



# Cyber Final Project

Project's subject: Chat-Forum Android Application

Student's name:

Student's ID:

School and city: De-Shalit High School, Rehovot

Teacher's name: Eran Bineth

Hand-in date: 28.3.2020

## Table of Contents

---

1 Introduction .....	3
2 Theory .....	4
3 Final Product .....	12
4 The Project-Writing Process.....	24
5 Solution Components .....	26
6 Testing Scenarios.....	35
7 Reflection .....	37
8 Instructions for Installation and Operation .....	38
9 Bibliography .....	39
10 Appendix .....	40

# 1 Introduction

---

## 1.1 [The Subject](#)

The subject of my final cyber project is a social media application, which allows users to post questions, messages, and other forms of posts in different per-subject rooms. The link to an ongoing room relies on a statistical match between user messages and ongoing posts, to generate best alignment to users' interest.

The application is a platform for discussion on various topics, primarily used as an educational platform for school course debates.

The project is a unique cross-discipline system that combines user interfaces from both network desktops and Android applications, via client-server and DB protocols.

## 1.2 [Main Goals](#)

- Supporting multiple android and desktop clients
- Maintaining an appealing user interface
- Keeping historical and ongoing discussions across multiple runs
- Developing a reliable and robust platform for educational discussions through message / questions posts
- Generating topic-varied rooms based on user post information to host different discussions of subjects
- Creating a database connection that can reliably and quickly transfer user data

## 1.3 [The Rationale](#)

The rationale for this project was an opportunity arising from a school's need to create a platform that could connect School's student users into discussion rooms that are differentiated by topic/subject. From this request, I settled on the solution of a joint-project Android application, since a smartphone application could provide a more convenient and accessible platform for students.

I also considered the material I will use from the Cyber course as well as learning many new things about app development, Python data structures, Server-Client interaction, Socketing, and Database Management. I hope that I will be able to

improve my ability and efficiency of Python coding, and general time management for projects.

## 1.4 The Reference to School Material

The Project is composed of few subjects that were taught in the Cyber course

- Server-Client Interaction
  - Client inputting commands which Server executes
- Socketing
  - Client-Server protocols, and porting (Finding free ports)
- Python
  - Coding Language that Client and Server are constructed
  - Data Structures
  - Working with databases
  - Graphics using tkinter

## 2 Theory

---

### 2.1 Theory

(a) MongoDB – MongoDB is a NoSQL, cross-platform, document-oriented database used for large volume data storage / Big Data. MongoDB stores JSON-like documents and has high flexibility when it comes to querying and indexing. MongoDB includes drivers for many programming languages, including Python and Java.

In the project, I need to store, transfer, and manipulate data from all user messaging and actions in each room, which means the scalability and performance of the database needs to be optimal and run quickly. Thus, I decided to use MongoDB, due to its high performance with Big Data. Also, Python and Java MongoDB drivers allow the Python Server and Android Clients to be connected to the same, constantly updating room databases.

MongoDB data is stored on the cloud. MongoDB users can select which cloud provider and which region will the MongoDB cluster be created, in which databases, collections, and documents can be created/updated/deleted. For this project, Amazon Web Services is the selected web service provider, and the recommended region that was selected was Frankfurt – Central Europe.

MongoDB database is used in this project to provide information (rooms, messages, users, etc) for both tkinter and Android clients. Specifically, when users activate a command, the relevant information from MongoDB is displayed to users just through query methods such as “find”, “findall”, “find with a query filter”. For real time messaging synchronization, each message document in the chats collection in MongoDB has a flag that signifies if the message has been posted to both interfaces (tkinter and Android) for the room in which it has been posted.

MongoDB mirroring for both types of clients is demonstrated by using documents with the same fields in order for android clients to recognize tkinter client documents, have query methods to find data from either client type, and message documents have a specific flag to check for total synchronization.

(b) Keyword Extraction is a substantial part of the theory of the project

- Search Room Scenario / RAKE – Keyword Extraction

When users request to search room, they are given the option to search via room name or via keywords.

Room name search is a basic form of search, as it traverses the Open Rooms collection and searches the specified room name from the documents. If found then users can join the room and start chatting, through Join and Chat functions. If not found then message is return to user, that room has not been found and suggests searching for a different name, keyword, or display popular/available rooms (Join function)

Keyword search is more complex, in which keywords from all messages in each room are extracted. When a user types in a keyword, the server goes through the extracted keywords in each room to find if the searched keyword exists. If searched keyword is located in a set of rooms, the rooms with a high "score" or concentration of the searched keyword will be displayed to the user. Then, users can choose any given room, and thus proceed to the Join function.

RAKE, standing for Rapid Automatic Keyword Extraction technique, is a python library that can be used for keyword extraction that can be installed through pip installer. The library also requires text file that includes the most common words, which acts as a word blacklist.

For example - Text: “Google quietly rolled out a new way for Android users to listen to podcasts and subscribe to shows they like, and it already works on your phone. Podcast production company Pacific Content got the exclusive on it. This text is taken from Google news.”

1. Converts user message text into lower case, and takes all words into an array
  - a. Using split method with commas, periods, etc

```
['google',
 'quietly',
 'rolled',
 'out',
 'a',
 'new',
 'way',
 'for',
 'android',
 'users',
```

2. The array is then split into different parts or sequences of contiguous words by blacklisted words and punctuation marks.

Split by delimiters	Split by stop word	Candidate Keyword
['google',	'google'	['google', 'quietly']
'quietly',	'quietly'	
'rolled',		
'out',		
'a',		
'new',		
'way',		['android', 'users']
'for',		
'android',	'android'	
'users',	'users'	

- a. Each sequence is considered as a candidate keyword since keywords can include a phrase (ex: Star Wars)
- b. From the text above, the array holding all the words in a given text is split based on a set of delimiters and stop words; delimiters are spaces and punctuation, and stop words are words located in blacklist. They do essentially the same thing, which is to split the general array into separate arrays with potential keywords/phrases.

- c. The end sub arrays of words are classified as candidate keywords, since they are not stop words, they are grouped together since there is nothing (delimiters and stop words) that separate the words.

```
'google', 'quietly', 'rolled'
'android', 'users'
'listen'
'podcasts'
'subscribe'
'shows'
'works'
'phone'
'podcast', 'production', 'company', 'pacific', 'content'
'exclusive'
'text'
'google', 'news'
```

result from delimiters and stop word filtration

### 3. Calculation of keyword "score" – ratio of word degree to word frequency

#### a. Creation of co-occurrence graph – matrix of word occurrences

	google	quietly	rolled	android	users	listen	podcasts	subscribe	shows	works	phone	podcast	production	company	pacific	content	exclusive	text	news
google	2	1	1																1
quietly	1	1	1																
rolled	1	1	1																
android				1	1														
users				1	1														
listen						1													
podcasts							1												
subscribe								1											
shows									1										
works										1									
phone											1								
podcast												1	1	1	1	1			
production												1	1	1	1	1			
company												1	1	1	1	1			
pacific												1	1	1	1	1			
content												1	1	1	1	1			
exclusive																	1		
text																		1	
news	1																		1

- Frequency is defined as the number of times a word from the initial array, appears among all candidate keywords
- Degree is defined from the matrix below as the sum of “numbers” for a given word (in a row) - Since each number

represents an “occurrence” in a candidate keyword, thus degree is the frequency of a word + the sum of the words that the initial word appears with another word in a candidate keyword

- iii. Thus, the matrix below displays the number of occurrences that a word is placed in a keyword with another word; and on a row-column with the same word, the value displayed is the number of times that word is located in a keyword (sub array (B/C))
- iv. Example word – “google” → Example c. keyword – “google news” (see below)

b. Calculating Word Frequency value

- i.  $\text{Frequency}(\text{word}) = \# \text{ of times a word appears in a candidate keyword}$
- ii. Since “google” appears in 2 c. keywords (image C),  
 $\text{freq}(\text{google}) = 2$

c. Calculating Word Degree value

- i.  $\text{Degree}(\text{word}) = \text{Frequency}(\text{word}) + \text{sum of } \# \text{ of times (word) is located in a candidate keyword with another word}$
- ii.  $\text{Freq}(\text{google}) = 2$  (above)
- iii. Since “google” has been with “quietly”, “rolled”, and “news” all once, thus “sum of # of times (“google”) is in a c. keyword with another word” is 3
- iv.  $\text{Degree}(\text{google}) = 2 + 3 \rightarrow 5$

d. Combination of multiple words in candidate keyword / Final Score

- i. To find the score of a candidate keyword, the ratios ( $\text{deg}/\text{freq}$ ) of each word in a keyword are added together.

Word	google	News
Degree (word)	5	$1 + 1 \rightarrow 2$
Frequency (word)	2	1

$$\text{“google news”} = (\text{freq}(\text{“google”})/\text{deg}(\text{“google”})) + (\text{freq}(\text{“news”})/\text{deg}(\text{“news”}))$$



- ii. The table below displays the values of frequency and degree of each word in “google news” candidate keyword
- iii. Thus the score of “google news” equals  $2.5 + 2.0 \rightarrow 4.0$

4. Array of tuples, keyword & score, is returned, which is from the library

```
('keywords: ', [('podcast production company pacific content', 25.0), ('google quietly rolled', 8.5), ('google news', 4.5), ('android users', 4.0), ('exclusive', 1.0), ('works', 1.0), ('phone', 1.0), ('text', 1.0), ('podcasts', 1.0), ('subscribe', 1.0), ('listen', 1.0), ('shows', 1.0)])
```

## 2.2 Existing Similar Products

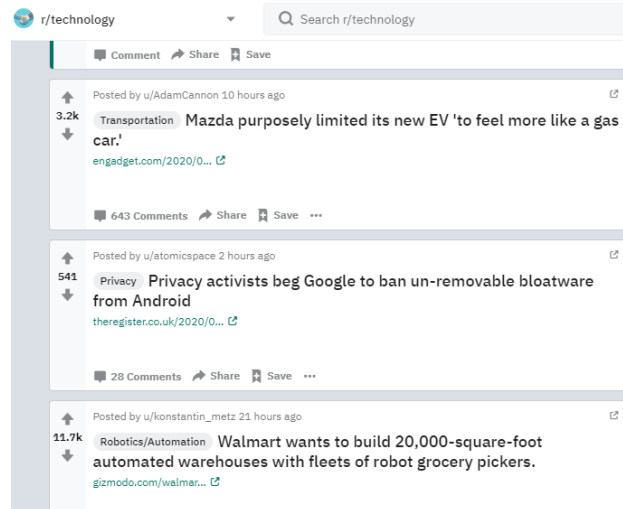
The project is a hybrid between a chat room application and a forum application/website. Chat room aspects include ability to send messages in rooms consisting of other users; private or direct messaging between users; message search. Forum aspects include topic specific rooms; key word to topic search; and question and community sharing in rooms. The main separation between the two different types of products is that chat rooms allow you to communicate with people in real time, whereas forums are more suited for discussions where not all participants have to be online at the same time.

Since user posting is stored in MongoDB, the application can support real-time chat rooms, as well as structured forum rooms.

Similar products that relate to Forum aspects:

- Reddit – (Founded 2005)
  - Website and application which anonymous users can post messages and media (Images & Video) in over a million of “communities” called a “sub-reddit” threads. Each subreddit thread is a different forum room with a different topic that is given by admins; and messages that do not apply to a subreddit topic or that are inappropriate are moderated by moderators
  - In each post in a subreddit, users can post replies and “Up vote” or “Down vote”, Like/Dislike system, all posts and replies in order to increase/decrease viewership
  - Similarities include different rooms for various topics, topic search, and room messaging
  - Differences:
    - Topic variation – keyword base (Project)

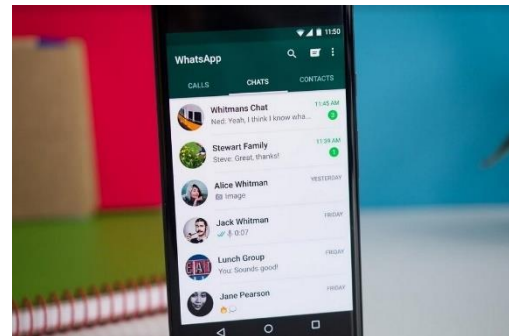
- Voting System (Reddit)
- URL topic search (Reddit) - /r/topicname



<https://www.reddit.com/r/technology/>

Similar products that relate to Chat aspects:

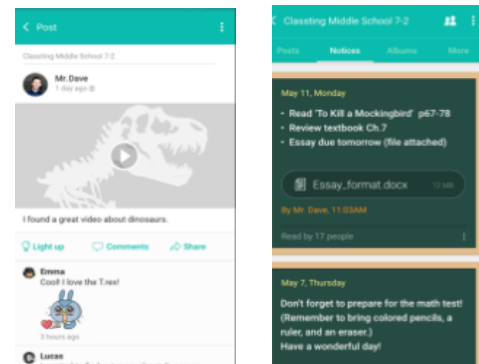
- WhatsApp – (Founded 2009)
  - Messaging application, which allows users to send messages, pictures, videos, stickers, and voice recordings over internet-based chat rooms rather than mobile network. WhatsApp uses end-to-end encryption as its security feature. Chat rooms can include direct messaging and group messaging
  - Similarities
    - Real-time user messaging
    - Group messaging
  - Differences
    - Topic variation – key word base (Project)
    - End-to-end encryption (Whatsapp)



[https://www.phonearena.com/news/WhatsApp-mysteriously-disappears-from-the-Google-Play-Store\\_id119603](https://www.phonearena.com/news/WhatsApp-mysteriously-disappears-from-the-Google-Play-Store_id119603)

Similar product that relate to Educational aspects:

- Classting – (Founded 2012)
  - educational social media application where users are split into classrooms, which are essentially chat rooms/ message boards which are defined by school data/classrooms. These classrooms allow for admins/teachers to post announcements,



homework, etc. so students and parents are notified. Classting provides an educational platform for students to interact with teachers via messages and media, and for parents to be updated about student/teacher announcements.

- Similarities:
  - Different Rooms for specific topics (Ex: Reading Period 6)
  - User Messaging
  - Educational Platform
- Differences:
  - Fixed topic rooms (Classting) vs Topic variation – keywords (Project)

<i>Feature/Product</i>	<b>Reddit</b>	<b>Whatsapp</b>	<b>Classting</b>	<b><u>Project</u></b>
Display Past Messages / Posts	✓	X	/ (Special posts are displayed)	✓
Topic specific Rooms	✓ (subreddits)	/ (Chat room names)	✓ (classrooms)	✓
Keyword dependent Topic Definition	X (subreddit has defined topic, moderation involved)	X	X	✓
Application with PC integration	✓	✓	✓	✓
Keyword score-based Room Search	✓ (URL Search/Room Titles)	/ (From all messages)	X (Classrooms Private/ID based)	✓
Public Rooms	✓	X	X	✓
Encryption?	✓ (subreddits and comment sections not sniff able)	✓	X	✓ (All versions of MongoDB support TLS, SSL, and AES)

This project provides users a platform with chat - forum with functionality such as Whatsapp, an educational social platform such as Classting, and a forum

platform with discussion areas with topic variation and fluctuation such as Reddit, in one application.

## 3 Final Product

---

### 3.1 Project Description

The project is a python program- where users can post messages in rooms that are topic varied. The main “topics” of each room are based on keywords that are extracted in accumulation of users’ posts. In other words, the “topics” of the rooms are not fixed and can vary, and change based on the current discussion.

The project is intended to be an educational platform where students from a school can search rooms based on specific keywords/topic keywords, in which students can join rooms in which searched keywords are prevalent, in order to learn or ask other students/teachers. For example, if a student wants to know about physics related material, the student can search the term “physics” or “newton” and rooms that have those topic keywords will appear, for user selection. When a user requires expertise or general help from a specific field, the keyword / topic search allows the user to find rooms that are potentially helpful in giving information.

Entering the application, a user needs to input username and password credentials that are saved in a database section in MongoDB. Then the user has three main commands in which the application is developed upon, *Create Room*, *Join Room*, and *Search Room*. All three commands then allow users to chat/post messages

- **Create Room:** Creates new room that users can join (initially without an assigned topic)
- **Join Room:** Displays top available rooms based on popularity, and user choice (Number of messages or Participating users), and allows users to join in.
- **Search Room:** based on topic/keywords - Displays rooms with higher concentration (RAKE score) of searched keyword (from highest to lowest) or by Room Name

The program has other features such as *delete room*, *user search*, *private message*, and *change password*.

- **Delete Room:** Delete room with double confirmation, along with Room Database section in MongoDB

- User Search: Search based on username, given public user info such as current room or most active room
- Private Message: Direct messaging between two users
- Change Password, following initial authentication

## 3.2 Main Algorithms

### Authentication

Since the application is an educational platform, user data will be solely based on school data. Thus all eligible users (students, teachers) will have access to the application using username and a created code password (generated and given to students) which is stored in MongoDB, and the password can change.

1. Function receives username and password, and identification query is created for given username against user instance/document in MongoDB in "Users" collection
2. For loop – for each instance in MongoDB with given username - If "password" value in instance is the same with inserted password – returns TRUE (identification matched, else FALSE

### Create Room

1. Function receives username, and client-socket for python clients; just username for Android clients
2. Creates new document instance of new room, default name Room(#), which is stored in room collection in main database in MongoDB, updates collections in main database, note topic is not defined yet.
3. Insert JSON-like document into main database in rooms collection that displays all available rooms - Ex: {"Room Number": roomnum, "Created by": username}
4. Redirect to chat function to start sending chats with action flag is "Create"

### Join Room

1. Function receives clientsock and username
2. Room query is created to count iteratively number of available rooms (created rooms) in room collection in main database

3. If number of available rooms is less than 25-30, display all available rooms to user, otherwise displays popular rooms based on user choice – Number of messages or Participating users  
Number of Messages – queries the “chat collection” and Participating Users – queries the "Participating Users" collection. Rooms with highest number of messages are displayed to user, then placed in data structure (list) Room number is inserted and validated using for function
4. Finally JSON doc is created for the user joining in specific room (user action) and then chat function is initiated (user can send chats)

### MongoDB Actions based on Client Commands

- Main database and separate collection instances are called/defined in initialization process
- Create Command
  1. Inserts room document for every created room in room collection under the main database in MongoDB
- Join Command
  1. Uses find method to retrieve all rooms, using a for loop (for each room) I use find one method in chat collection in count total chats and the rooms with the highest amounts of chats I will display the room names to the user for selection
  2. Once user selects, user is directed to chat function
  3. Active user document is inserted in active user collection with the room name field being the room name selected
  4. User Action document is inserted in actions collection with the room name field being the room name selected
- Search Command
  1. User can decide to search using the room name itself, or through a keyword.
  2. Uses find method with a filter of user inputted keyword/phrase (if keyword) for search, if no keyword documents in keyword collection are retrieved with filter, then user can input again or join by room

- Find method with query filter of user inputted room name (if room name) through room collection. If there are no room documents then user can input again/create room/search by keyword
3. Once keyword documents are retrieved, the corresponding room names with the retrieved documents are posted to the user.
    - The information included in keyword documents are the word/phrase, RAKE score, and the room that the word/phrase is located in, so to the user the room along with the RAKE score will be shown from the highest score to the lowest score
    - If room documents are retrieved (search by room name), then they are displayed for user selection
  4. Once user selects room, active user and user action document is inserted to respective collections, with selected room name being in the room name field
    - The active user and user action documents will similar to join room (“user joined by keyword search”) or (“user joined by room search”)

#### Chat – Python (Tkinter test clients)

1. After joining or creating a room, the user can post messages.
2. For Tkinter test clients, messages are (a) sent through client sockets and then (b) saved to the chat collection in the main database and is (c) broadcasted to all tkinter clients that are in the specific room. Android users trigger activity by DB update.
3. Since messages that are posted in a room are saved in MongoDB, users that join rooms can see previous messages that were sent by other users before that specific user has entered, creating a forum-like environment.
4. After a certain timeframe, messages in a room are deleted however they are still saved in the database for keyword extraction.
5. Joined users see limited history of messages according to the time stamp from messages .vs. today's date.
6. “end” chat removes users from current room and removes client-sock from dictionary, and all messages from Room are placed into one string

and then RAKE algorithm (explained later) extracts keywords which are then inserted in Keyword collection in each room database

### Handler – Socketing Function

1. Function receives clientsock and addr (address – Host, Port)
2. Authentication Function
3. Creates Actions/Commands dictionary with possible commands
  - a. Join, Create, Search, etc. as keys
  - b. Functions as values for each key
  - c. Ex: ('JOIN': 'join\_room(clientsock,username)')
4. While loop for client actions
  - a. Client can quit
  - b. Data from user → data.upper() → compared to Actions/Commands dictionary keys then value function is executed if comparison is found using "exec" method
5. Clientsock is closed once Client quits from button press (tkinter/application) → sends "QUIT" message and executes value with key of "QUIT" in Actions/Commands dictionary

### 3.3 Constraints and Requirements

The constraints/requirements of the final products are that keyword topic search is supported only with the English language (letters, words), thus keyword topic search and definition can only be used with English messages.

Another constraint/requirement is that in order for keyword topic search to be initialized there needs to be data already registered, thus in the beginning stages of product release, keyword search will not produce useful results until there is an increase in user message data.

Media integration is not supported in tkinter clients. There needs to be rooms created for join room/search (by room name) commands to be operable (used for its function)

### 3.4 Outgoing and Incoming Interfaces

My project interacts with Android devices via the MongoDB interface, as discussed extensively in this document. Other than that, it has no outgoing and/or incoming interfaces.

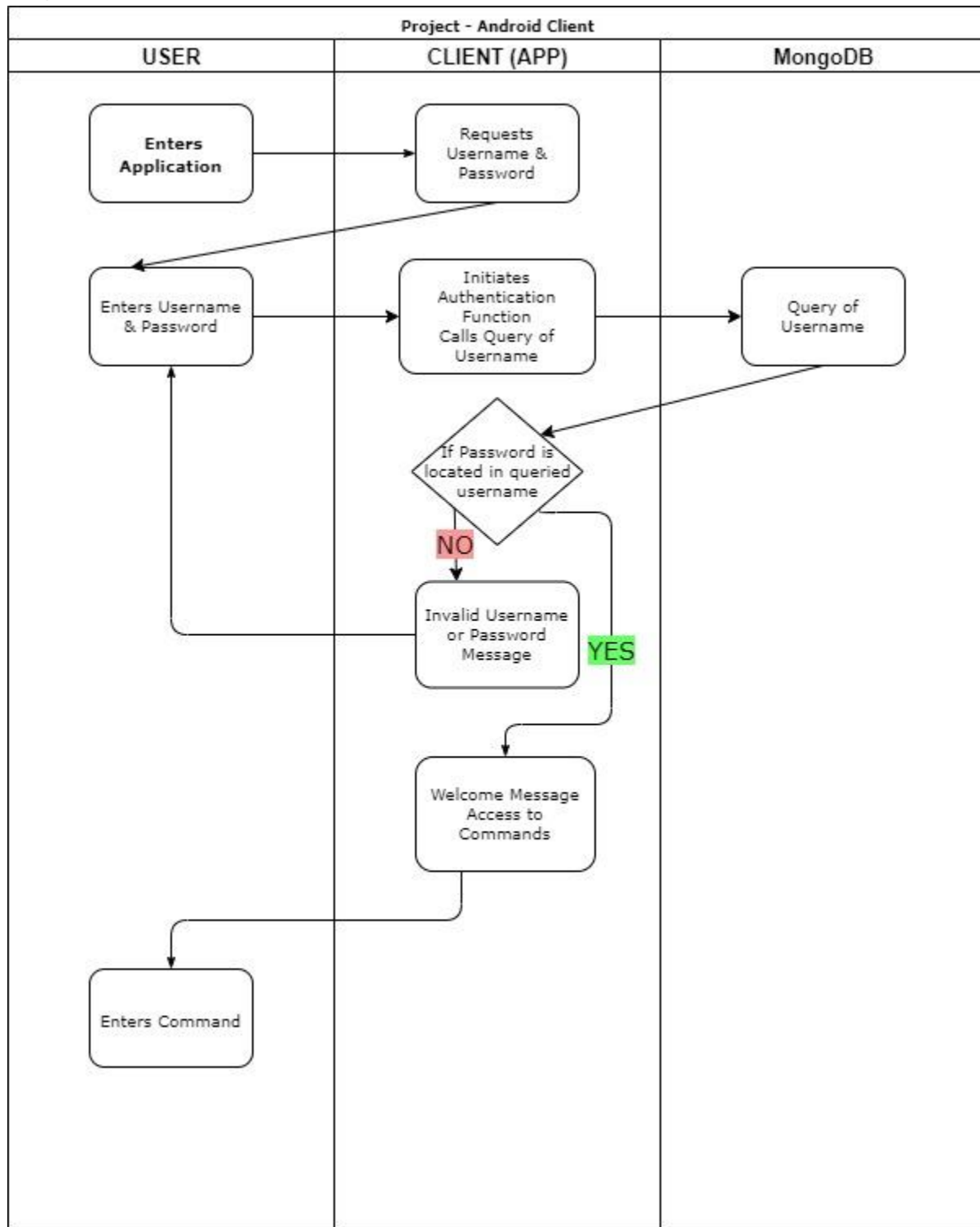


### 3.5 [Reference to Security](#)

The application includes an authentication process, where users need to input correct username and password information, and new users can sign up. Password change is also an option for those who forgot password information through android clients.

### 3.6 Use Cases

#### Login Process



Project - Android Client

```

graph TD
    subgraph USER
        UC[Enters Command]
        EC[Enters Chat]
        ERN[Enters Room Name]
    end

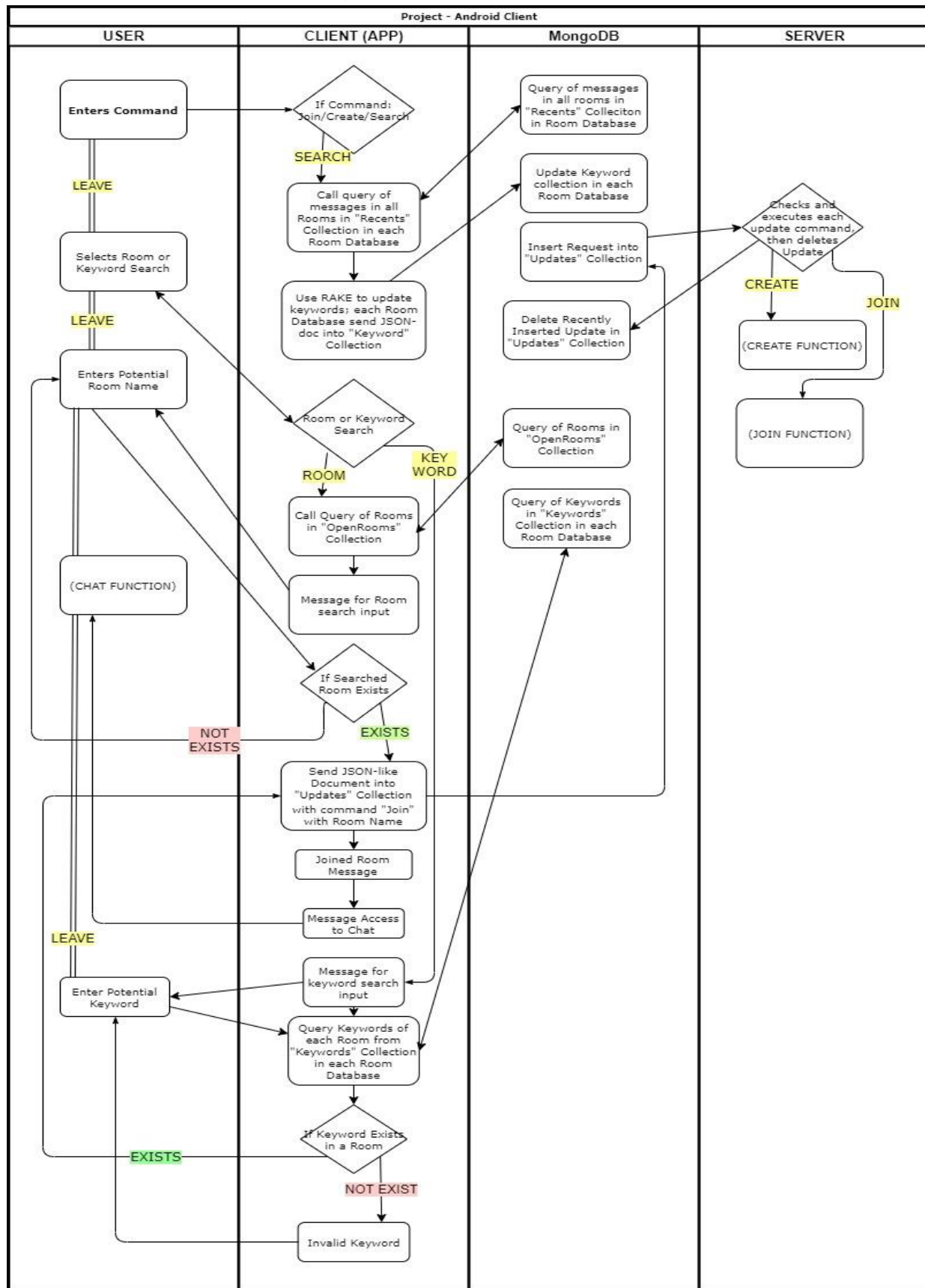
    subgraph CLIENT_APP [CLIENT (APP)]
        ICCC{If Command: Join/Create/Search}
        C1[CREATE]
        SJD1[Send JSON-like Document into "Updates" Collection with command "Create"]
        CRM[Created Room Message]
        MAC1[Message Access to Chat]
        DMS1{Display Message, Send JSON-doc with message info Updates Keywords}
        END1[END]
        LR[Leave Room]
        CQR{Calls Query of number of Rooms from "OpenRooms" Collection}
        IECN{If exist certain # of rooms}
        LCA[Displays List of Available Rooms]
        MNR[Message - No Rooms Available refers to Create Room]
        CQM[Calls query of message info from "Recents" collection and displays most popular rooms]
        CQOV{Calls Query of Open Rooms to check validity}
        INV[Invalid Room Name Message]
        SJD2[Send JSON-like Document into "Updates" Collection with command "Join" with Room Name]
        JRM[Joined Room Message]
        MAC2[Message Access to Chat]
    end

    subgraph MONGODB [MongoDB]
        IRU[Insert Request into "Updates" Collection]
        DRIU[Delete Recently Inserted Update in "Updates" Collection]
        INR[Insert new room into "OpenRooms" Collection]
        CRD[Create Room Database, Inserts User into "Active Users", and "Actions" Collections]
        IMR[Insert message info into "Recents" collection in Room Database; Update Keywords]
        RU[Remove User from "Active User" collection in Room Database; Insert user action in "Actions" collection]
        QRR1[Query Rooms from "OpenRooms"]
        QMR[Query of messages from "Recents" collection in each Room Database]
        QRR2[Query Rooms from "OpenRooms"]
        IU[Insert User to "Active Users" collection in given Room Database; Insert user action in "Actions" collection]
    end

    subgraph SERVER
        CEU{Checks and executes each update command, then deletes Update}
        C2[CREATE]
        SJD3[Send JSON-Doc into "OpenRooms" Collection]
        SJD4[Send JSON-Doc into "Active Users" & "Actions" Collection in given Room Database]
    end

    UC --> ICCC
    ICCC --> C1
    C1 --> SJD1
    SJD1 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ERN
    CQOV --> SJD2
    SJD2 --> IRU
    IRU --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR2
    QRR2 --> IU
    IU --> SJD4
    SJD4 --> CEU
    CEU --> DRIU
    DRIU --> INR
    INR --> CRD
    CRD --> IMR
    IMR --> RU
    RU --> QRR1
    QRR1 --> CQR
    CQR --> IECN
    IECN --> LCA
    LCA --> ERN
    IECN --> MNR
    MNR --> ERN
    IECN --> CQM
    CQM --> CQOV
    CQOV --> INV
    INV --> ER
```

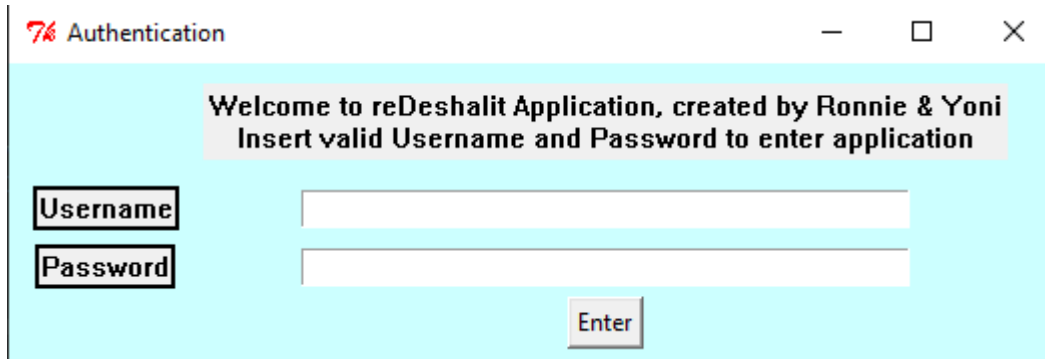
## Main Command 2 (Search/Keywords)



### 3.7 User Interface TKINTER users

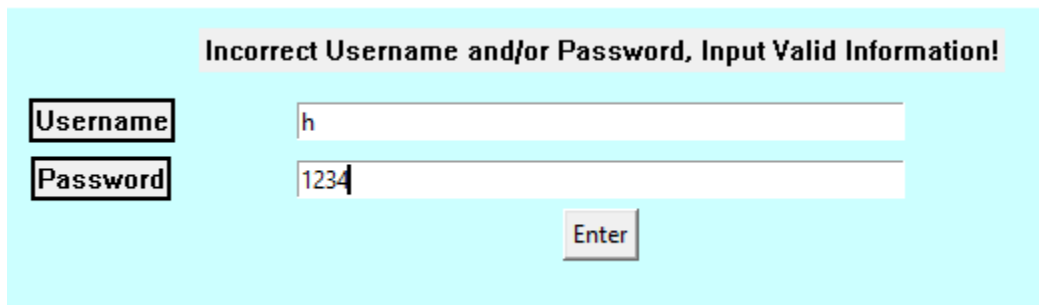
The document does not discuss the Android user interface (out of scope).

#### Authentication Screen



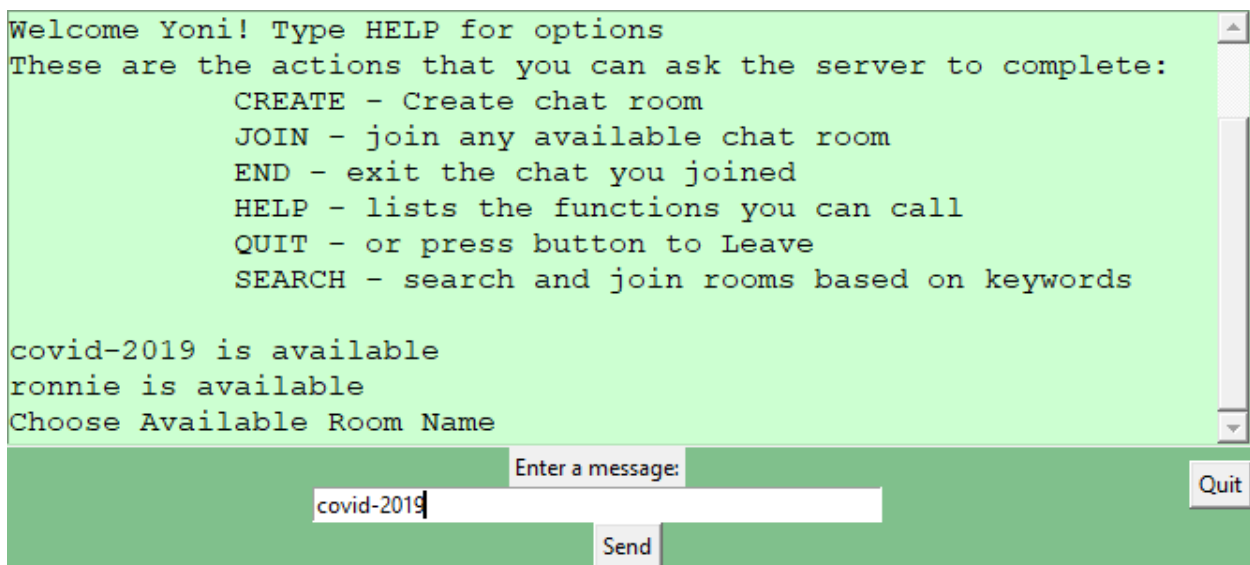
A window titled "7% Authentication" with a light blue background. It contains a message box that says "Welcome to reDeshalit Application, created by Ronnie & Yoni" and "Insert valid Username and Password to enter application". Below the message are two input fields labeled "Username" and "Password", and an "Enter" button.

#### Authentication - Wrong Username/Password



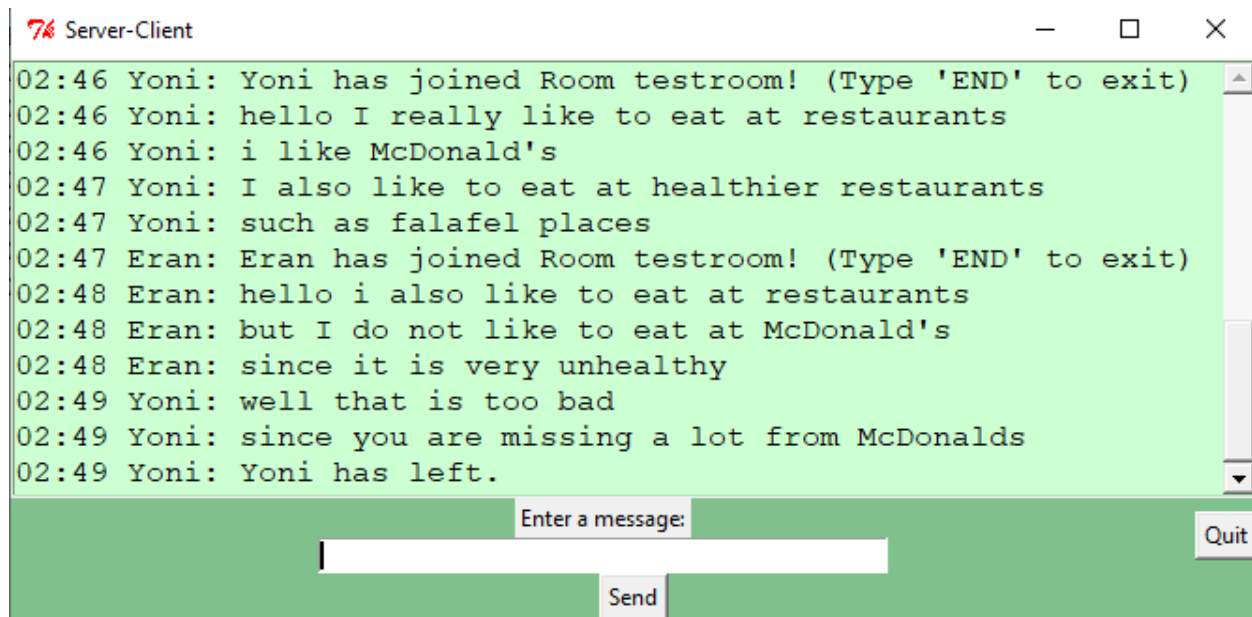
A window showing an error message: "Incorrect Username and/or Password, Input Valid Information!". The "Username" field contains the letter "h" and the "Password" field contains "1234". An "Enter" button is visible below the fields.

#### Request to Join a Room



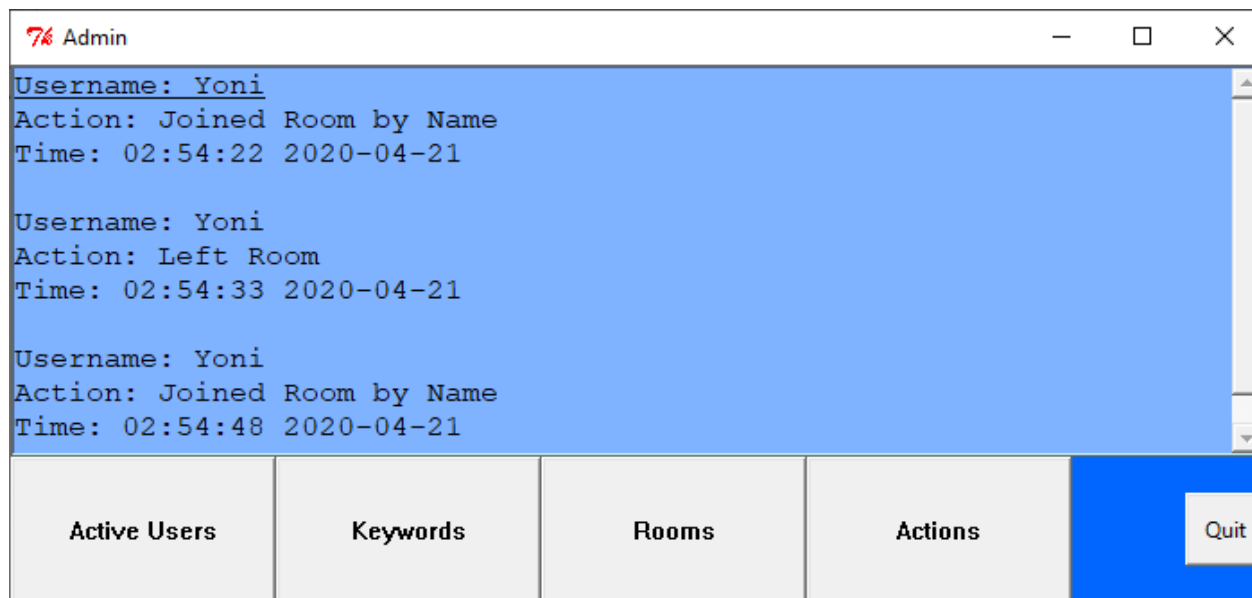
A window with a green background displaying a list of available rooms: "covid-2019 is available" and "ronnie is available". It prompts the user to "Choose Available Room Name". Below the list is a text input field with "covid-2019" entered. To the right of the input field is a "Send" button. Above the input field is a "Quit" button. The window also displays a list of actions that can be asked the server to complete: "CREATE - Create chat room", "JOIN - join any available chat room", "END - exit the chat you joined", "HELP - lists the functions you can call", "QUIT - or press button to Leave", and "SEARCH - search and join rooms based on keywords".

## Chatting and Exiting



## 3.8 User Interface ADMIN

### Actions



## Rooms

7% Admin

Room Name: covid-2019

Room Name: ronnie

Room Name: test1

Room Name: test22

Room Name: test420

Active Users

Keywords

Rooms

Actions

Quit

## Active Users

7% Admin

Username: Yoni

Since Active in Room: 02:57:02 2020-04-21

ChatroomName: test420

Username: Eran

Since Active in Room: 02:58:33 2020-04-21

ChatroomName: test22

Username: Roni

Since Active in Room: 02:58:54 2020-04-21

ChatroomName: test1

Active Users

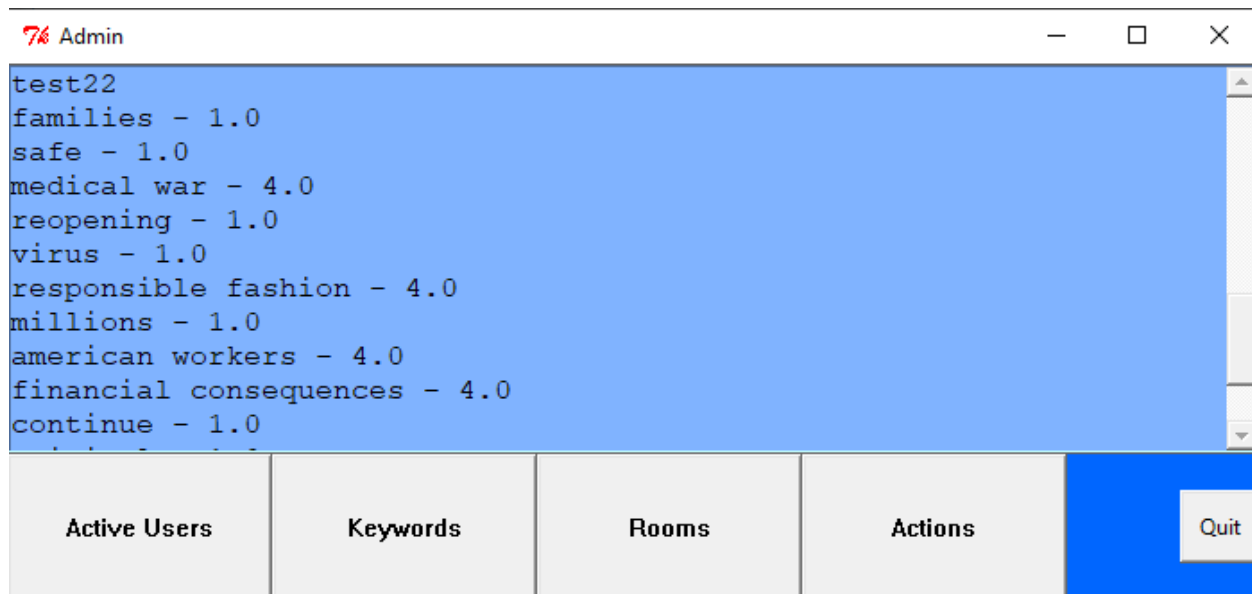
Keywords

Rooms

Actions

Quit

## Keywords



## 4 The Project-Writing Process

### 4.1 [The Process](#)

First, I wrote a basic server and client application in which you could transfer echo text data, and later implemented multi-client chat functionality. Then I started writing the first version of some of the commands I was going to insert to the program, create and join. Since I decided to use a non-SQL database, MongoDB, I took a considerably amount of time to plan the database/collection configuration as well as implement MongoDB methods to the program commands that I was working on; so in parallel, I wrote the commands using only data structures (lists, dictionaries) and tested MongoDB database/collection/document configuration.

After I understood how to insert, remove, search documents in MongoDB, I wrote the Join and Create commands using MongoDB methods and my first version of database/collection configuration in MongoDB. Where each room would be a separate database, and each room database would have the collections for recent messages, user actions, and old messages.

Then after finishing Join and Create commands, I used the RAKE library to write the Search command function, which is the search room by keyword function. I added a keyword collection to each room database, and inside the Search command function I added loops to update keywords of message data of each room and save the keywords to the keyword collection in each room database.



After finishing the Join, Create, and Search functions I started on integration with Android clients. Since the current database/collection configuration in MongoDB would not be supported on Android Clients, I changed the configuration, in which there would be one general database that would have the collection of rooms, user actions, active users, keywords, port, and chats. Each document in the collections would have a room field for search/capture purposes.

Then database configuration, changes in the command functions were made to help organization, chat synchronization, and adding more features. In parallel, I wrote the Admin which was connected directly to the server through socketing, and would display server information (information that the server would send to MongoDB, but would also send to Admin), such as keywords, active users, user actions, and rooms.

After creating the Admin, I optimized keyword filtration and improved GUI elements for client and admin. Added final add on features, such as direct messaging.

## 4.2 Challenges and Different Implementation Options

### **Challenges**

- Mirroring Information from tkinter clients to android, vice versa
  - Displaying chats in real time from different types of clients
  - Most efficient database format (how to store information in MongoDB- collections, databases)
- Server-Admin Data Transfer
  - Difficult time finding an efficient way of sending server dictionary information to Admin client -
- Keyword Optimization
  - How to filter out relevant keywords only
- GUI Elements for Clients/Admin
  - Cleaner Look

### **Implementations for Resolving Challenges**

- Mirroring Information - MongoDB
  - Thread use when tkinter users are in chat function to display real time chats from android clients
  - Uploading MongoDB documents with the same property fields

- Central database design, with collections holding each type of data (rooms, users, messages)
- Server-Admin Data Transfer
  - String transfer
    - Putting all dictionaries into string, and splitting string into list in Admin
- Keyword Optimization
  - Testing using long texts, together and separate
  - Changing filter options such as max number of words in keyword, number of occurrences, minimum amount of characters per keyword/phrase
  - Updating blacklist to include common words, or non-keyword type words from text testing

## 5 Solution Components

---

### 5.1 [Project Disciplines](#)

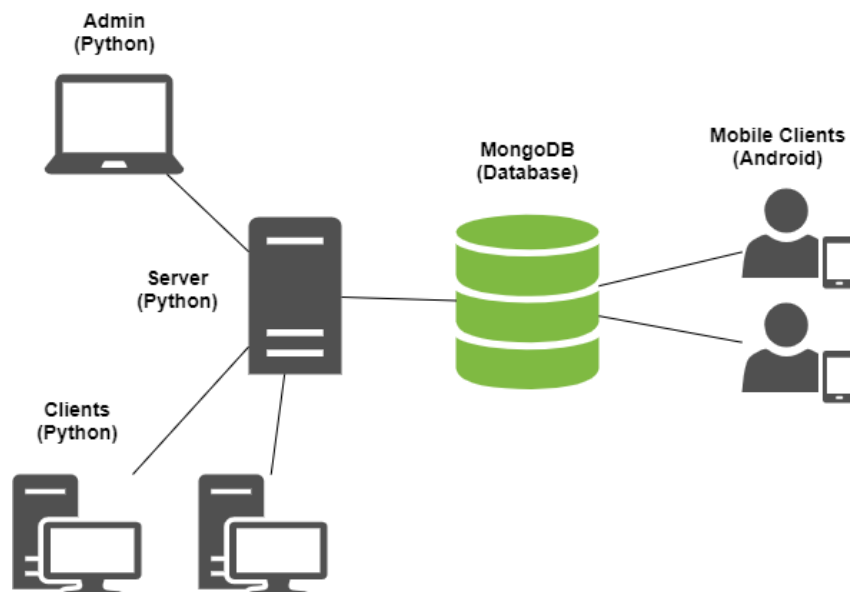
- Networking -
  - Socketing, managing Server Client Interaction
- Coding in Python
  - Libraries
  - Functions
  - Python Data Structures – List, Dictionaries
- Data Storage
  - MongoDB Database with JSON documentation
- Documentation
  - Milestone documentation
  - Code and future Updates
  - Product review definition documents and final book
- Security
  - User Authentication
- Display
  - Android application display
  - Tkinter test display

## 5.2 Environmental Requirements

- Programming Languages
  - Python 2
    - Socket Libraries
    - MongoDB driver
    - RAKE - Keyword extraction library,
      - With modified blacklist, and keyword filter values
        - # of times a given word/phrase is present in text
        - Min amount of characters
        - Max amount of words per keyword phrase (“Google Plus Application”) → 3 words
    - Tkinter for test clients
    - MongoDB driver
- Environments
  - MongoDB through MongoDB Atlas on browser
  - PyCharm

For the development of this project, I used Python because I have the most knowledge in the language and it is very practical and relatively simple. I decided to use MongoDB to store user data in order for the application to work optimally as well as provide direct connections via drivers for Python and Java. Tkinter and Socketing to test messaging and database management.

## 5.3 Topological View



The program-application is based on a central database using MongoDB. Directly connected to the MongoDB database is a server, which connected to it are Tkinter Users and Admin clients. The server sends information taken from MongoDB, room, message, keyword search data, to users and admin clients. Client functions are also located in the server code. On the other side, Android clients are connected directly to the central database. Database information (rooms, messages, users, etc) are taken directly and client functions are written in the application, instead of from a server that sends information and runs client functions

## 5.4 Data Structure Definition

Lists:

- actions\_list
  - List of action dictionaries (join\_info/user\_action) to be sent to Admin
  - Actions include users joining, leaving, creating, searching, etc.
  - (“<USERNAME> joined <ROOM NAME>”)
- active\_users\_list
  - List of active\_user dictionaries (act\_user\_info) to be sent to Admin
  - Active Users are users that are currently located in a room (but in Admin, time since room join is displayed)
    - Once a user joins a room, he is considered “active”, and a dictionary is created which includes the room the user joined and the time stamp of when the user entered, which can be viewed through admin clients.
- room\_choice\_list
  - List of Room Names that are available for the user to join
  - Depending of the number of rooms after going through room collection in MongoDB - Only up to first pulled 30 rooms (based on number of chats) will be appended to room\_choice\_list
- keywords
  - in keyword\_extract & keyword\_update functions
  - List of tuples (key word/phrase, score) from RAKE Library
  - Used for updating keyword collection in MongoDB after every chat
- available\_rooms
  - In search\_room function
  - List of available rooms a user could join based on keyword search
- clients
  - List of clientsockets for broadcasting information to all clients

- Users that connect have their clientsocket information appended to clients list and removed when they quit.

#### Dictionaries:

- kw\_main\_room\_dict
  - In admin\_keywords function
  - A dictionary where key is Room Name and the key is a dictionary of keywords from message data in the specific room (Words/Phrases: Scores) – ex: {RoomName: {Word1:Score1, Word2:Score2, ....}}
  - A kw\_main\_room\_dict is created for every room then turned into a string to be sent to Admin client(s).
- chat\_clients\_Room
  - Keys are clients-ockets and values are room names
  - When users (clients) join a room (through join or search functions) their client-socket information is added to chat\_clients\_Room dictionary with room name of the one they are entering, as the value.
  - Every message (including action messages, “<USERNAME> joined <ROOMNAME>”) will be broadcasted to every user that is also in the same room (roomname of the user is equal the room value of clientsocket in chat\_clients\_Room dictionary)
  - Users that leave a room will have their client-socket information removed from the dictionary
- join\_info / user\_action
  - User action dictionary, which is appended to actions\_list and is sent to admin client(s); and is inserted into Actions collection in MongoDB
  - (Username, Action (“Joined Room By Name”), Time)
- act\_user\_info
  - Active user dictionary, which is appended to active\_users\_list and is sent to admin and inserted into Active Users collecting in MongoDB
  - (Username, RoomName, Time Since Active)
- Actions
  - Text based function activation (one way of accessing functions) for messaging clients
  - Key is code word for function (“JOIN/CREATE/SEARCH/QUIT”), value is the function based on code word that will be executed using exec method (“JOIN”: “join\_room(clientsock,username)”)

- admin\_actions
  - Text based function activation for Admin Client(s)
  - Key is code word for function (“active/actions/keywords/rooms”), value is function based on code word that will be executed using exec method (“actions”: “admin\_actions(clientsock)”)
- intro\_msg
  - Dictionary which includes intro messages of users (“<USERNAME joined/left/created <ROOMNAME>”) that will be broadcasted to all users that are present in the room (if user joins/leaves).
  - The dictionary will be inserted into AllChats collection in MongoDB but will have the key “Intro” = 1 instead of 0, so the message will not be keyword filtered.
- user\_msg\_data
  - Dictionary which includes messaging of users, will be broadcasted to all users that are present in the room
  - The dictionary will be inserted into AllChats collection in MongoDB, intro = 0 so it will be keyword filtered.

Other (Non-Database Related):

- actives\_tot / actions\_tot / kw\_tot / num\_of\_rooms
  - Strings in which active user dictionaries, user action dictionaries, keyword dictionaries, and room dictionaries are saved and sent to Admin client(s)

## 5.5 Database Definition

Database in MongoDB includes all chats collections, active user collections, user actions collection (user joins/leaves), room collection, port collection (port number where clients can connect to server)

Main Database (“ChatIt”)

- Users
  - Username
  - Password
  - Email (Android)
  - Profile Picture (Android)
  - LoginDate/StartDate
  - UserID (Android)

- Keywords
  - Word/Phrase
  - Score
  - ChatroomName
- ActiveUsers
  - Username
  - ChatroomName
  - Time Since Active
- Actions
  - Username
  - Action
  - Time
- AllChats
  - Username
  - User\_id (Android)
  - Time
  - Delete (For synchronized chats between tkinter and android clients)
  - Chatroom\_id (Android)
  - ChatroomName
  - Intro - For differentiation between intro messages and regular messages for keyword filtration
- ChatroomList
  - ChatroomName
  - Created by – Username
  - Time created
  - MembersNum
  - MessagesNum
- Port
  - Port number for client connections

## 5.6 [Modular View](#)

### Server:

- Client Functions
  - Connect clients, execute client commands, execute other functions, synchronize clients, update client status
- Port Functions
  - Finding open ports and saving to MongoDB

- Admin Functions
  - Connect Admins, send updated server information from database and data structures to Admins.

### Client (Tkinter):

- Port Function
  - Reads the free port saved in MongoDB to connect to server.
- Send/Receive Functions
  - Send and Receive data from Server

### Admin:

- Port Function
  - Reads the free port saved in MongoDB to connect to server.
- Send/Receive Functions
  - Send and Receive data from Server
- Information Display Functions
  - Display real-time information, that is sent from server.

## 5.7 Main Modules

### Server modules:

#### Client Class

Function	Input / Output	Description
Run	Input: stop flag (if user disconnected then Stop = True, thread stops)  Output: None	Executed when thread starts, broadcasts every message from chat collection in MongoDB that has delete=False, to every client that is located in the same room as the message in order to post real time messages from android users.
create_room	Input: socket, username  Output: None	User enters room name of their choice, new room is created, with the room dictionary being inserted to room collection in MongoDB, with username being one of the fields, then chat function is executed which takes in socket
keyword_extract	Input: room name  Output: keywords/list of tuples (word/phrase, score)	Takes every message from chat collection in MongoDB that the room name of the message = inputted room name and that "intro" = 0 in the message, saves as one string and keyword filters it through rake library



keyword_update	Input: None  Output: None	For every room, executes keyword_extract to retrieve keywords; then updates keyword collection in MongoDB, with each item in keywords list.
search_room	Input: socket, username  Output: None	Executes keyword_update, If keyword collection in MongoDB is empty, the prints message to user then returns None. User then inputs keyword for search, then user input is checked in keyword collection if there is match the room of the keyword given as a choice for the user to join, and once user joins a given room, action and active user info is uploaded to MongoDB and user is redirected to chat function.
join_room	Input: socket, username  Output: None	Counts rooms in room collection in MongoDB, if number is 0, then prints create room recommendation then returns None. If number is above 25 then prints available rooms to user that have most chats. If number is below 25 then all available rooms are printed to user. Once user correctly selects room, action and active user information is uploaded to MongoDB and user is redirected to chat function.
broadcast_chat	Input: user message, username, room number  Output: None	Broadcasts user message to every socket that has the same room number value as user with time stamp and username
chat	Input: socket, username, room number, action (separate string flag)  Output: None	If user was directed from join_room or search_room (action = "Join") message that will broadcasted will be (User joined Room), else it will (User created Room); thread starts → run function is executed. User can input chat data that is uploaded to chat collection in MongoDB; keyword_update is executed; and if user quits, active user collection is updated

**Port class**

Function	Input / Output	Description
find_open_port	Input: None  Output: port number	Creates socket, binds, listens, receives port number, and closes socket
save_free_port	Input: None  Output: None	Creates port dictionary, which includes port number (saved from find_open_port to global variable), uploaded to port collection in MongoDB

**Admin class**

Function	Input / Output	Description
admin_active	Input: socket Output: None	If length of list of active user dictionaries is 0 then "0" is sent to Admin, else every active user dictionary is put in one string and sent to Admin
admin_actions	Input: socket Output: None	If length of list of user action dictionaries is 0 then "0" is sent to Admin, else every user action dictionary is put in one string and sent to Admin
admin_keywords	Input: socket Output: None	If there are no keywords in keyword collection in MongoDB, then "0" is sent to Admin; else, for every room in room collection, a dictionary with room name and dict(keywords and score) is created. All room – keyword dictionaries are saved to one string and sent to Admin
admin_rooms	Input: socket Output: None	All rooms in rooms collection in MongoDB are put in one string and sent to Admin, but if there are no rooms then "0" is sent to Admin

**Client/Admin class**

Function	Input / Output	Description
handler	Input: socket, address (Host, Port) Output: None	If Client, user inputs credentials, if inputted password matches user data in user collection in MongoDB, then access is granted; then user can select command (ex: Search, Create, Join) and the associated function is executed, Admin → same as client without authentication.

**Client**

Type	Function	Input / Output	Description
Port	read_free_port	Input: None Output: port number	Reads port number from port collection in MongoDB collection, and returns port number
Send/Receive	send	Input: cmd (=None) Output: None	Send text to Server from specific buttons in tkinter client
	receive	Input: socket Output: None	Receives messages from Server

**Admin**

Type	Function	Input / Output	Description
Port	read_free_port	Input: None Output: port number	Reads port number from port collection in MongoDB collection, and returns port number

Send/Receive	send	Input: cmd (=None) Output: None	Send text to Server from specific buttons in tkinter client
	receive	Input: socket Output: None	Receives messages from Server
Information Display	display_actions	Input: None Output: None	Receives action dictionary string from client, and displays each action dictionary from string using json.loads
	display_active_users	Input: None Output: None	Receives active user dictionary string from client, and displays each active user dictionary from string using json.loads
	display_keywords	Input: None Output: None	Receives keyword dictionary string from client, and displays each keyword dictionary from string using eval
	display_rooms	Input: None Output: None	Receives room dictionary string from client, and displays each room dictionary from string using eval

## 6 Testing Scenarios

---

### 6.1 Testing Emphases

- Multi-client connection (tkinter clients)
  - Direct Messaging
    - Proper User Search
- Multi-client connection (Android – tkinter, Android – Android)
  - Chat synchronization
    - Real time messaging
  - Direct Messaging
    - Proper User Search
- Keyword Search
  - Searching from information retrieved from MongoDB
- Keyword Filtration
  - Optimizing Keyword filtration
- Admin Tracking

## 6.2 Main Testing Scenarios

- Sending messages through android and tkinter clients to test chat synchronization and real-time messaging
- Sending large messages through keyword filtration to see results, in order to optimize general keyword filtration
  - Add to blacklist
  - Edit capture filters (Max words per phrase, number of occurrences, number of characters)
- Doing many actions (creating/joining rooms, sending messages, leaving rooms) to see if Admin client(s) are/is tracking the right information and the data transfer between messaging clients and admin client(s) is functioning properly

## 7 Reflection

### 7.1 Time Table

October (POC)	<ul style="list-style-type: none"> <li>• Basic Server to run txt file to test database management</li> <li>• Input/output of data to &amp; from MongoDB</li> </ul>
November (POC)	<ul style="list-style-type: none"> <li>• Updating MongoDB data collection – defined user DB created</li> <li>• Tkinter Client to develop basic commands for user testing</li> <li>• CREATE room, JOIN room, CHAT commands for tkinter Client</li> <li>• Server-Client connection through Socketing</li> <li>• Updating server to run multiple test clients – data structure driven (lists, dictionaries, etc)</li> </ul>
December	<ul style="list-style-type: none"> <li>• Modifying of keyword extraction library RAKE</li> <li>• SEARCH command for tkinter client – keyword search command</li> <li>• Updating MongoDB data collection – Room is one Database, which has many collections (Recent messages, Keywords, User Actions)</li> </ul>
January	<ul style="list-style-type: none"> <li>• Mock Android Client (Python) – works only with MongoDB for server interaction</li> <li>• Updating MongoDB data collection – Updates (Commands from Android Client)</li> <li>• Updating server so all data (Rooms, keywords) is driven from MongoDB – MongoDB data driven</li> <li>• JOIN, CREATE commands for mock Android Client</li> </ul>
February	<ul style="list-style-type: none"> <li>• Basic Java client</li> <li>• CHAT, SEARCH commands mock Android Client</li> <li>• ADMIN creation</li> <li>• User Search</li> </ul>
March	<ul style="list-style-type: none"> <li>• Improved GUI Android/Mock Android clients</li> <li>• Tagging/Reply system</li> </ul>
April	<ul style="list-style-type: none"> <li>• Improved GUI</li> <li>• Fixes</li> </ul>

### 7.2 Challenges and Personal Contribution

The process of designing, programming, reviewing, and fixing the project was very lengthy, and provided a new type of challenge that I had not experienced before. A long-term deadline-based computer project was a new type of project that had its own challenges.

A long term based project is much different than a short term assignment or work, in which it is very difficult for me to maintain main project objectives over a long period of time, maintaining a deadline for each feature wasn't simple, and constant improvements and changes need to be thought and executed over long term without changing main project goals.

The main challenges and personal contributions from this project were:

- Maintaining main project objectives / constant changes
  - Going over original idea concepts and aligning them with recent changes in code, design, etc.
- Deadline management
  - Setting basic schedules, for various fixes
- Dual Interface Work
  - Setting meetings about command/database design fir Android and Python interfaces

### 7.3 [Insights](#)

The construction of this project assisted me in learning more about computer concepts I had previous knowledge about, as well as new fields. Outside of computer theory, this project also helped me learn how to maintain long term project goals, properly examine and change the project during its design and construction.

I chose this topic since I thought would be very beneficiary and practical to the students and teachers around me. However, I was concerned that the main concept that I came up with would drastically and negatively change through time, and would not meet original project goals; however, the end result of the project turned out to become a general vision of my goals from the start, with some changes during the process.

## 8 Instructions for Installation and Operation

---

### 8.1 [Configuration and Prerequisites](#)

For the application to run, user needs to download the application, and the server needs to run from a different system that has Python 2 downloaded and installed as well as the libraries that are called and used in the server.

### 8.2 [Installation](#)

Python, Installation of Python 2, and associated libraries with server and client programs.

Android, Installation of application.

## 9 Bibliography

---

Bolton, N. (2017, November 21). What Are the Differences Between Chat Rooms & Forums? Retrieved from <https://smallbusiness.chron.com/differences-between-chat-rooms-forums-71296.html>.

Medelyan, A. (n.d.). NLP keyword extraction tutorial with RAKE and Maui. Retrieved from <https://www.airpair.com/nlp/keyword-extraction-tutorial>.

Naskar, A. (2018, September 27). Automatic Keyword extraction using RAKE in Python. Retrieved from <https://www.thinkinfi.com/2018/09/keyword-extraction-using-rake-in-python.html>.

Python MongoDB. (n.d.). Retrieved from [https://www.w3schools.com/python/python\\_mongodb\\_getstarted.asp](https://www.w3schools.com/python/python_mongodb_getstarted.asp).

What is MongoDB? Introduction, Architecture, Features & Example. (n.d.). Retrieved from <https://www.guru99.com/what-is-mongodb.html>.

Why Classting? (n.d.). Retrieved from <https://support.classting.com/hc/en-us/articles/115005712847-Why-Classting->.

Widman, J. (2019, December 21). What is Reddit? A Beginner's Guide to the Front Page of the Internet. Retrieved from <https://www.digitaltrends.com/social-media/what-is-reddit/>

## 10 Appendix

---