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.**

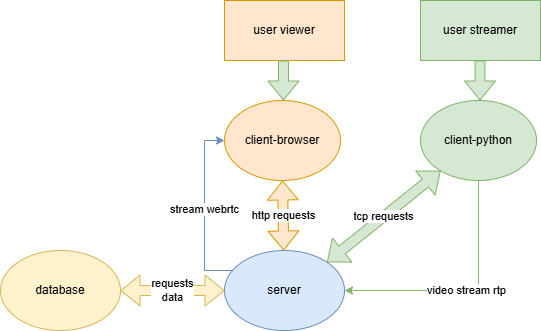
**מסד הנתונים**

* **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).

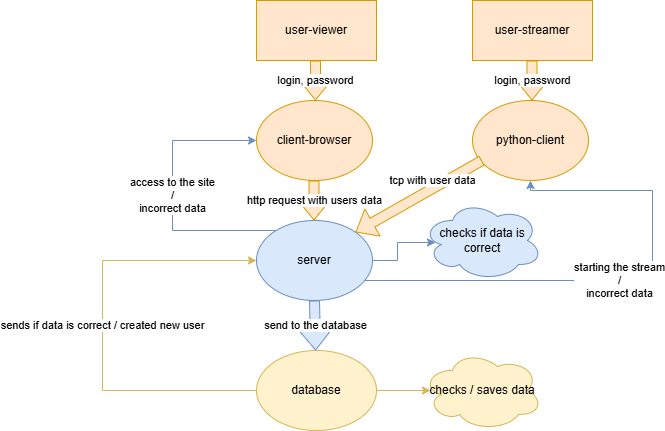
**בקשה מהצופה:**

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

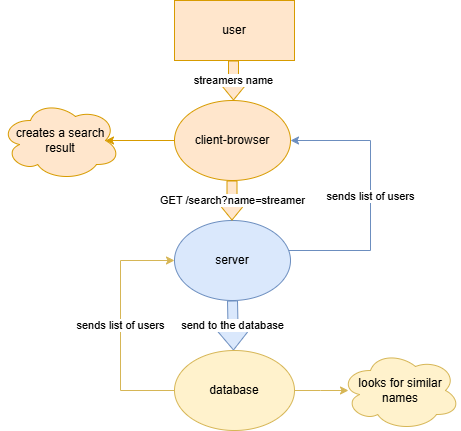
**סיום הזרם:**

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

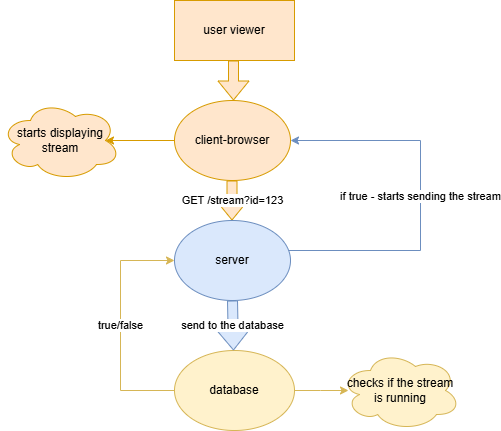
הרשמה / כניסה



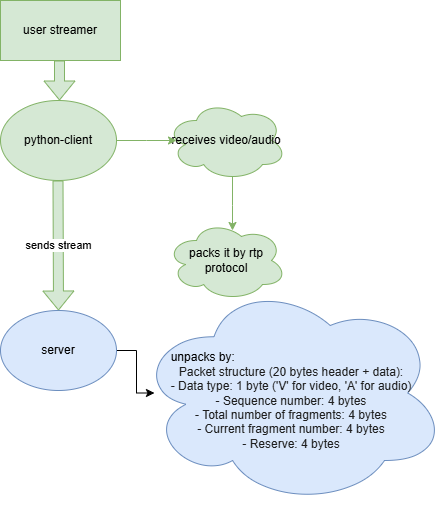
חיפוש



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)**

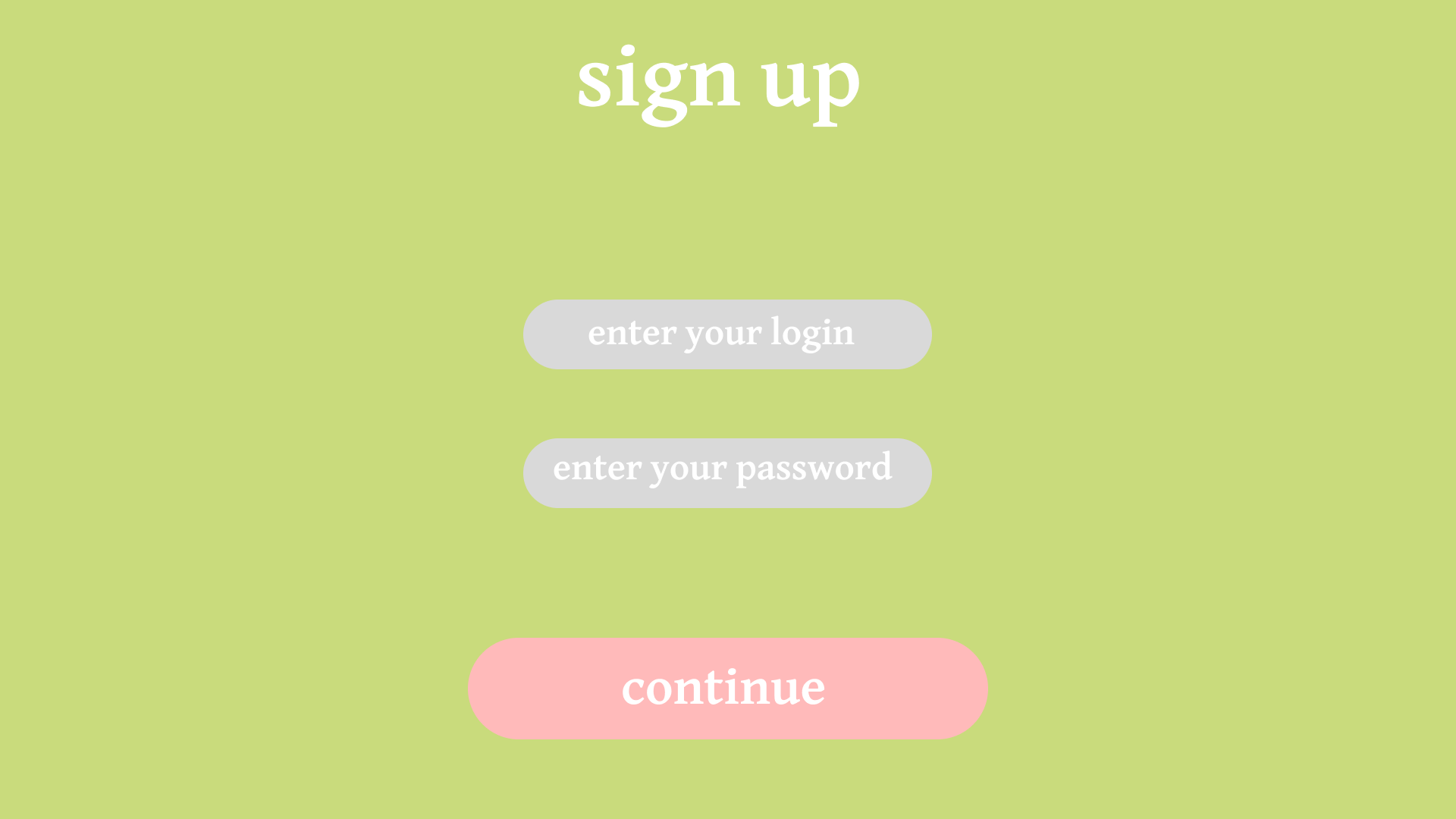
**תפקיד המסך**: התחברות וניהול שידור עבור הסטרימר.  
**תיאור**: הסטרימר בוחר שרת, מתחבר, מתחיל שידור ויכול לסיים אותו.  
**פונקציונליות**:

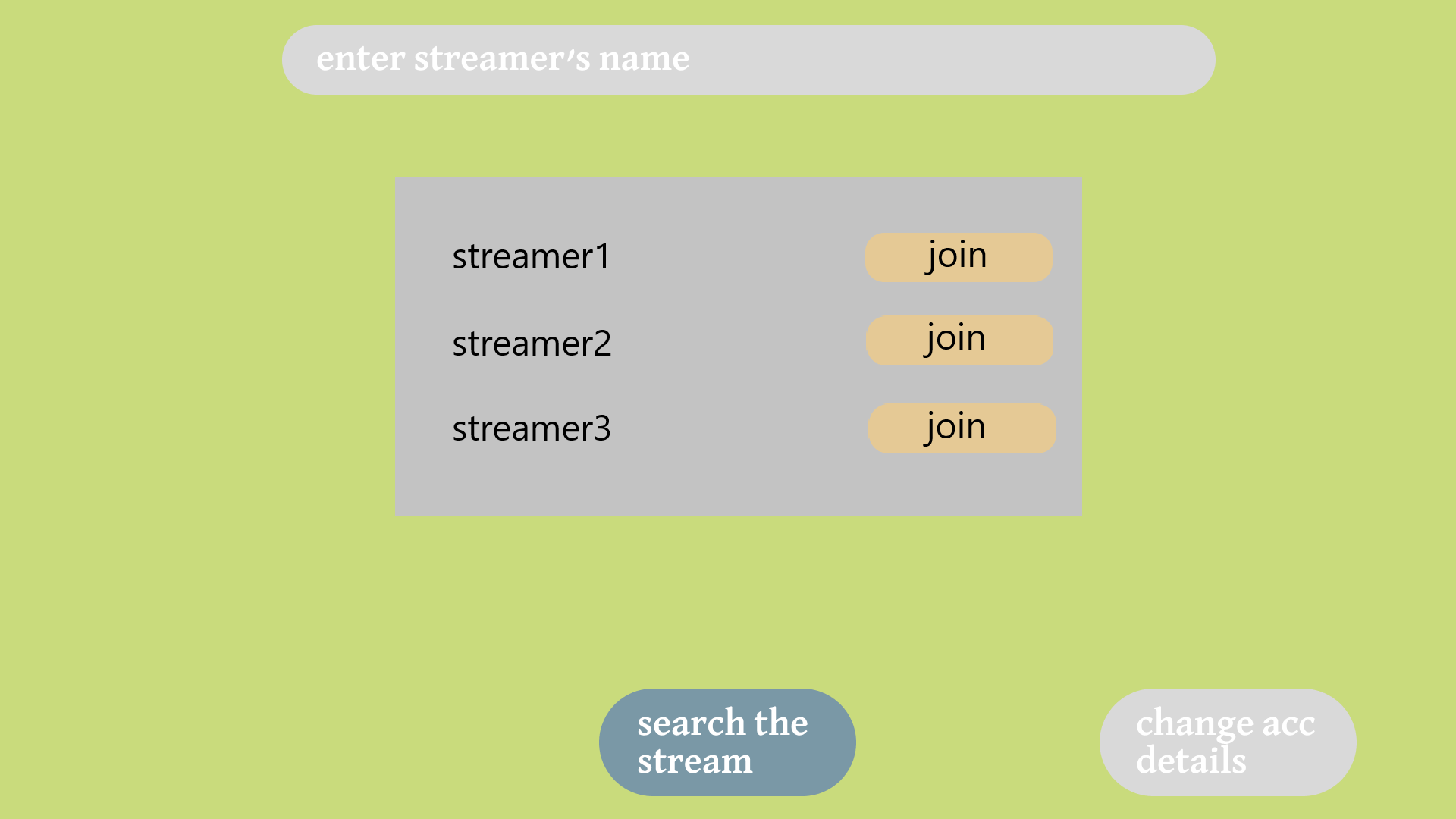
* התחברות ובחירת כתובת IP ו־port של השרת
* כפתור "סיים שידור"

### **מעברים בין מסכים**

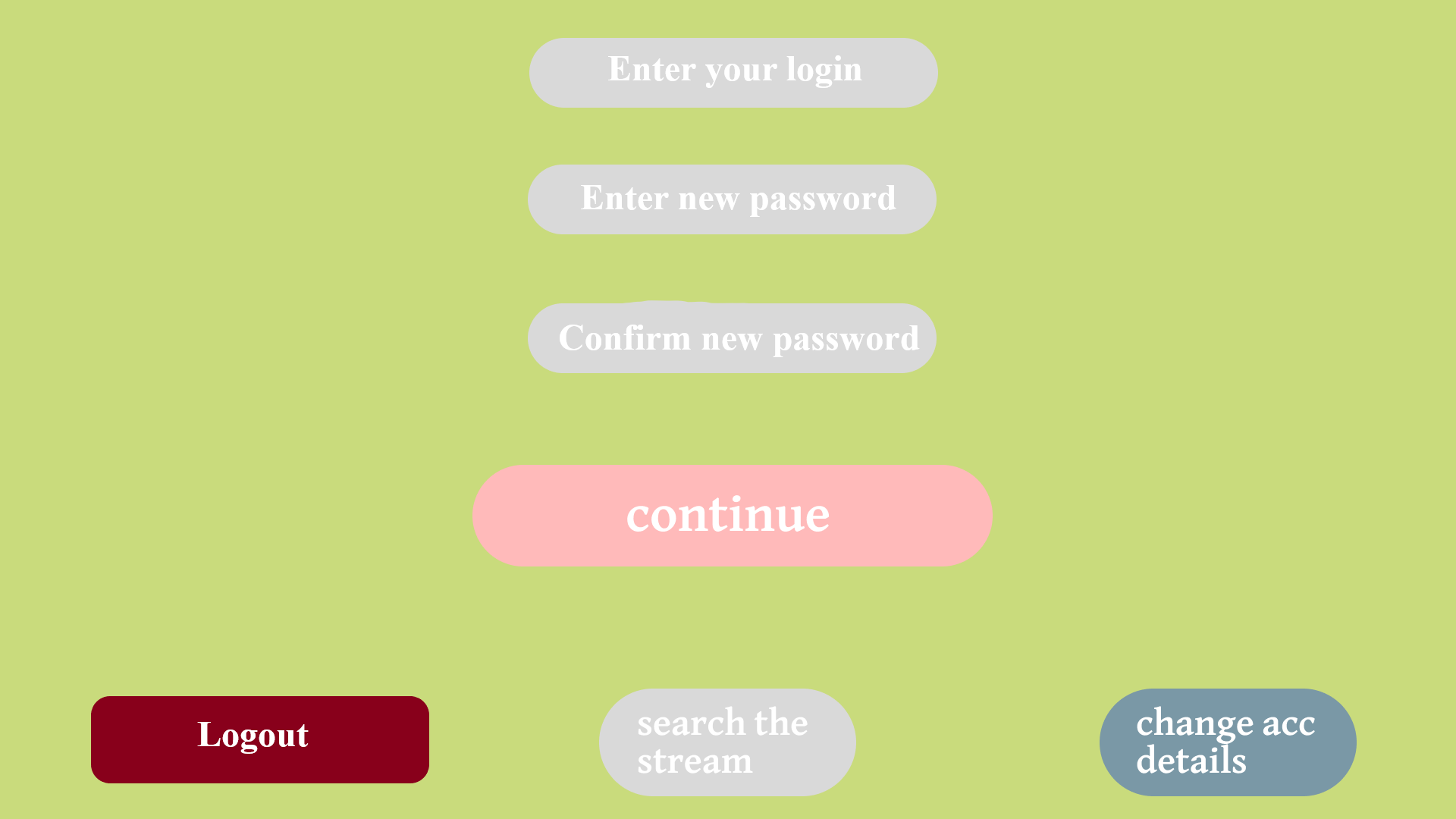
* אם המשתמש אינו רשום — הוא מועבר למסך הרישום
* לאחר רישום/התחברות מוצלחת — מעבר לדף הבית
* מהדף הראשי ניתן לעבור למסך צפייה בשידור
* מהשידור ניתן לעבור לעמוד הפרופיל, עם אפשרות יציאה וחזרה

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 נתונים user\_databse.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

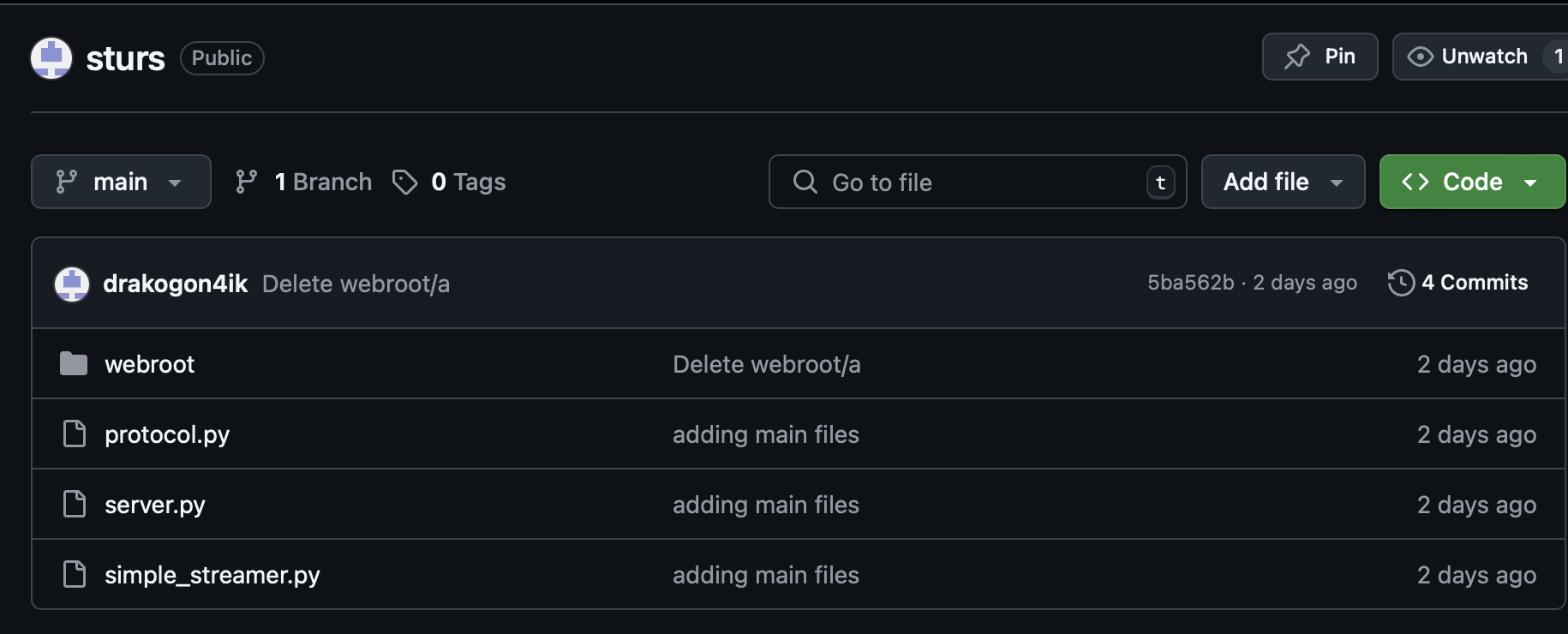
לקוח<----[TCP, SSL]---- שרת

* 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

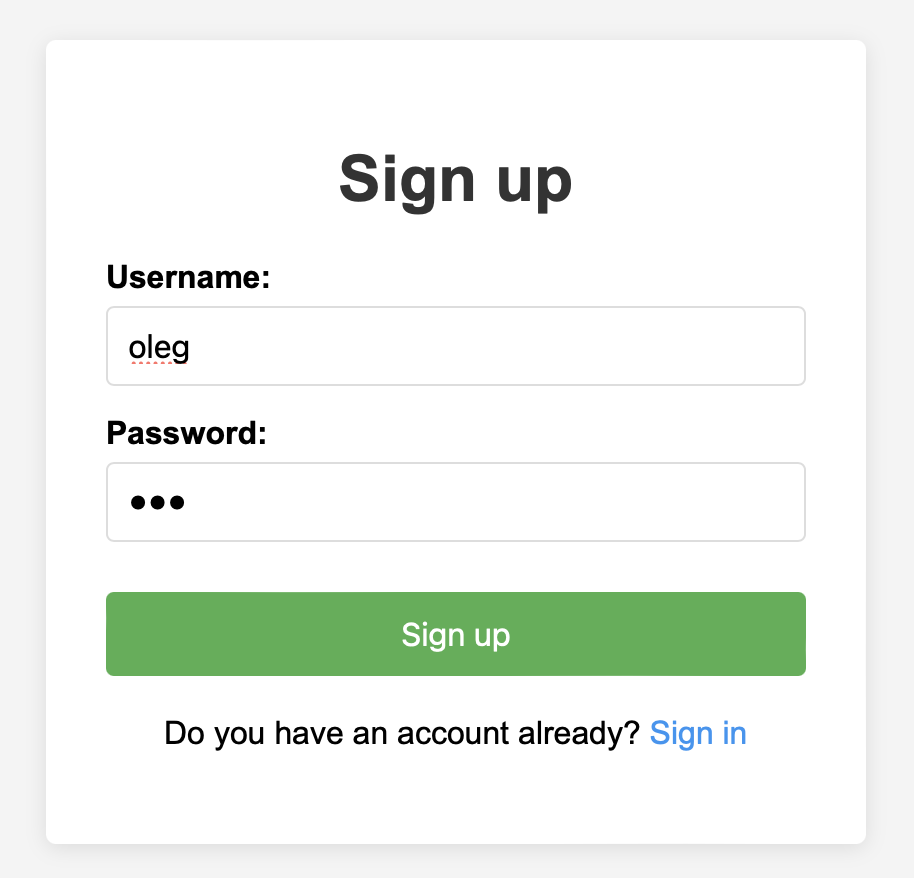
* [server.py](http://server.py) זה השרת שפועל אתר לצופים ושומר על קשר עם סטרימר (מוכן לקבל סטרימר חדש), מפעיל websocket, מבצע העברות נתונים הן ל-websocket והן לצופי udp פשוטים.
* [simple-streamer.py](http://simple-streamer.py) זהו קובץ להרצת הקוד עם ממשק הסטרימינג - המשתמש מציין את ה-IP והיציאה וכן מכניס 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.

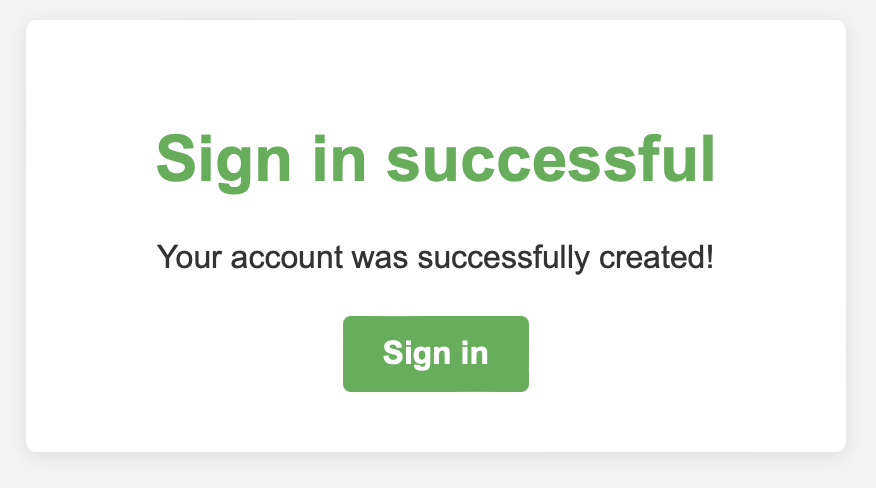
תהליך עבודת האתר

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

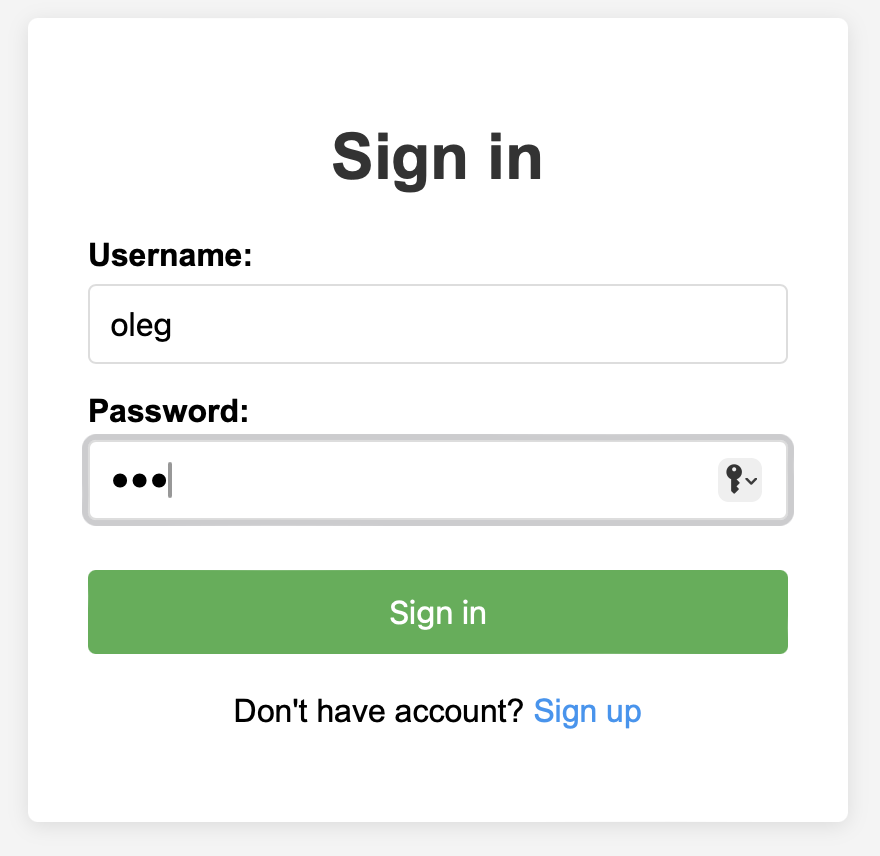
אתר:

הצופה מועבר לעמוד הרשמה בו הוא יכול ליצור את חשבונו.

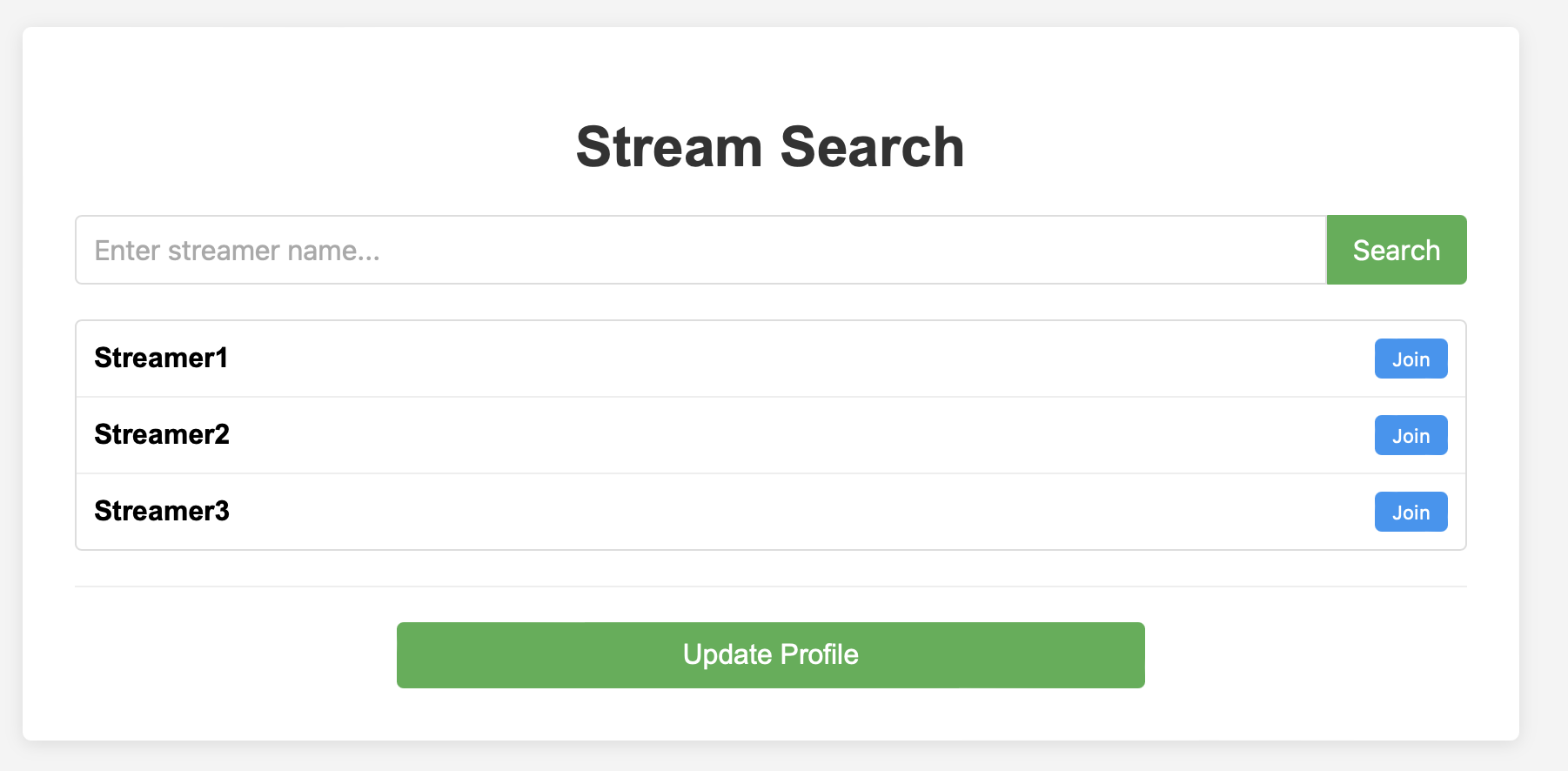




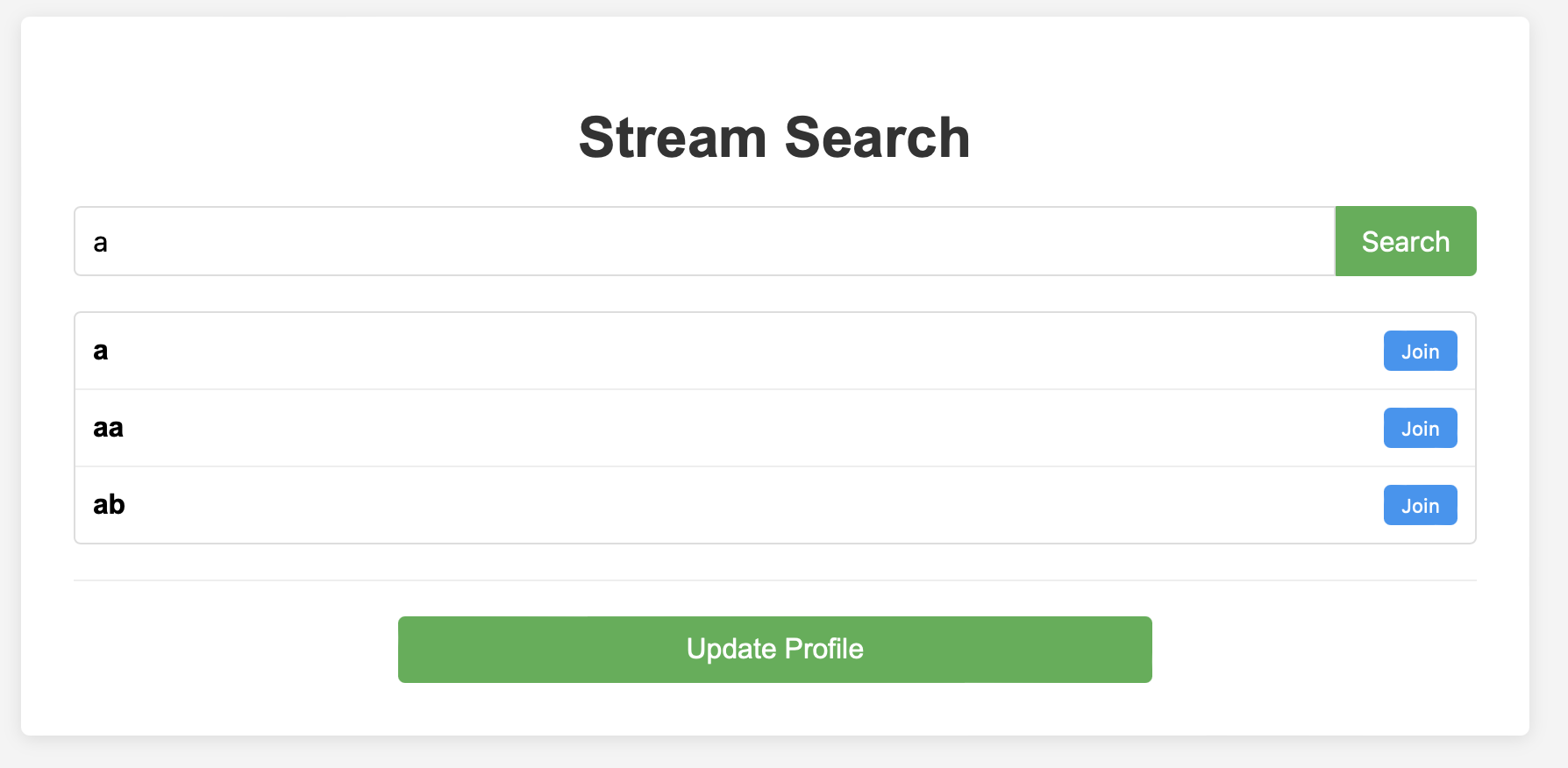
לאחר מכן הצופה מקבל הודעה שהשם נלקח או שהרישום הצליח.



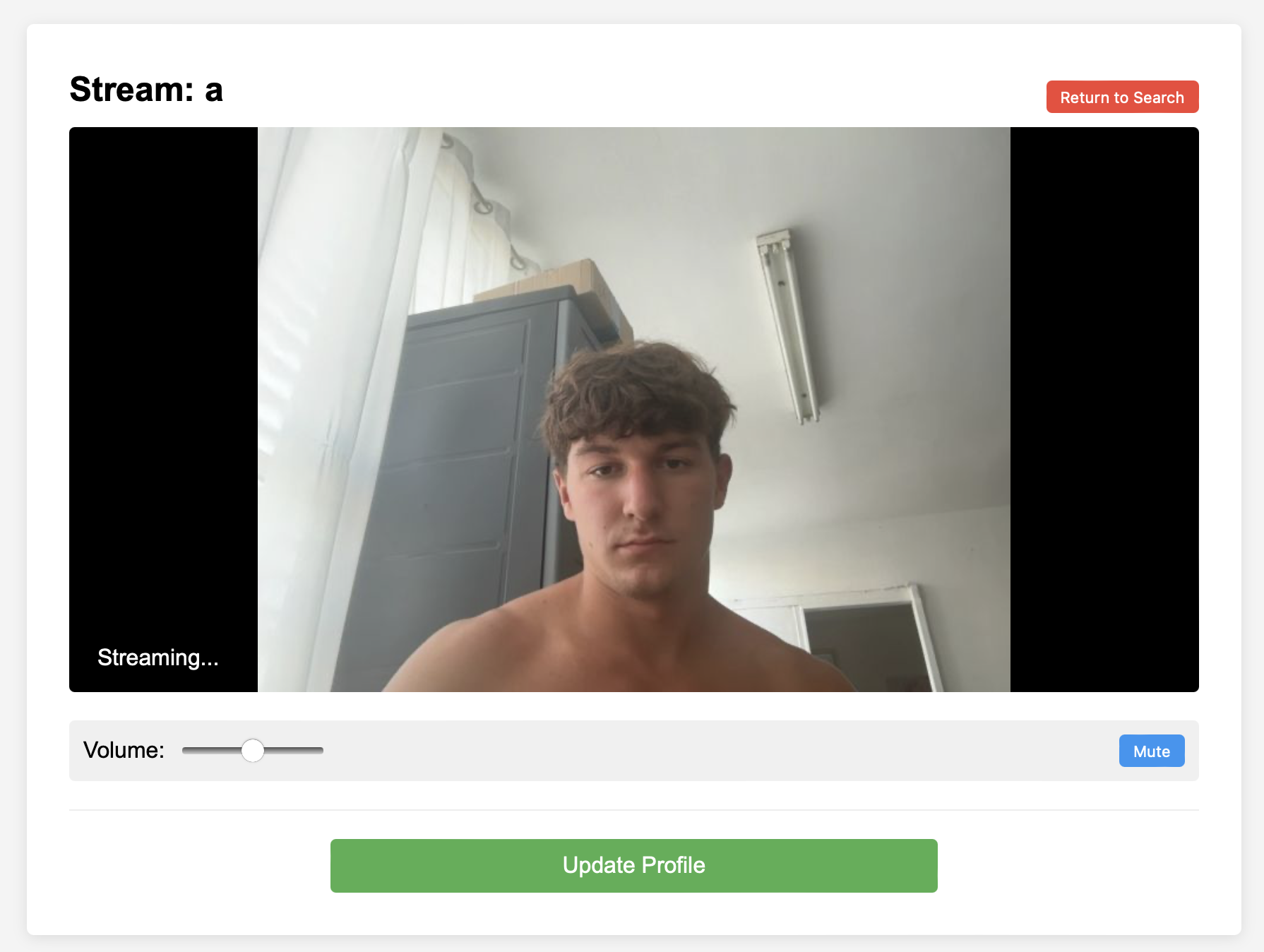
הצופה מועבר לדף הכניסה בו הוא מזין את פרטי החשבון שלו.



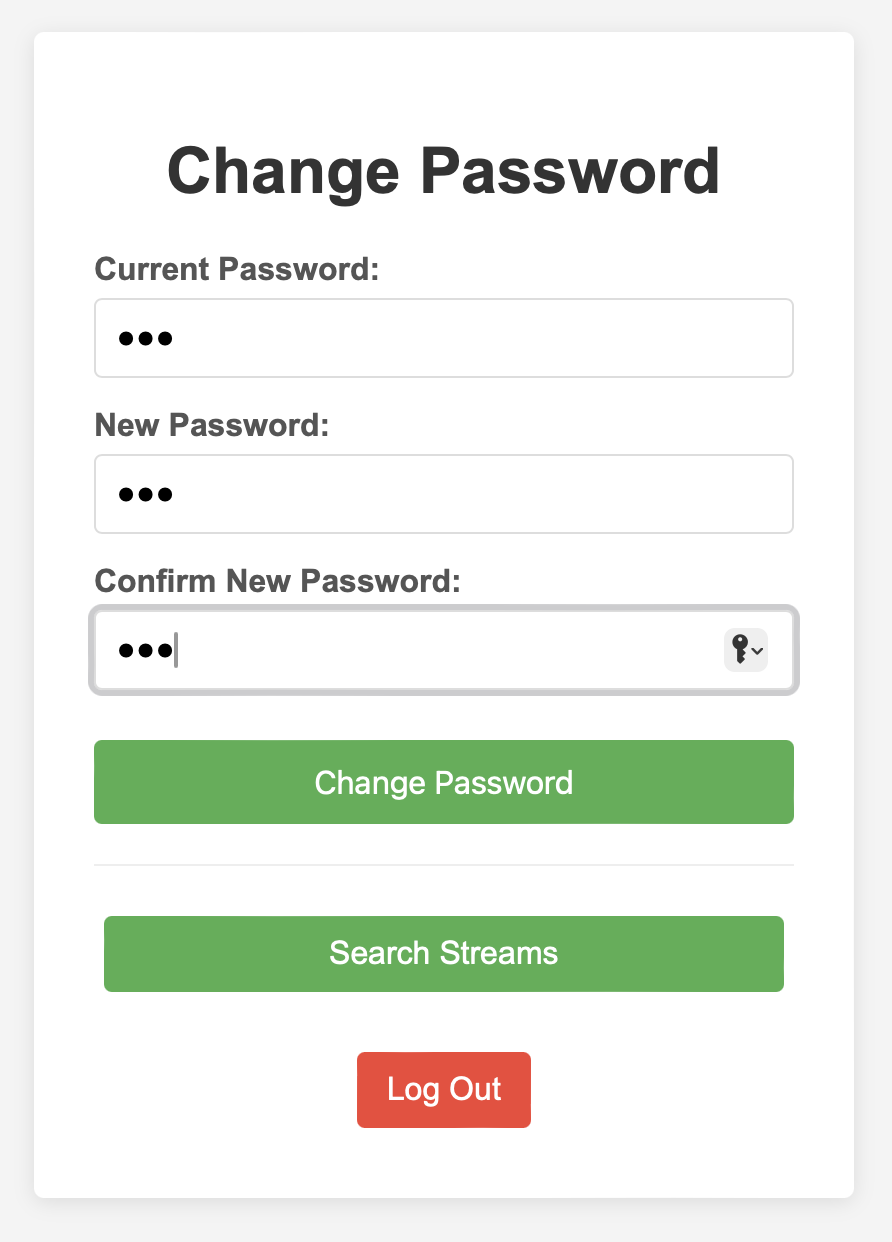
הצופה עובר לדף החיפוש של הסטרימר.



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

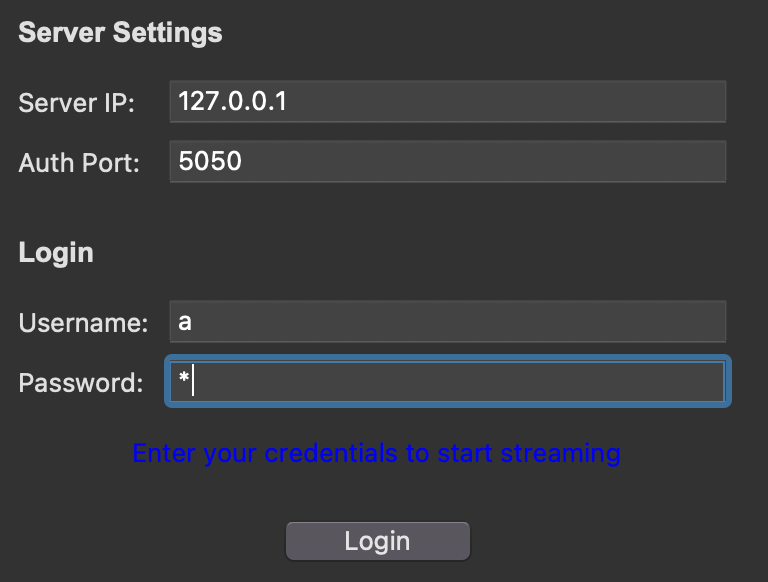


כאשר אתה מתחבר לזרם עובד, מתחיל זרם שבו אתה יכול לשנות את רמת הווליום ולהתנתק בכל עת.

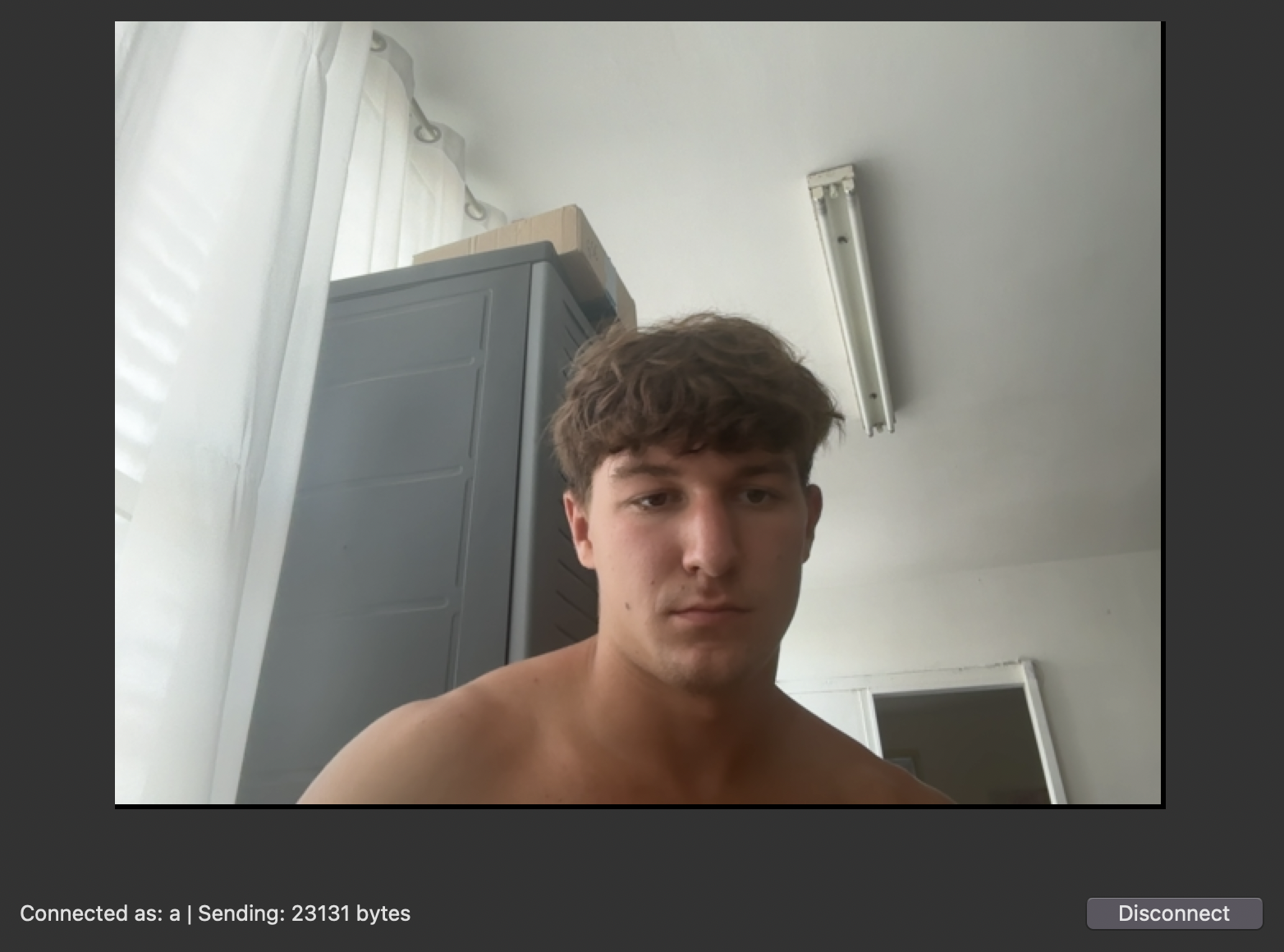


מדף החיפוש הראשי של הזרם המרכזי, תוכלו לעבור לדף שינוי הסיסמה - בו תוכלו גם לשנות את הסיסמה וגם להתנתק מחשבונכם.

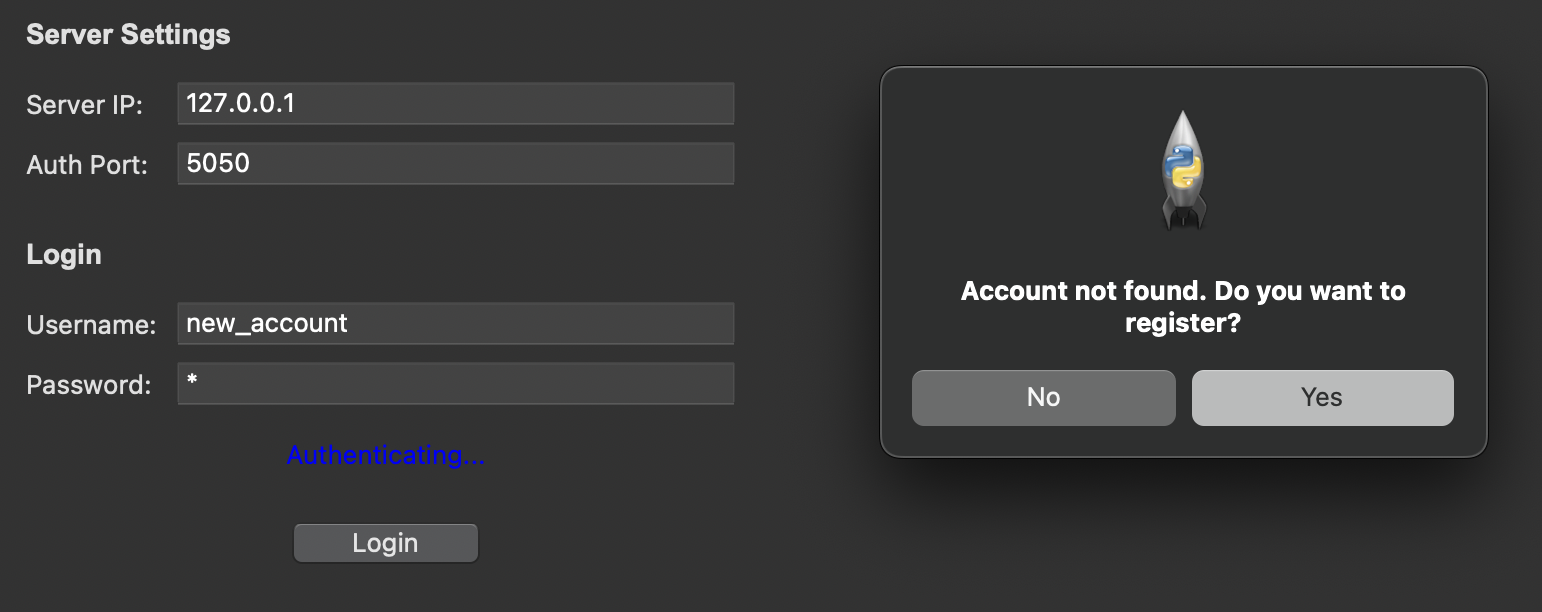
סטרימר:



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!

קוד הפרויקט

server

*"""*

*Author: Oleg Shkolnik*

*Description: before running the server its importnatn to download ssl from the site https://slproweb.com/products/Win32OpenSSL.html*

*and then write this command in the powershell*

*& "C:\Program Files\OpenSSL-Win64\bin\openssl.exe" req -x509 -newkey rsa:2048 -keyout server.key -out server.crt -days 365 -nodes -subj "/CN=localhost"*

*Server that receives GET and POST requests from client (site)*

*and sends appropriate responses with user authentication support.*

*Date: 3/06/25*

*"""*

import socket

import os

import sqlite3

import urllib.parse

import hashlib

import re

import json

import time

import struct

import threading

import cv2

import numpy as np

import queue

import websockets

import asyncio

import base64

import datetime

from protocol import StreamProtocol, AuthProtocol

import ssl

import logging

# Storage for WebSocket connections

ws\_clients = {}

HOST = '0.0.0.0'

PORT = 80

HTTPS\_PORT = 443 # Standard port for HTTPS

SSL\_CERT\_FILE = 'server.crt' # Path to certificate

SSL\_KEY\_FILE = 'server.key' # Path to private key

TCP\_AUTH\_SSL = True # Flag to enable/disable SSL

AUTH\_SSL\_CERT\_FILE = 'server.crt' # Path to SSL certificate

AUTH\_SSL\_KEY\_FILE = 'server.key'

SITE\_FOLDER = 'webroot'

MOVED\_URL = '/index.html'

SOCKET\_TIMEOUT = 10 # Increased socket timeout for Windows

specific\_urls = ['forbidden', 'moved', 'error']

request\_error = b"HTTP/1.1 400 Bad Request\r\n\r\n<h1>400 Bad Request</h1>"

types = {

'tml': 'text/html;charset=utf-8',

'css': 'text/css',

'.js': 'text/javascript; charset=UTF-8',

'txt': 'text/plain',

'ico': 'image/x-icon',

'gif': 'image/jpeg',

'jpg': 'image/jpeg',

'png': 'image/jpeg'

}

protected\_pages = [

'search-stream.html',

'profile.html',

'start-stream.html',

'profile.html',

'profile-true.html'

]

# TCP port for authentication

TCP\_AUTH\_PORT = 5050

# UDP port for streaming

PORT\_UDP = 5001

PORT\_CLIENT = 5002

ws\_port = 8765

# Global variables

frame = None

frame\_lock = threading.Lock()

clients = []

clients\_lock = threading.Lock()

running = True

# Queue for broadcast delivery

broadcast\_queue = queue.Queue(maxsize=100)

# Global variable to track active streamer

current\_streamer = None

streamer\_disconnected = threading.Event()

def setup\_logging():

*"""*

*Sets up logging configuration for the server*

*"""*

# Create logs directory if it doesn't exist

if not os.path.exists('logs'):

os.makedirs('logs')

# Configure logging format

log\_format = '%(asctime)s - %(name)s - %(levelname)s - %(funcName)s:%(lineno)d - %(message)s'

# Set up file handler (logs to file)

file\_handler = logging.FileHandler(

f'logs/server\_{datetime.datetime.now().strftime("%Y%m%d")}.log'

)

file\_handler.setLevel(logging.DEBUG)

file\_handler.setFormatter(logging.Formatter(log\_format))

# Set up console handler (logs to terminal)

console\_handler = logging.StreamHandler()

console\_handler.setLevel(logging.INFO)

console\_handler.setFormatter(logging.Formatter(log\_format))

# Create logger

log = logging.getLogger('HTTPServer')

log.setLevel(logging.DEBUG)

# Add handlers to logger

log.addHandler(file\_handler)

log.addHandler(console\_handler)

return log

# Initialize logger at module level

logger = setup\_logging()

class StreamManager:

*"""Class for managing streams and streamer state"""*

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

self.db\_name = db\_name

self.current\_streamer = None

self.streamer\_disconnected = threading.Event()

self.broadcast\_queue = queue.Queue(maxsize=100)

self.ws\_clients = {}

self.frame = None

self.frame\_lock = threading.Lock()

def is\_streaming(self, username):

*"""Checks if user is streaming"""*

try:

conn = sqlite3.connect(self.db\_name)

cursor = conn.cursor()

cursor.execute('SELECT \* FROM active\_streams WHERE username = ?', (username,))

result = cursor.fetchone() is not None

conn.close()

logger.debug(f"Checking stream for {username}: {result}")

return result

except Exception as e:

logger.error(f"Error checking stream status: {e}")

return False

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

*"""Adds user to active streams table"""*

conn = None

try:

conn = sqlite3.connect(self.db\_name)

cursor = conn.cursor()

cursor.execute("DELETE FROM active\_streams WHERE username = ?", (username,))

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

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

conn.commit()

logger.info(f"User {username} added to active streams")

except sqlite3.Error as e:

logger.error(f"Error adding to active streams: {e}")

if conn:

conn.rollback()

finally:

if conn:

conn.close()

def cleanup\_database(self, username):

*"""Removes user from active streams table"""*

try:

conn = sqlite3.connect(self.db\_name)

cursor = conn.cursor()

cursor.execute("DELETE FROM active\_streams WHERE username = ?", (username,))

conn.commit()

logger.info(f"User {username} removed from active streams")

conn.close()

except sqlite3.Error as e:

logger.error(f"Error removing from active streams: {e}")

def cleanup\_after\_streamer(self):

*"""Cleanup after streamer disconnection"""*

# Clear broadcast queue

while not self.broadcast\_queue.empty():

try:

self.broadcast\_queue.get\_nowait()

except queue.Empty:

pass

# Clear frame

with self.frame\_lock:

self.frame = None

# Send notification to clients

if self.current\_streamer and self.current\_streamer in self.ws\_clients:

for client in self.ws\_clients[self.current\_streamer][:]:

try:

asyncio.run\_coroutine\_threadsafe(

send\_ws\_message(client, json.dumps({

"type": "info",

"message": "Streamer disconnected, broadcast ended"

})),

loop

)

except Exception as e:

logger.error(f"Error sending notification: {e}")

self.current\_streamer = None

class SessionManager:

*"""Class for managing user sessions"""*

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

self.db\_name = db\_name

self.active\_sessions = {}

def create\_session(self, username):

*"""Creates a new session for user"""*

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

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

return session\_id

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

*"""Extracts username from session"""*

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

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

*"""Removes session"""*

if session\_id in self.active\_sessions:

del self.active\_sessions[session\_id]

return True

return False

def authenticate\_user(self, username, password):

*"""Verifies user credentials"""*

try:

conn = sqlite3.connect(self.db\_name)

cursor = conn.cursor()

hashed\_password = hashlib.sha256(password.encode()).hexdigest()

cursor.execute('SELECT \* FROM users WHERE username = ? AND password = ?',

(username, hashed\_password))

user = cursor.fetchone()

conn.close()

return user is not None

except Exception as e:

logger.error(f"Error authenticating user: {e}")

return False

def register\_user(self, username, password):

*"""Registers a new user"""*

try:

conn = sqlite3.connect(self.db\_name)

cursor = conn.cursor()

cursor.execute('SELECT \* FROM users WHERE username = ?', (username,))

if cursor.fetchone():

conn.close()

return False

hashed\_password = hash\_password(password)

cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)',

(username, hashed\_password))

conn.commit()

conn.close()

return True

except Exception as e:

logger.error(f"Error registering user: {e}")

return False

stream\_manager = StreamManager()

session\_manager = SessionManager()

stream\_protocol = StreamProtocol()

def create\_ssl\_context():

*"""Creates and configures SSL context for authentication"""*

ssl\_context = ssl.create\_default\_context(ssl.Purpose.CLIENT\_AUTH)

try:

ssl\_context.load\_cert\_chain(certfile=AUTH\_SSL\_CERT\_FILE, keyfile=AUTH\_SSL\_KEY\_FILE)

return ssl\_context

except Exception as e:

logger.error(f"Error setting up SSL for auth server: {e}")

logger.warning("Running auth server without SSL")

return None

def auth\_server\_thread():

*"""Thread to handle TCP authentication with SSL support"""*

global running

# Create TCP socket for auth

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

tcp\_auth\_socket.listen(5)

# Setup SSL if enabled

ssl\_context = None

if TCP\_AUTH\_SSL:

ssl\_context = create\_ssl\_context()

if ssl\_context:

logger.info(f"TCP Authentication server started with SSL on {HOST}:{TCP\_AUTH\_PORT}")

else:

logger.info(f"TCP Authentication server started without SSL on {HOST}:{TCP\_AUTH\_PORT}")

else:

logger.info(f"TCP Authentication server started on {HOST}:{TCP\_AUTH\_PORT}")

while running:

try:

# Accept connection

client\_socket, addr = tcp\_auth\_socket.accept()

logger.debug(f"New auth connection from {addr}")

# Wrap socket with SSL if enabled

if TCP\_AUTH\_SSL and ssl\_context:

try:

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

logger.debug(f"SSL handshake successful with {addr}")

except ssl.SSLError as e:

logger.warning(f"SSL handshake failed with {addr}: {e}")

client\_socket.close()

continue

except Exception as e:

logger.error(f"SSL error with {addr}: {e}")

client\_socket.close()

continue

# Process auth in a separate thread

auth\_thread = threading.Thread(

target=handle\_tcp\_authentication,

args=(client\_socket, addr)

)

auth\_thread.daemon = True

auth\_thread.start()

except Exception as e:

if running: # Only log error if not shutting down

logger.error(f"Error in auth server thread: {e}")

tcp\_auth\_socket.close()

def handle\_tcp\_authentication(client\_socket, addr):

*"""Handle single client authentication over TCP"""*

global current\_streamer

auth\_protocol = AuthProtocol()

try:

# Authenticate user

username = auth\_protocol.server\_authenticate(

client\_socket,

auth\_function=session\_manager.authenticate\_user,

register\_function=session\_manager.register\_user

)

if not username:

logger.warning(f"Authentication failed for {addr}")

return

logger.info(f"User {username} successfully authenticated from {addr}")

# Send UDP port for streaming back to the client

auth\_protocol.send\_message(client\_socket, str(PORT\_UDP))

# Add user to active streams

current\_streamer = username

stream\_manager.add\_user\_to\_active\_streams(username)

# Notify current streamer is ready for UDP connection

logger.info(f"Streamer {username} authorized and ready to connect via UDP")

except Exception as e:

logger.error(f"Error in TCP authentication handler: {e}")

finally:

client\_socket.close()

def streamer\_listener():

*"""Main loop waiting for streamer connection"""*

global running, current\_streamer, streamer\_disconnected

logger.info("Streamer UDP connection handler started")

while running:

try:

# Wait and process new streamer via UDP

# Only after authentication has been completed via TCP

if current\_streamer:

handle\_streamer\_udp\_connection(current\_streamer)

# Notify all connected clients about new streamer

if current\_streamer and current\_streamer in ws\_clients:

for client in ws\_clients[current\_streamer][:]:

try:

asyncio.run\_coroutine\_threadsafe(

send\_ws\_message(client, json.dumps({

"type": "new\_streamer",

"streamer": current\_streamer,

"message": "New streamer started broadcasting"

})),

loop

)

except Exception as e:

logger.error(f"Error sending new streamer notification: {e}")

# Wait for disconnect signal to move to next streamer

streamer\_disconnected.wait()

streamer\_disconnected.clear()

# Cleanup before next connection

stream\_manager.cleanup\_after\_streamer()

# Reset current streamer

current\_streamer = None

# Sleep a bit to prevent CPU overload if no streamer

else:

time.sleep(0.5)

except Exception as e:

logger.error(f"Error in streamer processing loop: {e}")

time.sleep(1) # Pause before retry

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

*"""Handle UDP streaming connection for authenticated user"""*

global frame, broadcast\_queue

# Clear buffers through protocol

stream\_protocol.clear\_buffers()

# Clear broadcast queue when connecting new streamer

while not broadcast\_queue.empty():

try:

broadcast\_queue.get\_nowait()

except queue.Empty:

pass

# Create UDP socket for streaming

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

logger.info(f"UDP server ready for streamer {username} on {HOST}:{PORT\_UDP}")

# Receive stream from streamer who was already authenticated

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

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

*"""Receive stream data from authorized streamer"""*

global frame, streamer\_disconnected

# Set timeout to detect disconnection

udp\_socket.settimeout(1.0)

disconnect\_counter = 0

last\_packet\_time = time.time()

while running:

try:

# Get packet

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

# Reset disconnect counter when data received

disconnect\_counter = 0

last\_packet\_time = time.time()

# Process fragment through protocol

parsed\_data = stream\_protocol.process\_fragment(data)

if not parsed\_data:

continue

data\_type, packet\_seq, total\_fragments, fragment\_num, payload = parsed\_data

# Try to assemble full packet

full\_data = stream\_protocol.reassemble\_fragment(

data\_type, packet\_seq, total\_fragments, fragment\_num, payload

)

if full\_data:

if data\_type == "V":

# Decode video through protocol

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

if new\_frame is not None:

with frame\_lock:

frame = new\_frame

# Add to broadcast queue

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

elif data\_type == "A":

# Add audio to broadcast queue

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

except socket.timeout:

# Handle timeout - possible streamer disconnection

disconnect\_counter += 1

current\_time = time.time()

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

# If no packets for a long time (5 seconds), consider streamer disconnected

if disconnect\_counter >= 5 or inactive\_time > 10:

logger.info(f"Streamer {username} disconnected (no data for {inactive\_time:.1f} sec)")

stream\_manager.cleanup\_database(username)

streamer\_disconnected.set() # Signal for next streamer

break

except Exception as e:

logger.error(f"Error receiving stream: {e}")

# For serious errors, also disconnect streamer

stream\_manager.cleanup\_database(username)

streamer\_disconnected.set()

break

def init\_database():

*"""*

*Creates SQLite database and necessary tables if they don't exist*

*"""*

conn = sqlite3.connect('user\_database.db')

cursor = conn.cursor()

# Create users table

cursor.execute('''

CREATE TABLE IF NOT EXISTS users (

id INTEGER PRIMARY KEY AUTOINCREMENT,

username TEXT UNIQUE NOT NULL,

password TEXT NOT NULL

)

''')

# Create active streams table

cursor.execute('''

CREATE TABLE IF NOT EXISTS active\_streams (

username TEXT PRIMARY KEY,

start\_time TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

FOREIGN KEY (username) REFERENCES users(username)

)

''')

conn.commit()

conn.close()

logger.info("Database initialized")

def hash\_password(password):

*"""*

*Hashes password using SHA-256*

*:param password: password in text form*

*:return: hashed password*

*"""*

return hashlib.sha256(password.encode()).hexdigest()

def choosing\_type(filename):

*"""*

*Function for searching type of file*

*:param filename: file which type we want to know*

*:return: type*

*"""*

extension = filename.split('.')[-1]

if extension == 'html' or extension == 'htm':

return types['tml']

elif len(extension) == 3 and extension in types:

return types[extension]

else:

return 'text/plain'

def specific(filename):

*"""*

*Function checks if we have specific url*

*:param filename: specific part*

*:return: true or false*

*"""*

return filename in specific\_urls

def searching\_url(filename):

*"""*

*Function that determines what response we need to send on specific url*

*:param filename: specific url*

*:return: specific response*

*"""*

response = b''

if filename == 'forbidden':

response = b"HTTP/1.1 403 Forbidden\r\n\r\n<h1>403 Forbidden</h1>"

elif filename == 'moved':

response = b"HTTP/1.1 302 Moved Temporarily\r\nLocation: " + bytes(MOVED\_URL, 'utf-8') + b"\r\n\r\n"

elif filename == 'error':

response = b"HTTP/1.1 500 Internal Server Error\r\n\r\n<h1>500 Internal Server Error</h1>"

return response

def validating\_get\_request(request):

*"""*

*Function validates if GET request is correct*

*:param request: GET request in type of list*

*:return: true if request is correct and false if not*

*"""*

return request[0] == "GET" and request[2] == "HTTP/1.1"

def validating\_post\_request(request):

*"""*

*Function validates if POST request is correct*

*:param request: POST request in type of list*

*:return: true if request is correct and false if not*

*"""*

return request[0] == "POST" and request[2] == "HTTP/1.1"

def receive\_all(sock, def\_size=1024):

*"""*

*Receives data from client until HTTP request is complete*

*:param sock: client socket*

*:param def\_size: buffer size for reading*

*:return: complete request in bytes*

*"""*

data = b''

try:

# Set timeout for socket

sock.settimeout(SOCKET\_TIMEOUT)

while True:

chunk = sock.recv(def\_size)

if not chunk: # Connection closed

break

data += chunk

# Determine end of request

if b'\r\n\r\n' in data: # Standard HTTP header separator

# For GET requests, headers are enough

if data.startswith(b'GET'):

break

# For POST requests, check if we received entire body

elif data.startswith(b'POST'):

match = re.search(rb'Content-Length:\s\*(\d+)', data)

if match:

content\_length = int(match.group(1))

headers\_end = data.find(b'\r\n\r\n') + 4

body\_length = len(data) - headers\_end

if body\_length >= content\_length:

break

# For other request types also finish reading

else:

break

except socket.timeout:

logger.warning("Timeout receiving data from client")

except Exception as e:

logger.error(f"Error reading data: {e}")

return data

def serve\_file(client\_socket, filename):

*"""*

*This function opens a file requested from the browser and sends its contents to the client via a socket.*

*If the file is not found, a "404 Not Found" message is sent.*

*:param client\_socket: socket of client*

*:param filename: path to the file*

*"""*

try:

with open(filename, 'rb') as file:

content = file.read()

content\_type = choosing\_type(filename)

headers = f"HTTP/1.1 200 OK\r\n"

headers += f"Content-Type: {content\_type}\r\n"

headers += f"Content-Length: {len(content)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode() + content

client\_socket.sendall(response)

except FileNotFoundError:

try:

with open(os.path.join(SITE\_FOLDER, 'imgs', 'error.jpg'), 'rb') as file:

content = file.read()

headers = f"HTTP/1.1 404 Not Found\r\n"

headers += f"Content-Type: {types['jpg']}\r\n"

headers += f"Content-Length: {len(content)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode() + content

client\_socket.sendall(response)

except FileNotFoundError:

# If error file not found, send simple error message

content = b"<h1>404 Not Found</h1>"

headers = f"HTTP/1.1 404 Not Found\r\n"

headers += f"Content-Type: text/html\r\n"

headers += f"Content-Length: {len(content)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode() + content

client\_socket.sendall(response)

def handle\_registration(client\_socket, form\_data):

*"""*

*Processes user registration request*

*:param client\_socket: client socket*

*:param form\_data: form data*

*"""*

# Check for required fields

if 'username' not in form\_data or 'password' not in form\_data:

send\_error\_page(client\_socket, "Username and password are required", "register.html")

return

username = form\_data['username']

password = form\_data['password']

# Check that username is not empty

if not username or not password:

send\_error\_page(client\_socket, "Username and password cannot be empty", "register.html")

return

# Try to register user

if session\_manager.register\_user(username, password):

# Successful registration - redirect to success page

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

response += b"Location: /registration-success.html\r\n"

response += b"Content-Length: 0\r\n"

response += b"Connection: close\r\n\r\n"

client\_socket.sendall(response)

logger.info(f"User {username} successfully registered")

else:

# User already exists

send\_error\_page(client\_socket, "Username already taken", "register.html", username)

def handle\_login(client\_socket, form\_data):

*"""*

*Processes user login request*

*:param client\_socket: client socket*

*:param form\_data: form data*

*"""*

# Check for required fields

if 'username' not in form\_data or 'password' not in form\_data:

send\_error\_page(client\_socket, "Username and password are required", "login.html")

return

username = form\_data['username']

password = form\_data['password']

# Verify credentials

if session\_manager.authenticate\_user(username, password):

# Create session

session\_id = session\_manager.create\_session(username)

# Redirect to stream search page

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

response += b"Location: /search-stream.html\r\n"

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

response += b"Content-Length: 0\r\n"

response += b"Connection: close\r\n\r\n"

client\_socket.sendall(response)

logger.info(f"User {username} successfully logged in")

else:

# Invalid credentials

send\_error\_page(client\_socket, "Invalid username or password", "login.html", username)

def send\_error\_page(client\_socket, error\_message, template\_page, username=""):

*"""*

*Sends page with error message*

*:param client\_socket: client socket*

*:param error\_message: error message text*

*:param template\_page: template page*

*:param username: username for prefilling*

*"""*

try:

with open(os.path.join(SITE\_FOLDER, template\_page), 'r', encoding='utf-8') as file:

content = file.read()

# Add error message

error\_html = f'<div class="message error">{error\_message}</div>'

content = content.replace('<div id="message-container"></div>',

f'<div id="message-container">{error\_html}</div>')

# Prefill username field if it exists

if username:

content = content.replace('name="username" required>',

f'name="username" value="{username}" required>')

content\_bytes = content.encode('utf-8')

headers = f"HTTP/1.1 200 OK\r\n"

headers += f"Content-Type: text/html; charset=utf-8\r\n"

headers += f"Content-Length: {len(content\_bytes)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode('utf-8') + content\_bytes

client\_socket.sendall(response)

except FileNotFoundError:

# If template file not found, send simple message

content = f"<h1>Error</h1><p>{error\_message}</p>".encode('utf-8')

headers = f"HTTP/1.1 200 OK\r\n"

headers += f"Content-Type: text/html; charset=utf-8\r\n"

headers += f"Content-Length: {len(content)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode('utf-8') + content

client\_socket.sendall(response)

def handle\_search\_request(client\_socket, query\_params):

*"""*

*Processes streamer search request*

*:param client\_socket: client socket*

*:param query\_params: query parameters*

*"""*

search\_term = query\_params.get('term', [''])[0].lower()

logger.debug(f"Searching streamers with term: {search\_term}")

# Get list of all registered users from database

conn = sqlite3.connect('user\_database.db')

cursor = conn.cursor()

cursor.execute('SELECT username FROM users')

all\_users = [user[0] for user in cursor.fetchall()]

conn.close()

# Filter users by search term

matched\_users = [user for user in all\_users if search\_term in user.lower()]

# Limit result to three users

matched\_users = matched\_users[:3]

# Form JSON response

response\_data = json.dumps(matched\_users).encode('utf-8')

headers = f"HTTP/1.1 200 OK\r\n"

headers += f"Content-Type: application/json; charset=utf-8\r\n"

headers += f"Content-Length: {len(response\_data)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode('utf-8') + response\_data

client\_socket.sendall(response)

logger.debug(f"Search result sent: {matched\_users}")

def handle\_join\_stream\_request(client\_socket, query\_params):

*"""*

*Processes request to join stream*

*:param client\_socket: client socket*

*:param query\_params: query parameters*

*"""*

streamer\_name = query\_params.get('streamer', [''])[0]

logger.info(f"Request to join stream of user: {streamer\_name}")

# Check if user exists

conn = sqlite3.connect('user\_database.db')

cursor = conn.cursor()

cursor.execute('SELECT username FROM users WHERE username = ?', (streamer\_name,))

user\_exists = cursor.fetchone() is not None

conn.close()

# Check if stream is active

is\_active = stream\_manager.is\_streaming(streamer\_name)

# Form JSON response with clearer status indication

response\_data = json.dumps({

"active": is\_active,

"exists": user\_exists,

"message": "Stream is active" if is\_active else "Stream is not active"

}).encode('utf-8')

headers = f"HTTP/1.1 200 OK\r\n"

headers += f"Content-Type: application/json; charset=utf-8\r\n"

headers += f"Content-Length: {len(response\_data)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode('utf-8') + response\_data

client\_socket.sendall(response)

logger.info(f"Stream status sent for {streamer\_name}: active = {is\_active}, exists = {user\_exists}")

def parse\_query\_string(query\_string):

*"""*

*Parses query string into parameters*

*:param query\_string: query string*

*:return: dictionary of parameters*

*"""*

if not query\_string:

return {}

result = {}

params = query\_string.split('&')

for param in params:

if '=' in param:

key, value = param.split('=', 1)

key = urllib.parse.unquote\_plus(key)

value = urllib.parse.unquote\_plus(value)

if key in result:

if isinstance(result[key], list):

result[key].append(value)

else:

result[key] = [result[key], value]

else:

result[key] = [value]

return result

def parse\_post\_data(request):

*"""*

*Extracts data from POST request*

*:param request: POST request as string*

*:return: dictionary with form data*

*"""*

form\_data = {}

try:

# Find boundary between headers and body

headers\_body\_separator = '\r\n\r\n'

if headers\_body\_separator in request:

headers, body = request.split(headers\_body\_separator, 1)

else:

return {}

# Look for Content-Type and Content-Length

content\_type = None

content\_length = 0

for line in headers.split('\r\n'):

if line.lower().startswith('content-type:'):

content\_type = line.split(':', 1)[1].strip()

elif line.lower().startswith('content-length:'):

try:

content\_length = int(line.split(':', 1)[1].strip())

except ValueError:

pass

# Check that request body has sufficient length

if len(body) < content\_length:

logger.warning(f"Received {len(body)} bytes, expected {content\_length}")

# Process data depending on content type

if content\_type and 'application/json' in content\_type:

# JSON data

try:

form\_data = json.loads(body)

except json.JSONDecodeError:

logger.error("Error parsing JSON data")

elif content\_type and 'application/x-www-form-urlencoded' in content\_type:

# Regular form data

pairs = body.split('&')

for pair in pairs:

if '=' in pair:

key, value = pair.split('=', 1)

form\_data[urllib.parse.unquote\_plus(key)] = urllib.parse.unquote\_plus(value)

return form\_data

except Exception as e:

logger.error(f"Error parsing POST request data: {e}")

return {}

def change\_user\_password(username, current\_password, new\_password):

*"""*

*Changes user password after verifying current password*

*:param username: username*

*:param current\_password: current password*

*:param new\_password: new password*

*:return: True if password changed successfully, False otherwise*

*"""*

try:

# Verify current password

if not session\_manager.authenticate\_user(username, current\_password):

return False, "Incorrect current password"

# Change password

conn = sqlite3.connect('user\_database.db')

cursor = conn.cursor()

hashed\_password = hash\_password(new\_password)

cursor.execute('UPDATE users SET password = ? WHERE username = ?',

(hashed\_password, username))

conn.commit()

conn.close()

return True, "Password changed successfully"

except Exception as e:

logger.error(f"Error changing password: {e}")

return False, "Server error while changing password"

def handle\_change\_password(client\_socket, form\_data, request):

*"""*

*Processes user password change request*

*:param client\_socket: client socket*

*:param form\_data: form data*

*:param request: original HTTP request*

*"""*

logger.info("Processing password change")

logger.debug(f"Received form data: {form\_data}")

headers = request.split('\r\n')[:10]

logger.debug(f"Request headers: {headers}")

# Get cookies from the request

cookie\_header = None

for line in request.split('\r\n'):

if line.lower().startswith('cookie:'):

cookie\_header = line

break

logger.debug(f"Cookie header: {cookie\_header}")

# Get username from session

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

logger.debug(f"Extracted username: {username}")

if not username:

# If username not found, return error

error\_response = {

"success": False,

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

}

send\_json\_response(client\_socket, error\_response, 401)

return

# Check for required data

if 'current-password' in form\_data and 'new-password' in form\_data:

current\_password = form\_data['current-password']

new\_password = form\_data['new-password']

success, message = change\_user\_password(username, current\_password, new\_password)

response\_data = {

"success": success,

"message": message

}

send\_json\_response(client\_socket, response\_data)

else:

# Incomplete form data

error\_response = {

"success": False,

"message": "Current and new passwords are required"

}

send\_json\_response(client\_socket, error\_response, 400)

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

*"""*

*Processes account logout request*

*:param client\_socket: client socket*

*:param request: HTTP request*

*"""*

# Remove session

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

if cookie\_match:

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

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

# Redirect to login page

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

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

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

response += b"Content-Length: 0\r\n"

response += b"Connection: close\r\n\r\n"

client\_socket.sendall(response)

logger.info("User logged out and redirected to login page")

def send\_json\_response(client\_socket, data, status\_code=200):

*"""*

*Sends JSON response to client*

*:param client\_socket: client socket*

*:param data: data to send in JSON format*

*:param status\_code: HTTP status code*

*"""*

status\_text = {

200: "OK",

400: "Bad Request",

401: "Unauthorized",

404: "Not Found",

500: "Internal Server Error"

}.get(status\_code, "OK")

response\_data = json.dumps(data).encode('utf-8')

headers = f"HTTP/1.1 {status\_code} {status\_text}\r\n"

headers += f"Content-Type: application/json; charset=utf-8\r\n"

headers += f"Content-Length: {len(response\_data)}\r\n"

headers += "Connection: close\r\n\r\n"

response = headers.encode('utf-8') + response\_data

client\_socket.sendall(response)

def handle\_client\_connections():

*"""Accept client connections"""*

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

client\_socket.bind((HOST, PORT\_CLIENT))

logger.info(f"Client UDP server started on {HOST}:{PORT\_CLIENT}")

while running:

try:

# Get data from client

data, client\_addr = client\_socket.recvfrom(1024)

# If this is a connection packet

if data == b'CONNECT':

with clients\_lock:

if client\_addr not in clients:

clients.append(client\_addr)

logger.info(f"New client: {client\_addr}")

except Exception as e:

logger.error(f"Error accepting clients: {e}")

async def handle\_websocket(websocket, path=None):

# Default streamer name handling

streamer\_name = ""

try:

# Default path handling

if path is None:

path = websocket.path if hasattr(websocket, 'path') else ""

logger.info(f"WebSocket connection request received with path: {path}")

# Try to extract streamer name from various sources

try:

# 1. Try to get from path parameters

if '?' in path:

query\_string = path.split('?', 1)[1]

query\_params = parse\_query\_string(query\_string)

if 'streamer' in query\_params:

streamer\_name = query\_params['streamer'][0]

logger.debug(f"Found streamer in URL params: {streamer\_name}")

# 2. Try to get from request headers if available

if not streamer\_name and hasattr(websocket, 'request\_headers'):

headers = websocket.request\_headers

if 'streamer' in headers:

streamer\_name = headers['streamer']

logger.debug(f"Found streamer in headers: {streamer\_name}")

# 3. Try to get from path directly if it has a simple format

if not streamer\_name and path.startswith('/ws-proxy/'):

potential\_streamer = path.split('/ws-proxy/', 1)[1]

if potential\_streamer:

streamer\_name = potential\_streamer

logger.debug(f"Found streamer in path: {streamer\_name}")

# 4. Try to get from initial message (for backward compatibility)

if not streamer\_name:

try:

# Wait for a short time for initial message

message = await asyncio.wait\_for(websocket.recv(), timeout=2.0)

data = json.loads(message)

if isinstance(data, dict) and 'action' in data and data['action'] == 'join':

streamer\_name = data.get('streamer', '')

logger.debug(f"Found streamer in initial message: {streamer\_name}")

except (asyncio.TimeoutError, json.JSONDecodeError) as e:

logger.debug(f"No initial message received or error: {e}")

except Exception as e:

logger.error(f"Error extracting streamer name: {e}")

logger.info(f"WebSocket connection established for stream: {streamer\_name}")

# Add client to the list for this streamer

if streamer\_name not in ws\_clients:

ws\_clients[streamer\_name] = []

ws\_clients[streamer\_name].append(websocket)

# Send welcome message and audio initialization signal

await websocket.send(json.dumps({"type": "info", "message": f"Connected to stream of {streamer\_name}"}))

await websocket.send(json.dumps({"type": "audio\_init", "streamer": streamer\_name}))

# Keep connection open

while True:

try:

# Wait for message from client (just to keep connection alive)

message = await websocket.recv()

# Process messages from client

try:

data = json.loads(message)

if isinstance(data, dict) and data.get('type') == 'request\_audio\_reinit':

# Client requests audio reinitialization

await websocket.send(json.dumps({"type": "audio\_init", "streamer": streamer\_name}))

except json.JSONDecodeError as e:

logger.warning(f"Invalid JSON received from client: {e}")

except Exception as e:

logger.error(f"Error processing client message: {e}")

except websockets.exceptions.ConnectionClosed:

break

except Exception as e:

logger.error(f"WebSocket error: {e}")

finally:

# Remove client on disconnect

if streamer\_name in ws\_clients and websocket in ws\_clients[streamer\_name]:

ws\_clients[streamer\_name].remove(websocket)

if not ws\_clients[streamer\_name]:

del ws\_clients[streamer\_name]

def broadcast\_stream():

*"""Broadcast media stream to clients via UDP and WebSocket"""*

udp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

while running:

try:

# Get data from broadcast queue

try:

data\_type, payload, timestamp = broadcast\_queue.get(timeout=0.1)

except queue.Empty:

continue

# Don't send data if there's no active streamer

if current\_streamer is None:

continue

# UDP broadcast (fragmentation)

if clients: # If there are UDP clients

fragments = fragment\_data(data\_type, payload)

with clients\_lock:

for client in clients:

try:

for fragment in fragments:

udp\_socket.sendto(fragment, client)

except Exception as e:

logger.error(f"Error sending to UDP client {client}: {e}")

# WebSocket broadcast

if ws\_clients: # If there are WebSocket clients

try:

if data\_type == "V": # Video

# Convert frame for WebSocket (JPEG + base64)

ws\_data = prepare\_video\_for\_ws(payload)

elif data\_type == "A": # Audio

# Convert audio for WebSocket (base64)

ws\_data = prepare\_audio\_for\_ws(payload)

else:

continue

# Send to all connected WebSocket clients

for streamer, clients\_list in list(ws\_clients.items()):

for client in clients\_list[:]:

asyncio.run\_coroutine\_threadsafe(send\_ws\_message(client, ws\_data), loop)

except Exception as e:

logger.error(f"WebSocket broadcast error: {e}")

except Exception as e:

logger.error(f"Error in broadcast process: {e}")

def fragment\_data(data\_type, payload):

*"""*

*Fragment data for UDP transmission*

*:param data\_type: data type (V - video, A - audio)*

*:param payload: payload*

*:return: list of fragments*

*"""*

max\_payload = 1024

fragments = []

total\_fragments = (len(payload) + max\_payload - 1) // max\_payload

for i in range(total\_fragments):

start = i \* max\_payload

end = min(start + max\_payload, len(payload))

data = payload[start:end]

# Form fragment header

header = struct.pack(

"!5I",

ord(data\_type[0]), # Data type as byte

0, # Sequence number

total\_fragments, # Total number of fragments

i, # Current fragment number

0 # Reserved field

)

fragments.append(header + data)

return fragments

def prepare\_video\_for\_ws(payload):

*"""*

*Prepare video data for sending via WebSocket*

*:param payload: raw video data*

*:return: JSON string for sending via WebSocket*

*"""*

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

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

if img is not None:

ret, jpeg = cv2.imencode('.jpg', img, [cv2.IMWRITE\_JPEG\_QUALITY, 80])

jpeg\_bytes = jpeg.tobytes()

base64\_frame = base64.b64encode(jpeg\_bytes).decode('utf-8')

return json.dumps({

"type": "video",

"data": base64\_frame

})

return None

def prepare\_audio\_for\_ws(payload):

*"""*

*Prepare audio data for sending via WebSocket*

*:param payload: raw audio data*

*:return: JSON string for sending via WebSocket*

*"""*

base64\_audio = base64.b64encode(payload).decode('utf-8')

return json.dumps({

"type": "audio",

"data": base64\_audio

})

# Helper function for asynchronous WebSocket message sending

async def send\_ws\_message(client, message):

try:

await client.send(message)

return True

except websockets.exceptions.ConnectionClosed:

# Client disconnected

logger.debug("Client connection closed")

return False

except Exception as e:

logger.error(f"Error sending WebSocket message: {e}")

return False

# broadcast function to send message to all clients for a streamer

async def broadcast\_to\_streamer\_clients(streamer\_name, message):

*"""*

*Broadcast a message to all clients connected to a specific streamer*

*:param streamer\_name: Name of the streamer*

*:param message: Message to broadcast (already formatted JSON string)*

*"""*

if streamer\_name not in ws\_clients:

return

# Track clients to remove (avoid modifying the list while iterating)

to\_remove = []

# Send to all connected clients

for client in ws\_clients[streamer\_name]:

success = await send\_ws\_message(client, message)

if not success:

to\_remove.append(client)

# Remove disconnected clients

for client in to\_remove:

if client in ws\_clients[streamer\_name]:

ws\_clients[streamer\_name].remove(client)

# Clean up empty lists

if streamer\_name in ws\_clients and not ws\_clients[streamer\_name]:

del ws\_clients[streamer\_name]

# Define a flexible WebSocket handler

async def ws\_handler(websocket, path=None):

try:

# Extract path information from the websocket object if available

if path is None and hasattr(websocket, 'path'):

path = websocket.path

elif path is None:

path = "" # Default empty path

await handle\_websocket(websocket, path)

except Exception as e:

logger.error(f"Error in WebSocket handler: {e}")

async def run\_websocket\_server():

global loop

loop = asyncio.get\_event\_loop()

stop = asyncio.Future()

logger.info(f"Starting WebSocket server on port {ws\_port}")

# Create SSL context for secure WebSocket connections

ssl\_context = None

if os.path.exists(SSL\_CERT\_FILE) and os.path.exists(SSL\_KEY\_FILE):

try:

ssl\_context = ssl.SSLContext(ssl.PROTOCOL\_TLS\_SERVER)

ssl\_context.load\_cert\_chain(certfile=SSL\_CERT\_FILE, keyfile=SSL\_KEY\_FILE)

logger.info(f"Using SSL for WebSocket server with cert: {SSL\_CERT\_FILE}")

except Exception as e:

logger.error(f"Error loading SSL certificates for WebSocket: {e}")

logger.warning("WebSocket server will run without SSL")

# Set proper SSL verification settings

if ssl\_context:

ssl\_context.check\_hostname = False

ssl\_context.verify\_mode = ssl.CERT\_NONE

# Start WebSocket server on the dedicated port

server = await websockets.serve(

ws\_handler,

HOST,

ws\_port,

ssl=ssl\_context,

ping\_interval=30,

ping\_timeout=60

)

# Start an additional non-SSL WebSocket server if SSL is enabled

# This ensures we have both secure and non-secure WebSocket endpoints

if ssl\_context:

non\_ssl\_ws\_port = ws\_port + 1 # Use next port for non-SSL

non\_ssl\_server = await websockets.serve(

ws\_handler,

HOST,

non\_ssl\_ws\_port,

ssl=None, # No SSL for this server

ping\_interval=30,

ping\_timeout=60

)

logger.info(f"Non-SSL WebSocket server running on ws://{HOST}:{non\_ssl\_ws\_port}")

logger.info(f"WebSocket server running on {'wss' if ssl\_context else 'ws'}://{HOST}:{ws\_port}")

await stop

def start\_websocket\_server():

asyncio.set\_event\_loop(asyncio.new\_event\_loop())

wloop = asyncio.get\_event\_loop()

wloop.run\_until\_complete(run\_websocket\_server())

wloop.run\_forever()

def is\_safe\_path(base\_dir, requested\_path):

*"""*

*Checks if the requested path is inside the base directory*

*:param base\_dir: base directory (site root)*

*:param requested\_path: requested path*

*:return: True if path is safe, False otherwise*

*"""*

# Normalize paths for correct comparison

base\_dir = os.path.abspath(base\_dir)

requested\_path = os.path.abspath(os.path.join(base\_dir, str(requested\_path)))

# Check that requested path starts with base directory

return requested\_path.startswith(base\_dir)

def create\_self\_signed\_cert():

*"""*

*Creates a self-signed SSL certificate if it doesn't exist*

*"""*

if os.path.exists(SSL\_CERT\_FILE) and os.path.exists(SSL\_KEY\_FILE):

logger.info(f"Using existing SSL certificate: {SSL\_CERT\_FILE}")

return

logger.info("Creating self-signed SSL certificate...")

# Command to create self-signed certificate

cmd = f'openssl req -x509 -newkey rsa:2048 -keyout {SSL\_KEY\_FILE} -out {SSL\_CERT\_FILE} ' \

f'-days 365 -nodes -subj "/CN=localhost"'

try:

import subprocess

subprocess.run(cmd, shell=True, check=True)

logger.info(f"SSL certificate created: {SSL\_CERT\_FILE}")

except Exception as e:

logger.error(f"Error creating SSL certificate: {e}")

logger.warning("Please create SSL certificate manually using OpenSSL")

logger.warning(f"Example: {cmd}")

def main():

# Initialize protocols and database at startup

global stream\_manager, session\_manager, running, ws\_port, stream\_protocol

# Initialize protocols

stream\_protocol = StreamProtocol()

init\_database()

# Create SSL certificate if needed

create\_self\_signed\_cert()

# Create HTTP server

http\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Create HTTPS server

https\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Wrap socket in SSL context

ssl\_context = ssl.create\_default\_context(ssl.Purpose.CLIENT\_AUTH)

try:

ssl\_context.load\_cert\_chain(certfile=SSL\_CERT\_FILE, keyfile=SSL\_KEY\_FILE)

https\_server\_socket = ssl\_context.wrap\_socket(

https\_server\_socket, server\_side=True

)

https\_enabled = True

except Exception as e:

logger.error(f"Error setting up HTTPS: {e}")

logger.warning("Running with HTTP only")

https\_enabled = False

try:

# Start TCP authentication server thread

tcp\_auth\_thread = threading.Thread(target=auth\_server\_thread)

tcp\_auth\_thread.daemon = True

tcp\_auth\_thread.start()

# Start remaining threads

streamer\_thread = threading.Thread(target=streamer\_listener)

streamer\_thread.daemon = True

streamer\_thread.start()

client\_thread = threading.Thread(target=handle\_client\_connections)

client\_thread.daemon = True

client\_thread.start()

broadcast\_thread = threading.Thread(target=broadcast\_stream)

broadcast\_thread.daemon = True

broadcast\_thread.start()

# Define WebSocket port and non-SSL port

ws\_port = 8765

ws\_non\_ssl\_port = 8766 # This port will be used for non-SSL WebSockets

# Update global variable to be accessible elsewhere

import builtins

setattr(builtins, 'ws\_non\_ssl\_port', ws\_non\_ssl\_port)

ws\_thread = threading.Thread(target=start\_websocket\_server)

ws\_thread.daemon = True

ws\_thread.start()

# Configure HTTP server

http\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

http\_server\_socket.bind((HOST, PORT))

http\_server\_socket.listen(5)

logger.info(f"HTTP server started on port {PORT}")

# Configure HTTPS server if enabled

if https\_enabled:

https\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

https\_server\_socket.bind((HOST, HTTPS\_PORT))

https\_server\_socket.listen(5)

logger.info(f"HTTPS server started on port {HTTPS\_PORT}")

logger.info(f"Open browser and go to https://localhost/register.html")

else:

logger.info(f"Open browser and go to http://localhost/register.html")

logger.info(f"WebSocket server runs on ports: secure={ws\_port}, non-secure={ws\_non\_ssl\_port}")

logger.info(f"TCP Authentication server running on port {TCP\_AUTH\_PORT}")

logger.info(f"UDP Streaming server running on port {PORT\_UDP}")

# Create list for tracking sockets

sockets\_to\_monitor = [http\_server\_socket]

if https\_enabled:

sockets\_to\_monitor.append(https\_server\_socket)

while running:

# Use select to monitor multiple sockets

import select

readable, \_, exceptional = select.select(sockets\_to\_monitor, [], sockets\_to\_monitor, 1.0)

for s in readable:

try:

client\_socket, addr = s.accept()

# Separate thread for handling request

client\_thread = threading.Thread(

target=handle\_client\_request,

args=(client\_socket, addr, s == https\_server\_socket)

)

client\_thread.daemon = True

client\_thread.start()

except Exception as e:

logger.error(f"Error accepting connection: {e}")

for s in exceptional:

logger.error(f"Socket {s} had an exception")

sockets\_to\_monitor.remove(s)

s.close()

# If this was one of the server sockets, try to recreate it

if s == http\_server\_socket:

http\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

http\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

http\_server\_socket.bind((HOST, PORT))

http\_server\_socket.listen(5)

sockets\_to\_monitor.append(http\_server\_socket)

logger.info(f"HTTP server restarted on port {PORT}")

elif https\_enabled and s == https\_server\_socket:

https\_server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

https\_server\_socket = ssl\_context.wrap\_socket(

https\_server\_socket, server\_side=True

)

https\_server\_socket.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

https\_server\_socket.bind((HOST, HTTPS\_PORT))

https\_server\_socket.listen(5)

sockets\_to\_monitor.append(https\_server\_socket)

logger.info(f"HTTPS server restarted on port {HTTPS\_PORT}")

except socket.error as err:

logger.error('Server socket error: ' + str(err))

except KeyboardInterrupt:

logger.info("Server shutting down...")

running = False

finally:

http\_server\_socket.close()

if https\_enabled:

https\_server\_socket.close()

def handle\_client\_request(client\_socket, addr, is\_https):

*"""*

*Processes client requests for both HTTP and HTTPS connections*

*:param client\_socket: client socket*

*:param addr: client address*

*:param is\_https: flag indicating this is an HTTPS connection*

*"""*

protocol = "HTTPS" if is\_https else "HTTP"

try:

client\_socket.settimeout(SOCKET\_TIMEOUT)

logger.info(f"{protocol} connection established with {addr}")

raw\_request = receive\_all(client\_socket)

if raw\_request:

request = raw\_request.decode('utf-8', errors='ignore')

request\_lines = request.split('\r\n')

# Check for empty request

if not request\_lines or not request\_lines[0]:

logger.warning("Received empty request")

response = request\_error

client\_socket.sendall(response)

return

request\_split = request\_lines[0].split()

logger.info(f"Received {protocol} request: {request\_lines[0]}")

# Check if this is a WebSocket upgrade request

if len(request\_lines) > 1:

headers = {}

for line in request\_lines[1:]:

if line and ': ' in line:

key, value = line.split(': ', 1)

headers[key.lower()] = value

if 'upgrade' in headers and headers['upgrade'].lower() == 'websocket':

logger.info(f"WebSocket upgrade request detected via {protocol}")

handle\_websocket\_proxy(client\_socket, request, is\_https)

return

# Check request format validity

if len(request\_split) < 3:

logger.warning(f"Invalid request format: {request\_lines[0]}")

response = request\_error

client\_socket.sendall(response)

return

# Process GET requests

if validating\_get\_request(request\_split):

if len(request\_split) > 1:

request\_path = request\_split[1]

# Check for query parameters

path\_parts = request\_path.split('?', 1)

filename = path\_parts[0][1:] # Remove leading slash

query\_string = path\_parts[1] if len(path\_parts) > 1 else ""

# Handle WebSocket proxy path

if filename == 'ws-proxy':

logger.info("WebSocket proxy path detected")

handle\_websocket\_proxy(client\_socket, request, is\_https)

return

if filename == '':

filename = 'register.html' # Start with registration

# Fix link in register.html

if filename == 'index.html':

filename = 'login.html'

# Process special API requests

if filename == 'search':

query\_params = parse\_query\_string(query\_string)

handle\_search\_request(client\_socket, query\_params)

return

elif filename == 'join-stream':

query\_params = parse\_query\_string(query\_string)

handle\_join\_stream\_request(client\_socket, query\_params)

return

elif filename == 'logout':

handle\_logout(client\_socket, request)

return

# Check if authorization is required for file access

if filename in protected\_pages:

# Get username from session

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

if not username:

# If user is not authorized, redirect to login page

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

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

response += b"Content-Length: 0\r\n"

response += b"Connection: close\r\n\r\n"

client\_socket.sendall(response)

logger.info(f"Redirecting to login: unauthorized access to {filename}")

return

filepath = os.path.join(SITE\_FOLDER, filename)

logger.debug(f"Requested file: {filepath}")

if specific(filename):

response = searching\_url(filename)

client\_socket.sendall(response)

elif is\_safe\_path(SITE\_FOLDER, filename):

serve\_file(client\_socket, filepath)

else:

# Request to unsafe path - send 403 Forbidden error

response = b"HTTP/1.1 403 Forbidden\r\n\r\n<h1>403 Forbidden</h1>"

client\_socket.sendall(response)

logger.warning(f"Forbidden access attempt to unsafe path: {filename}")

# Process POST requests

elif validating\_post\_request(request\_split):

path = request\_split[1]

logger.info(f"Received {protocol} POST request for path: {path}")

form\_data = parse\_post\_data(request)

logger.debug(f"Extracted form data: {form\_data}")

if path == '/register':

handle\_registration(client\_socket, form\_data)

elif path == '/login':

handle\_login(client\_socket, form\_data)

elif path == '/change-password':

handle\_change\_password(client\_socket, form\_data, request)

else:

logger.warning(f"Unknown POST request: {path}")

response = request\_error

client\_socket.sendall(response)

else:

logger.warning(f"Unsupported request type: {request\_split[0]}")

response = request\_error

client\_socket.sendall(response)

else:

logger.warning("Received empty request (no data)")

response = request\_error

client\_socket.sendall(response)

except socket.error as err:

logger.error(f'Client socket error ({protocol}): {str(err)}')

except Exception as e:

logger.error(f"Unhandled exception in {protocol} handler: {e}")

# Send 500 Internal Server Error for unexpected exceptions

error\_response = b"HTTP/1.1 500 Internal Server Error\r\nContent-Type: text/html\r\nContent-Length: 35\r\nConnection: close\r\n\r\n<h1>500 Internal Server Error</h1>"

try:

client\_socket.sendall(error\_response)

except socket.error as err:

logger.error(f'Client socket error during error response ({protocol}): {str(err)}')

finally:

client\_socket.close()

def handle\_websocket\_proxy(client\_socket, request, is\_https):

*"""*

*Handles WebSocket upgrade requests coming through the HTTP/HTTPS server*

*by implementing the WebSocket protocol directly*

*"""*

protocol = "HTTPS" if is\_https else "HTTP"

logger.info(f"WebSocket upgrade request detected via {protocol}")

# Parse the request to get headers

request\_lines = request.split('\r\n')

headers = {}

for line in request\_lines[1:]: # Skip the first line which is the request line

if line and ': ' in line:

key, value = line.split(': ', 1)

headers[key.lower()] = value

# Extract query parameters to get streamer name

request\_path = request\_lines[0].split()[1]

streamer\_name = ""

if '?' in request\_path:

query\_string = request\_path.split('?', 1)[1]

query\_params = parse\_query\_string(query\_string)

if 'streamer' in query\_params:

streamer\_name = query\_params['streamer'][0]

logger.info(f"WebSocket proxy request for streamer: {streamer\_name} via {protocol}")

# Verify this is a valid WebSocket request

if 'upgrade' not in headers or headers['upgrade'].lower() != 'websocket':

logger.warning("Not a valid WebSocket upgrade request")

response = b"HTTP/1.1 400 Bad Request\r\n\r\n"

client\_socket.sendall(response)

return

if 'sec-websocket-key' not in headers:

logger.warning("WebSocket key missing")

response = b"HTTP/1.1 400 Bad Request\r\n\r\n"

client\_socket.sendall(response)

return

# Create the WebSocket accept key

ws\_key = headers['sec-websocket-key']

ws\_accept = base64.b64encode(hashlib.sha1(

f"{ws\_key}258EAFA5-E914-47DA-95CA-C5AB0DC85B11".encode()

).digest()).decode()

# Send the WebSocket handshake response

response = (

b"HTTP/1.1 101 Switching Protocols\r\n"

b"Upgrade: websocket\r\n"

b"Connection: Upgrade\r\n"

b"Sec-WebSocket-Accept: " + ws\_accept.encode() + b"\r\n\r\n"

)

client\_socket.sendall(response)

logger.info(f"WebSocket handshake completed via {protocol}")

# Now the connection is a WebSocket connection

handle\_websocket\_connection(client\_socket, streamer\_name)

def handle\_websocket\_connection(client\_socket, streamer\_name):

*"""*

*Handles an established WebSocket connection*

*"""*

global ws\_clients

logger.info(f"WebSocket connection established for streamer: {streamer\_name}")

# Register this client

if streamer\_name not in ws\_clients:

ws\_clients[streamer\_name] = []

# Use a custom class to track this connection

class WSClient:

def \_\_init\_\_(self, socket):

self.socket = socket

self.closed = False

async def send(self, message):

if self.closed:

return False

try:

# Format message according to WebSocket protocol

message\_bytes = message.encode() if isinstance(message, str) else message

# Create WebSocket frame

length = len(message\_bytes)

frame = bytearray()

# First byte: FIN bit (1) + RSV bits (000) + Opcode (0001 for text)

frame.append(0x81) # 10000001

# Second byte onwards: MASK bit (0) + Payload length

if length < 126:

frame.append(length)

elif length < 65536:

frame.append(126)

frame.append((length >> 8) & 0xFF)

frame.append(length & 0xFF)

else:

frame.append(127)

for k in range(7, -1, -1):

frame.append((length >> (k \* 8)) & 0xFF)

# Add payload

frame.extend(message\_bytes)

# Send frame

self.socket.sendall(frame)

return True

except Exception as err:

logger.error(f"Error sending WebSocket message: {err}")

self.closed = True

return False

# Create client object and add to clients list

client = WSClient(client\_socket)

ws\_clients[streamer\_name].append(client)

# Send welcome message

asyncio.run\_coroutine\_threadsafe(

client.send(json.dumps({"type": "info", "message": f"Connected to stream of {streamer\_name}"})),

loop

)

# Send audio initialization signal

asyncio.run\_coroutine\_threadsafe(

client.send(json.dumps({"type": "audio\_init", "streamer": streamer\_name})),

loop

)

# Now handle incoming WebSocket frames

try:

while True:

# Read WebSocket frame header

try:

header = client\_socket.recv(2)

if not header or len(header) < 2:

break

# Parse header

fin = (header[0] & 0x80) != 0

opcode = header[0] & 0x0F

masked = (header[1] & 0x80) != 0

payload\_length = header[1] & 0x7F

# Get extended payload length if needed

if payload\_length == 126:

payload\_length = struct.unpack(">H", client\_socket.recv(2))[0]

elif payload\_length == 127:

payload\_length = struct.unpack(">Q", client\_socket.recv(8))[0]

# Get masking key if frame is masked

masking\_key = client\_socket.recv(4) if masked else None

# Read payload

payload = client\_socket.recv(payload\_length)

# Unmask payload if needed

if masked and masking\_key:

unmasked = bytearray(payload\_length)

for i in range(payload\_length):

unmasked[i] = payload[i] ^ masking\_key[i % 4]

payload = bytes(unmasked)

# Process based on opcode

if opcode == 0x1: # Text frame

try:

message = payload.decode('utf-8')

logger.debug(f"Received text message: {message}")

# Process message if needed

try:

data = json.loads(message)

if isinstance(data, dict) and data.get('action') == 'ping':

# Respond to ping with pong

asyncio.run\_coroutine\_threadsafe(

client.send(json.dumps({"type": "pong"})),

loop

)

except json.JSONDecodeError:

pass

except UnicodeDecodeError:

logger.error("Error decoding message")

elif opcode == 0x8: # Close frame

logger.info("Received close frame")

break

elif opcode == 0x9: # Ping frame

# Respond with pong

pong\_frame = bytearray([0x8A, len(payload)])

pong\_frame.extend(payload)

client\_socket.sendall(pong\_frame)

elif opcode == 0xA: # Pong frame

# Just acknowledge

pass

except socket.timeout:

# Check if client is still connected

try:

# Send a ping frame

ping\_frame = bytearray([0x89, 0x00])

client\_socket.sendall(ping\_frame)

except:

break

except Exception as e:

logger.error(f"Error reading WebSocket frame: {e}")

break

except Exception as e:

logger.error(f"WebSocket connection error: {e}")

finally:

# Clean up

if streamer\_name in ws\_clients and client in ws\_clients[streamer\_name]:

ws\_clients[streamer\_name].remove(client)

if not ws\_clients[streamer\_name]:

del ws\_clients[streamer\_name]

try:

client\_socket.close()

except:

pass

logger.info(f"WebSocket connection closed for streamer: {streamer\_name}")

if \_\_name\_\_ == "\_\_main\_\_":

"""

checking function situations and launching the main

"""

# basic tests

assert specific('error')

assert specific('forbidden')

assert specific('moved')

assert not specific('abc')

assert searching\_url('error')

assert not searching\_url('abc')

assert choosing\_type('test.jpg') == types['jpg']

assert choosing\_type('test.html') == types['tml']

assert validating\_get\_request(['GET', '/', 'HTTP/1.1', 'headers'])

assert not validating\_get\_request(['POST', '/', 'HTTP/1.1', 'headers'])

assert validating\_post\_request(['POST', '/', 'HTTP/1.1', 'headers'])

main()

streamer

*"""*

*Author: Oleg Shkolnik*

*Description: application for streamer*

*Date: 3/06/25*

*"""*

import socket

import cv2

import sounddevice as sd

import numpy as np

import threading

import time

import tkinter as tk

from tkinter import ttk, messagebox

import PIL.Image, PIL.ImageTk

from protocol import StreamProtocol, AuthProtocol

import ssl

import platform

import logging

import os

import datetime

SERVER\_IP = "127.0.0.1"

SERVER\_AUTH\_PORT = 5050

CAMERA\_WIDTH = 640

CAMERA\_HEIGHT = 480

AUDIO\_RATE = 44100

AUDIO\_CHANNELS = 1

AUDIO\_CHUNK = 1024

AUDIO\_DTYPE = np.int16

stream\_protocol = StreamProtocol()

auth\_protocol = AuthProtocol()

def setup\_logging():

if not os.path.exists('logs'):

os.makedirs('logs')

log\_format = '%(asctime)s - %(name)s - %(levelname)s - %(funcName)s:%(lineno)d - %(message)s'

file\_handler = logging.FileHandler(

f'logs/streamer\_{datetime.datetime.now().strftime("%Y%m%d")}.log'

)

file\_handler.setLevel(logging.DEBUG)

file\_handler.setFormatter(logging.Formatter(log\_format))

console\_handler = logging.StreamHandler()

console\_handler.setLevel(logging.INFO)

console\_handler.setFormatter(logging.Formatter(log\_format))

log = logging.getLogger('StreamerApp')

log.setLevel(logging.DEBUG)

log.addHandler(file\_handler)

log.addHandler(console\_handler)

return log

logger = setup\_logging()

class StreamerApp:

def \_\_init\_\_(self, root):

logger.info("Initializing StreamerApp")

self.root = root

self.root.title("Video Streamer")

self.root.geometry("800x600")

self.root.protocol("WM\_DELETE\_WINDOW", self.on\_closing)

self.running = False

self.authenticated = False

self.username = None

self.udp\_port = None

self.client\_socket = None

self.cap = None

self.create\_login\_frame()

self.create\_stream\_frame()

self.show\_login\_frame()

logger.info("StreamerApp initialization completed")

def create\_login\_frame(self):

logger.debug("Creating login frame UI components")

self.login\_frame = ttk.Frame(self.root, padding=20)

ttk.Label(self.login\_frame, text="Server Settings", font=("Arial", 14, "bold")).grid(row=0, column=0,

columnspan=2, pady=10,

sticky='w')

ttk.Label(self.login\_frame, text="Server IP:").grid(row=1, column=0, sticky='w', pady=5)

self.server\_ip\_var = tk.StringVar(value=SERVER\_IP)

ttk.Entry(self.login\_frame, textvariable=self.server\_ip\_var, width=30).grid(row=1, column=1, sticky='w', padx=5)

ttk.Label(self.login\_frame, text="Auth Port:").grid(row=2, column=0, sticky='w', pady=5)

self.auth\_port\_var = tk.StringVar(value=str(SERVER\_AUTH\_PORT))

ttk.Entry(self.login\_frame, textvariable=self.auth\_port\_var, width=30).grid(row=2, column=1, sticky='w', padx=5)

ttk.Label(self.login\_frame, text="Login", font=("Arial", 14, "bold")).grid(row=3, column=0, columnspan=2,

pady=(20, 10), sticky='w')

ttk.Label(self.login\_frame, text="Username:").grid(row=4, column=0, sticky='w', pady=5)

self.username\_var = tk.StringVar()

ttk.Entry(self.login\_frame, textvariable=self.username\_var, width=30).grid(row=4, column=1, sticky='w', padx=5)

ttk.Label(self.login\_frame, text="Password:").grid(row=5, column=0, sticky='w', pady=5)

self.password\_var = tk.StringVar()

ttk.Entry(self.login\_frame, textvariable=self.password\_var, width=30, show="\*").grid(row=5, column=1,

sticky='w', padx=5)

self.status\_var = tk.StringVar(value="Enter your credentials to start streaming")

ttk.Label(self.login\_frame, textvariable=self.status\_var, foreground='blue').grid(row=6, column=0, columnspan=2,

pady=10)

self.login\_button = ttk.Button(self.login\_frame, text="Login", command=self.authenticate)

self.login\_button.grid(row=7, column=0, columnspan=2, pady=10)

logger.debug("Login frame created successfully")

def create\_stream\_frame(self):

logger.debug("Creating streaming frame UI components")

self.stream\_frame = ttk.Frame(self.root, padding=10)

self.video\_frame = ttk.Frame(self.stream\_frame)

self.video\_frame.pack(fill=tk.BOTH, expand=True, pady=10)

self.canvas = tk.Canvas(self.video\_frame, width=CAMERA\_WIDTH, height=CAMERA\_HEIGHT, bg="black")

self.canvas.pack()

control\_frame = ttk.Frame(self.stream\_frame)

control\_frame.pack(fill=tk.X, pady=10)

self.stream\_status\_var = tk.StringVar(value="Connected as: Not logged in")

ttk.Label(control\_frame, textvariable=self.stream\_status\_var).pack(side=tk.LEFT, padx=10)

self.stop\_button = ttk.Button(control\_frame, text="Disconnect", command=self.stop\_streaming)

self.stop\_button.pack(side=tk.RIGHT, padx=10)

logger.debug("Streaming frame created successfully")

def show\_login\_frame(self):

logger.debug("Switching to login frame")

self.stream\_frame.pack\_forget()

self.login\_frame.pack(fill=tk.BOTH, expand=True)

def show\_stream\_frame(self):

logger.debug("Switching to streaming frame")

self.login\_frame.pack\_forget()

self.stream\_frame.pack(fill=tk.BOTH, expand=True)

def authenticate(self):

logger.info("Starting authentication process")

server\_ip = self.server\_ip\_var.get()

auth\_port = int(self.auth\_port\_var.get())

username = self.username\_var.get()

password = self.password\_var.get()

logger.debug(f"Authentication attempt for user: {username} to server: {server\_ip}:{auth\_port}")

if not username or not password:

logger.warning("Authentication failed: empty username or password")

messagebox.showerror("Error", "Username and password cannot be empty")

return

self.login\_button.config(state=tk.DISABLED)

self.status\_var.set("Authenticating...")

self.root.update()

auth\_thread = threading.Thread(target=self.authentication\_thread,

args=(server\_ip, auth\_port),

daemon=True)

auth\_thread.start()

logger.debug("Authentication thread started")

def authentication\_thread(self, server\_ip, auth\_port):

logger.debug("Authentication thread executing")

try:

auth\_result, username, udp\_port = self.gui\_authenticate(server\_ip, auth\_port)

if auth\_result:

logger.info(f"Authentication successful for user: {username}, UDP port: {udp\_port}")

self.authenticated = True

self.username = username

self.udp\_port = udp\_port

self.root.after(0, lambda: self.start\_streaming())

else:

logger.warning("Authentication failed")

self.root.after(0, lambda: self.status\_var.set("Authentication failed. Try again."))

self.root.after(0, lambda: self.login\_button.config(state=tk.NORMAL))

except Exception as e:

logger.error(f"Authentication thread error: {str(e)}", exc\_info=True)

error\_msg = f"Authentication error: {str(e)}"

self.root.after(0, lambda: messagebox.showerror("Error", error\_msg))

self.root.after(0, lambda: self.login\_button.config(state=tk.NORMAL))

def gui\_authenticate(self, server\_host, server\_port):

logger.debug(f"Starting GUI authentication to {server\_host}:{server\_port}")

tcp\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

try:

ssl\_context = ssl.create\_default\_context()

ssl\_context.check\_hostname = False

ssl\_context.verify\_mode = ssl.CERT\_NONE

tcp\_socket = ssl\_context.wrap\_socket(tcp\_socket, server\_hostname=server\_host)

logger.debug("SSL context created and socket wrapped")

except Exception as e:

logger.warning(f"SSL setup error: {e}. Continuing without SSL")

self.root.after(0, lambda: self.status\_var.set(f"SSL setup error: {e}. Trying without SSL..."))

try:

tcp\_socket.connect((server\_host, server\_port))

logger.info(f"Connected to authentication server at {server\_host}:{server\_port}")

response = 'f'

flag = False

while response != 't':

if response == 'fr':

logger.warning("Username is already taken")

self.root.after(0, lambda: self.status\_var.set("This username is already taken"))

elif flag:

logger.warning("Username or password is incorrect")

self.root.after(0, lambda: self.status\_var.set("Username or password is incorrect"))

flag = True

username = self.username\_var.get()

password = self.password\_var.get()

user\_data = username + ',' + password

logger.debug("Sending credentials to server")

auth\_protocol.send\_message(tcp\_socket, user\_data)

response\_data = auth\_protocol.receive\_message(tcp\_socket)

if not response\_data:

logger.error("Connection closed by server during authentication")

self.root.after(0, lambda: self.status\_var.set("Connection closed by server"))

return False, None, None

response = response\_data.decode()

logger.debug(f"Received authentication response: {response}")

if response == 'f':

logger.info("Account not found, asking user about registration")

answer = messagebox.askyesno("Registration", "Account not found. Do you want to register?")

if answer:

logger.info("User chose to register new account")

auth\_protocol.send\_message(tcp\_socket, 'r')

response\_data = auth\_protocol.receive\_message(tcp\_socket)

if not response\_data:

logger.error("Connection closed during registration")

self.root.after(0, lambda: self.status\_var.set("Connection closed during registration"))

return False, None, None

response = response\_data.decode()

logger.debug(f"Registration response: {response}")

else:

logger.info("User declined registration")

return False, None, None

if response == 't':

logger.info("Authentication successful, receiving UDP port")

port\_data = auth\_protocol.receive\_message(tcp\_socket)

if port\_data:

udp\_port = int(port\_data.decode())

logger.info(f"Received UDP streaming port: {udp\_port}")

self.root.after(0, lambda: self.status\_var.set(

f"Authentication successful. Streaming port: {udp\_port}"))

return True, username, udp\_port

return False, None, None

except Exception as e:

logger.error(f"Authentication error: {e}", exc\_info=True)

self.root.after(0, lambda: self.status\_var.set(f"Authentication error: {e}"))

return False, None, None

finally:

tcp\_socket.close()

logger.debug("Authentication socket closed")

def start\_streaming(self):

if not self.authenticated:

logger.error("Attempted to start streaming without authentication")

messagebox.showerror("Error", "You need to authenticate first")

return

try:

logger.info("Starting streaming process")

self.stream\_status\_var.set(f"Connected as: {self.username} | Port: {self.udp\_port}")

self.show\_stream\_frame()

self.client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

logger.debug("UDP socket created for streaming")

if platform.system() == "Windows":

logger.info("Windows detected, using DirectShow backend for camera")

self.cap = cv2.VideoCapture(0, cv2.CAP\_DSHOW)

else:

logger.info("Non-Windows system detected, using default backend for camera")

self.cap = cv2.VideoCapture(0)

if not self.cap.isOpened():

logger.error("Camera failed to open")

messagebox.showerror("Error", "Failed to open camera")

return

logger.info("Camera opened successfully")

logger.debug("Setting camera properties")

self.cap.set(cv2.CAP\_PROP\_FRAME\_WIDTH, CAMERA\_WIDTH)

self.cap.set(cv2.CAP\_PROP\_FRAME\_HEIGHT, CAMERA\_HEIGHT)

self.cap.set(cv2.CAP\_PROP\_FPS, 30)

self.cap.set(cv2.CAP\_PROP\_BUFFERSIZE, 1)

actual\_width = int(self.cap.get(cv2.CAP\_PROP\_FRAME\_WIDTH))

actual\_height = int(self.cap.get(cv2.CAP\_PROP\_FRAME\_HEIGHT))

actual\_fps = int(self.cap.get(cv2.CAP\_PROP\_FPS))

logger.info(f"Camera configured: {actual\_width}x{actual\_height} @ {actual\_fps}fps")

logger.debug("Testing initial frame capture")

ret, test\_frame = self.cap.read()

if not ret:

logger.error("Cannot read frame from camera")

messagebox.showerror("Error", "Camera opened but cannot read frames")

return

if test\_frame is None:

logger.error("Camera returned empty frame")

messagebox.showerror("Error", "Camera returned empty frame")

return

logger.debug(f"Test frame captured successfully: {test\_frame.shape}")

logger.debug(f"Frame mean brightness: {test\_frame.mean()}")

if test\_frame.mean() < 1.0:

logger.warning("Frame appears to be completely black - camera may be covered or malfunctioning")

self.running = True

logger.info("Streaming state set to active")

logger.info("Starting audio streaming thread")

audio\_thread = threading.Thread(target=self.send\_audio, daemon=True)

audio\_thread.start()

logger.info("Starting video streaming loop")

self.update\_video()

except Exception as e:

logger.error(f"Failed to start streaming: {str(e)}", exc\_info=True)

messagebox.showerror("Error", f"Failed to start streaming: {str(e)}")

self.stop\_streaming()

def update\_video(self):

logger.debug(f"Video update called - running: {self.running}, camera available: {self.cap is not None}")

if not self.running or not self.cap:

logger.debug("Exiting video update - streaming stopped or camera unavailable")

return

try:

ret, frame = self.cap.read()

if not ret or frame is None:

logger.warning("Failed to read video frame, retrying")

if self.running:

self.root.after(100, self.update\_video)

return

frame\_rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

img = PIL.Image.fromarray(frame\_rgb)

imgtk = PIL.ImageTk.PhotoImage(image=img)

self.current\_image = imgtk

self.canvas.delete("all")

self.canvas.create\_image(0, 0, anchor=tk.NW, image=imgtk)

try:

frame\_bytes = stream\_protocol.encode\_video\_frame(frame, quality=80)

video\_fragments = stream\_protocol.fragment\_data("V", frame\_bytes)

server\_ip = self.server\_ip\_var.get()

fragments\_sent = stream\_protocol.send\_fragments(

self.client\_socket,

video\_fragments,

(server\_ip, self.udp\_port)

)

logger.debug(f"Video frame sent: {len(frame\_bytes)} bytes in {len(video\_fragments)} fragments")

current\_time = time.time()

if not hasattr(self, 'last\_status\_update'):

self.last\_status\_update = 0

if current\_time - self.last\_status\_update > 5:

self.stream\_status\_var.set(f"Connected as: {self.username} | Sending: {len(frame\_bytes)} bytes")

self.last\_status\_update = current\_time

except Exception as e:

logger.error(f"Error sending video frame: {e}", exc\_info=True)

except Exception as e:

logger.error(f"Error in video update loop: {e}", exc\_info=True)

if self.running:

self.root.after(33, self.update\_video)

def send\_audio(self):

logger.info("Audio streaming thread started")

try:

logger.debug("Opening audio input stream")

with sd.InputStream(samplerate=AUDIO\_RATE, channels=AUDIO\_CHANNELS, dtype=AUDIO\_DTYPE) as stream:

logger.info(f"Audio stream opened: {AUDIO\_RATE}Hz, {AUDIO\_CHANNELS} channels")

audio\_frame\_count = 0

while self.running:

audio\_data, overflowed = stream.read(AUDIO\_CHUNK)

if overflowed:

logger.warning("Audio buffer overflow detected")

continue

audio\_frame\_count += 1

if audio\_frame\_count % 1000 == 0:

logger.debug(f"Audio frames processed: {audio\_frame\_count}")

server\_ip = self.server\_ip\_var.get()

audio\_fragments = stream\_protocol.fragment\_data("A", audio\_data.tobytes())

stream\_protocol.send\_fragments(

self.client\_socket,

audio\_fragments,

(server\_ip, self.udp\_port)

)

time.sleep(0.01)

except Exception as e:

logger.error(f"Audio streaming error: {str(e)}", exc\_info=True)

if self.running:

self.root.after(0, lambda: messagebox.showerror("Audio Error", f"Audio stream error: {str(e)}"))

self.root.after(0, self.stop\_streaming)

logger.info("Audio streaming thread ended")

def stop\_streaming(self):

logger.info("Stopping streaming session")

self.running = False

if self.cap:

logger.debug("Releasing camera resources")

self.cap.release()

self.cap = None

if self.client\_socket:

logger.debug("Closing UDP socket")

self.client\_socket.close()

self.client\_socket = None

self.authenticated = False

self.username = None

self.udp\_port = None

self.show\_login\_frame()

self.login\_button.config(state=tk.NORMAL)

self.status\_var.set("Enter your credentials to start streaming")

logger.info("Streaming stopped, returned to login screen")

def on\_closing(self):

logger.info("Application closing")

self.running = False

if self.cap:

logger.debug("Releasing camera on application close")

self.cap.release()

if self.client\_socket:

logger.debug("Closing socket on application close")

self.client\_socket.close()

self.root.destroy()

logger.info("Application closed successfully")

if \_\_name\_\_ == "\_\_main\_\_":

logger.info("Starting Video Streamer application")

root = tk.Tk()

app = StreamerApp(root)

root.mainloop()

logger.info("Video Streamer application terminated")

protocol

*"""*

*Author: Oleg Shkolnik*

*Description: protocol of streaming and authentication between streamer and server*

*Date: 3/06/25*

*"""*

import socket

import struct

import numpy as np

import cv2

import logging

import datetime

import os

# Protocol constants

MAX\_UDP\_PACKET\_SIZE = 1024 # Standard UDP packet size

FRAGMENT\_HEADER\_SIZE = 20 # Header size

DATA\_TYPE\_VIDEO = "V" # Data type - video

DATA\_TYPE\_AUDIO = "A" # Data type - audio

def setup\_logging():

*"""*

*Sets up logging configuration for the application*

*"""*

# Create logs directory if it doesn't exist

if not os.path.exists('logs'):

os.makedirs('logs')

# Configure logging format

log\_format = '%(asctime)s - %(name)s - %(levelname)s - %(funcName)s:%(lineno)d - %(message)s'

# Set up file handler (logs to file)

file\_handler = logging.FileHandler(

f'logs/protocol\_{datetime.datetime.now().strftime("%Y%m%d")}.log'

)

file\_handler.setLevel(logging.DEBUG)

file\_handler.setFormatter(logging.Formatter(log\_format))

# Set up console handler (logs to terminal)

console\_handler = logging.StreamHandler()

console\_handler.setLevel(logging.INFO)

console\_handler.setFormatter(logging.Formatter(log\_format))

# Create logger

log = logging.getLogger('StreamerApp')

log.setLevel(logging.DEBUG)

# Add handlers to logger

log.addHandler(file\_handler)

log.addHandler(console\_handler)

return log

# Initialize logger at module level

logger = setup\_logging()

class StreamProtocol:

def \_\_init\_\_(self):

self.packet\_sequence = 0

self.video\_reassembly\_buffer = {}

self.audio\_reassembly\_buffer = {}

logger.info("StreamProtocol initialized")

def fragment\_data(self, data\_type, data):

*"""*

*Splits large data into fragments*

*:param data\_type: Data type ('V' for video or 'A' for audio)*

*:param data: Data to be fragmented*

*:return: List of fragments with headers*

*"""*

# Calculate maximum payload size

max\_payload\_size = MAX\_UDP\_PACKET\_SIZE - FRAGMENT\_HEADER\_SIZE

# Split into fragments

fragments = []

total\_fragments = (len(data) + max\_payload\_size - 1) // max\_payload\_size

logger.debug(f"Fragmenting {data\_type} data: {len(data)} bytes into {total\_fragments} fragments")

for i in range(total\_fragments):

start = i \* max\_payload\_size

end = min(start + max\_payload\_size, len(data))

fragment\_data = data[start:end]

# Create fragment header

try:

fragment\_header = struct.pack(

"!5I", # 5 integers

ord(data\_type[0]), # Data type

self.packet\_sequence, # Packet sequence number

total\_fragments, # Total fragments count

i, # Current fragment number

0 # Add fifth element (CRC or reserve)

)

except Exception as e:

logger.error(f"Header packing error: {e}")

raise

fragment = fragment\_header + fragment\_data

fragments.append(fragment)

self.packet\_sequence += 1

logger.debug(f"Data fragmentation completed, packet sequence: {self.packet\_sequence}")

return fragments

def send\_fragments(self, socket, fragments, address):

*"""*

*Sends fragments to the specified address*

*:param socket: UDP socket*

*:param fragments: List of fragments*

*:param address: Destination address (IP, port)*

*:return: Number of sent fragments*

*"""*

logger.debug(f"Sending {len(fragments)} fragments to {address}")

for fragment in fragments:

socket.sendto(fragment, address)

logger.debug(f"Successfully sent {len(fragments)} fragments")

return len(fragments)

def process\_fragment(self, data):

*"""*

*Processes received data fragment*

*:param data: Received data with header*

*:return: Tuple (data type, packet number, total fragments, fragment number, payload)*

*or None in case of error*

*"""*

# Check minimum length for header unpacking

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

logger.warning(f"Packet too short: {len(data)} bytes, expected at least {FRAGMENT\_HEADER\_SIZE}")

return None

# Safe unpacking

try:

# Unpack 5 integers

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

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

)

# Convert byte to character

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

# Get payload

payload = data[FRAGMENT\_HEADER\_SIZE:]

logger.debug(f"Processed fragment: type={data\_type}, seq={packet\_seq}, "

f"fragment={fragment\_num}/{total\_fragments}, payload\_size={len(payload)}")

return data\_type, packet\_seq, total\_fragments, fragment\_num, payload

except struct.error as e:

logger.error(f"Error unpacking header: {e}")

logger.error(f"Header bytes: {data[:FRAGMENT\_HEADER\_SIZE].hex()}")

return None

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

*"""*

*Adds fragment to reassembly buffer and checks if a complete packet can be assembled*

*:param data\_type: Data type ('V' or 'A')*

*:param packet\_seq: Packet number*

*:param total\_fragments: Total number of fragments*

*:param fragment\_num: Current fragment number*

*:param payload: Fragment payload*

*:return: Fully assembled data if all fragments received, otherwise None*

*"""*

# Choose appropriate buffer based on data type

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

# Create new entry in buffer if packet doesn't exist yet

if packet\_seq not in reassembly\_buffer:

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

logger.debug(

f"Created new reassembly buffer entry for packet {packet\_seq} with {total\_fragments} fragments")

# Save fragment

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

logger.debug(f"Stored fragment {fragment\_num} for packet {packet\_seq}")

# Check if all fragments received

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

# Assemble full packet

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

# Remove assembled packet from buffer

del reassembly\_buffer[packet\_seq]

return full\_data

fragments\_received = sum(1 for f in reassembly\_buffer[packet\_seq] if f is not None)

logger.debug(f"Packet {packet\_seq} incomplete: {fragments\_received}/{total\_fragments} fragments received")

return None

def encode\_video\_frame(self, frame, quality=80):

*"""*

*Encodes video frame into compressed format*

*:param frame: Image frame*

*:param quality: JPEG quality (0-100)*

*:return: Encoded frame data*

*"""*

if frame.size == 0:

logger.error("Attempted to encode empty frame")

raise ValueError("Empty frame!")

logger.debug(f"Encoding video frame with quality {quality}")

\_, encoded\_frame = cv2.imencode('.jpg', frame, [cv2.IMWRITE\_JPEG\_QUALITY, quality])

if encoded\_frame is None or len(encoded\_frame) == 0:

logger.error("Failed to encode video frame")

raise ValueError("Failed to encode frame")

logger.debug(f"Frame encoded successfully, size: {len(encoded\_frame)} bytes")

return encoded\_frame.tobytes()

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

*"""*

*Decodes data into video frame*

*:param data: Encoded frame data*

*:return: Decoded frame or None on error*

*"""*

logger.debug(f"Decoding video frame from {len(data)} bytes")

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

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

if decoded\_frame is None:

logger.error("Failed to decode video frame")

else:

logger.debug(f"Frame decoded successfully, shape: {decoded\_frame.shape}")

return decoded\_frame

def clear\_buffers(self):

*"""*

*Clears video and audio reassembly buffers*

*:return: None*

*"""*

video\_count = len(self.video\_reassembly\_buffer)

audio\_count = len(self.audio\_reassembly\_buffer)

self.video\_reassembly\_buffer.clear()

self.audio\_reassembly\_buffer.clear()

logger.info(f"Cleared reassembly buffers: {video\_count} video packets, {audio\_count} audio packets")

class AuthProtocol:

*"""*

*Protocol for authentication over TCP*

*"""*

@staticmethod

def send\_message(socket, message):

*"""*

*Send message with length prefix*

*"""*

# Convert message to bytes if it's a string

if isinstance(message, str):

message = message.encode()

# Send message length as 4-byte integer

message\_len = len(message)

socket.sendall(struct.pack('!I', message\_len))

# Send actual message

socket.sendall(message)

logger.debug(f"Sent message of {message\_len} bytes")

@staticmethod

def receive\_message(socket):

*"""*

*Receive message with length prefix*

*"""*

# Get message length (4 bytes)

header = socket.recv(4)

if not header or len(header) < 4:

logger.warning("Failed to receive message header or incomplete header")

return None

message\_len = struct.unpack('!I', header)[0]

logger.debug(f"Expecting message of {message\_len} bytes")

# Get actual message

chunks = []

bytes\_received = 0

while bytes\_received < message\_len:

chunk = socket.recv(min(message\_len - bytes\_received, 4096))

if not chunk:

logger.error("Connection closed while receiving message")

return None

chunks.append(chunk)

bytes\_received += len(chunk)

logger.debug(f"Successfully received message of {bytes\_received} bytes")

return b''.join(chunks)

def server\_authenticate(self, client\_socket, auth\_function, register\_function=None):

*"""*

*Server-side authentication via TCP*

*:param client\_socket: Accepted TCP client socket*

*:param auth\_function: Function to verify credentials*

*:param register\_function: Function to register user*

*:return: Username or None on error*

*"""*

try:

answer = 'f'

username = None

logger.info("Starting server authentication process")

while True:

# Receive credentials

data = self.receive\_message(client\_socket)

if not data:

logger.warning("Client disconnected during authentication")

return None

# Check if it's a registration request

if data.decode() == 'r' and register\_function:

logger.info(f"Processing registration request for user: {username}")

if register\_function(username, password):

logger.info(f"User {username} registered successfully")

break

else:

answer = 'fr'

logger.warning(f"Registration failed for user: {username}")

self.send\_message(client\_socket, answer)

continue

# Process credentials

user\_data = data.decode().split(',')

username = user\_data[0]

password = user\_data[1]

logger.info(f"Authentication attempt for user: {username}")

# Authenticate

if auth\_function(username, password):

answer = 't'

logger.info(f"User {username} authenticated successfully")

self.send\_message(client\_socket, answer)

break

else:

logger.warning(f"Authentication failed for user: {username}")

self.send\_message(client\_socket, answer)

return username

except Exception as e:

logger.error(f"Server authentication error: {e}")

return None

@staticmethod

def create\_socket(host='0.0.0.0', port=0, tcp=False):

*"""*

*Creates socket and binds it to the specified address*

*:param host: IP address*

*:param port: Port number*

*:param tcp: If True creates TCP socket, otherwise UDP*

*:return: Created socket*

*"""*

socket\_type = "TCP" if tcp else "UDP"

logger.info(f"Creating {socket\_type} socket on {host}:{port}")

if tcp:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

else:

sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)

sock.bind((host, port))

actual\_port = sock.getsockname()[1]

logger.info(f"{socket\_type} socket created and bound to {host}:{actual\_port}")

return sock

webroot

search-stream (connect to stream)

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Stream Search</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

min-height: 100vh;

}

.container {

background-color: white;

border-radius: 5px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

padding: 30px;

width: 70%;

max-width: 800px;

}

h1 {

text-align: center;

color: #333;

margin-bottom: 20px;

}

.search-form {

display: flex;

margin-bottom: 20px;

}

.search-form input[type="text"] {

flex-grow: 1;

padding: 10px;

border: 1px solid #ddd;

border-radius: 4px 0 0 4px;

font-size: 16px;

}

.search-form button {

background-color: #4CAF50;

color: white;

padding: 10px 15px;

border: none;

border-radius: 0 4px 4px 0;

cursor: pointer;

font-size: 16px;

}

.search-form button:hover {

background-color: #45a049;

}

.streamers-list {

margin-bottom: 20px;

max-height: 200px;

overflow-y: auto;

border: 1px solid #ddd;

border-radius: 4px;

}

.streamer-item {

display: flex;

justify-content: space-between;

align-items: center;

padding: 10px;

border-bottom: 1px solid #eee;

}

.streamer-item:last-child {

border-bottom: none;

}

.streamer-name {

font-weight: bold;

}

.join-button {

background-color: #2196F3;

color: white;

padding: 5px 10px;

border: none;

border-radius: 4px;

cursor: pointer;

}

.join-button:hover {

background-color: #0b7dda;

}

.stream-window {

background-color: #000;

height: 400px;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

color: white;

margin-bottom: 20px;

border-radius: 4px;

overflow: hidden;

position: relative;

}

.stream-window img {

max-width: 100%;

max-height: 100%;

object-fit: contain;

}

.stream-window .status {

position: absolute;

bottom: 10px;

left: 10px;

background-color: rgba(0, 0, 0, 0.7);

padding: 5px 10px;

border-radius: 4px;

}

.audio-controls {

display: flex;

align-items: center;

justify-content: space-between;

padding: 10px;

background-color: #f0f0f0;

border-radius: 4px;

margin-bottom: 20px;

}

.audio-controls .volume-control {

display: flex;

align-items: center;

}

.audio-controls label {

margin-right: 10px;

}

.audio-controls input[type="range"] {

width: 100px;

}

.audio-controls .mute-button {

background-color: #f44336;

color: white;

padding: 5px 10px;

border: none;

border-radius: 4px;

cursor: pointer;

}

.audio-controls .mute-button.unmuted {

background-color: #2196F3;

}

.actions {

display: flex;

justify-content: center;

margin-top: 20px;

padding-top: 20px;

border-top: 1px solid #eee;

}

.button {

background-color: #4CAF50;

color: white;

padding: 10px 15px;

border: none;

border-radius: 4px;

cursor: pointer;

text-decoration: none;

text-align: center;

width: 50%;

margin: 0 auto;

}

.button:hover {

background-color: #45a049;

}

.return-button {

background-color: #f44336;

color: white;

padding: 5px 10px;

border: none;

border-radius: 4px;

cursor: pointer;

margin-top: 10px;

}

.return-button:hover {

background-color: #d32f2f;

}

#search-interface, #stream-interface {

width: 100%;

}

#stream-interface {

display: none;

}

.streamer-info {

display: flex;

justify-content: space-between;

align-items: center;

margin-bottom: 10px;

}

.streamer-info h2 {

margin: 0;

}

</style>

</head>

<body>

<div class="container">

<div id="search-interface">

<h1>Stream Search</h1>

<form class="search-form">

<input type="text" placeholder="Enter streamer name..." id="streamer-search">

<button type="submit">Search</button>

</form>

<div class="streamers-list">

<div class="streamer-item">

<span class="streamer-name">Streamer1</span>

<button class="join-button">Join</button>

</div>

<div class="streamer-item">

<span class="streamer-name">Streamer2</span>

<button class="join-button">Join</button>

</div>

<div class="streamer-item">

<span class="streamer-name">Streamer3</span>

<button class="join-button">Join</button>

</div>

</div>

<div class="actions">

<a href="/profile.html" class="button">Update Profile</a>

</div>

</div>

<div id="stream-interface">

<div class="streamer-info">

<h2>Stream: <span id="streamer-name"></span></h2>

<button class="return-button" id="return-button">Return to Search</button>

</div>

<div class="stream-window">

<img id="stream-image" src="" alt="Video stream">

<div class="status" id="status">Connecting...</div>

</div>

<div class="audio-controls">

<div class="volume-control">

<label for="volume-slider">Volume:</label>

<input type="range" id="volume-slider" min="0" max="1" step="0.1" value="0.5">

</div>

<button class="mute-button unmuted" id="mute-button">Mute</button>

</div>

<audio id="audio-player" style="display: none;"></audio>

<div class="actions">

<a href="/profile.html" class="button">Update Profile</a>

</div>

</div>

</div>

<script>

// WebSocket connection for stream receiving

let ws = null;

// Audio context and variables for audio stream

let audioContext = null;

let audioSource = null;

let audioQueue = [];

let audioBuffer = null;

let isAudioInitialized = false;

let isMuted = false;

let audioGainNode = null;

// Variables for stream activity monitoring

let lastDataTimestamp = 0;

let activityCheckInterval = null;

const INACTIVITY\_TIMEOUT = 5000; // 5 seconds without data is considered stream stop

// Function to check stream activity

function checkStreamActivity() {

const currentTime = Date.now();

if (ws && ws.readyState === WebSocket.OPEN) {

if (currentTime - lastDataTimestamp > INACTIVITY\_TIMEOUT) {

console.log('Stream inactive, no data received');

document.getElementById('status').textContent = 'Stream inactive - waiting for data...';

if (currentTime - lastDataTimestamp > INACTIVITY\_TIMEOUT \* 2) {

alert('Stream ended - no data received');

clearInterval(activityCheckInterval);

returnToSearch();

}

}

}

}

// Initialize audio context

function initAudioContext() {

try {

if (!audioContext) {

window.AudioContext = window.AudioContext || window.webkitAudioContext;

audioContext = new AudioContext();

if (audioContext.state === 'suspended') {

audioContext.resume();

}

audioGainNode = audioContext.createGain();

audioGainNode.gain.value = document.getElementById('volume-slider').value;

audioGainNode.connect(audioContext.destination);

isAudioInitialized = true;

console.log('Audio context initialized');

}

} catch (error) {

console.error('Error initializing audio context:', error);

}

}

// Function to process audio data

function processAudioData(audioData) {

if (!isAudioInitialized) {

initAudioContext();

}

if (!isAudioInitialized || isMuted) return;

if (audioContext && audioContext.state !== 'running') {

console.log('Audio context not running, attempting resume');

audioContext.resume().catch(e => console.error('Error resuming audio context:', e));

}

try {

// Decode base64 to ArrayBuffer

const binary = atob(audioData);

const len = binary.length;

const buffer = new ArrayBuffer(len);

const view = new Uint8Array(buffer);

for (let i = 0; i < len; i++) {

view[i] = binary.charCodeAt(i);

}

// Assume data is in PCM 16-bit Little Endian format

// with sampling rate of 44100Hz and 1 channel (mono)

const frameCount = view.length / 2; // 16 bits = 2 bytes per sample

const sampleRate = 44100;

const audioBuffer = audioContext.createBuffer(1, frameCount, sampleRate);

const channelData = audioBuffer.getChannelData(0);

// Convert Int16 to Float32 (-1.0 to 1.0)

for (let i = 0; i < frameCount; i++) {

// Little Endian decoding 16-bit int

const sample = ((view[i\*2] & 0xff) | ((view[i\*2+1] & 0xff) << 8));

// Convert to signed int16

const signedSample = sample >= 0x8000 ? sample - 0x10000 : sample;

// Normalize to range -1.0...1.0

channelData[i] = signedSample / 32768.0;

}

// Playback

const source = audioContext.createBufferSource();

source.buffer = audioBuffer;

source.connect(audioGainNode);

source.start(0);

audioSource = source;

} catch (error) {

console.error('Error processing audio data:', error);

}

}

// Function to connect to stream via WebSocket

function connectToStream(streamerName) {

if (ws) {

ws.close();

}

const host = window.location.hostname;

const wsProtocol = window.location.protocol === 'https:' ? 'wss:' : 'ws:';

const wsPath = '/ws-proxy';

let wsUrl;

if (window.location.protocol === 'https:') {

wsUrl = `${wsProtocol}//${host}${wsPath}?streamer=${encodeURIComponent(streamerName)}`;

} else {

wsUrl = `${wsProtocol}//${host}${wsPath}?streamer=${encodeURIComponent(streamerName)}`;

}

console.log(`Connecting to WebSocket: ${wsUrl}`);

try {

ws = new WebSocket(wsUrl);

setupWebSocketHandlers(ws, streamerName);

if (window.location.protocol === 'http:') {

ws.addEventListener('error', (error) => {

console.warn('Error with primary WebSocket connection, trying fallback port');

const fallbackPort = 8766;

const fallbackUrl = `ws://${host}:${fallbackPort}${wsPath}?streamer=${encodeURIComponent(streamerName)}`;

console.log(`Connecting to fallback WebSocket: ${fallbackUrl}`);

ws = new WebSocket(fallbackUrl);

setupWebSocketHandlers(ws, streamerName);

}, { once: true });

}

return true;

} catch (err) {

console.error('Error creating WebSocket connection:', err);

document.getElementById('status').textContent = 'Failed to connect';

return false;

}

}

// Function to set up WebSocket handlers

function setupWebSocketHandlers(websocket, streamerName) {

websocket.onopen = function() {

console.log(`WebSocket connection established for ${streamerName}`);

document.getElementById('status').textContent = `Connected to ${streamerName}'s stream`;

const initMessage = JSON.stringify({

action: 'join',

streamer: streamerName

});

websocket.send(initMessage);

console.log('Sent join message:', initMessage);

initAudioContext();

};

websocket.onmessage = function(evt) {

try {

lastDataTimestamp = Date.now();

let message;

try {

message = JSON.parse(evt.data);

} catch (jsonError) {

console.error('Failed to parse message as JSON:', jsonError, evt.data);

return;

}

console.log('Message received, type:', message.type);

switch (message.type) {

case 'video':

const videoElement = document.getElementById('stream-image');

if (videoElement && message.data) {

videoElement.src = `data:image/jpeg;base64,${message.data}`;

document.getElementById('status').textContent = 'Streaming...';

} else {

console.error('Video element not found or empty data');

}

break;

case 'audio':

if (message.data) {

processAudioData(message.data);

} else {

console.error('Empty audio data received');

}

break;

case 'audio\_init':

console.log('Audio initialization signal received');

initAudioContext();

document.getElementById('status').textContent = 'Audio initialized';

break;

case 'info':

console.log(message.message);

document.getElementById('status').textContent = message.message;

break;

case 'error':

console.error(message.message);

document.getElementById('status').textContent = `Error: ${message.message}`;

break;

default:

console.log('Unknown message received:', message);

}

} catch (error) {

console.error('Error processing message:', error, evt.data);

}

};

websocket.onerror = function(error) {

console.error('WebSocket error:', error);

document.getElementById('status').textContent = 'Connection error';

};

websocket.onclose = function(event) {

console.log(`WebSocket connection closed: ${event.code} ${event.reason}`);

document.getElementById('status').textContent = 'Connection closed';

if (event.code !== 1000) {

console.log('Unexpected connection close');

}

};

const pingInterval = setInterval(() => {

if (websocket && websocket.readyState === WebSocket.OPEN) {

websocket.send(JSON.stringify({ action: 'ping' }));

} else {

clearInterval(pingInterval);

}

}, 30000);

}

// Function to handle search result and start stream

function joinStream(streamerName) {

document.getElementById('search-interface').style.display = 'none';

document.getElementById('stream-interface').style.display = 'block';

document.getElementById('streamer-name').textContent = streamerName;

document.getElementById('status').textContent = 'Connecting...';

lastDataTimestamp = Date.now();

if (activityCheckInterval) {

clearInterval(activityCheckInterval);

}

activityCheckInterval = setInterval(checkStreamActivity, 2000);

if (audioContext) {

audioContext.close().then(() => {

audioContext = null;

isAudioInitialized = false;

console.log('Audio context reset before new connection');

checkAndConnectToStream(streamerName);

}).catch(e => {

console.error('Error closing audio context:', e);

checkAndConnectToStream(streamerName);

});

} else {

checkAndConnectToStream(streamerName);

}

}

// Function to return to search

function returnToSearch() {

if (ws) {

ws.close();

ws = null;

}

if (activityCheckInterval) {

clearInterval(activityCheckInterval);

activityCheckInterval = null;

}

if (audioContext && audioSource) {

try {

audioSource.stop();

audioSource = null;

} catch (e) {

// Ignore errors if source is already stopped

}

}

if (audioContext) {

audioContext.close().then(() => {

audioContext = null;

isAudioInitialized = false;

console.log('Audio context closed and reset');

}).catch(e => console.error('Error closing audio context:', e));

}

audioQueue = [];

document.getElementById('stream-interface').style.display = 'none';

document.getElementById('search-interface').style.display = 'block';

}

function checkAndConnectToStream(streamerName) {

fetch(`/join-stream?streamer=${encodeURIComponent(streamerName)}`)

.then(response => {

if (!response.ok) {

throw new Error('Network response was not ok');

}

return response.json();

})

.then(data => {

if (!data.active || !data.exists) {

alert(data.exists ? 'Stream is not active' : 'User does not exist');

returnToSearch();

return false;

}

console.log('Stream available, connecting to WebSocket');

return connectToStream(streamerName);

})

.catch(error => {

console.error('Error checking stream availability:', error);

document.getElementById('status').textContent = 'Failed to check stream availability';

return connectToStream(streamerName);

});

}

// Function to search streamers

document.querySelector('.search-form').addEventListener('submit', function(event) {

event.preventDefault();

const searchTerm = document.getElementById('streamer-search').value;

fetch(`/search?term=${encodeURIComponent(searchTerm)}`)

.then(response => response.json())

.then(data => {

const streamersListElement = document.querySelector('.streamers-list');

streamersListElement.innerHTML = '';

if (data.length === 0) {

streamersListElement.innerHTML = '<div class="streamer-item">No streamers found</div>';

} else {

data.forEach(streamer => {

const streamerItem = document.createElement('div');

streamerItem.className = 'streamer-item';

const nameSpan = document.createElement('span');

nameSpan.className = 'streamer-name';

nameSpan.textContent = streamer;

const joinButton = document.createElement('button');

joinButton.className = 'join-button';

joinButton.textContent = 'Join';

joinButton.addEventListener('click', function() {

joinStream(streamer);

});

streamerItem.appendChild(nameSpan);

streamerItem.appendChild(joinButton);

streamersListElement.appendChild(streamerItem);

});

}

})

.catch(error => {

console.error('Search error:', error);

alert('An error occurred during search. Please try again later.');

});

});

// Initialize join buttons on page load

document.querySelectorAll('.join-button').forEach(button => {

button.addEventListener('click', function() {

const streamerName = this.parentElement.querySelector('.streamer-name').textContent;

joinStream(streamerName);

});

});

// Return to search handler

document.getElementById('return-button').addEventListener('click', function() {

returnToSearch();

});

// Function to check connection state and reconnect if necessary

function checkConnection(streamerName) {

if (ws && ws.readyState !== WebSocket.OPEN) {

document.getElementById('status').textContent = 'Reconnecting...';

connectToStream(streamerName);

}

}

// Handler for reconnecting after losing connection

window.addEventListener('online', function() {

const streamerName = document.getElementById('streamer-name').textContent;

if (streamerName && document.getElementById('stream-interface').style.display !== 'none') {

checkConnection(streamerName);

}

});

// Volume change handler

document.getElementById('volume-slider').addEventListener('input', function(event) {

if (audioGainNode) {

audioGainNode.gain.value = parseFloat(event.target.value);

}

});

// Mute button handler

document.getElementById('mute-button').addEventListener('click', function() {

const muteButton = document.getElementById('mute-button');

isMuted = !isMuted;

if (audioGainNode) {

audioGainNode.gain.value = isMuted ? 0 : document.getElementById('volume-slider').value;

}

if (isMuted) {

muteButton.textContent = 'Unmute';

muteButton.classList.remove('unmuted');

} else {

muteButton.textContent = 'Mute';

muteButton.classList.add('unmuted');

}

});

// Initialize audio context on user click

document.addEventListener('click', function() {

if (!isAudioInitialized) {

initAudioContext();

}

if (audioContext && audioContext.state === 'suspended') {

audioContext.resume();

}

}, { once: true });

</script>

</body>

</html>

log in

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sign in</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.container {

background-color: white;

border-radius: 5px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

padding: 30px;

width: 350px;

}

h1 {

text-align: center;

color: #333;

}

.form-group {

margin-bottom: 15px;

}

label {

display: block;

margin-bottom: 5px;

font-weight: bold;

}

input[type="text"],

input[type="password"] {

width: 100%;

padding: 10px;

border: 1px solid #ddd;

border-radius: 4px;

box-sizing: border-box;

font-size: 16px;

}

button {

background-color: #4CAF50;

color: white;

padding: 12px 15px;

border: none;

border-radius: 4px;

cursor: pointer;

width: 100%;

font-size: 16px;

margin-top: 10px;

}

button:hover {

background-color: #45a049;

}

.links {

text-align: center;

margin-top: 20px;

}

.links a {

color: #2196F3;

text-decoration: none;

}

.links a:hover {

text-decoration: underline;

}

.message {

text-align: center;

padding: 10px;

margin-bottom: 15px;

border-radius: 4px;

}

.error {

background-color: #f8d7da;

color: #721c24;

border: 1px solid #f5c6cb;

}

.success {

background-color: #d4edda;

color: #155724;

border: 1px solid #c3e6cb;

}

</style>

</head>

<body>

<div class="container">

<h1>Sign in</h1>

<div id="message-container"></div>

<form action="/login" method="post">

<div class="form-group">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

</div>

<div class="form-group">

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

</div>

<button type="submit">Sign in</button>

</form>

<div class="links">

<p>Don't have account? <a href="/register.html">Sign up</a></p>

</div>

</div>

</body>

</html>

sign up

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sign up</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.container {

background-color: white;

border-radius: 5px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

padding: 30px;

width: 350px;

}

h1 {

text-align: center;

color: #333;

}

.form-group {

margin-bottom: 15px;

}

label {

display: block;

margin-bottom: 5px;

font-weight: bold;

}

input[type="text"],

input[type="password"] {

width: 100%;

padding: 10px;

border: 1px solid #ddd;

border-radius: 4px;

box-sizing: border-box;

font-size: 16px;

}

button {

background-color: #4CAF50;

color: white;

padding: 12px 15px;

border: none;

border-radius: 4px;

cursor: pointer;

width: 100%;

font-size: 16px;

margin-top: 10px;

}

button:hover {

background-color: #45a049;

}

.links {

text-align: center;

margin-top: 20px;

}

.links a {

color: #2196F3;

text-decoration: none;

}

.links a:hover {

text-decoration: underline;

}

.message {

text-align: center;

padding: 10px;

margin-bottom: 15px;

border-radius: 4px;

}

.error {

background-color: #f8d7da;

color: #721c24;

border: 1px solid #f5c6cb;

}

.success {

background-color: #d4edda;

color: #155724;

border: 1px solid #c3e6cb;

}

</style>

</head>

<body>

<div class="container">

<h1>Sign up</h1>

<div id="message-container"></div>

<form action="/register" method="post">

<div class="form-group">

<label for="username">Username:</label>

<input type="text" id="username" name="username" required>

</div>

<div class="form-group">

<label for="password">Password:</label>

<input type="password" id="password" name="password" required>

</div>

<button type="submit">Sign up</button>

</form>

<div class="links">

<p>Do you have an account already? <a href="/index.html">Sign in</a></p>

</div>

</div>

</body>

</html>

registration-success

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Sign in successful</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.container {

background-color: white;

border-radius: 5px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

padding: 30px;

width: 350px;

text-align: center;

}

h1 {

color: #4CAF50;

}

p {

color: #333;

margin-bottom: 20px;

}

.button {

display: inline-block;

background-color: #4CAF50;

color: white;

padding: 10px 20px;

text-decoration: none;

border-radius: 4px;

font-weight: bold;

}

.button:hover {

background-color: #45a049;

}

</style>

</head>

<body>

<div class="container">

<h1>Sign in successful</h1>

<p>Your account was successfully created!</p>

<a href="/login.html" class="button">Sign in</a>

</div>

</body>

</html>

change password, logout

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Change Password</title>

<style>

body {

font-family: Arial, sans-serif;

background-color: #f4f4f4;

margin: 0;

padding: 0;

display: flex;

justify-content: center;

align-items: center;

height: 100vh;

}

.container {

background-color: white;

border-radius: 5px;

box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

padding: 30px;

width: 350px;

}

h1 {

text-align: center;

color: #333;

}

.form-group {

margin-bottom: 15px;

}

label {

display: block;

margin-bottom: 5px;

font-weight: bold;

color: #555;

}

input[type="password"] {

width: 100%;

padding: 10px;

border: 1px solid #ddd;

border-radius: 4px;

box-sizing: border-box;

font-size: 16px;

}

button {

background-color: #4CAF50;

color: white;

padding: 12px 15px;

border: none;

border-radius: 4px;

cursor: pointer;

width: 100%;

font-size: 16px;

margin-top: 10px;

}

button:hover {

background-color: #45a049;

}

.message {

text-align: center;

padding: 10px;

margin-bottom: 15px;

border-radius: 4px;

display: none;

}

.success {

background-color: #d4edda;

color: #155724;

border: 1px solid #c3e6cb;

}

.error {

background-color: #f8d7da;

color: #721c24;

border: 1px solid #f5c6cb;

}

.actions {

text-align: center;

margin-top: 20px;

padding-top: 20px;

border-top: 1px solid #eee;

display: flex;

justify-content: center;

}

.button {

background-color: #4CAF50;

color: white;

padding: 10px 15px;

border: none;

border-radius: 4px;

cursor: pointer;

text-decoration: none;

display: inline-block;

margin: 5px;

text-align: center;

width: 100%;

}

.button:hover {

background-color: #45a049;

}

.button.secondary {

background-color: #2196F3;

}

.button.secondary:hover {

background-color: #0b7dda;

}

.button.danger {

background-color: #f44336;

}

.button.danger:hover {

background-color: #d32f2f;

}

.return-link {

display: block;

text-align: center;

margin-top: 15px;

color: #2196F3;

text-decoration: none;

}

.return-link:hover {

text-decoration: underline;

}

.logout-button {

margin-top: 20px;

display: flex;

justify-content: center;

}

.logout-button .button {

width: auto;

}

</style>

</head>

<body>

<div class="container">

<h1>Change Password</h1>

<div id="message" class="message"></div>

<form id="password-form" action="/change-password" method="post">

<div class="form-group">

<label for="current-password">Current Password:</label>

<input type="password" id="current-password" name="current-password" required>

</div>

<div class="form-group">

<label for="new-password">New Password:</label>

<input type="password" id="new-password" name="new-password" required>

</div>

<div class="form-group">

<label for="confirm-password">Confirm New Password:</label>

<input type="password" id="confirm-password" name="confirm-password" required>

</div>

<button type="submit">Change Password</button>

</form>

<div class="actions">

<a href="/search-stream.html" class="button">Search Streams</a>

</div>

<div class="logout-button">

<a href="/logout" class="button danger">Log Out</a>

</div>

</div>

<script>

document.getElementById('password-form').addEventListener('submit', function(e) {

e.preventDefault();

const currentPassword = document.getElementById('current-password').value;

const newPassword = document.getElementById('new-password').value;

const confirmPassword = document.getElementById('confirm-password').value;

const messageElement = document.getElementById('message');

if (newPassword !== confirmPassword) {

messageElement.textContent = 'New passwords do not match!';

messageElement.className = 'message error';

messageElement.style.display = 'block';

return;

}

fetch('/change-password', {

method: 'POST',

headers: {

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

},

body: `current-password=${encodeURIComponent(currentPassword)}&new-password=${encodeURIComponent(newPassword)}&confirm-password=${encodeURIComponent(confirmPassword)}`,

credentials: 'same-origin'

})

.then(response => response.json())

.then(data => {

if (data.success) {

messageElement.textContent = 'Password changed successfully!';

messageElement.className = 'message success';

} else {

messageElement.textContent = data.message || 'Error changing password!';

messageElement.className = 'message error';

}

messageElement.style.display = 'block';

if (data.success) {

this.reset();

}

})

.catch(error => {

messageElement.textContent = 'An error occurred while sending the request!';

messageElement.className = 'message error';

messageElement.style.display = 'block';

console.error('Error:', error);

});

});

</script>

</body>

</html>