Team member CWID# Role

Alex Tran 891297442 Ops

Tu Tran 888750130 Dev (Posting Microservices)

Joseph Hoang 889782900 Dev (Voting Microservices)

**Project Overview**

The scope of this project is to be able to implement two Microservices: Posting, Voting.

We will also need to be able to run multiple instances of each Microservices while using

load balancer to divide up the requests from each instance.

**Instructions to run**

**These are the library, and tools that will need to be install in order to run the project**

pip3 install --user Flask-API python-dotenv

pip3 install --user pugsql

sudo apt install --yes ruby-foreman

**First, we need to init the database for the both Posting and Voting Microservices**

flask init

**If you only run one instance per Microservices, please refresh multiple times.**

**Because of load balancer, it will take 1-2 to get to the correct $PORT**

**Then we will start the project by runing:**

foreman start

**Since, this the project required starting 3 instances of each Microservices.**

**Port number already been hardcoded in to work with only 3 or less instances.**

**If you want to run multiple instances, please use this link to run the project:**

foreman start -m web1=<number of instances => 3>,web2=<number of instances => 3>,caddy=1

**Flask Error**

If you get Error: No such command init, please create .env file

.env will contain the following {

FLASK\_APP=vote

FLASK\_APP=api

FLASK\_ENV=development

APP\_CONFIG=api.cfg

}

**Port In Use Error**

sudo lsof -i -P -n | grep LISTEN

kill <port id>

**Developer Ops**

For multiple instances, we have to hardcode a specific port number in in order to run load balancer.

For example, posts will have port 5000-5002 available

Votes will have port 5100-5102 available

For Ops role, I will be using Caddy for webserver and Gunicorn for WSGI server

User will be able to access to Post or Vote Microservices by using these hosts:

localhost:2015/posts

localhost:2015/votes

**Voting Microservices**

* View all votes:  [localhost:2015/vote/api/v1/resources/votes/all](http://127.0.0.1:5000/api/v1/resources/votes/all)
* View votes by vote id:  [localhost:2015/vote/api/v1/resources/votebyid/1](http://127.0.0.1:5000/api/v1/resources/votebyid/1)
* Upvote a post:  [localhost:2015/vote/api/v1/resources/upvote](http://127.0.0.1:5000/api/v1/resources/upvote)

Have to input in json format.

Example {“postID”: 10}

* Downvote a post:  [localhost:2015/vote/api/v1/resources/downvote](http://127.0.0.1:5000/api/v1/resources/downvote)

Have to input in json format.

Example {“postID”: 11}

* Report the number of upvotes and downvotes for a post:

localhost:2015/vote/api/v1/resources/votesbypostid/101

* List the *n* top-scoring posts to any community:

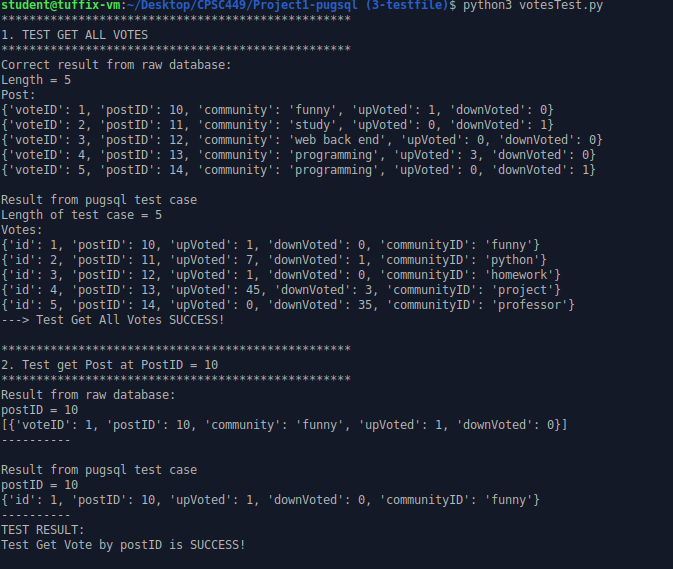
localhost:2015/vote/api/v1/resources/toppostscore/3

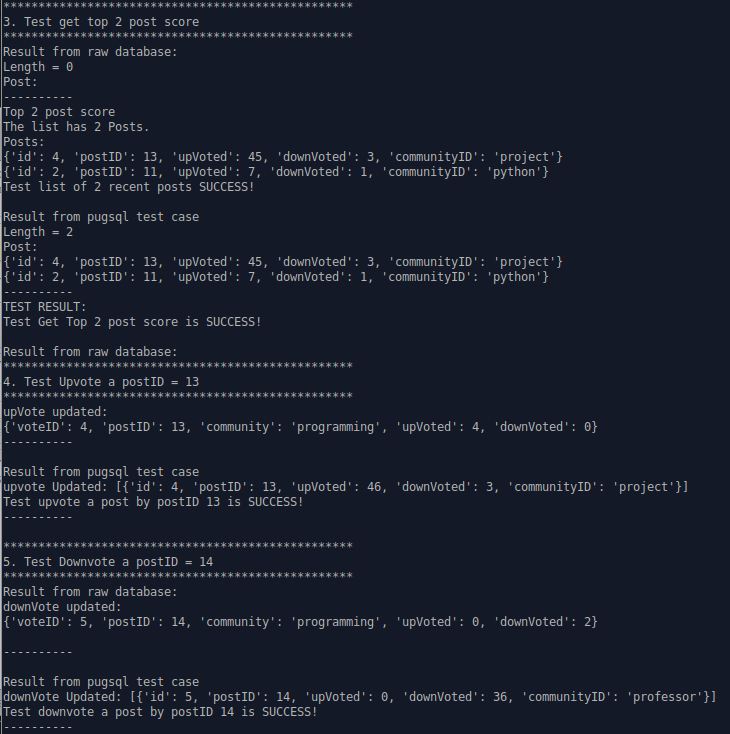
* Given a list of post identifiers, return the list sorted by score:

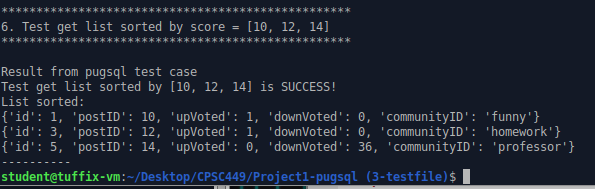
[localhost:2015/vote/api/v1/resources/listsortedbyscore](http://127.0.0.1:5000/api/v1/resources/listsortedbyscore)

Have to input Example: {"listID": listPostID}

* Test case:







**Posting Microservices**

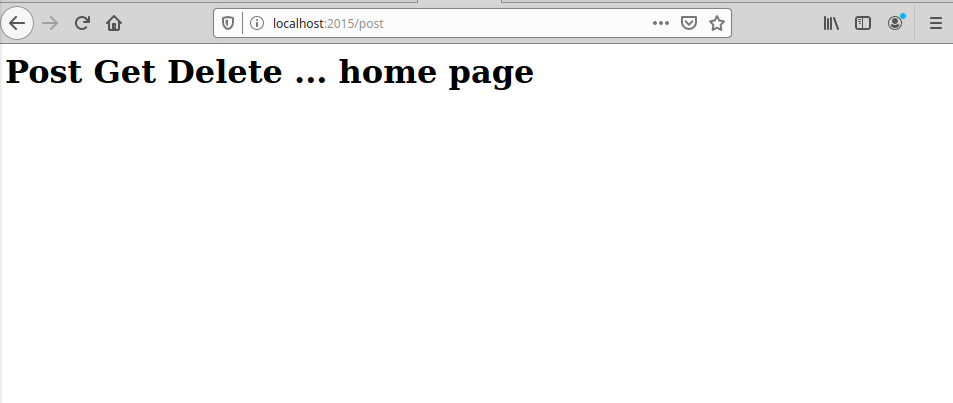
While having a terminal run “foreman start”, open another terminal in the same directory run “python3 PostTest.py”

For Posting Microservices, below is the following route to specific functions

To view home page for post:

REQUEST GET

localhost:2015/post



To view all posts:

REQUEST GET

http://localhost:2015/post/posts/all

The first 9 posts is generate when init. They all are created at the same time

From post number 10 to post number 19, there is increment in time creation.

Later on, it will be easier see the list by time when query.





To view a specific post:

REQUEST GET

localhost:2015/post/posts/<id>

Example:localhost:2015/post/posts/15



To view a specific community in posts:

REQUEST GET

http://localhost:5000/posts/<community name>

without n value, the default output will give 3 last posts

or

http://localhost:5000/posts/<community name>?n=<n>

with n value, the output will give n last posts

example:

localhost:2015/post/posts/Community\_1

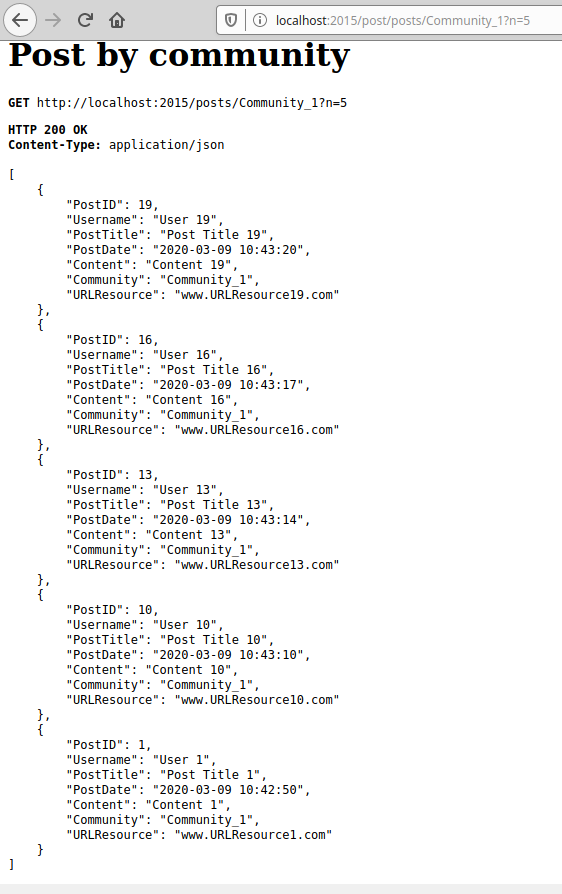
have 3 last posts

while

http://localhost:2015/post/posts/Community\_1?n=5

have 5 last posts

****

****

To view a specific, create:

The web will default display 5 last post on top

Bottom is a form to create a new post

json format is required, all fields are required,

REQUEST GET, POST

localhost:2015/post/posts

example:

{

"Username" : "User 100",

"PostTitle" : "Post Title 100",

"Content" : "Content 100",

"Community" : "Community 100",

"URLResource" : "www.URLResource100.com"

}

PostID and PostDate is generated automatically



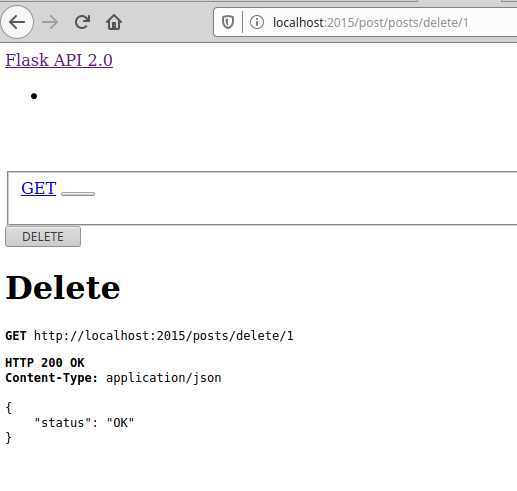
To delete a specific post:

REQUEST GET, DELETE

localhost:2015/post/posts/delete/<id>

example: localhost:2015/post/posts/delete/1

To delete the post, click on the delete button

****

**Testing**

1. Test get Post at ID = 17
   1. Correct Answer from database

Get post data from the database

Loop through the data to looking for post in which ID = 17

Store that JSON as answer

* 1. Result from test case

Get the JSON through request

Store that JSON as result

* 1. Print out Test result

Compare answer and result, if it’s equal then the test is successful

1. Test get n = 3 recent posts by the Community = Community\_2
   1. Correct Answer from database

Get post data from the database (already have from above)

Loop through the data from bottom to top

Looking for post Community in which community = Community\_2

Store those JSON as answer

* 1. Result from test case

Get the JSON through request

Store that JSON as result

* 1. Print out Test result

Compare answer and result, if it’s equal then the test is successful

1. Test get n = 4 posts by time
   1. Correct Answer from database

Get post data from the database (already have from above)

Loop through the data from bottom to top

Get 4 last posts

Store those JSON as answer

* 1. Result from test case

Get the JSON through request

Store that JSON as result

* 1. Print out Test result

Compare answer and result, if it’s equal then the test is successful

1. Test Post

myJSON = {

"Username" : "User 100",

"PostTitle" : "Post Title 100",

"Content" : "Content 100",

"Community" : "Community 100",

"URLResource" : "www.URLResource100.com"

}

* 1. Correct Answer from database

Get post data from the database (already have from above)

Current data have length = 19 which is before post

* 1. Result from test case

Post the “myJSON” to the database

The new data have length = 20 which is 1 greater than itself before add

* 1. Print out Test result

If the length after post – the length before post = 1

And check the post with community = “Community 100” Exist

Then the test is successful

Not only the length increase 1 but also JSON is exist be success

Note:

Post ID start at 1.

The database before post has 19 records

Add one more post, the last post will have PostID = 20

1. Test Delete post by PostID = 20
   1. Correct Answer from database

Get all post data from the database

The current data now have the length = 20

This length is also the length of the database before delete

* 1. Result from test case

Delete a post at PostID = 20

After delete Get the database won’t have the post at PostID = 20

* 1. Print out Test result

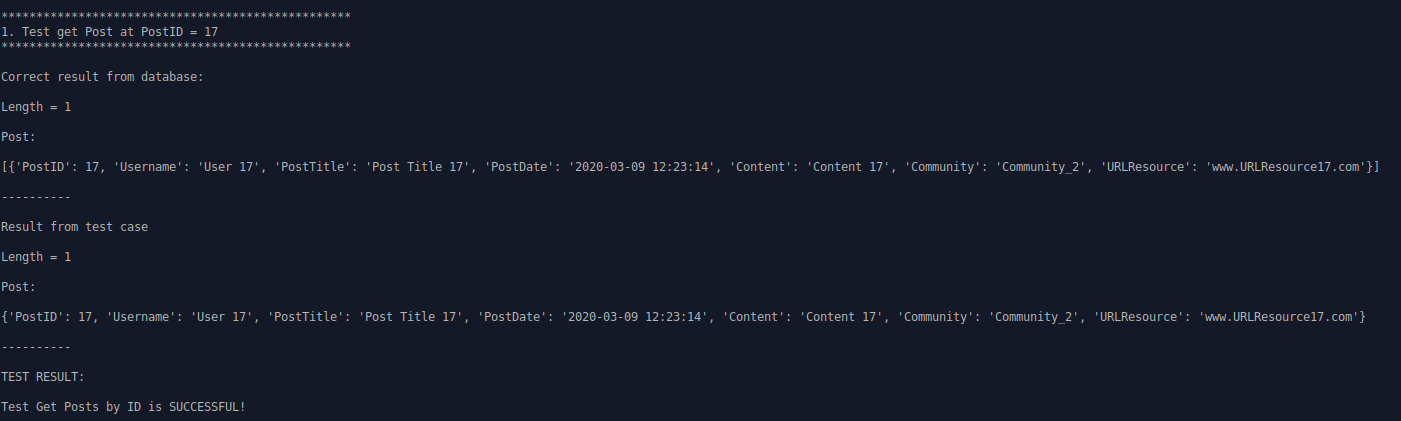
After delete Get the post at PostID = 20:

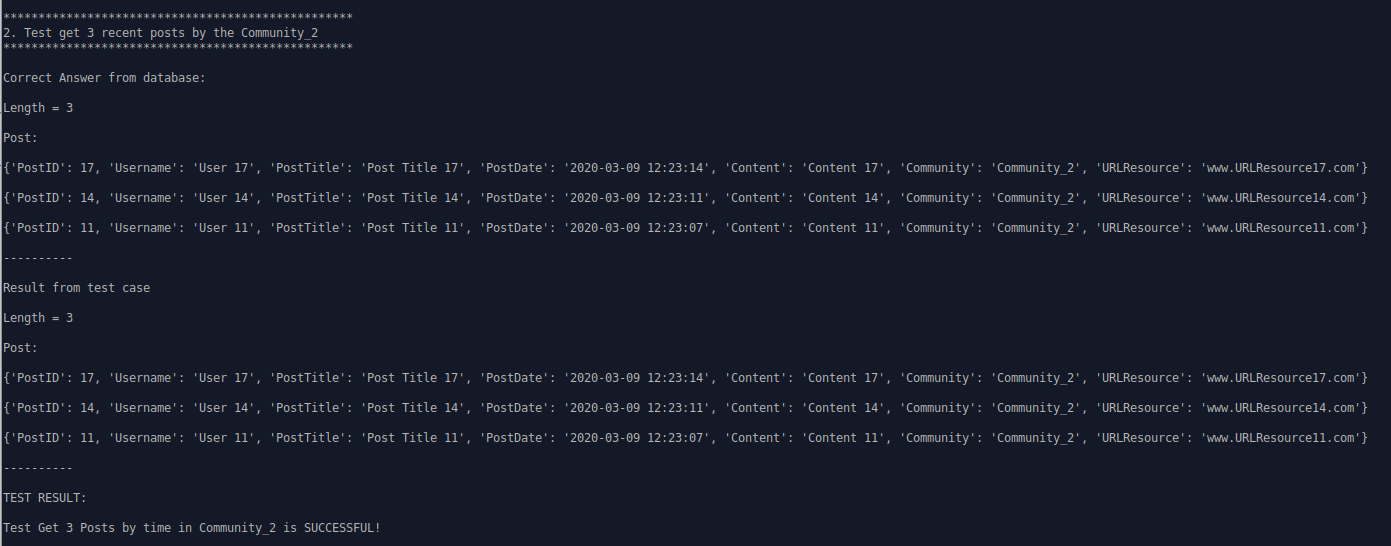
If the length before delete – the length after delete = 1

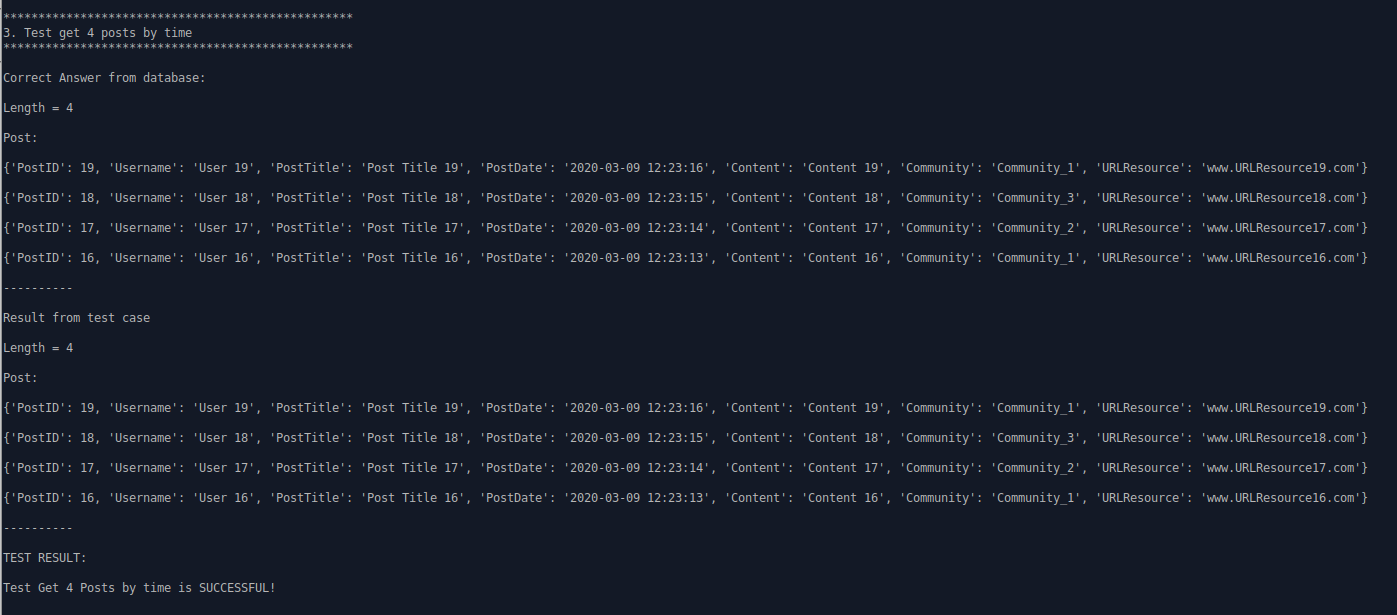
And check the post with community = “Community 100” not Exist

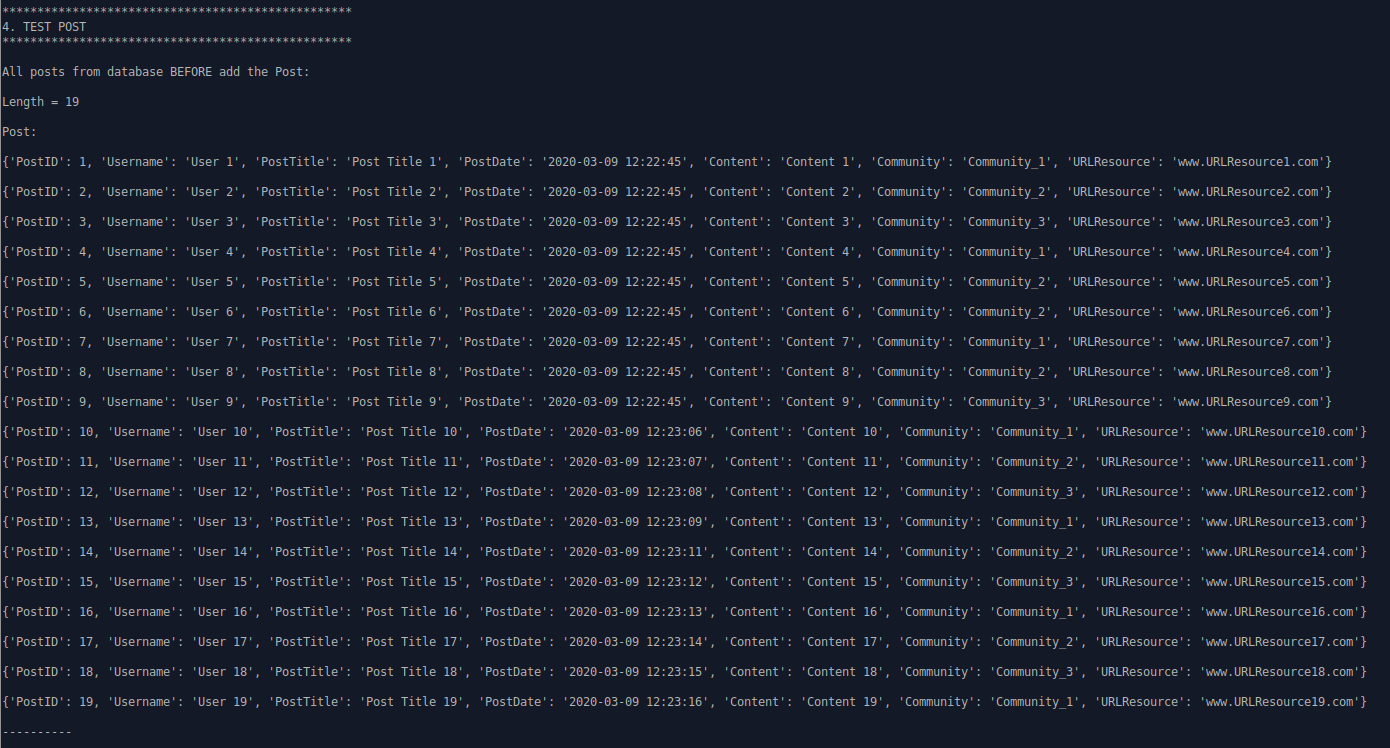
Then the test is successful

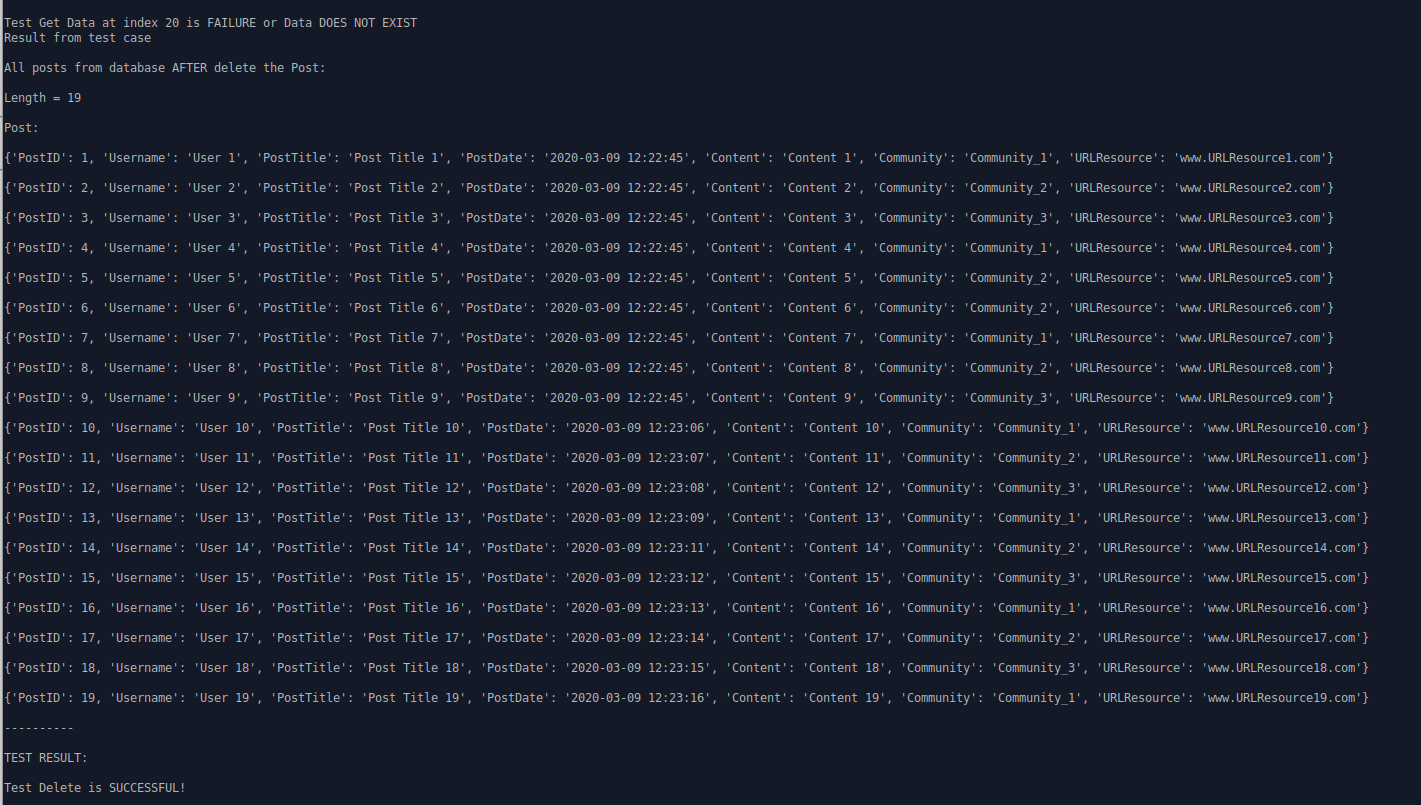
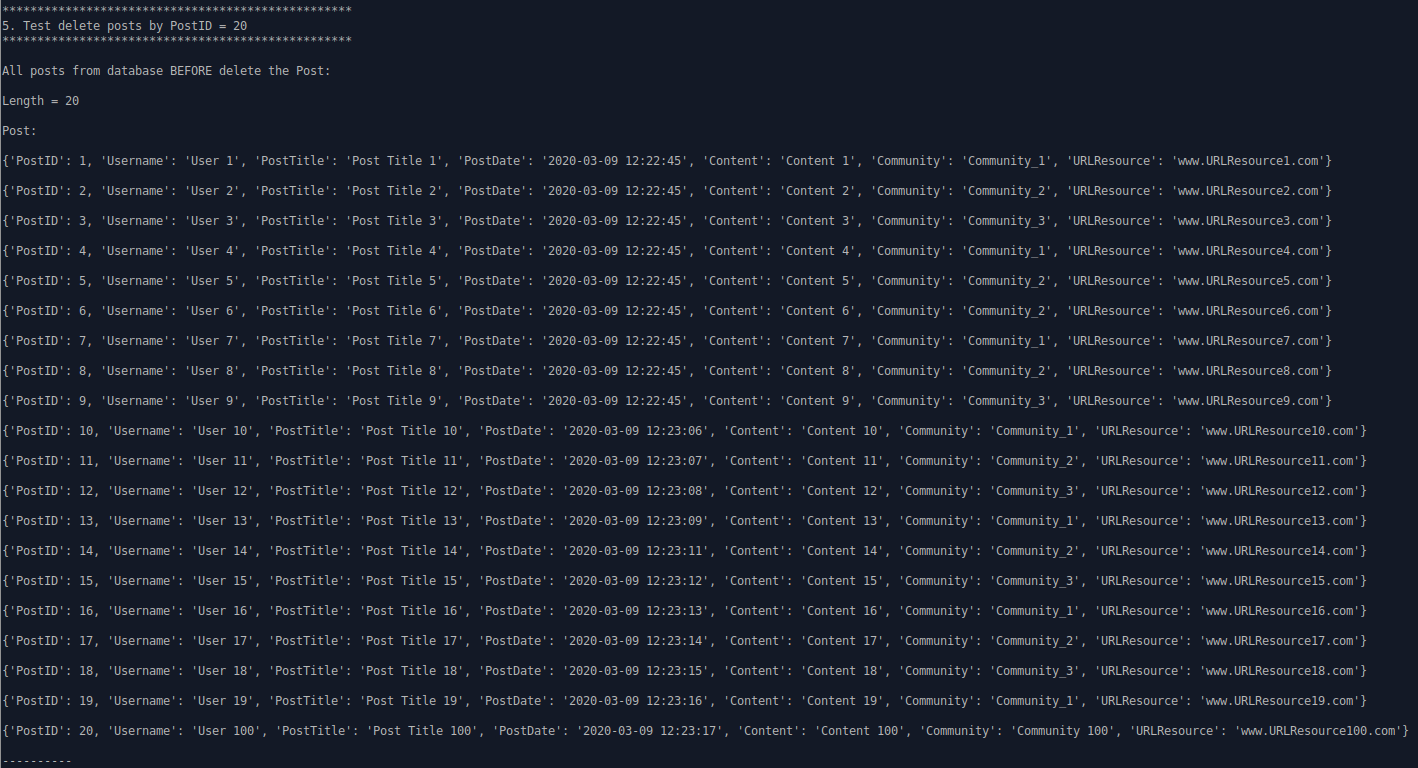
Not only the length decreases 1 but also deleted JSON need to be disappear to be success (because we can delete the wrong JSON and the length still match)

****

****

****

****

****