Hadar's Zoom

מגיש: הדר שחר

ת.ז: 214436966

בית הספר: תיכון רוטברג

כיתה: יב'4

מנחה: אילת משיח

תוכן עניינים

מבוא

במהלך השנה האחרונה למדנו שעות רבות ב-zoom בגלל מגפת הקורונה, ולכן עלה לי הרעיון לכתוב תוכנת zoom קטנה בעצמי. המטרה שלה היא כמובן לאפשר למשתמשים לקיים שיחות וידאו ואודיו קבוצתיות.

בחרתי ברעיון זה מכיוון שהוא משלב נושאים רבים שמעניינים אותי, בין היתר נושאים בתחום של תקשורת בין מחשבים. בנוסף, אהבתי את הרעיון משום שחשבתי שיהיה נחמד לבנות בעצמי תוכנה דומה מאוד לזאת שאנו משתמשים בה על בסיס יומי כבר יותר משנה...

בהמשך הספר אתאר בפירוט כל אחד מהפיצ'רים בתוכנה. כעת אציג אותם בקצרה:

- שיחות וידאו ואודיו בין מספר משתתפים.
- צ'אט בין המשתתפים – ניתן לשלוח הודעות לכולם או רק למשתמש ספציפי.
- שיתוף מסך – משתמש אחד משתף את מסך המחשב שלו וכל האחרים רואים אותו.
- לוח חכם – לוח ציור שמזהה קווים ישרים ומלבנים שהמשתמש מצייר.
- חלון מרוחק משותף – כרגע notepad, טקסט שייכתב בחלון של משתמש אחד, יסונכרן עם הטקסט שבחלונות של המשתמשים האחרים.

קהל היעד

התוכנה מיועדת לכל מי שרוצה לתקשר עם אנשים אחרים מרחוק. למשל:

- חברים שרוצים לדבר ביניהם ולראות אחד את השני.
- תלמידים שרוצים ללמוד מרחוק.
- מורים שרוצים להעביר שיעורים דרך התוכנה.
- אנשים מבוגרים שאינם מתמצאים במחשב ורוצים להיעזר מרחוק בבני משפחה צעירים. למשל סבא או סבתא שנתקלים בבעיה במחשב ונעזרים בנכדים דרך התוכנה.
- ועוד רבים אחרים...

מודל הפרויקט ומחקר מקדים

ראשית, כדי להוציא לפועל את הרעיון שחלק גדול ממנו עוסק בהעברת וידאו ואודיו בין מחשבים הייתי צריך קודם כל להתעמק בתחומים אלה.

Video Streaming

קריאת הווידאו מהמצלמה מתבצעת באמצעות קריאת פריימים. כל פריים שנקרא מומר למערך רב מימדי של מספרים שערכיהם מייצגים את הצבעים של כל פיקסל בפריים.

השתמשתי כבר בעבר בספרייה המצוינת opencv-python כאשר התעסקתי עם תמונות ווידאו ולכן כבר הכרתי חלק מהשימושים שלה. יש לציין שהספרייה opencv נכתבה בשפות התכנות c ו-c++ שהן הרבה יותר מהירות מאשר python והספרייה opencv-python רק עוטפת את הספרייה המקורית ומספקת ממשק עבור python. בזכות העבודה על הפרויקט התעמקתי בספרייה זו ובאמצעותה מימשתי את קריאת הוידאו מהמצלמה של המשתמש, פריים אחר פריים. השתמשתי גם בספרייה numpy, שמשמשת לעבודה עם מספרים ועם מערכים רב מימדיים, שכן כל פריים הוא מערך רב מימדי של מספרים.

כדי להקטין את גודל המידע שבפריים דחסתי אותו באמצעות פעולות של opencv ולאחר מכן שלחתי אותו לשרת. תחילה מימשתי פרוטוקול שהעביר כל תמונה מעל TCP. בהמשך מימשתי פרוטוקול מהיר יותר שעושה זאת מעל UDP ובו השתמשתי בפרויקט ואפרט עליו בהמשך הספר.

Audio Streaming

קול הוא סוג של גל אשר יכול להתפשט בתווך מסוים. התדירות של גל הקול, כלומר מספר המחזורים שלו בשנייה אחת, משפיעה על הצליל שהוא משמיע, וככל שהיא גבוהה יותר כך הצליל נשמע גבוה יותר. כדי לקלוט קול במחשב משתמשים במיקרופון, אשר ממיר את גלי הקול לייצוג דיגיטלי, כלומר לאפסים ואחדים. כדי להשמיע קול משתמשים ברמקול או באוזניות, אשר ממירים ייצוג דיגיטלי של קול לגלי קול.

לא ניתן לקיים שיחה ללא העברת אודיו ולשם כך למדתי כיצד לקלוט ולהשמיע קול במחשב. בעזרת הספרייה pyaudio יצרתי stream של אודיו אשר מספק ממשק נוח לפעולות אלה. כאשר קוראים ממנו בתים, קולטים סאונד מהמיקרופון וכאשר כותבים אליו בתים משמיעים סאונד ברמקול.

GUI

ממשק המשתמש הוא חלק חשוב ומרכזי בתוכנה שלי ולכן רציתי שהוא יהיה יפה וקל לשימוש. יצרתי אותו עם הספרייה PyQt5 ותוך כדי למדתי להשתמש בה. למדתי על האובייקטים השונים בה ועל מנגנון הסיגנלים שלה (Signals and Slots). מנגנון זה הוא שימושי מאוד עבור תקשורת בין רכיבים שונים בממשק. הרעיון הוא פשוט – מגדירים אובייקט סיגנל של הספרייה או משתמשים בסיגנל קיים שמאפשר "להקשיב" לאירוע מסוים. מגדירים פעולה כלשהי שתקרא כאשר האירוע יתרחש והסיגנל ישודר. פעולה זו, המקבלת את הסיגנל נקראת בטרמינולוגיה של הספרייה Slot.

חלק מהסיגנלים מובנים בספרייה, כמו סיגנל של לחיצת עכבר על אובייקט מסוים, אך ניתן גם להגדיר סיגנלים שישדרו אירועים אחרים. בנוסף, באמצעות הסיגנל ניתן להעביר מידע לפעולה שתקרא כאשר הוא ישודר. למשל, הגדרתי סיגנל במחלקה של לקוח הוידאו שמתאר

אירוע של פריים חדש שהתקבל והחלון הראשי הגדיר פעולה שנקראת כאשר סיגנל זה משודר. כך כאשר הלקוח מקבל פריים הוא משדר סיגנל זה, החלון הראשי קולט את הסיגנל באמצעות הפעולה שהגדיר והיא מציגה את הפריים שהתקבל.

WinApi

במהלך הכנת הפרויקט למדתי עוד על היכולות השונות של ה-WinApi.

השתמשתי בידע זה במימוש של פ'יצר מגניב שהוספתי לתוכנה: חלון מרוחק משותף – RemoteWindow. הרעיון היה ליצור חלון שיהיה משותף בין הלקוחות בפגישה. עשיתי זאת עם חלון של התוכנה notepad שמסנכרן את הטקסט עם הטקסט שבחלונות של המשתתפים האחרים בפגישה.

לשם כך למדתי על הממשק ש-Windows מספקת למתכנתים ועל חלק מהפעולות שהיא מייצאת. קראתי הרבה ב-msdn על הדברים השונים שרציתי לממש וכך למדתי בין היתר על Window Messages. אלה הודעות שמערכת ההפעלה מעבירה לכל חלון בתגובה לאירועים שונים, למשל הקשה על מקש במקלדת בתוך החלון. הודעות אלה הם בסך הכל מספרים קבועים ובדומה למערכת ההפעלה, גם מתכנתים יכולים לשלוח הודעות אלה לחלונות באמצעות הפונקציה SendMessage של WinApi.

כך למשל, על ידי שליחת ההודעה WM_SETTEXT לחלון מסוים ניתן לשנות את הטקסט שלו, עבור edit control (חלון שמשמש לעריכת טקסט, לדוגמה זה שקיים ב-notepad) הודעה זו משנה את התוכן שלו!

דוגמה נוספת היא ההודעה EM_SETSEL שמאפשרת לבחור טווח של תווים ב-edit control. בעזרתה מימשתי את המנגנון שמסמן את הטקסט שמשתמש בוחר בחלון אחד בחלונות של המשתמשים האחרים (שימושי למשל אם משתמש רוצה להדגיש מילה מסוימת).

כדי לשלוח הודעה לחלון מסוים, יש להשיג handle לחלון הזה. זהו בעצם מזהה ייחודי שמערכת ההפעלה מספקת כאשר רוצים לגשת לאובייקט מסוים. בפרויקט השגתי את ה-handle לחלון של notepad על ידי מעבר על החלונות הפתוחים באמצעות הפונקציה EnumWindows וחיפוש החלון שנוצר על ידי תהליך עם ה-pid (process id) של notepad. היה לי את ה-pid מכיוון שאני יצרתי את התהליך של notepad באמצעות הספרייה subprocess.

לאחר מכן, השגתי handle לחלון ה-edit control של notepad בעזרת הפונקציה FindWindowExA לפי שם המחלקה שלו ("Edit") וכך שלחתי לו הודעות שונות.

OAuth 2.0

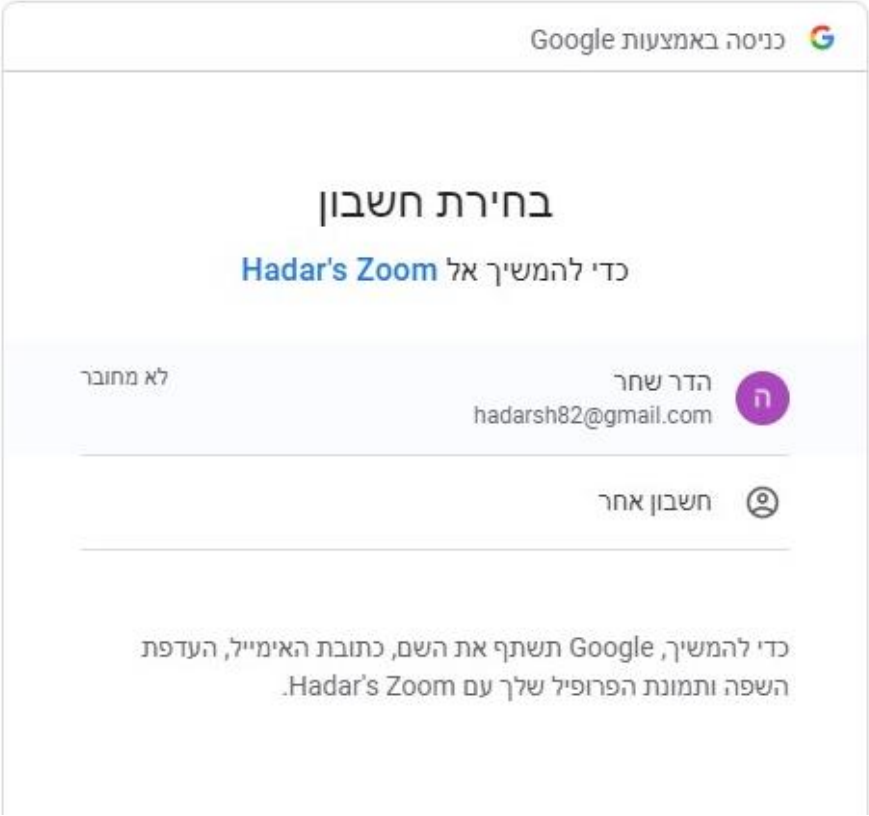
בפרויקט רציתי לאפשר למשתמשים להתחבר גם עם חשבון הגוגל שלהם. כלומר, כאשר הם ילחצו על כפתור ה - sign in with google במסך הפתיחה, הם יופנו לדף מתאים בדפדפן ויתבקשו להתחבר לחשבון הגוגל שלהם ולאשר לתוכנה שלי גישה לשמם, כתובת האימייל ותמונת הפרופיל שלהם. לאחר שהם יעשו זאת הם יופנו בחזרה לתוכנה שלי ושמם ותמונת הפרופיל שלהם יתעדכנו במסך הפגישה.

לשם כך למדתי על פרוטוקול ההזדהות OAuth 2.0 בו משתמשים Google APIS כדי לאמת משתמשים. לאחר שאני מאמת את המשתמש עם גוגל באמצעות פרוטוקול זה, אני מקבל access token שבעזרתו אני יכול לגשת ל – API מסוים של גוגל המספק מידע על המשתמש. מימשתי את התהליך המלא בעצמי ופירטתי על המימוש בחלק של הפרוטוקולים בספר.

אפיון – מדריך משתמש

המסך הראשון שנפתח למשתמש הוא מסך הפתיחה. במסך זו מוצגות למשתמש שתי אפשרויות להתחבר לתוכנה – עם חשבון ה-Google שלו או עם השם שלו.

אם המשתמש בוחר להתחבר עם Google הוא מופנה לדף בדפדפן המבקש ממנו את פרטי חשבונו ואת אישורו לשתף את המידע שלו עם התוכנה.

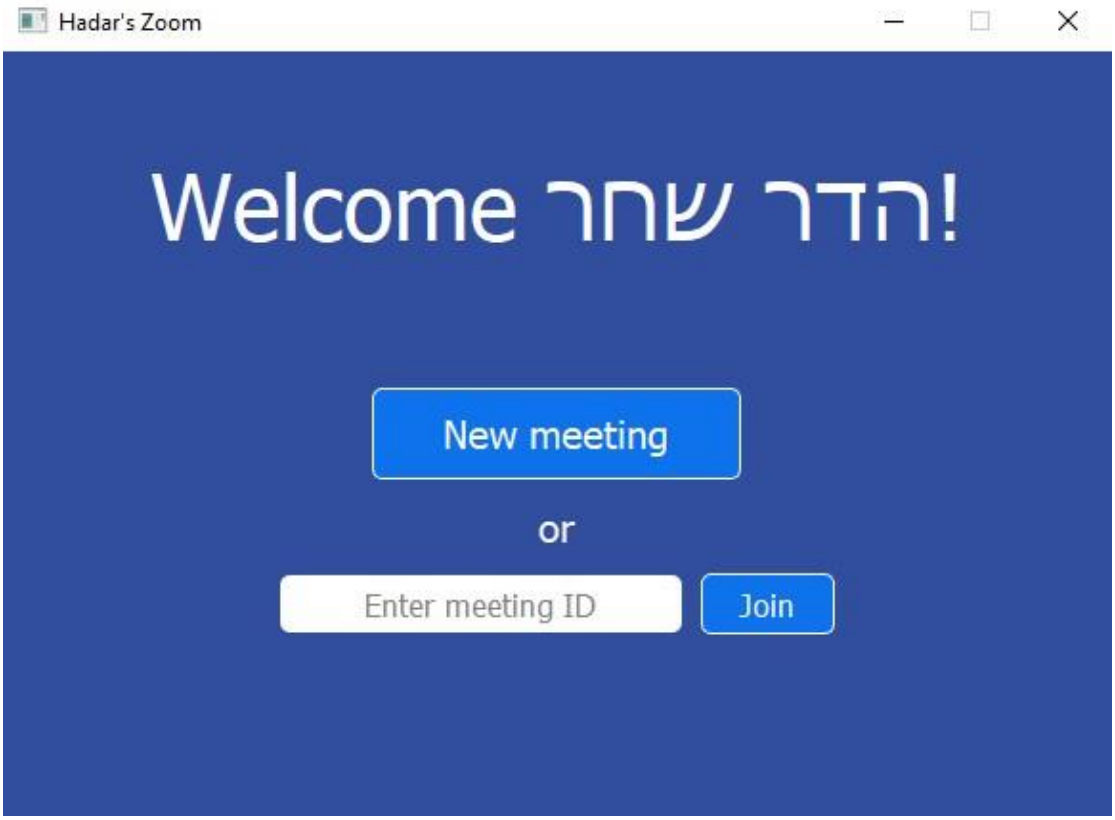


לאחר שהמשתמש עושה זאת, הוא מופנה לדף זמני בדפדפן שמאשר את הזדהותו:

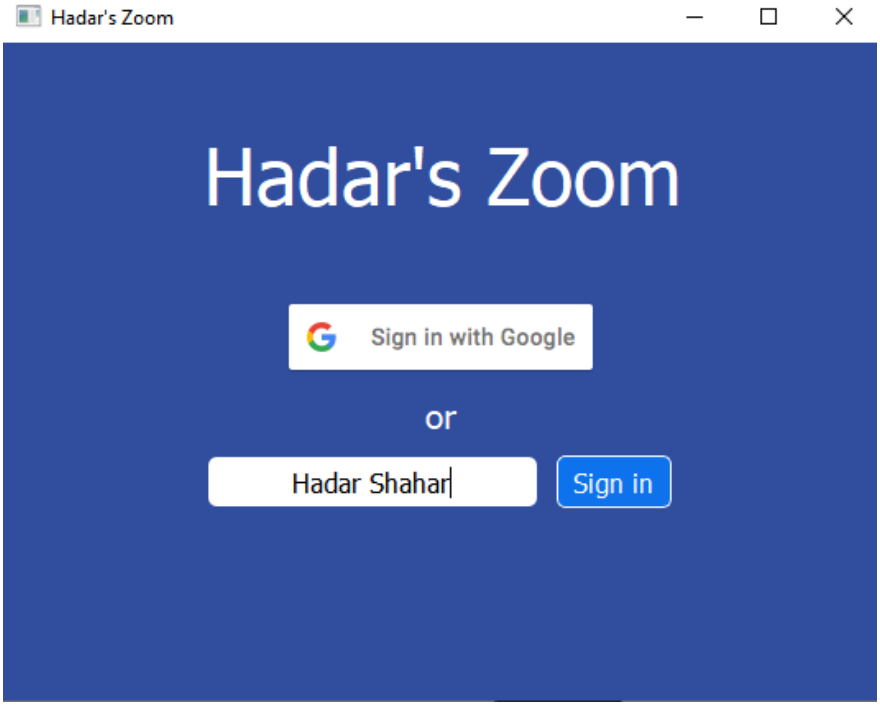
Hello הדר שחר!
You can return to the app now.

ומיד חלון התוכנה עובר לקדמה ומעדכן את החלון כך שיציג כפתור ליצירת פגישה חדשה וכפתור להצטרפות לפגישה קיימת, בהינתן ה-id שלה.

בנוסף, שם המשתמש מתעדכן לפי חשבון הגוגל שלו ובהמשך אף תמונת הפרופיל שלו תתעדכן ותופיע במסך הפגישה.



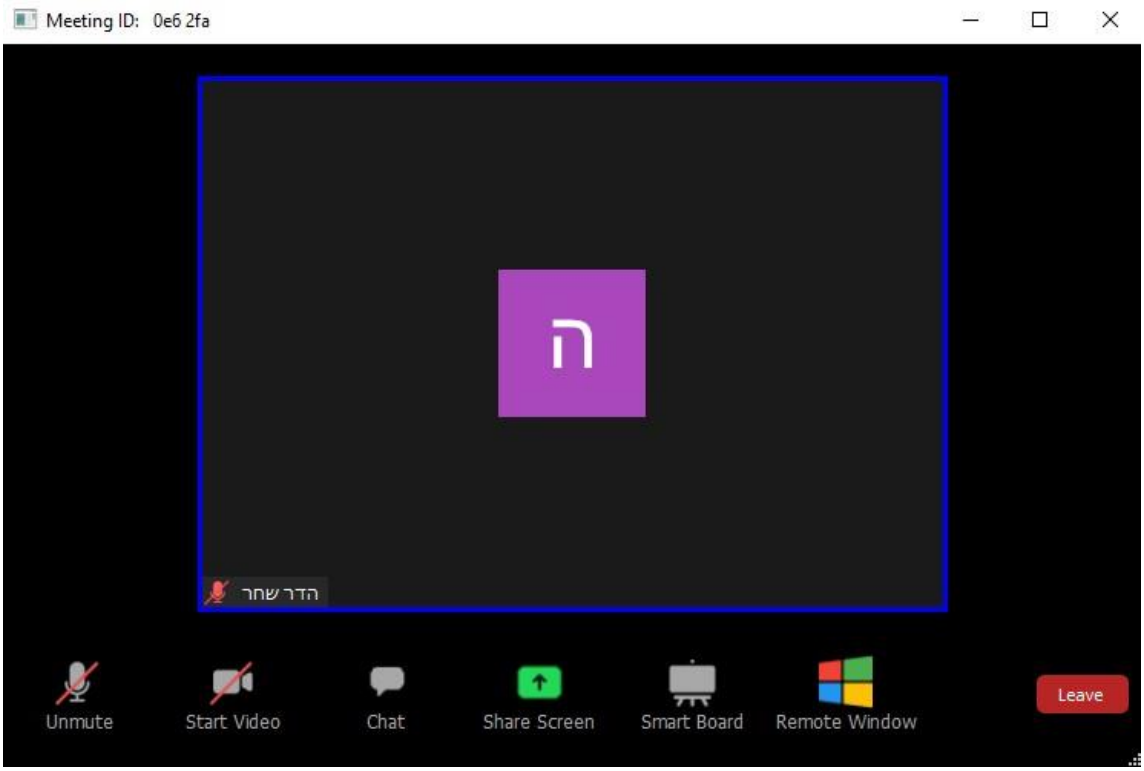
למשתמש קיימת גם אפשרות נוספת להיכנס לתוכנה עם השם שלו:



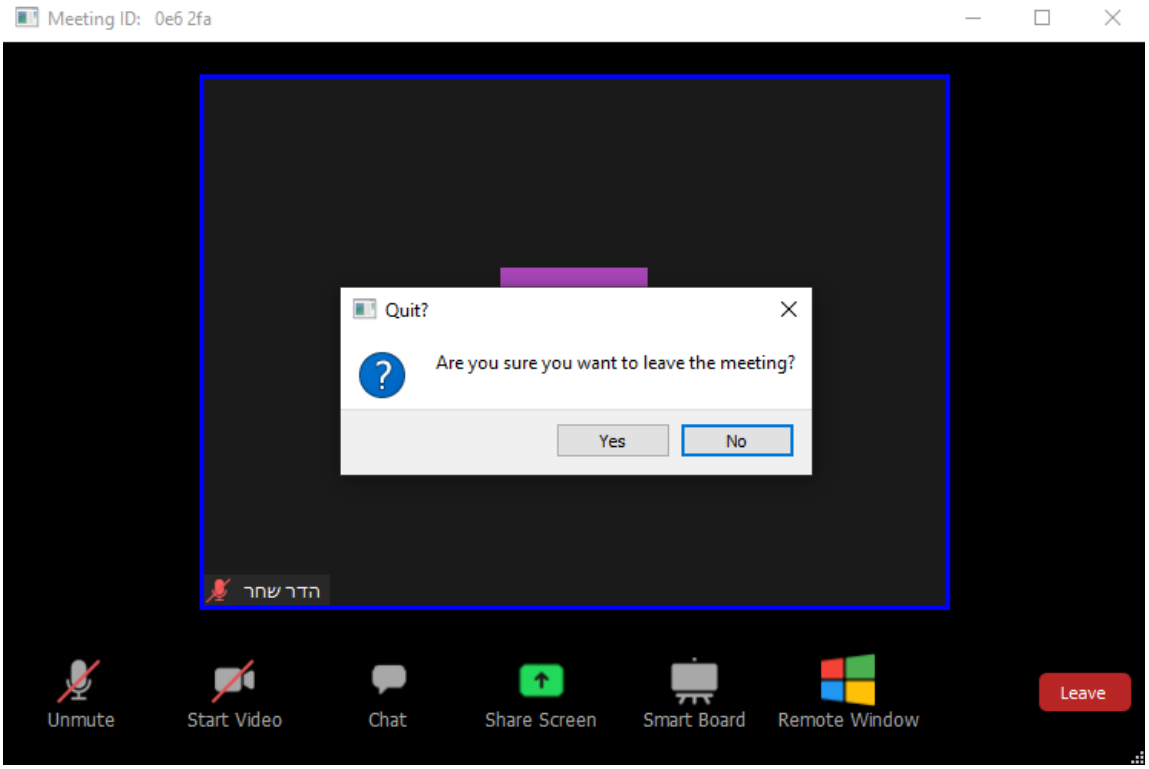
לאחר מכן המשתמש יעבור לחלון ההצטרפות לפגישה / יצירת פגישה חדשה בדומה לדרך ההזדהות עם Google.

יש לציין שבשתי דרכי ההזדהות לאחר לחיצה על הכפתור sign in, מוצג למשתמש מסך טעינה זמני, אך לרוב השרת יענה לבקשת ההזדהות במהירות ולא יהיה ניתן להבחין במסך הטעינה:

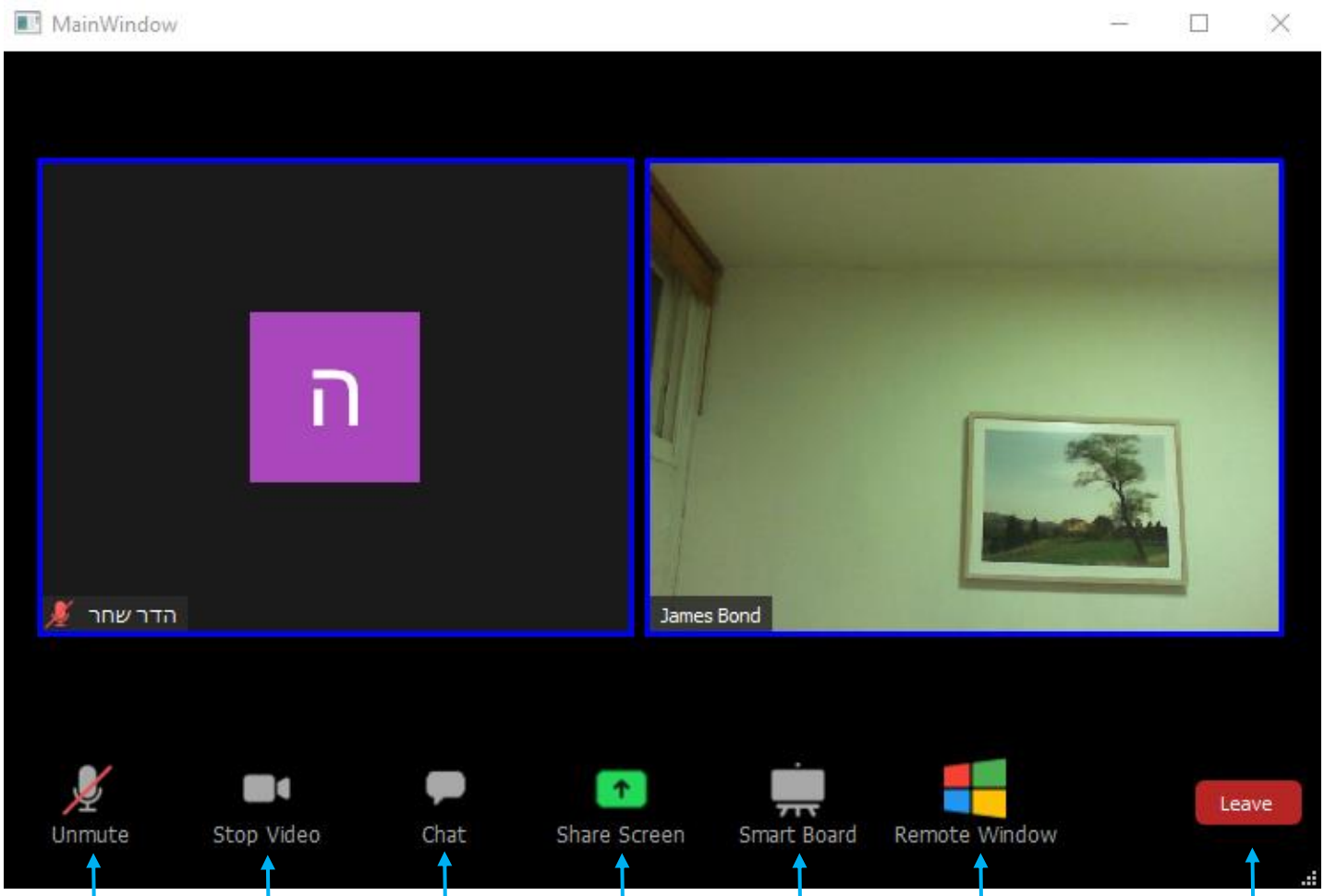
לאחר שהמשתמש יצר פגישה חדשה או הצטרף לפגישה קיימת הוא יעבור למסך הפגישה:



אם המשתמש ילחץ על הכפתור לעזיבת הפגישה או יסגור את חלון הפגישה, יופיע לו חלון לאישור הפעולה.



אסביר בקצרה על הכפתורים השונים במסך הפגישה.



מדליק/מכבה את המיקרופון

מדליק/מכבה את המצלמה

פותח/סוגר את חלון הצ'אט

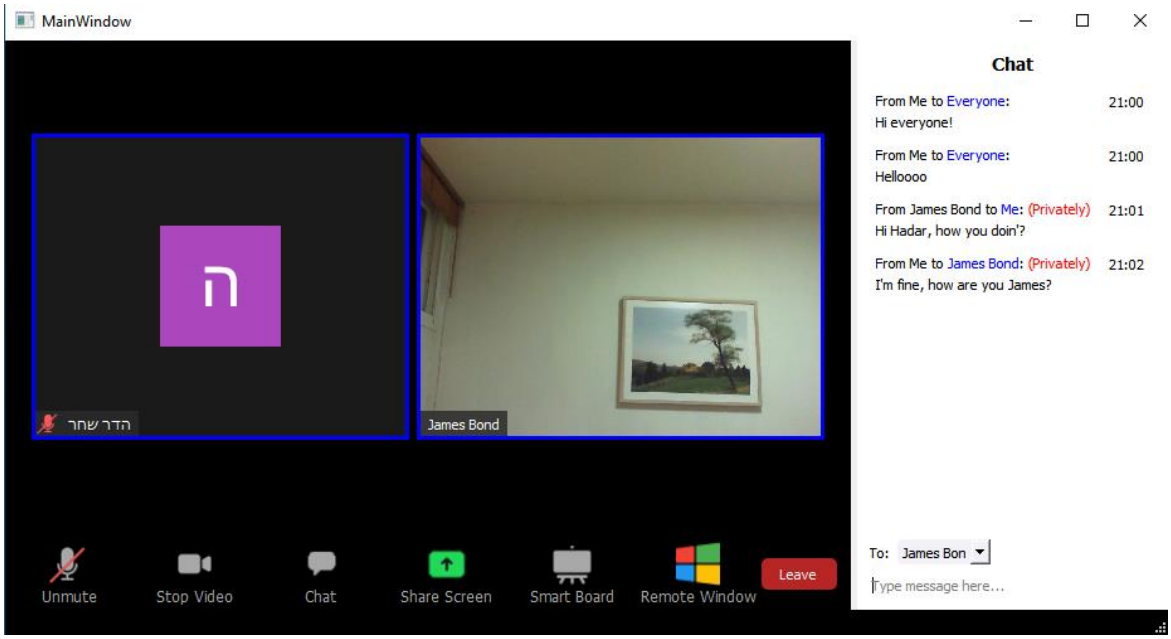
מתחיל/מפסיק שיתוף מסך

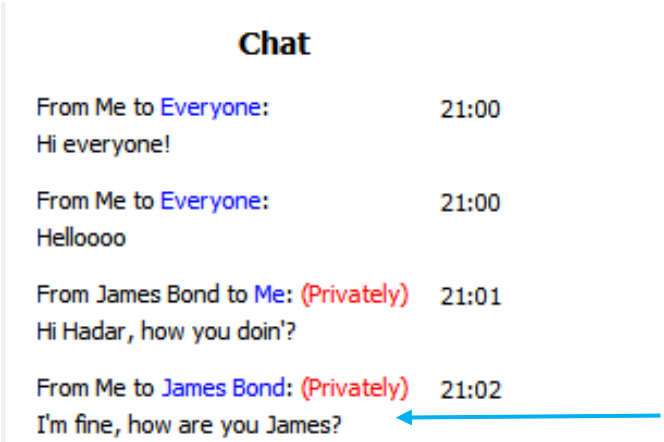
מתחיל/מפסיק את שיתוף הלוח החכם

מתחיל/מפסיק שיתוף חלון מרוחק

עוזב את הפגישה

חלון הצ'אט



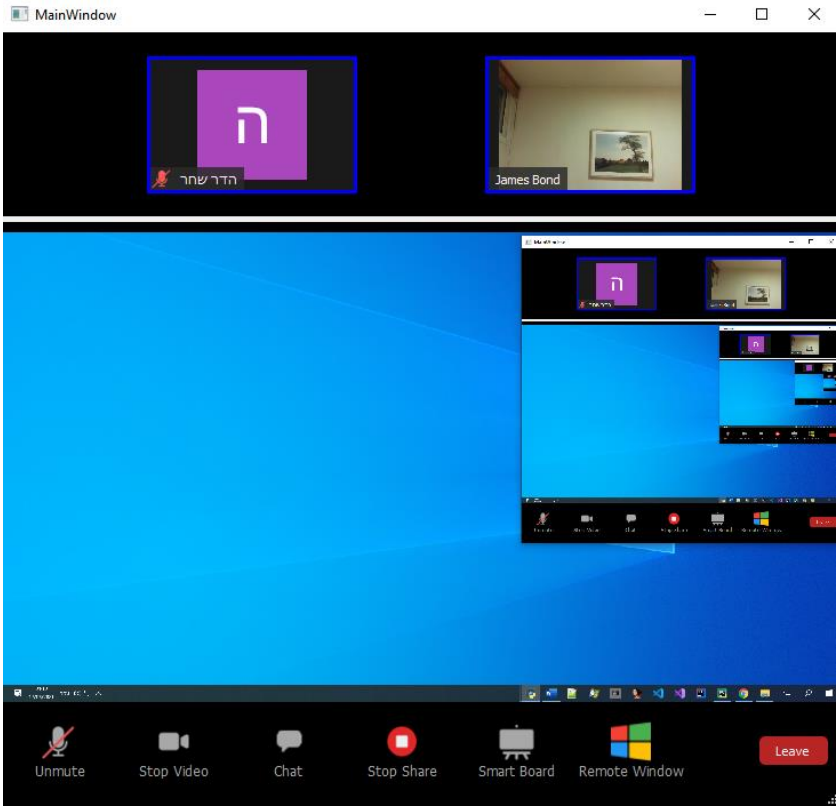


כל הודעת טקסט מכילה את שם המוען והנמען, תוכן, שעת השליחה והאם היא נשלחה לכולם או רק למשתמש ספציפי (privately)

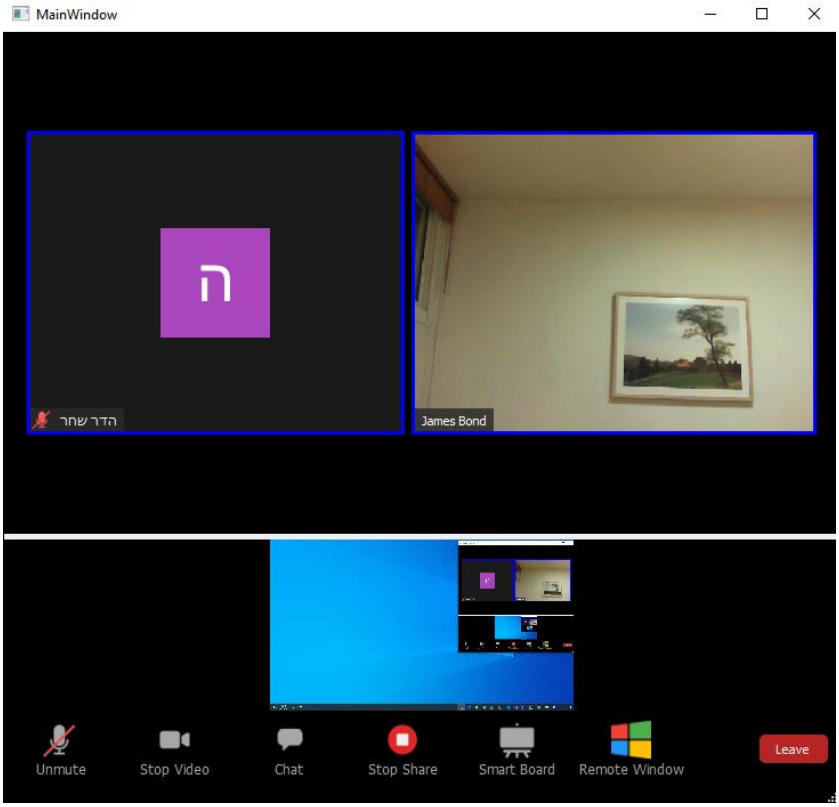
בחירת נמען ספציפי / Everyone

תוכן ההודעה

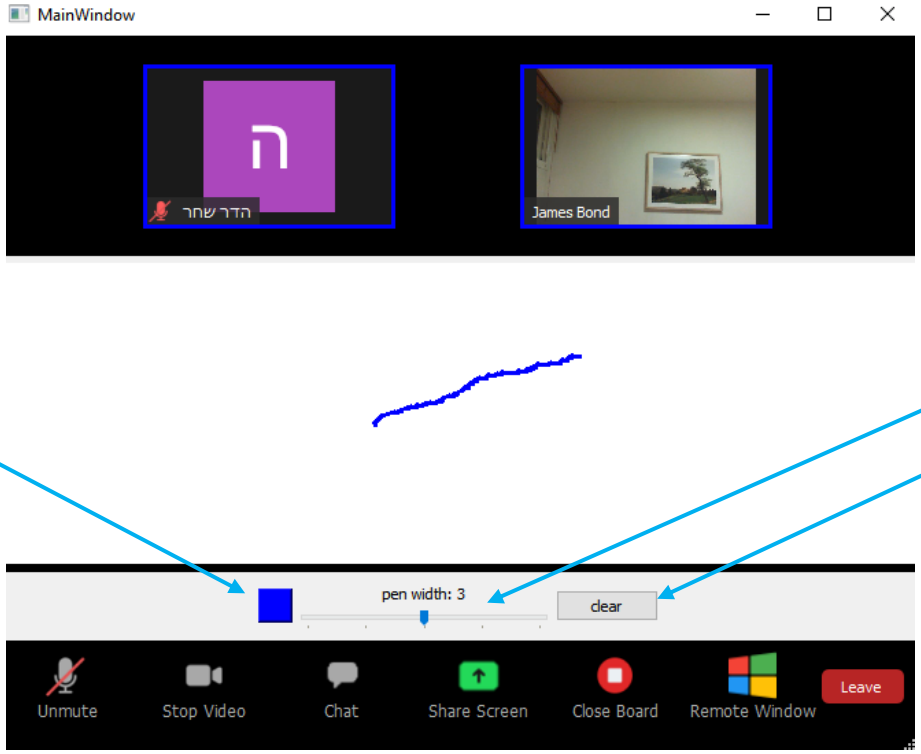
שיתוף מסך



בכל מצבי השיתוף ניתן להתאים את התצוגה של גודל המסך המשותף בעזרת הסרגל הלבן



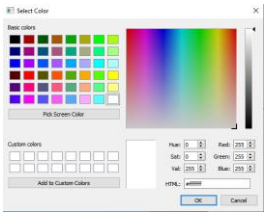
לוח חכם – מזהה קווים ישרים ומלבנים



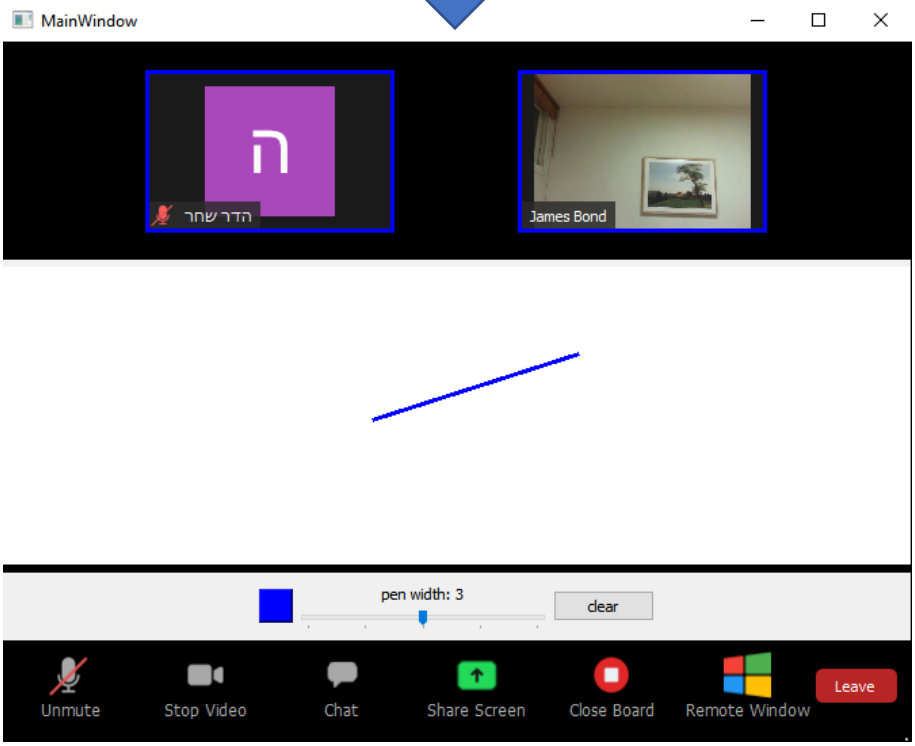
בחירת עובי העט

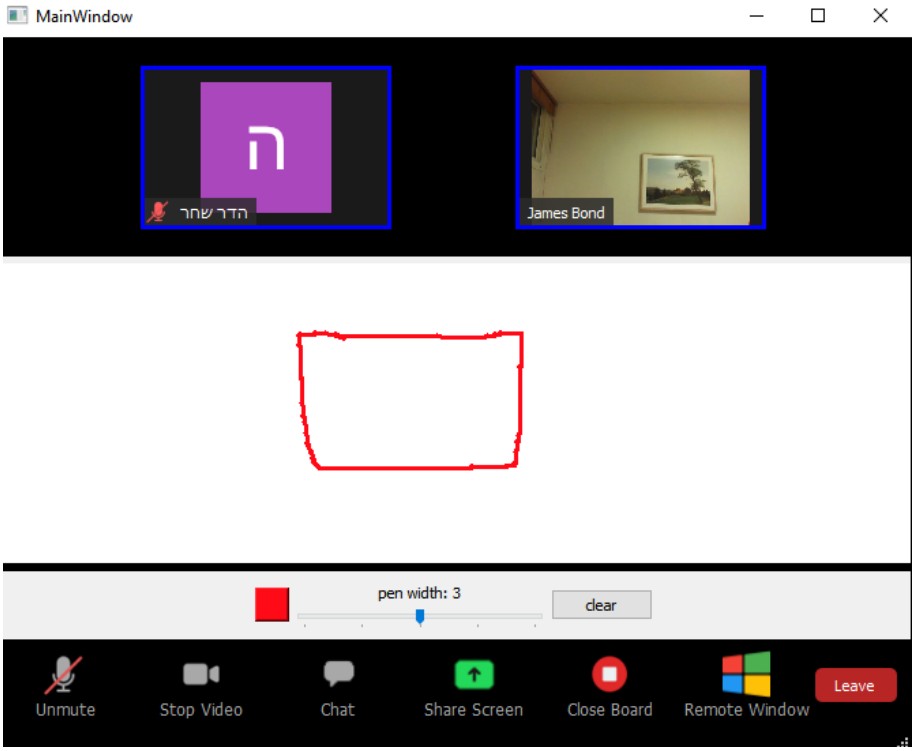
כפתור לניקוי הלוח (ניתן גם ללחוץ על המקש Delete)

כפתור בחירת צבע – בלחיצה פותח חלון:

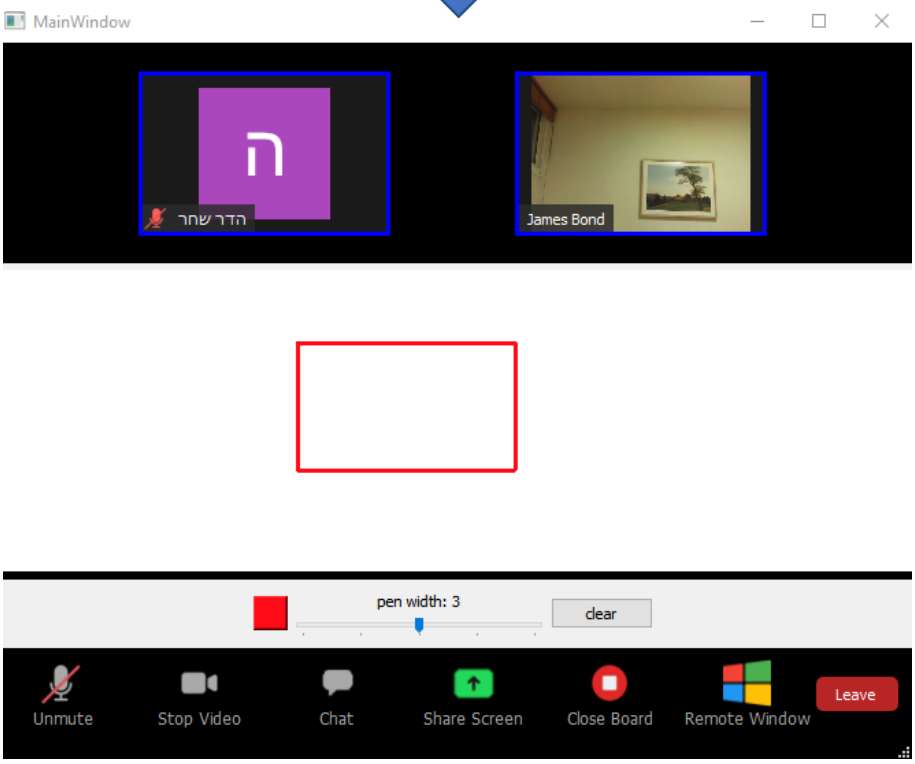


הלוח החכם הופך את הציור לקו ישר



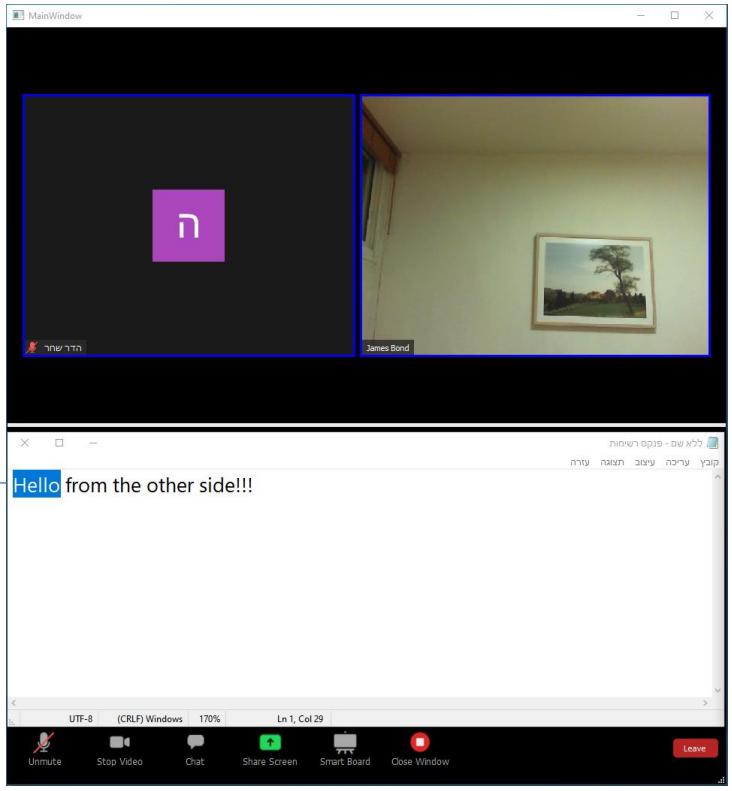


הלוח החכם הופך
את הציור למלבן

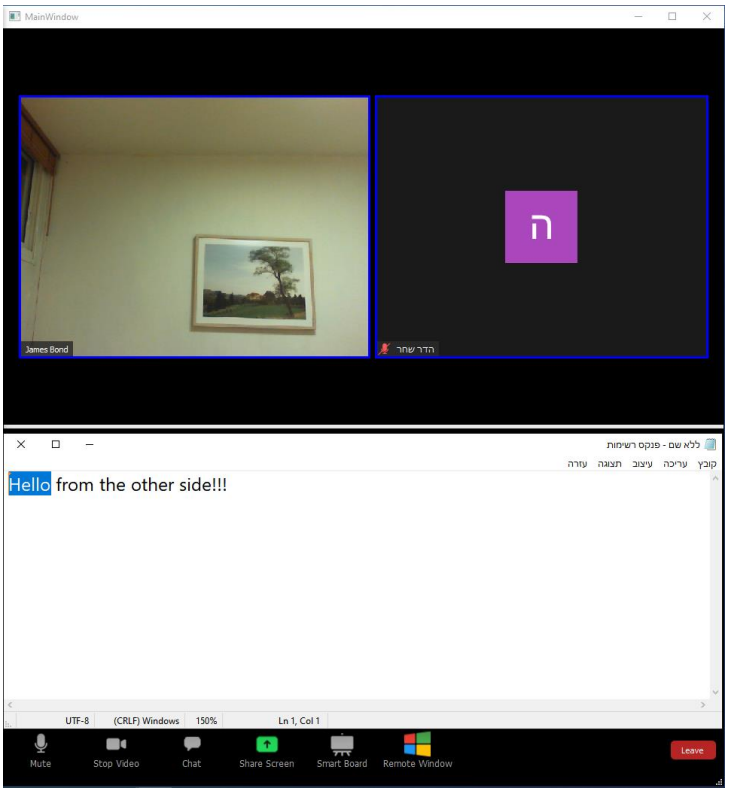


Remote window

כאשר לקוח מתחיל שיתוף חלון מרוחק, נפתח לו חלון של notepad שמתמקם בדיוק במרווח בין הוידאו של שאר המשתתפים וסרגל הכלים התחתון. חלון זה יזוז מעכשיו עם החלון של הפגישה וישנה את גודלו בהתאמה כך שהוא יורגש כחלק מחלון הפגישה.



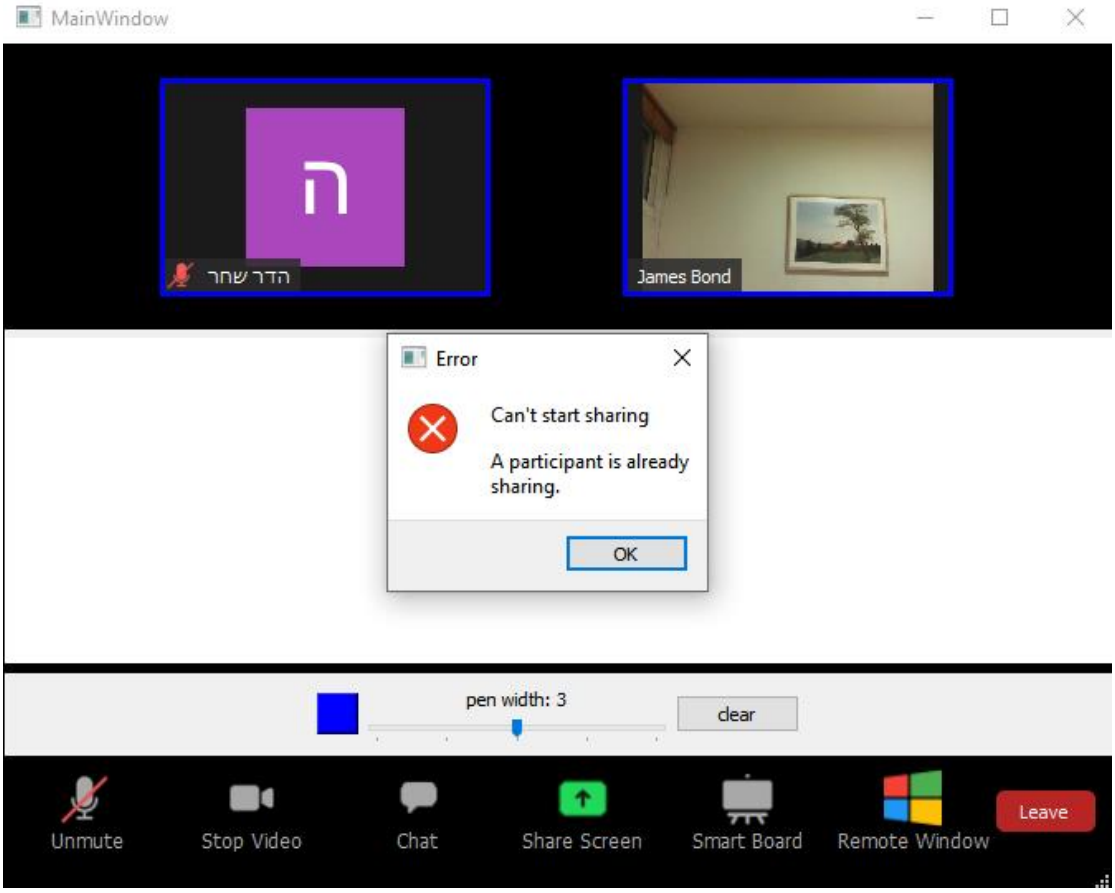
הטקסט שייכתב בחלון אחד יסונכרן עם הטקסט שבחלונות של המשתמשים האחרים, כך שייווצר מעין חלון מרוחק משותף.



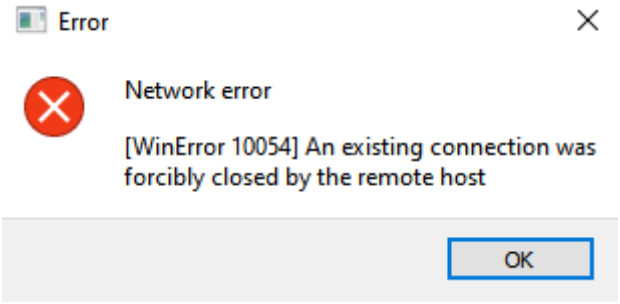
חלונות שגיאה

- אם לקוח מנסה להתחיל שיתוף מסוים (מסך/לוח חכם/חלון מרוחק) כאשר לקוח אחר באמצע שיתוף, תקפוץ הודעת שגיאה שמסבירה שלא ניתן לעשות זאת מכיוון שרק לקוח אחד יכול לשתף באותו רגע.

לדוגמה: בצילום המסך הבא ניתן לראות שלקוח מסוים נמצא באמצע שיתוף של הלוח החכם ולכן לקוח אחר לא יכול לקטוע את השיתוף שלו באמצע עם שיתוף חדש.



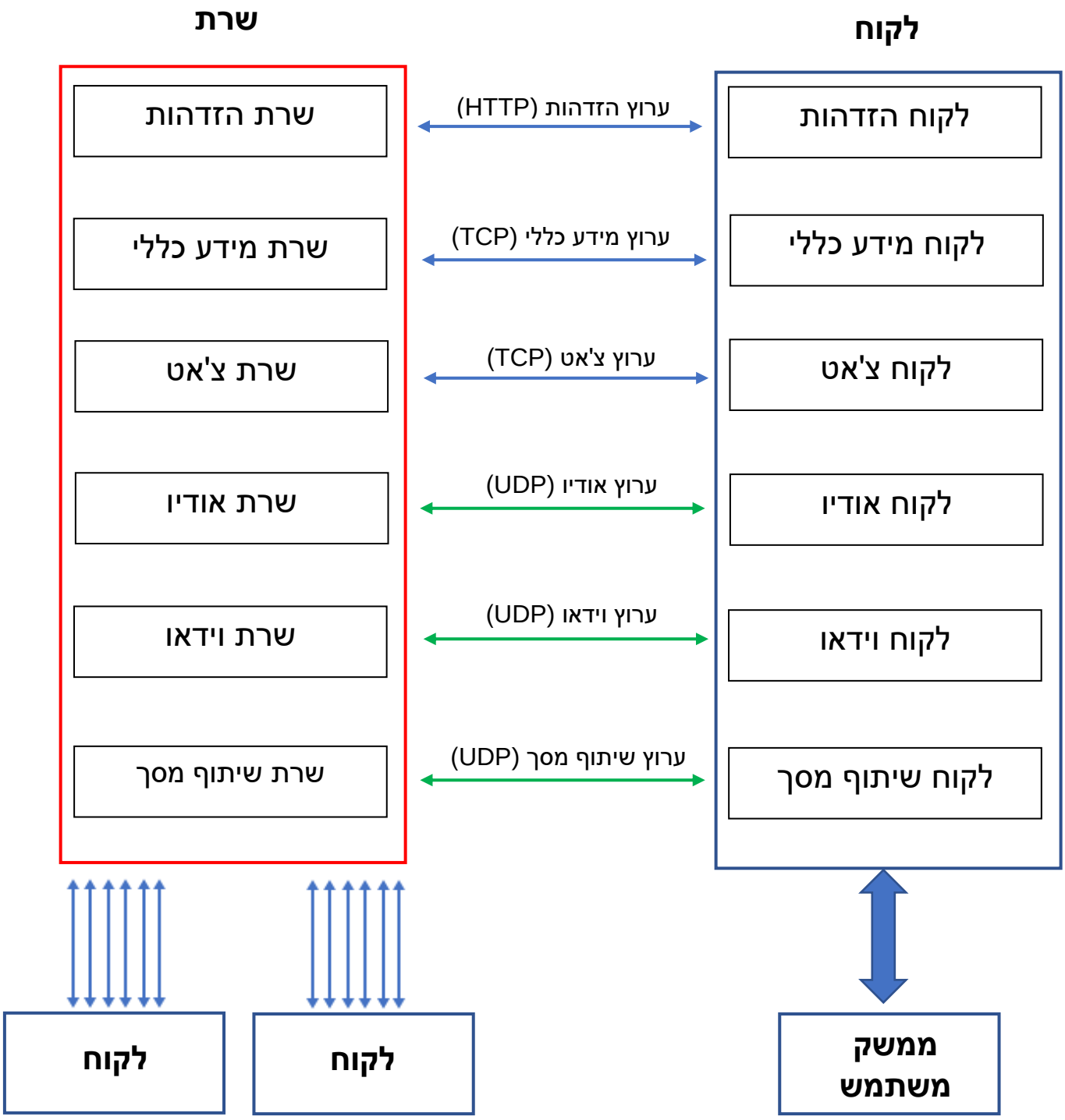
- הודעת שגיאה נוספת מתקבלת אם הייתה שגיאת רשת חמורה, למשל אם השרת קרס (אם זה קרה כנראה שמישהו סגר אותו בזדון).



ארכיטקטורת המערכת

מבט על

במערכת יש שרת מרכזי אחד ומספר לקוחות שעובדים מולו. השרת מחולק למספר שרתים קטנים וגם כל לקוח מחולק למספר לקוחות קטנים, לפי ערוצי התקשורת. כל לקוח קטן עובד מול השרת הקטן שמתאים לו.



הסבר כללי על הרכיבים

השרת

מורכב ממספר שרתים קטנים, כל אחד אחראי על ערוץ תקשורת מסוים:

- **שרת הזדהות** – שרת HTTP שאחראי לניהול הלקוחות והפגישות, יוצר id ייחודי לכל לקוח חדש שמצטרף ו-id ייחודי לכל פגישה חדשה שהוא יוצר, ואחראי לאימות ה-idים.
- **שרת מידע כללי** – שרת TCP שאחראי להעברת מידע כללי בין המשתתפים בפגישה מסוימת. מידע זה כולל הודעות על הצטרפות ועזיבה של לקוחות והודעות על כיבוי/הדלקת המיקרופון/המצלמה של לקוח מסוים.
בנוסף, הודעות מידע הקשורות לשיתוף הלוח החכם והחלון המרוחק עוברות דרך ערוץ המידע הכללי, למשל הודעות המציינות שלקוח החל שיתוף מסוג מסוים או הפסיק אותו.
- **שרת צ'אט** – שרת TCP אחראי להעביר הודעות הצ'אט מלקוח מסוים לשאר הלקוחות באותה פגישה או ללקוח ספציפי בפגישה (הודעה פרטית).
- **שרת אודיו** – שרת UDP שאחראי להעביר את האודיו בין כל לקוח לשאר הלקוחות באותה פגישה.
- **שרת וידאו** – שרת UDP שמעביר את הוידאו שנקלט מהמצלמה לשאר הלקוחות בפגישה.
- **שרת שיתוף מסך** – שרת UDP שאחראי להעביר את תמונות המסך כאשר הוא משותף.

הלקוח

בדומה לשרת, מורכב ממספר לקוחות קטנים, בהתאם לערוצי התקשורת:

- **לקוח הזדהות** – שולח בקשות הזדהות, יצירת פגישה, הצטרפות לפגישה והתנתקות לשרת ההזדהות.
- **לקוח מידע כללי** – שולח מידע כללי שמתקבל ממשק המשתמש ומקבל מידע מהשרת. הוא מעביר כל הודעת מידע שהוא מקבל לחלון הפגישה שמטפל בה בהתאם לסוגה.
- **לקוח צ'אט** – שולח הודעות צ'אט שהמשתמש מזין בחלון הצ'אט וכן מקבל הודעות צ'אט מהשרת, אותן הוא מעביר לחלון הצ'אט שמציג אותן.
- **לקוח אודיו** – מקבל מהשרת את האודיו של שאר המשתתפים בפגישה ומשמיע אותו. שולח לשרת את האודיו של הלקוח כאשר הלקוח מאפשר זאת בסרגל השליטה של חלון הפגישה.
- **לקוח וידאו** – מקבל מהשרת את הוידאו של שאר המשתתפים בפגישה ומעביר אותו לממשק שמציג אותו. שולח לשרת את הוידאו של הלקוח מהמצלמה שלו כאשר הלקוח מאפשר זאת בסרגל השליטה של חלון הפגישה.
- **לקוח שיתוף מסך** – מקבל את צילומי המסך מהשרת במידה ומשתמש מסוים משתף מסך, מעביר אותם לחלון הפגישה שמציג אותם. אם הלקוח הנוכחי משתף מסך, הוא מצלם את המסך ושולח אותו לשרת.

ממשק המשתמש

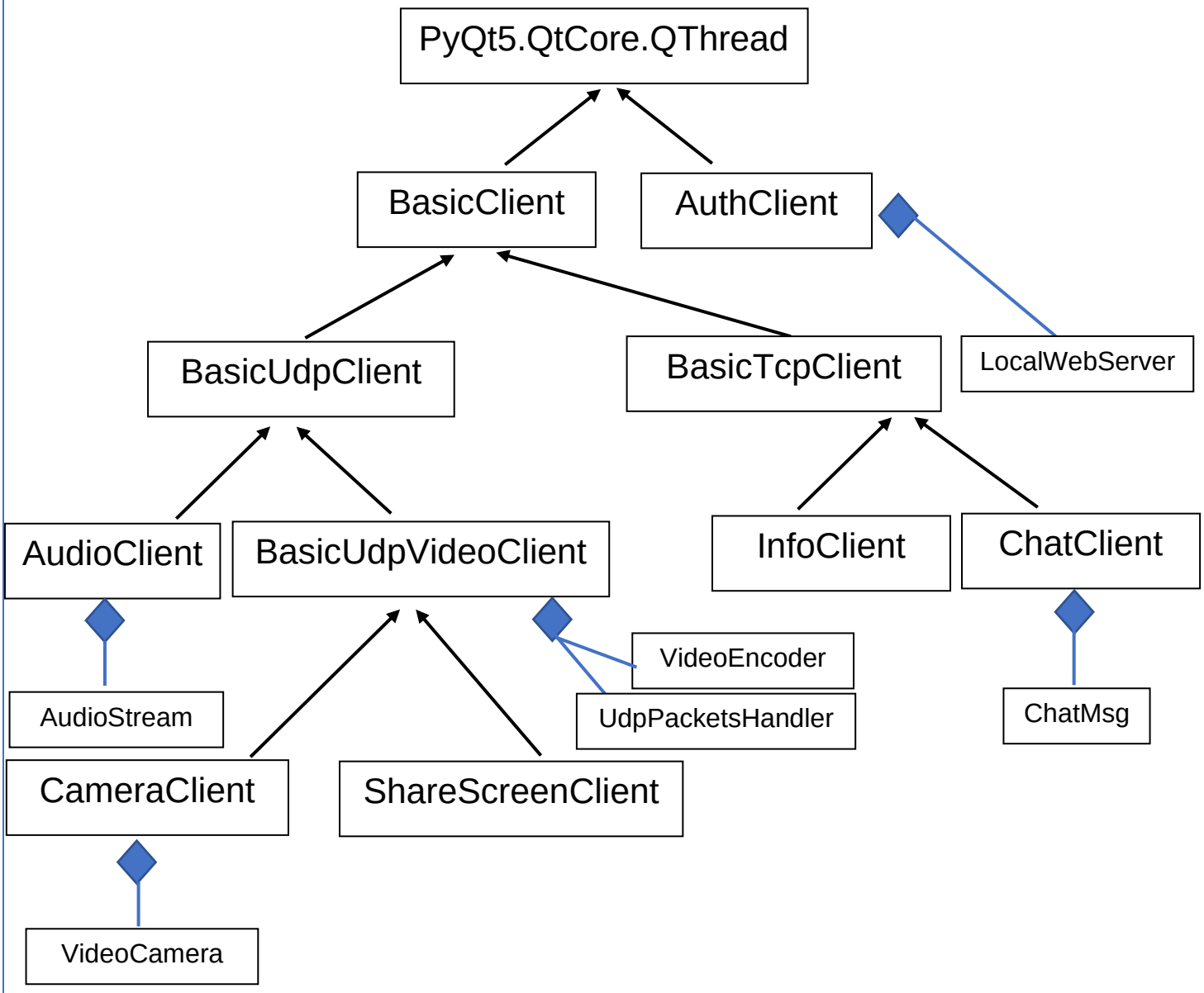
הממשק עובד עם הלקוחות הקטנים, מקבל מהם מידע ומעביר להם מידע בהתאם לפעולות המשתמש.

בממשק קיימים שני חלונות ראשיים:

- **חלון הפתיחה** – מכיל את לקוח ההזדהות, מסך ההזדהות וכן את מסך יצירת הפגישה / הצטרפות לפגישה קיימת. משנה את תוכן החלון שמוצג בהתאם למצב התוכנה – תחילה מציג את מסך ההזדהות ובהמשך מסתיר אותו ומציג את המסך המאפשר הצטרפות לפגישה או יצירת פגישה חדשה.

- **חלון הפגישה** – מכיל את הלקוחות הקטנים (פרט ללקוח ההזדהות שמוכל בחלון הפתיחה), טבלה של מסכי הוידאו של כל לקוח בפגישה לצד שמותיהם וסרגל כלים בתחתית החלון שמכיל את כפתורי השליטה בפגישה.
בנוסף, חלון זה מכיל אובייקטים שונים של הממשק שנבנים בעזרת מחלקות שונות.

Client UML


PyQt5.QtCore.QThread
BasicClient
AuthClient
BasicUdpClient
BasicTcpClient
LocalWebServer
AudioClient
BasicUdpVideoClient
InfoClient
ChatClient
VideoEncoder
AudioStream
UdpPacketsHandler
ChatMsg
CameraClient
ShareScreenClient
VideoCamera

הסבר על מחלקות ה - Client

כל מחלקת client שמתחילה במילה Basic היא מחלקה אבסטרקטית (בעזרת הספרייה abc).

כל מחלקה מכילה קוד שרץ בthread נפרד ולכן כולן יורשות מ-PyQt5.QtCore.QThread, מחלקת threading של PyQt5. הסיבה שהן יורשות ממחלקה זו ולא מ-threading.Thread היא שמוגדרים בהן סיגנלים של PyQt5 מסוג pyqtSignal שבאמצעותם מתבצעת התקשורת עם הGUI. סיגנלים אלה חייבים להיות מוגדרים בתוך מחלקות שיורשות מ-PyQt5.QtCore.QObject, והמחלקה QThread יורשת מהמחלקה QObject.

- `BasicClient` – מחלקה המגדירה תכונות בסיסיות של לקוח המשתמש בסוקטים, כמו ה-id שלו והאם הוא משתף מידע (is_sharing), למשל לקוח האודיו משתף מידע כאשר המיקרופון פתוח ולא משתף כאשר הוא סגור. בנוסף, במחלקה זו מוגדר סיגנל של PyQt5 הנקרא network_error. סיגנל זה משודר אם יש שגיאת תקשורת בלקוח, ומעביר את פרטי השגיאה לחלון הראשי שמציג אותם בחלון שגיאה מתאים. סיגנל זה רלוונטי בעיקר אם השרת נסגר כאשר הלקוח עדיין רץ.
  מחלקה זו מגדירה שתי פעולות אבסטרקטיות שלקוחות שיורשים ממחלקה זו יצטרכו לממש:

```
@abstractmethod
def send_data_loop(self):
```

  - `send_data_loop` – פעולה זו רצה בthread נפרד ושולחת נתונים לשרת בלולאה.

```
@abstractmethod
def receive_data_loop(self):
```

  - `receive_data_loop` - פעולה זו רצה בthread נפרד ומקבלת נתונים מהשרת בלולאה.
- `AuthClient` – מחלקה זו אחראית על תהליך ההזדהות של הלקוח. היא אינה יורשת מ-`BasicClient` מכיוון שבניגוד לשאר הלקוחות שממשים את פרוטוקול התקשורת בעצמם באמצעות סוקטים, התקשורת של תהליך ההזדהות מתבצעת מעל פרוטוקול HTTP בעזרת הספרייה requests. במחלקה זו מוגדרים מספר סיגנלים חשובים:

```
recv_client_info_signal = pyqtSignal(ClientInfo)
network_error = pyqtSignal(str)      # details
invalid_id_error = pyqtSignal(str)   # details
```

  - recv_client_info_signal – משודר כאשר פרטי המשתמש התקבלו מהשרת והומרו לאובייקט מסוג `ClientInfo`.
  - network_error - בדומה ל-`BasicClient` סיגנל זה משודר אם יש שגיאת תקשורת בלקוח, ומעביר את פרטי השגיאה לחלון הפתיחה שמציג אותם בחלון שגיאה מתאים.
  - invalid_id_error – משודר כאשר השרת מחזיר תגובה שמעידה שהוא קיבל id לא קיים (id של לקוח או של פגישה). מעביר את פרטי השגיאה.

כפי שציינתי בהתחלה, תהליך ההזדהות יכול להתבצע בשתי דרכים - עם חשבון ה-Google של המשתמש או עם שמו ולכן מחלקה זו מכילה בין היתר את הפעולות:

```
def google_sign_in(self):
```

- google_sign_in – פעולה זו נקראת כאשר המשתמש לוחץ על הכפתור sign in with google והיא מתחילה את תהליך ההזדהות עם Google שעליו אפרט בהמשך הספר. בשביל תהליך זה, המחלקה מכילה אובייקט מסוג LocalWebServer שהוא בסך הכל שרת HTTP מקומי שרץ בפורט רנדומלי פנוי, מקבל את ה- auth code מהשרת של Google ומעביר אותו למחלקה AuthClient שממשיכה את תהליך ההזדהות.

```
def name_sign_in(self, name: str) -> Union[ClientInfo, None]:
```

- name_sign_in – פעולה זו נקראת כאשר המשתמש לוחץ על הכפתור sign in, מקבלת את השם שהמשתמש הכניס במסך הפתיחה ושולחת אותו לשרת ההזדהות. מחזירה אובייקט מסוג ClientInfo אם ההזדהות התבצעה בהצלחה אחרת None.

```
def send_auth_request(self, endpoint: str, payload: dict) -> \
        Union[ClientInfo, None]:
```

- send_auth_request – שולחת בקשת HTTP מסוג POST לנקודת קצה נתונה בשרת (endpoint) עם תוכן נתון (payload). משדרת סיגנלים מתאימים בהתאם לתגובת השרת. מחזירה אובייקט מסוג ClientInfo שמכיל את פרטי המשתמש אם ההזדהות התבצעה בהצלחה אחרת None.

- BasicTcpClient – מחלקה פשוטה שמגדירה תכונות בסיסיות של לקוח המשתמש בסוקטים מעל TCP: סוקט TCP שמקבל מידע מהשרת (in_socket) וסוקט TCP ששולח מידע לשרת (out_socket). פעולות:

```
def __init__(self, ip: str, in_socket_port: int, out_socket_port:int,
             client_id: bytes, is_sharing=True):
```

- __init__ - יוצרת את הסוקטים ומקשרת אותם לשרת לפי ה-ip וה-port שהיא מקבלת עבור כל סוקט. מקבלת גם את ה-id של הלקוח ושולחת אותו לשרת כדי שיידע במי מדובר. המשתנה is_sharing היה רלוונטי כאשר הוידאו והאודיו היו מעל TCP והוא סימל האם הלקוח משתף מידע כרגע, כרגע הוא תמיד True.

```
def send_packet(self, data: bytes):
```

- send_packet – מקבלת מידע שצריך לשלוח לשרת, יוצרת ממנו פקטה ושולחת אותו לפי הפרוטוקול שאתאר בהמשך.

```
def close(self):
```

- close – שולחת EXIT_SIGN לשרת וסוגרת את הסוקטים.

- BasicUdpClient – מחלקה פשוטה שמגדירה תכונות בסיסיות של לקוח המשתמש בסוקטים מעל UDP: סוקט UDP שמקבל מידע מהשרת (in_socket) וסוקט UDP ששולח מידע לשרת (out_socket). פעולות חשובות:

```
def __init__(self, ip: str, in_socket_port: int, out_socket_port:int,
             client_id: bytes, is_sharing=True):
```

- __init__ - יוצרת את הסוקטים, מקשרת אותם לשרת לפי ה-ip וה-port שהיא מקבלת עבור כל סוקט. מקבלת גם את ה-id של הלקוח שולחת את ההודעה UDP_NEW_CLIENT_MSG מה-in_socket כדי שהשרת יידע לאן לשלוח את המידע.

המשתנה is_sharing מעיד האם הלקוח משתף מידע כרגע (למשל אם המיקרופון פתוח הוא משתף אודיו).

```
def send_data(self, data: bytes):
```

- send_data – שולחת מידע לשרת לפי הפרוטוקול שאתאר בהמשך.

```
def receive_data(self) -> Union[Tuple[bytes, bytes], Tuple[None, None]]:
```

- receive_data – מקבלת מידע מהשרת. מחזירה tuple שמכיל את ה-id של הלקוח ששלח את המידע, ואת המידע. אם המידע לא התקבל מהשרת הנכון, מחזירה (None, None).

```
def close(self):
```

- close – סוגרת את הסוקטים.

- AudioClient – לקוח האודיו שרק מממש את הפעולות האבסטרקטיות של המחלקה BasicClient, כלומר הפעולות ששולחות ומקבלות מידע בלולאה. מחלקה זו מכילה אובייקט מסוג AudioStream שאחראי על האודיו.
- AudioStream – מחלקת עזר פשוטה שיוצרת stream של אודיו בעזרת הספרייה pyaudio וכך קולטת סאונד מהמיקרופון ומשמיעה סאונד ברמקול.
- BasicUdpVideoClient – לקוח בסיסי ששולח ומקבל וידאו מעל UDP. מממש את הפעולות האבסטרקטיות של המחלקה BasicClient בעזרת מחלקות העזר: VideoEncoder, UdpPacketsHandler. במחלקה זו הגדרתי שני סיגנלים חשובים:
  - frame_captured – סיגנל שמשודר כאשר פריים חדש נקלט במצלמה. הוא מעביר את הפריים.
  - frame_receievd – סיגנל זה משודר כאשר פריים מתקבל מלקוח מסוים. מעביר את הפריים ואת ה-id של הלקוח ששלח אותו.

  שני הסיגנלים האלה נקלטים בחלקים אחרים של הממשק שמטפלים בהם ומציגים את הפריימים בהתאם.

  במחלקה זו מוגדרת פעולה אבסטרקטית אחת:

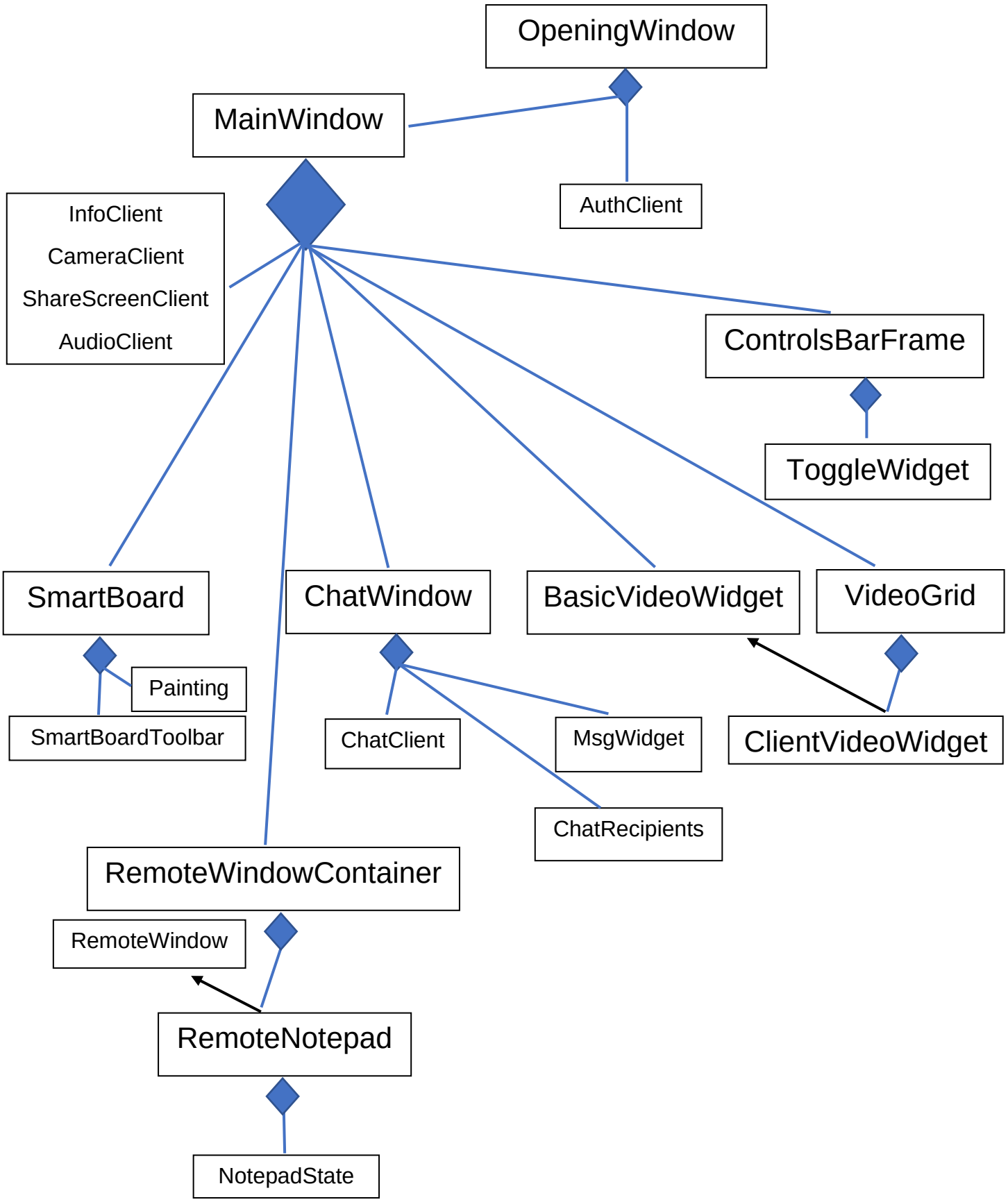
```
@abstractmethod
def get_frame(self):
```

- get_frame – פעולה זו מחזירה פריים שיישלח לשרת.

- VideoEncoder – מחלקת עזר פשוטה שאחראית לדחיסת הוידאו בפורמט JPEG בעזרת הספריות opencv ו-numpy.
- UdpPacketsHandler – מחלקת עזר שאחראית לסדר את פקטות ה-UDP שמתקבלות ולהרכיב את הפריימים בהתאם לפרוטוקול שאתאר בהמשך הספר.
- CameraClient – לקוח המצלמה, מחלקה פשוטה שרק מממשת את הפעולה get_frame - מצלמת פריים מהמצלמה בעזרת מחלקת העזר VideoCamera ומחזירה אותו.
- VideoCamera – מחלקת עזר שמצלמת תמונות מהמצלמה בעזרת הספרייה opencv.
  - ShareScreenClient – לקוח שיתוף המסך, מחלקה פשוטה שרק מממשת את הפעולה get_frame - מצלמת את המסך בעזרת הספרייה Pillow ומחזירה את הפריים.
- InfoClient – לקוח המידע הכללי, מעביר ומקבל הודעות מידע שאפרט עליהן בהמשך (למשל כאשר לקוח חדש מצטרף לפגישה). במחלקה זו מוגדר הסיגנל new_info שמשודר כאשר מידע חדש מתקבל ומעביר את המידע. הוא נקלט בחלון הראשי שמטפל במידע.

- `ChatClient` – לקוח הצ'אט, מקבל ושולח הודעות צ'אט לשרת. במחלקה זו מוגדר הסיגנל new_msg שמשודר כאשר הודעה חדשה מתקבלת והוא מעביר את ההודעה. הוא נקלט בחלון הראשי שמטפל במידע. הוא נקלט בחלון הצ'אט שמציג את ההודעה.
- `ChatMsg` – כל הודעת צ'אט היא אובייקט מטיפוס זה, שמכיל את ה-id של שולח ההודעה והנמען, את טקסט ההודעה ואת זמן השליחה.

GUI UML


OpeningWindow
MainWindow
AuthClient
InfoClient
CameraClient
ShareScreenClient
AudioClient
ControlsBarFrame
ToggleWidget
SmartBoard
ChatWindow
BasicVideoWidget
VideoGrid
Painting
SmartBoardToolbar
ChatClient
MsgWidget
ClientVideoWidget
ChatRecipients
RemoteWindowContainer
RemoteWindow
RemoteNotepad
NotepadState

הסבר על מחלקות ה - GUI

- OpeningWindow – חלון הפתיחה, הקובץ שהלקוח מריץ. בהרצת קובץ זה נוצרת האפליקציה (PyQt5.QtWidgets.QApplication) ומוגדר לה גיליון העיצוב השמור בנתיב STYLE_SHEET_PATH. גיליון עיצוב זה הוא קובץ מסוג qss, שמכיל הגדרות עיצוב של Qt, דומות מאוד ל-css (פורמט נפוץ לעיצוב דפי אינטרנט) למשל הגדרות צבעים, גודל, פונט...
אפרט על חלק מהפעולות במחלקה זו:

```
def __init__(self):
```

  - __init__ – מאתחלת את האובייקט, טוענת קובץ ui שיצרתי בעזרת התוכנה Qt Designer. קובץ זה מכיל אלמנטים שונים של הממשק הגרפי בפורמט XML.
לאחר טעינת הקובץ, פעולה הבנאי יוצרת אובייקט מסוג AuthClient, לקוח ההזדהות שעליו פירטתי בעמודים הקודמים וכן אובייקט מסוג MainWindow, החלון הראשי של התוכנה. בנוסף, פעולה זו מקשרת מספר כפתורים בממשק לפעולות שתקראנה כאשר הכפתורים יילחצו, בעזרת מנגנון סיגנלים של PyQt5.

```
def recv_client_info(self, client_info: ClientInfo):
```

  - recv_client_info - פעולה זו נקראת כאשר ה-AuthClient מקבל מהשרת אובייקט מסוג ClientInfo המכיל את פרטי הלקוח. אם ה-meeting_id של אובייקט זה אינו ריק, סימן שהלקוח הצטרף לפגישה ולכן במקרה זה הפעולה מעבירה את החלון לקדמת החלונות ויוצרת את החלון הראשי. אם עדיין אין ללקוח meeting_id, סימן שהוא רק התחבר לתוכנה ועוד לא נכנס לפגישה ולכן היא תציג את המסך שמאפשר יצירת פגישה והצטרפות לפגישה קיימת.
- MainWindow – החלון הראשי, חלון הפגישה. פעולות חשובות:

```
def __init__(self):
```

  - __init__ – מאתחלת את האובייקט, טוענת קובץ ui של החלון הראשי בדומה לבנאי במחלקה OpeningWindow.

```
def setup(self, client_info: ClientInfo) -> bool:
```

  - setup – מקבלת אובייקט המכיל את פרטי הלקוח וקוראת למספר פעולות שיוצרות את הלקוחות - לקוח מידע כללי, וידאו, אודיו, שיתוף מסך, מאתחלות אובייקטים שונים בGUI ומקשרות בין הסיגנלים של הלקוחות לממשק הגרפי. לאחר מכן, מתחילה כל אחד מהלקוחות בthread נפרד. מחזירה True אם כל אחד מהלקוחות הקטנים הצליח להתחבר לשרת שהוא עובד מולו ו-False אם לא.

```
def init_clients(self) -> bool:
```

  - init_clients – נקראת בבנאי ומאתחלת את הלקוחות. אם במהלך האתחול הייתה שגיאת רשת, כלומר אם אחד הלקוחות לא הצליח להתחבר לשרת שהוא עובד במולו, היא קוראת לפעולה handle_network_error ומחזירה False. אחרת היא מחזירה True.

```
def create_controls_bar(self):
```

  - create_controls_bar – יוצרת את סרגל הכלים התחתון שמכיל את כפתורי השליטה השונים (ControlsBarFrame).

```
def create_video_grid(self):
```

  - create_video_grid – יוצרת את הטבלה שמכילה את הוידאו של כל משתמש בפגישה (VideoGrid).

```
def create_chat_window(self):
```

- create_chat_window – יוצרת את חלון הצ'אט (ChatWindow).

```
def create_shared_screen(self):
```

- create_shared_screen – יוצרת אובייקט מסוג BasicVideoWidget שבו יוצג המסך שאחד הלקוחות יישלח כאשר הוא ישתף מסך.

```
def create_smart_board(self):
```

- create_smart_board – יוצרת את הלוח החכם.

```
def init_remote_window(self):
```

- init_remote_window – יוצרת אובייקט מסוג RemoteWindowContainer שיכיל את החלון המרוחק.

```
def connect_clients(self):
```

- connect_clients – מחברת את הסיגנלים (של PyQt5) שמוגדרים במחלקות הלקוחות לפעולות המתאימות שתקראנה כאשר הסיגנלים ישודרו.

```
def handle_network_error(self, details: str):
```

- handle_network_error – מטפלת בשגיאת רשת שקרתה באחד הלקוחות: מעבירה את החלון לקדמה, מציגה את פרטי השגיאה שהיא קיבלה וסוגרת את התוכנה.

```
def exit(self):
```

- exit – סוגרת את הלקוחות ואת החלון הראשי.

```
def handle_new_info(self, info: tuple):
```

- handle_new_info – מטפלת במידע חדש שהיא מקבלת מה-InfoClient : מעבירה אותו לאובייקט המתאים בממשק שהמידע רלוונטי אליו, למשל מעבירה הודעה שלקוח כיבה את המצלמה ל-VideoGrid.

```
def toggle_sharing(self, toggle_widget: ToggleWidget,
                   ui_widget: QtWidgets.QWidget,
                   start_msg: int, stop_msg: int) -> bool:
```

- toggle_sharing – מנסה להתחיל/להפסיק שיתוף מסוג מסוים (שיתוף מסך/לוח חכם/חלון מרוחק). מקבלת את המתג בממשק שאחראי להתחיל ולהפסיק את השיתוף (toggle_widget), את האובייקט בממשק שהשיתוף מוצג בו (ui_widget) ואת שתי ההודעות שמעידות על התחלת והפסקת השיתוף (start_msg, stop_msg).
  היא בודקת אם אחד הלקוחות משתף כרגע ואם כן מציגה הודעת שגיאה ומחזירה False, מכיוון שרק לקוח אחד יכול לשתף בזמן מסוים. אם אף אחד לא משתף, או שהלקוח הנוכחי משתף ומבקש להפסיק שיתוף, היא שולחת הודעת מידע מתאימה לשרת דרך ה-InfoClient , מתחילה/מפסיקה את השיתוף ומחזירה True.

- ControlsBarFrame – מחלקה קצרה שבונה את סרגל הכלים התחתון המכיל את כפתורי השליטה השונים. רוב הכפתורים הם כפתורי הדלקה/כיבוי מטיפוס ToggleWidget.
- ToggleWidget – כפתור הדלקה/כיבוי. ממחלקה זו נוצרים הכפתורים לשליטה במצלמה, במיקרופון, בהצגת חלון הצ'אט, הלוח החכם והחלון המרוחק. לאובייקט של מחלקה זו יש תכונה חשובה הנקראת toggle_dict, זהו dictionary המכיל את הטקסט והנתיב

```
TOGGLE_AUDIO_DICT = {
    True: ('Mute', f'{PATH_TO_IMAGES}\\open_mic.png'),
    False: ('Unmute', f'{PATH_TO_IMAGES}\\closed_mic.png')
}
```

לתמונה שתוצג על הכפתור עבור כל מצב שלו. לדוגמה אלה הערכים עבור הכפתור לשליטה במיקרופון:
המשמעות היא שכאשר המיקרופון יהיה פתוח, הכפתור יציג את הטקסט Mute ואת התמונה open_mic.png וכאשר המיקרופון יהיה סגור הכפתור יציג את הטקסט Unmute ואת התמונה closed_mic.png.

- VideoGrid – הטבלה שמכילה את הוידאו של כל משתמש בפגישה. הוידאו של כל משתמש הוא אובייקט מסוג ClientVideoWidget.
- ClientVideoWidget – אובייקט וידאו (יורש מ-BasicVideoWidget) שמכיל את הוידאו של כל לקוח, ותגית קטנה בצד שמאל למטה שמכילה את שם הלקוח. אובייקט זה משתנה בהתאם למצב המצלמה והמיקרופון של הלקוח – אם המצלמה פתוחה הוא מציג את התמונה ואילו היא סגורה הוא מציג את שם הלקוח באותיות גדולות על מסך כהה, או את תמונת הפרופיל של חשבו ן ה-Google שלו אם הוא בחר להתחבר איתו. בנוסף, אם המיקרופון של הלקוח סגור, תוצג תמונה קטנה של מיקרופון סגור בתגית השם התחתונה.
- BasicVideoWidget – אובייקט וידאו בסיסי שיורש מ-PyQt5.QtWidgets.QLabel כדי שיוכל לשנות את ה-pixmap שלו וכך להציג תמונות. פעולה חשובה:

```
def show_frame(self, frame: np.ndarray):
```

  - show_frame – מקבלת פריים של opencv (מטיפוס numpy.ndarray), הופכת אותו לאובייקט מסוג PyQt5.QtGui.QPixmap ומציגה אותו.
- ChatWindow – חלון הצ'אט. מכיל את לקוח הצ'אט (ChatClient) וגם את רשימת הנמענים בפגישה (ChatRecipients) עבור האובייקט בחלון הצ'אט שמאפשר לבחור נמען ספציפי. כל הודעת צ'אט שנשלחת או מתקבלת היא אובייקט מסוג
- ChatRecipients – מחלקה פשוטה שאחראית לארגון הנמענים בפגישה עבור חלון הצ'אט.
- ChatMsg – הודעת צ'אט. מכילה את ה-id של המוען והנמען, את טקסט ההודעה וזמן השליחה.
- SmartBoard – הלוח החכם. ניתן לצייר עליו בצבעים שונים ולמחוק אותו. הוא מזהה קווים ישרים ומלבנים שהמשתמש משרטט ומתקן אותם כך שיהיו מדויקים. סרגל הכלים של הלוח החכם הוא אובייקט מסוג SmartBoardToolbar שרק מכיל כפתורים לבחירת צבע ועובי העט וכפתור לניקוי הלוח. כל ציור הוא אובייקט מסוג Painting. פעולות חשובות במחלקת הלוח החכם:

```
def draw_painting(self, painting: Painting):
```

  - draw_painting – מקבלת ציור ומציירת אותו על הלוח.

```
def check_for_line(self) -> bool:
```

  - check_for_line – נקראת כאשר העכבר משוחרר לאחר שהיה לחוץ ובודקת אם הנקודות האחרונות שצוירו על הלוח מייצגות קו ישר בעזרת הפונקציה get_line_equation. אם כן, היא מוחקת אותן, מציירת את הקו הישר, שולחת לשרת את מה שהיא עשתה באמצעות האובייקט Painting ומחזירה True. אחרת מחזירה False.

```
def get_line_equation(self, xs: list, ys: list):
```

  - get_line_equation – מקבלת אוסף של נקודות (שתי רשימה של ערכי ה-x וה-y שלהן) ומחזירה את פונקציה המתארת את משוואת הקו הישר שהן מייצגות (אם קיים כזה) בעזרת הפונקציה numpy.polyfit. כמו כן, על הקו הישר לעמוד במספר

thresholdים, למשל אורך מינימלי. אם היא לא מצאה קו ישר שעומד בקריטריונים היא מחזירה None.

```
def check_for_rect(self) -> bool:
```

- check_for_rect - נקראת כאשר העכבר משוחרר לאחר שהיה לחוץ ובודקת אם הנקודות האחרונות שצוירו על הלוח מייצגות מלבן (ישר), על ידי בדיקת השיפועים של הצלעות – שיפוע הצלע העליונה והתחתונה צריך להיות קרוב לאפס, ושיפוע שתי הצלעות האחרות צריך להיות קרוב ללא מוגדר.

- Painting – אובייקט המתאר ציור מסוים שצויר על הלוח החכם או נמחק ממנו. סוג הציור יכול להיות קו ישר (LINE), מלבן (RECTANGLE), מחיקה של רצף נקודות (CLEAR_POINTS) או ניקוי הלוח (CLEAR_ALL). האובייקט מכיל גם נתונים שמשתנים בהתאם לסוג הציור וגם את עובי וצבע העט שצייר את הציור.
- RemoteWindowContainer – אובייקט פשוט שרק מכיל את החלון המשותף. כאשר הוא זז, או שגודלו משתנה, הוא מעדכן את החלון המשותף כדי שהוא יזוז בהתאם וייתן הרגשה שהוא חלק מהחלון הראשי.
- RemoteNotepad – החלון המשותף בפועל (יורש מ-RemoteWindow), חלון של התוכנה notepad.exe. מחלקה זו מכילה מספר פעולות לסנכרון הטקסט עם החלונות של המשתתפים האחרים. משתמשת במחלקת עזר פשוטה מאוד NotepadState המתארת את מצב ה-notepad (למשל מספר השורות). במחלקה זו מוגדר סיגנל הנקרא new_msg ומשמש להעברת הודעות מסוג RemoteWindowMsg לחלון הראשי שיעביר אותן ל-InfoClient. פעולות חשובות:

```
def create_window(self):
```

- create_window –קוראת לפעולה שנדרסה במחלקת האב ואז משיגה handle ל-edit control של notepad ושומרת אותו במשתנה hwnd_edit. זהו חלון בן של notepad ובו ניתן לערוך את הטקסט. היא מוצאת אותו לפי שם המחלקה שלו – "Edit".

```
def run(self):
```

- run – הלולאה הראשית שאחראית על החלון המשותף, רצה ב-thread נפרד ובודקת כל הזמן אם יש שינויים בטקסט שב-notepad. אם כן, היא שולחת הודעה ספציפית מסוג RemoteWindowMsg שמתארת את השינוי.

```
def msg_to_edit_control(self, msg: int, wparam, lparam) -> int:
```

- msg_to_edit_control - שולחת הודעה מסוימת של Windows ל-Edit Control של notepad ומחזירה ערך החזרה של הפעולה SendMessage.

```
def handle_new_msg(self, msg: RemoteWindowMsg):
```

- handle_new_msg – מטפלת בהודעה מידע מסוג RemoteWindowMsg שהתקבלה מה-InfoClient.

- RemoteWindow – מחלקה המגדירה חלון משותף בסיסי. פעולות חשובות:

```
def create_window(self):
```

- create_window – מריצה את התוכנה הנתונה (self.program_name שנקבעת בבנאי) באמצעות הספרייה subprocess ומשיגה handle אל החלון שנוצר באמצעות הפונקציה get_hwnds_for_pid.

```
@staticmethod
def get_hwnds_for_pid(pid: int) -> list:
```

- get_hwnds_for_pid – מקבלת id של תהליך מסוים (pid) ומחזירה את כל ה-handleים לחלונות ששייכים לאותו תהליך.

```
def set_window_pos(self, x: int, y: int, width: int, height: int):
```

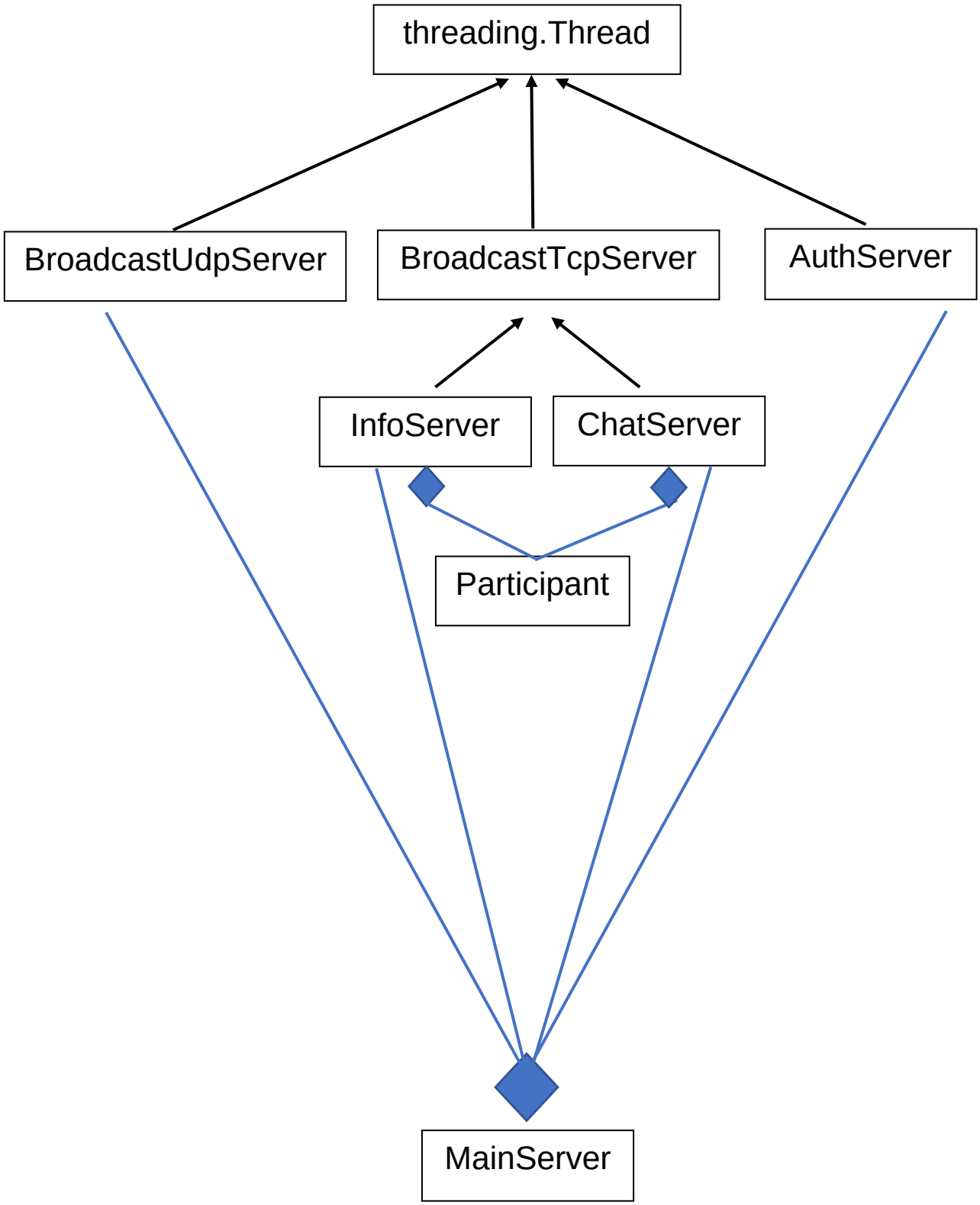
- set_window_pos – משנה את גודל ומיקום החלון (בהתאם לפרמטרים שהיא מקבלת) באמצעות הפונקציה SetWindowPos של WinApi.

```
def close(self):
```

- close – סוגרת את החלון המשותף על ידי שליחת הודעה WM_CLOSE לחלון.

- RemoteWindowMsg – הודעה שנשלחת ברשת לחלון המשותף של שאר המשתתפים בפגישה. מכילה את סוג ההודעה – אחד מהקבועים שהגדרתי במחלקה, ומידע נוסף שמשתנה בהתאם לסוג ההודעה:
  - SET_SELECTION – שינוי בחירת הטקסט הנוכחית. המידע שמועבר הוא האינדקס של התו שממנו תתחיל בחירת הטקסט והאינדקס של התו שעד אליו היא תסתיים.
  - REPLACE_LINE – החלפת שורה ספציפית. המידע שמועבר הוא האינדקס של השורה והשורה החדשה.
  - SET_TEXT – החלפת כל הטקסט. המידע שמועבר הוא הטקסט.

Server UML


threading.Thread
BroadcastUdpServer
BroadcastTcpServer
AuthServer
InfoServer
ChatServer
Participant
MainServer

הסבר על מחלקות ה - Server

כל מחלקת שרת יורשת מהמחלקה threading.Thread מכיוון שכל שרת רץ ב-thread נפרד.

- `MainServer` – השרת הראשי שמריצים, מכיל את כל השרתים הקטנים. פעולות:

```
def __init__(self, ip: str):
```

  - `__init__` - יוצרת כל אחד מהשרתים ומאתחלת אותם עם ה-ip שהיא מקבלת ועם הקבועים השמורים בקובץ server/network_constants.py. יש לציין ששרת הוידאו (מהמצלמה), שרת שיתוף המסך, ושרת האודיו הם אובייקטים של אותה המחלקה – `BroadcastUdpServer`, מכיוון שהתפקוד שלהם זהה.

```
def start(self):
```

  - `start` – מתחילה כל אחד מהשרתים ב-thread נפרד.

```
def client_disconnected(self, client_id: bytes):
```

  - `client_disconnected` – פעולת callback שמועברת ל-`InfoServer` ונקראת כאשר לקוח מתנתק עם ה-id של הלקוח. היא מודיעה לכל אחד משרתי ה-UDP וכן לשרת ההזדהות שהלקוח התנתק כדי שיוציאו אותו מרשימות המשתתפים שלהם.

- `BroadcastUdpServer` – שרת UDP שמקבל מידע ומפיץ אותו לשאר הלקוחות באותה פגישה של הלקוח ששלח את המידע. מכיל שני סוקטים: out_socket שדרכו הוא שולח מידע, ו-in_socket שדרכו הוא מקבל מידע. בנוסף, מכיל dictionary שנקרא client_addresses:

```
# { meeting_id: {client_id: client_in_address} }
self.clients_addresses: Dict[bytes, Dict[bytes, (str, int)]] = {}
```

מילון זה מכיל את ה-id של כל פגישה, לצד מילון המתאר את הלקוחות בפגישה. כל מילון של פגישה מכיל את ה-id של כל לקוח באותה פגישה לצד הכתובת שלו, אליה יש לשלוח לו מידע. פעולות חשובות:

```
def __init__(self, ip: str, client_in_port: int, client_out_port:int,
             server_name: str,
             client_id_validator: Callable[[bytes], bool]):
```

  - `__init__` - יוצרת את הסוקט ששולח מידע (out_socket) ואת הסוקט שמקבל מידע (in_socket) ומקשרת אותם ל-ip והפורטים שהיא מקבלת. מקבלת גם את שם השרת בשביל הדפסות שונות שהוא יבצע וכן פעולת callback שמשמשת לאימות ה-id של כל לקוח חדש.

```
def receive_and_broadcast(self):
```

  - `receive_and_broadcast` – מקבלת מידע מלקוח כלשהו, מפרידה אותו ל-id של הפגישה, ל-id של הלקוח ולתוכן (לפי הפרוטוקול שאתאר בהמשך הספר) ובודקת אם הוא לקוח מוכר (שכבר נמצא בפגישה ידועה ושלח מידע). אם כן, היא קוראת לפעולה `broadcast` שמפיצה את המידע לשאר הלקוחות באותה פגישה. אם המידע התקבל מלקוח שאינו מוכר, היא בודקת אם הוא שווה ל-`UDP_NEW_CLIENT_MSG`. הודעה זו נשלחת על ידי לקוח חדש שרוצה להתחבר לשרת ה-UDP ולכן במקרה זה היא תוסיף את הלקוח למילון הלקוחות. בכל מקרה אחר, זו כנראה פקטה מעוותת ולכן הפעולה תתעלם ממנה.

```
def broadcast(self, meeting_id: bytes, sender_client_id: bytes,
              packet: bytes):
```

- broadcast – פעולה פשוטה שמקבלת מידע ואת ה-id של הלקוח ששלח אותו ומפיצה אותו לכל שאר הלקוחות.

```
def client_disconnected(self, full_client_id: bytes):
```

- client_disconnected – פעולה זו נקראת על ידי ה-MainServer כאשר לקוח מתנתק מהפגישה. היא מקבלת את ה-id ה"מלא" שלו שמורכב מה-id של הפגישה בה הלקוח נמצא ומה-id של הלקוח. היא מסירה את הלקוח ממילון הלקוחות של שרת ה-UDP.

- BroadcastTcpServer – שרת TCP שמקבל מידע ומפיץ אותו לשאר הלקוחות באותה פגישה של הלקוח ששלח את המידע. מכיל שני סוקטים: accept_clients_in – סוקט שמקבל את ה-in_socket של כל לקוח, ו-accept_clients_out – סוקט שמקבל את ה-out_socket של כל לקוח. בנוסף מכיל dictionary שנקרא participants:

```
# { meeting_id: {client_id: Participant(...), ...} }
self.participants: Dict[bytes, Dict[bytes, Participant]] = {}
```

מילון זה מכיל את ה-id של כל פגישה, לצד מילון המתאר את הלקוחות בפגישה. כל מילון של פגישה מכיל את ה-id של כל לקוח באותה פגישה לצד אובייקט מסוג Participant שמתאר משתתף בפגישה. פעולות חשובות:

```
def handle_participant(self, par: Participant):
```

- handle_participant – מקבלת אובייקט שמתאר משתתף מסוים בפגישה ומטפלת במשתתף: מקבלת ממנו מידע וקוראת לפעולה שמטפלת במידע (handle_new_data) עד שהמשתמש שולח EXIT_SIGN, או עד שהוא מתנתק. לבסוף קוראת לפעולה par_disconnected.

```
def handle_new_data(self, par: Participant, data: bytes):
```

- handle_new_data – מקבלת אובייקט שמתאר משתתף בפגישה ואת המידע שהוא שלח. מטפלת במידע החדש - פעולה זו נדרסת על ידי המחלקות שיורשות ממחלקה זו, המימוש במחלקה הזאת פשוט מפיץ את המידע לשאר הלקוחות בפגישה.

```
def par_disconnected(self, par: Participant):
```

- par_disconnected – סוגרת את הסוקטים של משתתף מסוים ומסירה אותו ממילון המשתתפים של הפגישה.

```
def broadcast(self, sender_par: Participant, packet: bytes):
```

- broadcast – מקבלת פקטה של מידע ואת הלקוח ששלח אותה, ומעבירה אותה לכל שאר הלקוחות בפגישה.

- Participant – מחלקה המתארת לקוח בפגישה. מכילה מספר תכונות וביניהן כתובת הלקוח, ה-id שלו, הסוקטים שלו ומנעול (threading.Lock) כדי למנוע מצב בו מספר thread ים ישלחו מידע דרך הסוקט של הלקוח במקביל.
- InfoServer – שרת שיורש מ-BroadcastTcpServer, שולח ומקבל מידע כללי. הוא מפיץ כל מידע שהוא מקבל לשאר הלקוחות באותה פגישה של הלקוח השולח. מכיל רשימה של הודעות הסטטוס האחרונות של השיתופים השונים (שיתוף מסך/לוח חכם/חלון מרוחק) הנקראת last_status_msgs. הודעות אלה תשלחנה ללקוח חדש שיצטרף לפגישה וכך הוא יתעדכן במה שמתרחש. למשל אם אחד המשתתפים משתף מסך ולקוח חדש מצטרף לפגישה באמצע השיתוף עליו לדעת ששיתוף מסך פעיל כרגע ולכן הוא צריך להציג את המסך. פעולות חשובות:

```
def sync_info(self, new_par: Participant):
```

- sync_info – מקבלת אובייקט המתאר את הלקוח החדש שהצטרף ומסנכרנת את המידע בינו לבין שאר הלקוחות בפגישה.

```
def handle_new_data(self, par: Participant, data: bytes):
```

- handle_new_data – דורסת את הפעולה ב-BroadcastTcpServer, מטפלת במידע חדש שהתקבל (data) ממשתתף מסוים (par), בהתאם לסוג המידע הכללי ומפיצה אותו לשאר הלקוחות באותה פגישה של השולח. למשל אם לקוח כיבה את המצלמה היא מעדכנת את אובייקט ה-Participant המתאים כדי שמשתמש שיצטרף לפגישה בעתיד יידע שהמצלמה של משתתף זה כבויה.

- ChatServer – שרת פשוט שיורש BroadcastTcpServer, שולח ומקבל הודעות צ'אט. פעולה חשובה:

```
def handle_new_data(self, par: Participant, data: bytes):
```

- handle_new_data – דורסת את הפעולה ב-BroadcastTcpServer, מקבלת הודעת צ'אט (data) ממשתתף מסוים (par) ושולחת אותה לנמען המתאים או לכל המשתתפים בפגישה בהתאם למה שמצוין בה.

- AuthServer – שרת HTTP שאחראי על תהליך ההזדהות. בנוי בעזרת הספרייה flask. מכיל שני מילונים למעקב אחר ה-idים של הפגישות ושל הלקוחות בכל פגישה:

```
# { client_id: ClientInfo(...) }
self.authenticated_clients: [bytes, ClientInfo] = {}

# { meeting_id: [first_client_id, second_client_id, ...] }
self.meetings_dict: [bytes, [bytes]] = {}
```

המילון authenticated_clients מכיל את ה-id של כל לקוח שמתחבר לשרת לצד אובייקט שמכיל את פרטי הלקוח.

המילון meetings_dict מכיל את ה-id של כל פגישה לצד רשימה של ה-idים של הלקוחות באותה פגישה.

שרת זה יוצר מספר נקודות קצה שאליהן ניתן לשלוח בקשות מסוג POST ולקבל תגובות JSON. נקודות הקצה משמשות לביצוע פעולות שונות:

- /auth/google – הזדהות עם חשבון הגוגל של המשתתף ולכן היא זמינה רק אם קיימים משתני הסביבה: GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET שהכרחיים לביצוע תהליך הזדהות זה. אני טענתי אותם מקובץ env. בעזרת הספרייה python-dotenv.
- /auth/name – הזדהות עם השם של המשתמש.
- /new-meeting – יצירת פגישה חדשה.
- /join-meeting – הצטרפות לפגישה קיימת.
- /logout – התנתקות, כלומר מחיקת ה-id של המשתתף.

פעולות חשובות:

```
def run(self):
```

- run – מריצה את אפליקציית ה-Flask של השרת.

```
def google_auth(self) -> Response:
```

- google_auth – מבצעת את תהליך ההזדהות עם גוגל. פעולה זו נקראת כאשר פונים לנקודת הקצה המתאימה שתיארתי למעלה. מחזירה אובייקט מסוג Response של flask.

```
def name_auth(self) -> Response:
```

- `name_auth` – מבצעת את תהליך ההזדהות עם השם של המשתמש. פעולה זו נקראת כאשר פונים לנקודת הקצה המתאימה שתיארתי למעלה. מחזירה אובייקט מסוג Response של flask.

```
def new_meeting(self) -> Response:
```

- `new_meeting` – מקבלת את ה-id של הלקוח ויוצרת עוברה פגישה חדשה. מחזירה לו את מזהה הפגישה.

```
def join_meeting(self) -> Response:
```

- `join_meeting` – מקבלת מהלקוח את ה-id של פגישה קיימת ומצרפת אליה את הלקוח.

```
def validate_client_id(self, full_client_id: bytes) -> bool:
```

- `validate_client_id` – מקבלת את ה-id ה"מלא" של הלקוח – ה-id של הפגישה בה הוא נמצא וה-id שלו, ובודקת אם הוא קיים במבני הנתונים של השרת. פעולה זו היא ה-callback המועברת לשאר השרתים לאימות ה-id של הלקוחות שמתחברים אליהם.

עיצוב נתונים ופרוטוקולים

Auth

כפי שציינתי המשתמש יכול להתחבר לתוכנה בשתי דרכים, עם חשבון הגוגל שלו או עם השם שלו.

תהליך האימות שלי עם Google מתבצע לפי הפרוטוקול OAuth 2.0 עבור אפליקציות Desktop, אותו מימשתי בעצמי.

ראשית יצרתי פרויקט ב-Google API Console, ויצרתי authorization credentials - GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET. אלה בעצם מזהים ייחודיים שמשמשים לזיהוי התוכנה שלי על ידי השרת OAuth 2.0 של גוגל.

לאחר מכן, מימשתי את התהליך בתוכנה שלי לפי השלבים הבאים:

1. המשתמש מופנה לכתובת https://accounts.google.com/o/oauth2/v2/auth בדפדפן עם מספר פרמטרים, ביניהם ה-GOOGLE_CLIENT_ID שמזהה את התוכנה שלי וה-redirect_uri. פרמטר זה קובע כיצד השרת של גוגל יחזיר תשובה לאפליקציה. קבעתי אותו להיות Loopback IP address הנראית כך: `http://127.0.0.1:port` כאשר **port** הוא פורט רנדומלי פנוי במחשב של הלקוח בו יאזין שרת HTTP מקומי (אובייקט מסוג המחלקה `LocalWebServer` שיצרתי). שרת זה יקבל את התשובה של השרת של גוגל ויחזיר ללקוח דף פשוט שיוצג בדפדפן (שמנחה את הלקוח לחזור לתוכנה).
2. המשתמש מכניס את האימייל והסיסמה שלו בדף ההזדהות של גוגל ומאשר לתוכנה שלי לגשת לפרטי המשתמש שלו (שם, תמונת פרופיל...).
3. השרת של גוגל מחזיר לשרת המקומי שיצרתי במחשב של הלקוח את ה-authorization code שלו והוא מעביר את זה ל-`AuthClient`.
4. ה-`AuthClient` שולח את ה-authorization code ל-`AuthServer` (השרת שלי) בתוך גוף בקשה מסוג HTTP POST request.
5. ה-`AuthServer` שולח לשרת של גוגל את ה-authorization code ומחליף אותו ב-access token.
6. ה-`AuthServer` שולח בקשה לAPI של גוגל המספק מידע על משתמשים בהינתן access token מסוים.
7. ה-`AuthServer` מקבל את פרטי המשתמש מהשרת של גוגל, יוצר למשתמש id עבור הפגישה ומחזיר את המידע ל-`AuthClient`.

דרך ההתחברות השנייה, עם שם המשתמש בלבד, פשוטה בצורה משמעותית:

1. ה-`AuthClient` שולח את השם שהמשתמש הזין בחלון הפתיחה ל-`AuthServer` בתוך גוף בקשה מסוג HTTP POST request.
2. ה-`AuthServer` מקבל את שם המשתמש, יוצר לו id ומחזיר אותו ל-`AuthClient`.

בשתי דרכי ההזדהות ה-`AuthServer` מחזיר ל-`AuthClient` תגובת JSON שה-`AuthClient` מתרגם לאובייקט מסוג `ClientInfo`.

תגובה זו מכילה בין היתר את השדות:

- id - ה-id הייחודי שהשרת יוצר עבור המשתמש.
- name - שם המשתמש.
- img_url - ה-URL לתמונת הפרופיל של המשתמש (רלוונטי רק עבור הזדהות עם גוגל, במקרה אחר שדה זה יהיה סטרינג ריק).

ניהול פגישות

שתי בקשות נוספות בהן מטפל שרת ה-Auth הן יצירת פגישה חדשה והצטרפות לפגישה קיימת. בשתי הבקשות השרת מקבל מהלקוח את ה-id שהוא נתן לו בתהליך ההזדהות. כל id נשלח בצורה של ספרות hex משום שהוא במקור רצף בתים ולכן לא ניתן להעבירו בפורמט JSON. בבקשת הצטרפות לפגישה קיימת הלקוח שולח גם את ה-id של הפגישה.

בכל מקרה, השרת מאמת את ה-id של הלקוח ומחזיר לו את אחת התגובות הבאות:

- **תגובת הצלחה** – מכילה את פרטיו המעודכנים של הלקוח, כלומר את ה-meeting_id החדש שלו וכן ה-id המעודכן שלו. ה-id המעודכן נראה כך:
<meeting_id><client_id>

  כאשר client_id הוא ה-id ההתחלתי, "הגולמי", שהשרת נתן ללקוח בתהליך ההזדהות.

  החלטתי לעדכן את ה-id של הלקוח כדי שכל בקשה עתידית שהוא יישלח לאחד השרתים עם ה-id שלו תכלול כבר את ה-id של הפגישה בה הוא נמצא, מבלי שהוא יצטרך להתעסק בחיבור ה-idים בעצמו. כך השרתים יכולים לזהות במהירות באיזו פגישה הלקוח נמצא ולהעביר את ההודעות שהוא שולח לשאר הלקוחות באותה הפגישה.

- **תגובת שגיאה** – מכילה שדה אחד הנקרא message לצד הודעת שגיאה מתאימה וסטטוס שגיאה תואם. למשל, עבור בקשה של לקוח להצטרף לפגישה עם id לא תקין, כלומר פגישה שאינה קיימת, השרת יחזיר לו את הודעת השגיאה " Invalid meeting ID.".

TCP Sockets

הפעולות והקבועים הקשורים לפרוטוקול זה נמצאים בתיקייה networks.

כל הודעת נתונים ברשת הנשלחת מעל TCP בנויה בצורה הבאה:

<שאר הבתים - תוכן ההודעה> <4 בתים ראשונים – אורך ההודעה>

הבתים המציינים את אורך ההודעה מסודרים לפי הקבוע NETWORK_BYTES_FORMAT.

- ההודעה הראשונה של כל לקוח מסוג `BasicTcpClient` שמתחבר לשרת מסוג `BroadcastTcpServer` היא ה-id של הלקוח.

תוכן שאר ההודעות משתנה בהתאם לסוג ההודעה.

- הודעות מידע כלליות, שנשלחות על ידי ה - `InfoClient` הן מספרים קבועים השמורים בקובץ network/custom_messages/general_info.py.
  כל הודעה מסוג זה בנויה בצורה הבאה:

שאר הבתים - > <מספר המציין את סוג ההודעה> <4 בתים ראשונים – אורך ההודעה> <תוכן ההודעה>

אפרט בקצרה על כל הודעה:

תוכן רוב ההודעות הוא ה-`id` של הלקוח בהן הן עוסקות, אשר משמש לזיהוי הלקוח. `Id` זה הוא רצף בתים שהשרת נותן לכל לקוח חדש שמצטרף לפגישה.

כל אובייקט שנשלח עובר סריאליזציה (`serialization`) לפני השליחה, ו-`deserialization` לאחר הקבלה, באמצעות הספרייה `pickle`.

`serialization` הוא תהליך של תרגום מבני נתונים או מצב של אובייקטים לפורמט שניתן לאחסן או להעביר על גבי רשת מחשבים (הספרייה `pickle` מתרגמת כל אובייקט לרצף בתים) ו-`deserialization` הוא התהליך ההפוך.

סוג ההודעה	תוכן ההודעה	הסבר
`NEW_CLIENT`	אובייקט מסוג `ClientInfo`.	לקוח חדש הצטרף לפגישה. האובייקט שנשלח מכיל מידע אודות הלקוח, בין היתר השם שלו וה-id הייחודי שהשרת נתן לו.
`CLIENTS_INFO`	רשימה של אובייקטים מסוג `ClientInfo`.	הודעה המכילה את כל הנתונים של הלקוחות בפגישה – נשלחת לכל לקוח חדש שמצטרף.
`CLIENT_LEFT`	ה-`id` של הלקוח.	לקוח עזב את הפגישה.
`TOGGLE_AUDIO`	ה-`id` של הלקוח.	לקוח הדליק/כיבה את המיקרופון שלו.
`TOGGLE_VIDEO`	ה-`id` של הלקוח.	לקוח הדליק/כיבה את המצלמה שלו.
`START_SCREEN_SHARING`	ה-`id` של הלקוח.	לקוח התחיל שיתוף מסך.

STOP_SCREEN_SHARING	ה-id של הלקוח.	לקוח הפסיק שיתוף מסך.
START_PAINTING	ה-id של הלקוח.	לקוח התחיל את שיתוף הלוח החכם והתחיל לצייר.
STOP_PAINTING	ה-id של הלקוח.	לקוח סגר את שיתוף הלוח החכם והפסיק לצייר.
NEW_PAINTING	אובייקט מסוג Painting.	לקח צייר ציור חדש על הלוח החכם. האובייקט שנשלח מכיל את פרטי הציור החדש שצויר.
START_REMOTE_WINDOW	ה-id של הלקוח.	לקוח התחיל שיתוף מסוג חלון מרוחק.
STOP_REMOTE_WINDOW	ה-id של הלקוח.	לקוח הפסיק שיתוף מסוג חלון מרוחק.
REMOTE_WINDOW_MSG	אובייקט מסוג RemoteWindowMsg.	ההודעה שיש לשלוח לחלון המרוחק (למשל לשנות את הטקסט בשורה 0 לטקסט XXX).

- הודעות צ'אט שנשלחות על ידי ה – ChatClient הן אובייקטים מסוג ChatMsg, שמכילים את ה-id של שולח ההודעה והנמען וכן את הטקסט והזמן שבו נשלחה על ההודעה.
- אובייקטים מסוג Painting נוצרים על ידי הלוח החכם ונשלחים דרך ה-InfoClient. הם מתארים ציור מסוים שצויר על הלוח.
- אובייקטים מסוג RemoteWindowMsg נוצרים על ידי החלון המרוחק המשותף ונשלחים גם הם דרך ה-InfoClient. הם מתארים הודעה שיש לשלוח לחלון המרוחק כדי לסנכרן את מצבו עם החלונות האחרים.
- לקוח שרוצה להתנתק משרת מסוג BroadcastTcpServer שולח לו EXIT_SIGN.

UDP Sockets

- כל הודעת נתונים ברשת מעל UDP בנויה בצורה הבאה:

שאר הבתים > <id של שולח ההודעה> <4 בתים ראשונים – אורך הid של שולח ההודעה> <- תוכן ההודעה

- כל לקוח שרוצה להתחבר לשרת מסוג `BroadcastUdpServer` שולח מה-in_socket (הסוקט שממנה הוא יקבל מידע) שלו את ההודעה `UDP_NEW_CLIENT_MSG` בתבנית שמתוארת למעלה. כך השרת יודע את הכתובת של הלקוח שאליה הוא יישלח מידע בהמשך ואת ה-id שלו. את שאר המידע הלקוח שולח דרך ה-out_socket שלו.

תוכן ההודעה משתנה בהתאם לסוג ההודעה.

- אודיו – תוכן ההודעה הוא פשוט הבתים שנקלטו מהמיקרופון בעזרת הספרייה pyaudio.
- וידאו - כל פריים מחולק למספר פקטות מסוג `UdpPacket`, והתוכן של כל פקטה נשלח בתבנית שתיארתי למעלה.

כל `UdpPacket` מכילה header שכולל מספר ערכים שמאפשרים למקבל הפקטות לסדר אותן מחדש במידה וסדר הפקטות השתנה בדרך, או לזרוק את הפקטה במידה וכבר הגיע פריים חדש יותר.

המחלקה `UdpPacketsHandler` אחראית לסידור הפקטות והרכבת הפריים.

היא עושה זאת באמצעות הערכים הקיימים בכל פקטה:

- האינדקס של הפריים הנוכחי, כך שאם הגיע פריים חדש יותר, מקבל הפקטות יידע להתעלם מכל הפקטות של הפריים הישן.
- האינדקס של הפקטה הנוכחית כדי לאפשר סידור מחדש של הפקטות במידה וסדר הפקטות השתנה בדרך.
- מספר הפקטות שמרכיבות פריים אחד, כדי שמקבל הפקטות יידע מתי הוא יכול להרכיב את הפריים מהפקטות.
- גודל המידע בפקטה.

לאחר הheader מגיע המידע שהפקטה מכילה והוא חלק מסוים מהתמונה הדחוסה שנשלחת.

הסבר על המחלקה `UdpPacketsHandler`

כפי שציינתי, מחלקה זו אחראי לסידור פקטות ה-UDP בסדר הנכון והרכבת הפריים.

היא מכילה שתי פעולות חשובות:

- `create_packets` – פעולה סטטית שמקבלת את האינדקס של הפריים ואת הפריים הדחוס (bytes). היא מחלקת אותו למספר פקטות מסוג `UdpPacket` כך שכל אחת מהן מכילה חלק אחר מהפריים.
- `process_packet` – פעולה שמקבלת פקטה מסוג `UdpPacket` ומטפלת בה. אם זאת הפקטה הראשונה שנשלחת או פקטה של פריים חדש יותר (האינדקס של הפריים בה גדול מהאינדקס של הפריים הנוכחי) הפעולה זורקת את כל הפקטות הישנות ומתחילה להרכיב את הפריים החדש. ראשית היא שומרת את האינדקס של הפריים החדש ואת מספר הפקטות שמרכיבות ואתו (ערכים אלה שמורים בכל פקטה כפי שציינתי מקודם). לאחר מכן, היא יוצרת רשימה בגודל של מספר הפקטות שמרכיבות את הפריים ומתחילה למלא אותה בהדרגה.
  כל פעם שמתקבלת פקטה שהאינדקס של הפריים שלה זהה לאינדקס של הפריים הנוכחי, הפעולה מכניסה את התוכן שלה למקום המתאים ברשימה המכילה את כל חלקי הפריים. בנוסף, הפעולה מקטינה את מספר הפקטות שנשארו לפריים ב-1. מספר זה שווה בהתחלה למספר הפקטות בפריים וכאשר הוא מגיע לאפס, הפעולה מחברת את כל האיברים ברשימה המכילה את חלקי הפריים ומחזירה פריים שלם.

בעיות ופתרונן

אפרט כעת על מספר בעיות שעלו במהלך כתיבת בפרויקט ופתרונן:

- בפרויקט קיימים ערוצי תקשורת רבים, מסוגים שונים – אודיו, וידאו, שיתוף מסך, הודעות מידע כלליות, הודעות צ'אט ועוד... כדי שהם לא יתערבבו ויפריעו אחד לשני בחרתי להפריד את השרת למספר שרתים קטנים שכל אחד מהם יהיה אחראי על ערוץ תקשורת מסוים וכך גם הלקוח מחולק למספר לקוחות קטנים. כל לקוח קטן עובר מול שרת קטן שמתאים לו. הפרדה זו עשתה סדר במערכת והגדירה בצורה ברור באיזה ערוץ תקשורת עובר כל סוג מידע. בנוסף, רוב הלקוחות הקטנים שולחים ומקבלים מידע, למשל כאשר המצלמה והמיקרופון פתוחים לקוחות האודיו והוידאו שולחים ומקבלים כל הזמן. לכן הפרדתי גם אותם לשני ערוצים: ערוץ מידע שנכנס דרך סוקט אחד (in_socket) וערוץ מידע שיוצא דרך סוקט אחר (out_socket).

- בעיה נוספת שהייתה לי הייתה כיצד להעביר וידאו על גבי UDP. לאחר חשיבה מרובה, החלטתי לחלק כל פריים למספר פקטות מסוג `UdpPacket`. כל אחת מהן מכילה חלק מהפריים ו-header הכולל מספר ערכים שמאפשרים לסדר את הפקטות מחדש במידה והסדר שלהן השתנה בדרך. המחלקה `UdpPacketsHandler` אחראית לסידור הפקטות ועל כך פירטתי בעמוד הקודם.

- במהלך כתיבת הלקוח נתקלתי בבעיה הידועה בשם metaclass conflict. בעיה זו מתרחשת לעיתים כאשר ישנה ירושה מרובה. המחלקה `BasicClient` שכתבתי מתארת לקוח בסיסי שרץ ב-thread אחר ומכיוון שמוגדרים בה סיגנלים של PyQt5 היא חייבת לרשת מ-QObject. משתי הסיבות האלה בחרתי שהיא תירש PyQt5.QtCore.QThread (שיורש מ-QObject) כפי שתיארתי בהסבר על המחלקות. עד כאן הכל בסדר. אך רציתי גם שמחלקה זו תהיה מחלקה אבסטרקטית שתגדיר פעולות אבסטרקטיות שכל לקוח שיירש ממנה יצטרך לממש. לשם כך השתמשתי בספרייה abc והגדרתי שהמחלקה `BasicClient` תירש גם מהמחלקה abc.ABC שבעזרתה ניתן ליצור מחלקה אבסטרקטית בפייתון. אך כעת התעוררה הבעיה - " metaclass conflict: the metaclass of a derived class must be a (non-strict) subclass of the metaclasses of all its bases".

  לאחר חיפוש באינטרנט למדתי מספר דברים. ראשית אגדיר בקצרה מהי metaclass. כל דבר בפייתון הוא אובייקט, אפילו מחלקות הן אובייקטים. כולנו יודעים שיוצרים אובייקט בעזרת מחלקה. אבל איך יוצרים מחלקה? בעזרת metaclass.

  הבעיה שקרתה לי היא שניסיתי לרשת משתי מחלקות שיש להן metaclasses שונות. לכן פייתון לא יכול להחליט מי היא ה-metaclass של המחלקה החדשה שנוצרת. הפתרון לבעיה זו הוא פשוט בצורה מפתיעה: יוצרים metclass שיורשת משתי ה-metaclasses השונות וקובעים אותה להיות ה-metclass של המחלקה החדשה.

הצעות לשיפור

פגישה בין לקוחות שנמצאים ברשתות שונות

כרגע הפרויקט שלי עובד ברשת המקומית. הייתי רוצה לשדרג אותו ולאפשר פגישה בין מספר לקוחות שנמצאים ברשתות שונות.

שיפור מהירות הוידאו

כרגע הוידאו של המצלמות יחסית מהיר, אך יש מקום לשיפור מהירות שיתוף המסך. אני חושב שניתן לשפר את מהירות הוידאו על ידי שימוש באלגוריתמי דחיסה מהירים יותר מאלה שבהם השתמשתי וכך לשפר את חוויית המשתמש.

סיכום

על אף שתחילה התלבטתי אם אני רוצה לבחור בנושא זה כנושא הפרויקט, לבסוף נהניתי מהכנת הפרויקט ולמדתי ממנו הרבה.

במהלך הכנתו התעמקתי בנושאים חדשים שפחות התנסיתי בהם בעבר, למשל העברת וידאו ואודיו ועבודה עם WinApi. כמו כן, למדתי לכתוב ממשק משתמש עשיר בפייתון.

את הקוד המלא של הפרויקט ניתן למצוא ב-github שלי:

https://github.com/HadarShahar/zoom

תודות

אני רוצה להודות למנחת הפרויקט שלי ולמורתי למחשבים אילת משיח, שלימדה אותי בשנים האחרונות וליוותה אותי לאורך כל הדרך. אני מודה לה מאוד על למידה מהנה וחווייתית שהייתה בזכותה בשנים האחרונות.

כמו כן, אני רוצה להודות לחברי הכיתה שלמדו לצידי ויצרו אווירה טובה ומהנה בשיעורים.