**נושא פרויקט סוף שנה כיתה י"א**

**מגיש: עומר כהן**

**מורה: ערן בינט**

**תאריך הגשה: 13.5.22**

****

תוכן עניינים

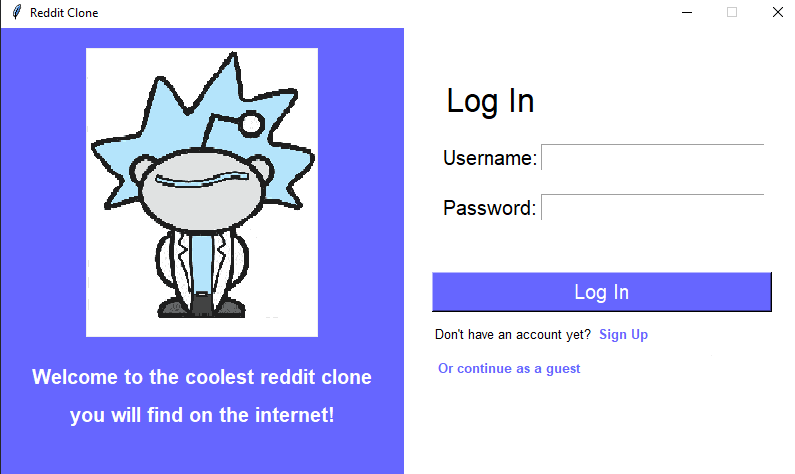
Contents

[**מבוא** 4](#_Toc103364623)

[**יישומי הפרויקט** 5](#_Toc103364624)

[**תיאור תרחישים עיקריים** 6](#_Toc103364625)

[**רישום וכניסת משתמש** 6](#_Toc103364626)

[ 6](#_Toc103364627)

[**מסך ראשי** 7](#_Toc103364628)

[**מסך בתוך חדר דיון** 8](#_Toc103364629)

[**הוספת הודעה לחדר** 10](#_Toc103364630)

[**הגדלת הודעה** 11](#_Toc103364631)

[**יצירת חדר חדש** 11](#_Toc103364632)

[**הצטרפות לחדר** 12](#_Toc103364633)

[**חיפוש על פי keyword** 12](#_Toc103364634)

[**תפריטים מיוחדים של מנהל מערכת** 13](#_Toc103364635)

[**תצוגת כל החדרים הקיימים** 13](#_Toc103364636)

[**תצוגת כל משתמשים מחוברים** 14](#_Toc103364637)

[**שינוי תאריך על מנת ליצור מציאות מואצת** 14](#_Toc103364638)

[**הסרת מילת קישור מכל החדרים** 15](#_Toc103364639)

[**צפיית כל המשתמשים בחדר ספציפי** 15](#_Toc103364640)

[**רשימת מילות קישור של חדר** 16](#_Toc103364641)

[**סביבת העבודה** 17](#_Toc103364642)

[**מבני הנתונים** 17](#_Toc103364643)

[**Chatroom** 17](#_Toc103364644)

[**message** 18](#_Toc103364645)

[**user** 18](#_Toc103364646)

[**מסד הנתונים** 18](#_Toc103364647)

[**קבצי הפרויקט** 19](#_Toc103364648)

[**הנחות** 20](#_Toc103364649)

[**פרוטוקול התקשורת המרכזי** 20](#_Toc103364650)

[**אלגוריתמים עיקריים –** 21](#_Toc103364651)

[**רישום וכניסת משתמש** 21](#_Toc103364652)

[**קבלת חדר ממשתמש ויצירתו** 22](#_Toc103364653)

[**שליחת חדר למשתמש** 22](#_Toc103364654)

[**קבלת הודעה חדשה מהמשתמש** 23](#_Toc103364655)

[**ניתוח טקסטואלי באמצעות Rake** 24](#_Toc103364656)

[**התקנה והרצה** 25](#_Toc103364657)

# **מבוא**

מטרת הפרויקט היא חיקוי של Reddit, פלטפורמה של שיתוף פוסטים בחדרי צ'אט שונים, ניהול דיונים, העלת תמונות, חיפוש חדרים על פי keyword וקבלת תשובה על פי score דבר המאפשר ניהול שיחות בצ'אטים מתאימים. למנהלי מערכת יש אפשרות לחסום משתמשים, להוריד keywords מחדרים, לחסום keywords בכל האפליקציה ואף לשנות את תאריך האפליקציה.

הפרויקט ממומש בשפת פייטון, GUI נעשה באמצעות tkinter וPIL.

התכנית משתמשת בספריות הבאות –

* Datetime
* Functools
* Operator
* Tkinter
* Tkcalendar
* PIL
* Pickle
* Time
* \_thread
* Select
* Os
* Warning
* Ssl
* Socket
* Redis
* Rake
* Re
* Rake\_nltk

# **יישומי הפרויקט**

המשתמשים בפרויקט הינם הלקוח (user\_controller) אשר יכול להיות רשום או לא רשום ומשתמש מנהל(admin\_controller). החלק האחרון הוא השרת אשר אינו נחשב כמשתמש אך מטפל בבקשות מצד משתמשים ומנהלים, מנהל חדרים, חישוב score למילים, ניהול הודעות ותאריכים ושמירת log אירועים.

היישום מאפשר שיתוף תמונות או דיון בפורמט של חדרי דיון בהם כל משתמש יכול לשתף טקסט ותמונה. משתמשים יכולים לחפש חדרים על פי keyword, שם חדר או ID. משתמש גם יכול ליצור חדר דיון חדש אם לא ימצא חדר דיון אשר מתקשר לנושא שלו. משתמש גם יכול לבחור לא להירשם לאפליקציה ולהמשיך כאורח, במקרה כזה יוכל המשתמש לצפות בחדרי דיון נוכחיים ולהשתמש בפונקציות חיפוש חדרים אשר ניתנות גם למשתמש רגיל, אך אורח לא יכול לשתף טקסטים ותמונות בחדרים שונים ונמצא על תקן "spectator".

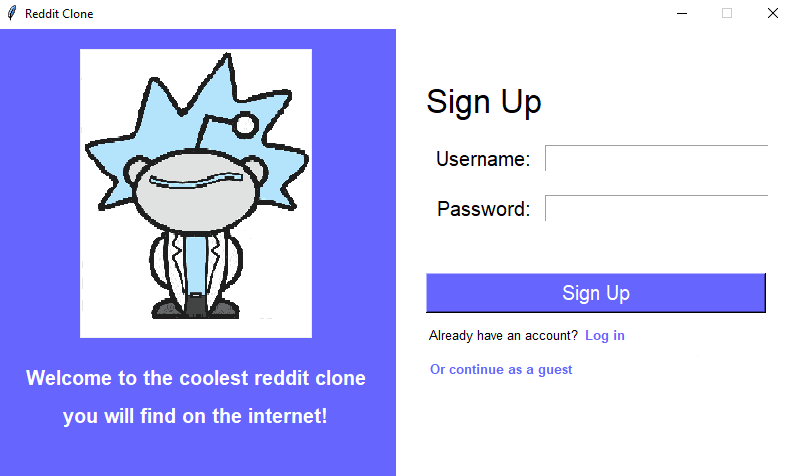
למנהל מערכת יש מספר אופציות מעבר לשיתוף פוסטים ויצירת חדרים. מנהל מערכת יכול לראות את רשימת כל החדרים הקיימים כרגע ומידע עדכני לגביהם ובכך יכול לבקר על החדרים. הוא יכול לראות את כל המשתמשים המחוברים כרגע, באיזה חדר נמצאים וכמה זמן הם מחוברים. מנהל מערכת יכול גם לקבל מידע לגבי חדר ספציפי כגון כל המשתמשים אשר הצטרפו לחדר זה ומילות הקישור שלו, המנהל יכול להסיר מילים מרשימת מילות הקישור של החדר וגם לחסום משתמשים מרשימת המשתמשים של החדר ובכך הוא מקבל שליטה על כל התעבורה אשר עוברת בחדר. בנוסף יכול מנהל לחסום מילות קישור בהקשר של כל החדרים ובכך לא לאפשר למילים מסוימות לדוגמה מילים גסות, בעלות מטען פוליטי וכו' לקחת חלק כנושאים של חדרים.

# **תיאור תרחישים עיקריים**

## **רישום וכניסת משתמש**

כניסת משתמש רשום -

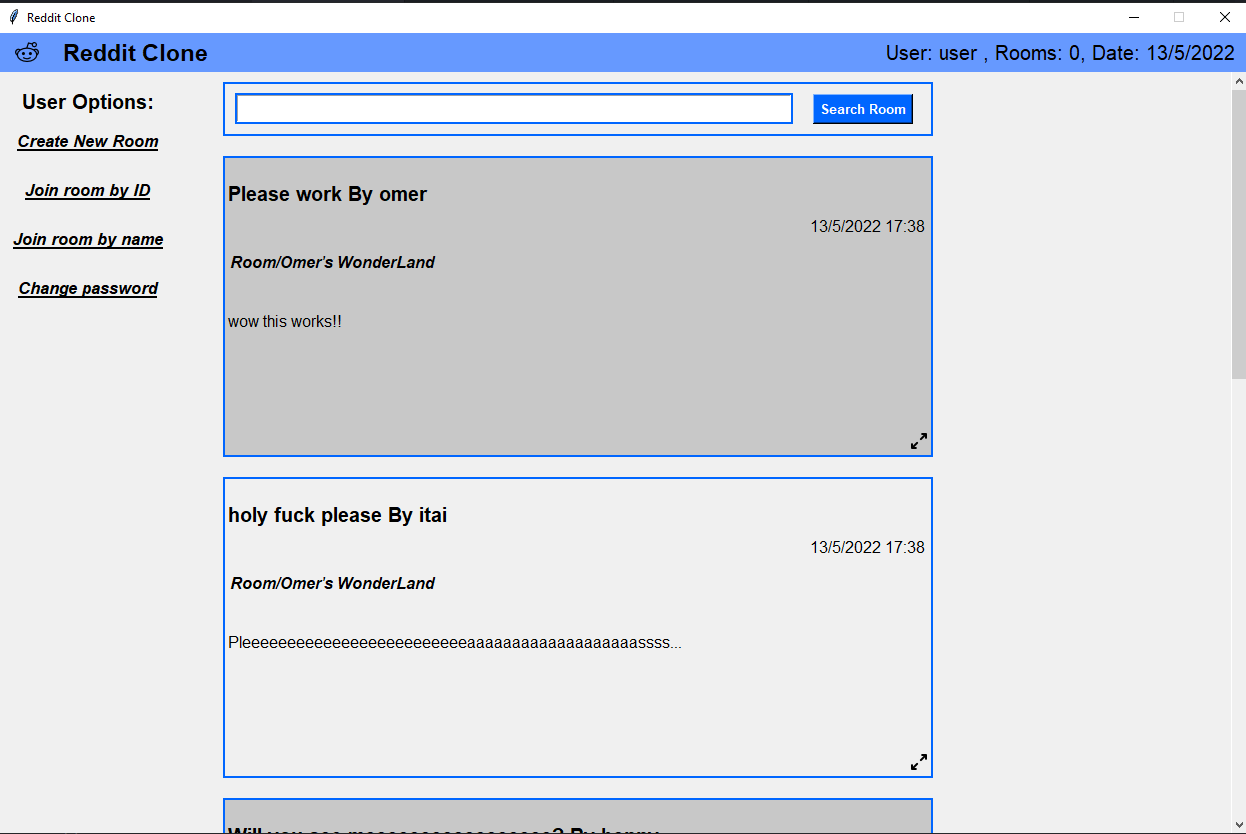
# 

רישום משתמש חדש

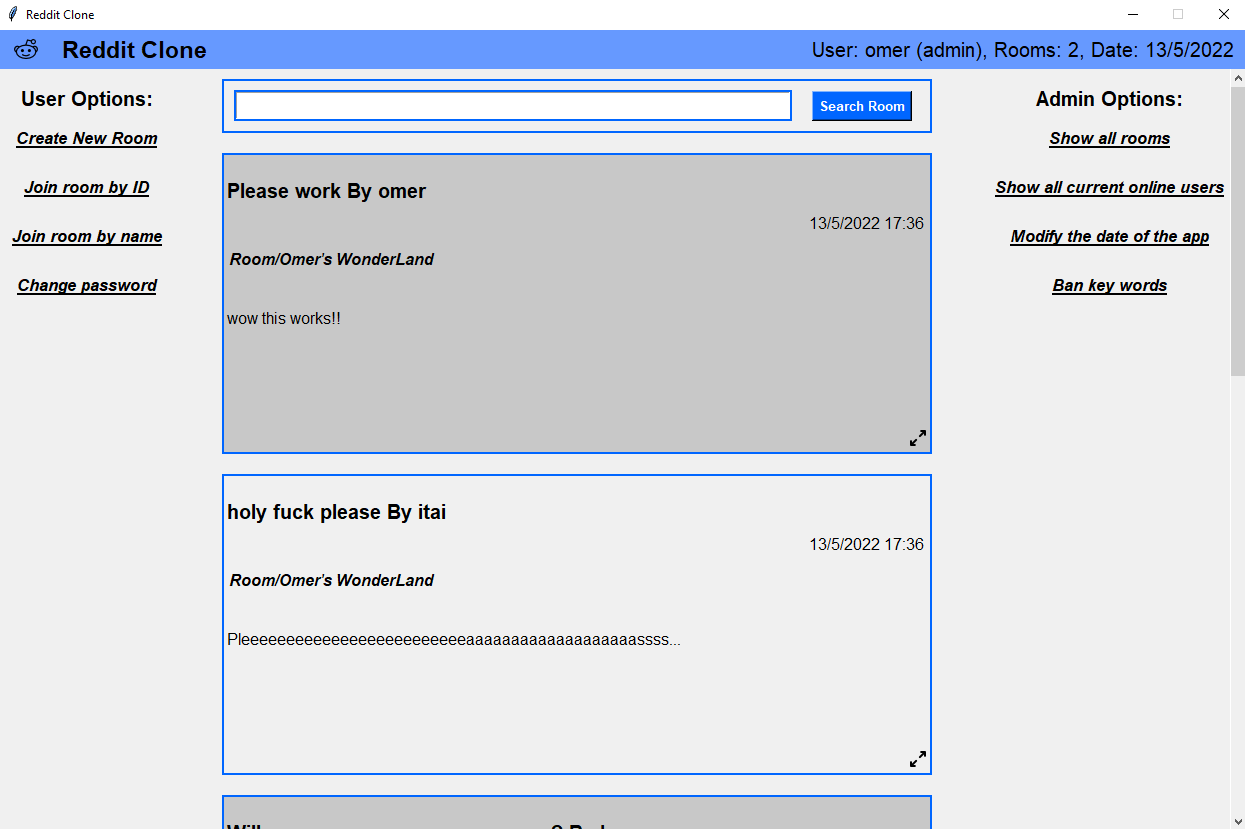
משתמש חדש יכול להמשיך כאורח – continue as a guest או להירשם דרך sign up ולאחר מכן להיכנס דרך הlog in. בשביל משתמשים רשומים כל מה שנדרש זה להכניס שם משתמש וסיסמה.

## **מסך ראשי**

מסך ראשי בשביל user



מסך ראשי בשביל admin

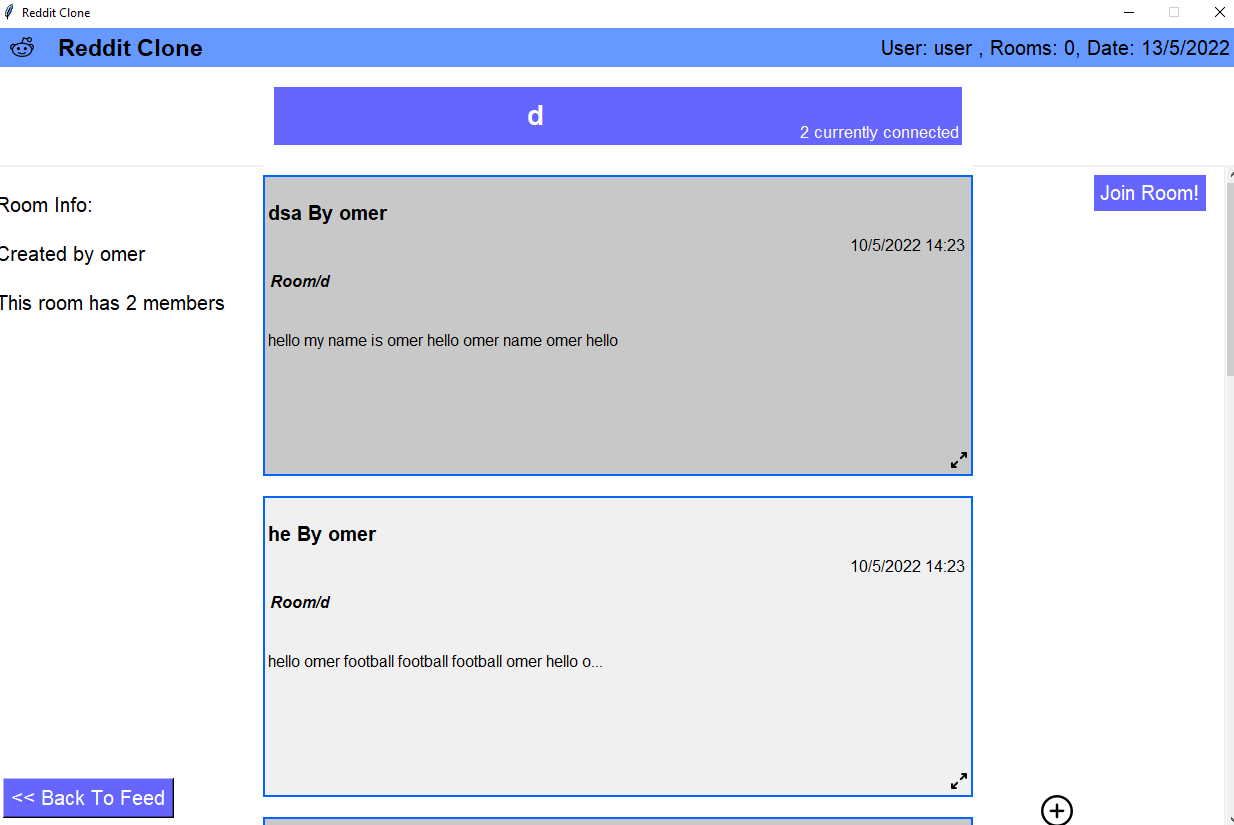


תפריט האפשרויות של משתמש רגיל מופיע מצד שמאל ושם הוא יכול ליצור חדרים חדשים, להצטרף לחדר וגם לשנות סיסמה. במרכז המסך מופיע הfeed שם מופיעות הודעות ראנדומליות מחדרים שונים שהמשתמש יכול לעיין בהם על מנת שיוכל לבחור לאיזה חדר יכול הוא להצטרף. מצד ימין מופיע התפריט של מנהל שם הוא יכול לבצע פעולות כמו לשנות את תאריך האפליקציה על מנת ליצור מציאות מואצת, לראות את כל החדרים הנוכחים, לראות את כל המשתמשים הנוכחים ולחסום מילות קשר מכל החדרים הקיימים. מידע על מצב האפליקציה מופיע בתפריט העליון שם כתוב את שם המשתמש, מספר החדרים אליהם הוא שייך, תאריך האפליקציה בהתאם למציאות המואצת וגם כפתור חזרה לתפריט הראשי.

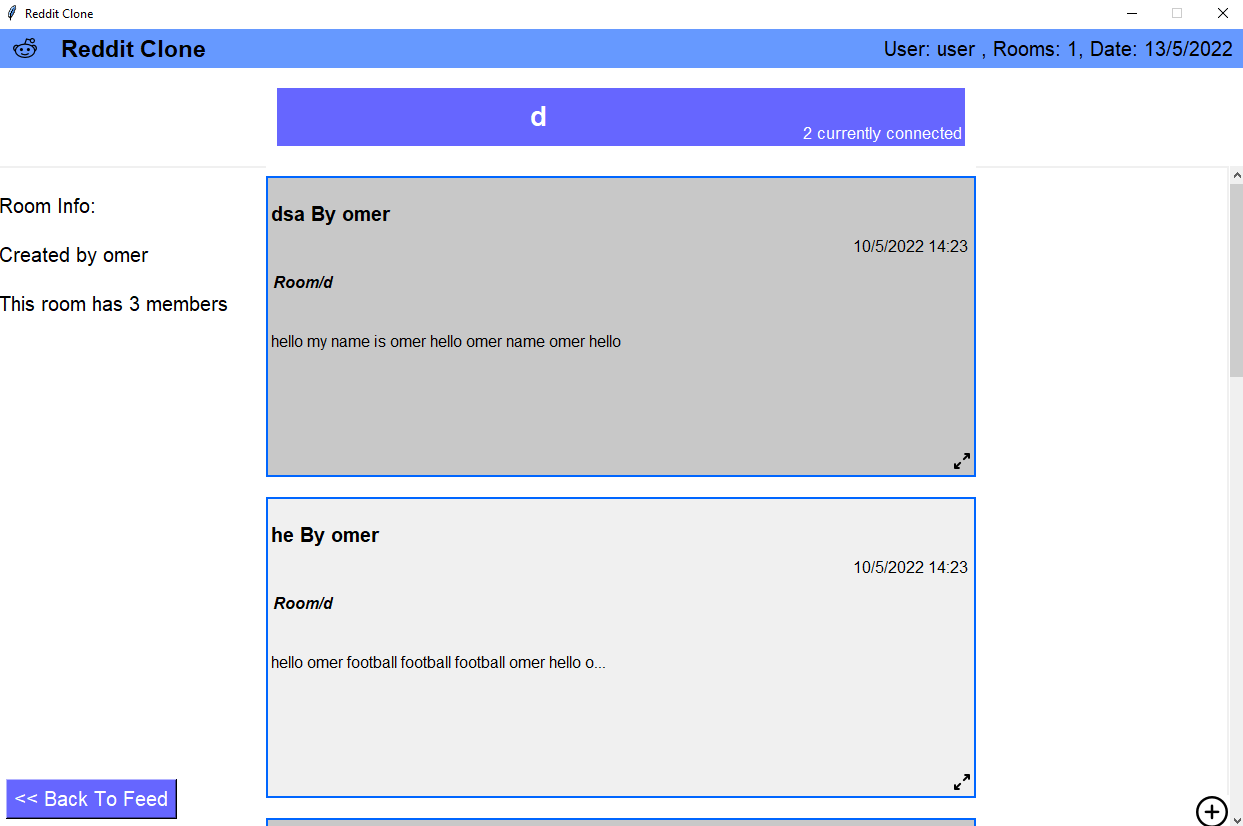
מעל הfeed מופיע הsearch bar אפשר מאפשר לחפש חדרים על פי keyword שונות.

## **מסך בתוך חדר דיון**

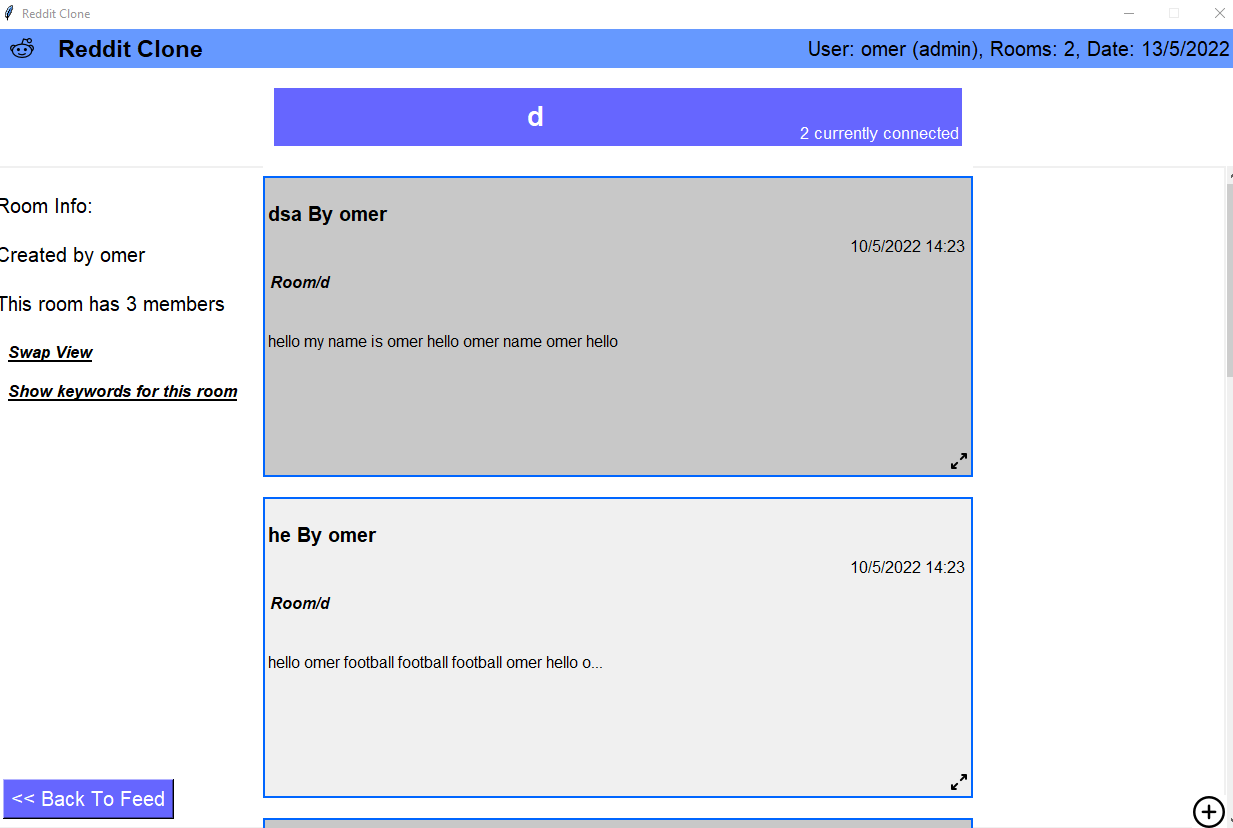
מסך של משתמש רגיל אשר לא רשום לחדר



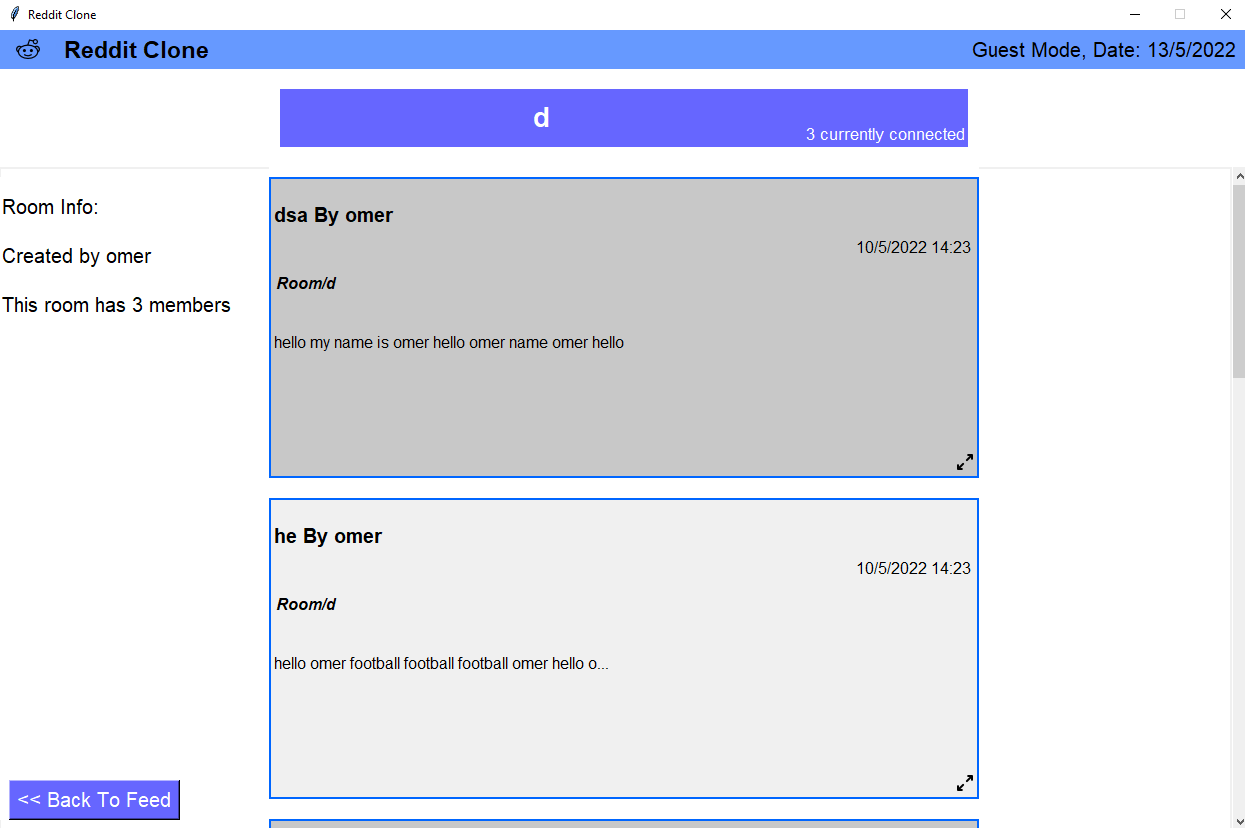
מסך של משתמש רגיל רשום לחדר



מסך של מנהל

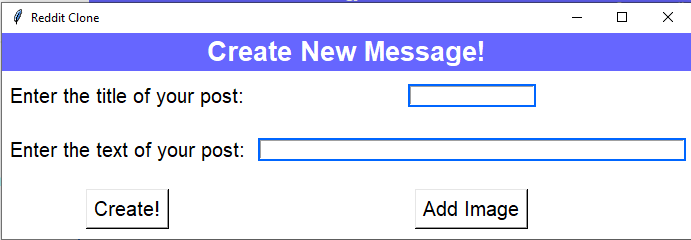


מסך של אורח



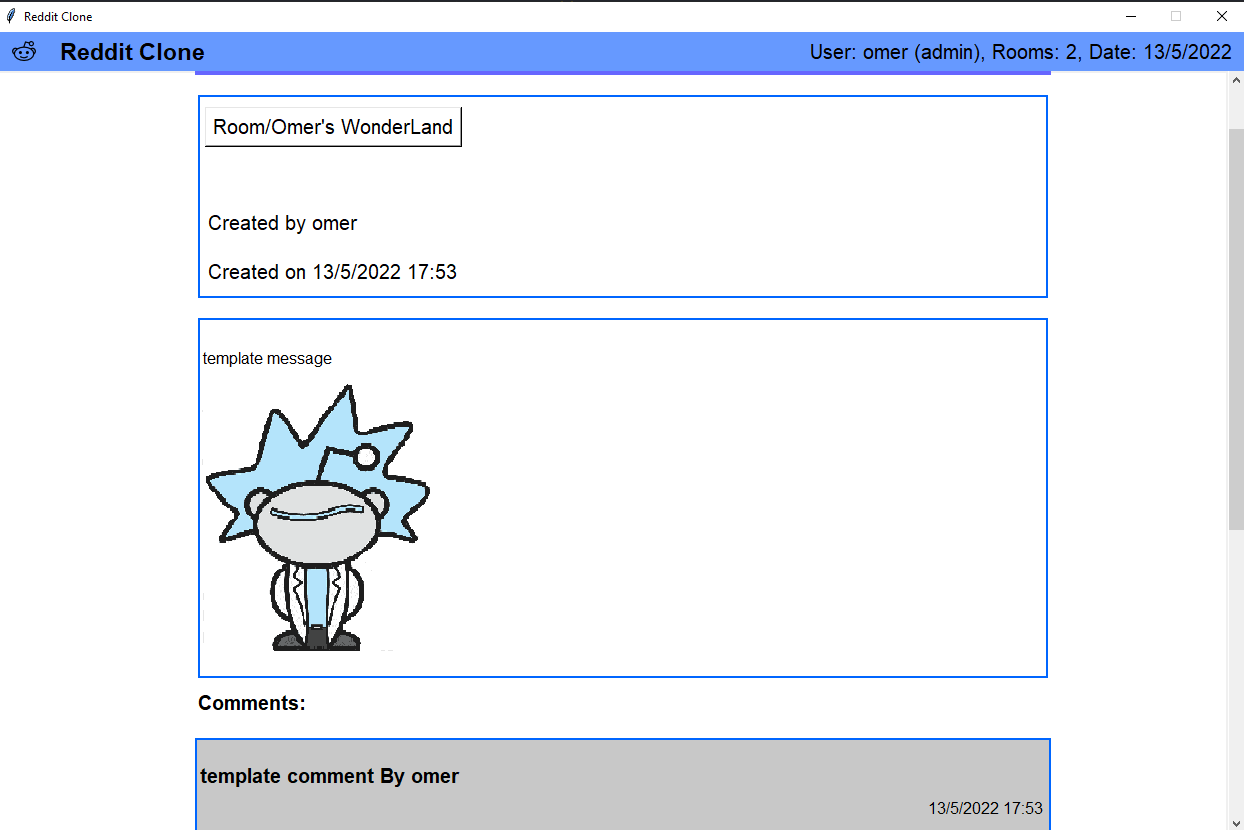
ניתן לראות כי למשתמש רגיל אשר לא רשום לחדר יש אפשרות להצטרף לחדר על ידי כפתור הjoin אשר יחשיב אותו כחלק מהמשתמשים הרשומים לחדר. משתמשים רגילים יכולים להוסיף הודעות בחדר ולראות הודעות קיימות, מנהלי מערכת יכולים בנוסף לראות את כל המשתמשים אשר נרשמו לחדר הזה על ידי הכפתור swap view או לראות את כל הkeywords של החדר ולשנות אותן דרך הכפתור show keywords for this room. משתמש אורח יכול לראות את כל ההודעות אשר נשלחו בחדר אך אין לו את הכפתור בצד שמאל למטה כמו לשאר המשתמשים אשר מאפשר להוסיף הודעות לחדר.

## **הוספת הודעה לחדר**



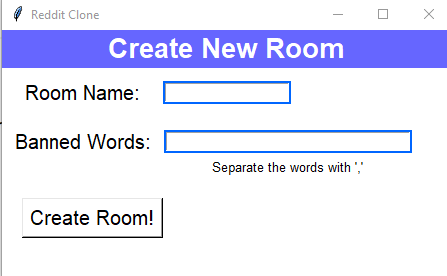
כאשר משתמש רוצה להוסיף הודעה אין הבדל בין משתמש רגיל למנהל מערכת. המשתמש מקבל אפשרות לבחור כותרת להודעה, תוכן וגם תמונה.

## **הגדלת הודעה**



משתמשים יכולים לעשות expand להודעה וכך לראות את התוכן המלא שלה במקרה וכתוב בה יותר מ60 אותיות, מידע על ההודעה וגם את התגובות להודעה זו. אין הבדל בתפריט זה בין מנהל ומשתמש רגיל.

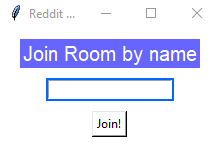
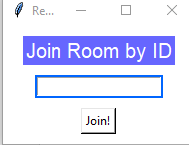
## **יצירת חדר חדש**



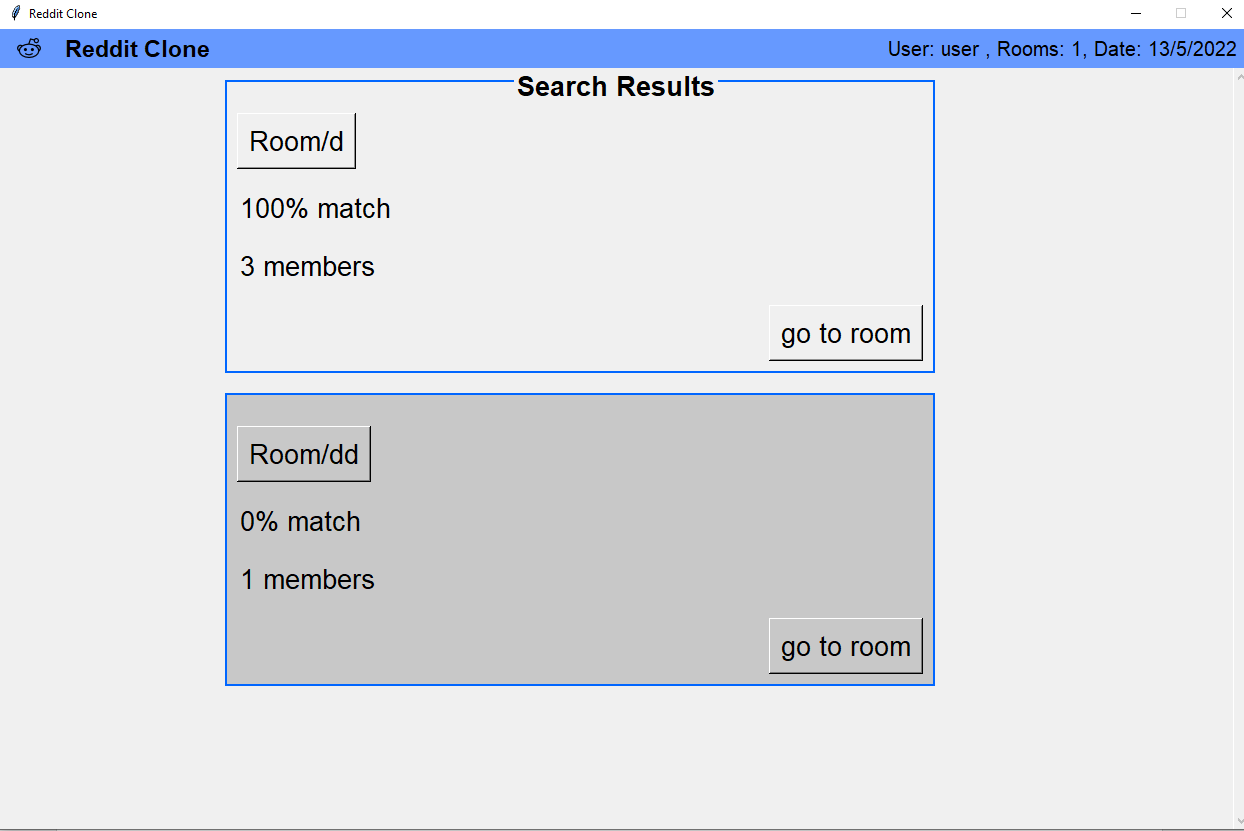
זהו תפריט יצירת חדר חדש בו משתמש מכניס את שם החדר ומילים אסורות בחדר שלו. אין הבדל בתפריט זה בין משתמש רגיל ומנהל.

## **הצטרפות לחדר**

מלבד לחיצה על השם של החדר אשר ייקח את המשתמש לחדר המבוקש. משתמשים יכולים להגיע לחדר דרך ID של החדר או השם שלו



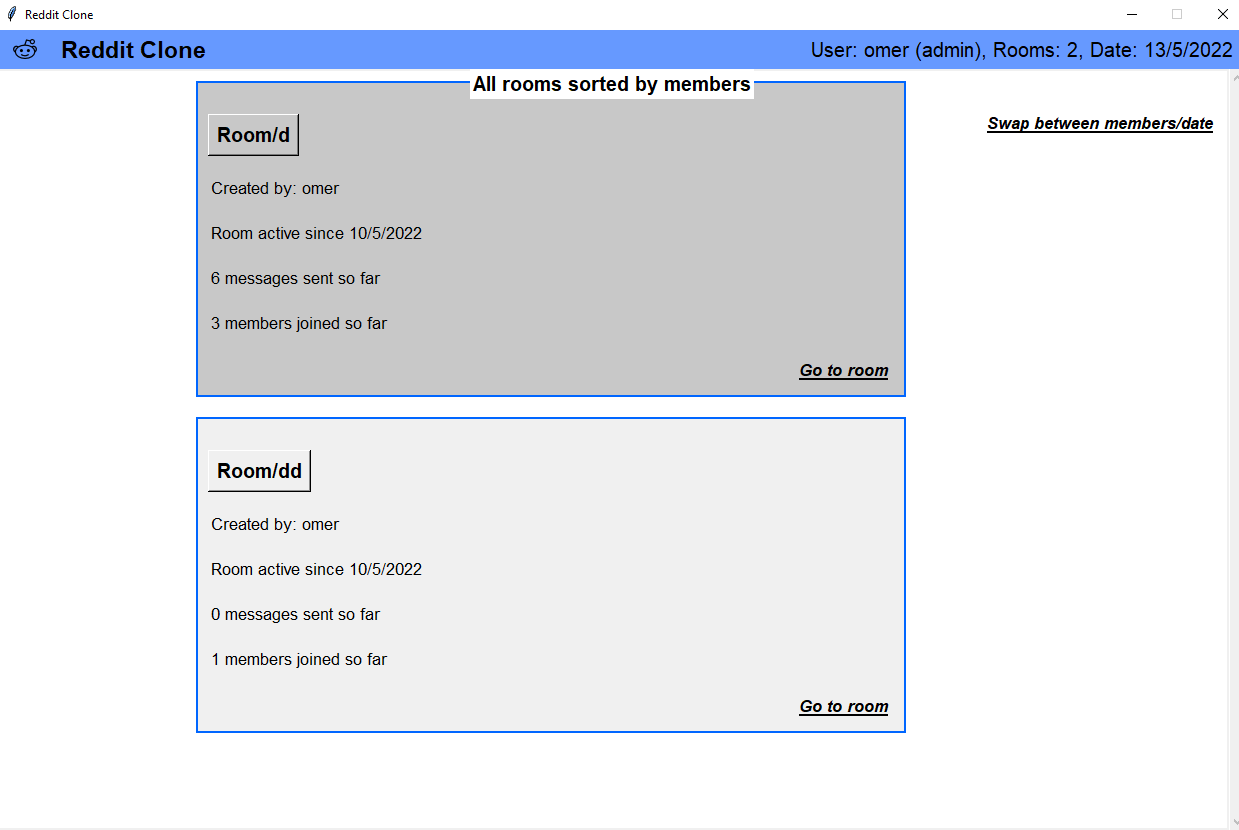
## **חיפוש על פי keyword**



משתמש יכול לחפש חדרים על פי keyword ולקבל רשימה של חדרים אשר ממויינים על פי דרגת התאמה.

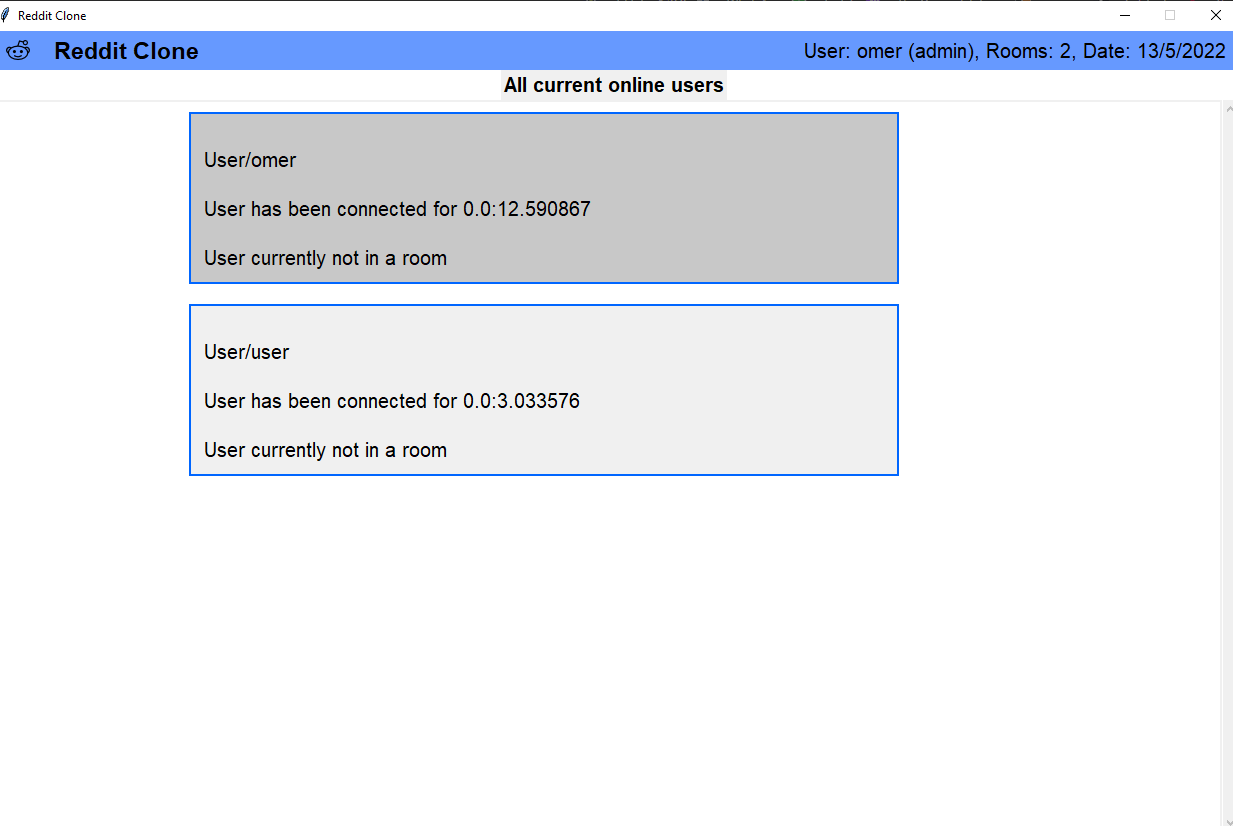
## **תפריטים מיוחדים של מנהל מערכת**

### **תצוגת כל החדרים הקיימים**



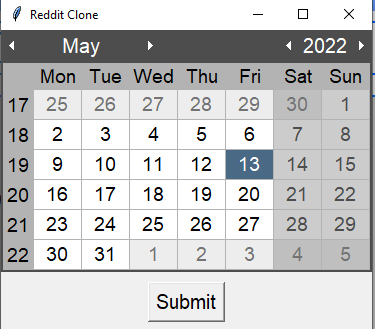
על ידי לחיצת הכפתור show all rooms בתפריט הראשי מנהל יכול להגיע לרשימה הזו אשר מציגה לו את כל החדרים ממויינים על פי כמות משתמשים או תאריך לפי בחירת המנהל, הוא יכול לשנות מיון זה על ידי לחיצה על הכפתור swap between members\date.

### **תצוגת כל משתמשים מחוברים**

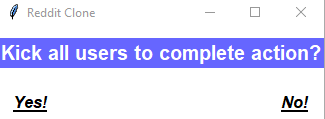


על ידי לחיצה על כפתור show all current online users מנהל מערכת יכול לקבל רשימה של כל המשתמשים אשר מחוברים לאפליקציה ברגע נתון, הוא מקבל מידע עליהם כגון באיזה חדר הם נמצאים כרגע וכמה זמן הם כבר מחוברים

### **שינוי תאריך על מנת ליצור מציאות מואצת**



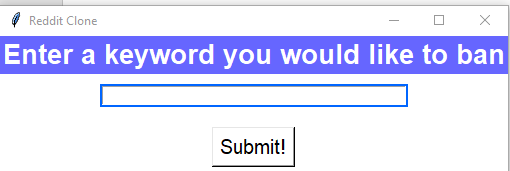
על ידי לחיצה על כפתור modify the date of the app מהתפריט הראשי יכול מנהל לפתוח את החלון הזה אשר מאפשר לו לשנות את תהליך האפליקציה לתאריך אחר (כמובן רק בעתיד). השרת יודיע לו כי על מנת לעשות תהליך זה הוא חייב להעיף את כל המשתמשים הפעילים כרגע ולהכריח אותם לפתוח מחדש את האפליקציה. השרת יודיע לו זאת כך:



אם ייבחר כן יקרה מה שתואר לעיל, במקרה ולא, יתבטל התהליך והכל ימשיך כשורה

### **הסרת מילת קישור מכל החדרים**

מנהל יכול להיכנס לחדר ספציפי ולהוריד ממנו מילות קשר שנראות לו לנכון אבל כאשר ישנה מילת קישור אשר הוא רוצה להוריד מכל החדרים בפעם אחת הוא יכול לעשות זאת מהתפריט הראשי דרך לחיצה על הכפתור ban key words אשר ייפתח את החלון



ממנו יכול המנהל להכניס מילות קישור אשר לא יופיעו יותר באף חדר.

### **צפיית כל המשתמשים בחדר ספציפי**

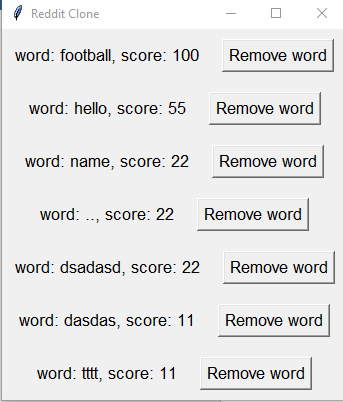
בתוך חדר מסוים מנהל יכול ללחוץ על הכפתור swap view וכך הוא עושה toggle בין לראות את כל ההודעות שנשלחו בחדר כמו משתמש רגיל ולראות את כל המשתמשים הרשומים לחדר:



המנהל יכול לחסום או להזהיר משתמשים על בסיס התנהגות לא נאותה וכו'

### **רשימת מילות קישור של חדר**

על ידי לחיצה על הכפתור show keywords for this room ייפתח למנהל החלון



אשר מכיל את מילות הקישור הפופלאריות ביותר בחדר ואת הscore היחסי שלהן. המנהל יכול להסיר מילה ואז היא תמחק מהרשימה והscore יחושב מחדש.

# **סביבת העבודה**

## **מבני הנתונים**

ארגנתי את הפרויקט שלי למספר מבני נתונים אשר שמורים בתיקיית classes על מנת לארגן את כל הנתונים כמה שאפשר –

## **Chatroom**

מבנה נתונים זה מייצג את החדר צ'אט, הינה המימוש שלו:

*class* chatroom:

*def* \_\_init\_\_(*self*,*creator*:User,*name*:str, *room\_id*:int, *banned\_words*):

        self.creator = creator

        self.name = name

        self.current\_members = 0

        self.members = [creator]

        self.room\_id = room\_id

        self.common\_words = {}

        self.removed\_common\_words = []

        self.banned\_words = banned\_words

        self.msgs = []

        self.last\_sent\_time = datetime.datetime.now()

        self.create\_time = datetime.datetime.now()

*def* *kick*(*self*,*user*:User) ->bool:

*try*:

*del* self.members[self.members.index(user)]

*return* True

*except*:

*return* False

*def* *add\_user*(*self*,*user*:str) ->bool:

*try*:

            self.members.append(user)

*return* True

*except*:

*return* False

*def* *add\_msg*(*self*,*msg*):

        self.msgs.append(msg)

מבנה נתונים זה מארגן לי באותו object את כל ההודעות אשר נשלחו בחדר, כל המשתמשים, תאריך בו נוצר ותאריך מאז ההודעה האחרונה של החדר, יוצר החדר וכו'. יש לobject זה מספר מועט של פונקציות מכיוון שכאמור רוב הלוגיקה מתבצעת בשרת, הוא יודע רק להוסיף הודעה לחדר, להעיף משתמש ולהוסיף משתמש.

## **message**

מבנה נתונים זה מייצג הודעה ומכיל את החדר בו נשלחה, המשתמש אשר שלח אותה, תוכן, תמונה וכו'. אין למבנה נתונים זה פונקציות נוספות מלבד add\_comment כיוון שלאחר יצירת הודעה אין צורך לשנות אותה או לשחק עם נתוניה בתוך הclass

*class* message:

*def* \_\_init\_\_(*self*,*sent\_by*:str,*msg*:str,*sent\_in*:chatroom,*img\_name*:str,*title*:str):

        self.sent\_by = sent\_by

        self.comments = [] *#expecting message class as comments*

        self.msg = msg

        self.sent\_in = sent\_in

        x = datetime.datetime.now()

        self.time\_str = *f*"{x.day}/{x.month}/{x.year} {x.hour}:{x.minute}"

        self.img\_name = img\_name

        self.title = title

*def* *add\_comment*(*self*,*msg*):

        self.comments.append(msg)

## **user**

מבנה נתונים זה מייצג משתמש רגיל ומכיל את החדרים בו נמצא, האם הוא מנהל וסיסמה.

גם במבנה נתונים זה אין פעולות פנימיות כיוון שאין צורך לשנות את נתוניו בדרכים מעבר להוספת איבר לרשימה וכו' דברים הנעשים מהשרת.

*class* User():

*def* \_\_init\_\_(*self*,*name*:str,*password*:str,*is\_sys\_admin*:bool):

        self.joined\_room = []

        self.admin\_in = []

        self.name = name

        self.password = password

        self.is\_sys\_admin = is\_sys\_admin

## **מסד הנתונים**

בפרויקט זה החלטתי לעבוד עם מסד נתונים redis אשר הינו nosql. מסד זה עובד בשיטת key – value ולכן בשיטה זו אני הייתי צריך לעבוד עם המון מבני נתונים כפי שתואר בsection הקודם. את הobjects שמרתי באמצעות serialization של pickle ועל מנת לא לדרוס שום מידע עבדתי עם מספר מוסכמות –

- הkey של חדרים יהיה הID שלהם ולכל חדר חייב להיות ID ייחודי, שמרתי גם מילון אשר ממיר שם לID לשם לוגיסטיקה ותכונות אחרות.

- העצם של משתמש יישמר עם key שיהיה שמו, כך למשתמשים אסור שיהיה אותו שם.

- ישנם מספר מילים שמורות אשר אשר למשתמשים להשתמש בהן כשם כיוון שהן שומרות מידע אחר.

כיוון שעבדתי עם מוסכמות אלה לא הייתה בעיה של overlapping כאשר ניסיתי לשמור מידע וכך העבודה עם מסד נתונים גמיש הייתה נוחה מאוד והרבה יותר משחררת מאשר sql רגיל, כאשר הייתי צריך דרך חדשה לשמור נתונים כל מה שהייתי צריך לעשות זה לקחת חמש דקות ליצור עצם חדש.

## **קבצי הפרויקט**

קבצי מבני הנתונים אשר שמורים בתיקייה classes –

chatroom.py

message.py

server.py

user.py

תיקיית program\_pics – מכילה את כל התמונות אשר משתמשת בהם האפליקציה

תיקיות server\_pics\pictures מכילות את התמונות אשר בהן משתמשות האפליקציות שרת\לקוח לפי מה שמתאים. בתיקיות אלה חל תהליך cache בו בודק המשתמש האם יש לו בתיקייה שלו את התמונה המבוקשת ורק אם אין לו תמונה שהוא צריך הוא מבקש אותה מהשרת.

קובץ מנהל –

admin\_controller.py

קובץ משתמש –

user\_controller.py

קובץ שרת –

send\_receive\_server.py

קובץ log –

log.txt

קובץ UI –

ui\_reddit.py

קובץ לניתוח טקסט של rake-

Stop.txt

קובץ לדבר עם מסד הנתונים –

Model.py

אבטחת SSL –

Key.pem

Cert.pem

**הנחות** –

* כל הקבצים נמצאים בתיקיית הריצה כפי שתואר לעיל.
* הפורט של השרת ידוע ונמצא ב50000, פורט הלקוח משתנה.
* הIP של השרת והלקוח כתובים סטטית בתכנית וידועים מראש.
* ישנו שרת redis שרץ locally בצד של השרת.

## **פרוטוקול התקשורת המרכזי**

אני עובד גם עם TCP וגם עם UDP בו זמנית בצורה הבאה.

מירב התקשורת מתרחשת בTCP ושם המשתמש שולח בקשות לשרת, השרת מחכה לבקשות וכל התקשורת ה"מתוכננת" כלומר שהלקוח יוזם ויודע שהולכת לקרות מתרחשת שם.

אך כאשר ישנה תקשורת שהשרת יוזם היא תשלח על UDP מהסיבות הבאות –

- אי אפשר לעשות שימוש כפול של הsocket בו זמנית – אם יפעלו שני תהליכים על אותו socket תהליך אחד ייקח את המידע לתהליך השני והתכנית תקפא.

- מירב התקשורת שנשלחת בUDP אינה דחופה והיא כוללת הודעות על refresh ולכן גם אם הודעה אחת תיפול, האפליקציה עדיין תעבוד ותגיע עוד הדעה באיזשהי נקודה.

תקשורת הTCP מתרחשת בצורה הבאה –

- השרת מחכה בלולאה אינסופית עם select להודעות, לאחר שמתקבלת הודעה הוא מזהה את סוג הבקשה ומעביר אותה לפונקציה הדרושה.

- כיוון שהלקוח לא "מפחד" מלקבל הודעה לא צפויה כיוון שהוא יוזם את כל התקשורת TCP אין צורך בTHREAD שיחכה להודעות, הuser controller שולח הודעה אשר מודיעה על בקשה, מתרחשת שיחה, הלקוח שקט (בTCP) עד שירצה לבצע עוד בקשה.

תקשורת הUDP מתרחשת בצורה הבאה –

- הלקוח מעביר את הפורט בו הוא מעוניין לבצע תקשורת UDP לאחר שהוא מתחבר.

- יש THREAD שמחכה להודעות UDP כל 5 שניות (על מנת לחסוך במשאבים כי ההודעות שיעברו שם אינן דחופות)

- במקרה ובו צריך לעשות ריפרש לחדר שולח השרת הודעה לכל הלקוחות בחדר בUDP.

# **אלגוריתמים עיקריים –**

אני אציג כאן בעיקר אלגוריתמים מהשרת כיוון שבשרת רוב הלוגיקה, המשתמש שולח בקשה לדוגמה ליצירת חדר או קבלת תמונה וכו' והשרת מחזיר לי עצם העונה על הבקשה אך רוב המחשוב קורה אצל השרת. אצל המשתמשים זה בעיקר שליחת וקבלת הודעות

## **רישום וכניסת משתמש**

*def* *get\_new\_user\_data*(*self*,*sock*:socket) ->None:

        sock.send(pickle.dumps('waiting for data'))

        user\_info = pickle.loads(sock.recv(1054))

        sock.send(pickle.dumps("ok"))

        is\_ok = self.create\_new\_user\_from\_lst(user\_info)

        print(*f*"server said {is\_ok}")

        sock.send(pickle.dumps(is\_ok))

*def* *check\_log\_in*(*self*,*sock*:socket) -> None:

        sock.send(pickle.dumps("waiting for data"))

        user\_info\_lst = pickle.loads(sock.recv(1054))

        name = user\_info\_lst[0]

        password = user\_info\_lst[1]

        sock.send(pickle.dumps("ok"))

        end = pickle.loads(sock.recv(1054))

*if*(end*==*"done"):

            print("in end if")

            check = self.db\_conn.get\_user(name)

*if*(check*==*"key didn't have a value"):

                sock.send(pickle.dumps(("no username","")))

*elif*(password *==* check.password):

*if* name *in* self.banned\_users:

                    sock.send(pickle.dumps(('banned','')))

*else*:

                    print(self.users\_time\_dict.keys())

*if* name *in* self.users\_time\_dict.keys():

                        sock.send(pickle.dumps(("User already in use","")))

*else*:

                        sock.send(pickle.dumps(('ok',check)))

                        self.current\_user\_socket\_dict[name] = sock

                        self.users\_time\_dict[name]=datetime.datetime.now()

*else*:

                sock.send(pickle.dumps(("password is inccorect","")))

        time.sleep(0.5)

        sock.send("stop".encode())

        udp\_port = pickle.loads(sock.recv(1054))

        print(*f*"finishing log in: {udp\_port}")

        self.name\_udp\_port\_dict[name] = udp\_port

        self.give\_messages\_to\_user(name,sock)

*def* *guest\_log\_in*(*self*,*sock*:socket):

        sock.send(pickle.dumps(self.guest\_number))

        name = "guest" + str(self.guest\_number)

        self.guest\_number+=1

        self.current\_user\_socket\_dict[name] = sock

        udp\_port = pickle.loads(sock.recv(1054))

        self.name\_udp\_port\_dict[name] = udp\_port

## **קבלת חדר ממשתמש ויצירתו**

*def* *get\_new\_room\_and\_add\_to\_db*(*self*,*sock*:socket):

        sock.send(pickle.dumps("ok"))

        new\_room = self.get\_large\_data(sock)

        new\_room.last\_sent\_time = new\_room.last\_sent\_time + datetime.timedelta(*days*=int(self.days\_to\_skip))

        new\_room.create\_time = datetime.datetime.now() + datetime.timedelta(*days*=int(self.days\_to\_skip))

        user = new\_room.creator

        user.joined\_room.append(new\_room)

        self.db\_conn.update\_user(user)

        sock.send(pickle.dumps(self.add\_chatroom\_to\_db(new\_room)))

## **שליחת חדר למשתמש**

*def* *return\_room\_by\_id*(*self*,*sock*:socket,*info*):

        id\_pattern = re.compile(*r*"id:<(\d+)>")

        name\_pattern = re.compile(*r*"name:<(.+?)>")

        id\_num = re.findall(*pattern*=id\_pattern,*string*=info)[0]

        name = re.findall(*pattern*=name\_pattern,*string* = info)[0]

        print(id\_num)

        print(name)

        chat\_room = self.db\_conn.get\_chat(id\_num)

*if* name*!=*"no":

            self.current\_user\_chat\_room\_dict[name] = id\_num

*try*:

            chat\_room.current\_members = self.get\_how\_many\_members\_are\_online\_to\_a\_room(id\_num)

*except* AttributeError:

            sock.send(pickle.dumps("no chatroom"))

            time.sleep(0.5)

            sock.send("stop".encode())

            print('no chatroom')

*return*

        sock.send(pickle.dumps(chat\_room))

        print('sent chatroom')

        print(chat\_room)

        time.sleep(0.5)

        sock.send("stop".encode())

        msg = pickle.loads(sock.recv(1054))

*if* msg *==* "no need":

            print("yeh")

*return*

        sock.send(pickle.dumps("need what?"))

        need = pickle.loads(sock.recv(1054))

        print(need)

*for* img\_name *in* need:

            self.send\_picture(sock,img\_name)

## **קבלת הודעה חדשה מהמשתמש**

האלגוריתם הזה מקבל נתונים של הודעה חדשה כמו טקסט, כותרת, תמונה וכו'.

מעביר את התמונה, ומודיע לכל שאר המשתמשים אשר מחוברים לחדר כי יש הודעה חדשה.

בנוסף הוא גם עושה שוב ניתוח טקסטואלי לחדר לאחר שנוספה ההודעה החדשה ומעדכן את מילות הקשר של החדר.

*def* *get\_new\_message\_and\_add\_to\_db*(*self*,*sock*:socket):

        print('sup')

        sock.send(pickle.dumps("send msg"))

        print('supsup?')

        data = sock.recv(1054)

*while* True:

            packet = sock.recv(1054)

*if* packet *==* "stop".encode():

*break*

            data+=packet

        new\_msg = pickle.loads(data)

        sock.send(pickle.dumps("ok"))

*if* new\_msg.img\_name *!=* "":

*if* exists(*f*"server\_pics\\{new\_msg.img\_name}"):

                extension\_name = new\_msg.img\_name[new\_msg.img\_name.find(".")+1:]

                new\_msg.img\_name = new\_msg.img\_name[:new\_msg.img\_name.find(".")]+*f*'1.{extension\_name}'

            self.get\_picture\_and\_save(sock,new\_msg.img\_name)

        print(new\_msg.img\_name)

        chat\_room = new\_msg.sent\_in

        chat\_room.msgs.append(new\_msg)

        chat\_room.last\_sent\_time = datetime.datetime.now() + datetime.timedelta(*days* = int(self.days\_to\_skip))

        self.db\_conn.insert\_chat(chat\_room.room\_id,chat\_room)

        chat\_room.common\_words = self.get\_word\_score\_dict\_for\_chat\_room(chat\_room.room\_id)

        self.db\_conn.insert\_chat(chat\_room.room\_id,chat\_room)

        self.notify\_all\_members\_of\_chatroom\_for\_new\_msg(chat\_room)

*def* *notify\_all\_members\_of\_chatroom\_for\_new\_msg*(*self*,*chat\_room*):

        print('in refresh function')

*for* name *in* self.current\_user\_chat\_room\_dict:

*if* self.current\_user\_chat\_room\_dict[name] *==* str(chat\_room.room\_id):

                sock = self.current\_user\_socket\_dict[name]

*try*:

                    ip\_addr,port = sock.getpeername()

*except* OSError:

                    print(*f*"Failed, sock: {sock}")

                print(name)

                print('sent refresh msg to',ip\_addr)

                self.udp\_sock.sendto(pickle.dumps("need refresh"),(ip\_addr,self.name\_udp\_port\_dict[name]))

## **ניתוח טקסטואלי באמצעות Rake**

*def* *get\_word\_score\_dict\_for\_chat\_room*(*self*,*id\_num*) ->dict:

        id\_text\_dict = self.get\_all\_words\_from\_chats()

        text = id\_text\_dict[id\_num]

        chat\_room = self.db\_conn.get\_chat(id\_num)

        text = self.remove\_unwanted\_words\_from\_chat\_room(text,chat\_room)

        self.rake\_obj.extract\_keywords\_from\_text(text)

        word\_score\_dict = dict(self.rake\_obj.get\_word\_degrees())

        print("rake calc word score dict",word\_score\_dict)

*return* word\_score\_dict

*def* *clac\_the\_score\_of\_a\_word\_in\_chat*(*self*,*id\_num*:int,*word*:str)->int:

        id\_text\_dict = self.get\_all\_words\_from\_chats()

        text = id\_text\_dict[id\_num]

        chat\_room = self.db\_conn.get\_chat(id\_num)

        text = self.remove\_unwanted\_words\_from\_chat\_room(text,chat\_room)

        self.rake\_obj.extract\_keywords\_from\_text(text)

        word\_score\_dict = dict(self.rake\_obj.get\_word\_degrees())

*try*:

*return* word\_score\_dict[word]

*except* KeyError:

*return* 0

# **התקנה והרצה**

כל הפרויקט נמצא בgithub בחשבון שלי כל מה שצריך לעשות זה להוריד redis ולהריץ בצד השרת, להוריד את הקבצים בZIP.

לאחר שמורידים הקבצים צריך להדליק את redis, לעדכן את כתובת הIP בתוך הserver.py והuser\_controller.py בתוך הפונקציית init שלהם. לאחר פעולות אלה אפשר להריץ את השרת ולהתחיל לפתוח לקוחות.

הערה –

Rake לפעמים צורך להוריד גם תוסף של nltk כיוון שזהו תוספת שלו, אם תתקבל שגיאה שקשורה לזה צריך לפתוח קובץ חדש ולכתוב את השורות הבאות –

Import nltk

nltk.download()

דבר זה ייפתח חלון בו צריך ללחוץ על download all ואחרי זה התכנית אמורה לרוץ