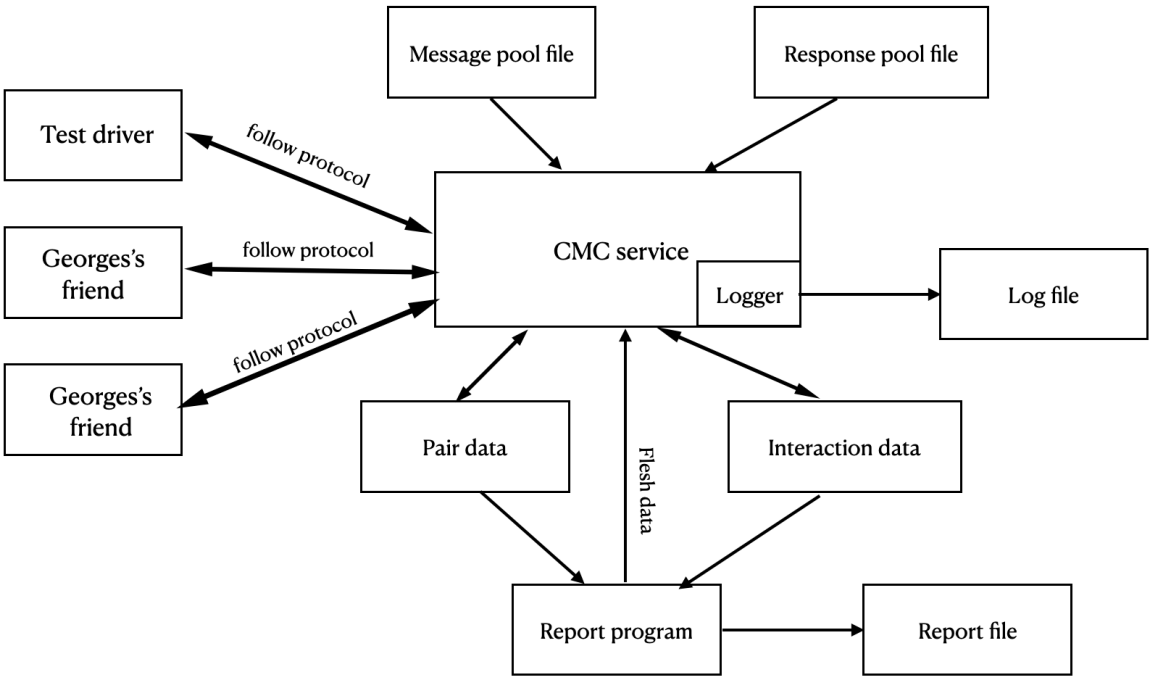
system diagram



protocol design

the CMC service have 5 APIs:

1. **/healthCheck** GET method, check whether CMC service is running.
2. **/message** POST method, send message to CMC service, and CMC service will choose a sentenct to respond. The request body should be like: `{ "friendName": "tuochao", "message": "Test message2" }`
   And response is the sentence.
3. **/rate** POST method, inform whether friend like the response or not. The request body should be like:
   `{ "friendName": "tuochao", "like": false }`
   true means this friend like the response of last message, **false** means don't like. This API should be called after already sending message, otherwise will receive an error reminder message.
   Besides, in order to debug easily this API have another format:
   `{ "friendName": "Tom", "message": "Test message2", "response": "Response message2", "like": true }`
   This means to set like/unlike result for the response of message in body directly, don't care the previous send message. This feature is not inclued in requirment, I just use it to quickly debug giving rate feature.
4. **/saveData** POST method, no request body, save the latest memory data structure into corresponding file, such as message pool, paired data or user interaction history data. This API is used for report program to get the latest statistics data or test driver to flesh data.
5. **/inspect** GET method, can set the query parameter **type** to "message", "response", "pairs" or "interaction" in order to get the corresponding memory data structue. It is used for test drvier to fullfill Bonus 3 requirement.
   There is "protocol_schema" folder which include two files: "CMC.postman_collection.json" is the json schema file of Postman, can be imported in Postman to view request detail. And I also provide request description and example in file "protocol_example.json".

Memory data structue

There are four data files under folder "data", and CMC service will load them when running:

1. **message.csv** This file stored all friends already sent message. It is a csv file, first column is ID and second column is message content. CMC service will load it as pandas dataframe.
2. **response.csv** This file stored all response of service. Same as message.csv first column is ID and second column is response content. CMC service will load it as pandas dataframe.
3. **pair_data.json** This file stored paired relationship between message and response, which are recored by id. Every item key represents the corresponding message id, and every friend has item to record which response he like/unlike for this message. Please note that for one messageevery friend only have one like response , but can havemultiple unlike responses. So the "like" field is int, but "unlike" field is list.
   For example:
   `{ "1": { "Tom": { "like": 1, "unlike": [ 5, 6 ] } } }`
   means for message id 1, Tom like response id 1, but don't like response id 5 and 6. CMC service will load this file as dict.
4. **interaction.json** This stored current user interaction history and statistic data.
   For example: `"Tom": { "totalMessageCount": 11, "uniqueMessageCount": 2, "giveRateCount": 9 }`
   means for now friend "Tom" has already send 11 messages, include 2 messages, and give 9 rate for response. CMC service will load this file as dict.

Data schema for report program

Report program use **interaction.json** to calculate the message count, and use **pair_data.json** to collect pair relationship, schema is aleady explained in above part.

Response select algorithm

pseudo code is as below:

```
find message from message pool according to message content
if not find matched message:
    insert as new data into message pool
    update unique message count of self.interaction_data
else:
    find the message id

record ths message id into self.last_record, which will be used in rate function

if this is a new friend, insert data into interaction_data and pair_data

update total message count of self.interaction_data

if message id already exist in pair_data:
    find corresponding pair data according to message id

    if this friend alread have like response, then return it
    if not:
        find all unlike response id of this friend
        find all like response id of other friends
        exclued unlike response id of this friend from other friends' like response id list
        if the result is not empty, then random select a response from result
        if the result is empty:
            if all response in response pool is unliked by this friend, then return message: "There is no
            if the union set of other friend unlike response cover all response pool, will random select
            if the union set of other friend unlike response NOT cover all response pool, then random sel


else:
    random select a response from response pool, return it
    record response id into self.last_record
```

To be simple, the algorithm can be concluded as:

1. if no pair relationship, then random select response
2. if I have like response, then return it
3. if I don't have like response, but have unlike response, then NOT select them, random select left response
4. if other friend have like response, then select them as priority, but need to excluded my unlike response
5. if other friend have unlike response, then NOT select them, random select left response. If no response left, then random select from other friends unlike list
6. if I don't like all response, then return a static message

Report program algorithm

pseudo code is as below:

```
if service is running, call service to save data
read data from file

traverse interaction data:
    if count is larger than current max count then replace max count value and friend name of max count.
    if count equals to current max count, then append friend name
output max count and friend name

traverse every message of pair data:
    traver every friend of message:
        if this friend have like response id, then find response content from response pool
        if this friend have unlike response id, then find every response content from response pool
        output the message, friend name and like/unlike response
```