IEMS5722 Assignment 3: Developing a Server Application

The revised environment setup was recently prepared by Mengjie Chen and Xiao Yi.

Please contact **Xiao Yi** at yx019@ie.cuhk.edu.hk, if you have problem setting up the environment.

Submit to Blackboard by **19:00, Monday, March 15, 2021**

Notes:

1. Read carefully the instructions and requirements to understand what you have to do.
2. Follow the instructions to submit your files for marking.
3. Late submissions will receive **30% mark penalty**.
4. This assignment accounts for **12%** of your final grade.

1. Objectives

- To learn how to develop a server-side application using Python and Flask
- To learn how to develop APIs for mobile apps
- To learn how to use HTTP for communication between server and client
- To learn how to deploy an HTTP server-side application using Nginx, Gunicorn and Supervisor

2. Instructions

In this assignment, you will try to develop the **server-side application** for the mobile app developed in Assignment 2. You should setup a server using **Nginx** as the Web server, develop the server application using **Python** and **Flask**, and deploy the application using **Gunicorn** and **Supervisor**. The API you develop in this assignment should be the same as the ones provided in Assignment 2. Hence, the app developed in Assignment 2 should work with the server application developed in this assignment without any modification except the URL of the APIs.

2.1 Setting up the Server

You are strongly advised to develop your server application on a cloud platform such as **Amazon AWS** or **Google Cloud Compute Engine**. Both of these services offer some forms of free trials. Whenever possible, choose Ubuntu Linux as the operating system. All the following instruction assumes that your server is running Ubuntu 18.04.

In this assignment, you will have to use the following software applications: Nginx Web server, Gunicorn WSGI server, Supervisor, MySQL Database and various Python modules. Use the following commands to install these packages first.

Before we start our tasks, we can firstly deploy a specific virtual environment for our app with python-virtualenv. Thus, we also need to install the package python-virtualenv.

To install **mysql**, **nginx** and **python-virtualenv**, you can use the following commands:

```
sudo apt-get update
sudo apt-get install python-virtualenv
sudo apt-get install mysql-server
sudo apt-get install nginx
```

2.2 Developing Server Application

First of all, develop your server application and implement the APIs for the mobile app using Python. Python is by default installed in Ubuntu. To check whether you have Python 3 installed:

```
$ python3
Python  3.6.9  (default,  Nov 7 2019, 10:44:02) [GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

If you see the interactive shell of Python 3 coming up, that means Python 3 is available.

```
>>> exit() # quit the interactive shell
```

We recommend to use an isolated Python environment rather than the default one. The following commands can set up a virtual Python 3 environment:

```
virtualenv -p python3 ~/python3_venv # setup a python3 virtual environment
source ~/python3_venv/bin/activate # activate the python3 virtual environment
# if you want to exit this virtual environment, you can input 'deactivate'
```

Then, we create a new directory to hold our app files and enter in this directory.

```
mkdir ~/iems5722/assignment_3
cd ~/iems5722/assignment_3
```

Now, we need to install the other packages (gunicorn, superviosr, mysql-connector and Flask) with pip.

```
pip install gunicorn
pip install supervisor
pip install mysql-connector
pip install Flask
```

2.2.1 Database

To support the functions to be developed, you will need to have a database that can help you store the data (e.g. messages from the users) and let you retrieve the data in the future. In this assignment, you should use the **MySQL** database. During installation, you should have set up an administration account. You can check whether you have successfully installed MySQL by invoking the command line interface:

```
$ sudo mysql -u root -p
Enter password:
Welcome  to the MySQL monitor.  Commands end with ; or \g. Your MySQL
connection id is 14807
Server version: 5.7.29-0ubuntu0.18.04.1 (Ubuntu)
Copyright (c) 2000, 2020, Oracle and/or its affiliates. All rights reserved.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates.
Other names may be trademarks of their respective owners.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
mysql>
```

If you can enter the command line client successfully, this means that MySQL has successfully installed. As root of the database, create and grant privileges to a new user. Choose your own user name ("**dbuser**" is used here) and password ("**password**" is used here).

```
mysql>  CREATE  USER  'dbuser'@'localhost'  IDENTIFIED  BY  'password';
    Query OK, 0 rows affected (0.01 sec)
mysql>  GRANT ALL PRIVILEGES ON *.* TO 'dbuser'@'localhost';
    Query OK, 0 rows affected (0.00 sec)
mysql>  exit
    Bye
```

To support the server application, you should first create a database for your application. Log in as the new user, and create a database called **"iems5722"**.

```
$ sudo mysql -u dbuser -p
```

```
mysql>  CREATE DATABASE iems5722;
    Query OK, 1 row affected (0.00 sec)
mysql>  USE iems5722;
    Database changed
```

You will then have to create some tables to store the data. You can create your own database schema if you like, otherwise you can use the following queries to create two tables for use in this assignment.

```
CREATE TABLE chatrooms (
    id  INT  NOT  NULL  AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    PRIMARY KEY (id)
) DEFAULT CHARSET = utf8

CREATE TABLE messages (
    id INT NOT NULL AUTO_INCREMENT,
    chatroom_id INT NOT NULL, user_id INT NOT NULL,
    name VARCHAR(20) NOT NULL, message VARCHAR(200) NOT NULL,
    message_time TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    FOREIGN KEY (chatroom_id) REFERENCES chatrooms(id)
) DEFAULT CHARSET = utf8
```

Go on to populate the “chatrooms” and “messages” tables by using the INSERT statements. You are advised to create 1–3 chatrooms, and 10–20 messages for any one of the chatrooms. Once you are done with creating the database and the tables, you can now focus on the development of the APIs.

2.2.2 APIs

You will have to implement the following three APIs in your server application. All responses should be in JSON format.

API	**GET /api/a3/get_chatrooms**
Descriptions	For retrieving a list of chatrooms from the server
Input Parameters	No input parameter is required
Example	/api/a3/get_chatrooms

Sample Output	{
	 "status": "OK",
	 "data": [
	 {
	 "id": 1,
	 "name": "generalRoom"
	 },
	 {
	 "id": 2,
	 "name": "Room IEMS5722"
	 }
	]
	 }

API	GET /api/a3/get_messages
Descriptions	For retrieving a list of messages in a specific chatroom (maximum 5 messages in one page)
Input Parameters	· chatroom_id (the ID of the chatroom) · page (the page number of the list of messages)
Example	/api/a3/get_messages?chatroom_id=1&page=1

Sample Output	<pre>{
	  "status": "OK",
	  "data": {
	  "current_page": 1,
	  "messages":[
	  {
	  "id": "1",
	  "chatroom_id": "1",
	  "user_id": 1234567890,
	  "name": "Alice",
	  "message": "Hi from IEMS5722",
	  "message_time": "2021-02-06 21:40:15"
	  }
	  ],
	  "total_pages": 1
	  }
	}</pre>

API	POST /api/a3/send_message
Descriptions	For sending a message in a specific chatroom
Input Parameters	· chatroom_id (the ID of the chatroom) · user_id (the unique user ID, use your student ID starting with 1) · name (the displayed name of the user) · message (the message input by the user)
Example	POST the following to the API: chatroom_id=1&user_id=1234567891&name=Bob&message=hi

Sample Output	<pre>{
	  "status": "OK"
	}</pre>

NOTE

The last API, i.e., the HTTP POST request, should be defined to accept form-data in the request body. Explicitly speaking, in Flask, you should retrieve request data by `request.form`, e.g., `request.form['chatroom_id']`.

All responses of your API **must follow** exactly the specified format. Otherwise the app you developed in Assignment 2 will not work. In addition, your application should check if the client side has supplied the correct parameters.

If any parameter is missing, you should return the following response instead:

```
{
  "message": "<error message>",
  "status": "ERROR"
}
```

2.3 Testing Your Application

Once you are done with the development of the server application, you can first test your application by directly executing the application (make sure that you have enabled debug mode, and allow access from anywhere in your Flask application). For example (your Python file name may be different):

```
$ python3 iems5722_a3.py
* Serving Flask app "iems5722_a3" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production
deployment.
Use a production WSGI server instead.
* Debug mode: on
* Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
* Restarting with stat
* Debugger is active!
* Debugger PIN: 1**-2**-3**
```

Then you can access port 5000 (or the port you have set) from your browser to test your application. For example, if the server's IP address is 18.220.14.97, the URL will be http://18.220.14.97:5000/{api_path}. By default, your server will block all the ports except port 22 (for SSH connection). Make sure you have configured the firewall of your server such

that it allows traffic through the port(s) you specified. For Amazon AWS, please refer to this link:
Authorize access to an instance

After you have tested your Flask application, ideally you should turn off debug mode and allow access to localhost only.

2.4 Deploying Your Application

If you have tested your application and see that it is ready to be deployed, then you should follow the steps below to deploy your application. When deploying an application, we would like to make sure that it is monitored, and when it crashes or stopped for some reasons, it will be automatically restarted, so that the client will not see an interruption of services for a long time. As discussed in lecture, we will use **Gunicorn** to deploy your Flask application, and then use **Supervisor** to

monitor the **Gunicorn** process. Finally, we have **Nginx** as the Web server to redirect requests to **Gunicorn** if the requests are for your app.

2.4.1 Gunicorn and Supervisor

You will use Gunicorn to host your Flask application. In the directory in which you have created the server application, you can use the following command to ask Gunicorn to run your application on port 8000 (Gunicorn listens at this port for incoming requests, and will route these requests to the functions in your application depending on the path requested).

```
$ gunicorn -b localhost:8000 -w 3 iems5722_a3:app # iems5722_a3 is your app
name
[2020-03-17 09:38:50 +0800] [1665] [INFO] Starting gunicorn 20.0.4
[2020-03-17 09:38:50 +0800] [1665] [INFO] Listening at: http://127.0.0.1:8000
(1665)
[2020-03-17 09:38:50 +0800] [1665]  [INFO]  Using worker: sync
[2020-03-17 09:38:50 +0800] [1668]  [INFO]  Booting worker with pid:  1668
[2020-03-17 09:38:50 +0800] [1669]  [INFO]  Booting worker with pid:  1669
[2020-03-17 09:38:50 +0800] [1670]  [INFO]  Booting worker with pid:  1670
```

If you see something like above, then you have successfully used Gunicorn to host your application. Note that the API is now served to localhost:8000 only. If you want to check the deployment from the browser, you can change the “localhost” in the command into “0.0.0.0” and allow access via port 8000 in your server firewall.

If you meet the **module import error** while executing the above command,it may be because you run **gunicorn** in the wrong python environment. You can try to fix the problem by uninstalling **gunicorn** outside your python environment and reinstalling the **gunicorn** in your python virtual environment.

When your system is in production, you would like to have it monitored by some other processes, such that it can be restarted automatically after it has crashed, or has been killed by the operating system for some reasons. This is where Supervisor comes into the scene. To put it in a simple way, instead of directly running Gunicorn, we ask Supervisor to help us execute and monitor Gunicorn to host our Flask application.

To use Supervisor, we need to create a configuration file. Firstly, you could echo a template configuration file locally.

```
echo_supervisord_conf > supervisord.conf
```

Then, you need to modify some lines of configuration in this file "supervisord.conf"

```
nano supervisord.conf
```

In "supervisord.conf", there is a line stated as following:

```
[program:theprogramname]
```

You can insert a ";" in front of this line to set it as a comment and add the following lines before this line ("iems5722_a3" is the name of your program):

```
[program:iems5722_a3]
command  =  gunicorn -b localhost:8000 -w 3 iems5722_a3:app
directory = PATH-TO-iems5722_a3
user = ubuntu
autostart = true
autorestart = true
stdout_logfile = PATH-TO-LOG-FILE # example: /home/ubuntu/iems5722_a3/app.log
redirect_stderr = true
```

For detailed descriptions of the commands and parameters, refer to the documentation of Supervisor . Then you can start supervisor and start running your program with the following commands.

```
supervisord # start supervisor
supervisorctl start iems5722_a3 # start the program
```

[Supplementary] The following are some common commands for supervisor:

```
supervisorctl status # check the status of programs in supervisor
supervisorctl stop all # stop all programs in supervisor
supervisorctl start PROGRAM_NAME # start the specified program
```

2.4.2 Nginx

To configure Nginx, create a configuration file named “iems5722_web.conf”. The following is an example that would probably work in your case (replace the value of “**server_name**” with your server’s public IP).

```
nano iems5722_web.conf
```

```
server {
    listen 80;
    listen [::]:80;
    server_name IP_ADDRESS; # your ip address

    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header  X-Forwarded-For  $proxy_add_x_forwarded_for;
        proxy_set_header  X-Real-IP  $remote_addr;
        proxy_set_header Host $http_host;
        proxy_http_version 1.1;
        proxy_redirect off;
        proxy_buffering off;
    }
}
```

The configuration file asks Nginx to forward any request to port 8000 of localhost, where Gunicorn is listening. The other parameters simply set the headers when Nginx is redirecting the HTTP requests. For details of these parameters, you can refer to the documentation of Nginx.

Copy the configuration file to the protected folder "/etc/nginx/sites-enabled/" and restart Nginx.

```
sudo cp PATH-TO-iems5722_web.conf /etc/nginx/sites-enabled/
sudo service nginx restart
```

Everything should be ready now. Remember to allow access via port 80 in your server firewall. Then you should be able to access the APIs by accessing URLs such as `http://{your_server_ip}/api/a3/get_chatrooms`.

3. Requirements

The server application you developed should have the following features:

- A server application written in Python using the Flask framework
- The server application should implement the three APIs to support the instant messaging app
    - API 1: GET: /api/a3/get_chatrooms

      Returns a list of chatrooms available (you should at least create one chatroom in your database).
    - API 2: GET: /api/a3/get_messages

      Returns the list of messages in a particular chatroom, sorted in reverse chronological order (the latest message comes first), and implements the paging mechanism as shown in the example.
    - API 3: POST: /api/a3/send_message

Allows the app to submit a new message to a chatroom, which should be inserted into the database.

4. Guidelines

4.1 Testing and Debugging

To making testing and debugging easier, make sure you ensure correctness first in MySQL, then Python and Flask, then Gunicorn, Supervisor and Nginx. If there are any changes upstream, make sure you update and restart all the related downstream services.

4.2 Hints on MySQL Statements

To sort the database result set in reverse chronological order, you can use the keywords **"ORDER BY"** and **"DESC"**. Think about which column to sort by.

To obtain only a limited number instead of all messages, you can use the **"LIMIT"** keyword. To find out the total number of messages (in a chatroom), you can use **"COUNT"**. To find out the number of pages, you need to calculate using the total number of messages, and number of messages per page (i.e. 5).

You can refer to the following link for syntax and examples of SQL statements. https://www.w3schools.com/sql/

5. Submission

To facilitate marking of the assignments, you should strictly follow the instructions below: To submit your assignment, create a folder name <your_student_id>_a3. In the folder, you should include the following items:

- All source codes of your **Python application**
- A text file containing the **full URLs** of the three APIs

Compress this folder using ZIP, you should now have a file named **<your_student_id>_a3.zip**. Submit it to Blackboard (https://blackboard.cuhk.edu.hk/).

6. Grading Scheme (Total 100 Marks)

- **(30 marks)** The /api/a3/get_chatrooms API correctly queries the database, retrieves the list of chat rooms, and returns a JSON response in the specific format.
- **(30 marks)** The /api/a3/get_messages API correctly queries the database, retrieves the list of messages (of a particular page), and returns a JSON response in the specific format.
- **(40 marks)** The /api/a3/send_message API correctly receives the parameters from the client, inserts a new record into the database, and returns a JSON response in the specific format.