Server Interface

# Code Request Summary

The server deciphers each request through a series of codes:

DEFAULT\_USER\_LEVEL = 0;

SERVER\_PORT = 80;

REGISTER\_ACTION = 100;

LOGIN\_ACTION = 101;

LOGIN\_SUCCESS\_CODE = 200;

LOGIN\_USER\_NON\_EXIST\_CODE = 201;

LOGIN\_EXIST\_CODE = 202;

RESISTER\_SUCCESS = 300;

REGISTER\_FAIL = 301;

FRIEND\_GET = 500;

FRIEND\_GET\_FAIL = 501;

FRIEND\_GET\_SUCCESS = 502;

FRIEND\_SEARCH = 503;

FRIEND\_SEARCH\_FAIL = 504;

FRIEND\_SEARCH\_SUCCESS = 505;

FRIEND\_ADD = 506;

FRIEND\_ADD\_FAIL = 507;

FRIEND\_ADD\_SUCCESS = 508;

FRIEND\_CHECK = 509;

FRIEND\_CHECK\_FAIL = 510;

FRIEND\_CHECK\_SUCCESS = 511;

MESSAGE\_SEND = 600;

MESSAGE\_RECEIVE = 601;

MESSAGE\_SEND\_SUCCESS\_ONLINE = 602;

MESSAGE\_SEND\_SUCCESS\_OFFLINE = 603;

MESSAGE\_SEND\_FAIL = 604;

MESSAGE\_OFFLINE\_GET = 605;

# Master Functions

## Send to all other users

Send a message to all users. This function needs user level > 0.

Request

{

“action”: 1001

“message”: string

}

Return

Nothing

# Notes

1. Both sending && receiving are using JSON format, other format will get no response
2. Sample request may have incorrect code of response, all correct codes are listed above or below each subtitle.

# User Online Checking Service

Check whether the user is online.

FRIEND\_CHECK = 509;

FRIEND\_CHECK\_FAIL = 510;

FRIEND\_CHECK\_SUCCESS = 511;

Request

{

Username: string

action: int

}

Return

{

Status: string

Code: int

}

Example Request

Send

{

“username”: 1

“action”: 509

}

Return

{

"status":"success",

"code":510

}

# Add Friend Service

Add a friend using the friend’s username.

FRIEND\_ADD = 506;

FRIEND\_ADD\_FAIL = 507;

FRIEND\_ADD\_SUCCESS = 508;

Request

{

Username: string

action: int

}

Return

{

Status: string

Code: int

}

Example Request

Send

{

“username”: 1

“action”: 506

}

Return

{

"status":"success",

"code":508

}

# Search User Service

Search a user’s profile and ready to add the user as a friend.

FRIEND\_SEARCH = 503;

FRIEND\_SEARCH\_FAIL = 504;

FRIEND\_SEARCH\_SUCCESS = 505;

Request

{

Username: string

action: int

}

Return

{

Status: string

Code: int

Result: array that include the user information

}

Example Request

Send

{

“username”: “1”,

“action”: 503

}

Return

{

"status":"success",

"code":502,

"result":[{"username":"1", “email”:”xxx”,”name”:”asd”}]

}

# Fetch Friend List Request Service

The interface between the server and client which allows clients to ask the server to send a JSON list of all the user’s friends.

FRIEND\_GET = 500;

FRIEND\_GET\_FAIL = 501;

FRIEND\_GET\_SUCCESS = 502;

Request

{

action: int

}

Return

{

Status: string

Code: int

Result: array

}

Example Request

Send

{

“action”: 500

}

Return

{

"status":"success",

"code":502,

"result":[

{"username":"1"},

{"username":"3"},

{"username":"admin3"},

{"username":"pls"},

{"username":"serious"}

]

}

# Server Message Sending Service

The interface between the server and client which allows clients send message to others. The user must log in first, or will get nothing response.

MESSAGE\_SEND = 600;

// send a message request to server

MESSAGE\_RECEIVE = 601;

// server send a message to you where the message is from another user

MESSAGE\_SEND\_SUCCESS\_ONLINE = 602;

// message send success to an online user

MESSAGE\_SEND\_SUCCESS\_OFFLINE = 603;

// message send success to an offline user

MESSAGE\_SEND\_FAIL = 604;

// message send failed to a user

MESSAGE\_OFFLINE\_GET = 605;

// send “get offline message” request to server

Message

{

action: int

username: string

message: string

}

Return

{

status: string,

code: int

}

Example Request

Send

{

“action”: 600

“username”: “zirenxiao”

“message”: “have you seen my beer Tibbers?”

}

Return

{

"status": "success",

"code": 602

}

# Server (Both Offline/Online) Message Receiving Service

The offline message is stored in database, and after the client logged in, it should check offline message first.

The offline message will send one-by-one.

If the user is online, the message will deliver directly to user without any request. The online message has the same format as the offline message.

Request

{

action: int

}

Return

{

action: int

message: string

from: string

}

Example Request

Send

{

“action”: 605

}

Return

{

“action”: 601,

“message”: “hahahah”,

“from”: “admin”

}

# Server Login Service

The interface between the server and client which allows clients send login request to server.

When user logs in successfully, the user will be assigned a token, which is a unique ID of current session. Whenever the user sends an action to the server, the token must be attached as the authentication.

LOGIN\_ACTION = 101;

// Send the log in request to server

LOGIN\_SUCCESS\_CODE = 200;

// Log in success

LOGIN\_USER\_NON\_EXIST\_CODE = 201;

// Log in failed, because the user does not exist

LOGIN\_EXIST\_CODE = 202;

// Log in failed, because the user has logged in.

Message

{

username: string,

password: string,

action: int

}

Return

{

status: string,

code: int,

}

Example Request

Send

{

“username”: “test”,

“password”: “donttellothers”,

“action”: 101

}

Return

{

"status": "success",

"code": 200

}

# Client Register Service

The interface between the server and client which allows clients to create a new account in database.

REGISTER\_ACTION = 100;

RESISTER\_SUCCESS = 300;

REGISTER\_FAIL = 301;

Message

{

username: string,

password: string,

email: string,

name: string,

action: int

}

Return

{

status: string,

code: int

}

Example Request

Send

{

“username”: “test”,

“password”: “donttellothers”,

“email”: “[auser@catchmeifyoucan.com](mailto:auser@catchmeifyoucan.com)”,

“name”: “tester”,

“action”: 100

}

Return

{

"status": "success",

"code": 300

}