



## נגן Streaming



מגיש: יובל הושע

ת"ז: 325320695

מנחים: תומר טלגם, ירון יצחקי

תיכון: אהל שם

חלופה:

תאריך:

## תוכן עניינים

<b>4.....</b>	<b>מבוא.....</b>
4.....	למה בחרתי בפרויקט הזה ?
4.....	איך זה עובד ?
<b>6.....</b>	<b>ארכיטקטורה.....</b>
6.....	תרשים התקשורת בין הלקוח לשרת
6.....	הסבר.....
7.....	התקשורת בין השרת ללקוח
9.....	הסטרמת תמונות
<b>11.....</b>	<b>מסד הנתונים.....</b>
<b>12.....</b>	<b>מדריך למשתמש.....</b>
12.....	השרת
12.....	הלקוח
<b>16.....</b>	<b>מדריך למפתח.....</b>
16.....	מודולים עיקריים
16.....	תיקיית server
16.....	server.py
16.....	ThreadedClient.py
17.....	ServerConfig.py
17.....	socket_functions.py
17.....	database.py
17.....	env
18.....	תיקיית client
18.....	main.py
18.....	client.py
18.....	videoplayer.py
18.....	gui.py
18.....	dialogs.py
18.....	image_functions.py
19.....	ClientConfig.py
19.....	socket_fucntions.py
19.....	env
19.....	title.jpg
19.....	כללי

19.....requirements.txt

20.....רפלקציה

## 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

**NETFLIX**



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

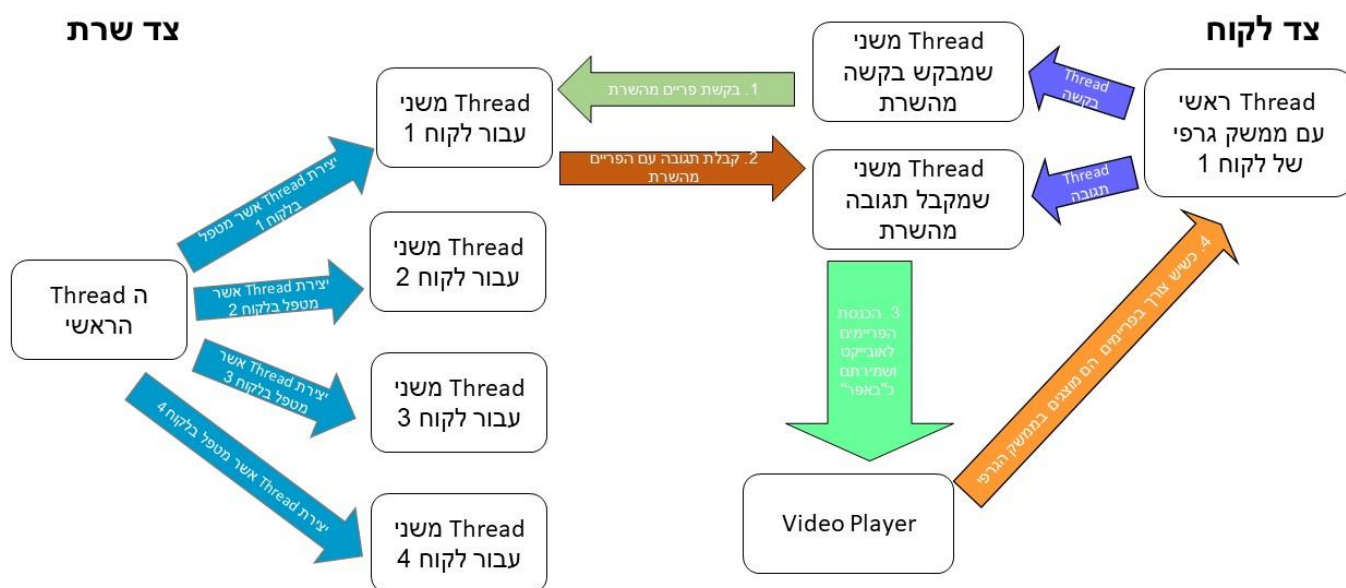
יעבוד. יש יותר מדי תמונות (הפריימים של הסרטון) אשר נשלחים כל שנייה. אם נשלח את כל התמונות כל רגע הסרטון יתקע. על מנת להקל על תעבורת הרשת דוחסים את הקובץ בשיטות שונות.

## 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.

### תרשים 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. ה Thread ירוץ כל עוד ה Thread הראשי רץ.

ה Thread הראשי ירוץ כל עוד לא יעצרו את התוכנה.

בצד הלקוח יש לקוח אשר מתחבר לשרת ומקבל ממנו

את כל התכנים שיש לו להציע. לאחר מכן, הלקוח בוחר את התוכן

הרצוי. ה Thread 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

```
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()
```

פעולת ההרצה של השרת

```
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)
```

ה Thread שמבקש פריימים מהשרת

מהשרת כי יש Thread  
נוסף שאחראי על הממשק  
הגרפי בו מציגים את  
הסרטון. ה Thread 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 הפריימים והלקוח מקבל את הפריימים.  
הלקוח מוסיף את הפריימים שקיבל לתוך אובייקט בשם  
VideoPlayer. האובייקט מכיל מבנה נתונים – תור  
אשר בתוכו נמצאים כל הפריימים שלא השתמשו בהם  
עוד.

```
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 ו Thread  
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

התקשורת כפי שצוין קודם מתרחשת באמצעות "סוקטים" (socket). התקשורת בין  
"הסוקטים" היא

```
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)
```

שימוש ב socket.SOCK\_STREAM ולא ב  
socket.SOCK\_DGRAM ובכך להשתמש ב TCP ולא ב UDP

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.

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

לאחר מכן אנחנו שולחים את כל הבתים.

קבלת המידע: לוקחים את header 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.



## 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 1200x900.

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 שבעזרתו בחרתי בפרוטוקול ה jpeg 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. כ-JPEG היא 12,000 בתים, 12 KB. עבור שאר הפורמטים האפשריים, גודל התמונה יותר גדול מאשר ב-JPEG ולכן בחרתי בפרוטוקול זה.

## מסד הנתונים

לתוכנה יש מסד נתונים. מסד הנתונים ממומש באמצעות SQLAlchemy.

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

מסד הנתונים:

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

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 את hash הסיסמאות. השרת מקבל את הסיסמה לפני שהיא הופכת לhash, ואז הופך את הסיסמה ל hash 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 hash:

```
d404559f602eab6fd602ac7680dacbfaadd13630335e951f097af3900e9de176b6db28
512f2e000b9d04fba5133e8b1c6e8df59db3a8ab9d60be4b97cc9e81db
```

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:

	username	password
	Filter	Filter
1	user	cdcc0069be983020d7c430419...
2	yuval	1aee6182437115a8a8d7d8579...
3	nice name	3e50b041b2feb5d6c073805ce...
4	US3R	d1c2e12cfeababc8b95daf6902...

## מדריך למשתמש

קישור לקוד: <https://github.com/Yuval-Hoshea/StreamingServer>

- client
- server
- venv
- requirements

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 הפרויקט שמכילה את כל המודולים שנמצאים בקובץ requirements.txt.

- ThreadedClient
- socket\_functions
- ServerConfig
- server
- database
- .env
- videos

### 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:

```
IP=0.0.0.0
PORT=19099
```

שורת ה IP זה IP 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.

לאחר שהגדרנו את כתובת ה IP והפורט בקובץ ".env", יש להריץ את הקובץ server.py. נריץ אותו באמצעות סביבה וירטואלית שתכיל את הספריות הנמצאות בקובץ requirement.txt שבתקייה הראשית של הפרויקט. נריץ ונקבל:

```
Select C:\WINDOWS\system32\cmd.exe
2021-06-19 12:27:23,299 [INFO]: LISTENING AT ('0.0.0.0', 19099)
```

זה אומר שהשרת פועל ומקשיב בכתובת שהפעלנו.



כל הסרטונים אשר השרת מציג נמצאים בתיקייה videos. התיקייה מכילה תיקיות אחרות אשר כל תיקייה היא סרטון אחר: בתוך כל תיקייה יש קובץ וידאו של הסרטון בשם <file\_type>.video, 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.

db.sqlite3

במידה 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:

- .env
- client
- ClientConfig
- dialogs
- gui
- image\_functions
- main
- socket\_functions
- title
- videoplayer

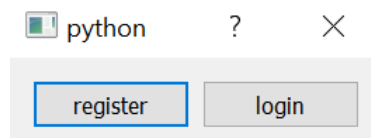
```
SERVER_IP=127.0.0.1
SERVER_PORT=19099
```

לפני הפעלת הלקוח נלך לקובץ env. שנראה כך:

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:

```
C:\WINDOWS\system32\cmd.exe
2021-06-19 12:39:21,728 [INFO]: LISTENING AT ('0.0.0.0', 19099)
2021-06-19 12:39:24,752 [INFO]: Got new client ('127.0.0.1', 1032).
2021-06-19 12:39:24,752 [INFO]: Starting new client thread!
```

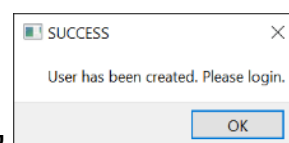
התווספו שני שורות ממקודם אשר מראות על כך שנוצר קשר עם לקוח והפעילו לו Thread שיטפל בו.

בחזרה ללקוח:

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

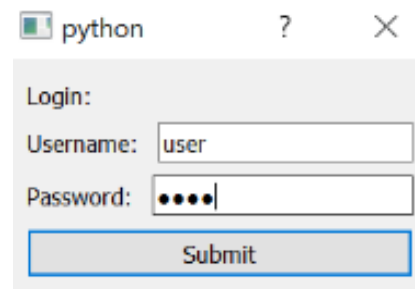
מסך ההרשמה:

כותבים את שם המשתמש והססמה.



קיבלנו הודעה שהחשבון נוצר.

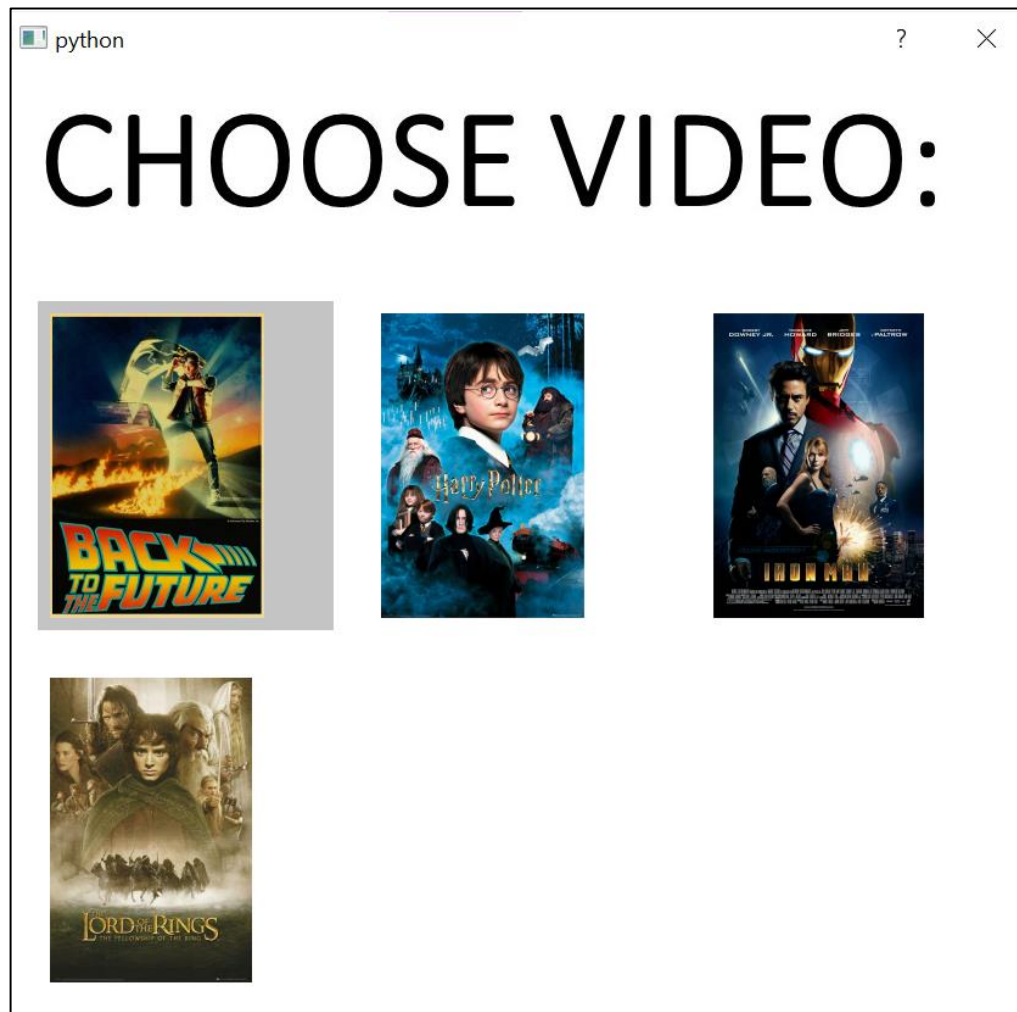
ואז חוזרים למסך הקודם. לוחצים על התחברות (login). ומגיעים למסך ההתחברות:



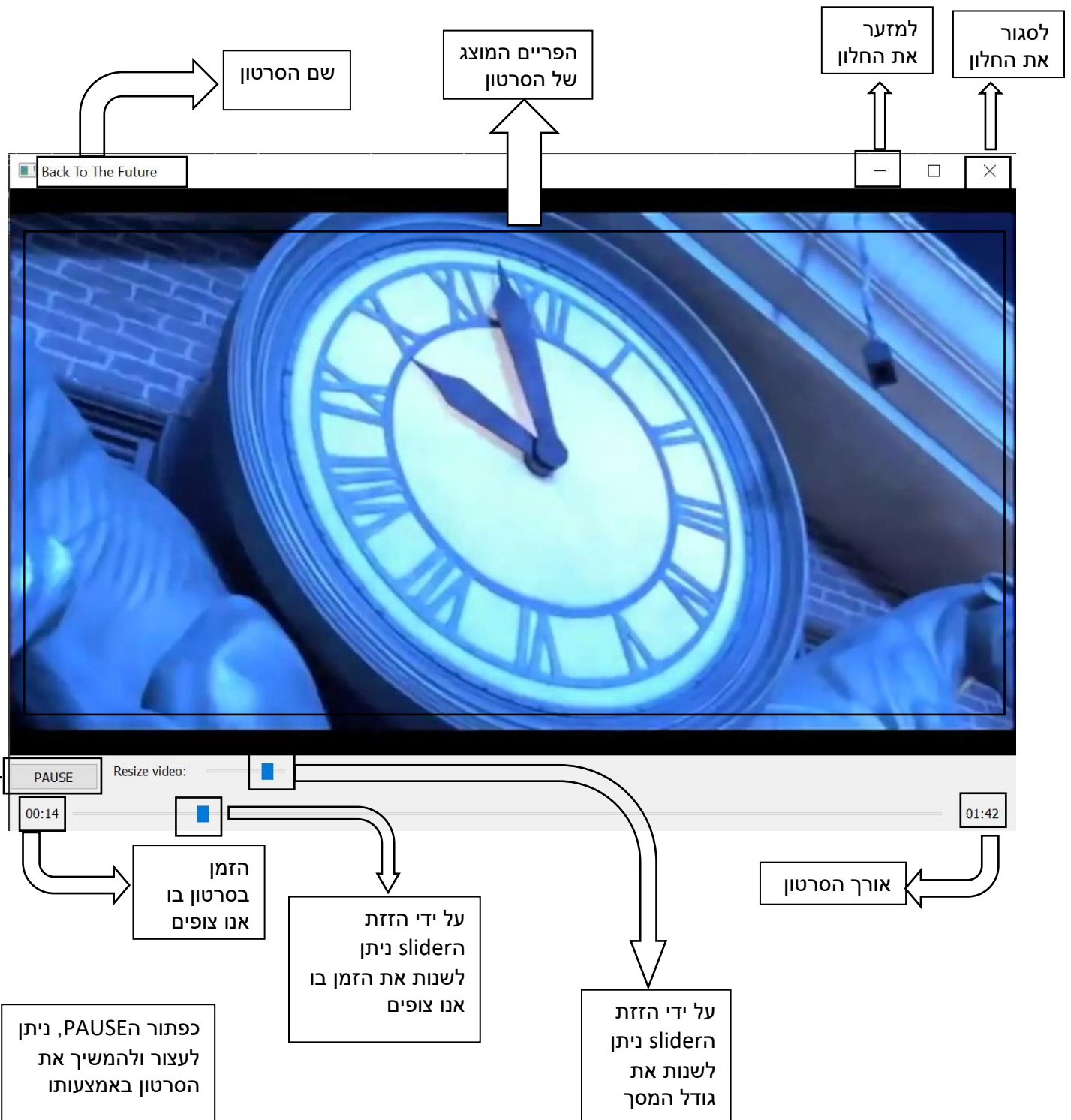
A small window titled 'python' with a question mark icon and a close button. It contains a login form with the following fields:

- Login:
- Username:
- Password:
- Submit button

ואז מתחברים. לאחר ההתחברות יפתח מסך הסרטים הקיימים:



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.



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 הפריימים, וכו'.



PyQt5 - היא אחת מן הספריות הכי נפוצות לבניית ממשקים גרפים בפייתון. היא תומכת בהרבה מערכות הפעלה כמו: windows, linux, 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 ב-ThreadedClient.py).

#### ThreadedClient.py

מכיל את המחלקה ClientThread אשר יורשת מן המחלקה Thread מספרייה threading. המחלקה הינה Thread אשר כפוף ל Thread הראשי של התוכנה הנמצא ב- server.py.

הפונקציות שהמחלקה מכילה:

- `__init__`, מקבל socket ואת הכתובת של הלקוח.



- run, פונקציה זאת מריצה את Threadn. לולאה אין סופית שבדקת אם התקבלה הודעה מ clientn, 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 logger, רשימת כל הסרטונים שקיימים. טעינת קובץ .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 (thumbnail).
- logger, בעזרתו דברים למסך לפי ההגדרה שנגדיר אותו, info, debug ועוד.
- ALL\_VIDEOS\_DIRECTORIES, VIDEOS\_DIR\_PATH, SERVER\_TIMEOUT, PORT, IP, משתנים גלובלים עם צרכים שונים בפרויקט

### socket\_functions.py

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

פונקציות:

- make\_header, יוצרת header לבקשות ותגובות הנשלחות בין סוקטים. בזכות headern ניתן לדעת את אורך הבקשה/תגובה.
- send\_data\_through\_socket, שולחת מידע דרך סוקט על ידי סירליזציה של הספרייה pickle.
- read\_data\_thorough\_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 headern ששולחים לפני כל בקשה/תגובה בתקשורת.
- משתני פונציות-ASK\_FOR\_VIDEO\_DETAILS, ASK\_FOR\_VIDEOS\_AVAILABLE, VIDEO\_THUMBNAIL, CHANGE\_VIDEO\_LOCATION, ASK\_FOR\_FRAME. 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_new_location`, `ask_for_frame`, `ask_for_video_details`
- `__handle_data`, הפונקציה שמטפלת במידע הנשלח מהשרת.

### videoplayer.py

הקובץ מכיל את המחלקה VideoPlayer אשר מכילה תור (queue) אשר מקבל פריימים ומוסיף אותו לתור. המחלקה אחראית על קבלת הפריימים, שמירתם, כמה זמן יש בין פריימים לפריימים, האם ניתן להוסיף עוד פריימים (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:

- Thread-AskingForFrameThread 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 ה`logger` ולקיחת ה`IP` וה`PORT` 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 ה`IP` וה`PORT` של השרת. משתמשים בערכים הללו בקובץ `ClientConfig.py`.

### title.jpg

תמונה שמשתמשים בה בקובץ `dialogs.py`

## **כללי**

### requirements.txt

קובץ הנמצא בתיקייה הראשית של הפרוייקט ולא בתיקיית השרת או הלקוח. מכיל את כל הספריות שצריך להתקין על מנת שהפרוייקט 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 בפיתוח. למדתי להשתמש בספריות חדשות שלא ידעתי להשתמש בהן לפני: OpenCV, numpy, PyQt5, sql. 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 הפריימים. הבעיה היא שמהרגע שהוא מבקש את הפריימים עד הרגע שהוא מקבל אותו עוברים כמה רגעים. באותם רגעים, עדיין יש מקום לעוד פריימים של הסרטון כי עוד לא קיבלנו את הפריימים הראשון שביקשנו. בגלל זה, הלקוח היה ממשיך את הפריימים 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 ב-RAM ולקח לי הרבה זמן להבין שזה הבאג. זהו רק באג אחד מתוך כמה שהיו. בנוסף, הספרייה PyQt5 לעתים קורסת עם קוד שגיא 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 הפרויקט.

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 ב-youtube ו-twitch. 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.