

נגן Streaming



מגיש: יובל הושע

ת"ז: 325320695

מנחים: תומר טלגם, ירון יצחקי

תיכון: אהל שם

חלופה:

תאריך:

תוכן עניינים

[מבוא 4](#_Toc75449472)

[למה בחרתי בפרויקט הזה ? 4](#_Toc75449473)

[איך זה עובד ? 4](#_Toc75449474)

[ארכיטקטורה 6](#_Toc75449475)

[תרשים התקשורת בין הלקוח לשרת 6](#_Toc75449476)

[הסבר 6](#_Toc75449477)

[התקשורת בין השרת ללקוח 7](#_Toc75449478)

[הסטרמת תמונות 9](#_Toc75449479)

[מסד הנתונים 11](#_Toc75449480)

[מדריך למשתמש 12](#_Toc75449481)

[השרת 12](#_Toc75449482)

[הלקוח 12](#_Toc75449483)

[מדריך למפתח 16](#_Toc75449484)

[מודולים עיקריים 16](#_Toc75449485)

[תיקיית server 16](#_Toc75449486)

[server.py 16](#_Toc75449487)

[ThreadedClient.py 16](#_Toc75449488)

[ServerConfig.py 17](#_Toc75449489)

[socket\_functions.py 17](#_Toc75449490)

[database.py 17](#_Toc75449491)

[.env 17](#_Toc75449492)

[תיקיית client 18](#_Toc75449493)

[main.py 18](#_Toc75449494)

[client.py 18](#_Toc75449495)

[videoplayer.py 18](#_Toc75449496)

[gui.py 18](#_Toc75449497)

[dialogs.py 18](#_Toc75449498)

[image\_functions.py 18](#_Toc75449499)

[ClientConfig.py 19](#_Toc75449500)

[socket\_fucntions.py 19](#_Toc75449501)

[.env 19](#_Toc75449502)

[title.jpg 19](#_Toc75449503)

[כללי 19](#_Toc75449504)

[requirements.txt 19](#_Toc75449505)

[רפלקציה 20](#_Toc75449506)

# מבוא

הזרמת מדיה ("סטרימינג") היא טכנולוגיית אינטרנט שבאמצעותה ניתן להעביר מדיה דיגיטלית למשתמש קצה באופן משתמש, רציף וממושך.

טכנולוגיית הסטרימינג משומשת בהמון פלטפורמות פופולריות ברחבי האינטרנט כמו: יוטיוב, twitch, נטפליקס, דיסקורד ועוד הרבה פלטפורמות פופולריות אחרות.

בשימוש בסטרימינג התוכן מועבר באופן שוטף למשתמש תוך כדי שהוא צורך אותו. התוכן מועבר בחלקים ובכל רגע המשתמש צופה בחלק מהתוכן. זאת בניגוד להעברת תוכן בדרך הרגילה, העברת כל התוכן ואז צפייה בכל התוכן. עבור קבצים גדולים, כמו קבצי וידאו, הקובץ מאוד גדול לכן ייקח זמן רב עד שהקובץ יעבור למשתמש ורק אז המשתמש יוכל לצפות בו.

מטרת הפרויקט היא העברת סרטון בצורת סטרימינג למספר רב של משתמשים בו זמנית. על הסרטון לעבור בצורה טובה, מהירה ואיכותית.

## למה בחרתי בפרויקט הזה ?

בזמני הפנוי אני אוהב לראות סדרות, ואיפה רואים היום סדרות אם לא בנטפליקס? כשצופים בנטפליקס אתה לא מוריד את הסדרה בה אתה צופה אלא אתה "מסטרים" אותה. תמיד התענייתי איך זה עובד מאחורי הקלעים, ואיך מתרחש "הקסם הזה" שאתה צופה במשהו בלי לשמור אותו במחשב. במיוחד שאם אתה שומר אותו במחשב אתה צריך לחכות עד שכולו יורד למחשב ורק אז אתה יכול לצפות בו. חשוב לציין שאתה גם מחכה המון זמן. בנוסף, בשנה האחרונה, "שנת הקורונה " כולנו השתמשנו ב"zoom", שגם הוא למעשה "מסטרים" תוכן.

## איך זה עובד ?

כמו שכתוב לעיל, צפייה בסטרימינג לעומת צפייה ב"שיטות" הישנות היא צפייה בה קובץ הוידאו אינו יורד בשלמותו למחשב. למשתמש נשלח משרת חלק מקובץ הוידאו תוך כדי שהוא צופה. חקרתי כיצד זה עובד על מנת ליישם את זה בעצמי. השרת שולח תוכן שהמשתמש מקבל. הוא נשמר ויורץ כאשר יגיעו לנקודה הרלוונטית בסרטון – זהו למעשה "הבאפר". בשיטה זאת, המשתמש לא מחכה דקות ארוכות עד שהתוכן אשר רוצה לראות יורד והוא צופה בו במיידית. בפועל, לדוגמא בסרטמינג של קבצי וידאו, הנושא בו הפרויקט עוסק, אם נשלח כל רגע פריים של תמונה זה לא יעבוד. יש יותר מדי תמונות (הפריימים של הסרטון) אשר נשלחים כל שנייה. אם נשלח את כל התמונות כל רגע הסרטון יתקע. על מנת להקל על תעבורת הרשת דוחסים את הקובץ בשיטות שונות.

הבאפר שנוצר כאשר מחכים שסרטון יטען

# ארכיטקטורה

הפרויקט ממומש בשפת פייתון 3.

בפרויקט ישנו שרת אשר יכול לטפל במספר רב של לקוחות בו זמנית, זאת בזכות השימוש ב-multi-threading. התקשורת בין הלקוח לשרת היא באמצעות socket.

## Diagram Description automatically generatedתרשים התקשורת בין הלקוח לשרת

## הסבר

ישנו Thread ראשי של התוכנה אשר מקבל לקוחות חדשים. השרת יוצר עבור כל לקוח חדש אשר מתחבר לשרת, Thread חדש אשר יטפל בו. הThread ירוץ כל עוד הThread הראשי רץ.

**def run(**self**):** # start listening  
 self.\_socket.bind**(**self.\_addr**)** self.\_socket.listen**(**self.\_max\_listeners**)** logger.info**(f"LISTENING AT {**self.\_addr**}")** # getting the clients  
 **while True:** conn, addr **=** self.\_socket.accept**()** logger.info**(f"Got new client {**addr**}.")** ClientThread**(**conn, addr**)**.start**()**

הThread הראשי ירוץ כל עוד לא יעצרו את התוכנה.

פעולת ההרצה של השרת

בצד הלקוח יש לקוח אשר מתחבר לשרת ומקבל ממנו את כל התכנים שיש לו להציע. לאחר מכן, הלקוח בוחר את התוכן הרצוי. הThread שנוצר בשביל הלקוח בצד שרת שולח ללקוח את הפריימים שהלקוח מבקש ממנו. למעשה יש בצד הלקוח Thread שכל מה שהוא עושה זה לבקש את הפריימים

מהשרת כי יש Thread נוסף שאחראי על הממשק הגרפי בו מציגים את הסרטון. הThread רץ במקביל להצגת הסרטון לפי עקרון הסטרמינג בו מבקשים חלק מהתוכן, במקרה הזה פריימים של הסרטון, תוך כדי הצגת התמונה בממשק הגרפי. הלקוח מבקש מהשרת פריימים, השרת שולח לו את הפריים והלקוח מקבל את הפריים. הלקוח מוסיף את הפריים שקיבל לתוך אובייקט בשם VideoPlayer. האובייקט מכיל מבנה נתונים – תור אשר בתוכו נמצאים כל הפריימים שלא השתמשו בהם עוד.

הThread שמבקש פריימים מהשרת

**class AskingForFrameThread(**threading.Thread**):  
  
 def \_\_init\_\_(**self, client**:** Client, video\_player**:** VideoPlayer, vid\_name**:** str**):  
 ...  
  
 def run(**self**)** -> **None:  
 """  
 This function executes when you starting the thread.  
 The functions ask frames from the server  
 """  
 while** self.\_\_alive**:  
 if** self.\_\_paused**:  
 continue  
  
 if** self.video\_player.can\_add\_frame**() and** self.client.can\_request\_frame**():** self.client.ask\_for\_frame**(**self.vid\_name**)** # Wait a little  
 time.sleep**(**0.001**)**

**class VideoPlayer:** MAX\_FRAMES **=** 50  
 \_\_NOT\_SET **= -**1  
  
 **def \_\_init\_\_(**self**):** self.\_queue **=** deque**() # תור** self.\_frames\_got\_counter **=** 0  
 self.\_frames\_played\_counter **=** 0  
  
 self.\_time\_between\_frames\_ms **=** self.\_\_NOT\_SET  
 self.\_frames\_amount **=** self.\_\_NOT\_SET  
  
 self.\_no\_frames\_from\_server **= False** self.\_\_scale\_percent **=** 100

במחלקה הזאת נשמרים כל הפריימים

**def \_listen\_to\_server(**self**):  
 while True:  
 try:** got\_data, data **=** read\_data\_from\_socket**(**self.\_sock, logger**)  
 except** pickle.UnpicklingError **as** e**:** logger.error**(**e**)  
 continue  
  
 if not** got\_data**:** time.sleep**(**0.001**)  
 continue** logger.debug**(f"Got data from server")** self.\_handle\_data**(**data**)**

בוא זמנית יש גם Thread אשר מקבל דברים מהשרת, כלומר יש Thread ששולח בקשות לשרת וThread שמקבל תגובות מהשרת. אני מפריד בין התגובה לבקשה על מנת שיהיה אפשר לבקש דברים חדשים גם לפני שמקבלים תגובה מהשרת לדברים קודמים.

## התקשורת בין השרת ללקוח

**class Server:  
  
 def \_\_init\_\_(**self, ip**:** str, port**:** int, max\_listeners**:** int**):** self.\_max\_listeners **=** max\_listeners  
 self.\_addr **= (**ip, port**)** self.\_socket **=** socket.socket**(**socket.AF\_INET, socket.SOCK\_STREAM**)**

התקשורת כפי שצוין קודם מתרחשת באמצעות "סוקטים" (socket). התקשורת בין "הסוקטים" היא באמצעות פרוטוקול TCP זאת על מנת שביתים לא יעלמו או ישתנו בתעבורת רשת, זאת **למרות** העובדה שבפרוטוקול החלופי, UDP אנו מעדיפים מהירות על איכות. כאן אנו נעדיף איכות כי מאוד חשוב שהביתים יעברו במדויק (הסבר בהמשך) כמו שצריך ויתרון המהירות זניח על פני יתרון האיכות.

שימוש ב socket.SOCK\_STREAM ולא ב socket.SOCK\_DGRAM ובכך להשתמש בTCP ולא בUDP

פרוטוקל שליחת המידע עובד כך:

header - ה-10 תווים הראשונים ששולחים כבתים הם גודל המידע ששולחים כבתים לאחר שעברו סירלזציה. את 10 התווים הראשונים אנחנו הופכים לבתים.

def make\_header(data: bytes):  
 *"""* ***:param*** *data: bytes of the data* ***:return****: the header of the bytes, the length of the data, for 2  
 """* return str(len(data)).zfill(HEADER\_LENGTH).encode()

שאר הבתים זה המידע כאשר הספרייה pickle עושה סירלזציה (אם זה תמונה, טקסט, מערך, למעשה כל פרויקט שקיים בפייתון) לבתים.

final\_data = pickle.dumps(data)  
final\_data = make\_header(final\_data) + final\_data  
sock.sendall(final\_data)

לאחר מכן אנחנו שולחים את כל הבתים.

לדוגמא אנחנו צריכים לשלוח את המחרוזת "hello". נעשה סירלזציה על המחרוזת ונקבל: b'\x80\x03X\x05\x00\x00\x00helloq\x00'. על הבתים נפעיל את הפונקציה make\_header ונקבל b'0000000015'. נחבר את הheader עם הבתים ונקבל b'0000000015\x80\x03X\x05\x00\x00\x00helloq\x00' ואת זה נשלח לצד השני (הלקוח או השרת). אמנם, שיטה זאת טיפה לא יעילה מאחר וניתן לעשות "hello".encode() במקום הסירליזציה של pickle אבל כאשר עובדים עם מבני נתונים אחרים זה מקל עלינו. בגלל הסירליזציה חשוב מאוד שהביטים יעברו במדויק כי אם ביט אחד ישתנה כל התהליך לא יעבוד ונקבל שגיאה.

קבלת המידע: לוקחים את הheader מהסוקט, ניתן לקחת אותו כי הוא בגודל קובע ואז הופכים את זה למספר על מנת לדעת מה גודל התוכן שאנו צפויים לקבל. לאחר מכן, לוקחים את התוכן מהסוקט, אנחנו יכולים לעשות זה כי אנחנו יודעים מה גודל התוכן. את הבתים של התוכן הופכים חזרה לאובייקטים של פייתון באמצעות הספרייה pickle, pickle.loads.

data **=** bytearray**()  
while** len**(**data**) <** size**:** packet **=** sock.recv**(**size **-** len**(**data**))** data.extend**(**packet**)**data **=** pickle.loads**(**data**)**

header = sock.recv(HEADER\_LENGTH)  
if not header:  
 return False, None  
size = int(header.decode())

## הסטרמת תמונות

חלק משמעותי בסטרמינג הוא למעשה הסטרמת התמונות. תמונה היא למעשה מערך תלת מימדי של אורך, רוחב ושלושת הצבעים: אדום, ירוק וכחול (RGB). תמונה סטנדרטית היא לדוגמה 900x1200. לרוב קצב הרענון של סרטונים הוא 30 FPS, כלומר 30 תמונות **כל שנייה**. כאשר עושים סירליזציה על מערך כזה, מקבלים גודל עצום של ביטים. אנו רוצים להקטין את כמות הביטים. בשביל זה נשתמש בפרוטוקול התמונות "jpeg". פרוטוקול זה מכווץ את התמונה ומקטין את גודלה כך שנוכל לשלוח אותה בהרבה פחות בתים. איכות התמונה טיפה יורדת אך הפער כמעט ולא מורגש.

הסריקפט שבעזרותו בחרתי בפרוטוקול הjpeg ולא אחד מפרוטוקולי התמונות האחרים:

**import** pickle  
**import** io  
**import** numpy **as** np  
**from** PIL **import** Image  
  
  
**def picture\_array():  
 with** Image.open**("title.jpg") as** img**:** img\_arr **=** np.asarray**(**img**)  
 return** img\_arr  
  
  
**def size\_at\_format(**img\_arr, img\_format**:** str**):** img\_bytes **=** io.BytesIO**()** img\_pil **=** Image.fromarray**(**img\_arr**)** img\_pil.save**(**img\_bytes, format**=**img\_format**)** bytes\_to\_send **=** img\_bytes.getvalue**()  
 return** len**(**pickle.dumps**(**bytes\_to\_send**))  
  
  
def main():** img\_arr **=** picture\_array**()** print**(f"Size as array: {**len**(**pickle.dumps**(**img\_arr**))} bytes.")** print**(f"Size as jpeg: {**size\_at\_format**(**img\_arr, **'jpeg')} bytes.")** print**(f"Size as png: {**size\_at\_format**(**img\_arr, **'png')} bytes.")** print**(f"Size as bmp: {**size\_at\_format**(**img\_arr, **'bmp')} bytes.")** print**(f"Size as gif: {**size\_at\_format**(**img\_arr, **'gif')} bytes.")** print**(f"Size as tiff: {**size\_at\_format**(**img\_arr, **'tiff')} bytes.")** print**(f"Size as eps: {**size\_at\_format**(**img\_arr, **'eps')} bytes.")  
  
  
if** \_\_name\_\_ **== "\_\_main\_\_":** main**()**

הסקריפט החזיר את הפלט הבא:

Size as array: 186404 bytes.

Size as jpeg: 12427 bytes.

Size as png: 29608 bytes.

Size as bmp: 186303 bytes.

Size as gif: 20263 bytes.

Size as tiff: 186389 bytes.

Size as eps: 377620 bytes.

מכאן אנו רואים שגודל התמונה כמערך היא 184,000 בתים, כלומר 180 KB. כJPEG היא 12,000 בתים, 12 KB. עבור שאר הפורמטים האפשריים, גודל התמונה יותר גדול מאשר בJPEG ולכן בחרתי בפרוטוקול זה.

# מסד הנתונים

לתוכנה יש מסד נתונים. מסד הנתונים ממומש באמצעות SQLAlcehmy.

על מנת להשתמש בתוכנה יש ליצור חשבון ולהתחבר אליו. על מנת לשמור את שמות המשתמש והסיסמאות ניצור מסד נתונים.

מסד הנתונים:

|  |  |
| --- | --- |
| str(255)- primarykey | username |
| str(128) | password |

על מנת שמסד הנתונים לא ישמור את הססמה המקומית (לדוגמא "1234") אנו נשמור את כל הססמאות כhash. פונקציית hash היא פונקציה שממירה קלט לפלט באורך קבוע (כאן האורך הוא 128 תווים). פונקצייה זאת היא פונקצייה חד כיוונית, כלומר ניתן להפוך את הקלט לפלט אך לא את הפלט לקלט. כך, גם אם מישהו ישיג את מסד הנתונים לא תהיה לו גישה אל הססמאות כי יהיה לו רק אתhash הסיסמאות. השרת מקבל את הסיסמה לפני שהיא הופכת לhash, ואז הופך את הסיסמה לhash ומשווה אותה למה ששמור

הפונקציה הממירה סיסמה לhash:

**def hash\_password(**password**:** str**)** -> str**:  
 return** hashlib.sha512**(**password.encode**())**.hexdigest**()**

עבור הססמה 1234 נקבל את הhash :

d404559f602eab6fd602ac7680dacbfaadd13630335e951f097af3900e9de176b6db28512f2e000b9d04fba5133e8b1c6e8df59db3a8ab9d60be4b97cc9e81db

הפונקציה שבודקת אם שם המשתמש מתאים לסיסמה:

**@**classmethod  
**def valid\_user(**cls, username**:** str, password**:** str**)** -> bool**:** user **=** cls.find**(**username**)  
 if** user **is None:  
 return False  
  
 return** hash\_password**(**password**) ==** user.password

דוגמה לטבלה אפשרית:

Graphical user interface, text, application

Description automatically generated

# מדריך למשתמש

Graphical user interface, application, table, Excel

Description automatically generatedקישור לקוד: <https://github.com/Yuval-Hoshea/StreamingServer>

לפרויקט יש שני תיקיות, תיקיית server ותיקיית client. על מנת להשתמש בפרויקט צריך להפעיל את השרת הנמצא בתיקיית server. תיקיית ה-venv היא הסביבה הוירטואלית של הפרוייקט שמכילה את כל המודולים שנמצאים בקובץ requerments.txt.

Graphical user interface, application, table, Excel

Description automatically generated

## השרת

תיקיית השרת היא server והתיקייה מכילה את הקבצים והתיקיות הללו:

לפני ההרצה של השרת יש להחליט באיזה IP וPORT נשתמש. נגדיר אותם בקובץ .env בתוך הקובץ יש:

שורת הIP זה הIP בוא נשתמש, 0.0.0.0 אם נרצה שכל מי שברשת יוכל להשתמש בשרת. 127.0.0.1 אם נרצה שרק המחשב של השרת יוכל להשתמש בשרת. PORT יהיה מספר בין 0 ל 65536.

IP=0.0.0.0

PORT=19099

לאחר שהגדרנו את כתובת הIP והפורט בקובץ ".env", יש להריץ את הקובץ server.py. נריץ אותו באמצעות סביבה וירטואלית שתכיל את הספריות הנמצאות בקובץ requirement.txt שבתיקייה הראשית של הפרויקט. נריץ ונקבל: Graphical user interface, application, Word

Description automatically generated

זה אומר שהשרת פועל ומקשיב בכתובת שהפעלנו.

Graphical user interface, application, Word

Description automatically generatedכל הסרטונים אשר השרת מציג נמצאים בתקייה videos. התיקייה מכילה תיקיות אחרות אשר כל תיקייה היא סרטון אחר: בתוך כל תיקייה יש קובץ וידאו של הסרטון בשם video.<file\_type>, תמונה של הסרטון (thumbnail) img.jpg, וקובץ בשם video\_type.txt שמכיל את סוג הפורמט של הווידאו. לדוגמא mp4. אם נרצה להוסיף סרטון חדש פשוט ניצור תיקייה חדשה עם שם הסרטון, ואת הקבצים הדרושים לעיל.

Graphical user interface, application, table, Excel

Description automatically generatedGraphical user interface, application, table, Excel

Description automatically generatedבמידה וזאת הייתה הפעם הראשונה בה הורץ השרת, נוצר הקובץ db.sqlite3. אם השרת כבר הורץ לפחות פעם אחת הקובץ לא ישמר מחדש. קובץ זה הוא קובץ מסד הנתונים בו נשמרים שמות המשתמש והסיסמאות.

## הלקוח

תיקיית הלקוח היא client והיא מכילה את הקבצים הללו:

לפני הפעלת הלקוח נלך לקובץ .env שנראה כך:

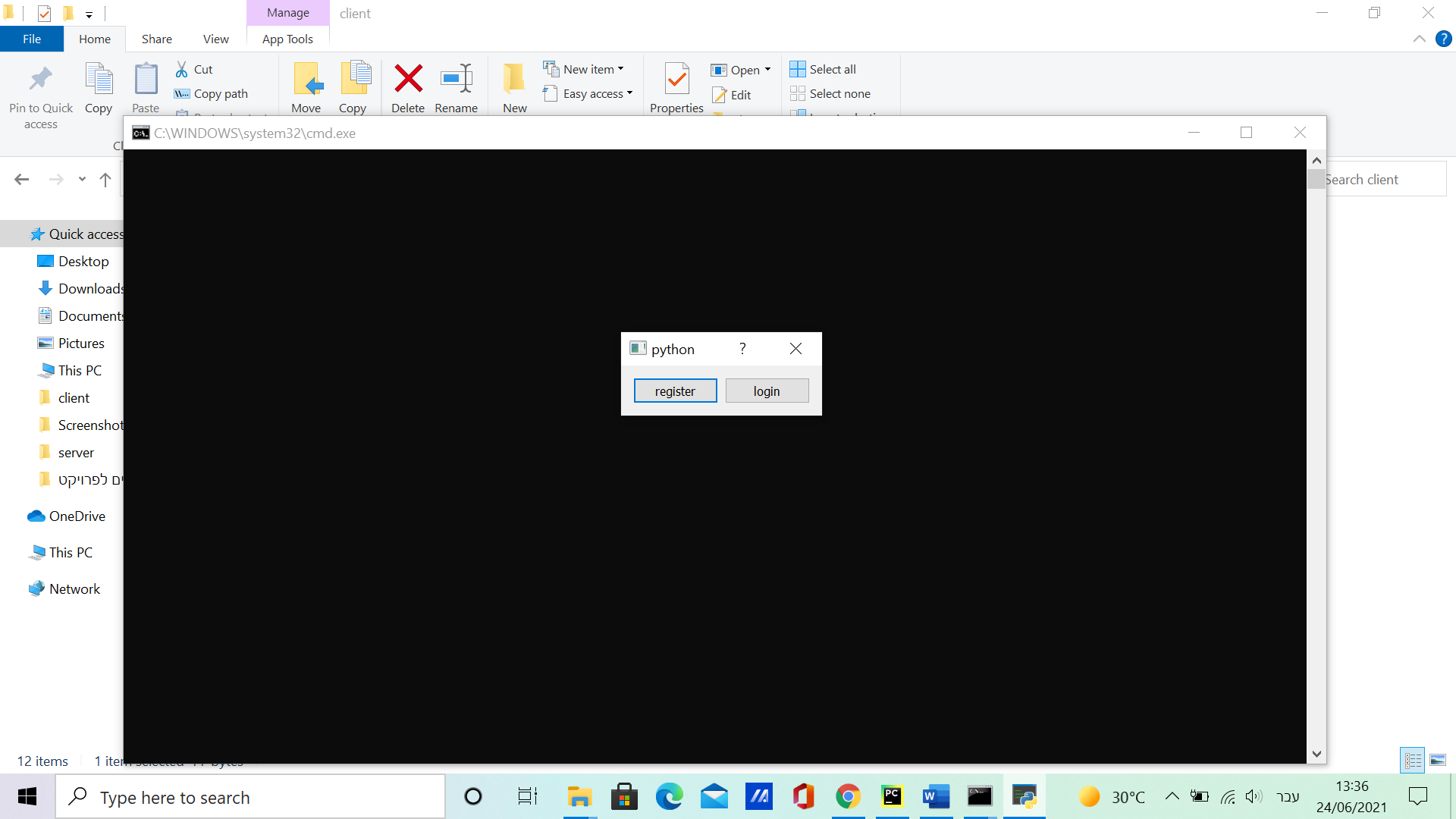
SERVER\_IP=127.0.0.1

SERVER\_PORT=19099

נכתוב בתוכו את כתובת הIP של השרת ואת הPORT שלו. לאחר מכן נריץ את הקובץ main.py באמצעות סביבה וירטואלית שתכיל את הספריות הנמצאות בקובץ requirements.txt שבתיקייה הראשית של הפרויקט.

**לפני ההרצה יש להפעיל את השרת.**

נריץ ונראה את הדבר הבא:



בוא זמנית בצד השרת:

A picture containing text, screenshot, electronics, computer

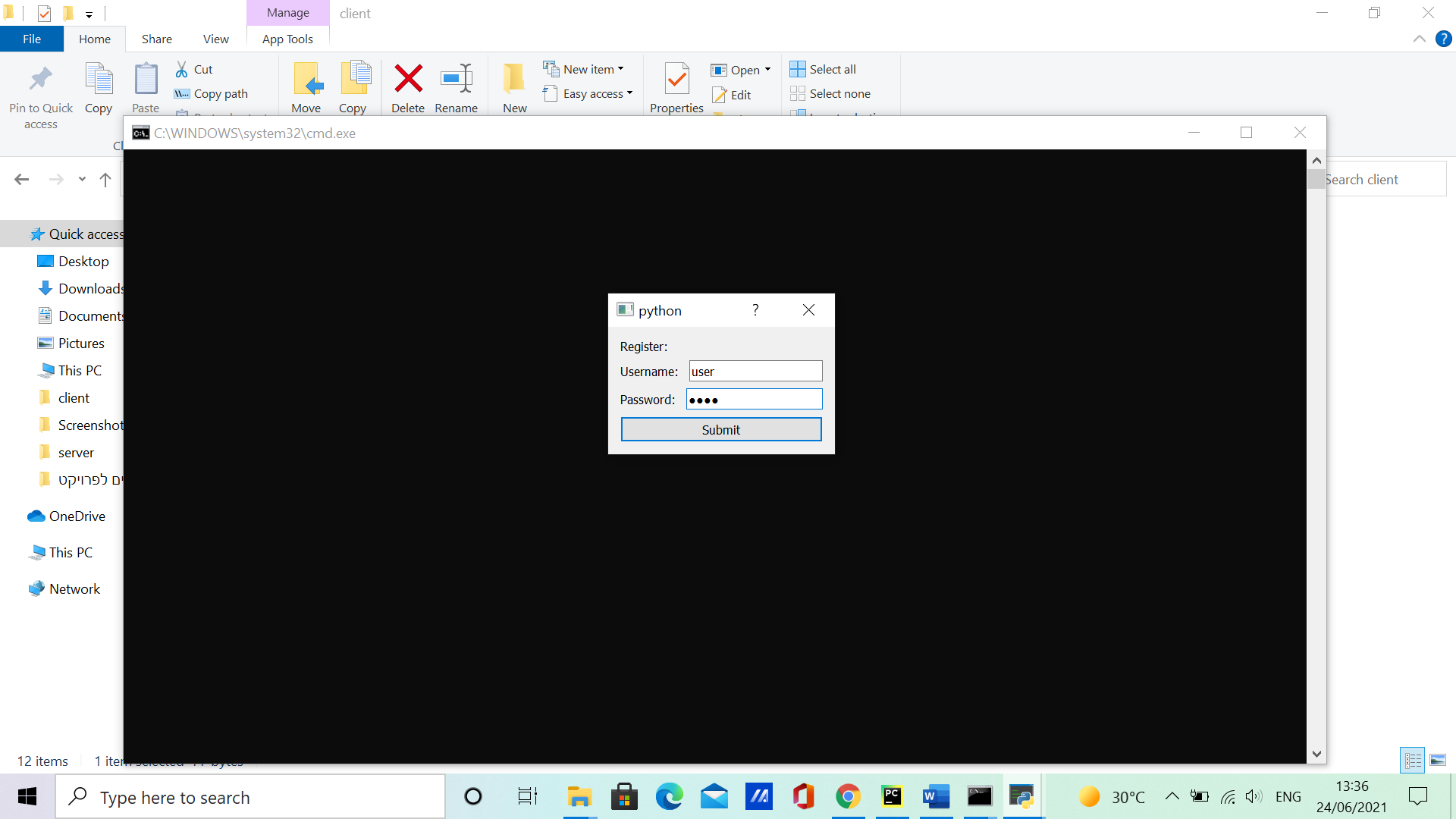
Description automatically generated

התווספו שני שורות ממקודם אשר מראות על כך שנוצר קשר עם לקוח והפעילו לו Thread שיטפל בו.

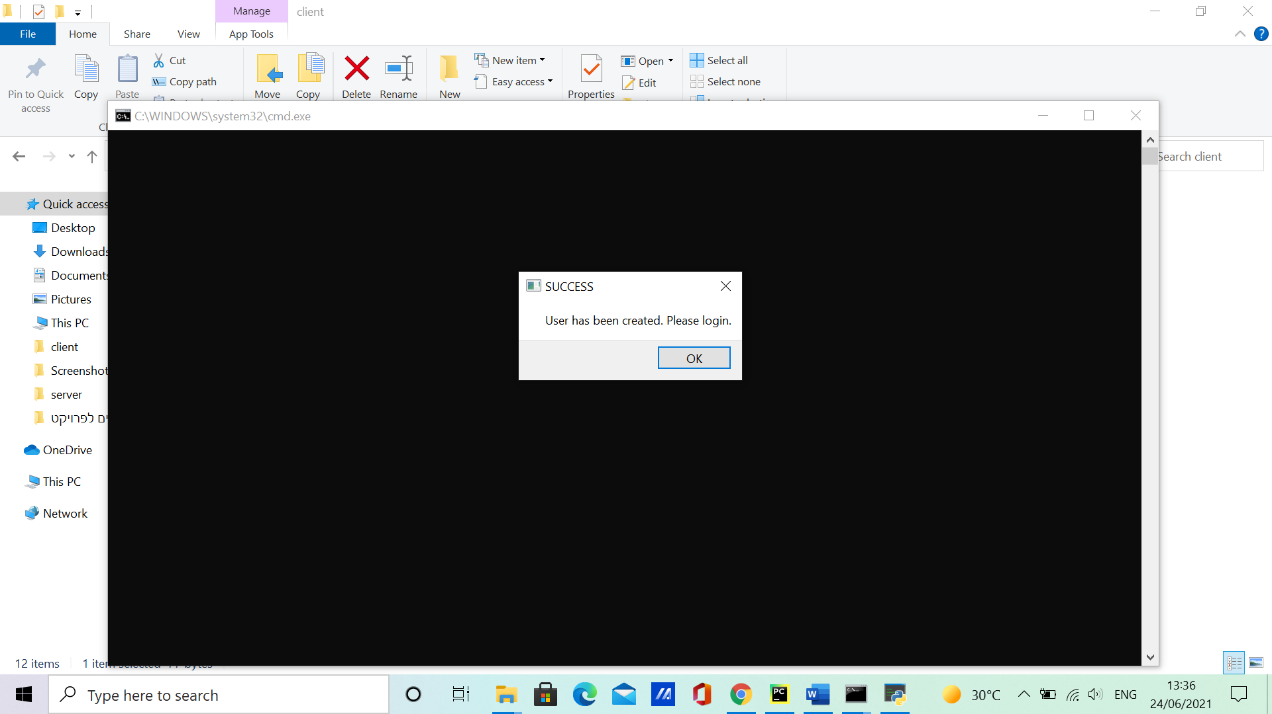
בחזרה ללקוח:

עלינו לבחור אם עלינו להירשם או להתחבר (אם אין חשבון קיים חייב להירשם קודם).

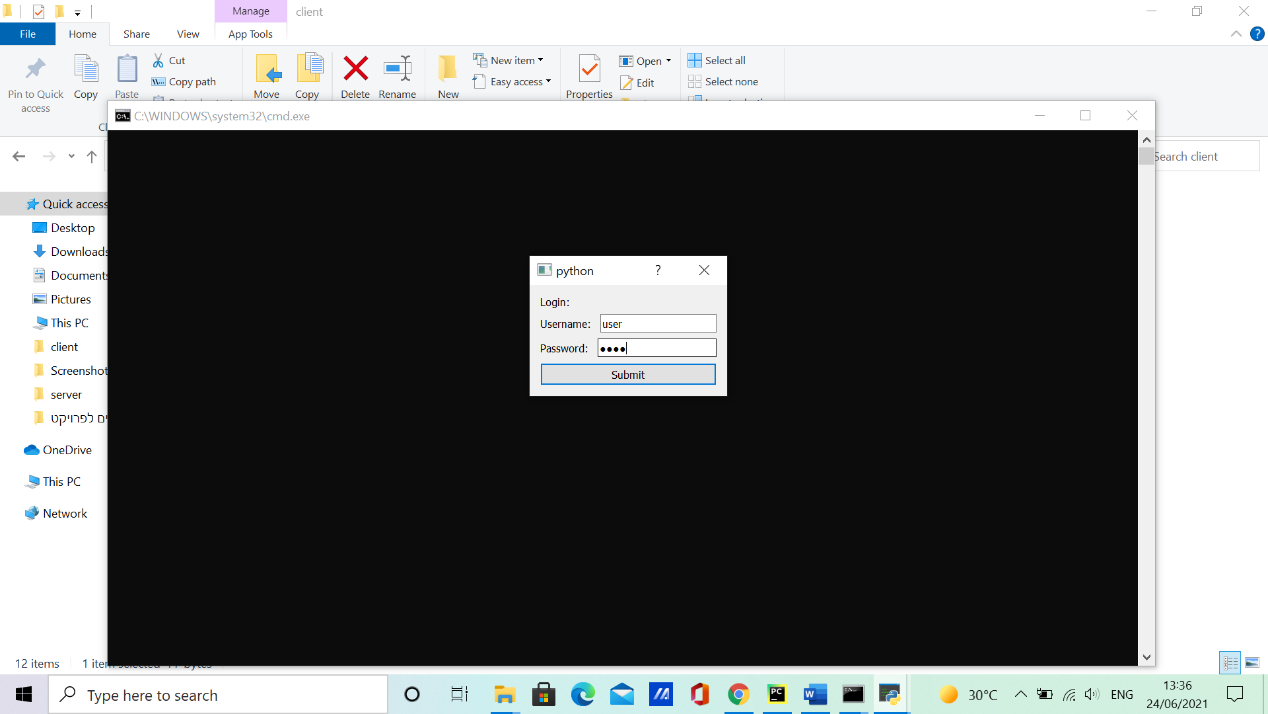
מסך ההרשמה:



כותבים את שם המשתמש והססמה.

 קיבלנו הודעה שהחשבון נוצר.

ואז חוזרים למסך הקודם. לוחצים על התחברות (login). ומגיעים למסך ההתחברות:

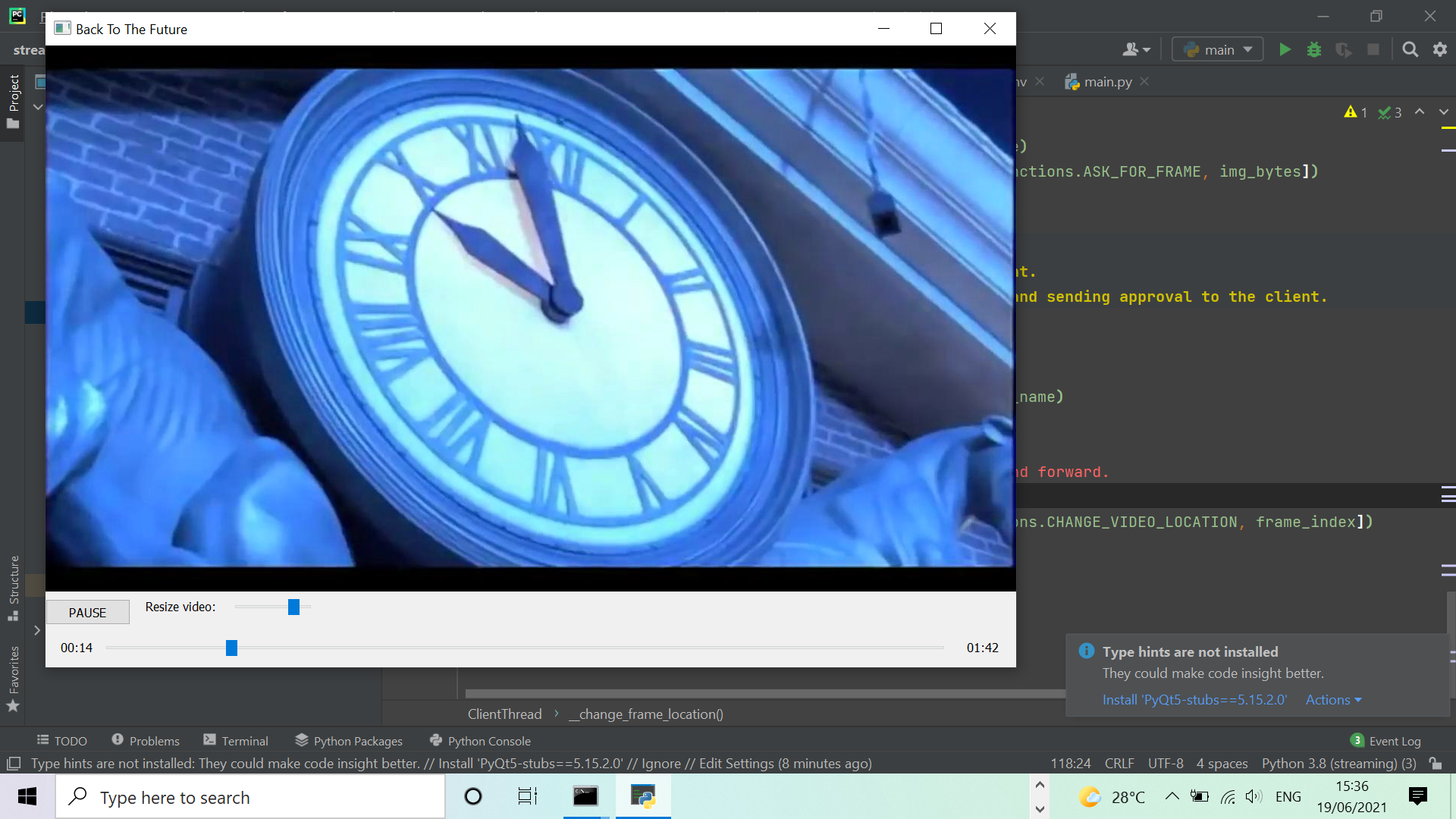


ואז מתחברים. לאחר ההתחברות יפתח מסך הסרטים הקיימים:

A picture containing text, screenshot, electronics, display

Description automatically generated

באמצעות הממשק הגרפי ניתן לבחור את הסרטון הרצוי. ניתן לסגור את התוכנה ואז הקשר עם הסרבר מתנתק ואפשר לבחור את הסרטון המבוקש. במידה ונלחץ על הסרטון המבוקש ייפתח מסך חדש (ראה עמוד הבא). באמצעות מסך זה נצפה בסרט.



הפריים המוצג של הסרטון

הזמן בסרטון בו אנו צופים

למזער את החלון

לסגור את החלון

כפתור הPAUSE, ניתן לעצור ולהמשיך את הסרטון באמצעותו

על ידי הזזת הslider ניתן לשנות את גודל המסך

על ידי הזזת הslider ניתן לשנות את הזמן בו אנו צופים

אורך הסרטון

שם הסרטון

לאחר שסוגרים את החלון חוזרים למסך בחירת הסרטונים, ניתן לבחור סרטון חדש או לצאת לחלוטין מן התוכנה.

# מדריך למפתח

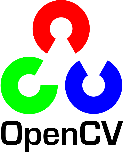
הפרוייקט נכתב בשפת פייתון גרסה 3.8. עקרונית הוא אמור לעבוד מגרסה 3.6 אך זה לא נבדק.

## מודולים עיקריים

socket- ספרייה זאת משמשת אותנו בתקשורת בין הלקוח לשרת. socket הוא חיבור של נקודות באינטרנט באמצעות כתובת ip וport. זאת באמצעות פרוטוקול TCP או UDP.

threading- שימוש בספרייה זאת על מנת לעשות דברים במקביל. אם בצד השרת בו אנו "מקשיבים" על מנת לקבל לקוחות חדשים ובוא זמנית יש threads אשר נותנים את השירות ללקוחות הקיימים. בצד הלקוח זה משמש אותנו בוא זמנית לבקש דברים מהשרת ולקבל בוא זמנית וגם להריץ את הממשק הגרפי במקביל לכך.

PIL- ספרייה לעיבוד תמונות התומכת בהרבה סוגי פורמטים של תמונות. מעבר בין מערך של תמונה לתמונה ולהפך. שמירת תמונות ועוד.

OpenCV - הספרייה OpenCV תומכת בקבלת תמונות מסרטון, מתמונה רגילה ועוד. היא משתמשת בnumpy בשביל ייצוג התמונות במערך. שני הספריות ממומשות לפחות בחלקן בשפת C מה שהופך את הפונקציות שלהם למאוד מהירות לעומת פונקציות בפייתון וגם גודל המערך יותר יעיל. בפרויקט הספריות משומשות בשביל לקחת פריימים מסרטון, לשנות את גודל הפריים, וכו'.

-PyQt5 היא אחת מן הספריות הכי נפוצות לבניית ממשקים גרפים בפייתון. היא תומכת בהרבה מערכות הפעלה כמו: windows, linux ו-MacOs. בפרויקט הממשק הגרפי משומש באמצעות הספרייה.

Sqlalchemy- ספרייה שבאמצעותה ניתן לשמור מסד נתונים. משומשת בשביל מסד הנתונים ששומר את שמות המשתמשים הרשומים והסיסמאות שלהם.

## תיקיית server

### server.py

הקובץ הראשי של השרת. מריץ את השרת. מכיל את המחלקה Server שמכילה שתי פונקציות:

* \_\_init\_\_, בנאי אשר מקבל ip, port, ולכמה קשרים מקשיבים.
* run, מפעיל לולאה אינסופית אשר בודקת אם יש תקשורת חדשה לסרבר. אם יש תקשורת חדשה פותחת Thread חדש, ClientThread (הסבר בThreadedClient.py).

### ThreadedClient.py

מכיל את המחלקה ClientThread אשר יורשת מן המחלקה Thread מספרייה threading. המחלקה הינה Thread אשר כפוף לThread הראשי של התוכנה הנמצא בserver.py.

הפונקציות שהמחלקה מכילה:

* \_\_init\_\_, מקבל socket ואת הכתובת של הלקוח.
* run, פונקציה זאת מריצה את הThread. לולאה אין סופית שבודקת אם התקבלה הודעה מן הclient, אם התקבלה מטפלת בה עם הפונקציה \_\_handle\_data.
* \_\_handle\_data, פונקציה אשר מטפלת בלקוח לפי הבקשה שהוא ביקש באמצעות הפונקציות: \_\_get\_videos\_list, \_\_get\_shows\_details, \_\_get\_frame, \_\_change\_frame\_location.

### ServerConfig.py

יוצר כל מיני קונפיגורציות לפרויקט. יצירת הlogger, רשימת כל הסרטונים שקיימים. טעינת קובץ ה.env ועוד. פונקציות ומשתנים גלובלים:

* all\_videos, פונקציה המחזירה את שמות כל הסרטונים.
* get\_video\_and\_thumbanil\_path, פונקציה המקבלת שם של סרטון ומחזירה את מיקום הסרטון ואת מיקום התמונה של הסרטון (thumnail).
* logger, בעזרתו דברים למסך לפי ההגדרה שנגדיר אותו, info,debug ועוד.
* IP, PORT, SERVER\_TIMEOUT, VIDEOS\_DIR\_PATH, ALL\_VIDEOS\_DIRECTORIES, משתנים גלובלים עם צרכים שונים בפרוייקט

### socket\_functions.py

אותו הקובץ נמצא גם בclient. מכיל פונקציות אשר שולחות מידע בין סוקטים. בנוסף פונקציות אשר הופכות תמונה לבתים וגם משתנים גלובלים אשר שימושיים גם ללקוח וגם לשרת (הפונקציות בינהם).

פונקציות:

* make\_header, יוצרת header לבקשות ותגובות הנשלחות בין סוקטים. בזכות הheader ניתן לדעת את אורך הבקשה/תגובה.
* - send\_data\_through\_socket, שולחת מידע דרך סוקט על ידי סירליזציה של הספרייה pickle.
* read\_data\_thorugh\_socket, קוראת את המידע מן הסוקט.
* encode\_img, הופכת תמונה לבתים לפי הפרוטוקול.
* decode\_img, הופכת את הבתים של התמונה בחזרה למערך של פיקסלים.

משתנים גלובלים:

* HEADER\_LENGTH, גודל הheader ששולחים לפני כל בקשה/תגובה בתקשורת.
* משתני פונצקיות- ASK\_FOR\_VIDEOS\_AVAILABLE, ASK\_FOR\_VIDEO\_DETAILS, ASK\_FOR\_FRAME, CHANGE\_VIDEO\_LOCATION, VIDEO\_THUMBNAIL. כל בקשה מתחילה עם אחד מהדברים הללו וכך ניתן לדעת מה ביקשו, פריים, פרטים על הסרטון, רשימת הסרטונים וכו'.
* IMAGE\_FORMAT, עם איזה קידוד של תמונה אנו משתמשים (jpeg).

### database.py

קובץ האחראי על יצירת מסד הנתונים. יוצר את מסד הנתונים ובתוכו יש את המחלקה User שלה יש פונקציות של בדיקה האם חשבון כבר קיים, האם הסיסמה מתאימה לשם המשתמש.

### .env

קובץ שמכיל את כתובת הIP והPORT שאנו משתמשים בהם.

## תיקיית client

### main.py

הקובץ שמריץ את כל התוכנה. משתמש בכל שאר הקבצים בשביל ליצור קשר עם השרת, לשמור את הפריימים בתוך תור, יצירת הממשק הגרפי ועוד.

### client.py

הקובץ מכיל את המחלקה של האובייקט Client. אובייקט זה אחראי על התקשורת בין השרת ללקוח. מיצירת התקשורת, קבלת הפריימים מהשרת, קבלת פרטים וכו'.

פונקציות מרכזיות במחלקה:

* \_\_init\_\_, הבנאי.
* פונקציות “ask”, פונקציות אשר מבקשות מידע מהשרת: ask\_for\_all\_videos\_available, ask\_for\_video\_details, ask\_for\_frame, ask\_for\_new\_location.
* \_\_handle\_data, הפונקציה שמטפלת במידע הנשלח מהשרת.

### videoplayer.py

הקובץ מכיל את המחלקה VideoPlayer אשר מכילה תור (queue) אשר מקבל פריים ומוסיף אותו לתור. המחלקה אחראית על קבלת הפריימים, שמירתם, כמה זמן יש בין פריים לפריים, האם ניתן להוסיף עוד פריים (Buffer), כמות הפריימים הכוללת שיש בסרטון ועוד.

### gui.py

הקובץ אחראי על הממשק הגרפי שבו צופים בסרטון (לא הממשק בו בוחרים סרטון).

בקובץ ישנם שלושה מחלקות:

* AskingForFrameThread- Thread אשר אחראי על בקשת הפריימים מן השרת עם שימוש באובייקט Client (client.py) ואובייקט VideoPlayer (videoplayer.py). יורש מן המחלקה thread של threading.
* FrameSlider – אובייקט שאחראי על ה"סליידר" של הזמן בסרטון. יורש מן המחלקה QSlider של PyQt5.QtWidgets.
* המחלקה Window שמשתמשת במחלקות הקודמות תוך כדי יצירת הממשק הגרפי אשר מראה את הפריימים של התמונות, מחליף בין הפריימים, מעדכן את הזמן בסרטון ועוד. יורש מן המחלקה QWidget של PyQt5.QtWidgets.

### dialogs.py

הקובץ אחראי על יצירת הדיאלוג של בחירת הסרטון. בשביל זה כתובה בקובץ המחלקה VideoDialog אשר יורשת מן QDialog של PyQt5.QtWidgets. המחלקה יוצרת את הדיאלוג בו מופיעים תמונות של כל הסרטונים האפשריים וכשהלקוח לוחץ על התמונה נפתח הממשק הגרפי שלהסרטון באמצעות המחלקות בקובץ gui.py. התמונה הלחיצה ממומשת באמצעות מחלקה נוספת אשר בקובץ ImageButton אשר יורשת מן החלקה QLabel של PyQt5.QtWidgets.

בקובץ זה נמצאים גם הדיאלוגים האחראיים להרשמה והתחברות, וגם לדיאלוג הבחירה בהתחלה בין הרשמה להתחברות.

### image\_functions.py

קובץ המכיל פונקציות שימושיות של תמונות.

הפונקציה resize\_image\_to\_specific\_height אשר משנה גודל תמונה לגודל מסוים לפי הגובה הרצוי.

הפונקציה convert\_numpy\_array\_to\_qimage שממירה מערך של תמונה (מערך numpy) לאובייקט QImage של PyQt5.QtGui שניתן להשתמש בו.

### ClientConfig.py

יצירת קונפיגורציות לפרוייקט מצד הלקוח. יצירת הlogger ולקיחת הIP והPORT של השרת מקובץ ה-.env

### socket\_fucntions.py

אותו הקובץ נמצא בשרת ולכן ראה את ההסבר על הקובץ בעמוד 17.

### .env

קובץ המכיל בתוכו את כתובת הIP והPORT של השרת. משתמשים בערכים הללו בקובץ ClientConfig.py.

### title.jpg

תמונה שמשתמשים בה בקובץ dialogs.py

## כללי

### requirements.txt

קובץ הנמצא בתיקייה הראשית של הפרויקט ולא בתיקיית השרת או הלקוח. מכיל את כל הספריות שצריך להתקין על מנת שהפרויקט יעבוד כשורה.

# רפלקציה

כשאמרו שצריך לבחור נושא לפרויקט הייתי אבוד. לא היה לי מושג איזה נושא לבחור ולקח לי הרבה זמן לבחור. לבסוף, כשבחרתי לעשות את הפרויקט על סטרימינג, חשבתי שאני אסיים אותו תוך כמה ימים מהרגע שאתחיל אותו. בפועל, המציאות הייתה שונה לגמרי. הפרויקט היה מאתגר ממה שחשבתי והייתי צריך להבין איך לעשות אותו מאפס בנוסף למחקר על איך סטרימינג עובד.

באמצעות הפרויקט צברתי ניסיון והרחבתי את הנסיון שלי בפייתון. למדתי להשתמש בספריות חדשות שלא ידעתי להשתמש בהן לפני: PyQt5, sql, numpy, OpenCV. למדתי איך תמונות בנויות ועל איך הן נשמרות במחשב.

מהלך הפיתוח היה מאתגר והיו הרבה בעיות ובאגים. לדוגמא היה לי באג מציק שגרם למחשב שלי לקרוס פעם אחת. הבאג נוצר מאחר ולפני שהלקוח שולח בקשה של הפריים הבא מהשרת, הוא בודק אם יש מקום בשביל הפריים העתידי. הגדרתי שאפשר לקבל 50 פריימים- לפחות חשבתי שהגדרתי... אם היה מקום, הלקוח היה מבקש מהשרת את הפריים. הבעיה היא שמהרגע שהוא מבקש את הפריים עד הרגע שהוא מקבל אותו עוברים כמה רגעים. באותם רגעים, עדיין יש מקום לעוד פריימים של הסרטון כי עוד לא קיבלנו את הפריים הראשון שביקשנו. בגלל זה, הלקוח היה ממשיך את הפריים הבא שוב ושוב. כך נוצר מצב שבמקום לקבל 50 פריימים, מקבלים 5,000. בהתחלה ,לא שמתי לב לבאג בכלל. מכיוון ששמתי סרטון קצר ככה שזה לא "הרג" את הזיכרון של המחשב כי לא היה בו הרבה פריימים. כששמתי סרטון יחסית יותר גדול פתאום המחשב קרס כי נגמר לו המקום בRAM ולקח לי הרבה זמן להבין שזה הבאג. זהו רק באג אחד מתוך כמה שהיו. בנוסף, הספרייה PyQt5 לעתים קורסת עם קוד שגיאה ארוך בסגנון 0xC000001 ואין לך מושג מה גרם לבעיה ואתה אבוד. בזכות הבאגים הללו הבנתי את הספרייה בצורה יותר טובה.

במבט לאחור כנראה שהייתי עושה את הפרויקט בצורה שונה. הייתי עושה אותו מהתחלה באופן מסודר, עם שלבים של מה אני עושה כל פעם ולא "הכול בבום". בנוסף, הייתי מנסה להבין איזה נושאים אני אוהב כי כמו שאמרתי, מאוד הסתבכתי עם בחירת נושא הפרוייקט.

אם היה לי יותר זמן הייתי מוסיף עוד שני דברים:

* שמע (קול), כרגע הסרטונים הם ללא שמע וביום יום אנשים רואים סרטונים תוך כדי שהם שומעים את הדברים בתוך הסרטונים.
* הסטרמה מהלקוח (שידור חי), הייתי רוצה להוסיף ללקוח את האפשרות לא רק לצפות בתכנים בשרת שמוכנים מראש (סרטון אשר שמור בשרת), אלא גם לצפות בשידורי חיים כמו שניתן לעשות זאת בtwitch וyoutube. בפועל, הטכנולוגיה כבר מוכנה מבחינת הפרויקט וצריך להוסיף פונקציה שבאמצעותה גם הלקוח יוכל לשלוח תוכן לצפייה לשרת.
* כל מיני פיצ'רים שיש ביוטיוב ונטפליקס כמו: סרטונים מומלצים לפי הסרטים שראית, דירוג סרטונים ועוד.

בנוסף, יכול להיות והייתי יכול לעשות כך שתעבורת הרשת תהיה יותר יעילה אבל לפחות כרגע זה עובד למרות שזה לא הכי יעיל כמו שזה יכול היה להיות.

בסופו של דבר, אני מרוצה מהתוצר שלי. הסרטון מורץ בצורה חלקה ואיכותית. מאוד נהניתי מהפרויקט ומתהליך הפיתוח. למדתי דברים וכלים חדשים לעתיד.