Sturs

בית הספר: ויצו הדסים 

שם התלמיד: אולג שקולניק

ת.ז.: 346700842

שם המנחה: ניר

מבוא

ייזום

Sturs - הוא שירות סטרימינג הנפתח באמצעות אתר אינטרנט. הוא נותן למשתמשים אפשרות להפעיל את הזרמים שלהם ולצפות בזרמים של אנשים אחרים. המשתמשים יוכלו להדגים את החיים שלהם, לדבר למצלמה ולשמור על קשר עם הצופים בשידור חי. המוצר המוגמר צריך לתת למשתמשים אפשרות להירשם בחופשיות ולקבל גישה לצפייה בזרמים של אחרים / הפעלת הזרמים שלהם.

בחרתי בפרויקט הזה מכיוון שאני מכיר זה זמן רב עם שירותים כאלה, הם קרובים אלי כמשתמש, ואני חושב שיהיה מעניין לממש פרויקט דומה.

המשימות העיקריות שאני רואה לפני הן להסתכל על עבודת שירותי הסטרימינג בעיני מפתח, ללמוד ביצוע מקצועי של משימות מבחינת מסגרות זמן וחלוקת עומסים, וכן להשלים בהצלחה את הפרויקט על ידי כתיבת קוד עובד, הסברים והגנה על הפרויקט.

הפרויקט שלי מיועד לשימוש כללי. Sturs יהיה נחוץ לאנשים הרוצים ליצור תוכן מקורי או לנהל שיחות רגילות, וכן לצופים המחפשים דרך לבלות בצורה מעניינת, להתבדר או לשמוע משהו חדש. המטרות של הפרויקט הן לתת למשתמשים ליצור חשבונות, לשתף תוכן או את החיים שלהם ולצפות בחופשיות בתוכן בשידור חי.

בשוק יש מספר מספיק של אפליקציות המספקות שירותי סטרימינג, למשל twitch או Youtube. בניגוד לדוגמאות שהוזכרו לעיל, sturs הוא שירות לשידורים חיים אך ורק של עצמך והחיים שלך למצלמה, גם ההבדל הגדול הוא שבאפליקציה שלי המשתמש לא יוכל להשפיע על הסטרים בשום צורה.

## **תחומים הנכללים בפרויקט**

**העברת נתונים:** שימוש בפרוטוקולי העברת נתונים (למשל, WebSocket ו-HTTP) לאינטראקציה בין הלקוח והשרת. זרמי וידיאו מועברים דרך webRTC, הדורש חיבור רשת קבוע להעברת הסטרים.

**פרוטוקולי רשת:** יישום וניהול רמות שונות של פרוטוקולים, כולל TCP/UDP, להבטחת העברת נתונים יציבה.

**פיצול:** חלוקת פריימי וידיאו/אודיו לחלקים והעברה עוקבת.

**אופטימיזציה של תעבורה:** הפרויקט כולל ניתוח ומזעור של תעבורת הרשת כדי להבטיח סטרימינג וידיאו איכותי גם ברוחב פס מוגבל.

**תת-מערכת רשת:** עיבוד בקשות רשת, כולל ניהול sockets ושמירה על חיבורים להעברת נתונים.

**מקביליות ורב-חוטיות:** עבודה עם threads ו-processes לעיבוד וידיאו, שירות לקוחות ואופטימיזציה של ביצועים.

**אבטחה:** הבטחת גישה בטוחה לנתונים והגנה מפני התקפות (הגנת עמודים וגישה לבסיס הנתונים).

**הגנה:** שימוש בהגנת SSL - עם מפתח ותעודה לתקשורת בטוחה בין השרת והלקוחות.

## **תחומים שאינם נכללים בפרויקט**

**ניהול רשתות ברמה נמוכה:** הפרויקט אינו מניח פיתוח של מנהלי התקנים לרשת או ניהול ציוד רשת (נתבים, מתגים וכו').

**חומרה:** הפרויקט אינו כולל עבודה עם היבטים של חומרה ברמה נמוכה, כמו הגדרת מצלמות, מיקרופונים או כרטיסי וידיאו. כל המכשירים נחשבים כתמוכים כבר על ידי מערכת ההפעלה.

**מונטיזציה ומערכות תשלום:** בפרויקט אין אלמנטים הקשורים לאינטגרציה של מערכות תשלום, חישוב עלות שירותים או ניהול כספים.

**מודלים של מחשוב מבוזר:** למרות שהפרויקט יכול להשתמש בשרתים, הוא אינו נוגע בהיבטים של מחשוב מבוזר מורכב (למשל, ניהול אשכול או רשתות חישוביות).

**ניתוח נתונים או למידת מכונה:** אין עבודה עם אלגוריתמי ניתוח נתונים, המלצות או אימון רשתות עצביות שיכולות לשמש לשיפור חוויית המשתמש.

## **פירוט תיאור המערכות**

**ייעוד כללי של המערכת:** המערכת מיועדת לארגון שידור זרמי וידיאו ואודיו בזמן אמת עם אפשרות חיפוש סטרימים, צפייה בשידורים, וכן ניהול חשבונות משתמשים.

### **יכולות פונקציונליות של המערכת**

#### **1. משתמשים**

המערכת מחלקת את המשתמשים לשני סוגים:

* **סטרימר:** משתמש היוצר שידורים.
* **צופה:** משתמש המצטרף לשידורים.

**פעולות עיקריות של משתמשים:**

**הרשמה והרשאה:**

* המשתמש יוצר חשבון עם שם משתמש וסיסמה ייחודיים.
* הרשאה באמצעות שם משתמש/סיסמה לגישה לחשבון האישי.

**צפייה בפרופילים:**

* צופים יכולים לחפש סטרימרים.

**הגדרת החשבון:**

* שינוי פרטים אישיים.

#### **2. חיפוש והתחברות לסטרימים**

**חיפוש סטרימרים:**

* המשתמש מזין שם סטרימר בשורת החיפוש.
* המערכת מחזירה תוצאות עם מידע על הסטרימרים שנמצאו וסטטוס הסטרימים שלהם.

**הצטרפות לשידור:**

* הצופה בוחר שידור ומתחיל לצפות בו בזמן אמת.

#### **3. יצירת סטרים (לסטרימרים)**

**זיהוי אוטומטי של התקני קלט:**

* מצלמה ומיקרופון נבחרים אוטומטית לשידור.

**הפעלת שידור:**

* זרם הוידיאו והאודיו מועבר עם עיכוב מינימלי לשרת, מעובד ומופץ לצופים.

**ניהול שידור:**

* הסטרימר יכול לסיים את הסטרים בכל רגע.

#### **4. העברת נתונים בזרימה**

**עיבוד וידיאו ואודיו:**

* זרמי וידיאו ואודיו מועברים מהסטרימר דרך השרת באמצעות פרוטוקול sockets הדומה ל-RTP.
* הזרמים מעובדים ומועברים לאתר בפרוטוקול הדומה ל-webRTC המבוסס על websocket.

#### **5. אבטחה ובקרת גישה**

**שאילתות SQL:**

* שאילתות SQL הנשלחות לשרת נבדקות לנוכחות הזרקות.

**גישה לעמודים:**

* גישה לעמודים יש רק למשתמשים מורשים - במקרה של הפעלה מחדש של השרת נדרשת הרשאה. כמו כן, המשתמש יכול לעבור לעמודים הרשומים ברשימת המותרים.

**הגנת SSL/TLS:**

* הגנת SSL הכוללת הצפנה אסינכרונית וסינכרונית של נתונים עם מפתחות ותעודות. כך מוצפנת התקשורת סטרימר-שרת, שרת-אתר (HTTPS).

תכנון בדיקות קופסא שחורה

| מספר  בדיקה | שם הבדיקה (שיעיד  על התוכן) | מה אמורה לבדוק | איך מתכננים לבדוק (לתאר בפירוט את שלבי  הבדיקה) |
| --- | --- | --- | --- |
| 1. | הרשמה/כניסה המשתמש למערכת | משתמש יכול להירשם או להיכנס למערכת עם לוגין וסיסמה | לפתוח את האתר, לבחור האם משתמש יוצר חשבון חדש או נכנס לחשבון הקיים.  בהרשמה להזין נתונים - לוגין וסיסמה, אם לוגין זה פנוי אז אישור הרשמה ומעבר לאתר עם החשבון הזה.  בכניסה לחשבון קיים - להזין לוגין וסיסמה, אם לוגין כזה קיים וסיסמה נכונה מעבר לאתר עם החשבון הזה. |
| 2. | חיפוש, בחירה של סטרימר והתחברות אליו | צופה יכול לכתוב שם הסטרימר, להסתכל האם יש לו סטרים עכשיו ואם כן - להתחיל להסתכל את הסטרים | לפתוח את מקטע החיפוש, להזין את שם הסטרימר, לראות רשימה של סטרימרים עם השמות הדומות, לבחור את הסטרימר המתאים לך, לראות אם יש לו שידור חי, להתחבר לשידור חי. |
| 3. | שידור של הסטרים | סטרימר יכול להפעיל את הסטרים בכל עת, וגם לעצור אותו | לבחור את מקטע ההתחלה של הסטרים. לבחור את המיקרופון והמצלמה שישתמשו בהם במהלך השידור, להתחיל את הסטרים, לראות כמה צופים יש בו כרגע, להפסיק את הסטרים בכל עת. |
| 4. | שינוי נתונים | כל משתמש יכול לשנות נתונים שלו | להיכנס למסך שינוי נתונים, להזין סיסמה/לוגין חדשים, אם לוגין פנוי - אישור שינוי הנתונים, אם לא - ביטול שינוי הנתונים. |
| 5. | כשל בשידור | עצירת סטרים במקרה בעיות מצד סטרימר או מצד צופה | לבדוק את חיבור הסטרימר, לעצור את הזרם, לעבור את המסך המתאים לצופים. |
| 6. |  |  |  |

תכנון לוחות זמנים

| פעילות | זמן התחלה מתוכנן | זמן סיום מתוכנן | זמן התחלה בפועל | זמן סיום בפועל | הערות |
| --- | --- | --- | --- | --- | --- |
| מבוא | 1/01 | 5/02 | 1/01 | 4/02 | סיום בזמן |
| תיאור תחום הידע - פרק מילולי | 1/01 | 5/02 | 1/01 | 4/02 | סיום בזמן |
| מבנה / ארכיטקטורה של הפרויקט | 1/01 | 5/02 | 1/01 | 4/02 | סיום בזמן |
| יצירת אתר | 7/02 | 21/02 | 7/02 | 19/02 | סיום לפני הזמן |
| עבודה עם פרוטוקול streamer - שרת | 25/02 | 3/03 | 25/02 | 3/03 | סיום בזמן |
| עבודה עם פרוטוקול שרת-צופה | 4/03 | 18/03 | 4/03 | 23/03 | סיום מאוחר |
| יצירת שרת לאתרים | 25/03 | 5/04 | 25/03 | 6/04 | סיום בזמן |
| עבודה עם פרוטוקול websocket | 7/04 | 20/04 | 7/04 | 18/04 | סיום קצת לפני הזמן |
| שינוי פרוטוקול streamer - שרת ל-udp | 23/04 | 2/05 | 23/04 | 5/05 | סיום מאוחר |
| כניסה ל-streamer | 6/05 | 9/05 | 6/05 | 9/05 | סיום בזמן |
| ssl-tls | 11/05 | 16/05 | 11/05 | 22/05 | סיום מאוחר |
| gui ל-streamer | 22/05 | 27/05 | 22/05 | 27/05 | סיום בזמן |
| שינוי וסיום ספר פרויקט | 28/05 | 3/06 | 28/05 | 3/06 | סיום בזמן |
|  |  |  |  |  |  |

ניהול הסיכונים בפרויקט

### **שגיאות בקוד**

**סיכון:** באגים בלתי צפויים העלולים לגרום לקריסות.  
**צעדים מתוכננים:**

* כיסוי הקוד בבדיקות.
* בדיקות אוטומטיות לפני פריסה.
* רישום שגיאות (לוגים).  
  **בוצע בפועל:**
* נכתבו בדיקות למודולים המרכזיים.
* נוספה מערכת asserts לשגיאות קריטיות.
* נוסף logging לתהליך איתור תקלות נוח יותר (debug).

### **ביצועי המערכת**

**סיכון:** עומס כבד על השרת עקב מספר משתמשים גדול, זרמי וידאו וצ'אט.  
**צעדים:**

* אופטימיזציה של עיבוד הווידאו בעזרת פרוטוקולים דמויי RTP ו-webRTC.
* הגבלת מספר החיבורים בו-זמנית לכל משתמש.

**סיכון:** אפשרות להתקפות (SQL Injection).  
**צעדים מתוכננים:**

* שימוש ב-HTTPS עם SSL.
* סינון קלט.  
  **בוצע בפועל:**
* הופעל הצפנת SSL.
* נעשה שימוש בשאילתות SQL עם פרמטרים.
* בוצע סינון לקלט מהמשתמש.

### 

### 

### **מדיניות פרטיות**

**סיכון:** דליפת מידע אישי של המשתמשים.  
**צעדים מתוכננים:**

* הצפנת מידע אישי.  
  **בוצע בפועל:**
* כל הסיסמאות השמורות מוצפנות (hashlib – SHA-256).
* קיימת הגנה מפני גישה בלתי מורשית.

תיאור תחום הידע - פרק מילולי

### 

### 

### **1. רישום במערכת**

### **מהות: רישום משתמש חדש במערכת, איסוף ובדיקת נתונים אישיים.**

### **רצף פעולות:**

### **המשתמש פותח את מסך הרישום.**

### **מזין שם משתמש וסיסמה.**

### **בדיקה בצד הלקוח (תוקף הנתונים: אורך סיסמה).**

### **הצפנת הסיסמה.**

### **שליחת הנתונים לשרת.**

### **בשרת:**

### **בדיקת ייחודיות שם המשתמש.**

### **אחסון סיסמה מוצפנת במסד הנתונים.**

### **שליחת אישור הצלחה או שגיאה.**

### **בצד הלקוח:**

### **קבלת תגובה והצגת הודעה מתאימה.**

### **אובייקטים נדרשים:**

### **ממשק משתמש: מסך רישום.**

### **שרת: טיפול בבקשות.**

### **מסד נתונים: אחסון נתוני משתמשים.**

### **הצפנה/פענוח: עבודה עם סיסמאות.**

### 

### **2. התחברות למערכת**

### **מהות: בדיקת פרטי התחברות ומתן גישה לפונקציונליות האתר.**

### **רצף פעולות:**

### **המשתמש מזין שם משתמש וסיסמה במסך ההתחברות.**

### **הלקוח מצפין את הסיסמה ושולח לשרת.**

### **השרת מחפש את המשתמש לפי שם המשתמש ומשווה סיסמאות (עם הסיסמה המוצפנת).**

### **אובייקטים נדרשים:**

### **ממשק משתמש: מסך התחברות.**

### **שרת: בדיקת נתונים.**

### **מסד נתונים: בדיקת חשבונות.**

### 

### **3. חיפוש סטרימרים**

### **מהות: חיפוש משתמשים עם שידורים פעילים או שהסתיימו.**

### **רצף פעולות:**

### **המשתמש מזין את שם הסטרימר או מילות מפתח בשורת החיפוש.**

### **הלקוח שולח בקשת חיפוש לשרת.**

### **השרת מחפש רשומות במסד הנתונים.**

### **מזהה את מצב הפעילות של הסטרימר.**

### **מחזיר רשימת פרופילים מתאימים.**

### **הלקוח מציג את תוצאות החיפוש.**

### **אובייקטים נדרשים:**

### **ממשק משתמש: שורת חיפוש, תוצאות חיפוש.**

### **שרת: טיפול בבקשות חיפוש.**

### **מסד נתונים: אחסון מידע על משתמשים ופעילותם.**

### 

### **4. יצירת שידור (סטרימר)**

### **מהות: שידור וידאו מהסטרימר לשרת עם אפשרות לצפייה מהקהל.**

### **רצף פעולות:**

### **הסטרימר מתחבר לשרת.**

### **הלקוח מאתחל את זרם הנתונים.**

### **הזרם נשלח לשרת.**

### **בשרת:**

### **עיבוד הזרם.**

### **עדכון סטטוס המשתמש כ"משדר".**

### **הצופים מקבלים גישה לשידור דרך רשימת השידורים הפעילים.**

### **אובייקטים נדרשים:**

### **לקוח: מצלמה, מיקרופון, מודול שליחת זרמים.**

### **שרת: עיבוד וידאו/אודיו.**

### 

### **5. הצטרפות לשידור (צופה)**

### **מהות: צפייה בשידור חי בזמן אמת.**

### **רצף פעולות:**

### **הצופה בוחר שידור מרשימת השידורים.**

### **הלקוח שולח בקשה לשרת לקבלת הזרם.**

### **השרת מתחיל את השידור.**

### **הלקוח מציג את הווידאו.**

### **אובייקטים נדרשים:**

### **שרת: העברת נתוני השידור.**

### **מסד נתונים: סטטוס השידורים.**

### 

### **6. סיום שידור**

### **מהות: סיום שידור על ידי הסטרימר.**

### **רצף פעולות:**

### **הסטרימר לוחץ על כפתור "סיום".**

### **הלקוח מודיע לשרת על סיום השידור.**

### **השרת מפסיק את עיבוד הווידאו.**

### **השרת מעדכן את סטטוס הסטרימר במסד הנתונים.**

### **אובייקטים נדרשים:**

### **לקוח: כפתור סיום.**

### **שרת: סיום הזרם.**

### **מסד נתונים: סטטוס פעילות.**

### 

### **7. עדכון פרופיל**

### **מהות: עריכת נתוני חשבון המשתמש.**

### **רצף פעולות:**

### **המשתמש מזין נתונים חדשים (למשל סיסמה).**

### **הלקוח שולח בקשה לשרת.**

### **השרת מעדכן את הנתונים במסד.**

### **הלקוח מקבל אישור.**

### **אובייקטים נדרשים:**

### **לקוח: ממשק עריכת פרופיל.**

### **שרת: עדכון נתונים.**

### **מסד נתונים: אחסון נתוני פרופיל.**

### .

מבנה / ארכיטקטורה של הפרויקט

**תיאור ארכיטקטורת המערכת המוצעת**

**המערכת מבוססת על ארכיטקטורת לקוח-שרת פשוטה.  
היא תומכת בהזרמת וידאו, ניהול חשבונות משתמשים וצ'אט בסיסי בזמן אמת.**

### **1. חומרה**

**חומרת השרת  
שרת מקומי (All-in-one):**

* **מטפל בבקשות משתמשים, מנהל זרמי וידאו, שומר נתונים ומספק אינטראקציה בין הרכיבים.**
* **נפרס על מחשב אישי.**

**חומרת הלקוח**

* **סטרימר: מחשב אישי או מחשב נייד עם מצלמה ומיקרופון.**
* **צופה: מחשב אישי או מחשב נייד.**

### **2. רכיבי המערכת**

**Frontend (צד לקוח)  
אפליקציית אינטרנט פשוטה הפועלת דרך דפדפן.**

**פונקציות עיקריות:**

* **רישום והתחברות משתמשים.**
* **ממשק לחיפוש סטרימים והצטרפות אליהם.**
* **ניגון זרם וידאו.**
* **צ'אט אינטראקטיבי.**

**מימוש:**

* **HTML/CSS לממשק המשתמש.**
* **JavaScript לאינטראקציה עם השרת (sockets, בקשות HTTP).**

**Backend (צד שרת)  
השרת כתוב ב-Python תוך שימוש בספריית socket.**

**פונקציות עיקריות:**

* **טיפול ברישום, התחברות ופרופילים.**
* **קבלת זרמי וידאו מהסטרימר.**
* **שידור הווידאו לצופים.**
* **טיפול בהודעות צ'אט.**

**אחסון נתונים:**

* **PostgreSQL עבור בסיס נתונים קל לפרויקטים קטנים.**

**שרת הווידאו**

* **משולב בצד השרת.**
* **מקבל את זרם הווידאו מהסטרימר (RTP).**
* **מעבד ומעביר את הנתונים בפרוטוקול דומה ל-WebRTC.**

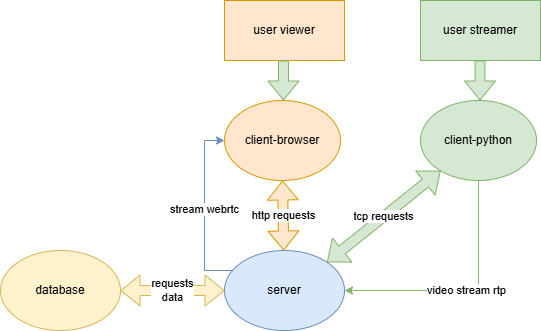
**מסד הנתונים**

* **MySQL:  
  לאחסון חשבונות משתמשים ומידע על שידורים.**

### **3. קשרים בין רכיבים**

* **סטרימר → שרת:  
  הסטרימר מתחבר ומעביר את זרם הווידאו.**
* **שרת → צופה:  
  השרת מעבד את הזרם ומעביר אותו לצופים דרך queue.**
* **צופה → שרת:  
  הצופה מתחבר לשרת וצופה בשידור דרך WebSocket.**
* **שרת → מסד נתונים:  
  נשמרים נתוני משתמשים ומידע על שידורים.**

שרטוט



### 

### **תיאור הטכנולוגיה הרלוונטית**

#### **1. שפת תכנות**

**Python:**

* שפת הפיתוח העיקרית בצד השרת.
* קלה לשימוש, כוללת ספריות לעבודה עם סוקטים וזרמי וידאו.
* מתאימה לפיתוח מהיר ובדיקת אבטיפוס.

**HTML, CSS, JavaScript:**

* משמשות בצד הלקוח (Frontend).
* HTML ו-CSS — ליצירת ממשק המשתמש.
* JavaScript — לאינטראקציה עם השרת (בקשות HTTP ו-WebSocket לשידור).

#### **2. תקשורת**

**Sockets (WebSocket):**

* משמשים לקבלת זרמים בזמן אמת.
* מאפשרים תקשורת דו-כיוונית בין השרת ללקוח.

**HTTP:**

* משמש לטיפול בבקשות רישום, התחברות, חיפוש סטרימים והצטרפות לוידאו.

**RTP / WebRTC:**

* **RTP (Real-time Transport Protocol):**משמש להעברת וידאו מהסטרימר לשרת.
* **WebRTC:**זרם הווידאו נשלח מהשרת לדפדפן המשתמש.  
  מתבצע תקשורת JSON בין הצדדים.

#### **3. תחומי עניין**

* **אימות משתמשים:**רישום, התחברות ויציאה מהמערכת.
* **שידור וידאו:**ארגון העברת זרם וידאו מהסטרימר לצופים.
* **ממשק נוח:**עיצוב פשוט וברור לשימוש אישי.

### **תיאור זרימת המידע במערכת**

**דוגמה: צפייה בשידור על ידי צופה**

**תחילת הזרם:**

* הסטרימר מתחיל את השידור ומתחבר לשרת דרך פרוטוקול RTP.
* שרת הווידאו מקבל את הזרם, מעבד אותו ושולח ללקוחות באמצעות WebSocket.

**עיבוד בשרת:**

* השרת רושם את הזרם כפעיל.
* מעדכן את הנתונים במסד (מידע על השידור, צופים זמינים).

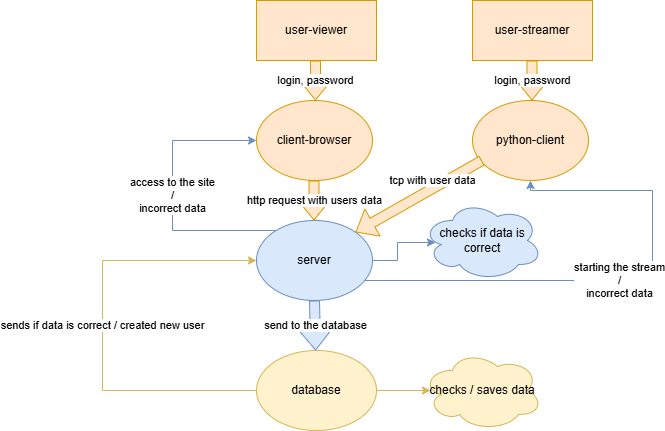
**בקשה מהצופה:**

* הצופה נכנס לאתר ושולח בקשה לצפות בשידור.
* השרת בודק את זמינות הזרם ומתחיל להעביר אותו.

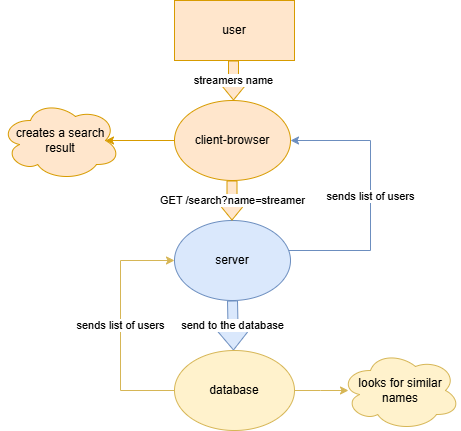
**סיום הזרם:**

* הסטרימר מסיים את השידור.
* השרת סוגר את הזרם, מעדכן את מסד הנתונים ומודיע לצופים.

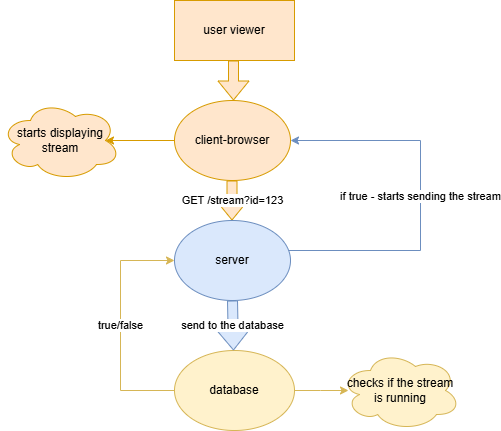
הרשמה / כניסה



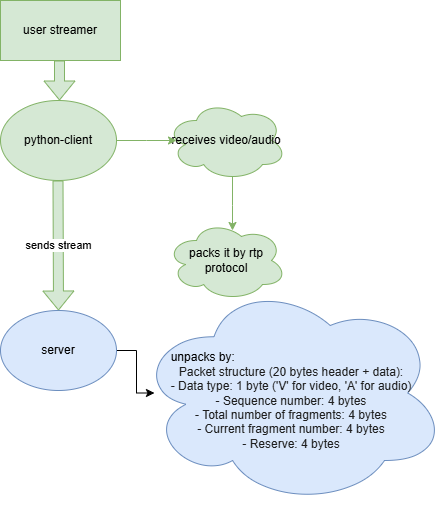
חיפוש



חיבור



העברת וידאו



### 

### 

### 

### 

### 

### 

### **תיאור מסכים עיקריים בפרויקט**

### **1. הגדרת וניתוח הבעיה האלגוריתמית**

#### **משימות עיקריות של הפרויקט:**

### **הרשמה ואימות משתמשים: אבטחת אחסון נתונים ובדיקת חשבונות משתמשים. טיפול מהיר בבקשות אימות.**

### **העברת וידאו מהמשדר לצופים: מזעור השהיות בזמן העברת הנתונים. התאמה דינמית של איכות הזרם למהירות האינטרנט של המשתמש.**

### **חיפוש ובחירת שידורים פעילים: חיפוש שידורים במסד הנתונים לפי בקשת המשתמש.**

### **ניהול סשנים: הפיכת HTTP ל"stateful" במקום "stateless". זיהוי מאובטח של משתמשים. שליטה בגישה למשאבים. התאמה אישית של חוויית המשתמש. שמירת מצב בין בקשות.**

### 

### **2. תיאור האלגוריתמים הקיימים לפתרון**

#### **2.1 הרשמה ואימות משתמשים**

### **אלגוריתמים של גיבוב סיסמאות: שמירת סיסמאות כ-hash לצורכי אבטחה. דוגמאות לאלגוריתמים: hashlib / bcrypt**

### 

#### **2.2 העברת וידאו**

### **RTP (Real-time Transport Protocol): פרוטוקול להעברת מדיה עם השהייה מינימלית (למשל, שיחות וידאו/שידורים). מהיר מאוד אך פחות אמין (עשוי לאבד מנות). פועל מעל UDP.**

### **מבנה מנות וידאו מקוטעות: כותרת של 20 בתים + נתונים:**

### **סוג הנתון: 1 בית ('V' לווידאו, 'A' לאודיו)**

### **מספר רצף (Sequence): 4 בתים**

### **מספר כולל של מקטעים: 4 בתים**

### **מספר המקטע הנוכחי: 4 בתים**

### **שדה שמור: 4 בתים**

### **נתונים: עד 1024 בתים**

### **WebRTC (דרך WebSocket):**

### **העברה: הודעות JSON דרך WebSocket**

### **וידאו: פריימים בפורמט JPEG, מקודדים ב-Base64**

### **אודיו: PCM 16-bit, תדר 44100Hz, מונו, גם כן ב-Base64**

### **בצד הלקוח:**

### **קבלה דרך WebSocket**

### **פענוח Base64 → נתונים בינאריים**

### **וידאו: תצוגה ישירה ב-<img>**

### **אודיו: Web Audio API לניגון עם השהייה מינימלית**

### **שמירה על חיבור פעיל:**

### **פינג-פונג כל 30 שניות**

### **מעקב אחרי פעילות**

### **התחברות אוטומטית מחדש במקרה של ניתוק**

### 

#### **2.4 חיפוש שידורים פעילים**

### **שאילתות למסד הנתונים: שימוש ב-SQL לסינון שידורים לפי שם ומצב (פעיל/לא פעיל). אופטימיזציה: אינדקסים על עמודות החיפוש.**

### 

### **3. סקירת הפתרון הנבחר**

#### **3.1 הרשמה ואימות**

### **פתרון נבחר: hashlib עם SHA-256 לגיבוב סיסמאות — אמין, עם הגנה בסיסית מובנית.**

### **פתרונות שנדחו:**

### **שמירת סיסמאות גלויה במסד (לא בטוח)**

### **שימוש ב-MD5 (חלש מדי לאבטחה מודרנית)**

### 

#### **3.2 העברת וידאו**

### **פתרון נבחר: שימוש ב-WebRTC להעברה מהמשדר לשרת**

### **יתרון: השהייה מינימלית**

### **חסרון: מורכב יחסית, אולי מוגזם לשימוש פרטי**

### **פתרון שנדחה:**

### **HTTP Progressive Streaming**

### **פחות גמיש**

### **לא מאפשר דילוג**

### 

#### **3.4 חיפוש שידורים פעילים**

### **פתרון נבחר: SQL עם אינדקסים לפי שם ומצב**

### **פתרון שנדחה: סינון בצד התוכנה בלי אינדקסים (איטי מאוד במסדי נתונים גדולים)**

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### **תיאור סביבת הפיתוח**

#### **כלים דרושים**

### **שפת תכנות: Python — לשני הצדדים (שרת ולקוח), מאפשרת שליטה ידנית בפרוטוקולים.**

### **ספריות:**

### **socket: לתקשורת ישירה**

### **threading / multiprocessing: לטיפול במספר חיבורים במקביל**

### **struct: לקידוד/פענוח של נתונים בינאריים**

### **OpenCV (אופציונלי): לעיבוד וידאו**

### **sounddevice: לעבודה עם אודיו**

### **כלי פיתוח:**

### **VS Code או PyCharm**

### **Wireshark: לניתוח תעבורה**

### **Git: לניהול גרסאות**

### 

#### **סביבת בדיקה**

### **שרת מקומי באותו מחשב של הפיתוח**

### **מספר לקוחות (מכונות וירטואליות או מחשבים אחרים)**

### **חיבור לרשת אמיתית לבדיקות**

### **מעקב ובדיקה:**

### **Wireshark**

### **טיימרים ולוגים בקוד**

### 

### **תיאור פרוטוקול התקשורת**

### **רמת פרוטוקול נמוכה (Low-level socket):**

### **TCP: לפקודות בקרה**

### **UDP: להעברת זרמי וידאו/אודיו**

### **מבנה חבילות:**

### **כותרת: סוג, גודל, מזהה משתמש**

### **גוף: נתוני וידאו/אודיו/טקסט**

### **סנכרון זרמים:**

### **בעיה: UDP לא מבטיח הגעה**

### **פתרון:**

### **מיספור רצף**

### **טיימאוטים**

### **דחיית חבילות ישנות**

### **בעיה נוספת: אי סנכרון בין וידאו (30FPS) לאודיו**

### **פתרון:**

### **באפר נפרד לכל סוג**

### **מיספור חבילות**

### **שמירת הפריים האחרון**

### **תור שידור מאוחד**

### 

### **מבנה המערכת (מספר "בתים")**

### **בית 1 – צד שרת: מנהל חיבורים, מעביר נתונים בין משדרים לצופים, שולח פקודות בקרה.**

### **בית 2 – לקוח-משדר: לוכד אודיו/וידאו מהחומרה, מקודד ושולח לשרת.**

### **בית 3 – לקוח-צופה: מקבל זרמים מהשרת, מפענח, מציג ומאזין.**

### **בית 4 – מסד נתונים: שומר נתוני משתמשים, שידורים פעילים ועוד.**

### **בית 5 – לוגיקת סנכרון זרמים: באפרים לאיחוד חבילות, שמירת פריים אחרון, תור להזרמה חלקה.**

### .

### **פיצול כל ההודעות העוברת במערכת**

#### **1. בקשות POST עם טפסים (application/x-www-form-urlencoded)**

**רישום משתמש**מהאתר לשרת:

POST /register HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 35

username=john\_doe&password=mypassword123

**התחברות משתמש**מהאתר לשרת:

POST /login HTTP/1.1

Content-Type: application/x-www-form-urlencoded

Content-Length: 35

username=john\_doe&password=mypassword123

**שינוי סיסמה**מהאתר לשרת:

POST /change-password HTTP/1.1

Cookie: session\_id=a1b2c3d4e5f6789

Content-Type: application/x-www-form-urlencoded

Content-Length: 65

current-password=oldpass123&new-password=newpass456

#### **2. תגובות JSON מהשרת**

**חיפוש סטרימרים**בקשה מהאתר:

sql

GET /search?term=john HTTP/1.1

Cookie: session\_id=a1b2c3d4e5f6789

תגובה מהשרת:

json

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 42

[

"john\_doe",

"johnny\_stream",

"john\_player"

]

**בדיקת פעילות סטרים**בקשה מהאתר:

GET /join-stream?streamer=john\_doe HTTP/1.1

Cookie: session\_id=a1b2c3d4e5f6789

תגובה מהשרת:

json

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 89

{

"active": true,

"exists": true,

"message": "Stream is active"

}

**תוצאה של שינוי סיסמה**תגובה מהשרת:

json

HTTP/1.1 200 OK

Content-Type: application/json; charset=utf-8

Content-Length: 56

{

"success": true,

"message": "Password changed successfully"

}

**במקרה של שגיאה:**

json

HTTP/1.1 401 Unauthorized

Content-Type: application/json; charset=utf-8

Content-Length: 78

{

"success": false,

"message": "Failed to identify user. Please login again."

}

#### **3. הפניות HTTP (302)**

**רישום הצליח**תגובה מהשרת:

HTTP/1.1 302 Found

Location: /registration-success.html

Content-Length: 0

Connection: close

**התחברות הצליחה**תגובה מהשרת:

HTTP/1.1 302 Found

Location: /search-stream.html

Set-Cookie: session\_id=a1b2c3d4e5f6789; Path=/

Content-Length: 0

Connection: close

**יציאה מהמערכת**תגובה מהשרת:

HTTP/1.1 302 Found

Location: /login.html

Set-Cookie: session\_id=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT

Content-Length: 0

Connection: close

#### 

#### 

#### 

#### 

#### 

#### **הודעות WebSocket (פורמט JSON)**

json

// אתחול חיבור

{

"action": "join",

"streamer": "streamer\_name"

}

// נתוני וידאו

{

"type": "video",

"data": "base64\_encoded\_jpeg"

}

// נתוני אודיו

{

"type": "audio",

"data": "base64\_encoded\_pcm"

}

// הודעות מערכת

{

"type": "info|error|audio\_init|pong",

"message": "text\_message"

}

// פינג לשמירת החיבור

{

"action": "ping"

}

#### **נתוני UDP לשידור (בינאריים)**

**נתוני וידאו מקוטעים**מבנה החבילה (20 בייט של כותרת + נתונים):

* סוג נתונים: 1 בייט (V לוידאו, A לאודיו)
* מספר רצף (Sequence number): 4 בייט
* מספר כולל של מקטעים: 4 בייט
* מספר המקטע הנוכחי: 4 בייט
* שמור (Reserve): 4 בייט
* תוכן מועיל (Payload): עד 1024 בייט

#### 

**תיאור המסכים של המערכת**

#### **1. מסך רישום (Sign-Up)**

**תפקיד המסך**: רישום משתמשים חדשים.  
**תיאור**: מסך זה מאפשר למשתמש ליצור חשבון חדש. המשתמש מזין שם משתמש, סיסמה ואישור סיסמה.  
**פונקציונליות**:

* הזנת שם משתמש וסיסמה
* כפתור להשלמת הרישום
* טיפול בשגיאות (למשל, אם שם המשתמש כבר תפוס או אם הסיסמאות אינן תואמות)
* הפניה למסך ההתחברות לאחר רישום מוצלח

#### **2. מסך התחברות (Login)**

**תפקיד המסך**: התחברות המשתמש.  
**תיאור**: המשתמש מזין את פרטי הגישה שלו (שם משתמש וסיסמה) כדי להיכנס למערכת.  
**פונקציונליות**:

* שדות להזנת שם משתמש וסיסמה
* כפתור "התחבר"
* קישור למסך רישום עבור משתמשים חדשים

#### **3. דף הבית (Home Page)**

**תפקיד המסך**: המסך הראשי של האפליקציה עבור צופים.  
**תיאור**: מציג רשימת שידורים פעילים ופונקציית חיפוש.  
**פונקציונליות**:

* רשימת סטרימרים שמשדרים כעת
* שדה חיפוש לפי שם סטרימר
* כפתור להצטרפות לשידור
* אפשרות לשנות פרופיל — לערוך פרטי משתמש

#### **5. מסך צפייה בשידור (Stream Viewer)**

**תפקיד המסך**: צפייה בשידור חי בזמן אמת.  
**תיאור**: המשתמש צופה בזרם הווידאו ויכול לשלוט בפרטי הצפייה.  
**פונקציונליות**:

* זרם וידאו ואודיו מהסטרימר
* שליטה על עוצמת הקול או השתקה
* אפשרות להתנתק מהשידור

#### **6. מסך פרופיל משתמש**

* אפשרות לשנות סיסמה על ידי הזנת הסיסמה הישנה ואישור כפול של הסיסמה החדשה
* אפשרות להתנתק מהחשבון

#### **7. אפליקציית הסטרימר (Streamer Application)**

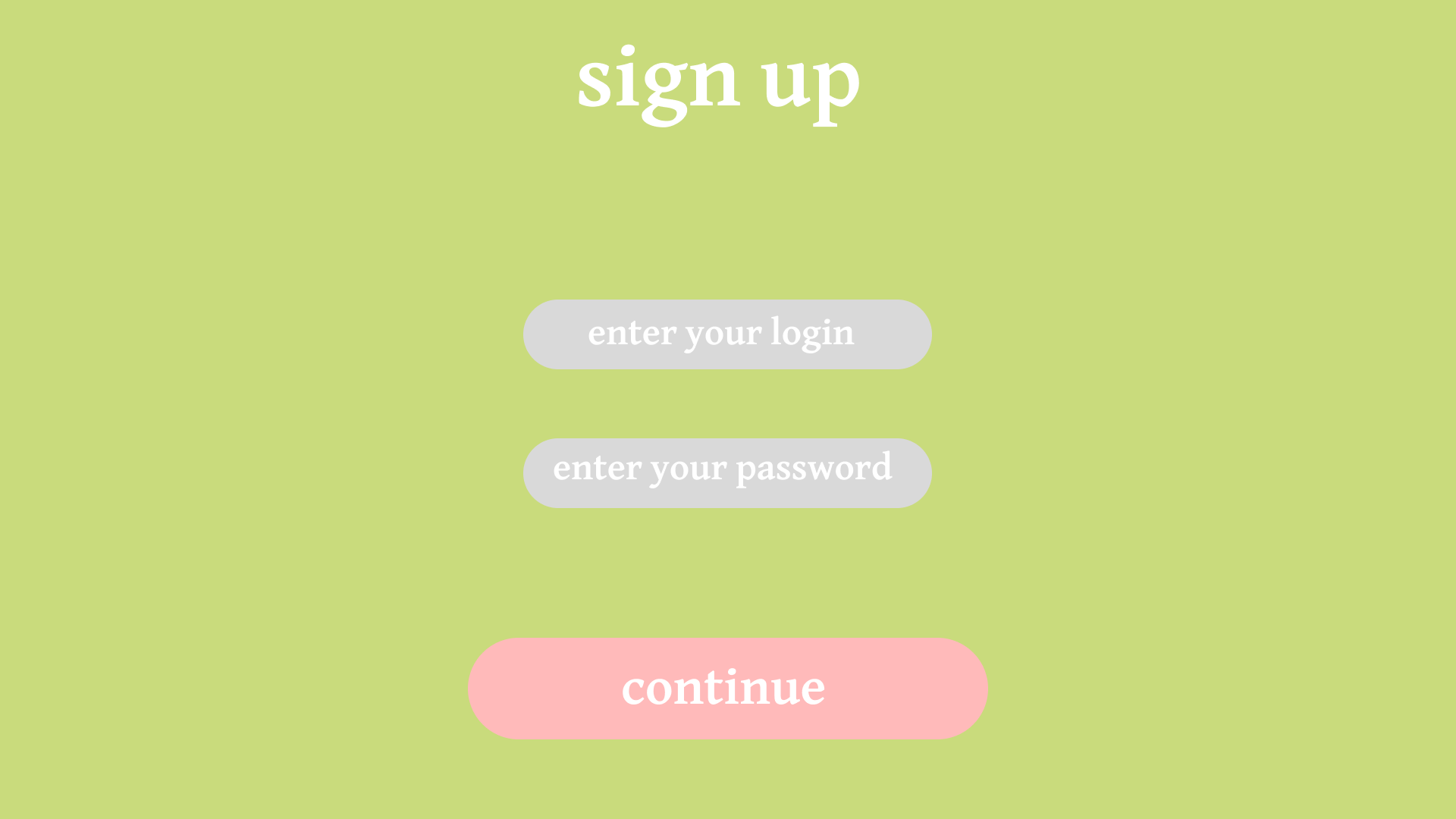
**תפקיד המסך**: התחברות וניהול שידור עבור הסטרימר.  
**תיאור**: הסטרימר בוחר שרת, מתחבר, מתחיל שידור ויכול לסיים אותו.  
**פונקציונליות**:

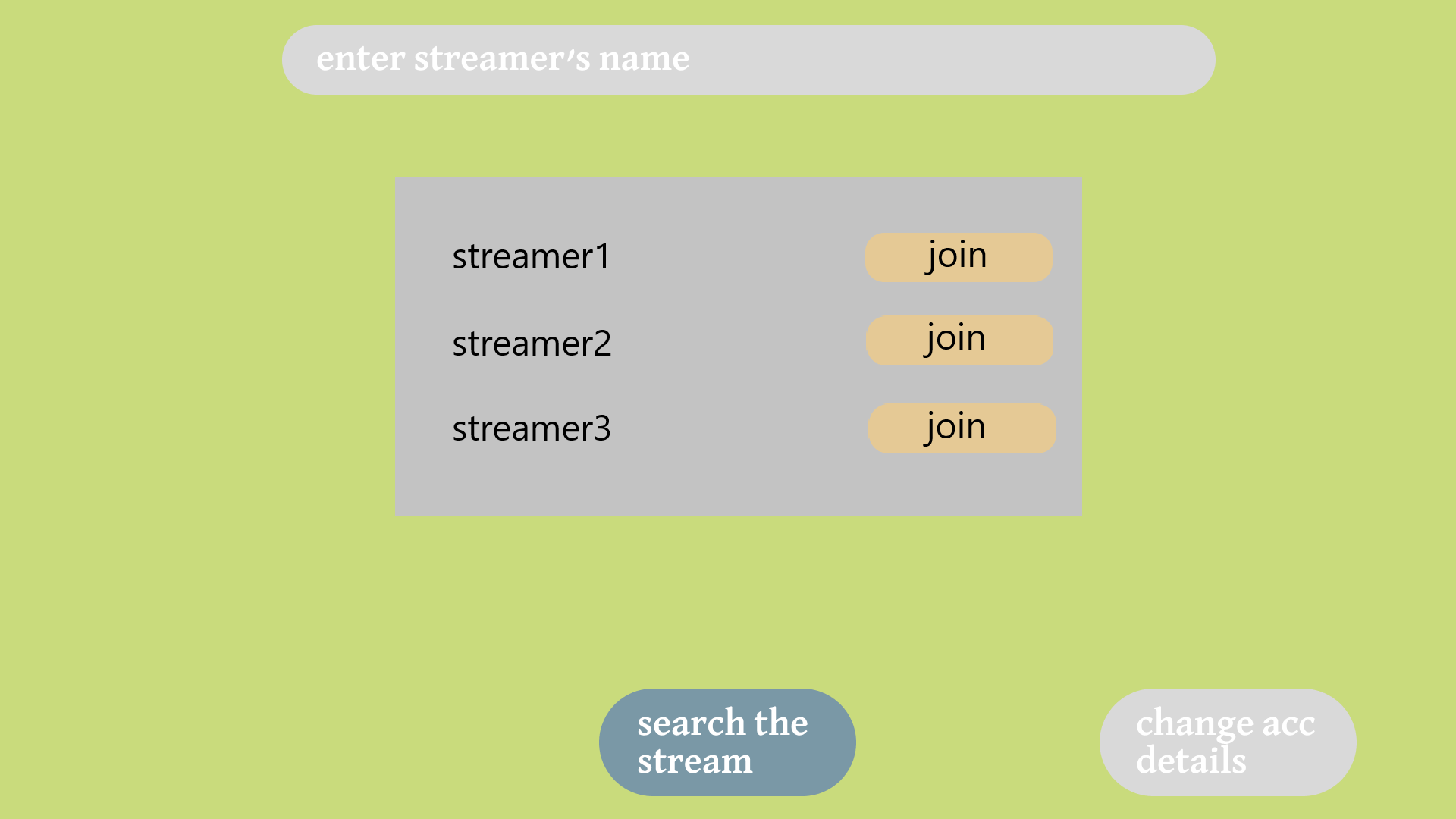
* התחברות ובחירת כתובת IP ו־port של השרת
* כפתור "סיים שידור"

### **מעברים בין מסכים**

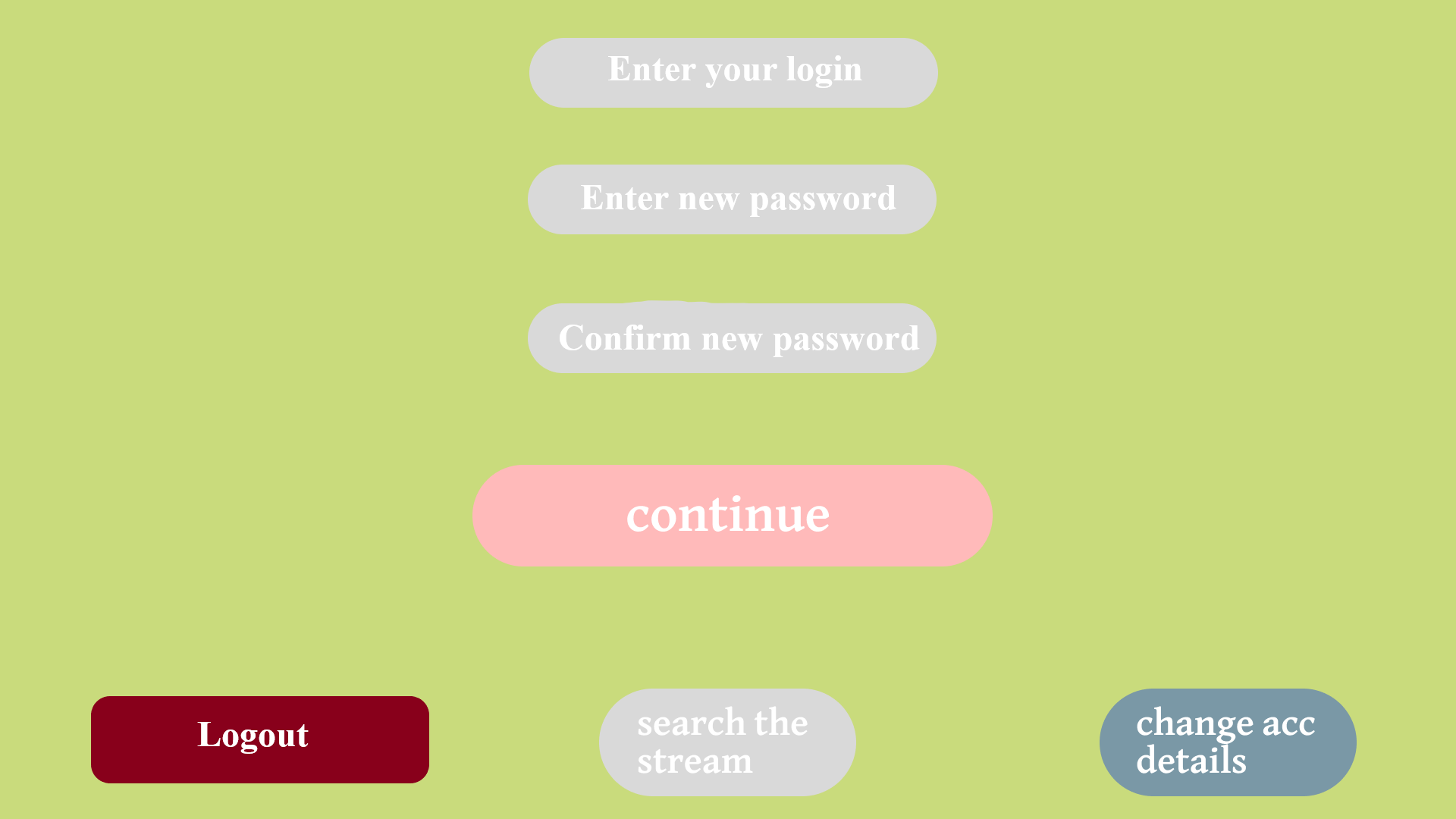
* אם המשתמש אינו רשום — הוא מועבר למסך הרישום
* לאחר רישום/התחברות מוצלחת — מעבר לדף הבית
* מהדף הראשי ניתן לעבור למסך צפייה בשידור
* מהשידור ניתן לעבור לעמוד הפרופיל, עם אפשרות יציאה וחזרה

שרטוטים









**מסדי נתונים**

שם מסד הנתונים: user\_database  
מסד הנתונים יכיל מספר טבלאות לאחסון מידע על משתמשים, סטרימינג. כל טבלה תכיל שדות ייחודיים לזיהוי רשומות.

### **מבנה טבלאות מסד הנתונים**

מסד הנתונים user\_database.db נוצר באמצעות SQLite ומכיל שתי טבלאות עיקריות:

1. טבלת users

sql

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT UNIQUE NOT NULL,

password TEXT NOT NULL

)

שדות:  
id - מזהה ייחודי למשתמש (אוטו-אינקרמנט)  
username - שם משתמש (ייחודי, חובה)  
password - סיסמת המשתמש בצורה מוצפנת (SHA-256)

1. טבלת active\_streams

sql

CREATE TABLE IF NOT EXISTS active\_streams (

username TEXT PRIMARY KEY,

start\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (username) REFERENCES users(username)

)

שדות:  
username - שם המשתמש של הסטרימר (מפתח ראשי)  
start\_time - זמן תחילת הסטרים (נקבע אוטומטית)  
מפתח זר המקשר לטבלת users

### **יכולות עיקריות**

**ניהול משתמשים:**

* רישום - יצירת משתמשים חדשים עם בדיקת ייחודיות השם
* אימות - בדיקת שם משתמש וסיסמה (הסיסמאות מוצפנות SHA-256)
* שינוי סיסמה - שינוי סיסמה עם בדיקת הסיסמה הנוכחית
* ניהול סשנים - יצירה ומחיקה של סשנים למשתמשים

**ניהול סטרימים:**

* מעקב אחר סטרימים פעילים - רישום מידע על מי סטרים כרגע
* חיפוש סטרימרים - חיפוש משתמשים לפי שם עם אפשרות לסינון
* בדיקת סטטוס סטרים - קביעה אם הסטרים פעיל עבור משתמש מסוים

**אבטחה:**

* סיסמאות מאוחסנות בצורה מוצפנת (SHA-256)
* בדיקת ייחודיות שמות משתמשים
* שימוש במפתחות זרים לשמירת שלמות הנתונים
* ניהול סשנים לצורך אימות

### **פעולות עם מסד הנתונים**

* יצירת משתמשים
* חיפוש משתמשים לפי שם
* בדיקת קיום משתמשים
* קבלת רשימת כל המשתמשים הרשומים
* ניהול סטרימים פעילים

מסד הנתונים מאותחל בעת הפעלת השרת באמצעות הפונקציה init\_database() ומשמש דרך המחלקה SessionManager לניהול משתמשים וסשנים.

### **3. מבני נתונים נוספים**

#### **3.1 רישום לוגים**

למעקב אחר אירועים וטעויות במערכת ייעשה שימוש ברישום קבצים. כל אירוע יירשם בקבצי טקסט עם חותמת זמן.

### **מבנה הרישום**

1. פורמט רישום הלוגים  
   מבנה:  
   %(asctime)s - %(name)s - %(levelname)s - %(funcName)s:%(lineno)d - %(message)s

רכיבים:

* %(asctime)s - חותמת זמן (תאריך ושעה)
* %(name)s - שם הלוגר ('HTTPServer')
* %(levelname)s - רמת חשיבות (DEBUG, INFO, WARNING, ERROR)
* %(funcName)s - שם הפונקציה בה נרשם הלוג
* %(lineno)d - מספר שורת הקוד
* %(message)s - טקסט ההודעה

### **3. דוגמת רישום לוג**

2024-11-01 15:23:45,123 - HTTPServer - INFO - handle\_login:542 - User john\_doe successfully logged in

### **קונפיגורציית רישום**

**מטפל קבצים (File Handler):**

* קובץ: logs/server\_YYYYMMDD.log (רוטציה יומית)
* רמה: DEBUG (כל ההודעות)
* קידוד: ברירת המחדל של המערכת

**מטפל קונסולה (Console Handler):**

* פלט: זרם סטנדרטי (טרמינל)
* רמה: INFO (ללא הודעות דיבאג)
* מטרה: ניטור בזמן אמת

### **סיכום**

מבני הנתונים הללו מהווים בסיס לניהול משתמשים, סטרימים ולוגים במערכת הסטרימינג. המערכת משתמשת במסדי נתונים רלציוניים לאחסון המידע, לניטור תהליכים במערכת ולהעלאת הביצועים.

### **סקירה של נקודות תורפה ואיומים במערכת:**

### **עבודה עם מסד הנתונים:**

**הזרקת SQL (SQL Injection):**זהו אחד האיומים הנפוצים ביותר באפליקציות ווב המשתמשות במסדי נתונים. תוקפים יכולים להחדיר קוד SQL זדוני לשאילתות, מה שמאפשר להם לגשת למידע, לשנות אותו או אפילו להריץ פקודות שרירותיות במסד הנתונים.  
**פתרון:** שימוש בשאילתות עם פרמטרים (prepared statements). זה מונע הזרקות מכיוון שהנתונים מועברים כערכים נפרדים ואינם מתפרשים כחלק מהקוד של השאילתה.

### **עבודה עם אתרי אינטרנט:**

**XSS (Cross-Site Scripting):**איום זה כולל החדרה של קוד JavaScript זדוני לדפי האתר, אשר מתבצע בצד הלקוח. תוקפים יכולים להשתמש ב-XSS כדי לגנוב נתוני סשן או לבצע מתקפות פישינג.  
**פתרון:** ביצוע ולידציה וסינון (escaping) של כל הנתונים שהוזנו על ידי המשתמש.

### **תהליך התחברות, אימות ובדיקה:**

**יירוט נתוני סשן:**אם נתוני האימות (למשל, טוקנים או קוקיז) יורטו על ידי תוקף, הוא יוכל לגשת למערכת ללא הרשאה.  
**פתרון:** שימוש ב-HTTPS להעברת מידע בערוץ מאובטח.

**MITM (Man-In-The-Middle):**יירוט ושינוי של מידע שמועבר בין לקוח לשרת.  
**פתרון:** שימוש ב-HTTPS עם SSL/TLS להצפנת המידע המועבר. לחיזוק האבטחה ניתן להשתמש בתעודות עם אימות מורחב (EV Certificates).

**DOS/DDOS:**מתקפות מניעת שירות (Denial of Service) או מתקפות מבוזרות (Distributed Denial of Service) יכולות להעמיס על השרתים ולגרום למערכת לפעול באיטיות או לקרוס.

### **פרוטוקול:**

**TCP, לחיצת יד תלת-שלבית (Three-Way Handshake):**פרוטוקול TCP, אשר משמש להעברת מידע ברשת, עשוי להיות פגיע למתקפות כמו TCP Sniffing (האזנה לתעבורה) או חטיפת חיבורים.  
**פתרון:** שימוש בפרוטוקולים מאובטחים כמו HTTPS (SSL/TLS), המבטיחים הצפנה של הנתונים ברמת התעבורה. יש לשלב גם מנגנוני אימות ברמת האפליקציה והגנה על שלמות הנתונים.

### **הצפנה:**

הצפנת המידע המועבר (למשל, באמצעות SSL/TLS) חיונית להגנה מפני יירוט המידע.  
**פתרון:** כל המידע המועבר בין לקוח לשרת צריך להיות מוצפן באמצעות TLS כדי למנוע גישה של צדדים שלישיים למידע רגיש.

### **נקודות תורפה ואמצעי מניעה:**

* **הזרקת קוד (SQL, XSS):** כפי שנאמר, יש להשתמש ב-prepared statements למניעת SQL injection וב-escaping למניעת XSS.
* **יירוט מידע (MITM):** שימוש ב-HTTPS עם הצפנת SSL/TLS והגדרות תעודה תקינות.
* **הגבלת גישה לדפים:** רק משתמשים מורשים יכולים לגשת לעמודים ספציפיים לפי הרשאות בשרת.
* **כניסה מחדש לחשבון לאחר הפעלה מחדש של השרת.**

### **סיכום:**

כדי להבטיח אבטחה גבוהה במערכת, יש ליישם מערך שלם של אמצעי הגנה הכוללים:

* הצפנה
* הגבלת הרשאות גישה
* סינון תעבורה
* עדכוני אבטחה שוטפים

מימוש הפרויקט

סקירת כל המודולים מחלקות

המרכיבים את המערכת וקשרי הגומלין ביניהם

# **מודולים מיובאים**

# **server.py**

| **שם המודול המיובא** | **מטרת המודול** |
| --- | --- |
| **socket** | **יצירת חיבורי רשת והעברת נתונים באמצעות TCP/UDP** |
| **os** | **עבודה עם מערכת ההפעלה ומערכת הקבצים** |
| **sqlite3** | **עבודה עם מסד נתונים SQLite לשמירת משתמשים** |
| **urllib.parse** | **ניתוח ופענוח כתובות URL ופרמטרי בקשות** |
| **hashlib** | **גיבוב סיסמאות לשמירה בטוחה** |
| **re** | **עבודה עם ביטויים רגולריים לפירוק בקשות HTTP** |
| **json** | **סידור וניתוח של נתוני JSON** |
| **time** | **עבודה עם זמן ויצירת חותמות זמן** |
| **struct** | **אריזה ופריסה של נתונים בינאריים** |
| **threading** | **יצירת ריבוי תהליכונים (threads) לעיבוד לקוחות במקביל** |
| **cv2** | **עיבוד וידאו ותמונות (OpenCV)** |
| **numpy** | **עבודה עם מערכים לצורכי עיבוד וידאו** |
| **queue** | **יצירת תורים להעברת נתונים בין תהליכונים** |
| **websockets** | **תמיכה בחיבורי WebSocket** |
| **asyncio** | **תכנות אסינכרוני** |
| **base64** | **קידוד ופענוח של נתונים בפורמט Base64** |
| **datetime** | **עבודה עם תאריך ושעה** |
| **protocol (StreamProtocol, AuthProtocol)** | **פרוטוקולים מותאמים אישית לזרימה ואימות** |
| **ssl** | **תמיכה בהצפנת SSL/TLS** |

# 

### **simple\_streamer.py**

| **שם המודול המיובא** | **מטרת המודול** |
| --- | --- |
| **socket** | **יצירת חיבורי רשת לשליחת נתונים לשרת** |
| **cv2** | **לכידת וידאו מהמצלמה ועיבוד תמונה** |
| **sounddevice** | **לכידת אודיו מהמיקרופון** |
| **numpy** | **עיבוד נתוני אודיו ווידאו כמערכים** |
| **threading** | **ריבוי תהליכונים לשליחה מקבילה של וידאו ואודיו** |
| **time** | **שליטה על זמן ו-FPS של הזרמת הווידאו** |
| **tkinter** | **יצירת ממשק משתמש גרפי** |
| **tkinter.ttk** | **ווידג'טים משופרים ל-GUI** |
| **tkinter.messagebox** | **תיבות הודעה להודעות למשתמש** |
| **PIL.Image** | **עיבוד תמונות להצגה בממשק** |
| **PIL.ImageTk** | **המרת תמונות לשימוש עם Tkinter** |
| **protocol (StreamProtocol, AuthProtocol)** | **פרוטוקולים מותאמים אישית לזרימה ואימות** |
| **ssl** | **הצפנת SSL/TLS לאימות בטוח** |

# 

## **ניתוח מחלקות של מערכת סטרימינג וידאו**

### **1. StreamProtocol**

* **תפקיד המחלקה**: מספקת פרוטוקול להעברת נתוני וידאו ואודיו דרך UDP תוך כדי פיצול חבילות גדולות.
* **מאפייני המחלקה**:  
  **מערכת פיצול נתונים (Data Fragmentation System)**תפקיד ושימוש: מפצלת נתונים גדולים (וידאו/אודיו) לקטעים בגודל של עד 1024 בייטים לצורך העברה ב-UDP. כל קטע מכיל כותרת עם מטא-נתונים (סוג הנתונים, מספר החבילה, מספר כולל של קטעים).
* **קידוד/פענוח מסגרות (Frame Encoding/Decoding)**תפקיד ושימוש: מקודדת פריימים של וידאו לפורמט JPEG עם איכות ניתנת להתאמה, ומפענחת את הנתונים חזרה לפורמט OpenCV לצורך תצוגה.
* **ניהול מאגר הרכבה (Reassembly Buffer Management)**תפקיד ושימוש: מנהלת מאגרי הרכבה של חבילות מפוצלות. מפרידה בין מאגרי וידאו ואודיו, ומרכיבה אוטומטית חבילות שלמות לאחר קבלת כל הקטעים.
* **רצף חבילות (Packet Sequencing)**תפקיד ושימוש: מבטיחה סידור נכון של החבילות באמצעות מספרים עוקבים, מונעת איבוד או שכפול של נתונים.

### **2. AuthProtocol**

* **תפקיד המחלקה**: מממשת פרוטוקול אימות לקוחות דרך חיבור TCP מאובטח עם תמיכה ב-SSL/TLS.
* **מאפייני המחלקה**:  
  **קידומת אורך הודעה (Message Length Prefixing)**תפקיד ושימוש: מוסיפה קידומת של 4 בתים לאורך כל הודעה כדי להבטיח העברת נתונים נכונה באורך משתנה דרך TCP.
* **שילוב SSL/TLS (SSL/TLS Integration)**תפקיד ושימוש: תומכת בחיבור מוצפן להעברת פרטי התחברות בצורה מאובטחת בין הלקוח לשרת.
* **תהליך אימות דו-כיווני (Bidirectional Authentication Flow)**תפקיד ושימוש: מאפשר אימות דו-כיווני עם אפשרות לרישום משתמשים חדשים ואימות חשבונות קיימים.
* **ניהול שקעי רשת (Socket Management)**תפקיד ושימוש: יוצר ומגדיר שקעי TCP/UDP עם הגדרות מתאימות (כגון SO\_REUSEADDR, timeouts, קישור לכתובות).

### **3. StreamManager**

* **תפקיד המחלקה**: מנהלת את מצב הסטרימים הפעילים ומתאמת בין סטרימרים לצופים.
* **מאפייני המחלקה**:  
  **מעקב אחרי סטרימים פעילים (Active Stream Tracking)**תפקיד ושימוש: עוקבת אחרי סטרימים פעילים בבסיס הנתונים SQLite ומעדכנת את הסטטוס כאשר סטרימרים מתחברים או מתנתקים.
* **ניהול לקוחות WebSocket (WebSocket Client Management)**תפקיד ושימוש: מנהלת חיבורי WebSocket, מקבצת לקוחות לפי סטרימרים להפצת תוכן בצורה יעילה.
* **מערכת תור שידור (Broadcast Queue System)**תפקיד ושימוש: מממשת תור לשליחת נתוני וידאו/אודיו למספר לקוחות עם בקרת עומס (maxsize=100).
* **ניקוי משאבים (Resource Cleanup)**תפקיד ושימוש: מנקה אוטומטית משאבים עם התנתקות הסטרימר: מאגרי הרכבה, תורים, ושליחת הודעת סיום שידור לצופים.

### **4. SessionManager**

* **תפקיד המחלקה**: מנהלת סשנים של משתמשים ומבצעת אימות באמצעות מערכת מבוססת קובצי Cookie.
* **מאפייני המחלקה**:  
  **יצירת מזהי סשן (Session ID Generation)**תפקיד ושימוש: יוצרת מזהי סשן ייחודיים על בסיס גיבוב MD5 של שם המשתמש וזמן הגישה, לצורך אבטחת הסשן.
* **אימות מבוסס Cookie (Cookie-based Authentication)**תפקיד ושימוש: שולפת ומאמתת session\_id מכותרות Cookie ב-HTTP כדי לשמר את מצב ההתחברות.
* **גיבוב סיסמאות (Password Hashing)**תפקיד ושימוש: משתמשת בגיבוב SHA-256 כדי לשמור סיסמאות בצורה מאובטחת בבסיס הנתונים ולמנוע פריצה לחשבונות.
* **מערכת רישום משתמשים (User Registration System)**תפקיד ושימוש: מממשת רישום משתמשים חדשים עם בדיקת ייחוד של שם משתמש וגיבוב סיסמה באופן אוטומטי.

### **5. StreamerApp**

* **תפקיד המחלקה**: אפליקציית GUI שולחנית עבור סטרימרים, מאפשרת לכידת והעברת וידאו/אודיו.
* **מאפייני המחלקה**:  
  **ממשק דו-שלבי (Dual-Interface GUI)**תפקיד ושימוש: מספקת שני ממשקים — טופס התחברות וממשק סטרימינג, מחליפה ביניהם לפי מצב האימות.
* **לכידת וידאו בזמן אמת (Real-time Video Capture)**תפקיד ושימוש: לוכדת וידאו מהמצלמה בעזרת OpenCV (640x480 @ 30fps), מציגה תצוגה מקומית ומשדרת לשרת.
* **שידור אודיו רב-תהליכי (Multi-threaded Audio Streaming)**תפקיד ושימוש: משתמשת ב-sounddevice ללכידת אודיו (44.1kHz, מונו, 16-bit) בתהליך נפרד כדי למנוע חסימת GUI או וידאו.
* **אימות אסינכרוני (Asynchronous Authentication)**תפקיד ושימוש: מבצעת אימות ברקע כדי לשמור על תגובתיות הממשק, עם עדכון סטטוס בזמן אמת.
* **תמיכה ב-SSL/TLS (SSL/TLS Client Support)**תפקיד ושימוש: תומכת בחיבורים מוצפנים מול השרת, כולל fallback במקרה של כשל SSL.

### **6. WebSocket Handler (ב-HTML/JavaScript)**

* **תפקיד המחלקה**: מודול לקוח לקבלת והצגת סטרים של וידאו/אודיו בדפדפן.
* **מאפייני המחלקה**:  
  **שילוב עם Web Audio API**תפקיד ושימוש: משתמשת ב-Web Audio API להשמעת נתוני PCM, כולל שליטה על עוצמת קול ואפשרות השתקה.
* **עיבוד נתוני Base64**תפקיד ושימוש: מפענחת נתוני וידאו (JPEG) ואודיו (PCM) מקודדים ב-base64 שמתקבלים דרך WebSocket, לצורך תצוגה וניגון.
* **ניהול מצב חיבור (Connection State Management)**תפקיד ושימוש: עוקבת אחר מצב חיבור ה-WebSocket, מתחברת מחדש אוטומטית בעת ניתוק, ומנטרת את פעילות הסטרים.
* **התאמת איכות סטרים (Adaptive Stream Quality)**תפקיד ושימוש: מעדכנת את סטטוס הסטרים באופן דינמי, ומטפלת בסוגים שונים של הודעות (וידאו, אודיו, מידע, שגיאה) מהשרת.

### 

### **תיעוד פונקציות של מחלקות**

### **מחלקה: StreamProtocol (בקובץ protocol.py)**

**fragment\_data**פרמטרים נכנסים:

* data\_type (str) – סוג הנתונים ('V' לוידאו, 'A' לאודיו)
* data (bytes) – נתונים לפיצול

פלט:

* מפצל נתונים גדולים למקטעים עם כותרות לשידור ב-UDP. מחזיר רשימת מקטעים.

**send\_fragments**פרמטרים נכנסים:

* socket (socket) – סוקט UDP
* fragments (list) – רשימת מקטעים
* address (tuple) – כתובת יעד (IP, port)

פלט:

* שולח את המקטעים לכתובת. מחזיר את מספר המקטעים שנשלחו.

**process\_fragment**פרמטרים נכנסים:

* data (bytes) – נתונים שהתקבלו עם כותרת

פלט:

* מעבד מקטע שהתקבל. מחזיר טפל (סוג נתונים, מספר חבילה, כמות מקטעים, מספר מקטע, תוכן) או Noneבמקרה של שגיאה.

**reassemble\_fragment**פרמטרים נכנסים:

* data\_type (str) – סוג הנתונים ('V' או 'A')
* packet\_seq (int) – מספר החבילה
* total\_fragments (int) – מספר כולל של מקטעים
* fragment\_num (int) – מספר המקטע
* payload (bytes) – תוכן המקטע

פלט:

* מוסיף מקטע לזיכרון ביניים ובודק אם ניתן להרכיב את החבילה. מחזיר את הנתונים המורכבים אם כל המקטעים התקבלו, אחרת None.

**encode\_video\_frame**פרמטרים נכנסים:

* frame (numpy array) – פריים של תמונה
* quality (int, ברירת מחדל 80) – איכות JPEG (0–100)

פלט:

* מקודד פריים לווידאו לפורמט JPEG דחוס. מחזיר את הנתונים המקודדים.

**decode\_video\_frame**פרמטרים נכנסים:

* data (bytes) – נתונים מקודדים של פריים

פלט:

* מפענח פריים וידאו מהנתונים. מחזיר את הפריים או None במקרה של שגיאה.

**clear\_buffers**פרמטרים נכנסים:

* אין

פלט:

* מנקה את הזיכרונות הביניים של וידאו ואודיו. לא מחזיר כלום.

### **מחלקה: AuthProtocol (בקובץ protocol.py)**

**send\_message** *(static method)*פרמטרים נכנסים:

* socket (socket) – סוקט TCP
* message (str או bytes) – הודעה לשליחה

פלט:

* שולח הודעה עם קידומת אורך דרך TCP. לא מחזיר כלום.

**receive\_message** *(static method)*פרמטרים נכנסים:

* socket (socket) – סוקט TCP

פלט:

* מקבל הודעה עם קידומת אורך. מחזיר את הנתונים או None במקרה של שגיאה.

**server\_authenticate**פרמטרים נכנסים:

* client\_socket (socket) – סוקט TCP של הלקוח
* auth\_function (function) – פונקציה לבדוק את פרטי ההתחברות
* register\_function (function, אופציונלי) – פונקציה לרישום משתמש

פלט:

* מבצע אימות TCP בצד השרת. מחזיר את שם המשתמש או None.

**create\_socket** *(static method)*פרמטרים נכנסים:

* host (str, ברירת מחדל '0.0.0.0')
* port (int, ברירת מחדל 0)
* tcp (bool, ברירת מחדל False) – אם True, יוצר סוקט TCP, אחרת UDP

פלט:

* יוצר סוקט וקושר אותו לכתובת. מחזיר את הסוקט.

### **מחלקה: StreamManager (בקובץ server-classes.py)**

**\_\_init\_\_**פרמטרים נכנסים:

* db\_name (str, ברירת מחדל 'user\_database.db') – שם קובץ מסד הנתונים

פלט:

* מאתחל את מנהל הזרמים עם זיכרונות ו-Qs. לא מחזיר כלום.

**is\_streaming**פרמטרים נכנסים:

* username (str) – שם משתמש

פלט:

* בודק אם המשתמש כרגע משדר. מחזיר True אם כן, אחרת False.

**add\_user\_to\_active\_streams**פרמטרים נכנסים:

* username (str) – שם משתמש

פלט:

* מוסיף את המשתמש לטבלת שידורים פעילים. לא מחזיר כלום.

**cleanup\_database**פרמטרים נכנסים:

* username (str) – שם משתמש

פלט:

* מסיר את המשתמש מטבלת שידורים פעילים. לא מחזיר כלום.

**cleanup\_after\_streamer**פרמטרים נכנסים:

* אין

פלט:

* מבצע ניקוי אחרי שהמשדר מתנתק, משחרר זיכרונות ושולח הודעות ללקוחות. לא מחזיר כלום.

### **מחלקה: SessionManager (בקובץ server-classes.py)**

**\_\_init\_\_**פרמטרים נכנסים:

* db\_name (str, ברירת מחדל 'user\_database.db')

פלט:

* מאתחל את מנהל הסשנים. לא מחזיר כלום.

**create\_session**פרמטרים נכנסים:

* username (str)

פלט:

* יוצר סשן חדש לפי MD5 של שם וזמן. מחזיר מזהה ייחודי.

**get\_session\_user**פרמטרים נכנסים:

* request (str) – בקשת HTTP עם עוגיות

פלט:

* מחלץ את שם המשתמש מהעוגיה. מחזיר את השם או None.

**remove\_session**פרמטרים נכנסים:

* session\_id (str)

פלט:

* מסיר סשן מהפעילים. מחזיר True אם הצליח, אחרת False.

**authenticate\_user**פרמטרים נכנסים:

* username (str), password (str)

פלט:

* בודק פרטי התחברות מול בסיס נתונים עם SHA256. מחזיר True אם הצלחה, אחרת False.

**register\_user**פרמטרים נכנסים:

* username (str), password (str)

פלט:

* רושם משתמש חדש, מאמת ייחודיות, שומר סיסמה עם SHA256. מחזיר True אם הצלחה, אחרת False.

### **מחלקה: StreamerApp (בקובץ simple\_streamer.py)**

**\_\_init\_\_**פרמטרים נכנסים:

* root (tkinter.Tk)

פלט:

* מאתחל את אפליקציית המשדר עם ממשק משתמש ומצב פנימי. לא מחזיר כלום.

**create\_login\_frame**פרמטרים נכנסים:

* אין

פלט:

* יוצר ממשק כניסה. לא מחזיר כלום.

**create\_stream\_frame**פרמטרים נכנסים:

* אין

פלט:

* יוצר ממשק שידור. לא מחזיר כלום.

**show\_login\_frame**פרמטרים נכנסים:

* אין

פלט:

* עובר למסך כניסה. לא מחזיר כלום.

**show\_stream\_frame**פרמטרים נכנסים:

* אין

פלט:

* עובר למסך שידור. לא מחזיר כלום.

**authenticate**פרמטרים נכנסים:

* אין

פלט:

* מבצע אימות מול השרת, בודק נתונים, מתחיל תהליך אימות. לא מחזיר כלום.

**authentication\_thread**פרמטרים נכנסים:

* server\_ip (str), auth\_port (int)

פלט:

* מבצע אימות בתהליך נפרד לשמירה על תגובתיות. לא מחזיר כלום.

**gui\_authenticate**פרמטרים נכנסים:

* server\_host (str), server\_port (int)

פלט:

* אימות מותאם GUI עם SSL. מחזיר (תוצאה, שם משתמש, פורט UDP).

**start\_streaming**פרמטרים נכנסים:

* אין

פלט:

* מתחיל שידור וידאו/אודיו, פותח מצלמה ו-UDP. לא מחזיר כלום.

**update\_video**פרמטרים נכנסים:

* אין

פלט:

* מעדכן פריים וידאו, מקודד, מציג ושולח. לא מחזיר כלום.

**send\_audio**פרמטרים נכנסים:

* אין

פלט:

* שולח אודיו מהמיקרופון, מקטעים ל-UDP. לא מחזיר כלום.

**stop\_streaming**פרמטרים נכנסים:

* אין

פלט:

* עוצר שידור, מחזיר למסך כניסה, משחרר מצלמה וסוקט. לא מחזיר כלום.

**on\_closing**פרמטרים נכנסים:

* אין

פלט:

* מטפל בסגירת החלון, עוצר שידור, משחרר משאבים. לא מחזיר כלום.

### **פונקציה אסינכרונית send\_ws\_message (בקובץ server-classes.py)**

**send\_ws\_message**פרמטרים נכנסים:

* client (websocket) – לקוח WebSocket
* message (str) – הודעה לשליחה

פלט:

* שולח הודעה דרך WebSocket עם טיפול בשגיאות. מחזיר True בהצלחה, False בשגיאה.

אלגוריתמיות

### **מחלקת SessionManager**

#### **מבנה ואתחול**

class SessionManager:

def \_\_init\_\_(self, db\_name='user\_database.db'):

self.db\_name = db\_name

self.active\_sessions = {} # מילון לאחסון סשנים פעילים

הסשנים נשמרים בזיכרון במילון active\_sessions, שבו:

* **מפתח** – מזהה ייחודי של הסשן (session ID)
* **ערך** – שם המשתמש

### **תהליך עבודה עם סשנים**

#### **יצירת סשן (create\_session)**

def create\_session(self, username):

session\_id = hashlib.md5(f"{username}:{time.time()}".encode()).hexdigest()

self.active\_sessions[session\_id] = username

return session\_id

**איך זה עובד**:

* נוצר מזהה ייחודי על בסיס שם המשתמש והזמן הנוכחי
* משתמשים ב־MD5 כדי ליצור מחרוזת באורך 32 תווים
* הסשן נשמר במילון active\_sessions
* session\_id מוחזר ונשלח ללקוח

#### **הגדרת Cookie בדפדפן**

לאחר התחברות מוצלחת בפונקציה handle\_login:

response += f"Set-Cookie: session\_id={session\_id}; Path=/\r\n".encode()

**מאפייני ה־Cookie**:

* שם: session\_id
* ערך: המזהה שנוצר
* Path: ‎/ (נגיש לכל האתר)

#### **שליפת שם המשתמש מהסשן (get\_session\_user)**

def get\_session\_user(self, request):

cookie\_match = re.search(r'Cookie:([^\r\n]+)', request, re.IGNORECASE)

if not cookie\_match:

return None

cookie\_header = cookie\_match.group(1).strip()

cookies = {}

for item in cookie\_header.split(';'):

if '=' in item:

key, value = item.strip().split('=', 1)

cookies[key] = value

session\_id = cookies.get('session\_id')

if session\_id and session\_id in self.active\_sessions:

return self.active\_sessions[session\_id]

return None

**תהליך האימות**:

* קריאת כותרת ה־Cookie מתוך בקשת HTTP
* פיענוח כל העוגיות
* חיפוש Cookie בשם session\_id
* בדיקת קיום במילון active\_sessions
* מחזיר את שם המשתמש או None

#### **הגנה על דפים**

protected\_pages = [

'search-stream.html',

'profile.html',

'start-stream.html',

'profile-true.html'

]

אם הקובץ נמצא ברשימת הדפים המוגנים:

if filename in protected\_pages:

username = session\_manager.get\_session\_user(request)

if not username:

response = b"HTTP/1.1 302 Found\r\n"

response += b"Location: /login.html\r\n"

#### **מחיקת סשן (remove\_session)**

def remove\_session(self, session\_id):

if session\_id in self.active\_sessions:

del self.active\_sessions[session\_id]

return True

return False

בעת יציאה (handle\_logout):

def handle\_logout(client\_socket, request):

cookie\_match = re.search(r'Cookie:.\*session\_id=([^;]+)', request)

if cookie\_match:

session\_id = cookie\_match.group(1)

session\_manager.remove\_session(session\_id)

response += b"Set-Cookie: session\_id=; Path=/; Expires=Thu, 01 Jan 1970 00:00:00 GMT\r\n"

### **מאפייני מימוש**

**יתרונות**:

* מהיר – כל הסשנים בזיכרון
* מאובטח – מזהים נוצרים עם חותמת זמן

**חסרונות**:

* זמני – כל הסשנים נעלמים לאחר אתחול השרת
* לא ניתן להרחבה – לא תומך בריבוי שרתים

### **זרימת תהליך ההתחברות**

* **התחברות**: המשתמש מכניס שם משתמש/סיסמה → בדיקה מול מסד הנתונים → יצירת סשן → הגדרת Cookie
* **בקשות**: הדפדפן שולח Cookie → השרת בודק את הסשן → מאשר או חוסם גישה
* **יציאה**: מחיקת הסשן מהזיכרון → מחיקת Cookie מהדפדפן

### **אלגוריתם קבלת סטרים מהמשדר לשרת**

#### **ארכיטקטורת פרוטוקול כללית**

הפרוטוקול פותח בצורה מותאמת אישית בדומה ל־RTP (Real-Time Transport Protocol), ומחולק לשני חלקים:

1. פרוטוקול אימות ב־TCP
2. פרוטוקול שידור מדיה ב־UDP

#### **שלב האימות (TCP)**

השרת מקבל חיבור TCP מהמשדר בפורט 5050. הוא תומך בהצפנת SSL להגנה על פרטי התחברות:

def auth\_server\_thread():

tcp\_auth\_socket = AuthProtocol.create\_socket(HOST, TCP\_AUTH\_PORT, tcp=True)

tcp\_auth\_socket.listen(5)

if TCP\_AUTH\_SSL:

ssl\_context = create\_ssl\_context()

if ssl\_context:

client\_socket = ssl\_context.wrap\_socket(client\_socket, server\_side=True)

האימות מתבצע לפי פורמט "username,password". הנתונים נבדקים במסד נתוני SQLite. בהצלחה, השרת מחזיר פורט UDP ומשייך את המשתמש לטבלת סטרימים פעילים.

#### **אתחול חיבור UDP**

לאחר אימות, נפתח שקע UDP על פורט 5001:

def handle\_streamer\_udp\_connection(username):

stream\_protocol.clear\_buffers()

udp\_socket = AuthProtocol.create\_socket(HOST, PORT\_UDP, tcp=False)

receive\_stream\_data(udp\_socket, username)

#### **פרוטוקול קיטוע (Fragmentation)**

הנתונים מקוטעים לפקטות בגודל עד 1024 בתים. כותרת של 20 בתים (5 שלמים):

* סוג: 'V' לוידאו או 'A' לאודיו
* מזהה פקטה
* סך חלקים
* מספר חלק נוכחי
* שדה שמור

fragment\_header = struct.pack(

"!5I",

ord(data\_type[0]),

self.packet\_sequence,

total\_fragments,

i,

0

)

#### **עיבוד קבלת פקטות**

def process\_fragment(self, data):

if len(data) < FRAGMENT\_HEADER\_SIZE:

return None

data\_type\_byte, packet\_seq, total\_fragments, fragment\_num, \_ = struct.unpack(

"!5I", data[:FRAGMENT\_HEADER\_SIZE]

)

data\_type = chr(data\_type\_byte)

payload = data[FRAGMENT\_HEADER\_SIZE:]

#### **הרכבת פקטות**

def reassemble\_fragment(self, data\_type, packet\_seq, total\_fragments, fragment\_num, payload):

reassembly\_buffer = self.video\_reassembly\_buffer if data\_type == DATA\_TYPE\_VIDEO else self.audio\_reassembly\_buffer

if packet\_seq not in reassembly\_buffer:

reassembly\_buffer[packet\_seq] = [None] \* total\_fragments

reassembly\_buffer[packet\_seq][fragment\_num] = payload

if None not in reassembly\_buffer[packet\_seq]:

full\_data = b''.join(reassembly\_buffer[packet\_seq])

del reassembly\_buffer[packet\_seq]

return full\_data

#### **עיבוד מדיה**

**וידאו**:

if data\_type == "V":

new\_frame = stream\_protocol.decode\_video\_frame(full\_data)

if new\_frame is not None:

with frame\_lock:

frame = new\_frame

broadcast\_queue.put(("V", full\_data, packet\_seq))

**דקוד עם OpenCV**:

def decode\_video\_frame(self, data):

np\_data = np.frombuffer(data, dtype=np.uint8)

decoded\_frame = cv2.imdecode(np\_data, cv2.IMREAD\_COLOR)

return decoded\_frame

**אודיו** – פורמט PCM 16 ביט, 44100 הרץ, מוכנס ישירות לתור השידור.

#### **גילוי ניתוק**

udp\_socket.settimeout(1.0)

disconnect\_counter = 0

last\_packet\_time = time.time()

try:

data, addr = udp\_socket.recvfrom(65535)

disconnect\_counter = 0

last\_packet\_time = time.time()

except socket.timeout:

disconnect\_counter += 1

current\_time = time.time()

inactive\_time = current\_time - last\_packet\_time

if disconnect\_counter >= 2 or inactive\_time > 5:

stream\_manager.cleanup\_database(username)

streamer\_disconnected.set()

#### **ניהול מצב סטרימר**

def add\_user\_to\_active\_streams(self, username):

cursor.execute("INSERT INTO active\_streams (username, start\_time) VALUES (?, ?)",

(username, datetime.datetime.now()))

בעת ניתוק:

* מנקה באפרים
* מוחק מה־DB
* מרוקן תור שידור
* מאפס פריים נוכחי
* שולח התראות ל־WebSocket

#### **הפצת נתונים**

המדיה נכנסת ל־broadcast\_queue (מקסימום 100). שאר רכיבי המערכת מפיצים לצופים ב־WebSocket.

הפרוטוקול מספק שידור מדויק בזמן אמת, עם טיפול אוטומטי בשגיאות רשת – קריטי לאיכות שידור.

asyncio

**asyncio** היא ספריית Python למשימות אסינכרוניות. היא מאפשרת לטפל במספר רב של חיבורי WebSocket בו-זמנית ללא חסימה.

**שימושים בקוד שלי:**

* **שרת WebSocket**: מטפל בהרבה לקוחות בו-זמנית
* **שליחת הודעות**: שולח נתונים ללקוחות מבלי לחכות
* **קבלת הודעות**: מקבל נתונים מלקוחות באופן אסינכרוני
* **ניהול חיבורים**: מנהל התחברויות והתנתקויות

**יתרונות:**

* ביצועים גבוהים עם הרבה חיבורים
* קוד פשוט וקריא
* חיסכון במשאבי מערכת

מסמך בדיקות מלא

1. הרשמה/כניסה המשתמש למערכת:

מטרת הבדיקה: משתמש יכול להירשם או להיכנס למערכת עם לוגין וסיסמה וזה ישמור ב-database.

תהליך הבדיקה היה דרך אתר שהוא רק לצופים בסוף. לפתוח את האתר, לבחור האם משתמש יוצר חשבון חדש או נכנס לחשבון הקיים.  
בהרשמה להזין נתונים - לוגין וסיסמה, אם לוגין זה פנוי אז אישור הרשמה ומעבר לאתר עם החשבון הזה.  
בכניסה לחשבון קיים - להזין לוגין וסיסמה, אם לוגין כזה קיים וסיסמה נכונה מעבר לאתר עם החשבון הזה.

בדיקת דרך אתר עברה בהצלחה.

1. חיפוש, בחירה של סטרימר והתחברות אליו:

מטרת הבדיקה: צופה יכול לכתוב שם הסטרימר, להסתכל האם יש לו סטרים עכשיו ואם כן - להתחיל להסתכל את הסטרים.

תהליך הבדיקה: לפתוח את מקטע החיפוש, להזין את שם הסטרימר, לראות רשימה של סטרימרים עם השמות הדומות, לבחור את הסטרימר המתאים לך, לראות אם יש לו שידור חי, להתחבר לשידור חי.

בעיות שפגשתי הן בעיות בהתחברות לסדר חי. צופה היה יכול להתחבר ל-streamer שאין לו זרם פעיל.

הסתדרתי עם הבעיה כשעספתי active streams כחלק ב-database שכולל שם של ה-streamer וזמן התחלה של הזרם.

1. עוקבת אחרי משתמשים שמבצעים שידור חי כרגע. הטבלה כוללת שני שדות:

* שדה username מסוג TEXT שמכיל את שם המשתמש של הסטרימר והוא המפתח הראשי (PRIMARY KEY);
* שדה start\_time מסוג TIMESTAMP שמכיל את זמן תחילת השידור עם ערך ברירת מחדל של CURRENT\_TIMESTAMP.  
  בנוסף, בטבלה יש מפתח חיצוני (FOREIGN KEY) שקושר את ה-username לשדה username בטבלת המשתמשים (users).

1. שידור של הסטרים:

מטרת הבדיקה: סטרימר יכול להפעיל את הסטרים בכל עת, וגם לעצור אותו.

תהליך הבדיקה: לבחור את המיקרופון והמצלמה שישתמשו בהם במהלך השידור, להתחיל את הסטרים, לראות כמה צופים יש בו כרגע, להפסיק את הסטרים בכל עת.

בסוף כתבתי את הסטירמר בנפרד לאתר והיו לי מספר בעיות איתו. חיבור איטי, כי בהתחלה כתבתי הכל ב-tcp. אחרי שעברתי ל-udp היו לי בעיות עם גודל audio, video frames.

כדי לפתור את הבעיות עברתי ל-udp ועספתי fragment\_data שמפרקת נתונים גדולים לחלקים.

fragment\_data(data\_type, data):  
מחלק נתונים גדולים לפרגמנטים בגודל של עד 1004 בתים (1024 פחות 20 בתים לכותרת).  
יוצר כותרת לכל פרגמנט בעזרת struct.pack("!5I", ...).  
מעלה את מספר הסידורי (sequence number) של החבילה הבאה.  
מחזיר רשימת פרגמנטים.

process\_fragment(data):  
בודק שאורך החבילה הוא לפחות 20 בתים.  
מפענח את הכותרת בעזרת struct.unpack("!5I", ...).  
מחלץ את המידע החשוב אחרי הכותרת.  
מחזיר טופל עם המידע: (סוג\_הנתונים, מספר החבילה, סך כל הפרגמנטים, מספר הפרגמנט הנוכחי, הנתונים).

reassemble\_fragment(...):  
משתמש בבופרים נפרדים לוידאו ולאודיו.  
יוצר מערך של None בגודל מספר הפרגמנטים.  
ממלא את המערך ככל שמתקבלים הפרגמנטים.  
כאשר כל הפרגמנטים התקבלו, מחבר אותם להודעה אחת אחידה.  
מוחק את החבילה המורכבת מהבופר.

1. שינוי נתונים:

מטרת הבדיקה: כל משתמש יכול לשנות את הנתונים שלו.

תהליך הבדיקה: להיכנס למסך שינוי נתונים, להזין סיסמה ולוגין חדשים, אם הלוגין פנוי - אישור שינוי הנתונים, אם לא - ביטול שינוי הנתונים.

הבדיקה עברה בהצלחה.

1. כשל בשידור:

מטרת הבדיקה: עצירת סטרים במקרה של בעיות מצד סטרימר או מצד צופה.

תהליך הבדיקה: לבדוק את חיבור הסטרימר, לעצור את הזרם, לעבור למסך המתאים לצופים.

לא היו בדיקות חיבור מצד הסטרימר, אז אפילו אחרי הפסקת הזרם הצופה לא יצא מהמסך של השידור.

הוספתי ניטור פעיל וכן הודעות ping כדי לבדוק את פעילות הזרם.

לוגיקת העבודה של המוניטורינג:

0-5 שניות ללא נתונים: עבודה תקינה, אין פעולה.  
5-10 שניות ללא נתונים: התרעה למשתמש "Stream inactive - waiting for data..."  
יותר מ-10 שניות ללא נתונים: מצב קריטי - מופיעה הודעת alert והמערכת מחזירה אוטומטית למסך החיפוש.

מטרת הודעות ה-ping:

שמירה על פעילות הקשר (מניעת timeout).  
בדיקת מצב החיבור.  
עקיפת הגבלות של שרתי proxy וחומות אש (firewalls).

המתנה לתשובה מהשרת:

למרות שבקוד אין טיפול מפורש בתשובות pong, השרת אמור להגיב להודעות ping.  
היעדר תגובה או שגיאות בשליחת ping יטופלו על ידי מערכת ניטור הפעילות.

בדיקות נוספות

1. הרשמה/כניסה המשתמש streamer למערכת:

* מטרת הבדיקה: משתמש streamer יכול להירשם או להיכנס למערכת עם לוגין וסיסמה וזה ישמור ב-database.
* תהליך בדיקת היה דרך דרך קוד של-streamer שעשיתי בנפרד בסוף.
* בבדיקות של streamer היו כמה דברים - בהתחלה הוא עשה פניות ממש ל-database ואז אני שחכתי על זה. אחרי בדיקות מכמה מחשבים הבנתי מה הבעיה ועשיתי תהליך של הרשמה דרך שרת (login/authorisation request). בסוף גם הייתי צריך ל-udp חלק עם tcp בשביל הרשמה מאובטחת.

SSL handshake עם הלקוח  
פרוטוקול אימות:  
קבלת: "username,password"  
בדיקה בבסיס הנתונים דרך session\_manager.authenticate\_user()  
רישום אופציונלי דרך session\_manager.register\_user()  
תגובות: 'f', 'fr', 't'  
בהצלחה נשלח פורט UDP: str(PORT\_UDP)  
הגדרת current\_streamer = username  
הוספה לטבלת active\_streams

לקוח (simple\_streamer) ----[TCP, SSL]----> שרת (recv-streamer)

* username,password
* בדיקת אישורי התחברות
* יצירת סשן ללקוח

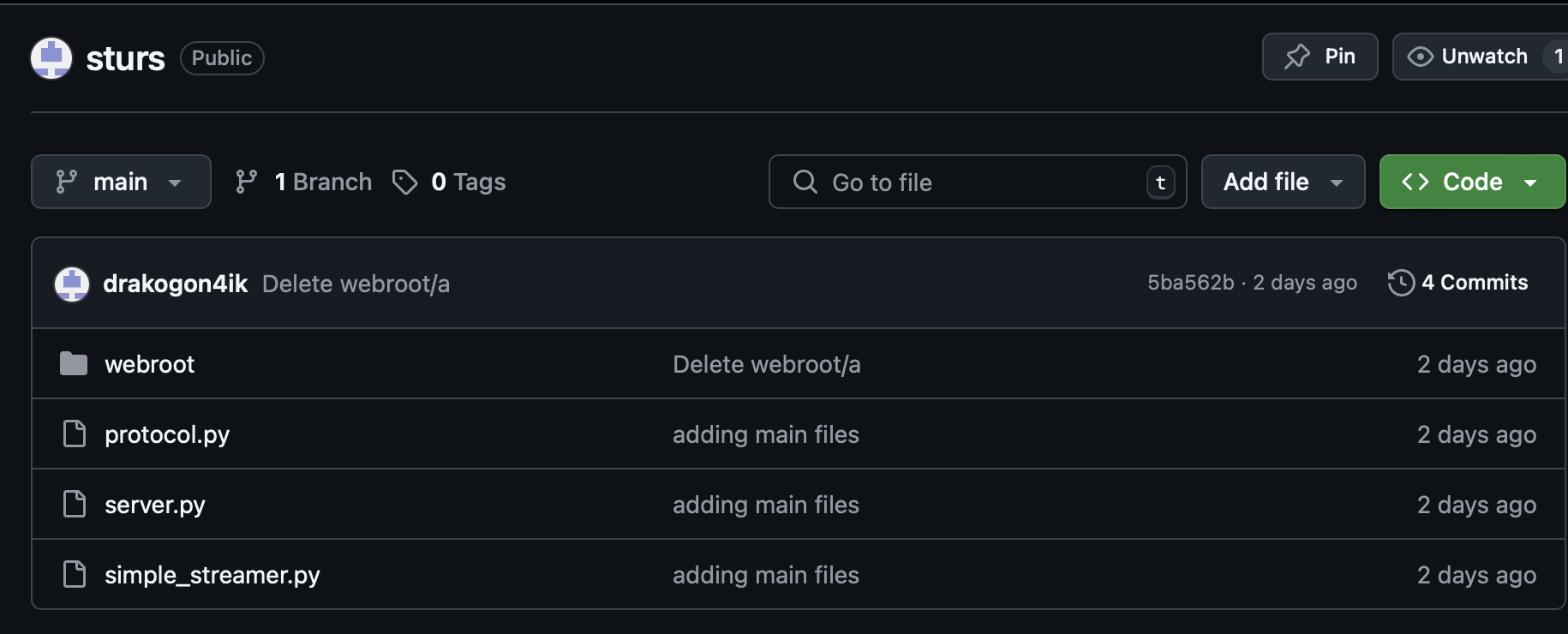
לקוח<----[TCP, SSL]---- שרת

* קוד תוצאה ('t'/'f'/'fr')
* תוצאת האימות
* פורט UDP (בהצלחה)
* הקצאת פורט UDP

מדריך למשתמש

עץ קבצים

הפרויקט שלי כולל שלושה python קבצים שהם: server, simple-streamer, protocol; וגם תיקייה webroot שבה נמצאות כל html קבצים של האתר: login, register, registration-success, profile, search-stream.





הסבר על הקבצים

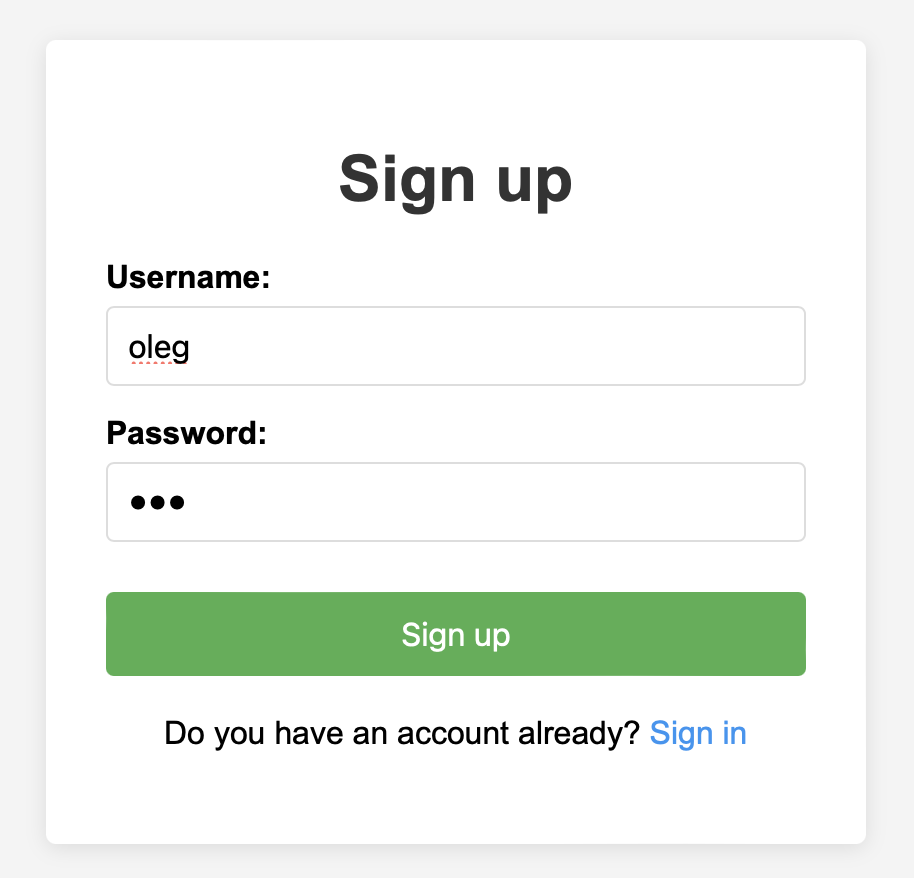
* [server.py](http://server.py) זה השרת שפועל אתר לצופים ושומר על קשר עם סטרימר (מוכן לקבל סטרימר חדש), מפעיל websocket, מבצע העברות נתונים הן ל-websocket והן לצופי udp פשוטים.
* [simple-streamer.py](http://simple-streamer.py) זהו קובץ להרצת הקוד עם ממשק הסטרימינג - המשתמש מציין את ה-IP והיציאה וכן מכניס את השם והסיסמה כדי להיכנס או להירשם. לאחר קבלת יציאת udp, הסטרימר יכול להתחיל להזרים. בכל רגע הסטרימר יכול לסיים אותו וללכת להתחלה.
* [protocol.py](http://protocol.py) הוא פרוטוקול תקשורת בין שרת לסטרימר.
* login.html הוא מסך כניסה באתר לצופים עם שם וסיסמה.
* register.html הוא מסך הרשמה באתר לצופים עם שם וסיסמה.
* registration-success.html הוא מסך הצלחה בהרשמה באתר לצופים.
* profile.html הוא מסך שינוי נתונים באתר לצופים עם סיסמה ישנה וסיסמה חדשה. בו גם אפשר לצאת מן החשבון.
* seacrh-stream.html הוא מסך ראשי של חיפוש סטרימרים וכניסה לזרם פעיל. בו גם אפשר לשנות את עוצמת הקול של הזרם.

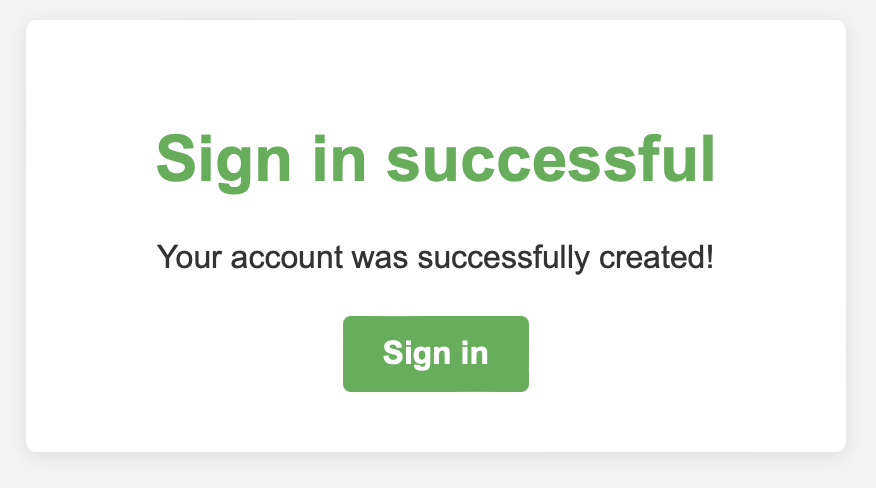
תהליך עבודת האתר

הפעלת פרויקט מתחילה בהפעלת האתר server.py ומעבר לאתר / חיבור הסטרימר. בואו נסתכל תחילה על המעבר לאתר.

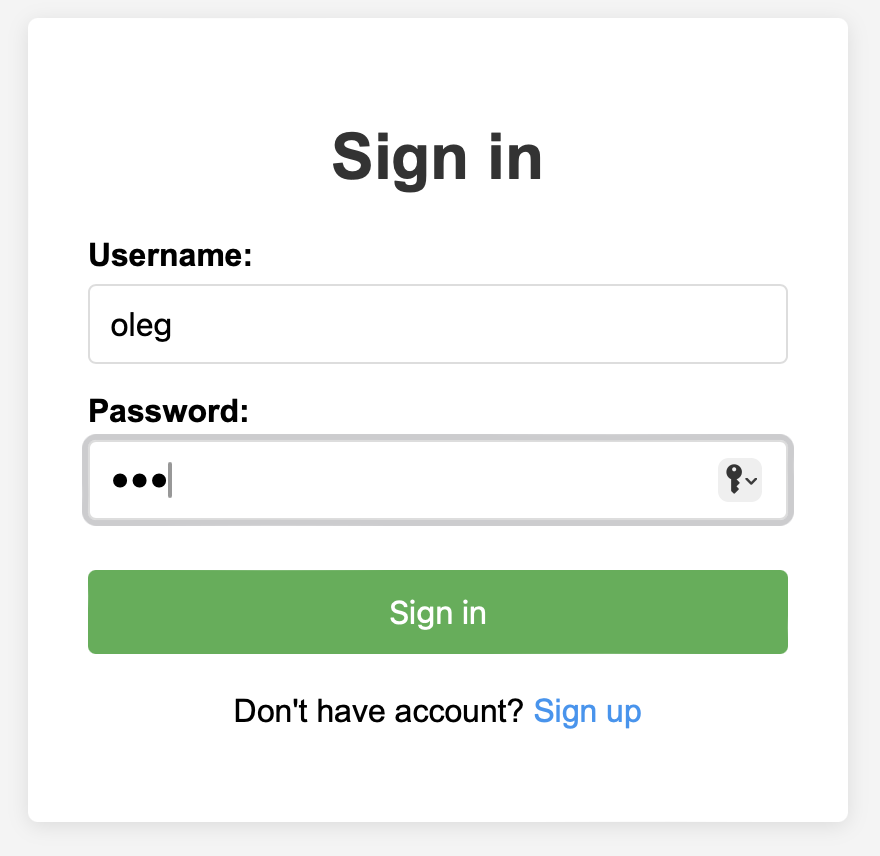
אתר:

הצופה מועבר לעמוד הרשמה בו הוא יכול ליצור את חשבונו.

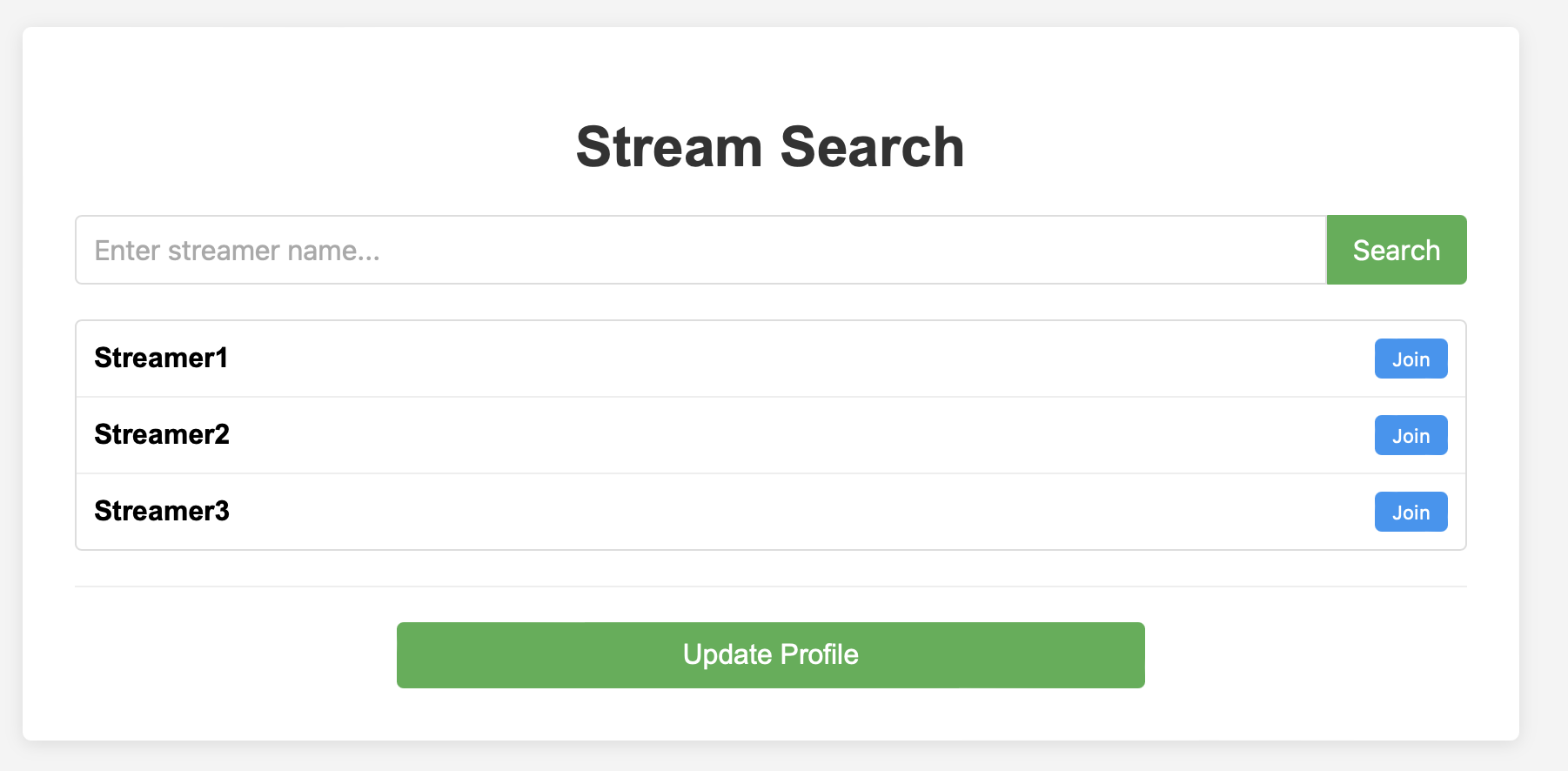




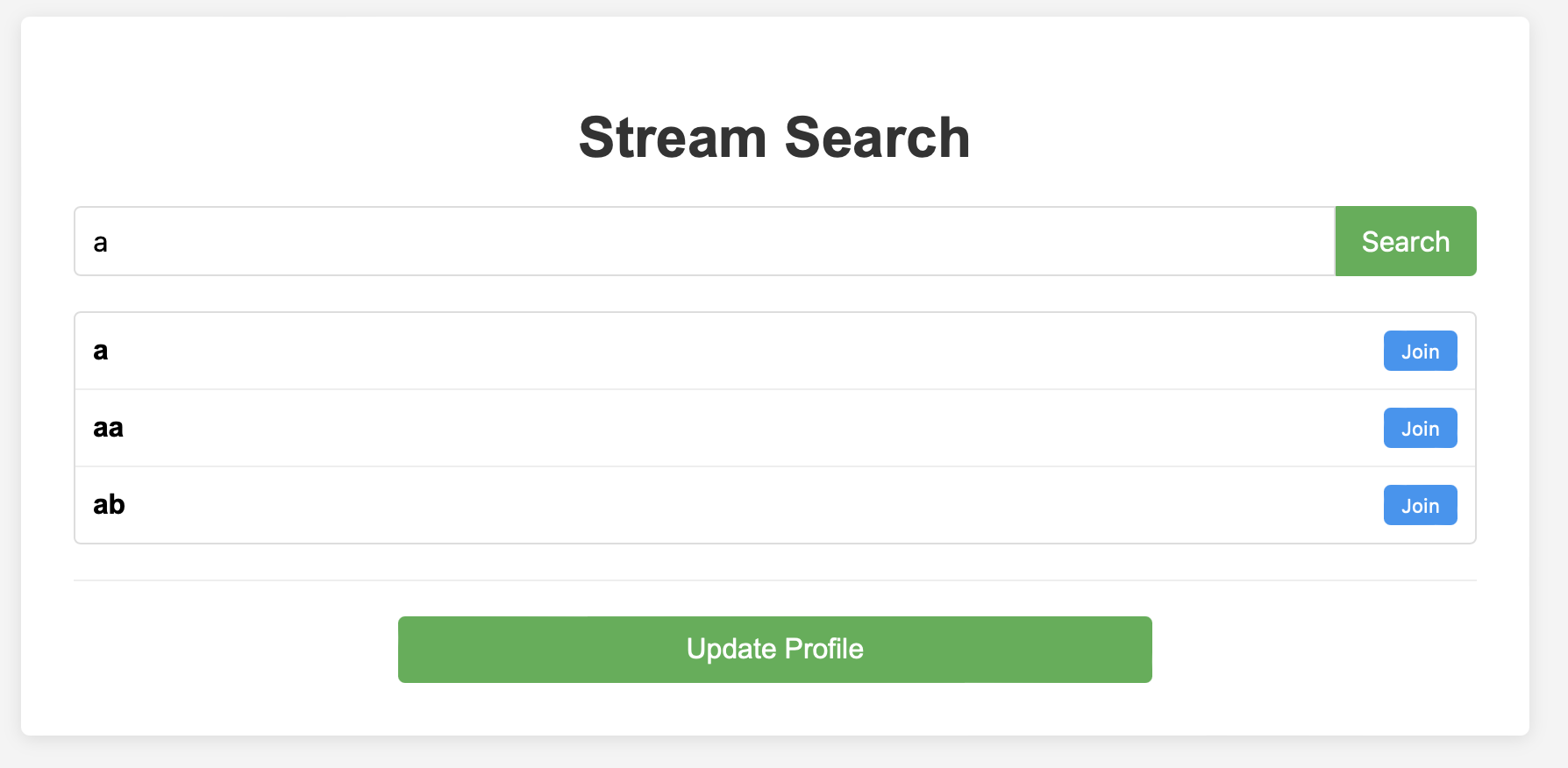
לאחר מכן הצופה מקבל הודעה שהשם נלקח או שהרישום הצליח.



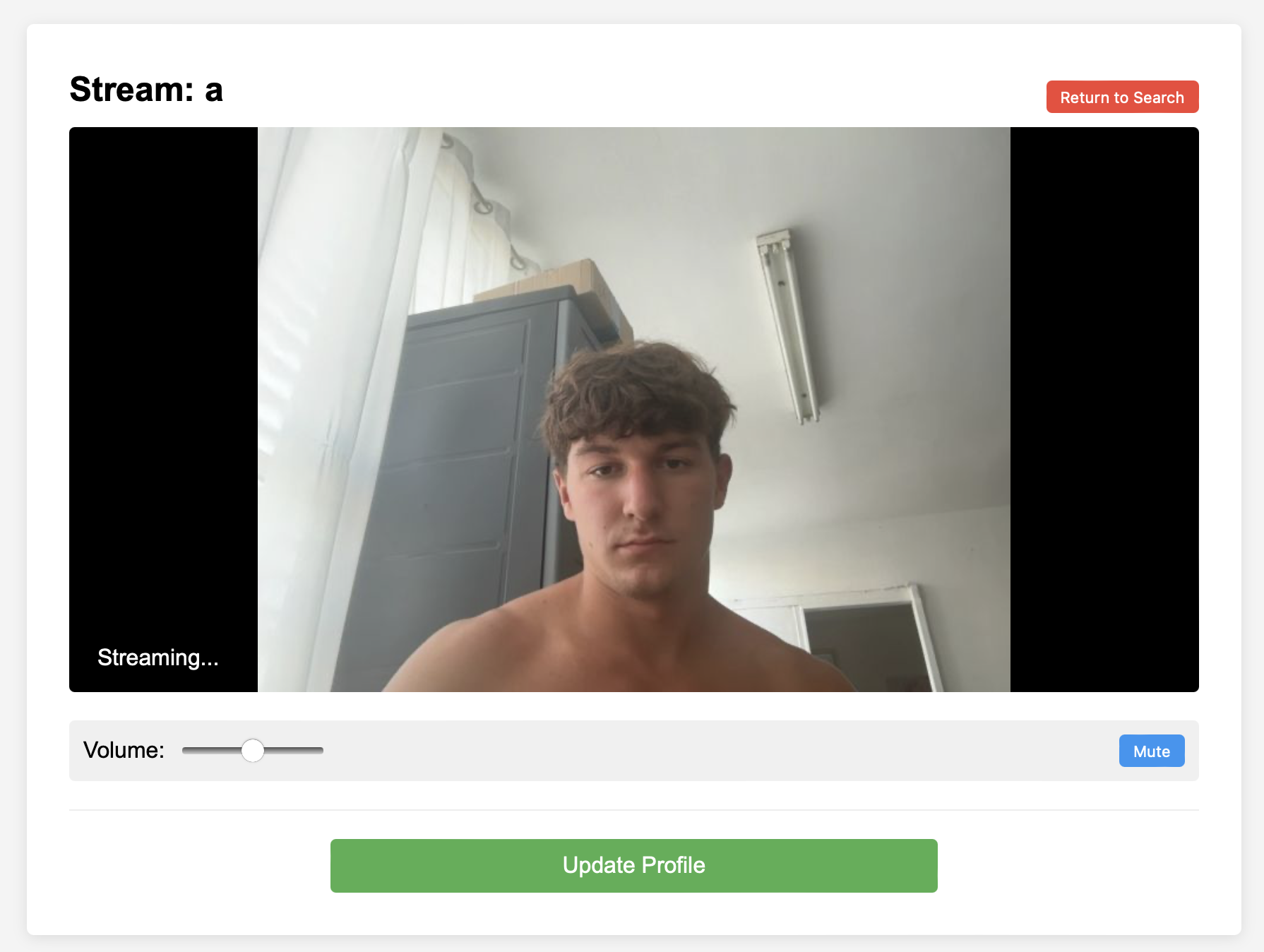
הצופה מועבר לדף הכניסה בו הוא מזין את פרטי החשבון שלו.



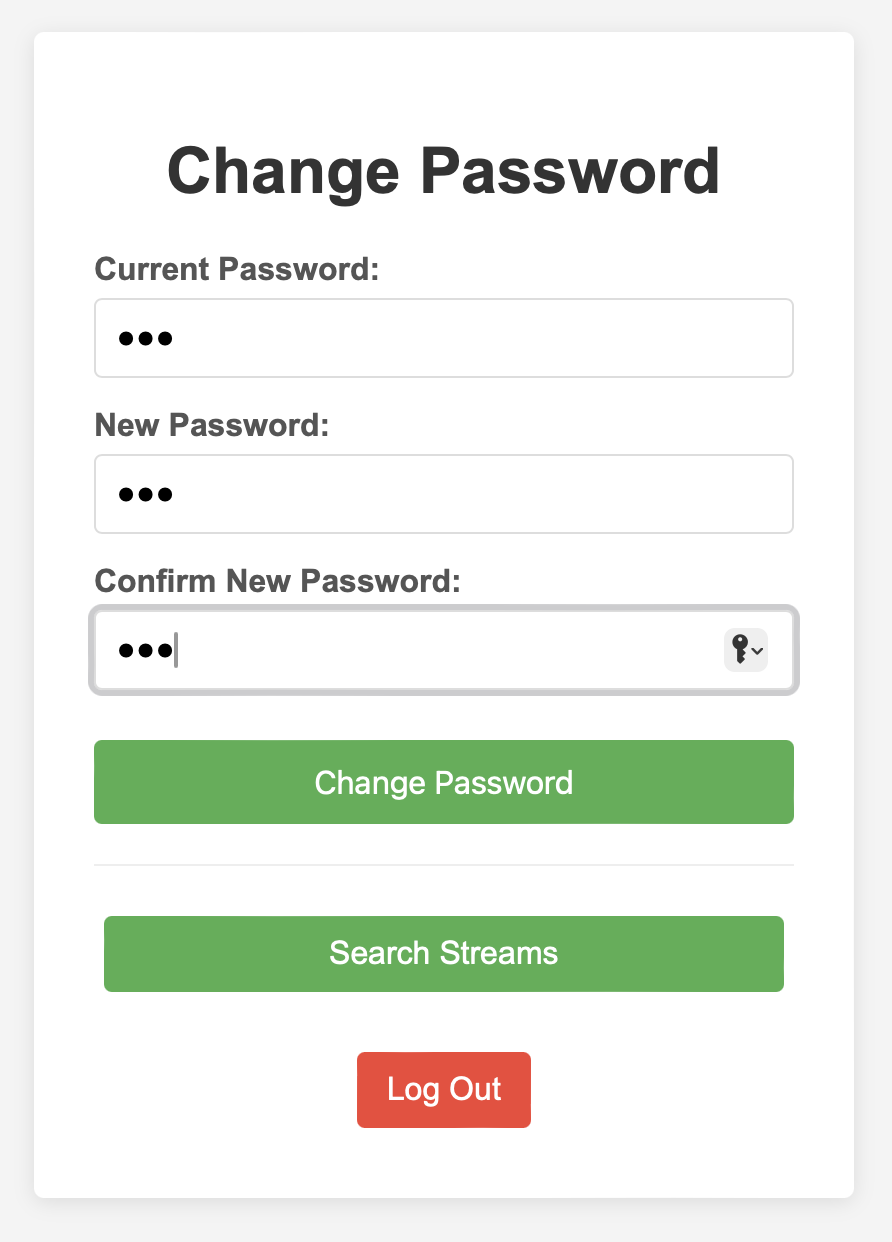
הצופה עובר לדף החיפוש של הסטרימר.



כאן תוכלו להזין את שם הסטרימר הרצוי ולמצוא את כל השמות הדומים לשאילתה שהוזנה.

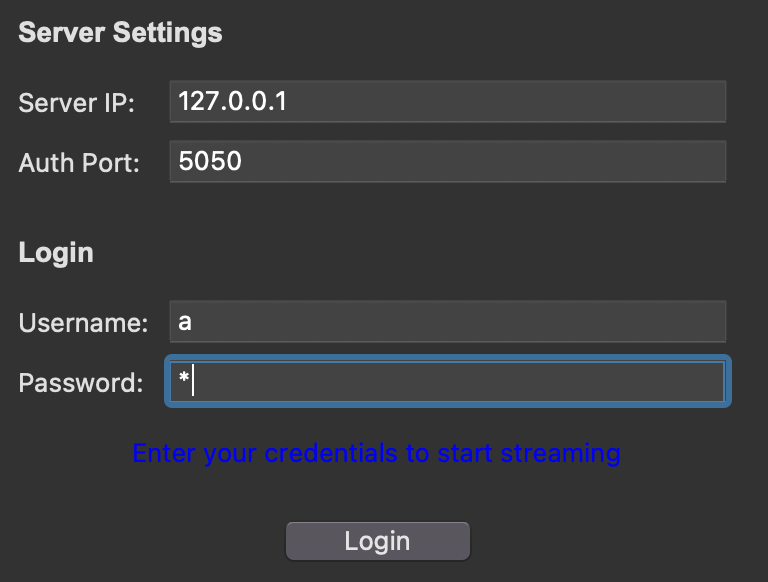


כאשר אתה מתחבר לזרם עובד, מתחיל זרם שבו אתה יכול לשנות את רמת הווליום ולהתנתק בכל עת.

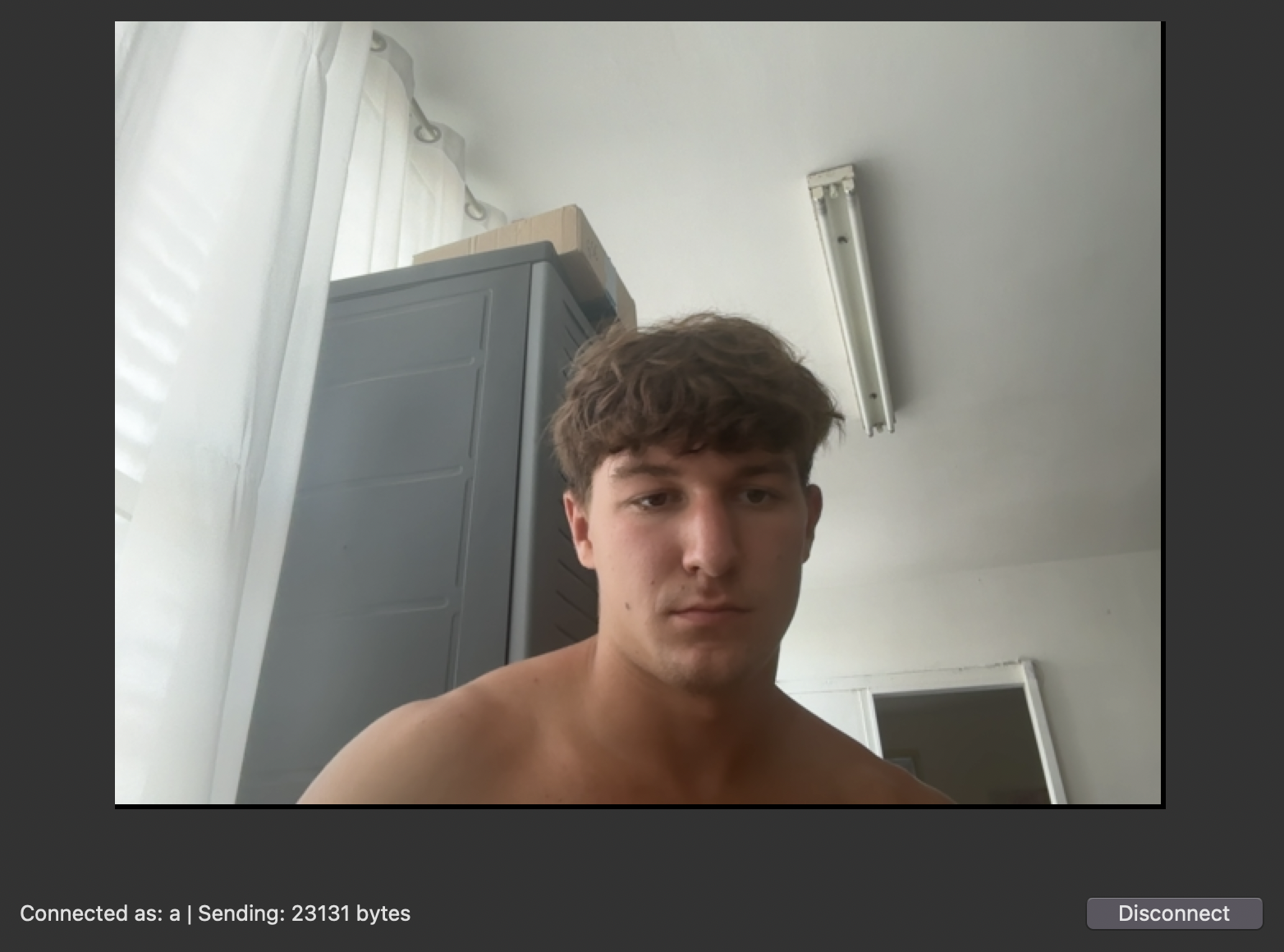


מדף החיפוש הראשי של הזרם המרכזי, תוכלו לעבור לדף שינוי הסיסמה - בו תוכלו גם לשנות את הסיסמה וגם להתנתק מחשבונכם.

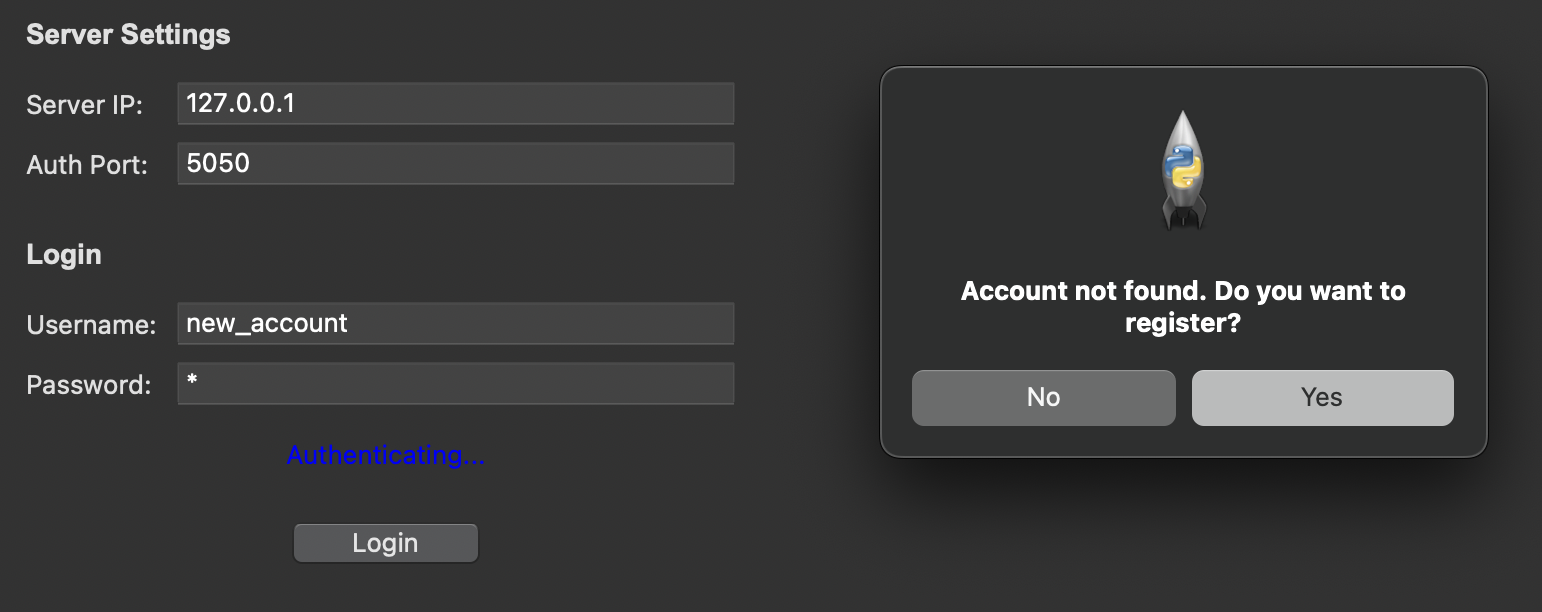
סטרימר:



לאחר הפעלת הקוד, נפתח חלון בו ניתן לציין את ה-IP והיציאה אליה יתחבר הסטרימר וגם להזין את פרטי הכניסה



כאשר מזין את נתוני החשבון הנכונים, המשתמש מתחיל את הזרם עם היכולת לעצור אותו בכל עת



אם אין לך חשבון אז גם ניתן להירשם שם המשתמש חדש

רפלקציה

התחלתי לעשות את הפרויקט שלי בתחילת החורף וסיימתי אותו בתחילת יוני, הוא לקח חצי שנה של עבודה ולימוד. תוך כדי עבודה על הפרויקט הזה למדתי הרבה, נתקלתי בכמות גדולה של בעיות, היה לי ניסיון של עבודה וחילופי מידע עם תלמידים אחרים שעבדו בכיוונים דומים - הייתי רוצה לחלוק קצת רשמים כלליים מהדרך שעברתי.

בתהליך העבודה על הפרויקט לא הכל הלך כמו שרציתי - נתקלתי בכמות מסוימת של קשיים. החל משינויים בטכנולוגיות שבהן השתמשתי: תכננתי להשתמש ב-PostgreSQL, בסופו של דבר השתמשתי ב-sqlite כיוון שהראשון התברר כמורכב יותר לשימוש וגדול יותר במשקל; כמו כן לאחר שכתבתי פרוטוקולי העברת נתונים דומים ל-hls ול-rtmp על פרוטוקול tcp הבנתי שזה פחות יעיל והייתי צריך להחליף ל-udp עם הוספה עוקבת של פיצול בגלל קשיים בגודל החבילות. כמו כן נתקלתי בבעיות במבנה הכללי של הפרויקט - בהתחלה תוכנן לעשות את הסטרימר באתר - אולם אחר כך הבנתי שזה הרבה יותר מורכב, בעיות של אבטחה וחוסר יכולת להשתמש בפרוטוקולים המתוכננים - בסופו של דבר הפתרון היה העברת הסטרימר ליישום נפרד עם GUI משלו.

בתהליך ביצוע הפרויקט למדתי הרבה הן באופן עצמאי והן במהלך עבודה עם תלמידים אחרים - חילופי מידע, ופגישות עם בוגרים, שהכירו ועזרו בפתרון כמה בעיות, הבהרת העבודה הכללית של המערכת. אני יכול לומר בביטחון שיש לי עכשיו הבנה מלאה של מימוש פרוטוקול עובד, למדתי עבודה עם websocket ובכלל עם אתרי http/s. חלק חשוב בלמידה אני חושב שהוא עבודה עם database. מלבד היתרונות האלה הפרויקט לימד אותי לסנן מידע שמקבלים, להחליף מידע בצורה יעילה עם תלמידים אחרים, וגם לחבר תוכנית עבודה ולהבין מטרות אקטואליות. חלק חשוב מזה אני חושב שהוא הרכבה ברורה של הצעד הבא בפרויקט. אני חושב שברובו בעתיד יעזור לי מערכת העבודה על הפרויקט וכישורי העבודה המסודרת שרכשתי, אולם גם הידע של הטכנולוגיות שבהן השתמשתי בפרויקט יכול להועיל.

בפרויקט השארתי מקום להוספה עתידית שלו - יש לי פונקציה שמבצעת העברת סטרים למשתמשים רגילים של יישומים - בלי websocket, דרך udp. החלק הזה משמש כתוספת לפרויקט הקיים ויכול להיות שיפור שלו. כמו כן בעתיד הייתי רוצה להוסיף הפעלה מרובה של סטרימרים - למימוש פלטפורמת סטרימינג מלאה עם כמה משתמשים שמנהלים סטרים במקביל.

אני מרוצה מאוד הן מהפרויקט בכלל והן מתהליך הביצוע שלו. אני חושב שבפרט הדרך שעברתי התברר במידה רבה כניסיון מעניין ומועיל שאיתו יתקשר לי באופן כללי הלימוד.

הייתי מאוד רוצה להודות בנפרד לכל מי שהשתתף בביצוע הפרויקט - הן לבוגרים שבאו לעזור, והן למנהל הכיתה cyber - ניר. הם עזרו להבין איך אפשר לממש משימה זו או אחרת ביעילות נותנים רעיונות או עוזרים לי להשיג הבנה של הצעדים הבאים בפרויקט. תודה!