Taskstack Documentation

(work in progress)

Tech Stack 4
Python, JavaScript (vanilla), HTML, CSS 4
Flask as a web application framework 4
Flask-SocketIO on the server + SOCKET.IO on client-side for bi-directional communication between the clients and the server(s) 4
uWSGI as the application server 4
NGINX web server as a reverse proxy and load balancer (with sticky sessions) for the application server(s) 4
MySQL database + SQLAlchemy 4
Redis as a message queue (for Flask-SocketIO to support multiple servers) 4
Docker (+ Docker-Compose) 4
AWS EC2 to host the server(s) 4
AWS S3 to host static files + client uploaded files 4
AWS RDS to host the database 4
AWS SES for email sending 4
Namecheap 4
Notes for development 5
Running for development 6
Requirements 6
Start 6
Set environment variables -> /taskstack/.env 6
Run the development server 6
Deployment 7
Server requirements 7
Initial deployment 7
Get the code 7
Set environment variables -> /taskstack/.env 7
Disable THP 7
Build and start 7
Enable https 7
Updating 7
Stop affected service (taskstack |nginx |redis) 7
Get the code 7
Re-build and start affected service (taskstack |nginx |redis) 7
Code style: naming conventions 8
Python 8
Javascript 8

Db: table and column names 8
SocketIO/ Fetching 8
HTML/ CSS 8
UX Design 9
Button placement and order 9
Dialog 9
Page 9

Tech Stack

Python, JavaScript (vanilla), HTML, CSS

Flask as a web application framework

Flask-SocketIO on the server + SOCKET.IO on client-side
for bi-directional communication between the clients and the server(s)

uWSGI as the application server

NGINX web server as a reverse proxy and load balancer (with sticky sessions)
for the application server(s)

MySQL database + SQLAlchemy

Redis as a message queue (for Flask-SocketIO to support multiple servers)

Docker (+ Docker-Compose)

AWS EC2 to host the server(s)

AWS S3 to host static files + client uploaded files

AWS RDS to host the database

AWS SES for email sending

Namecheap

Notes for development

- always import the render_template func from taskstack/app
- gevent monkey-patching is done in production environment (by uWSGI)

Running for development

Requirements

- Python 3.*
- Python modules: /taskstack/requirements.txt (except uwsgi, gevent, redis)
- Local MySQL database
- AWS CLI
- Port 5000 open

Start

Set environment variables -> /taskstack/.env

FLASK_ENV=development
TASKSTACK_DB_URI=mysql+pymysql://[username]:[password]@127.0.0.1/[db name]
TASKSTACK_SECRET_KEY=?
AWS_ACCESS_KEY_ID=?
AWS_SECRET_ACCESS_KEY=?
AWS_DEFAULT_REGION=eu-central-1

Run the development server

- Execute /taskstack/wsgi.py
  Access the server -> http://127.0.0.1:5000

Deployment

Server requirements

- OS: Ubuntu
- Docker (+ Docker-Compose)
- Git
- Port 80 and 443 open for all incoming and outgoing connections

Initial deployment

Get the code

```
$  git clone https://github.com/juliuskrahn/taskstack
```

Set environment variables -> /taskstack/.env

FLASK_ENV=production
TASKSTACK_DB_URI=?
TASKSTACK_SECRET_KEY=?
AWS_ACCESS_KEY_ID=?
AWS_SECRET_ACCESS_KEY=?
AWS_DEFAULT_REGION=eu-central-1

Disable THP

```
$  echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

(needs to be done again if the server is shut down)

Build and start

```
$  docker-compose build
$  docker-compose up -d
```

Enable https

```
$  docker exec -it nginx bash
$  sudo certbot –nginx
```

Updating

Stop affected service (taskstack |nginx |redis)

```
$  docker-compose stop [service]
```

Get the code

```
$  git pull
```

Re-build and start affected service (taskstack |nginx |redis)

```
$  docker-compose build [service]
$  docker-compose up -d [service]
```

(If you need to rebuild the nginx service, make sure to enable https again!)

Code style: naming conventions

Python

- Globals (hard coded string/ number): uppercase; words separated by underscores (A_GLOBAL_VARIABLE)
- Variable/ function name: lowercase; words separated by underscores (my_variable)
- Class: upper camel case (MyClass)
- Data dict for client side rendering - keys: camel case (myKey)

Javascript

- Globals (hard coded string/ number): uppercase; words separated by underscores (A_GLOBAL_VARIABLE)
- Local variable:
    - If refers to something: style of the name referred to (suffix/ prefix: lowercase; words separated by underscores)
    - Else: lowercase; words separated by underscores
- Variable/ function name: camel case (myVariable) (if refers to something -> suffix/ prefix: lowercase; words separated by underscores)
- Func collection object: upper camel case (MyFuncCollection)
- Class: upper camel case (MyClass)

Db: table and column names

- lowercase; words separated by underscores (my_table)

SocketIO/ Fetching

- SocketIO event name: lowercase; words separated by underscores (my_event)
- Json/ dict data key naming: camel case (myKey)

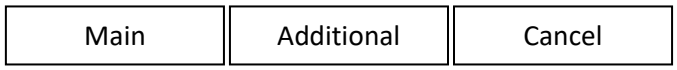
HTML/ CSS

- Id/ class name: camel case (myTextBox) (prefixes: lowercase; separated by a dash)

UX Design

Button placement and order

Dialog



Page