Project 2 - Section 01
Group 5: Bryan Mendoza-Trejo, Eduardo Gomez, Cristian Rodriguez

Project Description

This project focuses on creating a service(s) such as Twitter's microblogging. We have two main python files (users.py & timelines.py). We define users as objects that have a username, password, email and bio. Users can follow other users and repost posts. Posts are an action users are allowed to do which include text/messages. Posts include details such as what original post owner/username, the actual content of the post, and when it was posted. If a post is reposted it would include the path/URL of the original post and the new username. The project includes 3 types of timelines (home, user, public) which share similarities. We use Hug to create HTTP REST APIs. This framework enables calls such as POST/GET methods for users.py and timelines.py. The authentication is HTTP Basic access authentication that guarantees only authenticated users are able to create new posts. It also guarantees that the home timeline is only accessible to themselves. Database creation is handled with the sqlite-utils library where a bash script creates the database from the csv files. The profile initializes WSGI and gunicorn to synchronize and allow for foreman to run both REST APIs in parallel. The load balancer handles multiple instances when more than one timelines are running. This is done with configuring the haproxy.cfg file. Our port for users starts at 5000 while our timelines port starts at 5100 and increases by 1 (5101,5102,...) for each instance that is running. By binding to port 80 we are able to route the different timelines running and balance them using roundrobin.

Load balancing is handled with HAProxy. This file is located in the main project2 directory titled "haproxy.cfg". We recommended adding this file to the /etc/haproxy directory to include load balancing on all (users and timelines) microservices.

Also included, is a sql script creating the users and followers schema that we used to model our database approach titled "users.sql".

To populate test the creation of new users onto the database, we created a JSON file that includes all the necessary information for a new user. The "./bin/post.sh ./share/users.json" command creates a new dummy user.

How to create the databases and start the services

1. Run **./bin/init.sh** on the project terminal to create the database.
2. **Foreman start --formation users=1,timelines=3** to run our microservices (users and timelines).

Note: To include the haproxy file you must follow these steps:

On the project directory terminal run:

1. sudo cp -f haproxy.cfg /etc/haproxy/haproxy.cfg
2. sudo systemctl restart haproxy

Command to view the file: sudo nano /etc/haproxy/haproxy.cfg

Users.py

The users.py program utilizes the hug API framework to create custom REST API directives. We also use the sqlite-utils library to load our users and followers databases. Note: Our HAProxy server is configured to host the users REST API on port 5000 (foreman start default).

users()

Get All Users information including username, password, bio, email, and id from the users database.

Example:

```
http GET localhost:5000/users/
```

verifyUser()

Verifies the users information including username and password from the users database. Utilized mainly for timelines.py exists function.

Example:

```
http GET
localhost:5000/users/verify?username=”chris”&password=
”password2”
```

createUser(username, bio, email, password)

Create a new user. The API asks for the username, bio, email, and password of the new user.

Example:

```
http POST localhost:5000/users/new?username=”John”
bio=”I like pie!” email=”John123@gmail.com”
password=”notpassword4”
```

addFollower(username, friend_username)

Adds a new follower. The API asks for the username, and the friend username you’d like to follow.

Example:

```
http POST localhost:5000/followers/new?username=”John”
friend_username=”steve”
```

removeFollower(username, friend_username)

Deletes/removes a follower from the followers database. The API asks for the username, and the friend username you’d like to unfollow.

Example:

```
http DELETE localhost:5000/unfollow?username=”John”
friend_username=”steve”
```

timelines.py

The timelines.py program utilizes the hug API framework to create custom REST API directives. We also use the sqlite-utils library to load our users and posts databases. Note: Our HAProxy server is configured to host the users REST API on port 80 (localhost).

exists(username, password)

Fetches the user inputed authentication to verify if that username and password match to the users database. We do this by utilizing the requests library to request the user.py verifyUser API to verify that the username and password exists in the database, returning a 200 OK status code. If the user is not found or doesn’t exist, the appropriate status code is returned, not verifying the user.

addPost(message)

Creates a new post. The API asks for authentication, and if authenticated then asks for a message to post and returns the username, message, and timestamp of when the post was created.

Example:

```
http POST localhost:80/posts/new?message=”TestMessage”
--auth steve:password2
```

rePost(id)

Creates a repost. The API asks for authentication, and if authenticated then asks for the id number of the post they would like to repost and retrieves the message from that id and returns the username, message, timestamp, and url to the original post.

Example:

```
http POST localhost:80/repost?id=1 --auth
steve:password2
```

getPublicTimeline()

Gets all posts information including username, message, timestamp, repost url link, and id from the posts database.

Example:

```
http GET localhost:80/allposts/
```

getUserTimeline(username)

Gets all posts information from a specified user and and returns username, message, timestamp, repost url link, and id from the posts database.

Example:

```
http GET localhost:80/users/steve
```

getHomeTimeline()

The API asks for authentication, and if authenticated then gets all posts information from users followers and and returns followers_username, message, timestamp, repost url link, and id from the posts database. We call the users.py followers request to get all the users followers. Then we traverse the list and grab each post from the users followers.

Example:

```
http GET localhost:80/posts/home --auth
steve:password2
```