Overview

The goal of this project is to create a Reddit clone using microservices. The REST (REpresentational State Transfer) architectural design for web APIs is used to make requests to the database in the form of a JSON blob. These web services are created using Python and Flask.

Teams

Project 1

Name	Role	Contact Info
Javier Melendrez	Developer 1	javim1224@csu.fullerton.edu
Zexin Zhuang	Developer 2	zexinzhuang@csu.fullerton.edu
Nathaniel Richards	SDET	nathrich23@csu.fullerton.edu
Antonio Lopez	Operations	antonio_lopez@csu.fullerton.edu

Project 2

Name	Role	Contact Info
Javier Melendrez	Operations	javim1224@csu.fullerton.edu
Zexin Zhuang	SDET	zexinzhuang@csu.fullerton.edu
Nathaniel Richards	Developer 1	nathrich23@csu.fullerton.edu
Antonio Lopez	Developer 2	antonio_lopez@csu.fullerton.edu

Production Stack

This project is written in Python 3 and the deployment environment is in Tuffix 2019 Edition r2 (Xubuntu). Start by updating the Tuffix repo location by opening a terminal and running *sudo sed -i -re 's/([a-z]{2}.)?archive.ubuntu.com|security.ubuntu.com/old-releases.ubuntu.com/g' /etc/apt/sources.list* Enter your password if prompted. You can check the version of Python by running the command *python3 --version* If Python 3 is not installed you can get it using the command *sudo apt-get install python3* Once you have installed Python check the version of Pip you have by running the command *pip3 --version* Python includes Pip upon installation but if Pip cannot be found you can install it using the *sudo apt-get install pip3* command. Flask is a lightweight Python framework for web applications that provides the basics for URL routing and page rendering. To install, run *pip3 install flask*. The database is handled by the microservices using SQLite3 and is viewed using API requests, DB Browser for SQLite, or any database viewer of your choice. For this project we are using DB Browser for SQLite to visually show the database. You can install it by running the command *sudo apt-get install sqlitebrowser*. CURL (Client URL Request Library) requests are created using Postman. Requests update/retrieve information to/from the database. The retrieved information is outputted in JSON format.

These are the libraries/dependencies that are required for the project to run and test it. A shell script is provided at the end of the SETUP section to easily install all of them.

- Gunicorn 3
- Caddy
- Foreman

Architecture

Hosts

Environment	Microservice	Hostname
Test	Post (Project 1)	http://127.0.0.1:5000
Test	User (Project 1)	http://127.0.0.1:5000
Test	Message (Project 2)	http://127.0.0.1:5000
Test	Vote (Project 2)	http://127.0.0.1:5000
Prod	Post (Project 1)	http://localhost:2015/posts
Prod	User (Project 1)	http://localhost:2015/users
Prod	Message (Project 2)	http://localhost:2015/messages
Prod	Vote (Project 2)	http://localhost:2015/votes

Caddy configuration

Post Microservice Proxy (Production)	User Microservice Proxy (Production)
127.0.0.1:5000 127.0.0.1:5001 127.0.0.1:5002	127.0.0.1:5100 127.0.0.1:5101 127.0.0.1:5102
Message Microservice Proxy (Production)	Vote Microservice Proxy (Production)
127.0.0.1:5300 127.0.0.1:5301 127.0.0.1:5302	127.0.0.1:5200 127.0.0.1:5201 127.0.0.1:5202

Setup

These instructions will only work for Linux based distributions, specifically Tuffix. These steps will help create a working environment while installing all libraries and dependences.

Open a terminal and create a directory:
$ mkdir reddit_clone

Change the current directory:
$ cd reddit_clone/

Clone repository:
$ git clone https://github.com/antonio-lopez/cspc-449-project-1.git

Change the current directory:
$ cd cspc-449-project-1/

Create a virtual environment:
$ sudo apt-get update
$ sudo apt-get install python3-venv

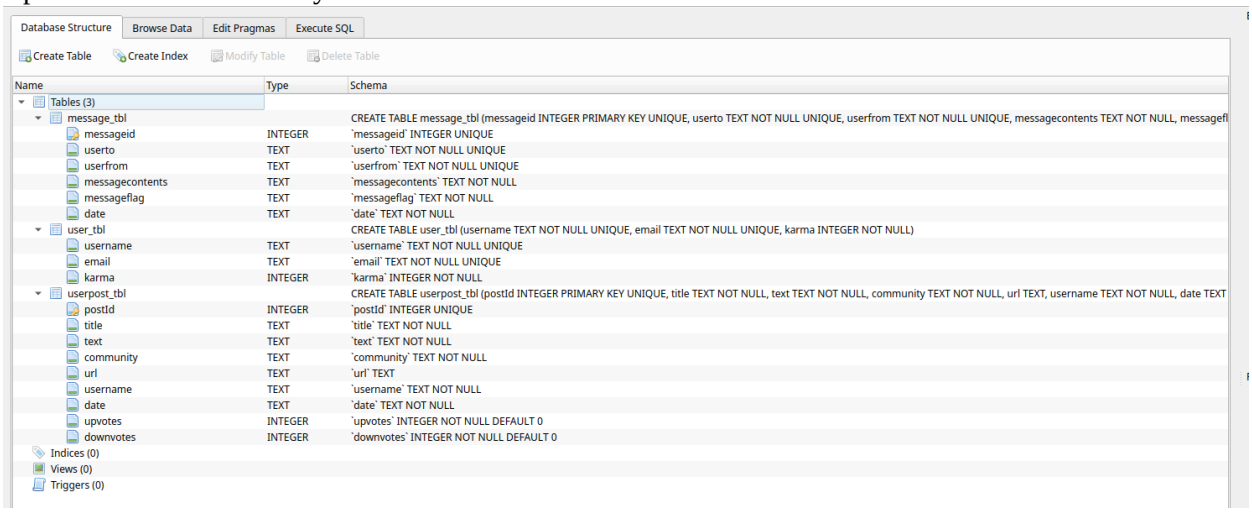
$ python3 -m venv ENV

Activate the virtual environment:
$ source ENV/bin/activate

Create the database by running the python file:
$ python3 create_db.py

Run shell script to download and install requirements:
$ bash requirements.sh

Open DB Browser to verify the Database has been created:

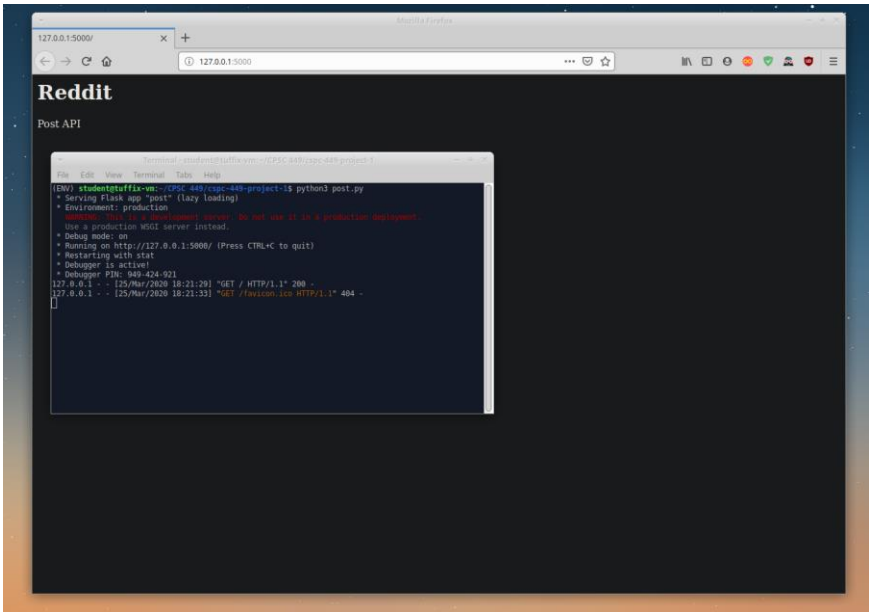


Run

To verify the setup is complete run any microservice.
$ python3 post.py
Or
$ python3 user.py
Or
$ python3 vote.py
Or
$ python3 message.py

Follow the link provided (ctrl + left click). Your browser will open with the main page greeting for the chosen microservice. Cancel the microservice (Ctrl + c).

Ex. Post microservice:



Testing

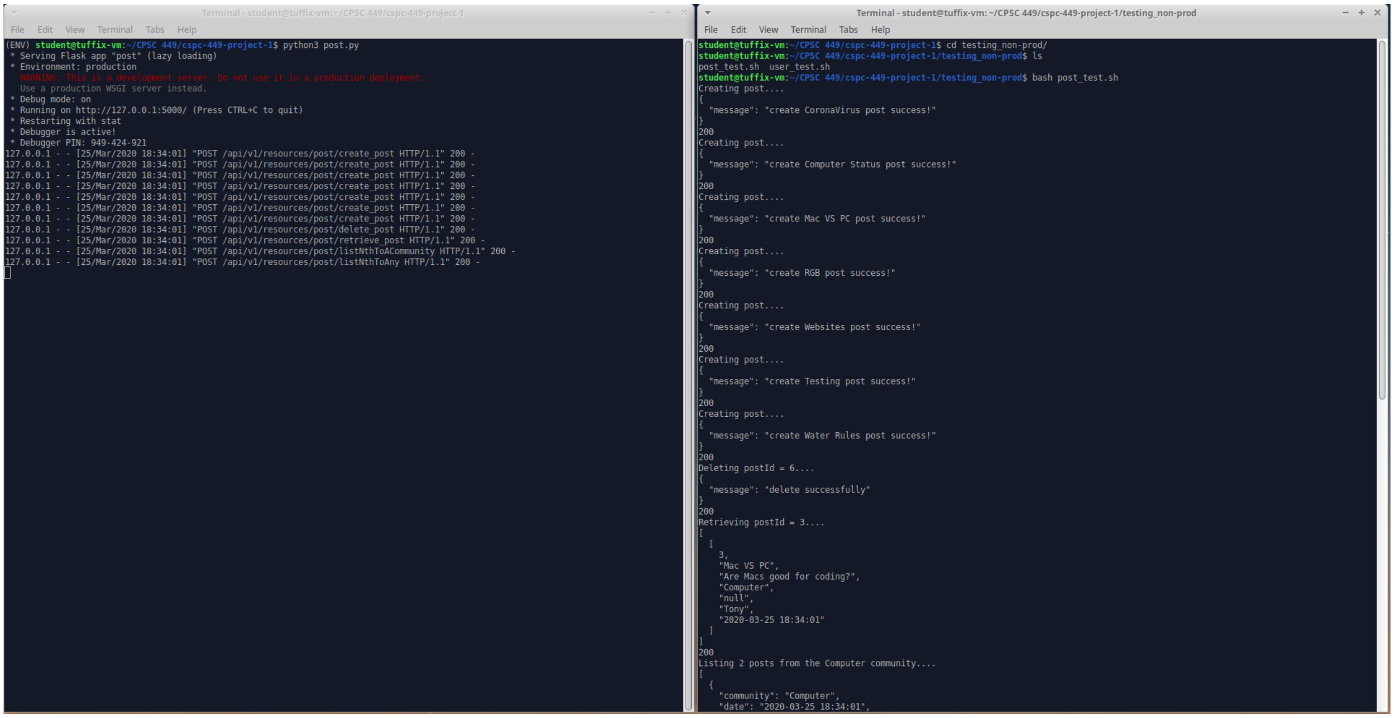
Testing can be done in either non-production or production. In the initial testing for either, make sure you are using a newly created database to view successful codes, otherwise you'll see conflict codes for duplicates. The testing scripts create new users and posts, send messages to users, and manipulates post voting operations. If you are following the instructions and have completed the SETUP and RUN sections continue with testing. If the database is populated, delete it and create a new one using the *$ python3 create_db.py* command.

Testing (Non-production):

Non-production testing is done using the Flask Development Server.

Post Microservice Testing:

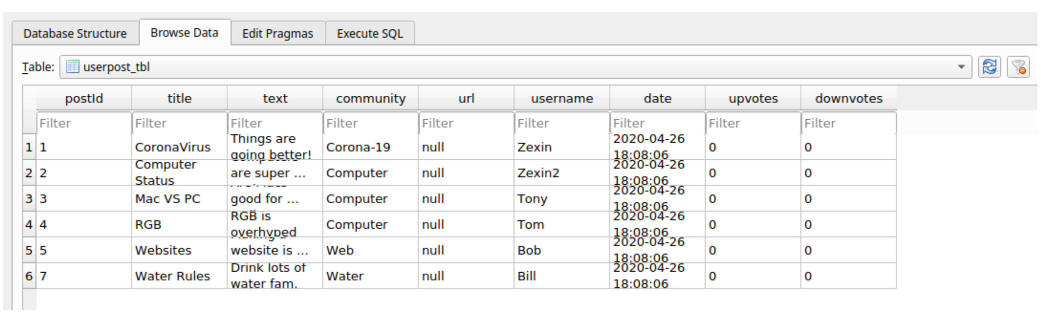
Run the Post microservice on a terminal from the cspc-449-project-1 directory:
$ python3 post.py
Open a new terminal from the cspc-449-project-1 directory, change to the testing directory, and run the Post testing shell script:
$ cd testing_non-prod/
$ bash post_test.sh



Running the script tests the following API requests:

- Create a new post
- Delete an existing post given a postId
- Retrieve an existing post given a postId
- List the n most recent posts to a particular community
- List the n most recent posts to any community

To verify if the posts were added to your database, open DB Browser and select the "Browse Data" tab and select the "userpost_tbl" table.



	postId	title	text	community	url	username	date	upvotes	downvotes
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
1	1	CoronaVirus	Things are going better!	Corona-19	null	Zexin	2020-04-26 18:08:06	0	0
2	2	Computer Status	are super ...	Computer	null	Zexin2	2020-04-26 18:08:06	0	0
3	3	Mac VS PC	good for ...	Computer	null	Tony	2020-04-26 18:08:06	0	0
4	4	RGB	RGB is overhyped	Computer	null	Tom	2020-04-26 18:08:06	0	0
5	5	Websites	website is ...	Web	null	Bob	2020-04-26 18:08:06	0	0
6	7	Water Rules	Drink lots of water fam.	Water	null	Bill	2020-04-26 18:08:06	0	0

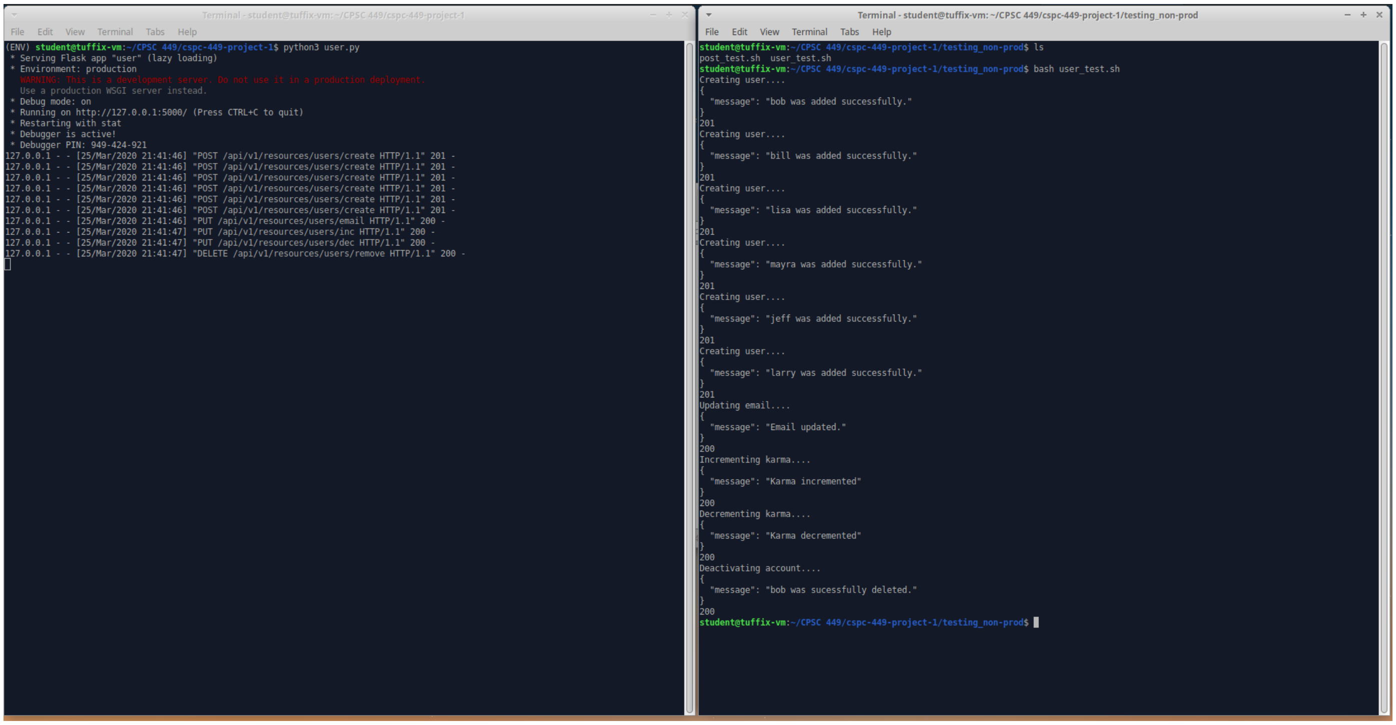
<u>User Microservice Testing:</u>

Cancel the Post microservice (Ctrl + C) and run the User microservice in the same terminal:

$ python3 user.py

On the other terminal run the User testing shell script:

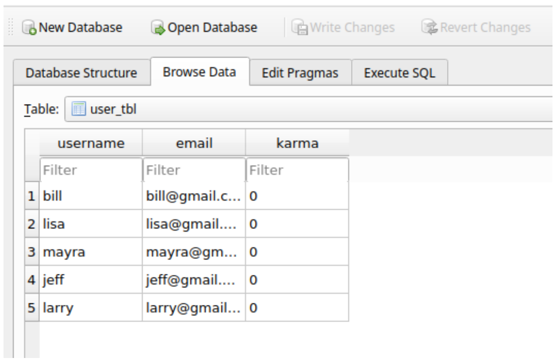
$ bash user_test.sh



Running the script tests the following API requests:

- Create user
- Update email
- Increment Karma
- Decrement Karma
- Deactivate account

Check DB Browser to verify if the data was inserted into the database by selecting the “user_tbl” table.

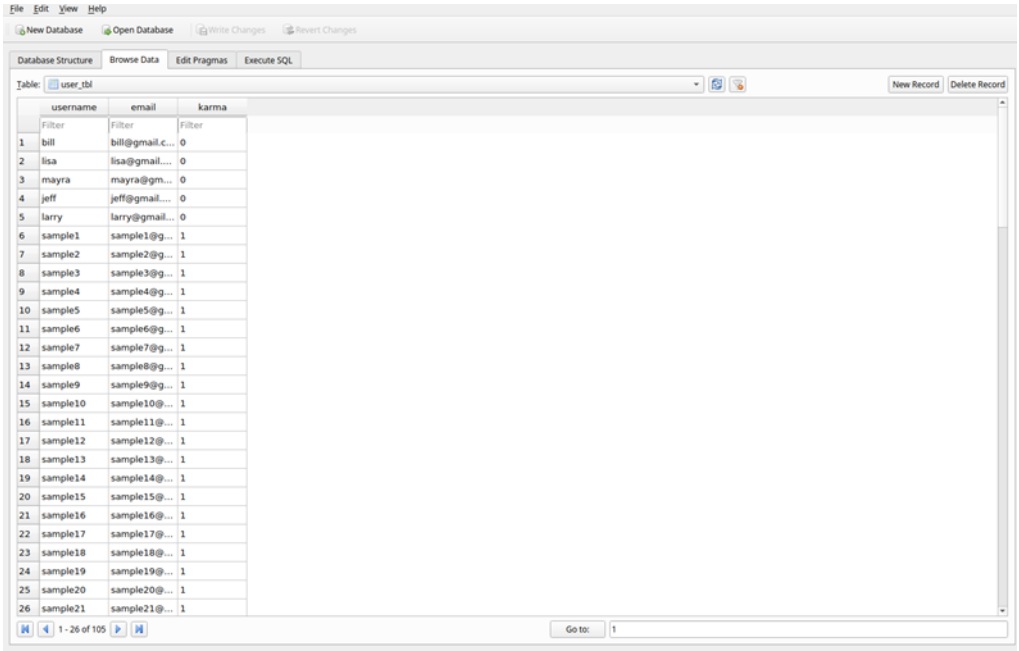


<u>100 User Testing:</u>

On the same terminal where you ran the User testing shell script, run the 100 Users Test shell script:

$ bash 100_users_test.sh

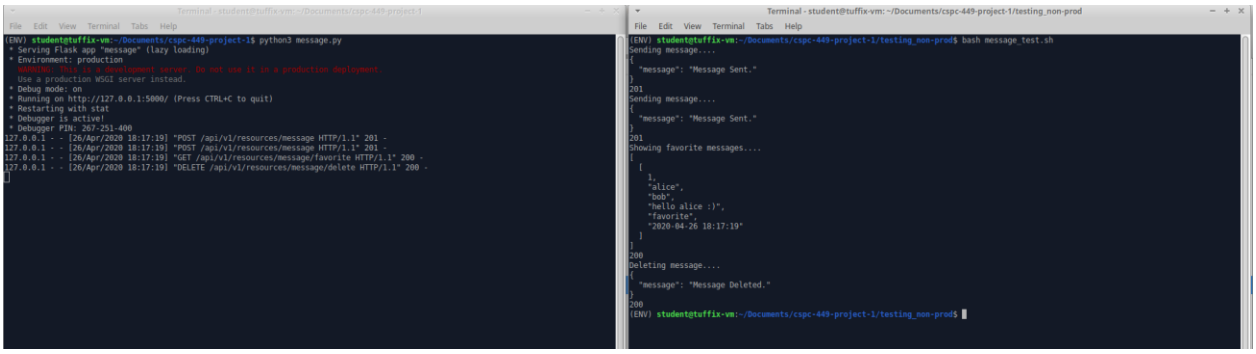
Refresh DB Browser and verify the 100 users were added to the database:



Message Microservice Testing:

Cancel the User microservice (Ctrl + C) and run the Message microservice in the same terminal:
$ python3 message.py

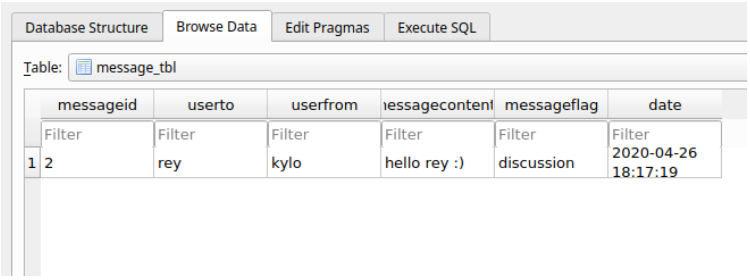
On the other terminal run the Message testing shell script:
$ bash message_test.sh



Running the script tests the following API requests:

- Send message
- Delete message
- Favorite message

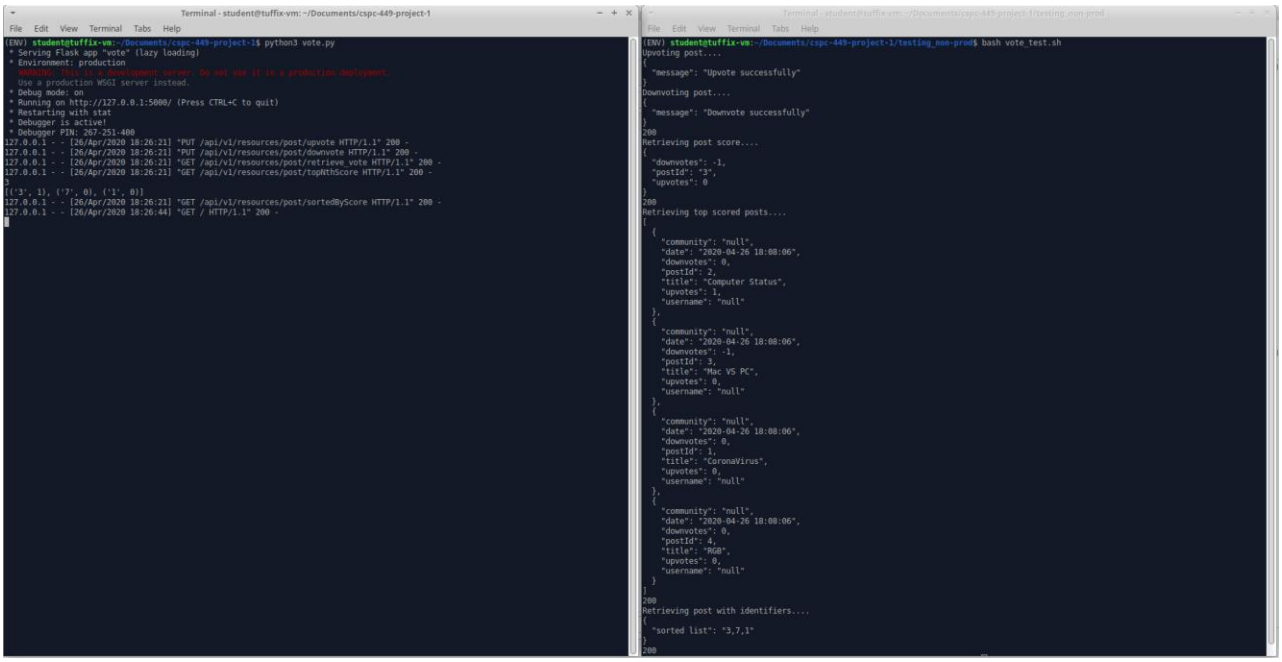
Check DB Browser to verify if the data was inserted into the database by selecting the "message_tbl" table.



Voting Microservice Testing:

Cancel the Message microservice (Ctrl + C) and run the Voting microservice in the same terminal:
$ python3 vote.py

On the other terminal run the Voting testing shell script:
$ bash vote_test.sh



Running the script tests the following API requests:

- Upvote a post
- Downvote a post
- Report the number of upvotes and downvotes for a post
- List the *n* top-scoring posts to any community
- Given a list of post identifiers, return the list sorted by score.

Check DB Browser to verify if the data was updated in the database by selecting the "userpost_tbl" table. The post scores will have been updated.

Testing (Production)

Production testing is done using a WSGI server called Gunicorn. Foreman is used to run a Procfile that manages the Gunicorn and Caddy processes for each microservice. 3 instances are created for each microservice. The Caddyfile proxies' direct requests for "http://localhost:2015/posts" to the Posts microservice, requests for "http://localhost:2015/users" to the Users microservice, http://localhost:2015/messages" to the Messages microservice, and http://localhost:2015/votes" to the Votes microservice. It also handles the load balancing method which in this case is round robin. The script tests are the same tests that were executed in non-production testing, besides the *test_all.sh* script.

Be sure to have a fresh database to view the successful HTTP status codes, delete existing database and run *python3 create_db.py,* otherwise you will be getting the conflict status codes.

Individual Microservice Testing:

Open 6 new terminals from the cspc-449-project-1 directory, be sure to activate the virtual environment for each terminal using the *$ source ENV/bin/activate* command.

On one terminal run Foreman for the Post microservice:
$ foreman start -m posts=3

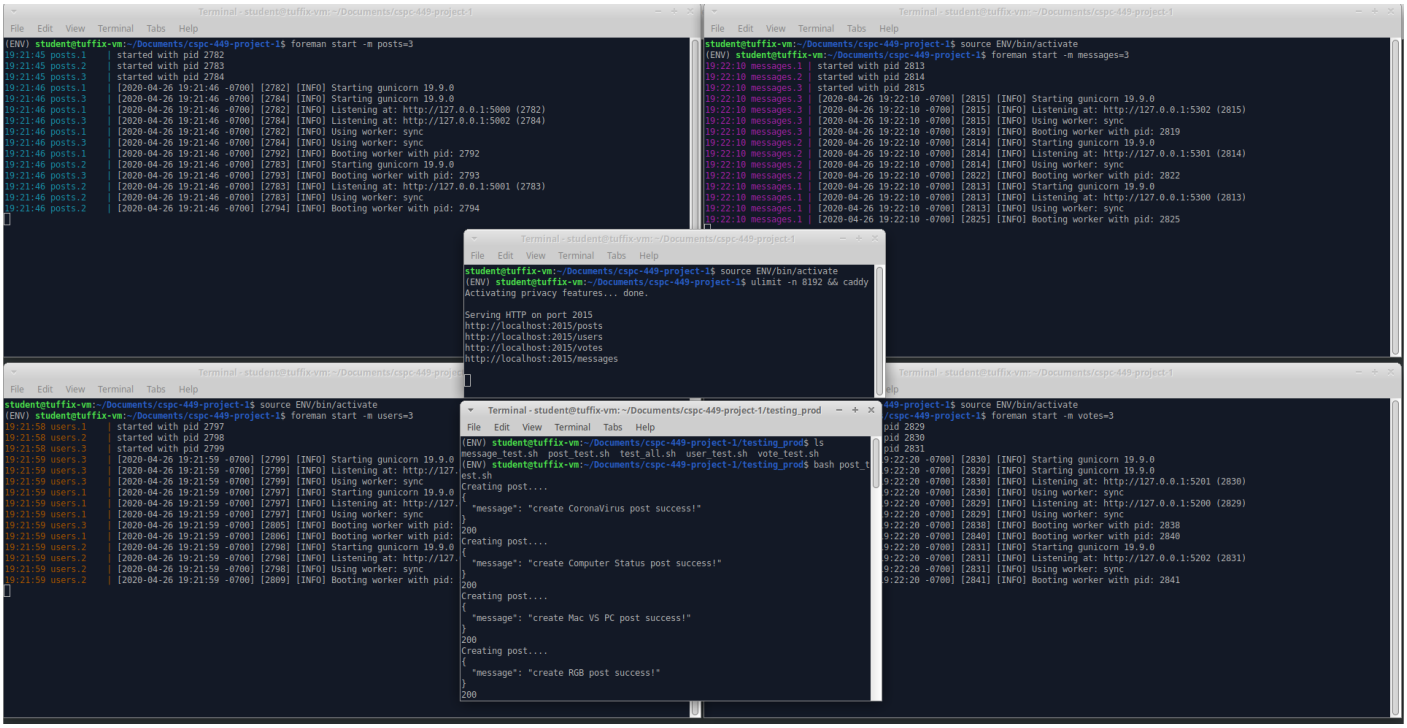
On the second terminal run Foreman for the User microservice:
$ foreman start -m users=3

On the third terminal run Foreman for the Message microservice:
$ foreman start -m messages=3

On the fourth terminal run Foreman for the Vote microservice:
$ foreman start -m votes=3

On the fifth terminal run the Caddyfile:
$ ulimit -n 8192 && caddy

On the sixth terminal change to the testing_prod directory and run the Post testing shell script:
$ cd testing_prod/
$ bash post_test.sh



Check DB Browser to verify if the data was inserted into the database.

On the sixth terminal run the User testing shell script:
$ bash user_test.sh
Check DB Browser to verify if the data was inserted into the database.

On the sixth terminal run the Message testing shell script:
$ bash message_test.sh
Check DB Browser to verify if the data was inserted into the database.

On the sixth terminal run the Vote testing shell script:
$ bash vote_test.sh
Check DB Browser to verify if the data was updated in the database.

All Microservice Testing:

On the sixth terminal run the shell script to test all the microservices:
$ bash test_all.sh
Check DB Browser to verify if the data was inserted into the database.

API Requests

CURL commands for User microservice (non-production)

- For production commands, change http://127.0.0.1:5000 → http://localhost:2015/users
- Ex: http://127.0.0.1:5000/api/v1/resources/users/create → http://localhost:2015/users/api/v1/resources/users/create

HTTP Method	CURL Command Example
POST	#Create user curl --location --request POST 'http://127.0.0.1:5000/api/v1/resources/users/create' \ --header 'Content-Type: application/json' \ --data-raw '{"email":"bob@gmail.com","username":"bob","karma":"0"}' \ --write-out '%{http_code}\n'
PUT	#Update email curl --location --request PUT 'http://127.0.0.1:5000/api/v1/resources/users/email' \ --header 'Content-Type: application/json' \ --data-raw '{"username":"bob","email":"b@gmail.com"}' \ --write-out '%{http_code}\n'
PUT	#Increment Karma curl --location --request PUT 'http://127.0.0.1:5000/api/v1/resources/users/inc' \ --header 'Content-Type: application/json' \ --data-raw '{"username":"bob"}' \ --write-out '%{http_code}\n'
PUT	#Decrement Karma curl --location --request PUT 'http://127.0.0.1:5000/api/v1/resources/users/dec' \ --header 'Content-Type: application/json' \ --data-raw '{"username":"bob"}' \ --write-out '%{http_code}\n'
DELETE	#Deactivate account curl --location --request DELETE 'http://127.0.0.1:5000/api/v1/resources/users/remove' \ --header 'Content-Type: application/json' \

	--data-raw '{"username":"bob"}' \ --write-out '%{http_code}\n'

CURL commands for Post microservice (non-production)

- For production commands, change http://127.0.0.1:5000 → http://localhost:2015/posts
- Ex: http://127.0.0.1:5000/api/v1/resources/post/create_post → http://localhost:2015/posts/api/v1/resources/post/create_post

HTTP Method	CURL Commands Example
POST	#Create a new post curl --location --request POST 'http://127.0.0.1:5000/api/v1/resources/post/create_post' \ --header 'Content-Type: application/json' \ --data-raw '{"community": "Corona-19", "text": "Things are going better!", "title": "CoronaVirus", "url": "null", "username": "Zexin"}' \ --write-out '%{http_code}\n'
DELETE	#Delete an existing post curl --location --request DELETE 'http://127.0.0.1:5000/api/v1/resources/post/delete_post' \ --header 'Content-Type: application/json' \ --data-raw '{"postId": "6"}' \ --write-out '%{http_code}\n'
GET	# Retrieves a post based on the id of a post curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/post/retrieve_post' \ --header 'Content-Type: application/json' \ --data-raw '{"postId": "3"}' \ --write-out '%{http_code}\n'
GET	#List the n most recent posts to a particular community curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/post/listNthToACommunity' \ --header 'Content-Type: application/json' \ --data-raw '{"nth": 2,"community": "Computer"}' \ --write-out '%{http_code}\n'
GET	#List the n most recent posts to any community curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/post/listNthToAny' \ --header 'Content-Type: application/json' \ --data-raw '{"nth": 5}' \ --write-out '%{http_code}\n'

CURL commands for Message microservice (non-production)

- For production commands, change http://127.0.0.1:5000 → http://localhost:2015/messages
- Ex: http://127.0.0.1:5000/api/v1/resources/post/message → http://localhost:2015/messages/api/v1/resources/post/message

HTTP Method	CURL Commands Example
POST	#send a message curl --location --request POST 'http://127.0.0.1:5000/api/v1/resources/message' \ --header 'Content-Type: application/json' \ --data-raw '{"userto":"alice","userfrom":"bob","messagecontents":"hello alice :)", "messageflag":"favorite"}' \ --write-out '%{http_code}\n'
GET	#show favorite messages curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/message/favorite' \ --header 'Content-Type: application/json' \ --data-raw '{"messageflag":"favorite"}' \ --write-out '%{http_code}\n'
DELETE	#delete message curl --location --request DELETE 'http://127.0.0.1:5000/api/v1/resources/message/delete' \ --header 'Content-Type: application/json' \ --data-raw '{"messageid":"1"}' \ --write-out '%{http_code}\n'

CURL commands for Vote microservice (non-production)

- For production commands, change http://127.0.0.1:5000 → http://localhost:2015/votes
- Ex: http://127.0.0.1:5000/api/v1/resources/post/ upvote → http://localhost:2015/votes/api/v1/resources/post/ upvote

HTTP Method	CURL Commands Example
PUT	# Upvote a post curl --location --request PUT 'http://127.0.0.1:5000/api/v1/resources/post/upvote' \ --header 'Content-Type: application/json' \ --header 'Content-Type: application/json' \ --data-raw '{"postId": "2"}'
PUT	# Downvote a post curl --location --request PUT 'http://127.0.0.1:5000/api/v1/resources/post/downvote' \ --header 'Content-Type: application/json' \ --data-raw '{"postId": "3"}' \ --write-out '%{http_code}\n'
GET	# Report the number of upvotes and downvotes for a post curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/post/retrieve_vote' \ --header 'Content-Type: application/json' \ --data-raw '{"postId": "3"}' \ --write-out '%{http_code}\n'
GET	# List the n top-scoring posts to any community curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/post/topNthScore' \

	--header 'Content-Type: application/json' \ --data-raw '{"nth": 4}' \ --write-out '%{http_code}\n'
GET	# Given a list of post identifiers, return the list sorted by score. curl --location --request GET 'http://127.0.0.1:5000/api/v1/resources/post/sortedByScore' \ --header 'Content-Type: application/json' \ --data-raw '{"list":"1,3,7"}' \ --write-out '%{http_code}\n'

Supplemental Issues

Virtual Machine Issues in Setup

Tuffix is known to have some errors when trying to download, update, or install many packages at a given time. If you are having locking issues when trying to install packages/dependences in the virtual environment, follow the instructions below.

See which processes are running:

$ ps aux | grep -I apt

```
student@tuffix-vm:~/Desktop/CPSC449$ ps aux | grep -i apt
root         525  0.0  0.0   4628   816 ?        Ss   21:23   0:00 /bin/sh /usr/lib/apt/apt.systemd.daily install
root         560  0.0  0.0   4628  1684 ?        S    21:23   0:00 /bin/sh /usr/lib/apt/apt.systemd.daily lock_is_held install
student     1656  0.0  0.0  22000  1148 pts/0    S+   21:29   0:00 grep --color=auto -i apt
```

Kill all root processes:

$ sudo kill <PID>

```
student@tuffix-vm:~/Desktop/CPSC449$ sudo kill 525
student@tuffix-vm:~/Desktop/CPSC449$ sudo kill 560
```

To kill the remaining process, we must identify the real PID of the process in the lock-frontend folder

$ sudo lsof /var/lib/dpkg/lock-frontend

```
student@tuffix-vm:~/Desktop/CPSC449$ sudo lsof /var/lib/dpkg/lock-frontend
lsof: WARNING: can't stat() fuse.gvfsd-fuse file system /run/user/1000/gvfs
      Output information may be incomplete.
COMMAND    PID USER   FD   TYPE DEVICE SIZE/OFF   NODE NAME
unattende 1010 root    5uW  REG    8,1        0 264117 /var/lib/dpkg/lock-frontend
```

Kill all the remaining process with PID you discovered from the previous command

$ sudo kill <PID>

Everything should be good to go. Install virtual environment with this command

$ sudo apt-get install python3-venv