REST API document of backend of mentorChat

This is document for RESTful API of backend of mentorChat
It should implement following functions

functions

- user: register a user, login a user, query a user's information, update a user's information, reset a user's password. (Deleting not required now)
- message: send a message to a user, get a MESSAGE
- friends: send a friend request, accept a friend request, retrieving friend list, announcing friends when going online

about MESSAGE

All what server actively sent to client is a MESSAGE.
Different from the common message, a MESSAGE here includes almost everything, from common messages to announcements. We implement a push system with the easy-to-understand blocking message model.

Blocking MESSAGE Model

The basic way we do things is that, when a user ask for a MESSAGE with a GET method (usually in HTTP but not limited), he doesn't recieve the response at once, but will recieve the response when the server want to push something to the client. After recieving a MESSAGE, the client should not wait but request the next one immediately, so that the server can push you the next one.

heartbeat

We don't implement a standalone heartbeat system here for convenience, and it is implemented along with the MESSAGE pushing. Even if there is nothing to be pushed, the server will still push an empty response (not really empty but no real content) to the client if a request last as long as MESSAGE_BLOCKING_LIMIT (in seconds, and it is a constant which will appear in the constant table). Exceeding the time, both the server and the client can assume that the connection is lost. Just like HTTP, we require that the underlaying protocal ensure the safety of transferring, to decrease the possibility that there can be time that only one end assume that connection is lost while the other end stay unknown.

Permissions

Permission system is a system that ensures users see what they should see and change what they should change.

Categorizing

Administrator

DO everything

User

See everything of himself, and changes everything of himsef.

Friend

See everything of the user, and sends messages to the user.

Passer-by

See basic information of the user, and sends friend request to user.

Unlogged in

Able to convert username or usermail into userid

User, password reset

Special permission with the only permission that changes the specific user's password.

API list

All APIs will stay under /api/ so that we can have a website if required.
POST methods should be transferring JSON to convoy parameters, for example

```
{
    "name":"wangyiru",
    "password":"secretDesu"
}
```

Caution : no comment should be included in JSON since it is not a JSON standard.
DEBUG means that the api is only for debugging and should be deprecated when deployed.
BLOCK means that the api may not return immediately, specially used for blocking message model.

- /api/users (DEBUG)
    - GET getting all user information. Fileds: "userid"(uint64[])
- /api/user/new
    - GET allocating a new user id whose information is not set yet, and should be set before using. A token will also be returned so that caller will be able to set password. User id with no password set for time EMPTY_USER_LIFE (constant, in seconds) will be deprecated and returned to the id pool. filed: "userid"(uint64)
- /api/user/{userid}
    - GET returns the information of the user. Content varies as permissions of user. Field names will be named the same as is in the defination of user.
    - POST setting user information, including password. Operation may be denied with low permission. If you don't wish to change a field, do not mention it, otherwise it will be cleared or result in other errors. Field names will be named the same as is in the defination of user.
- /api/user/{userid}/login
    - POST with the right password, a token indicating the user will be returned. filed: "password"(string)
- /api/user/{userid}/resetpassword
    - POST a mail will be sent to the user's mail address to set password. The mail will include a token which will can reset password of the user (with /api/user/{userid} POST). NOT TO BE IMPLEMENTED THIS VERSION
- /api/user/{userid}/messsage
    - POST sends a message to a user. fileds: "message"(string)
    - GET (BLOCK) returns a pushed MESSAGE. fileds: "type"(string, and should be defined in part Defination of MESSAGE Types)
- /api/user/{userid}/friendlist
    - GET returns the friend list of the user. fileds: "friendlist"(uint64[])
    - POST changes the friend list of the user. fields: "behavior"(string, only "DELETE" and adding friends should be done with friendrequest) "friendlist"(uint64[])

- /api/user/{userid}/friendrequest
    - POST sends a friend request to the user (Attention, the request will be recieved through a MESSAGE). A user should also accept a friend request in requesting to be the other user's friend in this way. filed: "requesteeid"(uint64) "description"(string)
- /api/username/{username}
    - GET returns the userid of the requested username. field: "userid"(uint64)
- /api/usermail/{usermail}
    - GET returns the userid of the requested usermail. field: "userid"(uint64)
- /api/file/new
    - POST upload a new file, in the way of multipart. field: "file"(multipart). Returns the file id for future use. field: "fileid"(uint64)
- /api/file/{fileid}
    - GET returns the file, in common raw file, not a multipart.
    - DELETE deletes the file. Only the uploader and administrators have this permission.

data types clarification

- uint64 : precise 64-bit integer unsigned, expressed in hexdecmical
- string : a string of characters, with a length of no limit if not ridiculous
- string(len) : a string with max length of len
- (typename)[]: array of a type of data, such as uint64[]

user properties

- name : string , the name of the user
- password : string, what used to verify user, and should be hashed before transferred to server.
- mail : string, specifically should be a validate mail address.
- description : string, a short description of the user

Defination of MESSAGE Types

constant table

```
int MESSAGE_BLOCKING_LIMIT = 15;
int EMPTY_USER_LIFE = 3600;
```