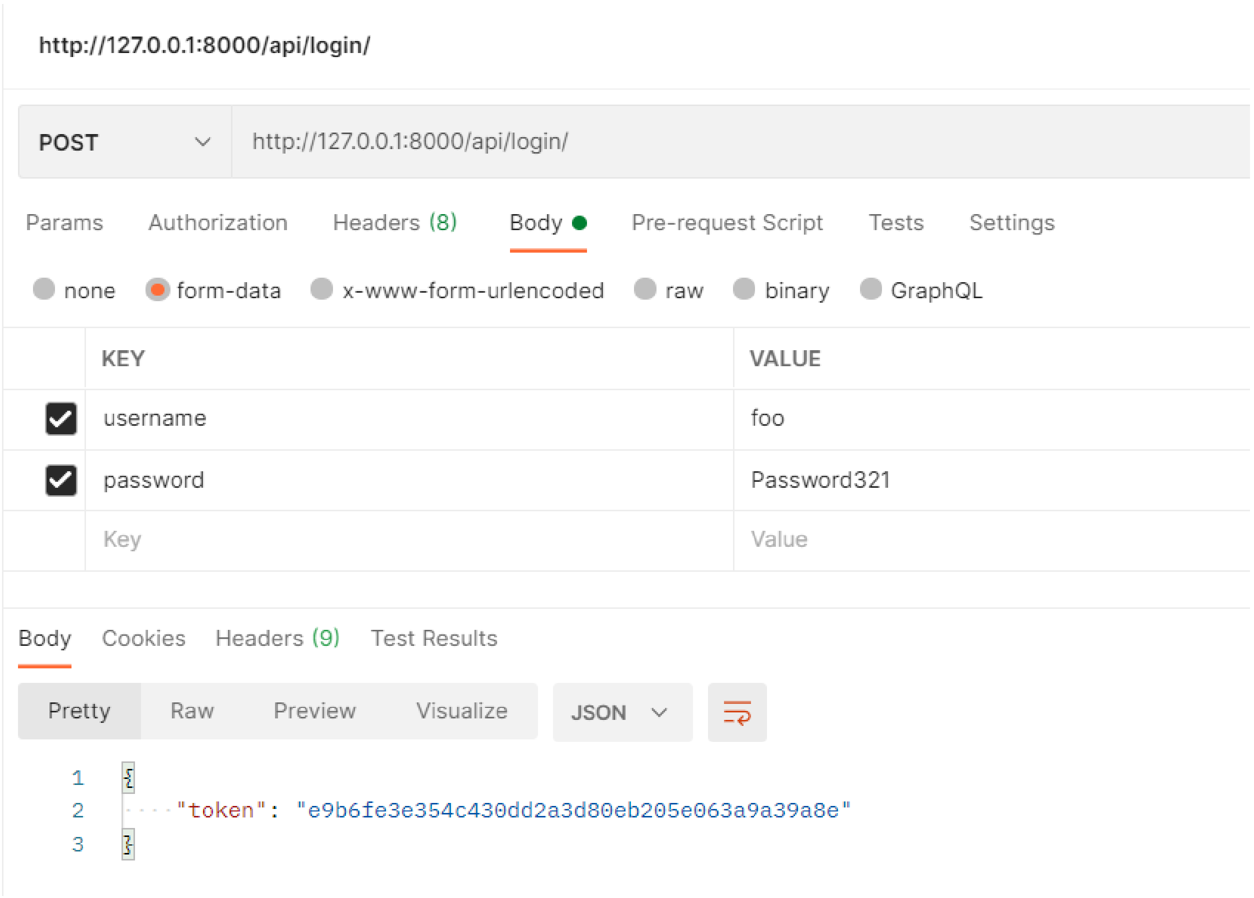
Screenshot of the user 'Foo'. As you can see, Foo is a member of the FooGroup permissions group.

By the same token, FooBar is a member of the FooBar Group.

2. I am guessing that I have to create users Foo and FooBar, in order to test the permission functionality - but it is not clear how to assign the permissions to Foo and FooBar. Do I create the groups FooGroup and FooBarGroup via the admin manager view and assign the users there?
A: The users and the groups are already in the DB. But should you wish to re-do the setup, you will need to create 2 users and 2 groups, assign Foo to FooGroup, and FooBar to FooBarGroup from the Django Administration UI.

3. Assuming that I have created the users Foo and FooBar and assigned permissions as described above - how is the authentication token passed in the subsequent calls to the permission restricted endpoints? This may seem an absurd question - BUT, I created the superuser neil and after successfully logging on, when I called the permission restricted endpoint foo(), I received an authorization error (please see attached screenshots).

A: With an auth-based API, you will need to do 1 call to the Login Endpoint initially, to get the Auth Token, and then use the token in the request's header every time you attempt to hit any endpoint.



http://127.0.0.1:8000/api/login/

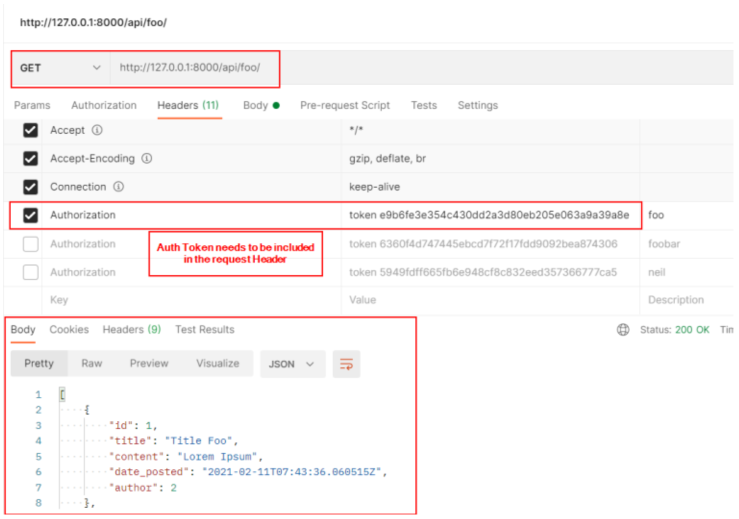
POST http://127.0.0.1:8000/api/login/

	KEY	VALUE
✓	username	foo
✓	password	Password321
	Key	Value

```
{
    "token": "e9b6fe3e354c430dd2a3d80eb205e063a9a39a8e"
}
```

Screenshot of the Login Endpoint. It’s a POST request to the API with 2 fields in the body of the request. The request Headers are the default POST request to a JSON API, automatically generated for you by Postman.

And here is the request to the Foo endpoint:



http://127.0.0.1:8000/api/foo/

GET http://127.0.0.1:8000/api/foo/

✓	Accept	*/*	
✓	Accept-Encoding	gzip, deflate, br	
✓	Connection	keep-alive	
✓	Authorization	token e9b6fe3e354c430dd2a3d80eb205e063a9a39a8e	foo
	Authorization	token 6360f4d747445ebcd7f72f17fdd9092bea874306	foobar
	Authorization	token 5949fdff665fb6e948cf8c832eed357366777ca5	neil
	Key	Value	Description

```
[
    {
        "id": 1,
        "title": "Title Foo",
        "content": "Lorem Ipsum",
        "date_posted": "2021-02-11T07:43:36.060515Z",
        "author": 2
    },
```

As you can see from the above screenshot, the token has to be included in the request Header. The Key is ‘Authorization’ and the value is ‘token [token]’. This format is not flexible unfortunately, so it has to be exactly this way.

Q: Last, but not least, as I mentioned, I want to run this via HTTPS. I have thus installed django-sslserver and modified settings.py accordingly. However, when I start up the SSL session successfully by running ./manage.py runsslserver, when I attempt to use postman, I get the error message: "Could not get response" (please see attached image).

A: it's quite tricky to run a local copy of a Dev server of Django on HTTPS. The best practice would be to run it locally on HTTP, and when hosted on a PROD server, you would use Gunicorn as a gateway (WSGI server) for Django, and host it behind a reverse-proxy (I use NGINX, but Apache can also be used, or any other server for the same) where a SSL Certificate will be defined, some configuration done on the server to encrypt the traffic and allow HTTPS only.

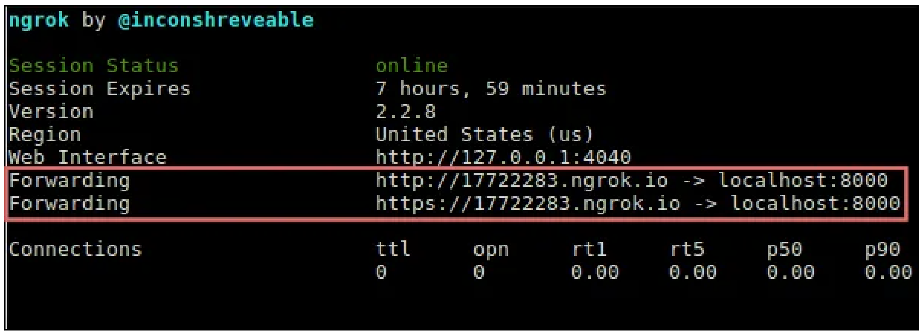
However, if you would like to test the HTTPS functionality locally, I came across a tool called NGROK which forwards HTTP and HTTPS traffic from the Internet to your server. Quite easy to setup and might give you what you need.

Setting Up Ngrok

To configure ngrok follow these steps:

1. Download ngrok for your OS at https://ngrok.com/
2. Unzip the downloaded file and you will get a binary file named `ngrok`.
3. Copy the `ngrok` file into the same directory as `manage.py`.
4. Start the HTTP tunnel by running `./ngrok http 8000`.

The output of the above command should look like this:



```
ngrok by @inconshreveable

Session Status                online
Session Expires               7 hours, 59 minutes
Version                       2.2.8
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://17722283.ngrok.io -> localhost:8000
Forwarding                    https://17722283.ngrok.io -> localhost:8000

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Our application is now accessible over the Internet using the following two URL:

- `http://17722283.ngrok.io/`
- `https://17722283.ngrok.io/`

There is no need to test HTTPS using Django-ssl. You can hit the endpoint using a public HTTPS request provided by NGROK.
Note: hitting an endpoint on HTTP or HTTPS is exactly the same from the user/request perspective. If you can get a valid response on HTTP, there is absolutely no reason why HTTPS would not work.