In the name of ALLAH

Short Document of Internship tasks

Set up a Django Rest frameWork Project from the base DeviceManagement Project

~$ Contents :

~$ In this Doc we mainly follow this source steps and beside that explain more about errors , faults you may face and a little more about Django project structure and files …

Pouria Dadkhah

Pouria.Dadkhah@gmail.com

1: https://www.django-rest-framework.org/

For building an API we need a Database , a provider for main content and options in our page and a web server to serve it up to other users; as you can see the most important part is making the API structure and we do this by Django Rest frame work ; and for other parts we use postgresql as data base provider and nginx to serve our API up.

First we got familiar with Django rest frame work , it's concepts , aim and structure in [Django Rest frame work] website[1] .

As you can follow its tutorial and other parts to learn more about Django , we don't talk about Django coding and modules deeply and concentrate on project path more…

1. First of all let's install requirements ;

```
sudo apt update

sudo apt install python3-pip python3-dev libpq-dev postgresql postgresql-contrib nginx curl

sudo -H pip3 install virtualenv
```

after these , let's make our project directory and set up virtual environment and install and work with django…

```
mkdir ~/myprojectdir

cd ~/myprojectdir

virtualenv myprojectenv

source myprojectenv/bin/activate
```

after activating your venv the terminal look like this :

```
(env) pouria@ubuntu:~$
```

Now it's ready to install our python packages which needed :

```
pip install django  djangorestframework markdown  django-filter pygments

\gunicorn psycopg2-binar
```

1: https://www.django-rest-framework.org/

~$ *** WARNING***

As you saw up we first came to our specific project directory and then , build our private V environment ; this cause that if you want to start another project somewhere else you should install it's own packages and has nothing to this project ; because of this you may think what if build venv in /home/ directory so all of my projects that have same packages , don't need reinstallation ! but as we see future we need to pass our specific working directory and execfiles(in venv files and installed packeges) to other services (like gunicorn , uwgsi , nginx , …) and if you do this and have a whole venv for all of your projects , the service setting can not recognize your project file and causes errors. So for ignoring this awful error to happen be careful now ☺

****

2. Now you are ready to creat your Django Rest frame work project and build your API …

In this Doc we make a project and an app in it . for details of project structure and coding follow [Django rest frame work](#) .

```
django-admin startproject tutorial

cd tutorial

python manage.py startapp snippets
```

now your tutorial project has a snippets app in it . first we should add this app and rest frame work to your project INSTULLED APPS which is in /project/settings.py

So open this .py file with any editor you like (in terminal you can use gedit or nano to do that or you can use high performance code editors like pycharm or vs code . for example in terminal :

```
(env) pouria@ubuntu:~/tutorial/tutorial$gedit settings.py
```

And then add these to your installed apps:

```
INSTALLED_APPS = [

    ...

    'rest_framework',

    'snippet' ,

]
```

~$ *** NOTICE *****

For just serve up our app locally on main Django port (127.0.0.1:8000/) it's enough to just add `'snippets.apps.SnippetsConfig'` file to your installed app and it will recognize other modules like `'snippet.model'` BUT when we pass it to other sockets like `gunicorn` you will face error and that can't understand your app so you need to add whole 'snippet' app.

****

Now the default Django project will use it's own Database to save your API data which is not appropriate for real projects that want to serve up to many users:

Default database setting in settings.py :

```
DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.sqlite3',

        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),

    }

}
```

And you can see `db.sqlite3` file in your project directory

3. So we should choose a database for our project . we use `postgresql` Data base .

Before going further and involving with Django code and debugging we reset the Data Base and then come back to Django …

Setting a postgresql Database :

```
~$sudo -u postgres psql

postgres=# CREATE DATABASE myproject;

postgres=# CREATE USER myprojectuser WITH PASSWORD 'password';

postgres=# ALTER ROLE myprojectuser SET client_encoding TO 'utf8';

postgres=# ALTER ROLE myprojectuser SET default_transaction_isolation TO 'read committed';

postgres=# ALTER ROLE myprojectuser SET timezone TO 'UTC';

postgres=# GRANT ALL PRIVILEGES ON DATABASE myproject TO myprojectuser;

\q
```

~$ ** Point : Be careful about all “ ; “ at the end of your postgres instructions ….

Now go to settings.py of your project and change the database :

```
DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql_psycopg2',

        'NAME': 'myproject',

        'USER': 'myprojectuser',

        'PASSWORD': 'password',

        'HOST': 'localhost',

        'PORT': '',

    }

}
```

And at the end check and add these STATICs to settings:

```
STATIC_URL = '/static/'

STATIC_ROOT = os.path.join(BASE_DIR, 'static/')
```

(If you just create project the first static should've already exist …)

Now it's Done. Let's update and collect primary database of our project and app by these instructions :

```
Python manage.py makemigrations snippet

Python manage.py migrate
```

~$ *** NOTICE:

Lets talk about more about the use of these orders;

1. When we make changes to our database project , we should collect our files in the new database mode.

2. when we create a model (usually for our apps we build a model.py file which consists of general settings and structure of our API page like title , language , .. and general methods like save , ..) we should build the model in date in our database so we run tese.

So we just run them to check that our new Database doen't have any problem , at the next step we will create a model.py for our snippet app and update database again !

****

~$ *** Point: it's good to notice to another part of setting in the beginning ;

```
ALLOWED_HOSTS = []
```

Shows the IP's and domains which can access to you API (through local server up or web servers or … ; which means any IP addresses , server names or .. that you will use later in nginx web server for example , should be in this part. If you don't want to limit servers here you can use just "*" in this part that include all available addresses !

```
ALLOWED_HOSTS = ['*']
```

Ok . now it's time to complete Rest API by setting it's Views , Urls , Model , Serializers ,… .

As we said before it will take more time and get us far from the main process of building project ; so the details of such files and codes are explained in Django_Rest_framework _Doc . you can see the final codes and files of project in this Repository.

At the end your project directory should be something like this :

```
myproject/
    manage.py
    myproject/
        __init__.py
        urls.py
        wsgi.py
        settings.py
    my_app/
        __init__.py
        models.py
        views.py
        urls.py
        serializers.py
        permissions.py
        tests/
```

~$ ***CHECK

Check your API working locally on Django port by running the server:

```
$ python manage.py runserver 0.0.0.0:8000

or just ;

$ python manage.py runserver
```

Notice in the first one it allow any IP address that include your own IP address to run our API through browser on port 8000(however because it's local you can't run it through any address and it should be one of your IP's

But in 2nd model it is accessible just through Django default IP 127.0.0.1:8000 . further in nginx we will use port 80 which is default so there will be no need to type port in browser to reach the API .

4. Django Channels ☺

In this part we want to add a very useful feature to our API which is real time connection . today web apps have been updated a lot and one of the most useful features is that you have a real time connection in a website to another site or app ; for example the number of site users changes in the website without any refreshing the page ; or another example is chat room !

You enter a chat room in your browser , and your friend does too and you start chatting and see each others messages without need to refresh the chat room page ! (just imagine how boring was that in this way !!)

Django also has prepared a platform called channels to support real time connections through websocket connections (instead of simple HTTP connections)

For more information about websockets and channels you can use these sources .

So first follow installation to install channel and docker to support and build channel layers on that .

Install Channel first :

```
(env)~$ python -m pip install -U channels
```

~$ Notice:

You now need to add some related features to your project's parts;

- First add it to your installed apps :

```
INSTALLED_APPS = (
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.sites',
    ...
    'channels',
)
```

- Then, make a default routing in `myproject/routing.py`:

```
from channels.routing import ProtocolTypeRouter


application = ProtocolTypeRouter({
    # Empty for now (http->django views is added by default)
})
```

- And finally, set your `ASGI_APPLICATION` setting to point to that routing object as your root application:

```
ASGI_APPLICATION = "myproject.routing.application"
```

And finally install channel layers ;

~$ Notice :

If you don't have docker installed , follow [docker website] installation !

****

We will use a channel layer that uses Redis as its backing store. To start a Redis server on port 6379, run the following command:

```
$ docker run -p 6379:6379 -d redis:5
```

And install channels_redis so that Channels knows how to interface with Redis. Run the following command:

```
$ python3 -m pip install channels_redis
```

At the end add channel layer default settings in your `settings.py :`

```
# mysite/settings.py

# Channels

ASGI_APPLICATION = 'mysite.routing.application'

CHANNEL_LAYERS = {

  'default': {

    'BACKEND': 'channels_redis.core.RedisChannelLayer',

    'CONFIG': {

      "hosts": [('127.0.0.1', 6379)],

    },

  },

}
```

So now we are ready to write our consumers which are actually act like views in http requests for websockets ; they get the request and handle it by our instructions and prepare a proper response and send back to websocket and channel . so we need to define some routings too , to link the websocket to it's consumer ; routings acts like urls !

As said before for more information about sockets and consumers and their concepts check channels doc .

After add routings and consumers , your app (not project !) should be sth like this :

```
Your app/
    __init__.py
    consumers.py              # new
    routing.py                # new
    serializers.py
    tests.py
    urls.py
    views.py
```

now you can run your api locally and test your channel too ☺

~$ Notice :

If you are just developing a backend now and UI is not ready yet (like me !)

For testing channel you should define a new websocket in browser page manually because this part is for javascript code to make a socket connection and use that

So the first way is : in your browser consol (press F12) write a java script code to do that !

Or the better way is using a test app ! until a year ago "postman" supported websocket connections too but now it just can be used for http requests but there is chrome extention that test ws connection for us and you can add it from here ☺

Gunicorn ☺

~$ *** LEARN MORE !

The Gunicorn "Green Unicorn" is a Python Web Server Gateway Interface (WSGI) HTTP server. It is a pre-fork worker model, ported from Ruby's Unicorn project. The Gunicorn server is broadly compatible with a number of web frameworks, simply implemented, light on server resources and fairly fast. Actually these web server aren't complete web server and we can't run them through any machine with arbitrary OS but in fact , WSGI's are interface and connectors for web servers like apache and nginx ; because nginx can't connect directly to Django and it just recognize the wsgi applications ; so there so more ways to connect our Django app to nginx (like uwsgi (which is a little old way)) but we chose gunicorn because it's easy to work and understand .

****

First let's test gunicorn by running the server through it ; it looks the previous part but instead of using manage.py pythin package we use gunicorn :

```
gunicorn --bind 0.0.0.0:8000 myproject.wsgi
```

~$ NOTICE:

pay attention to `myproject.wsgi` if you open your project folder you won't see such a file and don't worry ! this wsgi file refers to wsgi.py file which is in `/myproject/myproject` directory . but now and any time later that we want to pass the working directory we should come to this folder(which contains `manage.py`) not one step more (`/myproject/myproject`) that contains `settings.py` !

****

Now like before you can see your API through your IP address in browser .

Ok . now that we checked gunicorn run correctly , lets set up gunicon socket to control and link it to our API in Unix and can access with other servers like nginx.

The Gunicorn socket will be created at boot and will listen for connections. When a connection occurs, systemd will automatically start the Gunicorn process to handle the connection.

```
sudo gedit /etc/systemd/system/gunicorn.socket
```

In `gunicorn.socket`:

```
[Unit]

Description=gunicorn socket



[Socket]

ListenStream=/run/gunicorn.sock



[Install]

WantedBy=sockets.target
```

Next,create and open a systemd service file for Gunicorn with sudo privileges in your text editor.

~$ NOTICE: The service filename should match the socket filename with the exception of the extension:

```
sudo nano /etc/systemd/system/gunicorn.service
```

In `gunicorn.service`:

```
[Unit]

Description=gunicorn daemon

Requires=gunicorn.socket

After=network.target



[Service]
```

```
User=sammy

Group=www-data

WorkingDirectory=/home/sammy/myprojectdir

ExecStart=/home/sammy/myprojectdir/myprojectenv/bin/gunicorn \

          --access-logfile - \

          --workers 3 \

          --bind unix:/run/gunicorn.sock \

          myproject.wsgi:application



[Install]

WantedBy=multi-user.target
```

Now start and enable gunicorn socket and check its status working correctly without any errors :

```
sudo systemctl start gunicorn.socket

sudo systemctl enable gunicorn.socket

sudo systemctl start gunicorn.socket
```

If it has no problems the output should be something like this :

```
● gunicorn.socket - gunicorn socket

     Loaded: loaded (/etc/systemd/system/gunicorn.socket; enabled; vendor preset: enabled)

     Active: active (running) since Mon 2020-08-10 22:15:39 PDT; 24h ago

   Triggers: ● gunicorn.service

     Listen: /run/gunicorn.sock (Stream)

     CGroup: /system.slice/gunicorn.socket



Aug 10 22:15:39 ubuntu systemd[1]: Listening on gunicorn socket.
```

And the last check :

```
$ file /run/gunicorn.sock

RESULT SHOULD BE:

$ Output/run/gunicorn.sock: socket
```

It shows that socket exist in /run directory which we passed through service file .

Now it's `service` turns :

At the first we have not run socket and gunicorn status should be inactive :

```
sudo systemctl status gunicorn
```

Output :

```
Output● gunicorn.service - gunicorn daemon

   Loaded: loaded (/etc/systemd/system/gunicorn.service; disabled; vendor preset: enabled)

   Active: inactive (dead)
```

~$ *** NOTICE:

This step is important to check gunicorn has identified your project and specific file such as myproject.wsgi correctly !

If you passed last step (testing gunicorn) and here you face an error,

Actually this type of error :

```
gunicorn[42878]: ModuleNotFoundError: No module named 'tutorial.wsgi'
```

It refers to wrong directory that you have set in `gunicorn.service` file , working directory or `ExecStart` !

One possible problem is what we have mentioned at the beginning about venv . it can't recognize your `wsgi` in your working directory !

Another possible wrong directory is about `ExecStart`. this should refers to gunicorn file in your venv that has been installed earlier. ****

Now if every thing is correct lets enable gunicorn service and check the status again :

```
curl --unix-socket /run/gunicorn.sock localhost

sudo systemctl status gunicorn
```

Output:

```
● gunicorn.service - gunicorn daemon

     Loaded: loaded (/etc/systemd/system/gunicorn.service; enabled; vendor preset: enabled)

     Active: active (running) since Tue 2020-08-11 02:37:03 PDT; 20h ago

TriggeredBy: ● gunicorn.socket

   Main PID: 52119 (gunicorn)

      Tasks: 4 (limit: 4624)

     Memory: 103.2M

     CGroup: /system.slice/gunicorn.service

             ├─52119 /home/pouria/cascade/env/bin/python /home/pouria/cascade/env/bin/gunicorn --access-logfile - --worke
rs 3 --bind unix:/run/gunicorn.sock main.wsgi:application

             ├─52949 /home/pouria/cascade/env/bin/python /home/pouria/cascade/env/bin/gunicorn --access-logfile - --worke
rs 3 --bind unix:/run/gunicorn.sock main.wsgi:application

             ├─58606 /home/pouria/cascade/env/bin/python /home/pouria/cascade/env/bin/gunicorn --access-logfile - --worke
rs 3 --bind unix:/run/gunicorn.sock main.wsgi:application

             └─58795 /home/pouria/cascade/env/bin/python /home/pouria/cascade/env/bin/gunicorn --access-logfile - --worke
rs 3 --bind unix:/run/gunicorn.sock main.wsgi:application

Aug 11 20:42:03 ubuntu gunicorn[58523]: [2020-08-11 20:42:02 -0700] [58523] [INFO] Worker exiting (pid: 58523)

Aug 11 20:42:04 ubuntu gunicorn[58535]: [2020-08-11 20:42:04 -0700] [58535] [INFO] Booting worker with pid: 58535

Aug 11 20:42:34 ubuntu gunicorn[52119]: [2020-08-11 20:42:34 -0700] [52119] [CRITICAL] WORKER TIMEOUT (pid:58535)

Aug 11 20:42:34 ubuntu gunicorn[58535]: [2020-08-11 20:42:34 -0700] [58535] [INFO] Worker exiting (pid: 58535)

Aug 11 20:42:35 ubuntu gunicorn[58606]: [2020-08-11 20:42:35 -0700] [58606] [INFO] Booting worker with pid: 58606

Aug 11 20:45:38 ubuntu gunicorn[52119]: [2020-08-11 20:45:38 -0700] [52119] [CRITICAL] WORKER TIMEOUT (pid:52948)

Aug 11 20:45:40 ubuntu gunicorn[58795]: [2020-08-11 20:45:40 -0700] [58795] [INFO] Booting worker with pid: 58795
```

~$ NOTICE: every time you cange these two files (`gunicorn.socket` , `gunicorn.service`) you should reload the daemon :

```
sudo systemctl daemon-reload
```

and wholly when you change anything beyond like your Django API (and definitely for the first time you run !) restart gunicorn :

```
sudo systemctl restart gunicorn
```

and at the end for more debugging , beside of status which shows you the errors and states you can use the LOGs :

```
sudo journalctl -u gunicorn

sudo journalctl -u gunicorn.socket
```

****

NGINX ☺

At last we want to use nginx web server to load up our API through browsers of any machine which is connected to our network .

To do that we should make a config file in /etc/nginx/sites-available to pass our different addresses and locations and then link and copy this file to /etc/nginx/sites-enabled to make nginx access and understand that. (notice that nginx work directly whith nginx.Conf file in /etc/nginx . but there is a link there to sites-enabled directory for us to creat our specific Config files for our different projects to reject messing up differnet Configs in nginx.conf file

Again we don’t explain about config files and instructions and you can learn more in nginx website.

So lets do it :

```
$ sudo gedit /etc/nginx/sites-available/myproject
```

In myproject :

```
server {

    listen 80;

    server_name server_domain_or_IP;



    location = /favicon.ico { access_log off; log_not_found off; }

    location /static/ {

        root /home/Pouria/myprojectdir;

    }

    location / {

        include proxy_params;

        proxy_pass http://unix:/run/gunicorn.sock;

    }

}
```

And then link it to sites-enabled :

```
sudo ln -s /etc/nginx/sites-available/myproject /etc/nginx/sites-enabled
```

this order make a same file in 2nd location that any changes that you make in one of them it will make to other too ! (except of deleting the file :))

Ok . check the syntax and then run nginx :

```
$ sudo nginx -t

sudo systemctl restart nginx

sudo ufw allow 'Nginx Full'
```

congratulations ! you can now serve up your API through your IP address in any machine that is connected to your network !

http://Your_IP_address

And at the we need nginx to support websocket connections too.

The gunicorn interface doesn’t support ws and wss connections and it is just useful for http requests ; for this purpose we will use “daphne ” as an ws interface ; pay attention that daphne also support https too but we have already stablished gunicorn for them.

Daphne has already installed automatically in your virtual environment and you can check in your venv folder for sure and it’s running steps is almost like gunicorn so we ignore talking about it more and you can follow this source and this one instructions for that .

Thanks for your attention

Pouria Dadkhah