#### Technologies Used:

- Docker

- Python

- Flask

- SQLAlchimy

- SQLLite DB

- jquery

- Chart JS

- Tweepy

- semantic.min.css

- bootstrap 4

- Python Libraries

#### Project Files:

.

├── app.py ## Main file to run flask app

├── build.log ## log i have saved during build process

├── dashboard ## Dashboard module

│   ├── config ## config module

│   │   ├── app.db ## sqlLite DB file

│   │   ├── config.py ## config class which has some configuration properties

│   │   └── \_\_init\_\_.py ## init file for the config module

│   ├── \_\_init\_\_.py ## init file for dashboard module

│   ├── models ## model module

│   │   ├── base.py ## base db file which has the db connection variable

│   │   ├── \_\_init\_\_.py ## init file for the models module

│   │   ├── stream\_listener.py ## stream\_listener class

│   │   ├── tweets.py ## tweet model class to store and retrive tweets from DB

│   │   ├── twitter\_client.py ## TwitterClient class

│   │   └── users.py ## User class

│   ├── routes ## routes module

│   │   ├── \_\_init\_\_.py ## init file for routes module

│   │   └── main.py ## main controller class which has all the functions mapping

│   │   └── analysis.py ## analysis controller class which has all the functions mapping

│   │   └── stream.py ## stream controller class which has all the functions mapping

│   │   └── bot.py ## bot controller class which has all the functions mapping

│   │   └── DIY\_analysis.py ## DIY\_analysis controller class which has all the functions mapping

│   │   └── DIY\_visualization.py ## DIY\_visualization controller class which has all the functions mapping

│   ├── static ## static folder which has all static js and css files

│   │   └── css ## static css folder

│   │   └── user.css ## user page css file

│   │   └── data ## static css folder

│   │   └── bar.json ## sample json formate for bar graph

│   │   └── circle.json ## sample json format for circle graph

│   │   └── line.json ## sample json format for line graph

│   └── templates ## html templates folder

│   ├── index.html ## index page html file

│   ├── nav.html ## navigation html

│   └── user.html ## user page html file

│   └── footer.html ## footer which include al js file

│   └── DIY\_analysis.html ## DIY\_analysis html file

│   └── DIY\_visualization.html ## DIY\_visualization html file

│   └── user.html ## user page html file

│   ├── stream.html ## custom stream html

│   └── bot.html ## bot page html file

│   └── analysis.html ## analysis page html file

├── Dockerfile ## Docker config file

├── migrations ## migration folder which is autogenerated after flask db init command

├── readme\_flask.md ## readme file for flask/dashboard

├── readme.md ## readme for docker details

├── requirements.txt ## list of libraries used in this project and this file is used to install them

└── tests ## test module

├── \_\_init\_\_.py ## init for test module

├── test\_api.py ## test for api endpoints working or not

├── test\_cli.py ## test for cli commands working or not

├── test\_auth.py ## test for API key working or not

├── test\_db.py ## test for DB operations working or not

├── tests\_app.py ## test for APP running or not

└── test\_views.py ## test for correct view rendering or not

└── test\_bot.py ## test for bot functionalities

#### Steps to initialize the DB

- flask db init ## init the DB

- flask db migrate ## create migration files

- flask db upgrade ## commit changes to DB and creates tables

#### command for unit testing the files:

- python3 -m unittest tests/test\_views.py

- python3 -m unittest tests/test\_auth.py

- python3 -m unittest tests/test\_api.py

- python3 -m unittest tests/test\_db.py

- python3 -m unittest tests/test\_app.py

- python3 -m unittest tests/test\_bot.py

#### command for docker build:

- docker build -t <imagename:version> .

- -t for Allocate a pseudo-TTY

#### command to run docker container after build success

- docker run -it -d -p <outside-port-of-your-choice>:<inside port of application> <imagename:version>

- -i, --interactive Keep STDIN open even if not attached

- -t for Allocate a pseudo-TTY

- -d for Demon process

- -p port

#### Explain of Dockerfile:

```

# Install the latest alpine linux which is docker default

FROM alpine:latest

# install python3 and pip in container

RUN apk add --no-cache python3-dev \

&& pip3 install --upgrade pip

# set the working directory to /app

WORKDIR /app

# copy the content of current folder to the /app folder of the image

COPY . /app

# run the command pip3 to install requiremenrs on the image

RUN pip3 install -r requirements.txt

# expose the port 5000

EXPOSE 5000

# entry point is the command node when container starts

ENTRYPOINT ["python3"]

# the arguments of entry command : i.e command is : python3 app.py

CMD ["app.py"]

```

#### create\_app function:

```

# import flask

from flask import Flask

# import models and routes

from . import models, routes

# import db and migration objects

from .models.base import db, migrate

# import config file

from .config import Config

def create\_app():

# create flask app with static flder as /dashboard/static

app = Flask(\_\_name\_\_, static\_url\_path='/dashboard/static')

## Take the configuration for Config class we imported

app.config.from\_object(Config)

## initialize the db with flask app (context)

db.init\_app(app)

## initialize migrate object with DB and app object

migrate.init\_app(app, db)

## initialize the models with app context for Db operaions and all

models.init\_app(app)

## initialize the routes to be recognizable by application

routes.init\_app(app)

## return the created app object

return app

```

#### Register the controller main

```

## import the main

from .main import main

## use to initialize the main with app context (very imp to recognize the url mappings)

def init\_app(app):

app.register\_blueprint(main)

```

#### main.py

## register the Blueprint with the url prefix for the main module

## it is '/' as it is the main module

- main = Blueprint('main', \_\_name\_\_, url\_prefix='/')

## create the object of out TwitterClient (which is used to fetch data from twitter)

- api = TwitterClient()

## main.py file is commented

##

#### analysis.py

## register the Blueprint with the url prefix for the main module

## it is '/' as it is the analysis module

- analysis = Blueprint('analysis', \_\_name\_\_, url\_prefix='/')

## create the object of out TwitterClient (which is used to fetch data from twitter)

- api = TwitterClient()

## analysis.py file is commented

#### stream.py

## register the Blueprint with the url prefix for the stream module

## it is '/' as it is the analysis module

- stream = Blueprint('stream', \_\_name\_\_, url\_prefix='/')

## create the object of out TwitterClient (which is used to fetch data from twitter)

- api = TwitterClient()

## stream.py file is commented

#### bot.py

## register the Blueprint with the url prefix for the bot module

## it is '/' as it is the analysis module

- bot = Blueprint('bot', \_\_name\_\_, url\_prefix='/')

## create the object of out TwitterClient (which is used to fetch data from twitter)

- api = TwitterClient()

## bot.py file is commented

#### models we have used:

- Tweet in tweets.py

- define the table column details in the class

- function to save and rollback

- User in users.py

- defined table details in the user class

- functions to save and rollback

- StreamLisiner Class:

### Config File:

## file which will be used as sqllite db by sqlalchemy library

- SQLALCHEMY\_DATABASE\_URI = os.environ.get('DATABASE\_URL') or \

'sqlite:///' + os.path.join(basedir, 'app.db')

#### javascript and ajax used:

- ajax call on click of start stream togge;

- ajax call on automatic for total tweets and total user update

- ajax calls on graph generation per min and per 10 seconds

- # /\* on click of stream toggle button fire the streaming\*/

# $('#stream').on('click',function(){

# clear the auto interval call to function getCount

clearInterval(getCountInt);

# clear the auto interval call to function getCount10

clearInterval(getCount10Int);

# set the auto interval call to function getCount every 5 seconds

getCountInt=setInterval(getCount, 5000);

# set the auto interval call to function getCount10 every 5 seconds

getCount10Int=setInterval(getCount10, 5000);

setInterval(getTotal, 5000);

# set the auto interval call to function getTop100 every 5 seconds

setInterval(getTop100, 5000);

## Docker Commands:

- list all containers:

docker ps -a

- remove all containers

sudo docker rm $(docker ps -a -q)

- kill one contaner with iD

sudo docker kill ID\_OF\_CONTAINER

- docker build command:

sudo docker build -t flask-twitter:latest .

- docker run command:

docker run -it -p 5000:5000 flask-twitter:latest

- docker run the background (demon process)

docker run -it -d -p 5000:5000 flask-twitter:latest

### NEW

- In custom stream page you can enter multiple keywords and serach stream with that

- In analysis page we have analysis of whole twitter data we receicved

- In bot page we can comment retweet and like the tweets

- test cases have been updated

### NEW 3

- pages:

- DIY analysis

- DIY visualization

- commandline implementation

#### CLI APP:

- cli interface has been implemented with two features:

- the start stream feature

- start stream will show the live streaming tweets

- the my profile profile

- this will show the user data and tweets

#### DIY\_analysis.py

- it contains all the codes which is doing the diy analysis

- it takes user input and return the analysis results

- flixible to choose whcih to be positive and negative marks

- we are doing analysis on the user input text for entimental and tags

#### DIY\_visualization.py

- it contains all the codes which is doing the diy visualizartion

- it takes user input and return graph for the proper json input

- flixible to choose whcih type of graph line,bar and circle

- it accepts only json and has a specific formate and if format changes it is not going to visualize

### command line :

commands:

- root folder

export FLASK\_APP=app.py

- to run stream:

flask cli start\_stream

- to run my\_profile

flask cli my\_profile

If it doesn't work, download mobaxterm from https://mobaxterm.mobatek.net/download-home-edition.html

- go to the folder with your pem

- ssh -i "cs241.pem" ubuntu@ec2-34-217-212-73.us-west-2.compute.amazonaws.com

- go to the twitter\_diy folder

- try export FLASK\_APP=app.py

- then try flask cli start\_stream or flask cli my\_profile

### Files:

- Files are located in dashboard -> static -> data

- circle.json

- line.json

- bar.json