Design document for Task1

the "design document.pdf"

Source code for Task 2 and Task3

Source code is in **CMC.zip**, please unzip it, and README.md instruct how to install required package and run the code.
test case is in "test-case.xlsx"

Response sentence pool data file for Task 2

the "data/response.csv" file in source code folder

Three test sessions to simulate three friends

- screen shot for three sessions: "ScreenShot.png"
- conversation log files from the CMC server: "log*2023*07_30.log"
- output report of Task3: report file is "report/report.txt"

Project document

whether have finished all tasks:

For all tasks:

- Task1: Fininshed.
- Task2: Finished. For test driver ideally it should be a shell script, but I'm not very familar with shell, to save time used python to develop it. Can use `alias talk-to-george='python3 test_driver.py'` to make the script name more meaningful.
- Bonus 1: No
- Bonus 2: Finished, input `fleshdata` in test driver to trigger CMC service saving latest memory data into file, then can visit file content.

- Bonus 3: Finished, input `inspect` in test driver, then choose one option to inspoect the memory data structure.
- Task 3: Finished, run `python3 report_program.py` to generate report. Report will be printed in console and written into report/report.txt
- Bonus4: Finishd, additional output the like/unlike pairs for every friend.

layout of submission

the submission contains:

- **CMC.zip** (source code)
- **design document.pdf** (the design document)
- **project document.pdf** (the project document)
- **ScreenShot.png** (screen shot for three sessions)
- **system-diagram.pmg** (system diagram picture)
- **test-case.xlsx** (test case file)

and the source code contains:

- **data** (folder which include data structure file):
  - **interation.json** (record the friend interaction history data)
  - **message.csv** (message pool file, will add into it when friend send new message)
  - **pair_data.csv** (record pair relationship for every message and every friend)
  - **response.csv** (response pool file)
- **protocol_schema** (protocol schema folder):
  - **CMC.postman_collection.json** (the request collection schema of Postman)
  - **protocol_example.json** (example of every API)
- **report** (report folder)
  - **report.txt** (report result of report program)
- **app.py** (service entrance file, define the API interface by using flask)
- **CMC_service.py** (core service, implement algorithm)
- **log.py** (log handler file, to generate log)
- **log_{date}.log** (generated log file, every day will generate a new log file)
- **README.md** (help file, instruct how to install and run code)
- **report_program.py** (implement report program)
- **requirements.txt** (pip requirements file to manage dependency package)
- **test_driver.py** (implement test driver)

explain how to run

follow the README.md in source code

TODO items

1. **Login**: add user login and authentication feature.
2. **Match algorithm**: now I just use simple string equal method to search matched sentence, which is not efficient and intelligent. This should be improved, try to analyse natural language and use more effective fuzzy matching algorithm.
3. **Request schema validation**: for request schema validataion in app.py, should use schema template to validate request format instead of checking every field type.
4. **Performance**: I didn't do performance testing for service, some logic may be low speed and cost much time when comming high requests rate, especiall for file I/O operation in save_data method. Perhaps better solution is to import async method or aiohttp to handle time consuming logic, and import relevant message queue component to limit coming rate and make sure not lost message.
5. **Storage**: now all data are stored in file, and service load all of them into memory. Obviously this is not a good solution, when data become bigger it will cost all server memory. Common solution is to store them into database, because mesage and response pool are document structure and not updated frequently, MongoDB may be suitable for them. And for pair data and interaction data, because they are like key-value structure and changing all the time, maybe can use Redis to store them.
6. **Clean Code**: there is alway space to clean code, because of time limition some code are not elegant, such as string or dict hard code, a little duplicate code, not pretty log or output, can be refactored more.
7. **Supplement message and response pool** there are just a few sentences in message and response pool, can continuously add more data and use relevant machine learning algorightm to train CMC. I'm interested with that and studying relevant knowledge and framework, such as Tensorflow.

How spend time

This assignment takes me about 25 - 30 hours:

- read document, analyse requirement, doing research and considering data structure and technical stack, all of them cost about 2 - 3 hours
- implement app.py and API entrance, cost about 1 hours
- coding for CMC_service, first version cost 3 - 4 hours, and doing some optimize later, totally cost 8 - 10 hours
- coding for report program, cost about 2 hours
- coding for test driver, cost about 1 - 2 hours
- prepare test case, testing and fixing bugs, cost about 4 - 5 hours
- writing documents, cost about 4 hours