

**מגיש:**

**אופיר נבו מיכרובסקי**

**ת"ז:**

**216763433**

**מנחה:**

**ניר דוויק**

**בית-ספר:**

**תיכון הדסים**

**תאריך:**

**03/06/2025**

תוכן העניינים:

[תוכן העניינים: 2](#_Toc199902467)

[מבוא 4](#_Toc199902468)

[מבוא - ייזום 4](#_Toc199902469)

[אפיון 8](#_Toc199902470)

[תיאור תחום הידע 12](#_Toc199902471)

[יכולות בצד שרת 12](#_Toc199902472)

[יכולות בצד לקוח צופה 14](#_Toc199902473)

[יכולות כלליות 15](#_Toc199902474)

[מבנה / ארכיטקטורה 16](#_Toc199902475)

[תיאור הארכיטקטורה המוצעת: 16](#_Toc199902476)

[תיאור הטכנולוגיה הרלוונטית 20](#_Toc199902477)

[תיאור זרימת המידע במערכת 21](#_Toc199902478)

[תיאור האלגוריתמים המרכזיים 22](#_Toc199902479)

[תיאור סביבת הפיתוח 24](#_Toc199902480)

[תיאור הפרוטוקולים 26](#_Toc199902481)

[תיאור מסכי המערכת 29](#_Toc199902482)

[תיאור מבני הנתונים 32](#_Toc199902483)

[סקירת חולשות בפרויקט 32](#_Toc199902484)

[מימוש הפרויקט 33](#_Toc199902485)

[ספריות ומודולים ששומשו בפרויקט 33](#_Toc199902486)

[מודולים ומחלקות שאני פיתחתי 35](#_Toc199902487)

[מסמך בדיקות מלא 48](#_Toc199902488)

[מדריך למשתמש 50](#_Toc199902489)

[התקנת TC והרצתו 51](#_Toc199902490)

[התקנה והרצת השרת וה-RC 52](#_Toc199902491)

[רפלקציה אישית 54](#_Toc199902492)

[ביבליוגרפיה 55](#_Toc199902493)

מבוא

מבוא - ייזום

תיאור כללי:

המערכת המוצעת הינה מערכת שידור וידאו בזמן אמת, הכוללת מצלמה אלחוטית המחוברת למסך אינטרנטי, עם פונקציונליות מתקדמת של הוספת כתוביות לדיאלוג המתועד. המסך המרכזי יציג את המצלמה, יחד עם תמלול מדויק של המדובר.  
מוצר זה מיועד לשמש כרכיב תוכנתי חיוני במערכת עזר שמיעה אישית המותקנת ברכבים. פרויקט זה נולד מתוך הבנה עמוקה של הקשיים הייחודיים העומדים בפני אנשים כבדי שמיעה, במיוחד בסביבה רועשת כמו רכב נוסע. **המוצר הינו תוכנתי בלבד**.

אנשים עם לקויות שמיעה מתמודדים עם אתגרים רבים בעת נסיעה ברכב, שכן הם מתקשים לראות את שפתי הנוסעים האחרים ולקרוא אותן, ומכשירי השמיעה שלהם לעיתים קרובות לא יעילים בשל רעשי המנוע והכביש. מערכת זו נועדה לספק פתרון חדשני ויעיל, שיאפשר להם להשתתף באופן פעיל בשיחות המתנהלות ברכב, וליהנות מחוויית נסיעה נעימה ומכילה יותר.

המערכת תשתמש במצלמות אינטרנט פשוטות, ותציע מנגנון תמלול אוטומטי ע"י בינה מלאכותית. דגש מיוחד יושם על עיצוב ממשק משתמש נגיש, פשוט, נוח ואינטואיטיבי, שיתאים לצרכים הייחודיים של קהל היעד.

הגדרת לקוח:

מערכת זו מיועדת בראש ובראשונה לאנשים כבדי שמיעה, מכל קבוצות הגיל, המעוניינים להשתתף באופן מלא בשיחות המתנהלות ברכב. המערכת תוכננה במיוחד עבור משפחות ואנשים המרבים בנסיעות משותפות, ומעוניינים ליצור סביבה נעימה ומכילה עבור כלל הנוסעים. בנוסף, המערכת עשויה להתאים גם לאנשים הסובלים מקשיי שמיעה זמניים, הורים חדשים בתור מוניטור לתינוק, או לאנשים המעוניינים לשפר את חוויית הנסיעה שלהם באמצעות טכנולוגיה מתקדמת.

חשוב לציין כי המערכת מתאימה לסוגים שונים של רכבים, החל מרכבים פרטיים ועד לרכבי שטח. אנו שואפים להנגיש את הטכנולוגיה הזו לכמה שיותר אנשים, על מנת לאפשר להם ליהנות מנסיעה בטוחה, נעימה ומעשירה יותר.

הגדרת יעדים / מטרות:

מטרת העל של הפרויקט היא לשפר את איכות חייהם של אנשים כבדי שמיעה באמצעות טכנולוגיה מתקדמת. המערכת שואפת להעניק להם את היכולת להשתתף באופן פעיל בשיחות המתנהלות ברכב, דבר המהווה אתגר משמעותי עבורם כיום.

המטרות המרכזיות של המערכת כוללות:

* **שיפור הנגישות:** להפוך את הנסיעה ברכב לנגישה יותר עבור אנשים כבדי שמיעה.
* **עידוד השתתפות:** לאפשר לאנשים כבדי שמיעה להשתתף באופן מלא בשיחות ובפעילויות המתנהלות ברכב.
* **העלאת הביטחון העצמי:** להגביר את ביטחונם העצמי של אנשים כבדי שמיעה בעת נסיעה ברכב.
* **הפחתת תחושת הבדידות:** למנוע תחושת בדידות וניכור בקרב אנשים כבדי שמיעה במהלך נסיעות.
* **יצירת סביבה מכילה:** לתרום ליצירת סביבה נעימה ומכילה עבור כלל הנוסעים ברכב.

בעיות, תועלות וחסכונות:

מערכת זו נועדה לתת מענה למספר בעיות עיקריות:

* **קושי בתקשורת:** אנשים כבדי שמיעה מתקשים להשתתף בשיחות ברכב, במיוחד כאשר הם לא יכולים לראות את שפתי הדוברים.
* **רעש סביבתי:** רעשי מנוע, כביש וסביבה חיצונית מקשים על השימוש במכשירי שמיעה.
* **בידוד חברתי:** קושי בהשתתפות בשיחות גורם לבידוד חברתי ולתחושת ניכור.

התועלות הצפויות מהמערכת:

* **שיפור בתקשורת:** המערכת תאפשר לאנשים כבדי שמיעה להבין את הנאמר בצורה טובה יותר.
* **השתתפות פעילה:** המערכת תעודד השתתפות פעילה בשיחות ותמנע תחושת בדידות.
* **נוחות ובטיחות:** המערכת תשפר את חוויית הנסיעה ותגביר את תחושת הביטחון.
* **נגישות:** המערכת תהיה נגישה וקלה לשימוש עבור אנשים עם מוגבלויות שונות.

החסכונות הצפויים:

* **חסכון בזמן:** שיפור התקשורת ימנע אי הבנות ויחסוך זמן.
* **חסכון במאמץ:** המערכת תפחית את המאמץ הנדרש להבנת הנאמר.
* **חסכון רגשי:** המערכת תתרום לרווחה רגשית ולמניעת תסכול.

סקירת פתרונות קיימים:

בעת בחינת פתרונות קיימים עבור אנשים כבדי שמיעה ברכבים, נמצאו מספר גישות שונות. כל אחת מהגישות הללו מציעה מענה מסוים לאתגרים העומדים בפני אנשים כבדי שמיעה, אך לכל אחת גם מגבלות וחסרונות. הטבלה הבאה משווה בין הפתרונות הקיימים למערכת המוצעת, ומדגישה את היתרונות והחסרונות של כל אחד מהם:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| מכשירי שמיעה | אפליקציות תמלול | מערכות כתוביות | המערכת המוצעת | מאפיין |
| הגברת עוצמת הקול | תמלול בזמן אמת | הצגת כתוביות | שילוב וידאו ותמלול | עיקרון פעולה |
| שיפור שמיעה, ניידות | זמינות, עלות נמוכה | נוחות, קריאות | אינטגרציה, אוטומציה, נגישות, התאמה אישית | יתרונות |
| מוגבל ביעילות בסביבה רועשת, דורש התאמה אישית | לא נוח לשימוש ברכב, דורש הפעלה ידנית | לא מותאם לשימוש ברכב | עלות ראשונית | חסרונות |
| מוגבלת | לא מותאמת | לא מותאמת | מותאמת | התאמה לרכב |
| משתנה | לא נוח | לא נוח | נוח | נוחות שימוש |
| משתנה | משתנה | משתנה | גבוהה | נגישות |

לסיכום, המערכת המוצעת מהווה פתרון חדשני ויעיל עבור אנשים כבדי שמיעה ברכבים, ומהווה שיפור משמעותי על פני הפתרונות הקיימים.

סקירת טכנולוגיות הפרויקט:

פרויקט זה מתמקד בפיתוח רכיב התוכנה של מערכת עזר שמיעה לרכב, ולכן הטכנולוגיות שנבחרו ממלאות תפקיד קריטי בהצלחת הפרויקט. הטכנולוגיות הללו נבחרו בקפידה על מנת להבטיח את הפונקציונליות, היעילות והאמינות הגבוהה ביותר של המערכת.

להלן סקירה של הטכנולוגיות העיקריות שיילקחו בחשבון בפיתוח המערכת:

* **שפת תכנות:** פייתון. פייתון נבחרה בשל היותה שפה ורסטילית, קריאה ובעלת ביצועים גבוהים, המתאימה לפיתוח מערכות מורכבות כמו זו.
* **ספריית עיבוד תמונה:** OpenCV (Open Source Computer Vision Library). ספרייה זו מספקת כלים מקיפים לעיבוד וידאו בזמן אמת, החל מלכידת וידאו ממצלמות או קבצים, דרך עיבוד תמונה וזיהוי אובייקטים, ועד להצגת וידאו.
* **מערכת זיהוי דיבור ותמלול:** Google Cloud Speech-To-Text. מערכת זו מציעה תמלול אוטומטי בזמן אמת, זיהוי שפות והתאמה אישית, אשר חיוניים לפעילות המערכת.
* **מסד נתונים:** SQLite. מסד נתונים זה ישמש לאחסון וניהול נתונים, כגון הגדרות משתמש ונתוני תמלול.
* **פלטפורמת פיתוח**: JetBrains Pycharm Community Edition. פלטפורמה זו נבחרה בשל היותה סביבת פיתוח נוחה ויעילה, המאפשרת פיתוח חלק ומהיר.

חשוב להדגיש כי הפרויקט מתמקד במימוש **היבט התוכנה בלבד** של המערכת. לכן, לא תתבצע בפועל התקנה של המערכת ברכבים. הפיתוח יתמקד ביצירת סימולציה של המערכת, שתדגים את פעולתה ותאפשר הערכה של יעילותה.

במהלך הפיתוח, יושם דגש מיוחד על אופטימיזציה של ביצועים, עיצוב ממשק משתמש נגיש ואבטחת מידע. אלו נושאים בעלי חשיבות עליונה, אשר יבטיחו את איכותה ויעילותה של המערכת.

תיחום הפרויקט:

פרויקט זה מתמקד בפיתוח רכיב התוכנה של מערכת עזר שמיעה לרכב. במסגרת זו, הפרויקט נוגע במספר תחומים עיקריים:

* **עיבוד וידאו:** הפרויקט עוסק בלכידה, עיבוד והצגה של זרם וידאו ממספר מצלמות.
* **תקשורת אלחוטית:** הפרויקט כולל פיתוח מנגנון תקשורת **אלחוטית** בין המצלמות למסך המרכזי, לצורך שידור וידאו בזמן אמת.
* **זיהוי דיבור ותמלול:** הפרויקט משלב מערכת זיהוי דיבור ותמלול בזמן אמת, לצורך המרת דיבור לכתוביות.
* **פיתוח תוכנה:** הפרויקט כולל תכנון, פיתוח ובדיקה של רכיבי התוכנה השונים.
* **עיצוב ממשק משתמש:** הפרויקט שם דגש על עיצוב ממשק משתמש נגיש ואינטואיטיבי, המתאים לצרכים של אנשים כבדי שמיעה.

חשוב להדגיש כי הפרויקט **אינו** עוסק בתחומים הבאים:

* **התקנה פיזית:** הפרויקט אינו כולל התקנה פיזית של המערכת ברכבים.
* **חומרה:** הפרויקט אינו עוסק בפיתוח או רכישה של רכיבי חומרה, כגון מצלמות או מסכים.
* **אינטגרציה עם מערכות רכב:** הפרויקט אינו כולל אינטגרציה עם מערכות קיימות ברכב (כגון מערכת שמע או מערכת מולטימדיה).
* **בדיקות בשטח:** הפרויקט אינו כולל בדיקות של המערכת בתנאי שטח, כגון נסיעות מבחן.
* **מערכות הפעלה:** הפרויקט אינו עוסק במערכות הפעלה מכיוון שהפרויקט הולך לרוץ על מערכת ווינדוס עד שתהיה אפשרות להתקין את זה על חומרה.

אפיון

תיאור של המערכת:

המערכת שפותחה הינה מערכת תוכנה שמטרתה לסייע לאנשים כבדי שמיעה להשתתף בשיחות המתנהלות ברכב. היא עושה זאת על ידי לכידת וידאו ממצלמה אלחוטית ברכב, עיבוד התמונה והצגתה על מסך אינטרנטי, ותמלול בזמן אמת של השיחה.

המערכת פועלת באמצעות מספר מחשבי Windows אשר יריצו את התוכנה. מחשב אחד ישלח אודיו ווידאו למחשב שרת ייעודי. שרת זה ימיר את האודיו לכתוביות וישלח אותן ויעלה אותן לשרת ווב שיציג את התמלול עם הוידאו באתר, אשר מציג את השידור. על גבי שידור זה יוספו הכתוביות בזמן אמת.

פירוט יכולות:

* **לכידת ושידור וידאו ממצלמות:** המערכת מסוגלת לקבל ולעבד זרם וידאו ממספר מצלמות אלחוטיות הממוקמות ברכב.
* **עיבוד תמונה:** המערכת מבצעת עיבוד תמונה של זרם הווידאו, כגון התאמת גודל התמונה והצגת התמונות על המסך המרכזי.
* **תמלול בזמן אמת:** המערכת משלבת מערכת זיהוי דיבור ותמלול בזמן אמת, לצורך המרת דיבור לכתוביות המוצגות על המסך המרכזי.
* **הצגת כתוביות:** המערכת מציגה את הכתוביות על המסך המרכזי, באופן סינכרוני עם זרם הווידאו.
* **ממשק משתמש נגיש:** המערכת כוללת ממשק משתמש נגיש ואינטואיטיבי, המותאם לצרכים של אנשים כבדי שמיעה.

פירוט הבדיקות (קופסה שחורה)

במסגרת פרויקט זה, יבוצעו בדיקות קופסה שחורה אוטומטיות בתחילת ההרצה של התוכנה, על מנת להבטיח את תקינותה ומוכנותה לפעולה. להלן פירוט הבדיקות המתוכננות:

1. **בדיקות חיבוריות:**
   * **בדיקת חיבור לשרת:**
     + **מטרה:** לוודא חיבור מהיר ויציב לשרת התמלול.
     + **אופן הבדיקה:** התוכנה תנסה להתחבר לשרת.
     + **תוצאה מצופה:** חיבור מוצלח לשרת, או דיווח על שגיאת חיבור.
2. **בדיקות פונקציונליות בסיסיות:**
   * **בדיקת לכידת וידאו:**
     + **מטרה:** לוודא לכידה תקינה של וידאו מכל המצלמות המחוברות.
     + **אופן הבדיקה:** התוכנה תנסה ללכוד וידאו מכל המצלמות.
     + **תוצאה מצופה:** לכידת וידאו מוצלחת מכל המצלמות.
   * **בדיקת קליטת אודיו:**
     + **מטרה:** לוודא קליטה תקינה של אודיו מהמיקרופון המחובר.
     + **אופן הבדיקה:** התוכנה תנסה לקלוט אודיו מהמיקרופון.
     + **תוצאה מצופה:** קליטת אודיו מוצלחת מהמיקרופון.
   * **בדיקת תמלול בזמן אמת:**
     + **מטרה:** לוודא תמלול אודיו בזמן אמת בצורה מדויקת.
     + **אופן הבדיקה:** התוכנה תשלח אודיו לשרת ותבדוק את התמלול המתקבל.
     + **תוצאה מצופה:** תמלול מדויק של האודיו.
   * **בדיקת הצגת כתוביות:**
     + **מטרה:** לוודא הצגה תקינה של כתוביות על המסך.
     + **אופן הבדיקה:** התוכנה תציג כתוביות על המסך.
     + **תוצאה מצופה:** הצגה תקינה של הכתוביות, סנכרון עם הווידאו.

תכנון וניהול לו"ז לפיתוח המערכת:

לוח הזמנים הנ"ל הוא חילוק של הפרויקט ל-5 חלקים שונים של פיתוח המערכת, השלב הפרה-התחלתי, התחלתי, הבסיסי, המתקדם והסופי.

1. **השלב הפרה-התחלתי** – שלב זה יכיל הכנות של הפרויקט כולל כתיבת תבנית לפרויקט, התחלה של הלוגים ובנייה של הלוגרים. שלב זה יכיל את כל ההכנות כדי שהשלבים הבאים יוכלו להתקדם יותר ביעילות.  
   תוצרים: עיצוב של המערכת בנוסף לתבנית שאפשר להתבסס עליה.
2. **השלב ההתחלתי** – פיתוח של רכיבי המערכת יצירה של צ'אט שידור וידאו בסיסי לפי דרישת המערכת, יצירת מערכת משתמשים ורישום ויצירה של התמלול מוצמד לשידור.  
   תוצרים: קוד מקור של רכיבי המערכת, בדיקות יחידה
3. **השלב הבסיסי** – איחוד של חלקי המערכת שפותחו בשלב ההתחלתי.  
   תוצרים: מערכת בשלב alpha
4. **השלב המתקדם** – עיצוב של ממשק המשתמש לצורה יותר נעימה לעין כך ושיפור הביצועים ע"פ דו"ח הבדיקות ותיקון באגים סופיים.  
   תוצרים: מערכת מתוקנת ומשופרת.
5. **השלב הסופי** – סיכום של הפרויקט להוסיף בדיקות סופיות וסיום ספר פרויקט ומדריך משתמש ולהריץ בדיקות שטח.  
   תוצרים: מערכת מותקנת ומוכנה, מדריך משתמש וספר פרויקט.

|  |  |  |
| --- | --- | --- |
| שלב | לו"ז ראשוני | לו"ז בפועל |
| **השלב הפרה-התחלתי** | 1/3/25 | 16/3 |
| **השלב ההתחלתי** | 15/4/25 | 20/5 (ללא התמלול) |
| **השלב הבסיסי** | 15/5/25 | טרם |
| **השלב המתקדם** | 31/5/25 | טרם |
| **השלב הסופי** | 3/6/25 | טרם |

ניהול הסיכונים בפרויקט:  
במהלך פיתוח הפרויקט, זוהו מספר סיכונים פוטנציאליים אשר עשויים להשפיע על הצלחת הפרויקט. להלן פירוט הסיכונים והדרכים להתמודדות עמם:

|  |  |  |  |
| --- | --- | --- | --- |
| איך מתכננים לבדוק (לתאר בפירוט את שלבי הבדיקה) | מה אמורה לבדוק | שם הבדיקה (שיעיד על התוכן) | מס' בדיקה |
| בדיקה אוטומטית של חיבור לכל המצלמות המוגדרות. דיווח על שגיאות חיבור (אם קיימות). | חיבור מהיר ויציב לכל המצלמות המוגדרות | בדיקת חיבוריות - חיבור למצלמות אלחוטיות | 1 |
| בדיקה אוטומטית של חיבור למיקרופון המוגדר. דיווח על שגיאות חיבור (אם קיימות). | זיהוי וחיבור תקין למיקרופון המוגדר | בדיקת חיבוריות - חיבור למיקרופון | 2 |
| בדיקה אוטומטית של חיבור לשרת. דיווח על שגיאות חיבור (אם קיימות). | חיבור מהיר ויציב לשרת התמלול | בדיקת חיבוריות - חיבור לשרת | 3 |
| בדיקה אוטומטית של לכידת וידאו מכל המצלמות. וידוא איכות וידאו משתנה (רזולוציה, קצב פריימים). | לכידה תקינה של וידאו מכל המצלמות המחוברות | בדיקה פונקציונלית - לכידת וידאו | 4 |
| בדיקה אוטומטית של קליטת אודיו מהמיקרופון. וידוא איכות שמע משתנה (עוצמה, רעשי רקע). | קליטה תקינה של אודיו מהמיקרופון המחובר | בדיקה פונקציונלית - קליטת אודיו | 5 |
| שימוש בקובץ אודיו מקומי לבדיקה  בדיקה אוטומטית של התמלול המתקבל. וידוא תמיכה בשפות שונות, טיפול בהפסקות בדיבור. | תמלול אודיו בזמן אמת בצורה מדויקת | בדיקה פונקציונלית - תמלול בזמן אמת | 6 |
| בדיקה אוטומטית של הצגת כתוביות על המסך. וידוא סנכרון בין כתוביות לווידאו, אפשרויות עיצוב כתוביות. | הצגה תקינה של כתוביות על המסך | בדיקה פונקציונלית - הצגת כתוביות | 7 |
| ניטור אוטומטי של צריכת משאבים. דיווח על חריגות. | צריכת משאבים (CPU, זיכרון) תקינה בעת הפעלת התוכנה | בדיקת משאבי מערכת | 8 |
| בדיקה אוטומטית של תקינות רכיבי הממשק. וידוא נגישותיות למשתמשים עם מוגבלויות. | תקינות ממשק המשתמש (כפתורים, תפריטים, תצוגה) | בדיקת ממשק משתמש | 9 |

תיאור תחום הידע

יכולות בצד שרת

* **טיפול בחיבורים מרובים**
  + מהות: ניהול חיבורים של לקוחות משני הסוגים (משדר וצופה) וטיפול סימולטני ויעיל בהם.
  + אוסף יכולות/פעולות נדרשות:
    - קבלת חיבורים חדשים
    - זיהוי סוג חיבור (משדר/צופה)
    - ניהול רשימת חיבורים פעילים
    - טיפול בניתוקים
  + אובייקטים נחוצים: רשימת חיבורים, מנגנון זיהוי סוג חיבור, מנגנון ניהול משאבים, מנגנון ניהול משתמשים, ניהול משדרים וצופים
* **קבלת נתוני אודיו** **ווידאו**
  + מהות: קבלה ועיבוד של נתוני אודיו בזמן אמת מלקוח משדר.
  + אוסף יכולות/פעולות נדרשות:
    - קבלת נתוני אודיו מהלקוח
    - בדיקת תקינות הנתונים
  + אובייקטים נחוצים: מנגנון קבלה, מנגנון עיבוד אודיו ווידאו
* **תמלול טקסט מתוך אודיו**
  + מהות: המרת דיבור לתמלול טקסטואלי בזמן אמת.
  + אוסף יכולות/פעולות נדרשות:
    - זיהוי דיבור
    - המרת דיבור לטקסט
    - עיבוד תוצאות תמלול
  + אובייקטים נחוצים: מנוע זיהוי דיבור, מנגנון עיבוד טקסט
* **ניהול מאגר מידע של שיחות קודמות**
  + מהות: ניהול מאגר מידע של שיחות מתועדות.
  + אוסף יכולות/פעולות נדרשות:
    - גישה למסד נתונים
    - ניהול משתמשים (הוספה, מחיקה וקריאה)
  + אובייקטים נחוצים: מסד נתונים

**יכולות בצד לקוח משדר**

* **שליחת נתוני וידאו ואודיו**
  + מהות: לכידה ושליחה בזמן אמת של נתוני וידאו ואודיו מהמצלמה והמיקרופון לשרת.
  + אוסף יכולות/פעולות נדרשות:
    - גישה למצלמה ולמיקרופון
    - לכידת נתוני וידאו ואודיו
    - שליחה לשרת
  + אובייקטים נחוצים: ניהול קליטת וידאו, ניהול קליטת אודיו, פרוטוקול, מנהל תקשורת

יכולות בצד לקוח צופה

* **קבלת תמלול ווידאו**
  + מהות: קבלה והצגה של תמלול טקסט ווידאו בזמן אמת מהשרת.
  + אוסף יכולות/פעולות נדרשות:
    - קבלה של נתוני תמלול ווידאו מהשרת
    - פענוח הנתונים
  + אובייקטים נחוצים: מנגנון קבלה, מנגנון פענוח
* **הצגת וידאו עם תמלול**
  + מהות: הצגה סינכרונית של הווידאו והתמלול על המסך.
  + אוסף יכולות/פעולות נדרשות:
    - הצגת וידאו
    - הצגת תמלול
    - סנכרון בין וידאו לתמלול
  + אובייקטים נחוצים: מנגנון הצגה, מנגנון סנכרון
* **עיבוד תמונה**
  + מהות: עיבוד תמונה (התאמת גודל של התמונה למסך ע"פ כמות מצלמות).
  + אוסף יכולות/פעולות נדרשות:
    - קבלת תמונה
    - שינוי גודל תמונה
  + אובייקטים נחוצים: אין => נעשה דרך CSS

יכולות כלליות

* **תקשורת מוצפנת בהצפנה אסימטרית**
  + מהות: אבטחת התקשורת בין הלקוחות לשרת באמצעות הצפנה אסימטרית.
  + אוסף יכולות/פעולות נדרשות:
    - הצפנה של הודעות
    - פענוח של הודעות
    - **שימוש בספרייה חיצונית עקב שימוש בדפדפן**
  + אובייקטים נחוצים: מנגנון הצפנה אסימטרית
* **ממשק משתמש ויזואלי**
  + מהות: ממשק משתמש אינטואיטיבי המאפשר צפייה בשידור או שליחה
  + אוסף יכולות/פעולות נדרשות:
    - הצגת וידאו ותמלול
    - שידור וידאו
    - כניסת ויצירת משתמש
    - בחירה בין שידור לצפייה
  + אובייקטים נחוצים: רכיבי ממשק משתמש

מבנה / ארכיטקטורה

הפרויקט מכיל שלושה רכיבים עיקריים: שרת (Server), לקוח משדר (TC) ולקוח מקבל (RC).

לכל אחד מהרכיבים יש חלק משמעותי ושונה בפרויקט

תיאור הארכיטקטורה המוצעת:

שרת (Server):

השרת אחראי על העברת המידע בין הלקוח המשדר (TC) ללקוח המקבל (RC).

נוסף על כך השרת אחראי על פעולת התמלול והכתיבה למסד הנתונים.

השרת מתחלק לשלושה חלקים: TC\_Handler, RC\_Handler & Transcriptor.

כל אחד מהחלקים אחראי על אחת מהפעולות הראשיות של השרת (קבלה מלקוח משדר, תמלול וסכרון הוידאו והטקסט, העברה ללקוח המקבל).

TC\_Handler:

חלק זה בשרת אחראי על קבלת וידאו ואודיו מהלקוח המקבל בעזרת חיבור UDP עם פרוטוקול RTP ומסנכרן ביניהם לפני שהוא מעביר לרכיב השרת הבא.

Transcriptor:

חלק זה לוקח את האודיו ובעזרת קריאת API במטרה לתמלל את האודיו אשר התקבל מה-TC

RC\_Handler:

חלק זה אחראי על שליחת הוידאו והתמלול ל-RC ותקשורת עם ממסד הנתונים בעזרת חיבור TCP, HTTP, RTP ו-WebSockets.

A diagram of a machine

AI-generated content may be incorrect.

לקוח משדר (TC):

הלקוח המשדר אחראי על שידור הוידאו והאודיו לשרת.

רכיב זה הוא תוכנה לוקלית שמשדרת בעזרת חיבור UDP ופרוטוקול ה-RTP אודיו ווידאו בשני תהליכים נפרדים (פורט לוידאו ופורט לאודיו)

רכיב זה מכיל שני חלקים: Audio\_Handler, Video\_Handler.

כל חלק מהרכיב בנוי משלושה שלבים עיקריים: קליטה - הכנסה לפקטה – שליחה.

לדוגמא: קליטה של וידאו -> הכנסה לפקטת RTP -> שליחה לשרת.

A diagram of a product packaging

AI-generated content may be incorrect.

לקוח מקבל (RC):

תפקידו של רכיב ה-RC הוא הצגת שידורי וידאו ותמלול המתקבלים באופן שוטף מהשרת.

יישום זה מבוסס ווב (Web-based) ומנצל את פרוטוקול WebSocket להעברת נתונים בצורה מהירה ויעילה, המבטיחה תקשורת בזמן אמת (RTC) חיונית לשידורים חיים.

היתרונות בבחירת ארכיטקטורת ווב עבור ה-RC:

* נגישות גבוהה: המערכת נגישה מכל מכשיר המצויד בדפדפן אינטרנט.
* פיתוח ופריסה יעילים: קלות המימוש והתחזוקה בסביבת ווב.
* מתאים למגון רחב של פלטפורמות (cross-platform): עבודה חלקה על פני מערכות הפעלה ודפדפנים שונים ללא צורך בהתאמות ספציפיות.
* חווית משתמש נוחה: ממשק מוכר ואינטואיטיבי המקל על השימוש.

מאחר ומדובר באתר כל הקוד של ה-RC מאוחסן בתוך ה-Server אך ניגשים אל

ה-RC דרך הדפדפן.

כל שלושת הרכיבים הללו ביחד מנהלים את המוצר הסופי והכרחיים לתפעולו היעיל. A diagram of a server

AI-generated content may be incorrect.

תיאור הטכנולוגיה הרלוונטית

שפות תכנות: השפות השונות המשומשות בפרויקט הזה.

|  |  |  |
| --- | --- | --- |
| שם השפה | שימוש | סיבה לשימוש |
| Python | Server, RC, TC | שפה ראשית של הפרויקט.  נוחה לשימוש, כתיבה מהירה, מכילה ספריות רבות ומאוד ורסטילית. מוכרת במימוש הנוח שלה לרשתות |
| TypeScript | RC | משומשת למימוש ה-RC.  השפה משומשת בצורה נרחבת בעולם פיתוח הווב. נותנת את כל מה שיש ב-JS אבל עם בטחון של OOP ושימוש ב-Types. |
| HTML5 & CSS | RC | שפות ידועות לפיתוח ועיצוב אתרים. |
| SQL | Server | שפה ידועה ובטוחה לכתיבת מסדי נתונים. |

מערכות הפעלה: מערכות ההפעלה המשומשות בפרויקט

|  |  |  |
| --- | --- | --- |
| שם המערכת | שימוש | סיבה לשימוש |
| Windows | Server, RC, TC | מערכת הפעלה ידועה, נוחה לשימוש, ורסטילית ושומשה לפיתוח ובדיקות. |

ניתן להריץ את הפרויקט על כל מערכת הפעלה כל עוד מותקנים התוספים הרלוונטים

פרוטוקולים: הפרוטוקולים המשומשים בפרויקט

|  |  |  |
| --- | --- | --- |
| שם הפרוטוקול | שימוש | סיבה לשימוש |
| RTP | Server, RC, TC | פרוטוקול ידוע בתעשייה, נוח לשימוש וורסטילי |
| HTTP | Server, RC | פרוטוקול בו משתמשים בשביל לתקשר עם ה-browser |
| Websocket | Server, RC | פרוטוקול מעל HTTP שנותן לשרת יכולת לשלוח לקוח ווב הודעות ללא בקשה של הלקוח. |
| TCP | Server, RC | פרוטוקול אמין, מוודא הגעת הודעות, חובה לשימוש ב-HTTP |
| UDP | Server, TC | פרוטוקול מהיר יותר וידוע בשימוש בשביל סטרימינג |

תחומי עניין רלוונטים:

תקשורת, פרוטוקולים, בינה מלאכותית וטכנולוגיות web.

בינה מלאכותית: Vosk מודל תמלול קיים אשר מכיל תמיכה ישירה בפייתון ועבודה offline.

תיאור זרימת המידע במערכת

להלן תרשים המייצג את התהליך של המערכת

A diagram of a diagram

AI-generated content may be incorrect.

תיאור האלגוריתמים המרכזיים

חלוקת פריימים לפקטות והשלמתם בקבלה חזרה לפריימים:

לאחר קבלת הפריים ראיתי שהיו פריימים שהיו בגודל שהיה שדרש יותר מפקטה אחת. לכן אני החלטתי לבנות אלגוריתם לחלוקת הפריימים לפקטות.

שיטות רעיונות לאלגוריתם זה:

הגדרת מושגים:

* Marker Bit – ביט בפרוטוקול ה-RTP שאפשר להדליק בשביל לסמן בדרים לשרת/לקוח
* Extension – חלק מפרוטוקול הוא ה-extension אשר נותן להוסיף הרחבות לפרוטוקול במידת הצורך.

פתרון #1 – לא נבחר:

לשלוח את הפריים בחלקים, להשאיר לנמען להניח שכל סדרת פקטות שמגיעה בפרק זמן קצר שייכת לאותו פריים.  
פתרון זה עלול לגרום לבעיות כאשר יש איבוד פקטות או חפיפה בין פריימים. בנוסף, ניתן להשתמש בגודל פקטות קבוע מראש ללא התאמה לתוכן, מה שיכול לגרום לבזבוז רוחב פס ולפיצול לא אופטימלי.

פתרון #2 –נבחר:

לחלק את הפריימים לכמות פקטות אשר נרשמת בתוך הפקטות של הפריימים בעזרת הוספה של extension לפרוטוקול ה-RTP.

ה-extension יכיל את כמות הפקטות אשר אליהן חולק הפריים. בפקטה האחרונה אשר תשלח מדליקים את ה-Marker Bit בפרוטוקול.  
  
לאחר קבלת פקטת הmarker בודק השרת אם הוא קיבל את כל הפקטות של הפריים. אם לא הוא ממשיך לחכות אלא אם כן מתחיל להגיע פריים חדש.

במידה ומתחיל להגיע פריים חדש אז השרת זורק את הפריים החסר ומתחיל בקבלה של הפריים החדש.

אני בחרתי בפתרון השני מכיוון שאני הרגשתי שהוא גם הרבה יותר בטוח ויותר מתאים למבנה הפרוטוקול.

תזמון הוידאו והאודיו:

האמת שפה אין כל כך על מה לפרט מבחינת אפשרויות כי לא היו כל כך. יש אחת שאפשר לחשוב עליה שהיא הגיונית והיא לסנכרן את הוידאו והאודיו בעזרת ה-timestamp. ולכן זה הפתרון הנבחר.

הוידאו והאודיו של הלקוח המשדר יחלקו את אותו ה-timestamp בהתחלת השידור ובכך השרת יוכל לתזמן ביניהם

תיאור סביבת הפיתוח

סביבות פיתוח (IDE):

במהלך פיתוח פרויקט זה היה שימוש בשתי סביבות פיתוח:

* Pycharm
* Visual Studio Code

כל סביבה שומשה למטרה ספציפית בה היא מוכרת כהכי נוחה ומכילה כלים העוזרים בפיתוח.

|  |  |  |
| --- | --- | --- |
| סביבת הפיתוח | שימוש | סיבת הבחירה |
| Pycharm | פיתוח כל החלקים בפרויקט הדורשים הכתובים ב-Python  (TC & Server) | Pycharm היא סביבת הפיתוח הכי ידועה בשביל Python ומכילה כלים רבים (כגון דיבאגר מצוין) ואינטגרציות רבות (כגון גיט)  נוסף על כך אני משתמש ב-Pycharm שנים רבות ויודע איך לתפעל את התוכנה. |
| Visual Studio Code | פיתוח ה-RC. | VSC היא תוכנת עריכת קוד אשר משומשת רבות לפיתוח של אתרים וכתיבת תוכנה לווב. היא מכילה אינטגרציות של צפייה באתר ב"לייב" בנוסף לעוד כל מיני פיצ'רים נוספים העוזרים בבניית אפליקציות וובץ  התוכנה הזאת היא אחת התוכנות הראשונות שאני כתבתי בהן קוד ואני מכיר אותה היטב, דבר שעזר לי לבחור אותה בשביל הפרויקט הזה |
| ipython | בדיקות וניסיונות ע פייתון | טרמינל פייתון נוח וזמין שנותן פתרון פשוט ומהיר לעשייה של סניפטים של קוד ובדיקות אבל נותן יותר גמישות מ-Python Idle |

מערכת ניהול גרסאות:

בפרויקט זה כמו בכל פרויקט שלי בין אם הוא פרויקט תוכנתי או לא אני השתמשתי ב-git בשביל לשמור את השינויים שלי ולוודא שלא יקרה מצב שמשהו מפסיק לעבוד בפרויקט ואני לא יכול לחזור אחורה.

נוסף על כך על מנת שימוש נוח בגיט וצפייה נוחה בשינויים השתמשתי ב-github התור פלטפורמת ה-git שלי מאחר וזאת הפלטפורמה שאני מכיר הכי טוב.

כלים דרושים לבדיקות:

מאחר ופרויקט זה הוא פרויקט רשתי הכולל את אלמנטים של ווב, פייתון, רשתות, מדיה ועוד כל מיני נדרשים כל מיני כלים אשר עוזרים בבדיקות ודיבאגינג של הקוד והתוצר.

|  |  |  |
| --- | --- | --- |
| כלי | שימוש | סיבת בחירה |
| Wireshark | אבחון תקשורת | כלי מוכר, יש לי ניסיון רב איתו ואני מכיר אותו היטב. |
| Pycharm Debugger | דיבוג סקרפיטים של פייתון | כלי ברמה מאוד גבוהה שנותן תמיכה לדיבוג של תהליכים, תהליכונים רשתות ועוד |
| Chrome Dev Tools | דיבוג ואבחון בדפדפן | הכלי מובנה בתוך הדפדפן ונותן גישה מלאה לתקשורת גם מוצפנת, קוד FE ונותן לערוך שינויים זמניים בשביל בדיקות |
| Python logging | שמירה של נתונים בזמן הריצה שאפשר לנתח אחרי | ספרייה בנויה בפייתון שמספקת את הנדרש בצורה יעילה. |

תיאור הפרוטוקולים

בפרויקט יש שימוש בשלושה פרוטוקולים – HTTP, RTP, WebSocket

פרוטוקול ה-HTTP (גרסא מצומצמת):

**Hypertext Transfer Protocol (HTTP)** הוא פרוטוקול תקשורת סטנדרטי להעברת מידע בין לקוחות לשרתים ברשת האינטרנט.

הפרוטוקול פועל במודל בקשה-תגובה, שבו הלקוח שולח בקשה לשרת, והשרת מחזיר את המידע המבוקש.

שיטות HTTP עיקריות:

* **GET** - מיועדת לשליפת נתונים מהשרת. המידע נשלח בתוך כתובת ה-URL ואינו משפיע על נתוני השרת.
* **POST** - משמשת לשליחת נתונים לשרת, כגון נתוני טפסים או קובצי מדיה. הנתונים נשלחים בגוף הבקשה, מה שמאפשר אבטחה גבוהה יותר.

הפרדה בין נתונים בבקשותHTTP :

* A diagram of a computer code

  AI-generated content may be incorrect.)? (– מפריד בין נתיבים למשתנים בבקשות GET, לדוגמא: (example.com/page?user=123)
* )&( - מפריד בין שני משתנים שונים (user=123&age=25)
* )=( - מפריד בין שם המשתנה לערכו (name=John).
* \r\n - מסמן מעבר שורה ומפריד בין כותרות הבקשה (Headers)
* \r\n\r\n – הפרדה בין הכותרות לחלק הגוף של הבקשה בו מעבירים את המ ידע המועבר.

מאפייניHTTP :

HTTP הוא פרוטוקול stateless, כלומר כל בקשה נשלחת ונענית באופן עצמאי, ללא זיכרון של אינטראקציות קודמות. לשיפור ביצועים ואבטחה.

פרוטוקול ה-RTP:

Real-Time Transport Protocol (RTP) הוא פרוטוקול תקשורת המשמש להעברת מדיה בזמן אמת כגון אודיו ווידאו ברשתות IP. הפרוטוקול פותח כדי לאפשר סנכרון, ניהול רצף נתונים, וזיהוי אובדן פקטות, והוא משמש בעיקר ביישומי סטרימינג ושיחות וידאו.

מבנה פרוטוקול ה-RTP:

* **Sequence Number** - מזהה את סדר הפקטות כדי לאפשר שחזור נכון של הנתונים.
* **Timestamp** - מסייע בסנכרון בין זרמי מדיה שונים.
* **Payload Type** - מציין את סוג הנתונים (למשל, קידוד אודיו או וידאו).
* **Synchronization Source (SSRC)** - מזהה את מקור הנתונים כדי להבחין בין זרמים שונים.
* **Contribute Source Identifiers (CSRC)** - מזהה מקורות נוספים של מדיה (למשל, כשכמה משתמשים משתתפים בשיחת ועידה קולית, כל אחד מהם מקבל מזהה ייחודי בתוך הפקטות).
* **Extension Bit (X)** - אם מוגדר ל-1, זה מציין כי קיימים שדות הרחבה נוספים בפקטה. משתמשים בזה כדי להוסיף מידע נוסף מעבר למבנה הבסיסי של RTP
* **Marker Bit (M)** - מסמן פקטה מיוחדת, למשל פקטה שמסמנת את סוף מסגרת וידאו או נקודת התחלה של מקטע חשוב.
* **Payload Type (PT)** - מזהה את סוג הנתונים (לדוגמה, קידוד אודיו/וידאו כמו H.264 או Opus)

A diagram of a packet

AI-generated content may be incorrect.

פרוטוקול ה-WebSocket:

פרוטוקול הווב סוקט הוא רוטוקול המאפשר תקשורת רציפה מעל תקשורת אינטרנטית לעומת פרוטוקול ה-HTTP שהוא פרוטוקול stateless.

A diagram of a web browser

AI-generated content may be incorrect.פרוטוקול ה-Websocket מתחיל בלחיצת יד פשוטה דרך פרוטוקול ה-HTTP כמתואר בתרשים בצד.

לאחר לחיצת היד הצדדים המחוברים יכולים להתחיל לשלוח הודעות בעזרת נפרמטרים של הפרוטוקול:

* FIN (1 bit) – מסמן אם זאת הפקטה האחרונה בהודעה.
* RSV1..3 (1 bit each) – ביטים שמורים להרחבות של הפרוטוקול
* Opcode (4 bits) – סוג התוכן (בינארי, טקסט וכ"ו)
* MASK (1 bit) – מציין אם יש מפתח הצפנה בשימוש
* Payload Length (7 bit) – גודל הנתונים המועברים בפקטה. חלק זה יותר מסובך ובנוי בצורה הבאה. במידה ויש צורך בהרחבה אז מתווסף באפר שנקרא (Extended payload length) שהוא תוספת של בתים לאורך הנתונים

|  |  |  |
| --- | --- | --- |
| גודל נתונים | גודל Payload Length | ערך Payload Length |
| עד 125 בתים | 7 ביט | גודל הנתונים |
| 125 - 65,535 בתים | 23 ביט (עוד 2 בתים) | 7 ביט התחלתי = 126  שני בתים נוספים = גודל הנתונים |
| 65,535 ומעלה | 87 ביט (8 בתים נוספים) | 7 ביט התחלתי – 127  8 בתים נוספים = גודל הנתונים |

* Masking Key (4 bytes) – מפתח אקראי המשמש להצפנת הנתונים, **נדרש רק בהודעות מהלקוח לשרת.** לא קיים בהודעות מהשרת ללקוח.
* A screenshot of a computer

  AI-generated content may be incorrect.Payload Data – הנתונים עצמם אשר מועברים בפקטה

תיאור מסכי המערכת

מסכי ה-RC:

מסך פתיחה:

בפתיחת המסך מוצגות שתי אפשרויות – Connect & Conversations

Join stream – התחברות לשידור

View saved – צפייה בשיחות אשר תועדו.

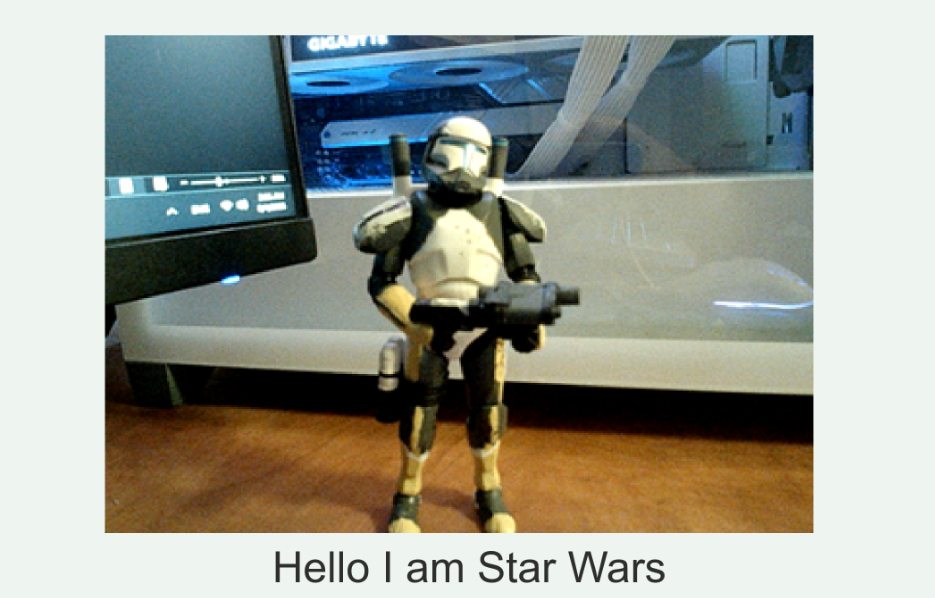
עיצוב:



מסך שידור:

במסך מוצג השידור המתומלל הנשלח מהשרת. במידה ואין שידור תוצג הודעה

עיצוב:



מסך שיחות שתועדו:

במסך מוצגת רשימה של כל השיחות שתועדו – כותרת, ותאריך

לכל שיחה יש שתי אפשרויות, הורדה ומחיקה.

הורדה: הורדה של תמלול השיחה בקובץ טקסט לטלפון

מחיקה: מחיקת תיעוד השיחה מממסד הנתונים.

A close-up of a date

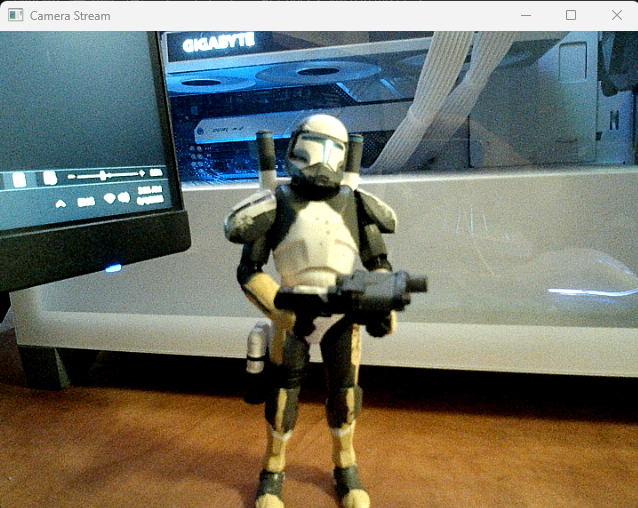
AI-generated content may be incorrect.עיצוב:

מסכי ה-TC:

מסך שידור:

מסך המציג את המצלמה והשידור

עיצוב:



תיאור מבני הנתונים

הפרויקט מכיל ממסד נתונים אחד ופשוט ממסד נתונים של השיחות השונות אשר תועדו וככה הוא נראה:

|  |  |  |  |
| --- | --- | --- | --- |
| Filename: string | Date: string | Title: string | Id: int |
| Conversation-1-23/5/2024.txt | 23/5/2024 | Conversation-1 | 1 |

בטבלה מוצגות כל השיחות השמורות ע"פ מספר מזהה ייחודי, הכותרת, התאריך ושם הקובץ של השיחה.

סקירת חולשות בפרויקט

פרויקט זה אינו מאוד מסובך לתפעול מבחינת אבטחה מאחר שאינו מכיל הרבה דברים.

ממסד נתונים:

כדי להגן מפני פריצות SQL Injection יש שימוש בשאילתות פרמטריות שהן חלק מספריית SQLite בשביל להגן מפני המתקפות הללו. השאילתות בנויות מיישום של (?) בכל מקום בו נועד להכנס מידע ולאחר מכן העברת המידע לפי סדר שלאחר מכן הספרייה בודקת ומאשרת.

Web:

מבחינת הווב הכל משתמש בהצפנת SSL/TLS (Transport Layer Security) ותעודה חתומה עצמאית (ssc) בשביל לשמור על כל המידע מוצפן ומאובטח. בשביל הגנה ממתקפות MITM.

UDP Connection:

בחלק זה גם כן יש שימוש בהצפנת DTLS (Datagram Transport Layer Security). הצפנה זאת חוסמת MITM ומאזינים לא רצויים.

מימוש הפרויקט

ספריות ומודולים ששומשו בפרויקט

|  |  |  |
| --- | --- | --- |
| שם המודול / ספרייה | סיבת השימוש | סיבת הבחירה |
| socket | ניהול תקשורת | ספרייה שמובנת בתוך Python, נוחה לשימוש, אני מכיר אותה היטב וידועה לניהול תקשורת. |
| os | אינטראקציה עם מערכת ההפעלה | מאפשר גישה לשירותי מערכת ההפעלה שמשומשים בפרויקט. |
| logging | ניהול ורישום לוגים | כלי מובנה לניהול לוגים בצורה מסודרת, גמיש וניתן להתאמה אישית בקלות. |
| time | ניהול זמן ושעון (fps) | מספק פונקציות לעבודה עם זמן, כולל מדידת ביצועים, השהיות, וקבלת תאריך נוכחי. |
| select | ניהול לקוחות מרובים  ב-RC | מאפשר ניהול לקוחות מרובים בשרתים בצורה פשוטה ונוחה |
| re | עיבוד ביטויים רגולריים | משמש לחיפוש, התאמה ומניפולציה של טקסטים עם ביטויים רגולריים, תומך בגמישות מירבית. |
| struct | עבודה עם מבני נתונים בינאריים | מאפשר המרה של נתונים בין פורמט בינארי לפורמט ש-Python מבין, חיוני בתקשורת והמרת נתונים. |
| zlib | דחיסת נתונים של וידאו ואודיו ב-TC | נוחה ופשוטה לשימוש ובנויה בתוך python |
| numpy | עיבוד וידאו | ספרייה נוחה ומוכרת שנותנת תמיכה  ב-opencv. |
| cv2 | משומשת לקליטת וידאו מה-TC ולקריאת הוידאו ב-Server | מוכרת בעולם, נוחה לשימוש ויש לי ניסיון רב איתה מפרויקטים קודמים |

|  |  |  |
| --- | --- | --- |
| שם המודול / ספרייה | סיבת השימוש | סיבת הבחירה |
| pyaudio | קליטת אודיו מה-TC | מאפשר ניהול והקלטה של אודיו, מתאים לעיבוד קובצי קול ולזיהוי דיבור. |
| math | פונקציות מתמטיות מתקדמות | מובנה בתוך python ונוח לשימוש. |
| random | יצירת מספרים רנדומליים ssrc (מספר מזהה בפרוטוקול RTP) | מובנה בתוך python, נוח וקל לשימוש. |
| threading | ניהול Threads ב-Server | מאפשר יצירה ועובדה של תהליכים בתוך התוכנה. |
| multiprocessing | ניהול תהליכים ב-TC שליחת אודיו ושליחת וידאו | מספק תמיכה לריבוי תהליכים, משפר ביצועים במערכות עם עיבוד כבד ע"י הפעלת מספר ליבות של המעבד. |
| ssl | http and websocket encryption | ספרייה שהדפדפן יודע לעבוד עם המידע שהיא שולחת אליו ומטפלת בכל העניין אוטומטית. |
| dtls | ספרייה משומשת להצפנת זרמי UDP | לפי מחקר אינטרנטי הספרייה הכי מומלצת להצפנת UDP |
| sqlite3 | ניהול DB | ספרייה שאני משתמש כבר הרבה שנים ומכילה פרמטריזציה מובנת |
| vosk | תמלול | נותן אפשרות לשימוש אופליין של מודלים לוקליים לתמלול טקסט |

מודולים ומחלקות שאני פיתחתי

**AudioReceiveHandler**

שימוש: Handles real-time UDP audio reception with delta time synchronization.

תכונות: אין

|  |  |
| --- | --- |
| Docs | Method name |
| Receives RTP audio packets and plays audio in real-time using delta time for better timing. :return: None | recv\_audio |

**VideoTransmission**

שימוש: Bundles video handlers and handles transmission

תכונות:

* video\_capture\_source – camera port
* port – port of udp connection
* payload\_type – rtp payload type of video
* rtp\_handle – instance of RTPHandler
* udp\_handle – instance of UDPClientHandler
* logger – class logger

|  |  |
| --- | --- |
| Docs | Method name |
| transmit video using rtp :return: None | transmit\_video |

**VideoCapture**

שימוש: This class handles the video capture and handling.

תכונות:

* logger – class logger
* source – camera source port
* capture – cv2 capture instance

|  |  |
| --- | --- |
| Docs | Method name |
| Retrieves a single frame from the video source.  :return tuple: (success, frame), where `success` is a boolean indicating if the frame was read successfully, and `frame` is the captured frame. | get\_frame |
| Releases the video capture resource.  :return: None | release |
| Ensures resources are released when the instance is deleted.  :return: None | \_\_del\_\_ |

**UDPClientHandler**

שימוש:

This class represents the video transmitting client using UDP.  
It initializes an RTPHandler instance to handle RTP packet creation  
and uses a UDP socket for sending packets.

תכונות:

* server\_address – server host
* server\_port – udp binding port
* sock – udp socket
* logger – class logger

|  |  |
| --- | --- |
| Docs | Method name |
| Captures video frames, creates RTP packets, and transmits them to the server via UDP. :param packets: (list[bytes]) packets to send :return bool: whether operation was successful | send\_packets |

**RTPHandler**

שימוש:

This class handles the RTP protocol, including creating and parsing RTP packets.

תכונות:

* payload\_type – rtp payload type
* ssrc – rtp ssrc
* sesquence\_number – rtp sequence number (init 0)
* logger – class logger

|  |  |
| --- | --- |
| Docs | Method name |
| Constructs the RTP header.  :param extension\_data: :param marker: (int) The marker bit. :param csrcs: (list[int]) List of contributing source identifiers. :return: bytes: The RTP header as a byte string. | build\_header |
| Creates an RTP packet by combining the header and payload.  :param payload: (bytes) payload to put in the rtp packet :param csrcs: (list[int]) List of contributing source identifiers. :return: bytes: The complete RTP packet. | create\_packets |
| Returns the SSRC for the RTP stream.  :return: int: The SSRC value. | get\_ssrc |
| updates timestamp :return: None | \_update\_timestamp |

**AudioTransmissionHandler**

שימוש:

Handles real-time audio transmission via RTP over UDP with delta time synchronization.

תכונות:

* logger – class logger
* audio\_capture – pyaudio audio capture instance
* rtp\_handler – rtpHandler instance
* udp\_handler – udpClientHandler instance

|  |  |
| --- | --- |
| Docs | Method name |
| Begins live audio transmission with delta time control for better synchronization. :return: None | start\_streaming |
| Cleans up resources after transmission stops. :return: None | stop\_streaming |

**AudioCapture**

שימוש:

Handles real-time audio capture and streaming

תכונות:

* rate – sample rate (const)
* channels – channels (const)
* chunk\_size – hunk size (const)
* logger -class logger
* audio – PyAudio instance
* stream – opening input stream in pyaudio

|  |  |
| --- | --- |
| Docs | Method name |
| Captures and returns a single chunk of audio data.  :return bytes: Raw audio data. | get\_audio\_chunk |
| Cleans up resources. :return: None | release |

**VideoReceiveHandler**

שימוש:

Handles real-time UDP video reception

תכונות: אין

|  |  |
| --- | --- |
| Docs | Method name |
| Receives video and presents it :return: None | recv\_video |

**UDPServerHandler**

שימוש:

This class represents the video receiving server using UDP.  
It initializes a UDP socket to listen for incoming packets and  
uses RTPacketDecoder to decode the received RTP packets.

תכונות:

* bind\_address – (const) the server address
* bind\_port – port to listen on
* sock – udp socket
* \_uncompleted\_frame\_packets – dictionary used to assemble messages
* logger – class logger

|  |  |
| --- | --- |
| Docs | Method name |
| Will receive a packet using a UDP connection :param buffer: (int) the buffer to receive :return bytes | None: The received packet if one was received | \_receive\_packet |
| Assembles / Discards frames :param seq\_start: (int) fragment sequence start :param seq\_end: (int) fragment sequence end :return bytes | None: bytes if a frame was assembled, None if frame was dropped | assemble\_rtp\_message |
| Receives RTP packets from the client & decodes them. :return bytes or None: If a message/packet is received then the payload will be returned. | receive\_rtp\_message |

**RTPPacketDecoder**

שימוש:

Decodes RTP packets into header fields and payload.

תכונות:

* packet – the raw packet
* version – rtp version
* padding – rtp padding bit
* extension – rtp extension bit
* csrc\_count – rtp csrc count
* marker – rtp marker bit
* payload\_type – rtp payload type
* sequence\_number – rtp sequence number
* timestamp – rtp timestamp
* ssrc – rtp ssrc
* csrcs – rtp csrcs list
* extension\_data – holds the data of the extension if there is one
* payload – rtp payload
* logger – class logger

|  |  |
| --- | --- |
| Docs | Method name |
| Decodes the RTP packet, extracting its header and payload. :return: None | decode\_packet |
| Returns the RTP header fields as a dictionary.  :return: dict: Header fields with their values. | get\_header\_info |
| Returns the payload from the RTP packet.  :return: bytes: The raw payload data. | get\_payload |

**Transcriptor**

מימוש:

Handles live speech recognition and captioning using AudioCapture.

תכונות:

* model – vosk model instance
* recognizer – vosk voice recognizer
* logger – class logger
* audio\_capture – audioCapture instance

|  |  |
| --- | --- |
| Method | Docs |
| transcribe\_audio | Continuously captures and transcribes live audio.  :return: None |

**HttpParser**

מימוש:

Class to make http requests usable in code more easily

תכונות:

* METHOD – http method (get / post)
* URI – uri of the request
* HTTP\_VERSION – version from the request
* QUERY\_PARAMS – query parameters given in the request
* HEADERS – headers given in the request
* BODY – body of the request
* COOKIES – cookies given in the request

|  |  |
| --- | --- |
| Method | Docs |
| \_\_header\_parser | Extracts the headers from the request :return dict[bytes]: {header: value} |
| \_\_body\_parser | Extracts the body from the requests :return bytes: the body of the request |
| \_\_section\_one\_parser | Splits the first section of the http\_ophir request (before the headers) :return tuple: (method, uri, http\_version, query\_params) |
| \_\_str\_\_ | Str dunder function for the HttpMsg class :return str: The http\_ophir message in full |

**HttpMsg**

מימוש:

Create easy to use http messages including responses and requests

תכונות:

* error\_codes – error codes in the http message
* body – body of the http message
* headers – dict of the headers

|  |  |
| --- | --- |
| Method | Docs |
| \_\_error\_code\_finder | Returns the error code and message using the error code. :param error\_code: The error code to be found :return bytes: The error code |
| prettify | Prints the message in a readable detailed format :return: None |
| \_\_build\_headers\_bytes | Formats the headers into bytes :return bytes: Byte string with the headers formatte |
| build\_message\_bytes | Builds the http\_ophir message :return bytes: The http\_ophir message |
| \_\_str\_\_ | Str dunder function for the HttpMsg class :return str: The http\_ophir message in full |

**App**

מימוש:

The Base for the web app

תכונות:

* routes – dict holding all the routes
* four\_o\_four – holds the path to the default 404 page
* \_\_closed – bool is connection closed

|  |  |
| --- | --- |
| Method | Docs |
| route | Route decorator - adds the route to the routes' dictionary. :param permission\_cookie: The cookie name that allows permission to page. :param route: the route for the uri :return: what the original function needs to return |
| \_\_receive\_message | Receives a message from a client :param client\_socket: :return bool or HttpParser: False if the message is invalid, HttpParser with the message |
| \_\_handle\_client | Handles the client input. :param request: The request that the server got. :param client\_socket: The client socket. :return: None |
| set\_four\_o\_four | Sets the route to the 404 page. :param route: The route to the 404 page html. :return: None |
| close\_app | Shuts down the server and closes the server :return: None |
| run | Starts the http server. :return: None |

מסמך בדיקות מלא

|  |  |  |  |
| --- | --- | --- | --- |
| דרך התמודדות | דרך הבדיקה והאם צלחה | מה אמורה לבדוק | שם הבדיקה (שיעיד על התוכן) |
| בדיקה של פרוטוקול ה-RTP ותיקון מימושו | בדיקה של חיבור למצלמה המוגדרת. דיווח על שגיאות חיבור (אם קיימות).  נכשל | חיבור מהיר ויציב למצלמה המוגדרת | בדיקת חיבוריות - חיבור למצלמות אלחוטיות |
|  | בדיקה של חיבור למיקרופון המוגדר. דיווח על שגיאות חיבור (אם קיימות).  עבר בהצלחה | זיהוי וחיבור תקין למיקרופון המוגדר | בדיקת חיבוריות - חיבור למיקרופון |
|  | בדיקה של חיבור לשרת. דיווח על שגיאות חיבור (אם קיימות).  עבר בהצלחה | חיבור מהיר ויציב לשרת התמלול | בדיקת חיבוריות - חיבור לשרת |
|  | בדיקה של לכידת וידאו מכל המצלמות. וידוא איכות וידאו משתנה (רזולוציה, קצב פריימים).  עבר בהצלחה | לכידה תקינה של וידאו מכל המצלמה המחוברת | בדיקה פונקציונלית - לכידת וידאו |
|  | בדיקה של קליטת אודיו מהמיקרופון. וידוא איכות שמע משתנה (עוצמה, רעשי רקע).  עבר בהצלחה | קליטה תקינה של אודיו מהמיקרופון המחובר | בדיקה פונקציונלית - קליטת אודיו |

|  |  |  |  |
| --- | --- | --- | --- |
| בדיקה על הליך התמלול  ובדיקת אלטרנטיבות | שימוש בקובץ אודיו מקומי לבדיקה בדיקה אוטומטית של התמלול המתקבל. וידוא תמיכה בשפות שונות, טיפול בהפסקות בדיבור.  נכשל | תמלול אודיו בזמן אמת בצורה מדויקת | בדיקה פונקציונלית - תמלול בזמן אמת |

|  |  |  |  |
| --- | --- | --- | --- |
| שכתוב פעולת הצגת הכתוביות בזמן אמת | בדיקה של הצגת כתוביות על המסך. וידוא סנכרון בין כתוביות לווידאו, אפשרויות עיצוב כתוביות.  נכשל | הצגה תקינה של כתוביות על המסך | בדיקה פונקציונלית - הצגת כתוביות |
|  | ניטור של צריכת משאבים. דיווח על חריגות.  עבר בהצלחה | צריכת משאבים (CPU, זיכרון) תקינה בעת הפעלת התוכנה | בדיקת משאבי מערכת |

מדריך למשתמש

**A screenshot of a computer

AI-generated content may be incorrect.ע**ץ קבצים

התקנת TC והרצתו

בשביל להתקין את ה-TC יש להתקין את פייתון דרך האתר הרשמי של פייתון ולהריץ לאחר מכן את הפקודות הבאות:

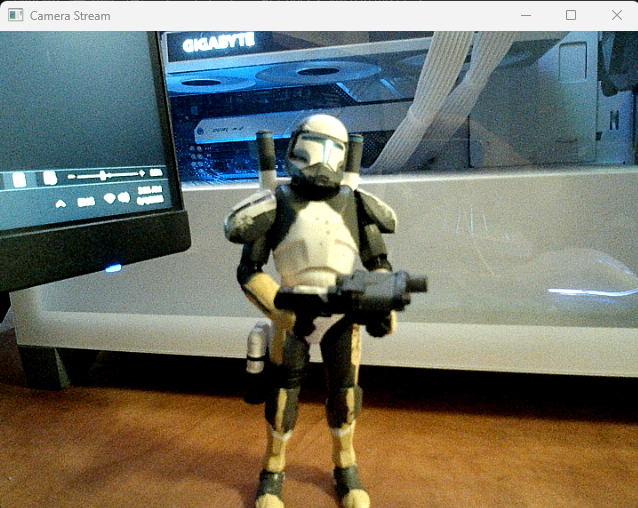
1. pip install opencv
2. pip install numpy
3. pip install pyaudio
4. pip install dtls

לאחר התקנת הספריות יש להתקין את תוכנת ה-TC.

עכשיו הכל מוכן ואפשר להריץ את הקובץ הראשי אשר נקרא "tc\_main.py" דרך ה-cmd אשר הוכוון לתיקייה אשר בה התקנתם את התוכנה. לדוגמא:



לאחר שיעלה חלון המצלמה יתחיל השידור.



התקנה והרצת השרת וה-RC

בשביל להתקין את ה-TC יש להתקין את פייתון דרך האתר הרשמי של פייתון ולהריץ לאחר מכן את הפקודות הבאות:

1. pip install opencv
2. pip install numpy
3. pip install pyaudio
4. pip install dtls
5. pip install vosk

עכשיו הכל מוכן ואפשר להריץ את הקובץ הראשי אשר נקרא "server.py" דרך ה-cmd אשר הוכוון לתיקייה אשר בה התקנתם את התוכנה. לדוגמא:



עכשיו השרת רץ ואפשר להתחיל להתחבר ל-RC.

התחברות ל-RC דרך השרת הלוקלי מתבצעת בצורה הבאה:

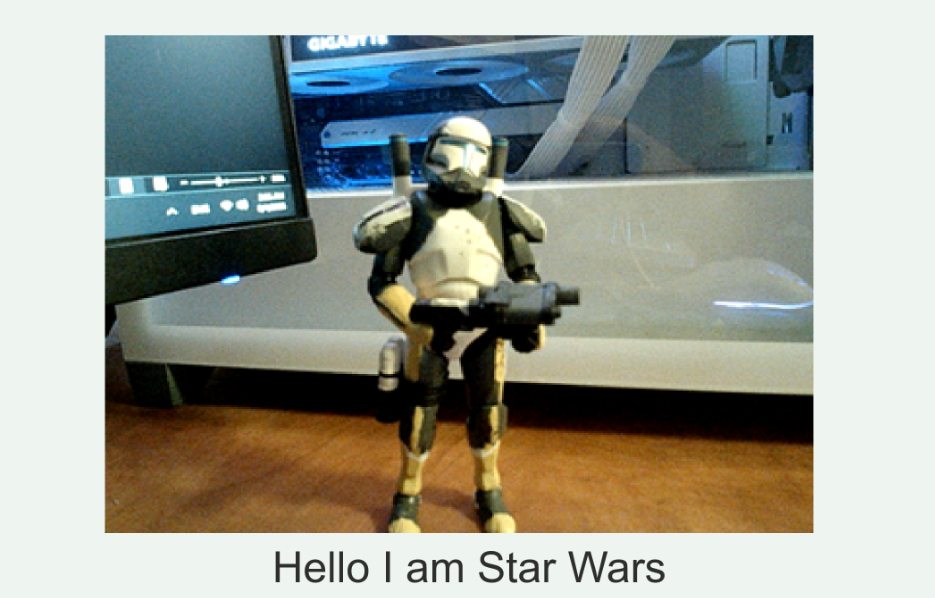
* 1. הכנסו לדפדפן שלכם והקלידו את הדבר הבא בשורת החיפוש:

****

* 1. תבחרו אם אתם מעוניינים לראות את השיחות אשר היו או להצטרף לשידור



במידה והצטרפתם לשידור השידור יוצג במסך הבא:



במידה ובחרתם לראות את השיחות יוצג המסך הבא: בו מוצגות האפשרויות להוריד את התמלול או למחוק אותו

A close-up of a date

AI-generated content may be incorrect.

רפלקציה אישית

אני התחלתי את תהליך העבודה על הפרויקט באיזור פברואר אך לצערי אני לקחתי על עצמי יותר מדי.

תוך כדי התמודדות עם הפרויקט אני הייתי צריך להקים רסיטל, לתחזק את העבודה שלי, ולהתמודד עם עוד מיליון מחויבויות שונות שהיו לי והאמת שזה היה יותר מדי.

הצבתי לעצמי לוח זמנים מפורט ופייר לפרויקט אך נכשלתי קשות בלעמוד בו והאתגר הכי גדול שלי היה לגרום לעצמי למצוא זמן לכל הדברים ביחד ובעיקר בהם הפרויקט.

היו פעמים שהצלחתי ורוב הדברים גם לקחו יותר זמן ממה שחשבתי וגם היו יותר מאתגרים (כמו מימוש הפרוטוקולים). בסופו של דבר מועד ההגשה התקרב ואני הבנתי שאני צריך לקחת את עצמי בידיים והתחלתי לעבוד יותר ברצינות והתחלתי לקבוע עם חברים לעבוד ביחד אך לצערי זה היה מאוד מאוחר ולמרות שהספקתי הרבה אני לא הספקתי מספיק.

תמיד היו לי בעיות של לדרום לעצמי להרגיע, לבחור ולשבת ולעשות משהו שמלחיץ אותי אך הפרויקט הזה הראה לי כמה הכישור של שליטה עצמית וניהול זמנים הוא חשוב וקריטי.

מן הפרויקט אני למדתי כל מיני כלים חדשים של דיבוג, ניהול זמנים, ניהול ציפיות, עבודה מסודרת ועוד. אבל הדבר שאני מוצא כהכי משמעותי הוא שלבי התכנון של הפרויקט. אני יודע שיש הרבה שמזלזלים בחשיבות של שלבי הארכטקטורה או כל שלבי התכנון ופשוט רוצים לצלול ישר לכתיבת הקוד (ואני הייתי ביניהם) אך תכנון הפרויקט הראה לי כמה התכנון הוא דבר חשוב ועוזר למען הפרויקט ולמען איך שהוא ייראה ויעבוד וזהו דבר שאני חד משמעית איישם לפרויקטי המשך ולשירות הצבאי שלי.

מבחינת יישום חלקי הפרויקט הדבר העיקרי שאני הייתי מיישם אחרת הוא התמלול. אני הייתי מטפל בו ישירות בלקוח המשדר ולא בשרת מאחר שזה יותר ישיר ויגרום לפחות בעיות סנכרון ככל הנראה במיוחד אם יתוזמן נכון בעזרת פרוטוקול ה-RTP. מעבר לזה אני מאוד מרוצה מהארכיטקטורה ואיך שהפרויקט בנוי ומאורגן.

אם היו משאבים נוספים אני הייתי מרחיב את הפרויקט לצד החומרתי ובונה מערכת שבאמת אפשר להרכיב על רכבים אמיתיים. נוסף על כך אני הייתי מוסיף אפשרות ליותר ממצלמה אחת כדי שתהיה לכל נוסע מצלמה משלו.

הפרויקט הזה הראה לי הרבה מאוד דברים על עצמי גם כאלה רעים וגם טובים ואני חושב שהוא בעיקר הראה לי נקודות לשיפור בהתנהלות שלי עם אחריות וניהול ציפיות.

ביבליוגרפיה

**WebSocket**

* **Mumtaz, W. (n.d.). *WebSocket-Video-Processing*. GitHub. Retrieved June 4, 2025, from** [**https://github.com/waqarmumtaz369/WebSocket-Video-Processing**](https://github.com/waqarmumtaz369/WebSocket-Video-Processing)
* **(n.d.). *WebSocket*. High Performance Browser Networking. Retrieved June 4, 2025, from** [**https://hpbn.co/websocket/**](https://hpbn.co/websocket/)

**RTP**

* **Wikipedia. (n.d.). *Real-time Transport Protocol*. Wikipedia. Retrieved June 4, 2025, from** [**https://he.wikipedia.org/wiki/Real-time\_Transport\_Protocol**](https://he.wikipedia.org/wiki/Real-time_Transport_Protocol)

**HTTP**

* **Wikipedia. (n.d.). *Hypertext Transfer Protocol*. Wikipedia. Retrieved June 4, 2025, from** [**https://he.wikipedia.org/wiki/Hypertext\_Transfer\_Protocol**](https://he.wikipedia.org/wiki/Hypertext_Transfer_Protocol)

**דברים שעשיתי בעזרת בינה מלאכותית:**

Microsoft copilot:

* 1. מחקר על הצפנות
  2. מחקר על RTP
  3. מחקר על WEBSOCKET
  4. דיונים על באגים
  5. התייעצות כללית
  6. התייעצות על ספר פרויקט
  7. פירוט מודולים וספריות

Gemini:

1. מחקר על HTTP
2. כתיבת ביבליוגרפיה

נספחים

**server\httpro\\_\_init\_\_.py**

"""

AUTHOR: Ophir Nevo Michrowski

DATE: 15/03/24

DESCRIPTION: HTTP protocol package. Includes (http parser and formatter, http request builder)

"""

import logging

import os.path

import httpro.constants as consts

import httpro.http\_parser

import httpro.http\_message

import httpro.app

import httpro.functions

import httpro.constants

def http\_setup() -> None:

"""

Function that contains all the auto\_checks for the http package and sets up logger

:return: None

"""

httpro.http\_parser.auto\_test\_http\_parser()

httpro.http\_message.auto\_test\_http\_message()

# is\_valid\_request auto tests #

request: bytes = b"GET /index.html HTTP/1.1\r\nHost: 192.168.37.45\r\n\r\n"

request\_invalid: bytes = b"GET /index.html HTTP/1.1\r\nHost:192.168.37.45\r\n\r\n"

assert is\_valid\_request(request)["valid"], is\_valid\_request(request)["reason"]

assert not is\_valid\_request(request\_invalid)["valid"]

# Start logger #

if not os.path.isdir(consts.LOG\_DIR):

os.makedirs(consts.LOG\_DIR)

consts.HTTP\_LOGGER.setLevel(consts.LOG\_LEVEL)

file\_handler = logging.FileHandler(consts.LOG\_FILE\_NAME)

file\_handler.setFormatter(logging.Formatter(consts.LOG\_FORMAT))

consts.HTTP\_LOGGER.addHandler(file\_handler)

consts.HTTP\_LOGGER.info("httpro initiated")

def is\_valid\_request(request: bytes) -> dict[bool, str] or dict[bool]:

"""

Checks if a http request is valid.

:param request: the request to validate.

:return dict[bool, string]: {"valid": True/False, "reason":"reason"}

"""

# Making sure that the argument is correct #

if not isinstance(request, bytes):

return {"valid": False, "reason": "request data type must be bytes"}

# Split the request into lines #

lines = request.decode('utf-8').split('\r\n')

# Check if there's at least one line (the request line) and a blank line separating headers and body #

if len(lines) < 2 or b'\r\n\r\n' not in request:

return {"valid": False, "reason": "At least one line (the request line) and a blank line separating headers "

"and body are needed."}

# Check the request line for the correct number of elements and separators #

request\_line\_parts = lines[0].split(' ')

if len(request\_line\_parts) < 3:

return {"valid": False, "reason": "Incorrect number of elements or separators."}

# Ensure method, path, and HTTP version are correctly separated #

method, path, version = request\_line\_parts[0], request\_line\_parts[1], request\_line\_parts[-1]

if method.encode() not in consts.REQUEST\_TYPESs():

return {"valid": False, "reason": f"{method} is not a real method in httpro."}

# Check if the request has a http version #

if not method or not path or not version.startswith('HTTP/'):

return {"valid": False, "reason": "Request must have httpro version."}

# Check headers for correct formatting

is\_host\_header = False

for header in lines[1:-2]: # Ignoring the request line and the last two elements (the last header and the body)

if ': ' not in header:

return {"valid": False, "reason": f"{header} header does not contain colon-space separator."}

if "Host" in header:

is\_host\_header = True # Mandatory host header is in the request

# Check if the request has mandatory host header #

if not is\_host\_header:

return {"valid": False, "reason": "Host header is mandatory."}

# Return true if the request is valid #

return {"valid": True}

def read\_file(path: str) -> http\_message:

"""

Create a request for a html page.

:param path: the path to the html.

:return http\_message: the http formatted message containing the html file.

"""

if not os.path.isfile(path):

consts.HTTP\_LOGGER.Exception("File not found.")

raise FileNotFoundError("File not found.")

else:

with open(path, 'rb') as f:

return f.read()

**server\httpro\app.py**

"""

AUTHOR: Ophir Nevo Michrowski

DATE: 15/04/24

DESCRIPTION: This is the app class. It will handle the webapp and the path.

"""

# Imports #

import socket

import time

import select

import httpro.constants as consts

import logging

import re

import httpro.http\_parser

import os

# global vars #

global readable\_socks\_list, writeable\_socks\_list, exception\_socks\_list

class App:

"""The Base for the web app."""

# ---------- Constructor ---------- #

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

"""

Constructor of class

:return: None

"""

self.routes = dict()

self.four\_o\_four = consts.FOUR\_O\_FOUR

self.\_\_closed = False

# ---------- Decorators ---------- #

def route(self, route: bytes, permission\_cookie: bytes = "None"):

"""

Route decorator - adds the route to the routes' dictionary.

:param permission\_cookie: The cookie name that allows permission to page.

:param route: the route for the uri

:return: what the original function needs to return

"""

def add\_to\_route\_dict(original\_function):

"""

This is the decorator function

:param original\_function: original function that decorator decorates

:return: what the original function needs to return

"""

try:

self.routes[route] = original\_function, permission\_cookie

except TypeError:

consts.HTTP\_LOGGER.debug("routes is not initialized.")

def wrapper\_function(\*args, \*\*kwargs):

"""

This is the wrapper function.

"""

raise Exception("This is a route function. It cannot be run.")

return wrapper\_function

return add\_to\_route\_dict

# ---------- Private functions ---------- #

@staticmethod

def \_\_receive\_message(client\_socket: socket) -> bool or httpro.http\_parser.HttpParser:

"""

Receives a message from a client

:param client\_socket:

:return bool or HttpParser: False if the message is invalid, HttpParser with the message

"""

try:

message = b""

# Multiple receives to eliminate server blocking without too many requests with the timeout #

time\_start = time.time()

while b"\r\n\r\n" not in message and time.time() - time\_start < consts.RECV\_TIMEOUT:

msg = b""

try:

client\_socket.settimeout(.5)

msg = client\_socket.recv(consts.RECV\_LENGTH)

except socket.timeout:

consts.HTTP\_LOGGER.debug("Got timeout on socket receive")

else:

consts.HTTP\_LOGGER.debug("Success on socket receive.")

if not msg:

continue

message += msg

try:

content\_length = int(re.search(rb'Content-Length: (\d+)', message).group(1))

# Check if body was already received #

if len(message.split(b'\r\n\r\n')) > 1 and len(message.split(b'\r\n\r\n')[1]) != content\_length:

body = b''

# Receiving body #

while len(body) < content\_length:

chunk = client\_socket.recv(consts.RECV\_LENGTH)

if not chunk:

consts.HTTP\_LOGGER.debug("Ended body receive")

break

body += chunk

message += body

except AttributeError:

consts.HTTP\_LOGGER.debug("The message does not have Content-Length header")

except Exception as e:

consts.HTTP\_LOGGER.exception(e)

return httpro.http\_parser.HttpParser(message)

except Exception as e:

consts.HTTP\_LOGGER.exception(e)

def \_\_handle\_client(self, request: httpro.http\_parser.HttpParser, client\_socket: socket) -> None:

"""

Handles the client input.

:param request: The request that the server got.

:param client\_socket: The client socket.

:return: None

"""

response: httpro.http\_message

if request is not None and request.URI is not None:

if request.URI in self.routes.keys() and \

(self.routes[request.URI][1] == "None" or

(request.COOKIES and self.routes[request.URI][1] in request.COOKIES.keys())):

response = self.routes[request.URI][0](request)

elif not os.path.isfile(request.URI[1:].replace(b"%20", b" ")):

with open(self.four\_o\_four, 'rb') as file:

response = httpro.http\_message.HttpMsg(error\_code=404,

headers={"content\_type": consts.MIME\_TYPES[".html"]},

body=file.read())

# If uri does not have a special path #

else:

# extract requested file type from URL (html, jpg etc)

file\_type = os.path.splitext(request.URI)[1]

# generate proper HTTP header

file\_data: bytes

with open(request.URI[1:].replace(b"%20", b" "), 'rb') as f:

file\_data = f.read()

response = httpro.http\_message.HttpMsg(headers={"content\_type": consts.MIME\_TYPES[file\_type.decode()]},

body=file\_data)

consts.HTTP\_LOGGER.info(f"Request to {request.URI} got response of {response.error\_code}")

client\_socket.send(response.build\_message\_bytes())

# ---------- Public functions ---------- #

def set\_four\_o\_four(self, route: str) -> None:

"""

Sets the route to the 404 page.

:param route: The route to the 404 page html.

:return: None

"""

if isinstance(route, str):

self.four\_o\_four = route

else:

raise TypeError("Function must get a string that holds the route to the 404 html file")

def close\_app(self) -> None:

"""

Shuts down the server and closes the server

:return: None

"""

self.\_\_closed = True

consts.HTTP\_LOGGER.debug("Closing app.")

def run(self, port=consts.PORT, host=consts.IP) -> None:

"""

Starts the http server.

:return: None

"""

global readable\_socks\_list, writeable\_socks\_list, exception\_socks\_list

# Setting up the socket #

sock = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

sock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

sock.settimeout(.5)

sock.bind((host, port))

sock.listen()

socket\_list = [sock]

try:

while not self.\_\_closed:

readable\_socks\_list, writeable\_socks\_list, exception\_socks\_list = select.select(socket\_list,

socket\_list,

socket\_list,

.1)

for notified\_socket in readable\_socks\_list:

if notified\_socket == sock: # checking for new connection #

consts.HTTP\_LOGGER.debug("Getting new connection")

client\_socket, client\_addr = sock.accept()

self.logger.info(consts.NEW\_CLIENT.format(client\_addr[0], client\_addr[1]))

client\_socket.settimeout(.5)

socket\_list.append(client\_socket) # Adding the socket to the connected clients #

else:

try:

consts.HTTP\_LOGGER.debug("Starting receive")

message: httpro.http\_parser.HttpParser = self.\_\_receive\_message(notified\_socket)

consts.HTTP\_LOGGER.info(b"Got Request: " + message.URI if message.URI else "None")

self.\_\_handle\_client(message, notified\_socket)

except Exception as e:

consts.HTTP\_LOGGER.exception(e)

finally:

notified\_socket.close()

socket\_list.remove(notified\_socket)

finally:

consts.HTTP\_LOGGER.debug("Closing main socket.")

sock.close()

consts.HTTP\_LOGGER.debug("Main socket closed.")

**server\httpro\constants.py**

"""

AUTHOR: Ophir Nevo Michrowski

DATE: 15/03/24

DESCRIPTION: This file contains all the needed constants for the HTTP packet

"""

import logging

# HTTP necessities #

HTTP\_VERSION = b"HTTP/1.1"

HEADER\_SEPERATOR = b"\r\n"

BODY\_SEPERATOR = HEADER\_SEPERATOR \* 2

# Other #

RECV\_TIMEOUT = 3

# Logging #

LOG\_LEVEL = self.logger.debug

LOG\_DIR = r"Logs"

LOG\_FORMAT = "%(asctime)s | %(levelname)s | %(message)s"

LOG\_FILE\_NAME = LOG\_DIR + r"\http\_logs.log"

HTTP\_LOGGER\_NAME = "http\_log"

HTTP\_LOGGER = logging.getLogger(HTTP\_LOGGER\_NAME)

# Headers #

LOCATION\_HEADER = b"Location: %s"

CONTENT\_TYPE\_HEADER = b"Content-Type: %s"

CONTENT\_LEN\_HEADER = b"Content-Length: %d"

HOST\_HEADER = b"Host: %s"

# Dictionaries #

REQUEST\_TYPES = {

"get": b"GET",

"post": b"POST",

"put": b"PUT",

"delete": b"DELETE"

}

ERROR\_CODES = {

200: b"OK",

500: b"INTERNAL SERVER ERROR",

302: b"MOVED TEMPORARILY",

403: b"FORBIDDEN",

404: b"NOT FOUND",

400: b"BAD REQUEST",

301: b"MOVED PERMANENTLY",

401: b"UNAUTHORIZED",

405: b"METHOD NOT ALLOWED",

413: b"PAYLOAD TOO LARGE",

502: b"BAD GATEWAY",

503: b"SERVICE UNAVAILABLE"

}

MIME\_TYPES = {

".html": "text/html;charset=utf-8",

".jpg": "image/jpeg",

".css": "text/css",

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

".txt": "text/plain",

".ico": "image/x-icon",

".gif": "image/jpeg",

".png": "image/png",

".ttf": "font/ttf",

"sse": "text/event-stream"

}

# Socket Related #

RECV\_LENGTH = 1024

PORT = 80

IP = "0.0.0.0"

# Logging level - info #

NO\_BODY = "Message has no body."

NEW\_CLIENT = "{}:{} connected."

TESTS\_RUN = "Auto tests were run."

FOUR\_O\_FOUR = "httpro/html\_defaults/404.html"

**server\httpro\functions.py**

"""

AUTHOR: Ophir Nevo Michrowski

DATE: 16/03/24

DESCRIPTION: Useful functions on dictionaries

"""

def dict\_to\_bytes(dictionary: dict) -> dict[bytes]:

"""

Takes a dictionary and casts all the elements within to bytes.

EXAMPLE: {"name": "John", "age": 45} -> {b"name": b"John", b"age": b"45"}

:param dictionary: the dict to convert

:return dict[bytes]: the converted dictionary .

"""

if dictionary:

return {(key.encode('utf-8') if isinstance(key, str) else str(key).encode('utf-8')):

(value.encode('utf-8') if isinstance(value, str) else str(value).encode('utf-8'))

for key, value in dictionary.items()}

else:

return dict()

**server\httpro\http\_message.py**

"""

AUTHOR: Ophir Nevo Michrowski

DATE: 15/03/24

DESCRIPTION: Class that allows to create easy to use http\_ophir messages including responses and requests

"""

# Imports #

import httpro.constants as consts

import httpro.functions

class HttpMsg:

"""Create easy to use http messages including responses and requests"""

def \_\_init\_\_(self, error\_code: int = 200, body: bytes = b"", \*\*headers) -> None:

"""

The constructor of the HttpMsg class.

:param error\_code: The error code of the message.

:param body: The body of the message.

:param headers: The headers of the message - host=127.0.0.1 -> { b"host": b"127.0.0.1"}

:return: None

"""

if headers is None:

headers = dict()

self.error\_code = self.\_\_error\_code\_finder(error\_code)

self.body = body

self.headers = httpro.functions.dict\_to\_bytes(headers)

if body != b"":

self.headers[b"content\_length"] = str(len(body)).encode()

@staticmethod

def \_\_error\_code\_finder(error\_code: int) -> bytes:

"""

Returns the error code and message using the error code.

:param error\_code: The error code to be found

:return bytes: The error code

"""

return\_value = b"-1"

if error\_code in consts.ERROR\_CODES.keys():

try:

return\_value = str(error\_code).encode() + b" " + consts.ERROR\_CODES[error\_code]

except KeyError:

consts.HTTP\_LOGGER.exception(f"{error\_code} is not valid")

raise KeyError(f"{error\_code} is not valid")

except Exception:

consts.HTTP\_LOGGER.exception(f"error while finding error code")

return return\_value

def prettify(self) -> None:

"""

Prints the message in a readable detailed format

:return: None

"""

print(f"## HTTP VERSION ##\n{consts.HTTP\_VERSION.decode('utf-8')}\n## ERROR CODE ##\n" +

f"{self.error\_code.decode('utf-8')}\n## HEADERS ##\n{self.\_\_build\_headers\_bytes().decode('utf-8')}\n" +

f"## BODY ##\n{self.body.decode('utf-8')}")

def \_\_build\_headers\_bytes(self) -> bytes:

"""

Formats the headers into bytes

:return bytes: Byte string with the headers formatted

"""

headers = b""

if self.headers.items():

for key, value in self.headers.items():

headers += key.title().replace(b'\_', b'-') + b": " + value + consts.HEADER\_SEPERATOR

else:

consts.HTTP\_LOGGER.debug("no headers found")

return headers

def build\_message\_bytes(self) -> bytes:

"""

Builds the http\_ophir message

:return bytes: The http\_ophir message

"""

headers = self.\_\_build\_headers\_bytes()

return consts.HTTP\_VERSION + b" " + self.error\_code + consts.HEADER\_SEPERATOR + headers + \

consts.HEADER\_SEPERATOR + self.body

def \_\_str\_\_(self) -> str:

"""

Str dunder function for the HttpMsg class

:return str: The http\_ophir message in full

"""

return self.build\_message\_bytes().decode('utf-8')

def auto\_test\_http\_message() -> None:

"""

Auto test for the HttpMsg class

:return: None

"""

http\_request\_example = b"HTTP/1.1 200 OK\r\nContent-Type: text/plain\r\nContent-Length: 5\r\n\r\nhello"

http\_message\_test = HttpMsg(body=b"hello", content\_type=consts.MIME\_TYPES[".txt"])

assert http\_message\_test.build\_message\_bytes() == http\_request\_example

assert http\_message\_test.headers == {b'content\_type': b'text/plain', b'content\_length': b'5'}

assert str(http\_message\_test) == http\_request\_example.decode()

assert http\_message\_test.body == b"hello"

assert http\_message\_test.error\_code == b"200 OK"

**server\httpro\http\_parser.py**

"""

AUTHOR: Ophir Nevo Michrowski

DATE: 15/03/24

DESCRIPTION: Takes a received http\_ophir request and formats it into a structure that is easy to use in code

"""

# Imports #

import httpro.constants as consts

import httpro

# HttpParser #

class HttpParser:

"""Class to make http requests usable in code more easily"""

def \_\_init\_\_(self, http\_request: bytes) -> None:

"""

The constructor of the HttpParser class.

:param http\_request: the request for the class to parse.

:return: None

"""

# If the request is invalid #

is\_request\_valid = httpro.is\_valid\_request(http\_request)

if not is\_request\_valid["valid"] or not http\_request:

self.HTTP\_REQUEST = is\_request\_valid["reason"]

self.METHOD = None

self.URI = None

self.HTTP\_VERSION = None

self.QUERY\_PARAMS = None

self.HEADERS = None

self.BODY = None

self.COOKIES = None

# If the request is valid #

else:

self.HTTP\_REQUEST = http\_request

self.HEADERS = self.\_\_header\_parser()

self.COOKIES = dict() if b"Cookie" in self.HEADERS.keys() else None

if b"Cookie" in self.HEADERS.keys():

if len(self.HTTP\_REQUEST.split(b";")) > 1:

tmp = [x.split(b"=") for x in self.HEADERS[b"Cookie"].split(b"; ")]

self.COOKIES = dict(tmp)

else:

key, value = self.HTTP\_REQUEST.split(b"=")

self.COOKIES[key] = value

self.BODY = self.\_\_body\_parser()

self.METHOD, self.URI, self.HTTP\_VERSION, self.QUERY\_PARAMS = self.\_\_section\_one\_parser()

# Whether the request is valid #

self.IS\_VALID: bool = is\_request\_valid["valid"]

def \_\_header\_parser(self) -> dict[bytes]:

"""

Extracts the headers from the request

:return dict[bytes]: {header: value}

"""

try:

return dict(x.split(b": ") for x in self.HTTP\_REQUEST.split(consts.HEADER\_SEPERATOR)[1:-2])

except IndexError:

consts.HTTP\_LOGGER.debug("Request has no headers.")

def \_\_body\_parser(self) -> bytes:

"""

Extracts the body from the requests

:return bytes: the body of the request

"""

return self.HTTP\_REQUEST.split(consts.BODY\_SEPERATOR)[1]

def \_\_section\_one\_parser(self) -> tuple[bytes, bytes, bytes, dict[bytes] or None]:

"""

Splits the first section of the http\_ophir request (before the headers)

:return tuple: (method, uri, http\_version, query\_params)

"""

request = self.HTTP\_REQUEST.split(b' ')

def \_\_query\_params\_parser() -> dict[bytes] or None:

"""

Extract the parameters from the http\_ophir request

:return: dictionary of all the query parameters

"""

try:

return dict(

x.split(b" ")[0].split(b"=", 1)[0:2] for x in request[1].split(b"?", 1)[1].split(b"&", 1))

# If there are no params #

except IndexError:

consts.HTTP\_LOGGER.error("no query parameters")

return None

except Exception as e:

consts.HTTP\_LOGGER.exception(e)

return request[0], request[1], request[2], \_\_query\_params\_parser()

def \_\_str\_\_(self) -> str:

"""

Str dunder function for the HttpMsg class

:return str: The http\_ophir message in full

"""

return self.HTTP\_REQUEST.decode('utf-8')

def auto\_test\_http\_parser() -> None:

"""

Automatic tests for the HttpParser class.

:return: None

"""

# Valid HTTP request with query parameters

valid\_request = b"GET /index.html?param1=value1&param2=value2 HTTP/1.1\r\nHost: example.com\r\n\r\n"

parser\_valid = HttpParser(valid\_request)

assert parser\_valid.IS\_VALID

assert parser\_valid.HTTP\_REQUEST == valid\_request

assert parser\_valid.QUERY\_PARAMS == {b"param1": b"value1", b"param2": b"value2"}

assert parser\_valid.HEADERS == {b"Host": b"example.com"}

assert parser\_valid.BODY == b""

# Invalid HTTP request

parser\_invalid = HttpParser(b"GET /index.html HTTP/1.1\r\nHost: example.com\r\nInvalid-Header\r\n\r\n")

assert not parser\_invalid.IS\_VALID

assert not parser\_invalid.QUERY\_PARAMS

assert not parser\_invalid.HEADERS

assert not parser\_invalid.BODY

**server\server.py**

"""

main file of the transmitting client

"""

import threading

from transmitting\_client\_handler.tc\_handle import start\_tc\_handle

def main() -> None:

"""

Starts streaming

:return: None

"""

tc\_thread = threading.Thread(target=start\_tc\_handle)

tc\_thread.start()

tc\_thread.join()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**server\transcriptor\transcript.py**

import json

import vosk

from src.transmitting\_client.audio\_capture import AudioCapture

from src.server.utils.logger import Logger

class Transcriptor:

"""

Handles live speech recognition and captioning using AudioCapture.

"""

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

"""

Initializes the speech-to-text engine using Vosk and sets up audio capture.

"""

self.model = vosk.Model(r"C:\Users\ophir\Downloads\vosk-model-small-en-us-0.15") # Make sure you have a Vosk model downloaded

self.recognizer = vosk.KaldiRecognizer(self.model, 16000) # Match sample rate

self.logger = Logger("Transcriptor").logger

# Initialize direct audio capture

self.audio\_capture = AudioCapture(self.logger)

def transcribe\_audio(self) -> None:

"""

Continuously captures and transcribes live audio.

:return: None

"""

self.logger.debug("Transcribing live audio... Speak now!")

while True:

try:

audio\_data = self.audio\_capture.get\_audio\_chunk()

if not audio\_data:

continue

# Process audio with Vosk

if self.recognizer.AcceptWaveform(audio\_data):

result = json.loads(self.recognizer.Result())

self.logger.debug(f"Caption: {result['text']}") # Display live transcription

except KeyboardInterrupt:

self.logger.debug("Stopping transcription...")

break

self.audio\_capture.release()```

## `server\transmitting\_client\_handler\audio\_receive\_handler.py`

```python

"""

Audio Handling

"""

# Imports #

import numpy as np

import pyaudio

import time

from src.server.transmitting\_client\_handler.udp\_handler\_generic import UDPServerHandler

from src.server.utils.consts.tc\_consts import Ports, AudioConsts

class AudioReceiveHandler:

"""

Handles real-time UDP audio reception with delta time synchronization.

"""

@staticmethod

def recv\_audio() -> None:

"""

Receives RTP audio packets and plays audio in real-time using delta time for better timing.

:return: None

"""

udp\_handler = UDPServerHandler(Ports.AUDIO\_PORT) # Use correct port

audio = pyaudio.PyAudio()

stream = audio.open(format=AudioConsts.FORMAT,

channels=AudioConsts.CHANNELS,

rate=AudioConsts.SAMPLE\_RATE,

output=True,

frames\_per\_buffer=AudioConsts.CHUNK\_SIZE)

print("Receiving Audio... Press Ctrl+C to stop.")

prev\_time = time.time() # Track last packet timestamp

expected\_interval = 0.02 # 20ms expected interval for real-time audio

while True:

try:

audio\_data = udp\_handler.receive\_rtp\_message()

if not audio\_data:

continue

current\_time = time.time()

delta\_time = current\_time - prev\_time # Time difference since last packet

# Adjust playback timing dynamically

if delta\_time < expected\_interval:

time.sleep(expected\_interval - delta\_time) # Compensate for timing drift

# Convert received data into NumPy array

audio\_array = np.frombuffer(audio\_data, dtype=np.int16)

# Play received audio

stream.write(audio\_array.tobytes())

prev\_time = current\_time # Update last received packet time

except KeyboardInterrupt:

break

# Cleanup

stream.stop\_stream()

stream.close()

audio.terminate()

print("Audio reception stopped.")

if \_\_name\_\_ == '\_\_main\_\_':

AudioReceiveHandler.recv\_audio()

**server\transmitting\_client\_handler\rtp\_parser.py**

"""

This file contains the RTPPacketDecoder class, which handles incoming RTP packets and parses them into a usable object.

"""

# Imports #

import struct

from src.server.utils.logger import Logger

class RTPPacketDecoder:

"""

Decodes RTP packets into header fields and payload.

"""

def \_\_init\_\_(self, packet: bytes) -> None:

"""

Initializes the RTPPacketDecoder instance with the given RTP packet.

:param packet: (bytes) The raw RTP packet.

"""

self.packet = packet

self.version = None

self.padding = None

self.extension = None

self.csrc\_count = None

self.marker = None

self.payload\_type = None

self.sequence\_number = None

self.timestamp = None

self.ssrc = None

self.csrcs = []

self.extension\_data = None

self.payload = None

self.logger = Logger("rtp-logger").logger

self.decode\_packet()

def decode\_packet(self) -> None:

"""

Decodes the RTP packet, extracting its header and payload.

:return: None

"""

try:

if len(self.packet) < 12:

raise ValueError("RTP packet too short to contain a valid header.")

# Parse the fixed header (12 bytes)

header = struct.unpack('!BBHII', self.packet[:12])

self.version = (header[0] >> 6) & 0b11

self.padding = (header[0] >> 5) & 0b1

self.extension = (header[0] >> 4) & 0b1

self.csrc\_count = header[0] & 0b1111

self.marker = (header[1] >> 7) & 0b1

self.payload\_type = header[1] & 0b01111111

self.sequence\_number = header[2]

self.timestamp = header[3]

self.ssrc = header[4]

# Calculate header size and extract CSRCs if present

header\_size = 12

if self.csrc\_count > 0:

csrc\_end = header\_size + 4 \* self.csrc\_count

if len(self.packet) < csrc\_end:

raise ValueError("RTP packet too short to contain all CSRCs.")

self.csrcs = struct.unpack(f'!{self.csrc\_count}I', self.packet[header\_size:csrc\_end])

header\_size = csrc\_end

# Handle extension header if present

if self.extension:

if len(self.packet) < header\_size + 4:

raise ValueError("RTP packet too short to contain the extension header.")

ext\_header = struct.unpack('!HH', self.packet[header\_size:header\_size + 4])

ext\_length = ext\_header[1] \* 4 # Length in 32-bit words

ext\_end = header\_size + 8 + ext\_length

if len(self.packet) < ext\_end:

raise ValueError("RTP packet too short to contain extension data.")

self.extension\_data = self.packet[header\_size + 8:ext\_end]

header\_size = ext\_end

# Extract payload

self.payload = self.packet[header\_size:]

self.logger.info("RTP packet successfully decoded.")

except struct.error as e:

self.logger.exception("Failed to unpack RTP packet: %s", e)

raise ValueError("Invalid RTP packet structure.") from e

except Exception as e:

self.logger.exception("Error decoding RTP packet: %s", e)

raise

def get\_header\_info(self) -> dict:

"""

Returns the RTP header fields as a dictionary.

:return: dict: Header fields with their values.

"""

return {

"version": self.version,

"padding": self.padding,

"extension": self.extension,

"csrc\_count": self.csrc\_count,

"marker": self.marker,

"payload\_type": self.payload\_type,

"sequence\_number": self.sequence\_number,

"timestamp": self.timestamp,

"ssrc": self.ssrc,

"csrcs": self.csrcs,

"extension\_data": self.extension\_data,

}

def get\_payload(self) -> bytes:

"""

Returns the payload from the RTP packet.

:return: bytes: The raw payload data.

"""

return self.payload

**server\transmitting\_client\_handler\tc\_handle.py**

"""

main file of the transmitting client

"""

import threading

from audio\_receive\_handler import AudioReceiveHandler

from video\_receive\_handler import VideoReceiveHandler

def start\_tc\_handle() -> None:

"""

Starts streaming

:return: None

"""

# Create processes

video\_thread = threading.Thread(target=VideoReceiveHandler.recv\_video())

audio\_thread = threading.Thread(target=AudioReceiveHandler.recv\_audio())

# Start processes

video\_thread.start()

audio\_thread.start()

# Keep the main program running

video\_thread.join()

audio\_thread.join()

**server\transmitting\_client\_handler\udp\_handler.py**

"""

Will hold the class in charge of communications

"""

# Imports #

import logging

import zlib

import socket

import struct

import cv2

import numpy as np

from src.server.transmitting\_client\_handler.rtp\_parser import RTPPacketDecoder

from src.server.utils.consts.logging\_consts import \*

from src.server.utils.consts.tc\_consts import CommunicationConsts

from src.server.utils.logger import Logger

class UDPServerHandler:

"""

This class represents the video receiving server using UDP.

It initializes a UDP socket to listen for incoming packets and

uses RTPacketDecoder to decode the received RTP packets.

"""

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

"""

Initializes the Server instance.

:return: None

"""

self.bind\_address = CommunicationConsts.HOST

self.bind\_port = CommunicationConsts.PORT

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

self.sock.bind((self.bind\_address, self.bind\_port))

self.\_uncompleted\_frame\_packets = dict()

self.sock.settimeout(CommunicationConsts.FRAGMENT\_RECEIVE\_TIMEOUT)

self.logger = Logger("UDP-logger").logger

self.logger.info(SuccessMessages.SERVER\_LISTENING.format(self.bind\_address, self.bind\_port))

def \_receive\_packet(self, buffer: int) -> bytes | None:

"""

Will receive a packet using a UDP connection

:param buffer: (int) the buffer to receive

:return bytes | None: The received packet if one was received

"""

packet = b''

try:

packet, \_ = self.sock.recvfrom(buffer)

except socket.timeout:

self.logger.debug("Got a None packet")

return None

except socket.error as e:

self.logger.exception("NETWORK ERROR: ", e)

return packet

def assemble\_frame(self, seq\_start: int, seq\_end: int) -> bytes | None:

"""

Assembles / Discards frames

:param seq\_start: (int) fragment sequence start

:param seq\_end: (int) fragment sequence end

:return bytes | None: bytes if a frame was assembled, None if frame was dropped

"""

frame = b''

for seq in range(seq\_start, seq\_end + 1):

packet = self.\_uncompleted\_frame\_packets.get(seq)

if not packet:

self.\_uncompleted\_frame\_packets = dict()

return None

assert isinstance(packet, RTPPacketDecoder)

frame += packet.payload

self.\_uncompleted\_frame\_packets = dict()

return zlib.decompress(frame)

def receive\_video(self):

"""

Receives RTP packets from the client, decodes them, and displays the video frames.

"""

fps = 0

import time

start = time.time()

while True:

try:

packet = self.\_receive\_packet(CommunicationConsts.BUFFER\_SIZE)

if not packet:

self.logger.debug("Got a None Packet")

continue

else:

decoded\_packet = RTPPacketDecoder(packet)

self.logger.debug("Got a good packet")

if len(self.\_uncompleted\_frame\_packets) > 0 and \

next(iter(self.\_uncompleted\_frame\_packets.values())).timestamp < decoded\_packet.timestamp:

self.\_uncompleted\_frame\_packets = dict()

self.\_uncompleted\_frame\_packets[decoded\_packet.sequence\_number] = decoded\_packet

if decoded\_packet.marker != CommunicationConsts.EXPECT\_ANOTHER\_FRAGMENT:

seq\_start = decoded\_packet.sequence\_number - \

struct.unpack('!I', decoded\_packet.extension\_data)[0] + 1

frame\_data = self.assemble\_frame(seq\_start, decoded\_packet.sequence\_number)

if frame\_data is None:

continue

array = np.frombuffer(frame\_data, dtype=np.uint8)

# frame = array.reshape((480, 640, 3))

frame = np.resize(array, (240, 320, 3))

frame = cv2.resize(frame, (640, 480))

# Display the frame using OpenCV

cv2.imshow("Received Video", frame)

# Count fps #

fps += 1

if time.time() - start >= 1:

self.logger.info(f"FPS: {fps}")

fps = 0

start = time.time()

# Break the loop if 'q' is pressed

if cv2.waitKey(1) & 0xFF == ord('q'):

break

except socket.error as e:

self.logger.exception(ErrorMessages.VIDEO\_RECEIVING\_ERROR, e)

break

# Cleanup OpenCV windows after exiting the loop

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

UDPServerHandler().receive\_video()

**server\transmitting\_client\_handler\udp\_handler\_generic.py**

"""

Will hold the class in charge of communications

"""

# Imports #

import zlib

import socket

import struct

from src.server.transmitting\_client\_handler.rtp\_parser import RTPPacketDecoder

from src.server.utils.consts.logging\_consts import \*

from src.server.utils.consts.tc\_consts import CommunicationConsts, Ports

from src.server.utils.logger import Logger

class UDPServerHandler:

"""

This class represents the video receiving server using UDP.

It initializes a UDP socket to listen for incoming packets and

uses RTPacketDecoder to decode the received RTP packets.

"""

def \_\_init\_\_(self, port: Ports):

"""

Initializes the Server instance.

:return: None

"""

self.bind\_address = CommunicationConsts.HOST

self.bind\_port = port.value

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

self.sock.bind((self.bind\_address, self.bind\_port))

self.\_uncompleted\_frame\_packets = dict()

self.sock.settimeout(CommunicationConsts.FRAGMENT\_RECEIVE\_TIMEOUT)

self.logger = Logger("UDP-logger").logger

self.logger.info(SuccessMessages.SERVER\_LISTENING.format(self.bind\_address, self.bind\_port))

def \_receive\_packet(self, buffer: int) -> bytes | None:

"""

Will receive a packet using a UDP connection

:param buffer: (int) the buffer to receive

:return bytes | None: The received packet if one was received

"""

packet = b''

try:

packet, \_ = self.sock.recvfrom(buffer)

except socket.timeout:

self.logger.debug("Got a None packet")

return None

except socket.error as e:

self.logger.exception("NETWORK ERROR: ", e)

return packet

def assemble\_rtp\_message(self, seq\_start: int, seq\_end: int) -> bytes | None:

"""

Assembles / Discards frames

:param seq\_start: (int) fragment sequence start

:param seq\_end: (int) fragment sequence end

:return bytes | None: bytes if a frame was assembled, None if frame was dropped

"""

frame = b''

for seq in range(seq\_start, seq\_end + 1):

packet = self.\_uncompleted\_frame\_packets.get(seq)

if not packet:

self.\_uncompleted\_frame\_packets = dict()

return None

assert isinstance(packet, RTPPacketDecoder)

frame += packet.payload

self.\_uncompleted\_frame\_packets = dict()

return zlib.decompress(frame)

def receive\_rtp\_message(self) -> bytes or None:

"""

Receives RTP packets from the client & decodes them.

:return bytes or None: If a message/packet is received then the payload will be returned.

"""

message\_data: None or bytes = None

while True:

try:

packet = self.\_receive\_packet(CommunicationConsts.BUFFER\_SIZE)

if not packet:

self.logger.debug("Got a None Packet")

continue

else:

decoded\_packet = RTPPacketDecoder(packet)

self.logger.debug("Got a good packet")

if len(self.\_uncompleted\_frame\_packets) > 0 and \

next(iter(self.\_uncompleted\_frame\_packets.values())).timestamp < decoded\_packet.timestamp:

self.\_uncompleted\_frame\_packets = dict()

self.\_uncompleted\_frame\_packets[decoded\_packet.sequence\_number] = decoded\_packet

if decoded\_packet.marker != CommunicationConsts.EXPECT\_ANOTHER\_FRAGMENT:

seq\_start = decoded\_packet.sequence\_number - \

struct.unpack('!I', decoded\_packet.extension\_data)[0] + 1

message\_data = self.assemble\_rtp\_message(seq\_start, decoded\_packet.sequence\_number)

if message\_data:

break

except socket.error as e:

self.logger.exception(ErrorMessages.VIDEO\_RECEIVING\_ERROR, e)

break

return message\_data

**server\transmitting\_client\_handler\video\_receive\_handler.py**

"""

This file holds the VideoReceiveHandler class

"""

# Imports #

import cv2

import numpy as np

from src.server.transmitting\_client\_handler.udp\_handler\_generic import UDPServerHandler

from src.server.utils.consts.tc\_consts import Ports

class VideoReceiveHandler:

"""

Handles real-time UDP video reception

"""

@staticmethod

def recv\_video() -> None:

"""

Receives video and presents it

:return: None

"""

udp\_handler = UDPServerHandler(Ports.VIDEO\_PORT)

while True:

frame = udp\_handler.receive\_rtp\_message()

array = np.frombuffer(frame, dtype=np.uint8)

# frame = array.reshape((480, 640, 3))

frame = np.resize(array, (240, 320, 3))

frame = cv2.resize(frame, (640, 480))

# Display the frame using OpenCV

cv2.imshow("Received Video", frame)

# Break the loop if 'q' is pressed

if cv2.waitKey(1) & 0xFF == ord('q'):

break

# Cleanup OpenCV windows after exiting the loop

cv2.destroyAllWindows()

if \_\_name\_\_ == '\_\_main\_\_':

VideoReceiveHandler.recv\_video()

**server\utils\consts\logging\_consts.py**

"""

Global Logging Messages for the server

"""

# Imports #

import logging

class LoggerConsts:

FORMAT = "%(asctime)s - %(levelname)s - %(message)s"

LOG\_LEVEL = logging.DEBUG

LOG\_DIR = r"../logs/"

LOG\_FILE\_EXTENSION = ".log"

class SuccessMessages:

"""

Enum to define success log messages.

"""

SERVER\_LISTENING = "Server is now listening on {}:{}"

PACKET\_SENT = "RTP packet successfully sent to the server."

PACKET\_RECEIVED = "RTP packet successfully received by the server."

VIDEO\_STREAM\_STARTED = "Video streaming has started successfully."

CONNECTION\_ESTABLISHED = "Connection successfully established with {}:{}"

class ErrorMessages:

"""

Enum to define error log messages.

"""

VIDEO\_TRANSMISSION\_ERROR = "An error occurred while transmitting video: {}"

VIDEO\_RECEIVING\_ERROR = "An error occurred while receiving video: {}"

DECODE\_PACKET\_ERROR = "Failed to decode the RTP packet: {}"

PAYLOAD\_EXTRACTION\_ERROR = "Error extracting payload from the RTP packet: {}"

SOCKET\_CREATION\_ERROR = "Error creating the UDP socket: {}"

CONNECTION\_ERROR = "Error while establishing connection to the server: {}"

VIDEO\_STREAM\_ERROR = "Error occurred during video streaming: {}"

**server\utils\consts\tc\_consts.py**

"""

Holds the consts for the udp server

"""

from enum import Enum

from pyaudio import paInt16

# Communication Consts #

class CommunicationConsts:

PORT = 5004

HOST = '127.0.0.1'

PAYLOAD\_TYPE = 32 # uncompressed video streams

BUFFER\_SIZE = 65535 # Max UDP packet

EXPECT\_ANOTHER\_FRAGMENT = 0

FRAGMENT\_RECEIVE\_TIMEOUT = .1 # In Seconds

class Ports(Enum):

"""

All available Ports

"""

VIDEO\_PORT = 5004

AUDIO\_PORT = 5006

# Audio Consts #

class AudioConsts:

CHANNELS = 1

SAMPLE\_RATE = 44100

CHUNK\_SIZE = 512

FORMAT = paInt16

**server\utils\logger.py**

"""

Holds a generic logger class to handle logs.

"""

# Imports #

import logging

import os

from src.server.utils.consts.logging\_consts import LoggerConsts

class Logger:

"""

Handles all logs

"""

def \_\_init\_\_(self, logger\_name: str) -> None:

"""

Initialize logger

:param logger\_name: (str) the name of the logger

:return: None

"""

self.format = LoggerConsts.FORMAT

self.level = LoggerConsts.LOG\_LEVEL

self.file\_name = logger\_name + LoggerConsts.LOG\_FILE\_EXTENSION

self.file\_path = LoggerConsts.LOG\_DIR + self.file\_name

if not os.path.isdir(LoggerConsts.LOG\_DIR):

os.makedirs(LoggerConsts.LOG\_DIR)

print("created dir")

self.logger = logging.getLogger(logger\_name)

self.logger.setLevel(self.level)

file\_handler = logging.FileHandler(self.file\_path)

file\_handler.setFormatter(logging.Formatter(self.format))

self.logger.addHandler(file\_handler)

**transmitting\_client\audio\_capture.py**

"""

This file contains the AudioCapture class

"""

# Imports #

from utils.consts import AudioCaptureConsts

import pyaudio

import logging

class AudioCapture:

""" Handles real-time audio capture and streaming """

def \_\_init\_\_(self, logger: logging.Logger) -> None:

"""

Initializes the audio capture instance.

:return: None

"""

self.rate = AudioCaptureConsts.SAMPLE\_RATE

self.channels = AudioCaptureConsts.CHANNELS

self.chunk\_size = AudioCaptureConsts.CHUNK\_SIZE

self.logger = logger

try:

self.audio = pyaudio.PyAudio()

self.stream = self.audio.open(format=pyaudio.paInt16,

channels=self.channels,

rate=self.rate,

input=True,

frames\_per\_buffer=self.chunk\_size)

self.logger.info("Audio capture initialized successfully.")

except Exception as e:

self.logger.exception(f"Error initializing audio capture: {e}")

raise

def get\_audio\_chunk(self) -> bytes:

"""

Captures and returns a single chunk of audio data.

:return bytes: Raw audio data.

"""

try:

data = self.stream.read(self.chunk\_size, exception\_on\_overflow=False)

return data

except Exception as e:

self.logger.exception(f"Error capturing audio chunk: {e}")

return b""

def release(self) -> None:

"""

Cleans up resources.

:return: None

"""

self.stream.stop\_stream()

self.stream.close()

self.audio.terminate()

self.logger.info("Audio capture released.")

**transmitting\_client\audio\_transmission\_handler.py**

import time

from audio\_capture import AudioCapture

from rtp\_handler import RTPHandler

from udp\_handler import UDPClientHandler

from utils.consts import Ports

from utils.logger import Logger

from utils.payload\_types import PayloadTypes

class AudioTransmissionHandler:

"""

Handles real-time audio transmission via RTP over UDP with delta time synchronization.

"""

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

"""

Initializes the full audio streaming pipeline.

"""

self.logger = Logger("audio-logger").logger

self.audio\_capture = AudioCapture(self.logger)

self.rtp\_handler = RTPHandler(PayloadTypes.AUDIO)

self.udp\_handler = UDPClientHandler(Ports.AUDIO\_PORT.value)

self.logger.info("AudioTransmissionHandler initialized.")

def start\_streaming(self) -> None:

"""

Begins live audio transmission with delta time control for better synchronization.

:return: None

"""

self.logger.info("Starting live audio stream...")

prev\_send\_time = time.time() # Track last packet send time

send\_interval = 0.02 # 20ms expected interval for real-time audio

while True:

try:

# Capture live audio chunk

audio\_chunk = self.audio\_capture.get\_audio\_chunk()

# Wrap chunk in RTP packets

rtp\_packets = self.rtp\_handler.create\_packets(audio\_chunk)

# Compute delta time

current\_time = time.time()

delta\_time = current\_time - prev\_send\_time

# Adjust sending rate dynamically

if delta\_time < send\_interval:

time.sleep(send\_interval - delta\_time) # Smooth out transmission timing

# Send packets via UDP

self.udp\_handler.send\_packets(rtp\_packets)

prev\_send\_time = current\_time # Update last sent timestamp

except Exception as e:

self.logger.exception(f"Error during audio transmission: {e}")

break

self.stop\_streaming()

def stop\_streaming(self) -> None:

"""

Cleans up resources after transmission stops.

:return: None

"""

self.logger.info("Stopping audio transmission...")

self.audio\_capture.release()

if \_\_name\_\_ == '\_\_main\_\_':

AudioTransmissionHandler().start\_streaming()

**transmitting\_client\rtp\_handler.py**

"""

This file holds the RTP handler class -

"""

# Imports #

import math

import random

import struct

import time

import zlib

from utils import consts

from utils.payload\_types import PayloadTypes

from utils.logger import Logger

class RTPHandler:

"""

This class handles the RTP protocol, including creating and parsing RTP packets.

"""

def \_\_init\_\_(self, payload\_type: PayloadTypes, start\_timestamp=(int(time.time() \* 1000) % (2 \*\* 32))) -> None:

"""

Initializes the RTPHandler instance.

:param start\_timestamp: (int) The start\_timestamp for the object

:param payload\_type: (PayloadTypes) The payload type for the RTP stream.

:return: None

"""

self.payload\_type = payload\_type.value

self.ssrc = random.randint(0, 2 \*\* 32 - 1) # Generate a random 32-bit SSRC

self.sequence\_number = 0

self.timestamp = start\_timestamp

self.\_update\_timestamp()

self.logger = Logger("rtp-logger").logger

def build\_header(self, marker: int = 0, csrcs: list[int] = None, extension\_data: bytes = None) -> bytes:

"""

Constructs the RTP header.

:param extension\_data:

:param marker: (int) The marker bit.

:param csrcs: (list[int]) List of contributing source identifiers.

:return: bytes: The RTP header as a byte string.

"""

if csrcs is None:

csrcs = []

cc = len(csrcs)

version = 2

padding = 0

extension = extension\_data is not None

self.sequence\_number += 1

header = (

(version << 30) |

(padding << 29) |

(extension << 28) |

(cc << 24) |

(marker << 23) |

(self.payload\_type << 16) |

(self.sequence\_number & 0xFFFF)

)

header\_bytes = struct.pack('!II', header, self.timestamp)

ssrc\_bytes = struct.pack('!I', self.ssrc)

csrc\_bytes = b''.join(struct.pack('!I', csrc) for csrc in csrcs)

extension\_bytes = b''

if extension\_data:

extension\_profile\_id = consts.CommunicationConsts.RTP\_EXTENSION\_PROFILE\_ID

extension\_length = struct.pack('!I', math.ceil(len(extension\_data) / 4))[2:]

extension\_header = consts.CommunicationConsts.RTP\_EXTENSION\_HEADER

extension\_bytes = extension\_profile\_id + extension\_length + extension\_header + extension\_data

return header\_bytes + ssrc\_bytes + csrc\_bytes + extension\_bytes

def create\_packets(self, payload: bytes ,csrcs: list[int] = None) -> list[bytes]:

"""

Creates an RTP packet by combining the header and payload.

:param payload: (bytes) payload to put in the rtp packet

:param csrcs: (list[int]) List of contributing source identifiers.

:return: bytes: The complete RTP packet.

"""

try:

payload = zlib.compress(payload)

self.\_update\_timestamp()

payloads = []

payload\_pointer = 0

header\_len = len(self.build\_header(0, csrcs, extension\_data=struct.pack('!I', len(payloads))))

while payload\_pointer < len(payload):

payload\_read\_end\_index = payload\_pointer + consts.CommunicationConsts.MAX\_UDP\_PAYLOAD\_SIZE - header\_len

# payload\_read\_end\_index = payload\_pointer + 1500 - header\_len

payloads.append(payload[payload\_pointer:payload\_read\_end\_index])

payload\_pointer = payload\_read\_end\_index

ext\_data = struct.pack('!I', len(payloads))

packets = []

for i in range(len(payloads)):

is\_last\_frag = i == len(payloads) - 1

header = self.build\_header(int(is\_last\_frag), csrcs, extension\_data=ext\_data)

packets.append(header + payloads[i])

self.logger.info("RTP packets created successfully.")

except Exception as e:

self.logger.exception("Error while creating RTP packet: %s", e)

raise

return packets

def get\_ssrc(self) -> int:

"""

Returns the SSRC for the RTP stream.

:return: int: The SSRC value.

"""

return self.ssrc

def \_update\_timestamp(self) -> None:

"""

updates timestamp

:return: None

"""

self.timestamp = int(time.time() \* 1000) % (2 \*\* 32) # Initialize with current time in milliseconds & 32 bit

**transmitting\_client\tc\_main.py**

"""

main file of the transmitting client

"""

import multiprocessing

from audio\_transmission\_handler import AudioTransmissionHandler

from video\_transmission\_handler import VideoTransmission

def main() -> None:

"""

Starts streaming

:return: None

"""

# Create processes

video\_process = multiprocessing.Process(target=VideoTransmission().transmit\_video)

audio\_process = multiprocessing.Process(target=AudioTransmissionHandler().start\_streaming)

# Start processes

video\_process.start()

audio\_process.start()

# Keep the main program running

video\_process.join()

audio\_process.join()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**transmitting\_client\udp\_handler.py**

"""

Will hold the class incharge of communications

"""

# Imports #

import logging

import socket

import time

from utils.consts import CommunicationConsts

from utils.logging\_messages import \*

from utils.logger import Logger

class UDPClientHandler:

"""

This class represents the video transmitting client using UDP.

It initializes an RTPHandler instance to handle RTP packet creation

and uses a UDP socket for sending packets.

"""

def \_\_init\_\_(self, port: int):

"""

Initializes the Client instance.

:param port: (int) the port to send the data through

:return: None

"""

self.server\_address = CommunicationConsts.HOST

self.server\_port = port

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

self.logger = Logger("UDP-logger").logger

def send\_packets(self, packets: list[bytes]) -> bool:

"""

Captures video frames, creates RTP packets, and transmits them to the server via UDP.

:param packets: (list[bytes]) packets to send

:return bool: whether operation was successful

"""

success = True

try:

# Send the packet to the server

self.logger.debug("Sending frame, fragmented into {} packets".format(len(packets)))

for packet in packets:

self.sock.sendto(packet, (self.server\_address, self.server\_port))

self.logger.info(SuccessMessages.PACKET\_SENT)

except Exception as e:

self.logger.exception(ErrorMessages.VIDEO\_TRANSMISSION\_ERROR, e)

success = False

return success

**transmitting\_client\utils\consts.py**

"""

This file holds all the constant variables of the transmitting client

"""

# Imports #

import logging

import pyaudio

from enum import Enum

class Ports(Enum):

"""

All available Ports

"""

VIDEO\_PORT = 5004

AUDIO\_PORT = 5006

class CommunicationConsts:

HOST = '127.0.0.1'

MAX\_UDP\_PAYLOAD\_SIZE = 65507

RTP\_HEADER\_SIZE = 12

MAX\_RTP\_PAYLOAD\_SIZE = MAX\_UDP\_PAYLOAD\_SIZE - RTP\_HEADER\_SIZE

RTP\_EXTENSION\_PROFILE\_ID = b'00'

RTP\_EXTENSION\_HEADER = b'0000'

class AudioCaptureConsts:

CHANNELS = 1

SAMPLE\_RATE = 44100

CHUNK\_SIZE = 512

FORMAT = pyaudio.paInt16

class LoggerConsts:

FORMAT = "%(asctime)s - %(levelname)s - %(message)s"

LOG\_LEVEL = logging.DEBUG

LOG\_DIR = r"logs/"

LOG\_FILE\_EXTENSION = ".log"

**transmitting\_client\utils\logger.py**

"""

Holds a generic logger class to handle logs.

"""

# Imports #

import logging

import os

from src.transmitting\_client.utils.consts import LoggerConsts

class Logger:

"""

Handles all logs

"""

def \_\_init\_\_(self, logger\_name: str) -> None:

"""

Initialize logger

:param logger\_name: (str) the name of the logger

:return: None

"""

self.format = LoggerConsts.FORMAT

self.level = LoggerConsts.LOG\_LEVEL

self.file\_name = logger\_name + LoggerConsts.LOG\_FILE\_EXTENSION

self.file\_path = LoggerConsts.LOG\_DIR + self.file\_name

if not os.path.isdir(LoggerConsts.LOG\_DIR):

os.makedirs(LoggerConsts.LOG\_DIR)

print("created dir")

self.logger = logging.getLogger(logger\_name)

self.logger.setLevel(self.level)

file\_handler = logging.FileHandler(self.file\_path)

file\_handler.setFormatter(logging.Formatter(self.format))

self.logger.addHandler(file\_handler)

**transmitting\_client\utils\logging\_messages.py**

"""

Global Logging Messages for the transmitting client

"""

class ErrorMessages:

"""

Constants to define error log messages.

"""

OPEN\_VIDEO\_SOURCE = "Unable to open video source: %s"

VIDEO\_CAPTURE\_INIT = "Error initializing video capture for source {source}: {error}"

RETRIEVE\_FRAME = "Failed to retrieve frame."

GET\_FRAME\_FROM\_SOURCE = "Error retrieving frame from video source {source}: {error}"

RELEASE\_SOURCE = "Error releasing video capture for source {source}: {error}"

VIDEO\_TRANSMISSION\_ERROR = "Error while transmitting video: %s"

class SuccessMessages:

"""

Constants to define success log messages.

"""

RETRIEVE\_FRAME = "Frame retrieved successfully."

RELEASE\_VIDEO\_SOURCE = "Video source %s released successfully."

VIDEO\_CAPTURE\_INIT = "Video Capture initiated successfully"

PACKET\_SENT = "Packet sent to server"

**transmitting\_client\utils\payload\_types.py**

"""

RTP payload types

"""

# Imports #

from enum import Enum

class PayloadTypes(Enum):

VIDEO = 0

AUDIO = 0```

## `transmitting\_client\video\_capture.py`

```python

"""

This file holds the VideoCapture class

"""

import logging

# Imports #

import cv2

import numpy as np

from src.transmitting\_client.utils.logging\_messages import ErrorMessages, SuccessMessages

class VideoCapture:

"""

This class handles the video capture and handling.

"""

def \_\_init\_\_(self, logger: logging.Logger, source=0) -> None:

"""

Initializes the video capture instance.

:param logger: (Logger) logger for this class

:param source: (int or str): The video source, can be a camera index or a video file path.

:return: None

"""

self.logger = logger

self.source = source

try:

self.capture = cv2.VideoCapture(self.source)

if not self.capture.isOpened():

self.logger.error(ErrorMessages.OPEN\_VIDEO\_SOURCE % self.source)

raise ValueError(ErrorMessages.OPEN\_VIDEO\_SOURCE % self.source)

self.logger.info(SuccessMessages.VIDEO\_CAPTURE\_INIT)

except Exception as e:

self.logger.exception(ErrorMessages.VIDEO\_CAPTURE\_INIT.format(source=self.source, error=e))

raise

def get\_frame(self):

"""

Retrieves a single frame from the video source.

:return tuple: (success, frame), where `success` is a boolean indicating if the frame was read successfully, and `frame` is the captured frame.

"""

res\_tuple: tuple[bool, np.ndarray]

try:

success, frame = self.capture.read()

if success:

self.logger.debug(SuccessMessages.RETRIEVE\_FRAME)

new\_size = (frame.shape[1] // 2, frame.shape[0] // 2)

img\_resized = cv2.resize(frame, new\_size, interpolation=cv2.INTER\_AREA)

frame = np.array(img\_resized)

else:

self.logger.error(ErrorMessages.RETRIEVE\_FRAME)

return success, frame

except Exception as e:

self.logger.exception(ErrorMessages.GET\_FRAME\_FROM\_SOURCE.format(source=self.source, error=e))

res\_tuple = (False, np.ndarray(shape=(0, 0)))

return res\_tuple

def release(self) -> None:

"""

Releases the video capture resource.

:return: None

"""

try:

if self.capture:

self.capture.release()

self.logger.info(SuccessMessages.RELEASE\_VIDEO\_SOURCE % self.source)

except Exception as e:

self.logger.exception(ErrorMessages.RELEASE\_SOURCE.format(source=self.source, error=e))

def \_\_del\_\_(self) -> None:

"""

Ensures resources are released when the instance is deleted.

:return: None

"""

self.release()

self.logger.info(SuccessMessages.RELEASE\_VIDEO\_SOURCE % self.source)

**transmitting\_client\video\_transmission\_handler.py**

"""

This file holds the VideoTransmission class.

"""

# Imports #

import logging

import time

import numpy as np

from rtp\_handler import RTPHandler

from udp\_handler import UDPClientHandler

from utils import consts

from utils.payload\_types import PayloadTypes

from video\_capture import VideoCapture

from utils.logger import Logger

class VideoTransmission:

"""

Bundles video handlers and handles transmission

"""

def \_\_init\_\_(self, video\_capture\_source: int = 0):

"""

Initiating class members

:param video\_capture\_source: (int) camera port

"""

self.port = consts.Ports.VIDEO\_PORT.value

self.video\_capture\_source = video\_capture\_source

self.payload\_type = PayloadTypes.VIDEO

self.rtp\_handle = RTPHandler(self.payload\_type)

self.udp\_handle = UDPClientHandler(self.port)

self.logger = Logger("video-logger").logger

def transmit\_video(self) -> None:

"""

transmit video using rtp

:return: None

"""

video\_capture = VideoCapture(self.logger, self.video\_capture\_source)

while True:

# Capture start time

start\_time = time.time()

success, frame = video\_capture.get\_frame()

if success and isinstance(frame, np.ndarray):

payload = frame.tobytes()

packets: list[bytes] = self.rtp\_handle.create\_packets(payload)

\_ = self.udp\_handle.send\_packets(packets)

# Handling delta time

delta\_time = time.time() - start\_time

time.sleep(max(0.033 - delta\_time, 0)) # ~30 frames per second