

**SEJONG UNIVERSITY**

**Department of Computer Science and Engineering**

**Project Report**

**WhiteChat: Android Chatting Service**

**<Jung Jin Young, #15011007>**

**Summer 2019**

**Table of Contents**

1. **Project Summary**
   1. WhiteChat Project
   2. Progress
   3. Develop Timeline
2. **WhiteChat**
   1. Architecture
   2. Android application
   3. Node.js server
   4. Database architecture
3. **Key Project Features**
   1. Full Open Source
   2. API friendly architecture
4. **End of Contents**

**I. Project Summary**

**WhiteChat Project**

WhiteChat Project is about making Chatting service which can be directly serviced with android application. I planned this ‘Android chatting service project’ since beginning of year. And when Database lecture needs some visual output project with database architecture, I started this WhiteChat project right away to finish (as prototype) in semester deadline.

So, in one sentence, WhiteChat is server-using, live, android chatting service like many other android applications. And its meaningful as it will be managed as open source both server-side, client side.

**Progress**

Current progress is a little bit smaller than original proposal. But its core feature and stability is fully proved. Submitted version contains following features.  
 - User management  
 Registration  
 Login, Logout  
 Withdraw  
 - Social management  
 Add friend by id  
 Remove friend  
 Show my friends list  
 - Chatting Room management  
 Create new chatting room  
 Join existing chatting room  
 Leave currently participating chatting room  
 Show currently participating chatting rooms list  
 - Chatting Service  
 Send message to room (live support)  
 Receive message in room (live support)  
 Chat messages store in server, so non-volatile  
 - Android Features  
 Basic User Interfaces  
 Chatting Message Live Receiver (Polling architecture with Google FCM)

**Develop Timeline**

March, 1st ~ March 14th: Database Architecture  
March 14th ~ March 31th: Server API Architecture  
April 1st ~ April 30th: Node.js Server development with database  
April 21th ~ May 14th: Android Application development  
May 14th ~ May 30th: Checked Stability and debugging

**II. WhiteChat**

**Architecture**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Database** Maria DB |  | **Server** Node.js |  | **Android** application |
|  | Socket.io (Web Socket) |
|  | Communicates with API |
| SQL |  |
|  |  |
|  |  |
|  | Google FCM register |
|  |  |
|  | FCM publish |  | FCM subscribe |
| FCM key | Google Firebase Messaging Service | | | |

**Android Application**

**- Application Start** Open socket with Server

**- Login & Register Page** Login Feature & Register Feature

|  |  |
| --- | --- |
|  |  |

If login success, go to <Main Page / Social Page>

- Main Page / Social Page  
 Show Friends  
 Add Friend by ID

|  |  |
| --- | --- |
|  |  |

- Main Page / Room Page  
 Show Participating Chatting Rooms  
 Create or Join Chatting Room

|  |  |
| --- | --- |
|  |  |

- Main Page / Setting Page   
 Foreground Notification on/off  
 Logout/Withdraw  
 Show open-source license

|  |
| --- |
|  |

- Chatting Page  
 Show Previous Chat Messages  
 Send & Receive New Chat Message (live)

|  |
| --- |
|  |

**Node.js Server API Architecture**

**USER MANAGEMENT**

**login**

**event name**

login

**input value**

(string) requestID

the string value of id user requested to login

(string) requestPW

the string value of pw user requested to login (AES encoded)  
  
**output value**  
result is JSON set

(int) result

the result of login attempts  
0 : success 1 : user not exists 2 : user is banned 3 : password error 4 : unknown error

(int) num

user number value (DB index value)

(string) id

user id

(**int**) rank

user rank value

(string) nickname

user nickname

(string) thumbnail

user thumbnail image url

(int) cash

user cash value

(string) created

(JSON ARRAY) rooms

user participating rooms as JSON ARRAY  
(int) room\_num : room number  
(string) room\_name : room name

(JSON ARRAY) socials

JSON ARRAY of users who have social relations with current user  
(int) user\_num : user number  
(string) user\_nickname : user nickname  
(string) user\_thumbnail : user thumbnail image url  
(int) user\_relation : relation type with current user(0:blocked, 1:friend, 2:favorite)

(string) **error**

error description if unknown error thrown

(string) key

session key needed to ensure true user request

**register**

**event name**

register

**input value**

(string) requestID

the string value of id user requested to register

(string) requestPW

the string value of pw user requested to register (AES encoded)  
  
**output value**  
result is JSON set

(int) result

the result of register attempt  
0 : success 1 : ID is already exists 2 : unknown error

(string) **error**

error description if unknown error thrown

**withdraw**

**event name**

withdraw

**input value**

(string) confirmPW

the password value of user requested to withdraw (AES encoded)

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of withdraw attempt fail reason

(string) **error**

error description if unknown error thrown

**refresh socials, rooms**

**event name**

refresh

**input value**

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

(JSON ARRAY) rooms

JSON ARRAY of participating rooms list  
(int) room\_num : room number  
(string) room\_name : room name

(JSON ARRAY) socials

JSON ARRAY of users who have social relations with current user  
(int) user\_num : user number  
(string) user\_nickname : user nickname  
(string) user\_thumbnail : user thumbnail image url  
(int) user\_relation : relation type with current user(0:blocked, 1:friend, 2:favorite)

**SOCIAL MANAGEMENT**

**relation management**

**event name**

relation

**input value**

(int) targetNUM

target user index number

(int) relation

relation type  
0 : remove relation with target 1 : make friend with target

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

(int) targetNUM

target user index number

(int) relation

relation type

**ROOM MANAGEMENT**

**create room**

**event name**

create

**input value**

(string) name

room title

(string) password

room password

(**int**) **target**

base participating target (can be null)

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

**join room**

**event name**

**join**

**input value**

(int) room

room number

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

**leave room**

**event name**

**leave**

**input value**

(int) room

leaving room number

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

**send message (in live)**

**event name**

**send**

**input value**

(int) room

room number of sending message target

(string) message

message string

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

**receive message (in live)**

**event name**

receive

**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(int) room

room index of received message

(JSONObject) message

received message

(string) **error**

error description if unknown error thrown

**retrieve data of selected room**

**event name**

retrieve

**input value**

(int) room

room number of retrieving messages

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) room

room number of retrieving data

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

(JSON ARRAY) messages

JSON ARRAY of room message list  
(int) message\_num : message number  
(string) message\_content : message content

(JSON ARRAY) participants

JSON ARRAY of room participant user list  
(int) user\_index : user index  
(string) user\_nickname : user nickname

**retrieve basics of selected room**

**event name**

retrieve\_basics

**input value**

(int) room

room number of retrieving messages

(string) key

session key needed to ensure true user request  
  
**output value**  
result is JSON set

(int) room

room number of retrieving messages

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

(JSONObject) last\_message

last stored message of selected room

(int) participant\_count

room participant count

**enter room (to track live notice)**

**event name**

**enter**

**input value**

(int) room

room number of retrieving messages

(string) key

session key needed to ensure true user request

**exit room (to stop live notice)**

**event name**

**exit**

**input value**

(int) room

room number of retrieving messages

(string) key

session key needed to ensure true user request

**SERVICE MANAGEMENT**

**register FCM client key**

**event name**

clientKey

**input value**

(string) <FCM client Key>

**output value**  
result is JSON set

(int) result

the result of withdraw attempt  
0 : success 1 : failed 2 : unknown error

(string) message

message of relation attempt fail reason

(string) **error**

error description if unknown error thrown

**Node.js Server SQL codes**

**Login**SELECT \* FROM User WHERE User.user\_id = ? LIMIT 1;SELECT User.user\_index, User.user\_id, User.user\_nickname, User.user\_rank, User.user\_thumbnail FROM User WHERE User.user\_index = ANY (SELECT Social.social\_to FROM Social WHERE Social.social\_from = (SELECT User.user\_index FROM User WHERE User.user\_id = ?));SELECT \* FROM Room WHERE Room.room\_index = ANY (SELECT Participant.room\_index FROM Participant WHERE Participant.user\_index = (SELECT User.user\_index FROM User WHERE User.user\_id = ?));  
[requestID, requestID, requestID]

**Register**  
INSERT INTO User (user\_id, user\_password, user\_nickname, user\_thumbnail) VALUES (?, ?, ?, ?);  
[requestID, requestPW, requestName, defaultThumbnail]

**Withdraw**  
DELETE FROM User WHERE User.user\_index = ? AND User.user\_password = ?;  
[socket.user.num, confirmPW]

**Refresh**  
SELECT User.user\_index, User.user\_id, User.user\_nickname, User.user\_rank, User.user\_thumbnail FROM User WHERE User.user\_index = ANY (SELECT Social.social\_to FROM Social WHERE Social.social\_from = (SELECT User.user\_index FROM User WHERE User.user\_id = ?));SELECT \* FROM Room WHERE Room.room\_index = ANY (SELECT Participant.room\_index FROM Participant WHERE Participant.user\_index = (SELECT User.user\_index FROM User WHERE User.user\_id = ?));  
[socket.user.id, socket.user.id]

**Relation**  
INSERT INTO Social (social\_from, social\_to, social\_type) VALUES (?, (SELECT User.user\_index FROM User WHERE User.user\_id =?), ?);  
[socket.user.num, targetID, relation]

**Create**  
INSERT INTO Room (room\_name, room\_password) VALUES (?, ?); INSERT INTO Participant (room\_index, user\_index) VALUES ((SELECT MAX(Room.room\_index) FROM Room), ?);  
[roomName, roomPassword, socket.user.num]

**Join**  
INSERT INTO Participant (room\_index, user\_index) VALUES ((SELECT Room.room\_index FROM Room WHERE Room.room\_index = ? AND Room.room\_password = ?), ?);  
[room, roomPassword, socket.user.num]

**Leave**  
DELETE FROM Participant WHERE Participant.user\_index = ? AND Participant.room\_index = ?;  
[socket.user.num, room]

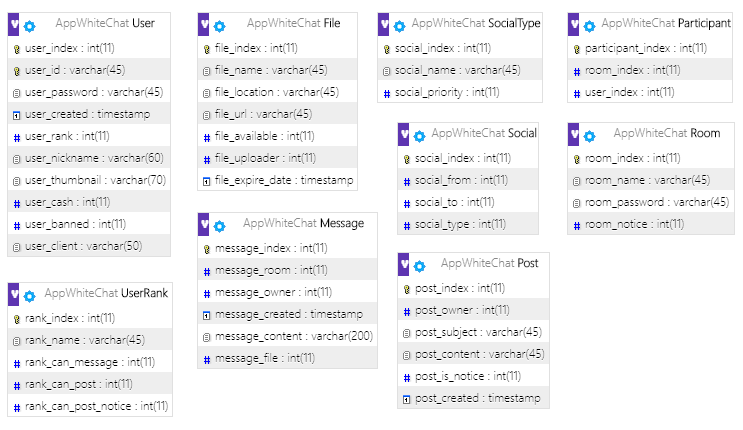
**Send**  
INSERT INTO Message (message\_room, message\_owner, message\_content) VALUES (?, ?, ?);SELECT NOW();  
[room, socket.user.num, content]

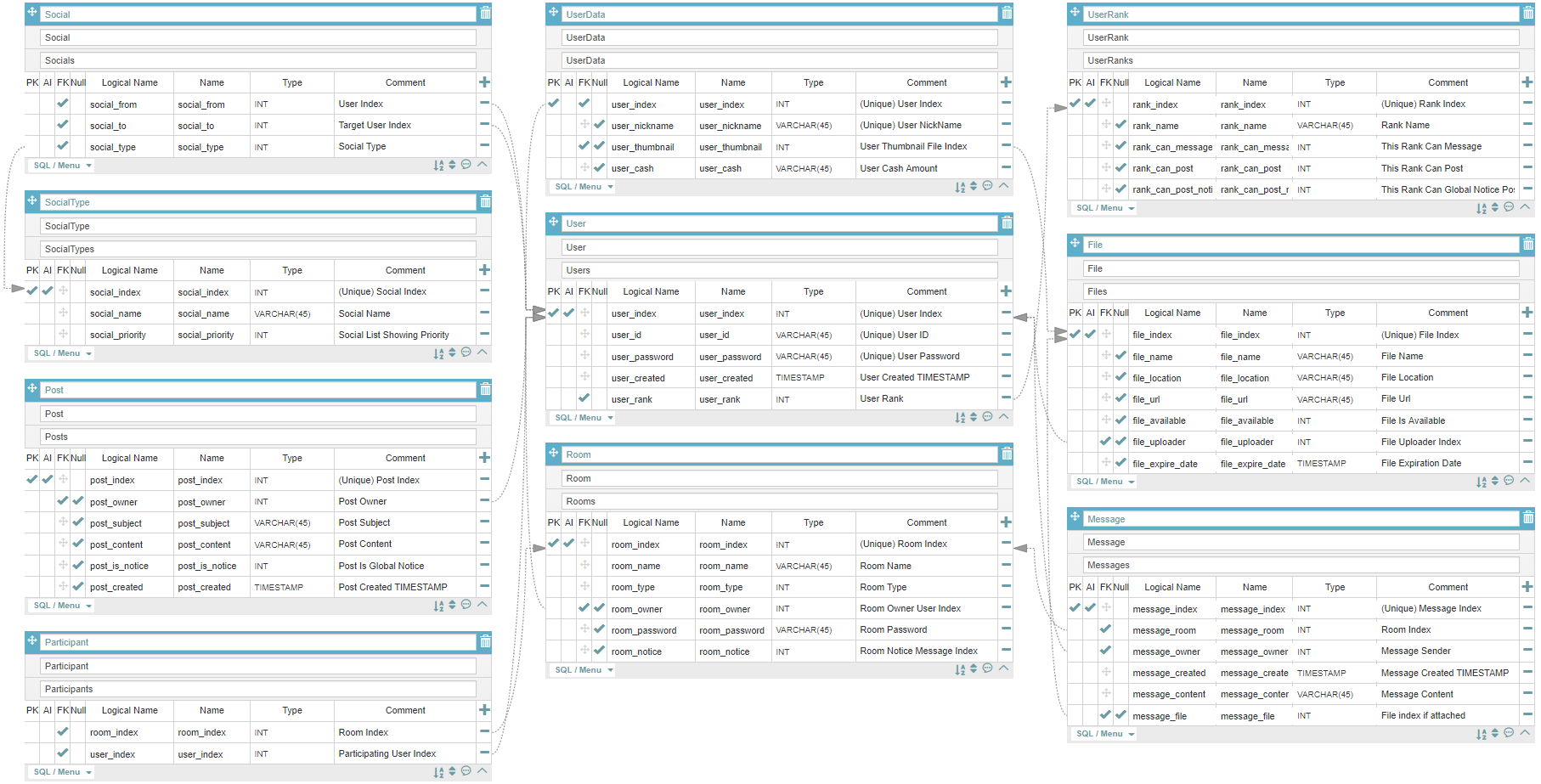
**Retrieve\_basics**  
SELECT \* FROM Message WHERE message\_room = ? ORDER BY message\_index DESC LIMIT 1;SELECT COUNT(Participant.user\_index) FROM Participant WHERE Participant.room\_index = ?;  
[room, room]

**Retrieve**  
SELECT \* FROM Message WHERE message\_room = ? AND message\_index > ?;SELECT User.user\_index, User.user\_nickname, User.user\_rank, User.user\_id, User.user\_thumbnail FROM User WHERE User.user\_index = ANY (SELECT Participant.user\_index FROM Participant WHERE Participant.room\_index = ?);  
[room, index, room]

**ClientKey**  
UPDATE User SET user\_client = ? WHERE user\_index = ?;  
[key, socket.user.num]

**Database Architecture**





**User**  


**UserRank**  


**Social**  


**SocialType**  
  
**Room**  


**Participant**  


**Message**  


**III. Key Project Features**

**Open Source**

This project codes and maintaining service can be one of my good portfolios after this project being free as open source. this project, with both client (android) side, and server (node.js & DB) side has opened as full-opensource with MIT License.

See: https://github.com/LIMECAKE/WhiteChat

**API friendly architecture**

As this project managed as open source, it should contain enough documents. So, I started this project with architecting clear API documents.

See : https://github.com/LIMECAKE/WhiteChat/wiki/API

**IV. End of Contents**

Android application is one of best visible result gained programming language with not that hard difficulty. And, chatting application is one of representing genre of android applications.  
Many programmers, business planner wants to make chatting service, but server-client both side engineering is not that easy thing.

This project codes and maintaining service can be one of my good portfolios. Thanks to this opportunity.