CS 2510: Assignment 1

-Ashwini Ananda

I have implemented the group chat using Python and gRPC library. The system implements a bidirectional gRPC streaming where the type of calls is response-stream type.

gRPC is a modern open-source high performance Remote Procedure Call (RPC) framework that can run in any environment. It can efficiently connect services in and across data centers with pluggable support for load balancing, tracing, health checking and authentication.

gRPC is combination of google protobufs and RPC. Google protobufs is a neutral platform and language structure created by Google for serializing structured data. It doesn't matter what programming language is used with it. It always gives the same output. This also enables quick message transfer. A .proto file has all the info about the protocol written in the protobuf language. After compiling that file with grpcio-tools, you get *_pb2.py and *_pb2_grpc.py files.

gRPC uses the CompletionQueue API for asynchronous operations. The completion queue is bound to an RPC call. To use an asynchronous client to call a remote method, we first create a channel and stub. Once we have a stub running, initiate the RPC. The server implementation requests an RPC call with a tag and then waits for the completion queue to return the tag.

There is a gRPC specific service that will provide a service to use that has the given RPC calls implemented. Each client opens a new connection and waits for the server to send messages. The server is open and can keep sending messages. The server can handle incoming messages from clients so as to broadcast this message to all the other people in the chatroom. Multiple clients/users can communicate with each other in the chatroom via multi-threading. Each user is runs on a separate thread as the main/UI thread.

I have used Tkinter, a standard Python interface to the Tk GUI toolkit. This gives out the chat UI. First the user enters their name. Then the Tkinter chat UI pops up. Thereafter, all "unread" messages are downloaded, and the user can join the conversation as well.

To run the project, in the app's directory, first install all necessary packages in the requirements document.

```
pip install reqs.txt
```

Once all the dependencies are successfully installed, one can run the server and client(s).

```
python chat_server.py 1,2,3 50001

python client.py 1 localhost 50001
```

This equips the chatroom up to a maximum of 3 clients who can successfully communicate with each other.

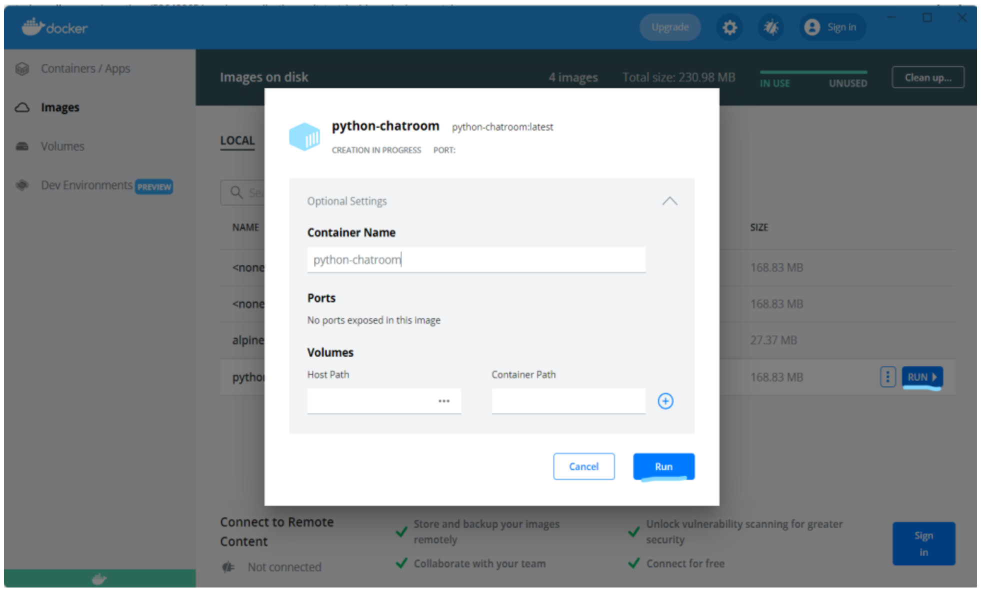
Scope for improvement in this project includes a better Tkinter interface. Currently a very simple and basic UI has been implemented. Scroll bar, buttons, overall better UI can all be included. There might also be a better way of implementing the assignment according to coding standards. I have utilized user-defined functions but there might be a more polished way.

Testing Appendix

The submission includes a `Dockerfile`. Build and run with docker commands.

```
docker build -t python-chatroom .
```

```
docker run python-chatroom
```



python-chatroom python-chatroom:latest

CREATION IN PROGRESS PORT:

Optional Settings

Container Name: python-chatroom

Ports: No ports exposed in this image

Volumes

Host Path

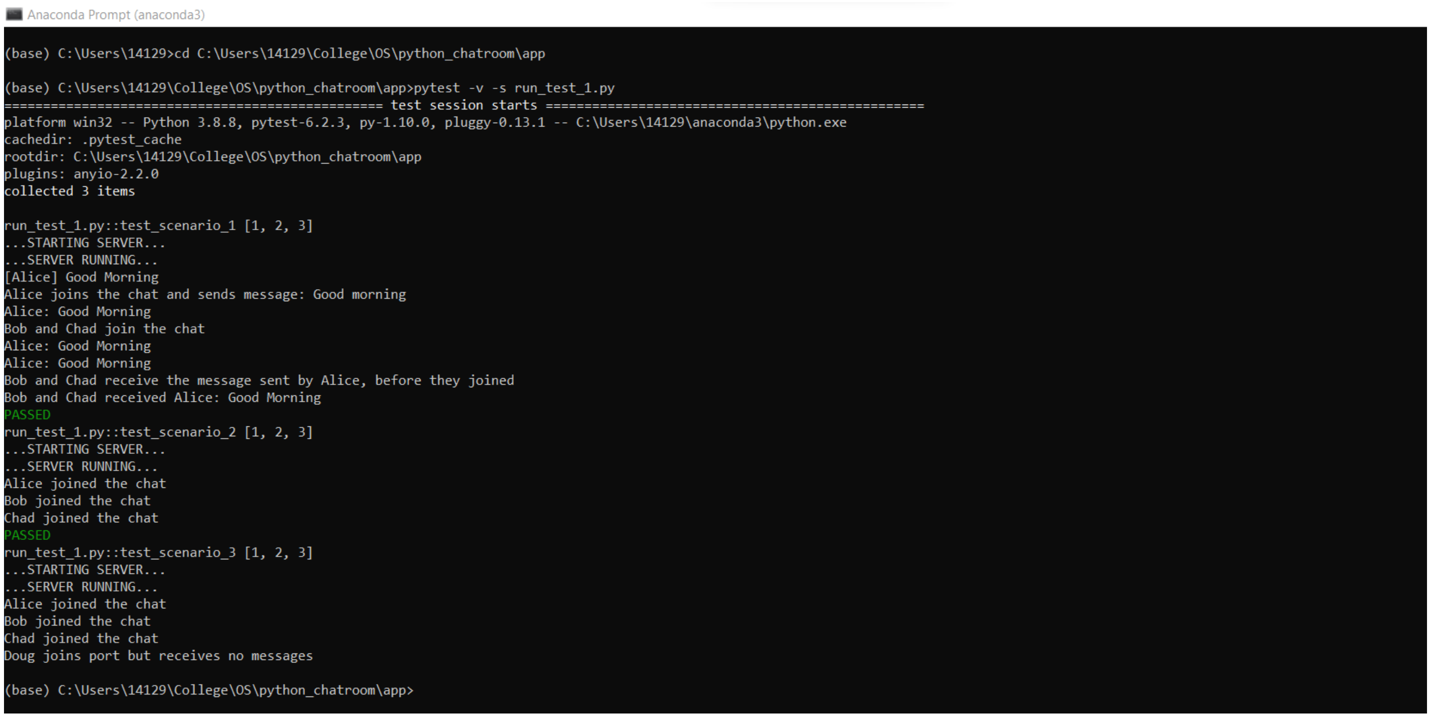
Container Path

Make sure the container is running before running test cases.

```
pytest -v -s app/run_tests.py
```

Unfortunately, this does not run from within the docker file itself and needs us to run pytest from command line. The issue is with using Tkinter as it does not get installed when python is compiled. This is a known issue and could be rectified in the further scope of this project. Detailed logs and screenshots can be found in the logs folder.

The `run_tests.py` houses a few test cases. It spawns the server and client processes implicitly, tries joining the server and sending messages. It also tries joining the server at delays and downloading unread messages. Finally, it also checks exceeding the maximum server limit of threads.



Anaconda Prompt (anaconda3)

```
(base) C:\Users\14129>cd C:\Users\14129\College\OS\python_chatroom\app

(base) C:\Users\14129\College\OS\python_chatroom\app>pytest -v -s run_test_1.py
================================================= test session starts =================================================
platform win32 -- Python 3.8.8, pytest-6.2.3, py-1.10.0, pluggy-0.13.1 -- C:\Users\14129\anaconda3\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\14129\College\OS\python_chatroom\app
plugins: anyio-2.2.0
collected 3 items

run_test_1.py::test_scenario_1 [1, 2, 3]
...STARTING SERVER...
...SERVER RUNNING...
[Alice] Good Morning
Alice joins the chat and sends message: Good morning
Alice: Good Morning
Bob and Chad join the chat
Alice: Good Morning
Alice: Good Morning
Bob and Chad receive the message sent by Alice, before they joined
Bob and Chad received Alice: Good Morning
PASSED
run_test_1.py::test_scenario_2 [1, 2, 3]
...STARTING SERVER...
...SERVER RUNNING...
Alice joined the chat
Bob joined the chat
Chad joined the chat
PASSED
run_test_1.py::test_scenario_3 [1, 2, 3]
...STARTING SERVER...
...SERVER RUNNING...
Alice joined the chat
Bob joined the chat
Chad joined the chat
Doug joins port but receives no messages

(base) C:\Users\14129\College\OS\python_chatroom\app>
```

Reference:

1. https://grpc.io/docs/languages/cpp/async/
2. https://developers.google.com/protocol-buffers/docs/pythontutorial
3. https://www.pythontutorial.net/tkinter/tkinter-pack/
4. https://github.com/grpc/grpc/tree/v1.43.0/examples/python/helloworld
5. https://blog.codefuture.dev/goLang-chat-server
6. https://www.youtube.com/watch?v=bi0cKgmRuiA