drf-microservice Documentation

Release latest

Jun 15, 2019

Contents

CHAPTER 1

About Drf-microservice

drf-microservice is a ready-to-use API skeleton:

- Cookiescutter-drf-microservice generated it,
- add your endpoints,
- test it,
- package it,
- validate it on stage and QA then
- deploy it on production.

It sounds simple and it is. Take a look at Drf-microservice it's now generated by Cookiescutter-drf-microservice.

Something disturb you in the code? Don't hesitate to open a an issue and contribute.

CHAPTER 2

Releases Notes

- 0.7.1: Remove all .md file, update the doc,
- 0.7.0: Cookiescutter-drf-microservice got it own separate repository
- 0.6.1: Update dependencies
- 0.6.0: total refactoring for add cookiecutter
- 0.5.2: fix depencies security alert
- 0.5.1: fix some document presentation on github and pypi
- 0.5.0: Initial publication version

CHAPTER 3

Bugs and evolution policies

When you will find a bug or propose an evolution create a ticket on:

- Issue Cookiescutter-drf-microservice if it's about the generation process
- Issue Drf-microservice if it's about a functionality in the generated drf process

CHAPTER 4

To setup

- **Now we just jump in the new directory and run tox to ::**
  - be sure that everything as worked fine
  - generate the documentation
  - generate an virtualenv

```
cd drf-microservice
tox
```

- An virtualenv is already ready for you at

```
tox -l
py36-django222
```

- or you can create your

```
python3 -m venv /pass/to/venv
```

- for bash, zsh

```
source .tox/py36-django222/bin/activate
```

- for fish

```
source .tox/py36-django222/bin/activate.fish
```

- for bash, zsh

```
SECRET_KEY=my_secret_key
python setup.py makemigration
python manage.py migrate
python manage.py createsuperuser
```

- for fish

```
env SECRET_KEY=my_secret_key
python setup.py makemigration
python manage.py migrate
python manage.py createsuperuser
```

- then run it

```
SECRET_KEY=my_secret_key
python manage.py runserver
```

- if you have any problem or you want enable the debug mode

```
ENABLE_DEBUG=1
```

CHAPTER 5

API

To see the documentation for the API In development mode, login at

```
curl --request POST \
  --url http://127.0.0.1:8000/api-auth/login/ \
  --header 'content-type: application/json' \
  --data '{
    "username": "admin",
    "password": "admin"
    }'
```

Actually the default mode is “development” (same to the state of this project) in that mode a default login is the the db with username=’admin’ password=’admin’ you will get back in return your token:

```
{"key":"400a4e55c729ec899c9f6ac07818f2f21e3b4143"}
```

Then open to see the full auto-generated documentation of you API:

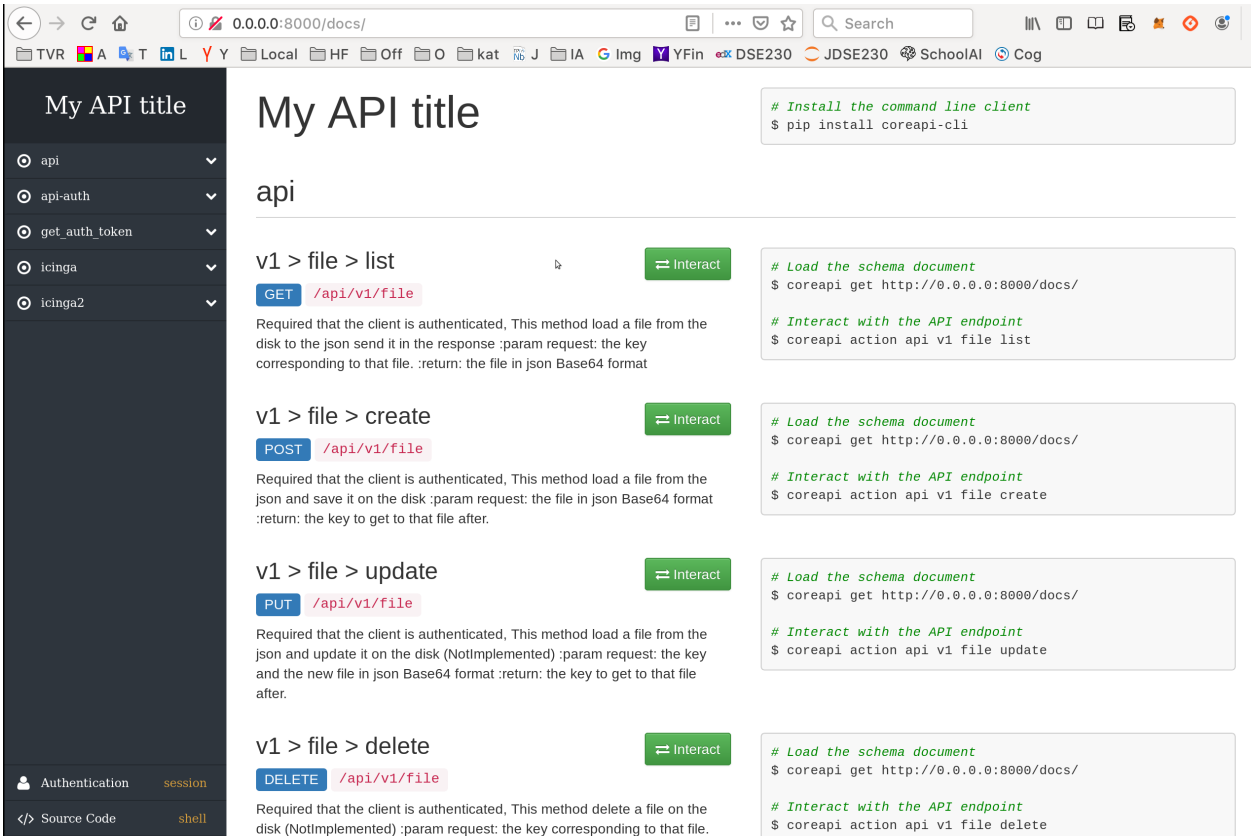
```
curl --request GET \
  --url http://127.0.0.1:8000/docs/ \
  --header 'authorization: Basic YWRtaW46YWRtaW4='
```

or by if BasicAuthentication is disabled and that wil be normally the case in prod and QA we use the Token:

```
curl --request GET \
  --url http://127.0.0.1:8000/docs/ \
  --header 'authorization: Token 400a4e55c729ec899c9f6ac07818f2f21e3b4143'
```

Then open

```
http://127.0.0.1:8000/docs/
```


0.0.0.0:8000/docs/
Search
TVR A T L Y Y Local HF Off O kat J IA Img YFin DSE230 JDSE230 SchoolAI Cog
My API title
api
api-auth
get_auth_token
icinga
icinga2
Authentication session
Source Code shell
My API title
# Install the command line client
$ pip install coreapi-cli
api
v1 > file > list
Interact
GET /api/v1/file
Required that the client is authenticated, This method load a file from the disk to the json send it in the response :param request: the key corresponding to that file. :return: the file in json Base64 format
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/
# Interact with the API endpoint
$ coreapi action api v1 file list
v1 > file > create
Interact
POST /api/v1/file
Required that the client is authenticated, This method load a file from the json and save it on the disk :param request: the file in json Base64 format :return: the key to get to that file after.
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/
# Interact with the API endpoint
$ coreapi action api v1 file create
v1 > file > update
Interact
PUT /api/v1/file
Required that the client is authenticated, This method load a file from the json and update it on the disk (NotImplemented) :param request: the key and the new file in json Base64 format :return: the key to get to that file after.
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/
# Interact with the API endpoint
$ coreapi action api v1 file update
v1 > file > delete
Interact
DELETE /api/v1/file
Required that the client is authenticated, This method delete a file on the disk (NotImplemented) :param request: the key corresponding to that file.
# Load the schema document
$ coreapi get http://0.0.0.0:8000/docs/
# Interact with the API endpoint
$ coreapi action api v1 file delete

CHAPTER 6

Testing

You can run the tests by

```
SECRET_KEY=my_secret_key python manage.py test
```

or by

```
python setup.py test
```

or by

```
DJANGO_SETTINGS_MODULE={{cookiecutter.app_name}}.config.local SECRET_KEY=my_secret_
↪key pytest
```

CHAPTER 7

Security check

Before dockerization for deployment to production, don't forget to check if by

```
SECRET_KEY=my_secret_key python manage.py check --deploy
```

CHAPTER 8

If you Use Aws

Aws secret:: WORK IN PROGESS

DRF_MICROSERVICE_PASSWORD => a client API password

Aws Env required:

```
AWS_REGION_NAME => default="eu-east-1"
AWS_DRF_MICROSERVICE_SECRET_NAME =>The name of the secret bucket
```

CHAPTER 9

Build and run the image with Docker

Build the Docker image:

```
docker build -t my-drf -f Dockerfile.drf-microservice .
docker build -t my-nginx -f Dockerfile.nginx .
```

Run the container:

```
docker network create my-network
docker run -d --name drf --net my-network -v /app my-drf
docker run -d --name nginx --net my-network -p "5000:80" my-nginx
```

If you want to change the port binding, it's here...

Build and run wit docker-compose:

```
docker-compose up
```

CHAPTER 10

Functionalities DONE

- support basic auth
- support token auth
- endpoint json file POST,GET
- endpoint login/logout
- endpoint get tocken
- postgreSQL support

CHAPTER 11

DevOps tools DONE

- the docker-compose configuration file
- endpoint get status Nagios/Icinga2

CHAPTER 12

Functionalities TODO

- AWS ssm secret
- endpoint json file DELETE,PUT?
- **create different version:**
  - Aws S3 support (in progress)
  - Aws RDS support
  - Aws Elastisearch support
  - Redis support
  - Aerospike support
  - ...

CHAPTER 13

DevOps tools TODO

- the docker-image configuration file (in progress)
- the Packer configuration file (in progress)
- the Terraform configuration file AWS (in progress)
- the Terraform configuration file GCD
- the Terraform configuration file Azure
- add getSentry support
- add Aws Cloudwatch support
- the Ansible configuration file AWS
- the Ansible configuration file GCD
- the Ansible configuration file Azure
- the Juju configuration file AWS
- the Juju configuration file GCD
- the Juju configuration file Azure