

פרויקט גמר בתוכנית 'גבהים' מערכת cloud





מגיש: דניאל וולקוביץ

ת.ז: 207257668

שם המנחים: יאיר מירסקי, חגי לוי ושאול שמולה

תאריך הגשה: ד' סיון התשע"ו (10.6.2016)

בית הספר: ישיבת תיכונית חורב ירושלים

תוכן עניינים

[תקציר כולל ורציונל הפרויקט 4](#_Toc111982630)

[מבוא ורקע כללי 5](#_Toc111982631)

[תיחום הפרויקט 6](#_Toc111982632)

[תיאור המוצר המוגמר 7](#_Toc111982633)

[שם המוצר 7](#_Toc111982634)

[מטרת המוצר 7](#_Toc111982635)

[מטרות מרכזיות 7](#_Toc111982636)

[שאלות שהמערכת עונה עליהן 7](#_Toc111982637)

[השוואת העבודה עם פתרונות ויישומים קיימים 7](#_Toc111982638)

[מבט אישי על העבודה ועל פיתוחה 8](#_Toc111982639)

[אתגרים 8](#_Toc111982640)

[התמודדות עם קשיים 8](#_Toc111982641)

[הערכת הפתרון לעומת התכנון והמלצות לשיפורו 8](#_Toc111982642)

[תיאור הממשק למשתמש 9](#_Toc111982643)

[מידע בסיסי 9](#_Toc111982644)

[דוגמאות להודעות חזותיות בצד הלקוח 9](#_Toc111982645)

[הסנכרון 10](#_Toc111982646)

[סיום התוכנית 10](#_Toc111982647)

[אמצעי הקלט בצד הלקוח 10](#_Toc111982648)

[סביבת העבודה 11](#_Toc111982649)

[סביבת פיתוח 11](#_Toc111982650)

[סביבת הרצה 11](#_Toc111982651)

[תיאור המערכת 12](#_Toc111982652)

[סקירת כל המודולים המרכיבים את המוצר וקשרי הגומלין ביניהם 12](#_Toc111982653)

[**מבט על** 12](#_Toc111982654)

[פירוט רכיבי המערכת 14](#_Toc111982655)

[פירוט ותיאור המודולים 19](#_Toc111982656)

[מודולים מובנים 19](#_Toc111982657)

[מודולים שאינם מובנים ושלא נכתבו על ידי 20](#_Toc111982658)

[מודולים שאינם מובנים ונכתבו על ידי 20](#_Toc111982659)

[תיעוד הקוד 22](#_Toc111982660)

[תיעוד הפונקציות 22](#_Toc111982661)

[פונקציות עיקריות במודול Cloud\_client 22](#_Toc111982662)

[פונקציות ומחלקות עיקריות במודול cloud\_server: 24](#_Toc111982663)

[פונקציות ומחלקות עיקריות במודול Folder\_monitor 24](#_Toc111982664)

[פונקציות עיקריות במודול Utils 25](#_Toc111982665)

[פונקציות ומחלקות עיקריות במודול User\_system 26](#_Toc111982666)

[פונקציות ומחלקות עיקריות במודול Folder\_manager 26](#_Toc111982667)

[פירוט מאגרי המידע של השרת 28](#_Toc111982668)

[האלגוריתמים המרכזיים בפרויקט 30](#_Toc111982669)

[סנכרון 30](#_Toc111982670)

[תיאור וניסוח הבעיה האלגוריתמית 30](#_Toc111982671)

[תיאור אלגוריתמים קיימים לפתרון הבעיה 30](#_Toc111982672)

[הפתרון הנבחר 30](#_Toc111982673)

[סנכרון בדגש סנכרון תיקיות 30](#_Toc111982674)

[תיאור וניסוח הבעיה האלגוריתמית 30](#_Toc111982675)

[תיאור אלגוריתמים קיימים לפתרון הבעיה 30](#_Toc111982676)

[הפתרון הנבחר 31](#_Toc111982677)

[שליחת קבצים ומשתנים על גבי הרשת 31](#_Toc111982678)

[תיאור וניסוח הבעיה האלגוריתמית 31](#_Toc111982679)

[הפתרון הנבחר 31](#_Toc111982680)

# תקציר כולל ורציונל הפרויקט

פרויקט זה הוא מערכת חכמה שמטרתה העיקרית היא לבצע סנכרון של קבצים ותיקיות ברקע כך שכל פעולה שנעשית בתיקיית הסנכרון מנוטרת בזמן אמת. כל פעולה כזו גוררת העלאה ישירה של השינוי לשרת. המשתמש אינו צריך ליזום סנכרון באופן ידני ואינו צריך להטריד עצמו בהעלאת קבצים לשרת מכיוון שהכל מתבצע באופן אוטומטי.

הגדרת ה"ענן" כיום אינה ברורה כלל. "ענן" הינו כינוי נפוץ לשרת אשר מאפשר סנכרון בין המחשב הביתי לשרת כלשהו. כיום השימוש במילה ענן נעשה בהקשר לכל מיני שרתים הפועלים בצורה שונה לחלוטין משרתים אחרים ששמם הוא גם כן "ענן".

בפרויקט זה משמעות המילה "ענן" הינו שרת המסנכרן קבצים ממספר בלתי מוגבל של משתמשים ולאו דווקא מאפשר שיתוף של הקבצים הנ"ל.

המערכת מאפשרת לכל משתמש להעלות אל השרת את כל הקבצים שאותם הוא מעוניין לגבות על גבי הענן. לאחר ביצוע הסנכרון, קבצי המשתמש יימצאו תמיד על גבי השרת. כאשר המשתמש יבחר להתחבר אל השרת ממחשב אחר, הוא יוכל לקבל את כל הקבצים אשר העלה לשרת בעבר. כמוכן הוא יוכל לעדכן אותם, לצפות בהם ואף למחוק אותם. לאחר מכן, כאשר המשתמש יבחר להתחבר לשרת הסנכרון פעם נוספת, כל הקבצים יהיו מעודכנים בהתאם להתחברותו הראשונית וכך הלאה בכל התחברות נוספת.

תחום הענן הינו חדש יחסית וכיום קיימות מספר לא מובטל של חברות העוסקות בו. בשוק זה שולטים מספר מוצרים בולטים: dropbox, Google drive, mega, and icloud . מוצרים אלו זכו להצלחה רבה ומשמשים רבבות אנשים ברחבי העולם כולו. הצלחת התחום עוררה את סקרנותי בקשר לכל הקשור למערכות הענן ועל כן החלטתי לנצל את פרויקט הגמר בתוכנית גבהים על מנת לחקור את התחום ואף להקים מערכת ענן פרטית משלי. כמו כן כתיבת תוכנת הלקוח ותוכנת השרת אפשרו לי להתעמק בנושא זה, ולפתח בעצמי אלגוריתמים ופרוטוקולים להתמודדות עם בעיות שונות. כך אני מצליח להבין את הפתרונות הקיימים והבעיות שאותם הם פותרים באופן עמוק יותר.

בנוסף, הצורך במחשוב ענן בתקופה זו הינו עצום ומהותי. כיום למשתמש הממוצע שבבעלותו נמצאים מספר מחשבים שונים בהם הוא משתמש בשגרת היום יום זקוק לגיבויים רבים מספור. המשתמש הממוצע זקוק למערכת שעל ידיה יוכל לסנכרן קבצים בקלות ובמהירות ללא צורך בזיכרון פיזי כדוגמת דיסק און-קי או CD. תחום הענן מציע פתרון נוח לסנכרון בין מחשבים שונים ובכך מהוה פתרון אידאלי לסוגיה.

# מבוא ורקע כללי

מחשוב בענן (באנגלית: cloud computing) הוא שמם של שירותי מחשב שונים הניתנים לשימוש באמצעות שרתים מרוחקים, אליהם מתחבר הלקוח דרך רשת האינטרנט או לחלופין באמצעות קו תקשורתי ייעודי. הענן נוח במיוחד מכיוון שכל שעל הלקוח לעשות על מנת להתחבר אליו, הוא להתקין תוכנת לקוח של הענן הספציפי. משלב זה יכול הלקוח להתחיל לסנכרן את קבציו במהירת ובקלות. בחלק מתוכנות הענן גם נחסך הצורך בתוכנת לקוח ייעודית וניתן פשוט לגלוש אל הענן דרך הדפדפן.

קיימים שלושה סוגים של ענן:

* שרת פרטי: שרת שאינו מאפשר שיתוף של קבצים בין משתמשים שונים ברחבי העולם. רק למשתמש אחד קיימת גישה לקבצים שהוא סינרכן לשרת.
* שרת ציבורי: כל קובץ שמועלה לשרת מסוג זה, חשוף בפני כולם ונתון לשימושם של כל המשתמשים.
* שרת היברידי: שרת המאפשר שיתוף של חלק מן הקבצים, וכמוכן מאפשר שמירה פרטית של קבצים ומונע את חשיפתם בפני הציבור הרחב.

בפרויקט זה נבנה שרת שבבסיסו הוא שרת פרטי אך ארכיטקטורת תשתית הענן, נבנתה בצורה כזו שבמקרה ויהיה צורך להמיר את השרת לשרת היברידי, הפעולה תהיה פשוטה יחסית.

# תיחום הפרויקט

* עיקר הפרויקט עוסק ברשתות מחשבים. התקשורת בין הישויות השונות נעשית באמצעות: Socket.internetsockets ,בהקשר שלנו, הsocket- מהווה נקודת קצה עבור תקשורת בין תהליכים על גבי רשת מחשבים. הפרוטוקול שבו עוברים רוב הנתונים בפרויקט הוא פרוטוקול שהומצא על ידי. פרוטוקול זה מבוסס על TCP, שחלק מתפקידיו הם ריבוב (העברת מידע לשרתים שונים על אותה הישות ברשת) והבטחה שהמידע לא יאבד ויגיע ליעדו במלואו.
* הפרויקט עוסק גם במערכות הפעלה. אחסון המידע בשרת וכמו כן הסנכרון עצמו מתאפשר תודות למערכת ניהול הקבצים הכפופה למערכת ההפעלה.
* באופן טבעי, קיים עיסוק רב בפרויקט ב-THREADS בעיקר בצד השרת וכל טיפול בלקוח נעשה על ידי THREAD נפרד.
* נעשה שימוש ב GUI בסיסי, ועיקר השימוש החזותי נעשה באמצעות סייר הקבצים של ווינדוס.
* חלק נרחב בפרויקט עוסק בניסיון להעביר קבצים ותיקיות ברשת בצורה הנוחה והמהירה ביותר.
* הפרויקט כולו מתבסס על תכנות מונחה עצמים. ומחולק בצורה המנסה למנוע תלויות בין המערכות השונות בפרויקט.

הפרויקט אינו עוסק בהצפנת הקבצים עצמם אינו עוסק בשיתופם על אף שהוא נבנה בצורה כזו שהתשתית שלו מאפשרת הרחבה עתידית שכזו באופן יחסית פשוט.

# תיאור המוצר המוגמר

## שם המוצר

Cloud system

## מטרת המוצר

המשתמש במכשיר הביתי שלו, יוכל ליהנות ממערכת פשוטה שאין צורך להתערב בה מכיוון שכל הסנכרון בה נעשה "מאחורי הפרגוד". כל פעולה שנעשית בתיקיית הסנכרון של המשתמש מנוטרת בזמן אמת. כמוכן, פעולה כזו גוררת העלאה ישירה של השינוי לשרת. המשתמש אינו צריך ליזום סנכרון ואינו צריך להטריד עצמו בטיפול בהעלאה לשרת מכיוון שהכל מתבצע באופן אוטומטי. השרת שומר תמונת ראי של תיקיית הסנכרון. על מנת לאפשר העברה מהירה של הקבצים, המידע יכווץ בעת התעבורה ברשת. בצד הלקוח יופיע אמצעי חזותי ייעודי, אך עיקר השימוש החזותי יתבסס על סייר ווינדוס הקיים. השרת מסוגל לתת שירות למספר בלתי מוגבל של לקוחות ומשתמשים שונים בו זמנית.

## מטרות מרכזיות

נוחות המשתמש על ידי מניעת צורך של התערבות המשתמש בסנכרון, והגבלת הלקוח באופן מינימליסטי.

## שאלות שהמערכת עונה עליהן

* כיצד אפשר לנתר התעדכנות של קבצים וספריות על מנת לסנכרן אותם אל השרת?
* כיצד לכווץ תקשורת באופן יעיל?
* כיצד למנוע התנגשויות שיכולות להיווצר כתוצאה משיתוף קבצים?
* כיצד לשמור על קבצי סיסמאות מוגנים?

## השוואת העבודה עם פתרונות ויישומים קיימים

בשוק הכלכלי כיום קיימים יישומים רבים המיישמים מערכות ענן. המפורסמים שבהם הם: icloud,mega, dropbox, and Google drive.

הפרויקט נבנה בצורה הדומה יותר לדרופבוקס מאשר לשאר התוכנות. זאת מכיוון שגם דרופבוקס נכתב בשפת פייתון ונראה כי הארכיטקטורה שלו מתאימה יותר לפייתון. בנוסף דרופבקוס גם הוא דורש שימשו בתוכנת לקוח ייעודית.

# מבט אישי על העבודה ועל פיתוחה

## אתגרים

* ניהול הזמן.
* מציאת האלגוריתם המתאים ביותר לניתור שינויים בקבצים בצד הלקוח.
* מניעת התנגשויות בין קבצים של משתמשים.
* ארכיטקטורת שמירת הקבצים על גבי השרת.
* בניית השרת כתומך בהרחבה לשיתוף משתמשים אחרים בתיקיות.
* טיפול בקבצים הוא קשה מאתגר ביותר שדורש דיבאגינג ארוך ומדוקדק.

## התמודדות עם קשיים

במהלך הפרויקט נתקלתי בקשיים לא מעטים. להלן הקשיים המרכזיים:

* כיצד ניתן לאפשר סנכרון של קבצים רבים בצורה שתחסוך בתעבורת רשת ותהיה יעילה ומהירה.
* כיצד יש לשמור את הרשימות החשובות שבהן משתמש השרת מבלי סיכון שיינזקו בעת נפילת השרת.
* באיזו צורה אשמור את קבצי המשתמשים הרבים בשרת? כיצד למנוע את התנגשות שמות הקבצים השונים והתיקיות השונות?
* כיצד לשלוח תיקיה שלמה בהודעה אחת במקום לשלוח כל קובץ לחוד?

כל הקשיים האלה נפתרו במהלך הפרוייקט, כפי שיובהר בהמשך.

## הערכת הפתרון לעומת התכנון והמלצות לשיפורו

המבנה הכללי של התוצר הסופי דומה לתכנון הראשוני. על כן, נראה כי התכנון נעשה בצורה שקולה ואחראית. מאידך, התכנון לא היה מספיק עמוק ודברים רבים נוספו "על הדרך". מודולים שונים תוכננו תוך כדי כתיבתם, עובדה זו הביאה לכך שנאלצתי להשקיע הרבה מזמני בדיבאג במקום להתקדם הלאה בפיתוח המערכת. כמוכן בתכנון לא נצפו מספר מגבלות שהתווספו במהלך הפיתוח.

יש אינספור שיפורים שהשרת מסוגל להיעזר בהם, זאת מכיוון ששרתים מסוג "ענן" הם בעלי מאפיינים רבים ואפשרויות פיתוח שונות. לעניות דעתי, אפשרויות הפיתוח המרכזיות הן:

* שיתוף משתמשים בתיקיות שונות.
* GUI המאפשר למשתמש לראות את המידע הרלוונטי אליו בשרת בזמן אמת.
* אפשור אופציה של סנכרון אל השרת במקביל לסנכרון מהשרת.

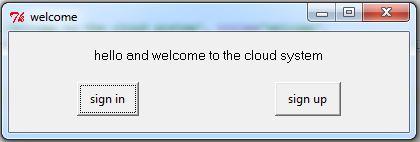
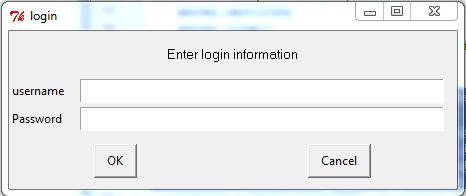
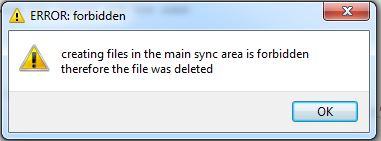
# תיאור הממשק למשתמש

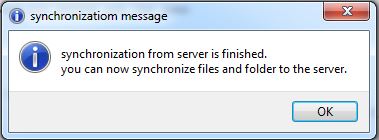
## מידע בסיסי

1. בתוכנת הענן הלקוחות (המשתמשים) יסנכרנו את הקבצים והתיקיות שלהם אל השרת הייעודי.
2. על מנת לסנכרן כל לקוח חייב לבצע כניסה לחשבון צ'אט שנמצא בשרת. הכניסה תתבצע על ידי שימוש בשם משתמש וסיסמא. משתמש שאין לו חשבון חייב להירשם בפעם ראשונה.
3. השרת מסונכרן עם משתמשים רבים במקביל ובנפרד לחלוטין.
4. השרת תומך בהתנתקות פתאומית של הלקוח.
5. קליטת המסרים מן המשתמש על ידי תוכנית הלקוח (client) מתבצעת בחלון המיועד לכך.
6. המשתמש יגיב להודעות הקופצות בעזרת הקלדה או בעזרת לחיצה על כפתורים שיופיעו על גבי ההודעות הקופצות
7. ללקוח אסור לבצע פעולת DRAG אל תוך תיקיית הסנכרון (אין מגבלה על פעולת העתק הדבק).
8. ללקוח אסור לבצע שינויים בתיקיית הסנכרון כל עוד תוכנת הלקוח לא רצה ברקע.
9. ללקוח אסור לבצע שינויים בתיקיית הסנכרון כל עוד לא קפצה הודעה שהשרת סיים לסנכרן אל המשתמש.
10. ללקוח מותר להניח תיקיות בלבד מתחת לתיקיית synchronization folder (כמובן שבתיקיות שמתחת לתיקייה זו מותר להניח קבצים כרגיל) ובמקרה שינסה לבצע פעולה כזו הקובץ ימחק ותופיע הודעה בהתאם.
11. התווים: / \ @ > < ? \* : !" '

אינם מורשים להימצא בסיסמא או בשם המשתמש.

## דוגמאות להודעות חזותיות בצד הלקוח

* דוגמא להודעה קופצת בעלת כפתורים מרובים:
* דוגמא להודעה קופצת בעלת שדה קבלת קלט מהמקלדת:
* דוגמא להודעת שגיאה:
* דוגמא להודעה רגילה בעלת כפתור OK בלבד:



* תיקיית הסנכרון שממנה יסונכרנו קבצי המשתמש לשרת:



## הסנכרון

לאחר תהליך הרישום מתבצע הסנכרון. לאחר שהלקוח קיבל הודעה כי השרת סיים לסנכרן אל הלקוח את כל הקבצים והתיקיות, המשתמש יכול לצפות בקבציו בתיקייה synchronization folder הנמצאת בתיקייה שבה רץ קובץ הלקוח. כמוכן הלקוח יכול להניח שם קבצים חדשים והם יסונכרנו אל השרת.

## סיום התוכנית

התוכנית מסתיימת כאשר המשתמש סוגר את התוכנית או כאשר הוא לוחץ על X בהודעות החזותיות לפני ביצוע כניסה לחשבון המשתמש.

## אמצעי הקלט בצד הלקוח

הלחצן השמאלי של העכבר – בחלונות הקופצים העכבר ישמש ללחיצה על כפתורים ( פעמים רבות מקש האנטר יכול להחליף את לחיצת העכבר)

כל שאר המקשים יתפקדו כמו בכל תוכנה רגילה.

# סביבת העבודה

## סביבת פיתוח

שפת הפיתוח היא python2.7 (שפה עילית).

סביבת הפיתוח הינה pycharm community edition 3.4.1.

התוכנה מספקת סביבת עבודה משולבת (IDE), הכוללת:

* עורך מסמכים.
* דיבאגר.
* טרמינל מובנה.
* מנהל מסדי נתונים.

בעזרת התוכנה ניתן לכתוב בקלות קטעי קוד שלמים ב Python. התוכנה מוצאת שגיאות ובעיות, ועוזרת לתקן אותן בקלות. כמו כן ניתן להריץ בקלות את התוכנית מתוך התוכנה ללא שימוש ב cmd.

## סביבת הרצה

סביבת ההרצה היא חלון ריצה של התוכנה ב-1pycharm community edition 3.4. שמריצה את התוכנית על גבי השפה python2.7.

היתרון של python הוא שניתן לכתוב בעזרתה קטעי קוד ארוכים במהירות, וניתן אחר כך לקרוא ולהבין אותם בקלות. בנוסף ניתן להריץ בסביבת ההרצה קבצי py באופן מהיר.

# תיאור המערכת

## סקירת כל המודולים המרכיבים את המוצר וקשרי הגומלין ביניהם

### **מבט על**

המערכת מורכבת מ-2 קומפוננטות עקריות: קומפוננטת שרת וקומפוננטת לקוח.



Cloud\_Server component הינה קומפוננטה המייצגת את התוכנה שרצה בשרת. המיקום הפיזי של השרת אינו משנה, ובכל מקרה התקשורת אליו מתבצעת לפי התרשים למעלה.

המבנה הפנימי של השרת בנוי בהתאם לתרשים מטה:



#### Cloud\_Client component

Cloud\_Client component הינה קומפוננטה המייצגת את התוכנה שרצה בלקוח:



### פירוט רכיבי המערכת

**המבנה הסטטי של התוכנה:**



**הערה**: בסעיפים הבאים יפורטו הפקג'ים/מודולים שכוללים מחלקות.

פקג'ים שאינם כוללים מחלקות כבר פורטו עם הפונקציות שלהם בתרשים הקודם.

#### Folder\_manager



#### :Folder\_monitor



#### User\_system



#### cloud\_server



#### InitSyncSequence

תרשים זה מפרט את המימוש של ה-use case הרשמה למערכת ואת המימוש של ה-use case כניסה לחשבון המשתמש שצוינו במסמך האפיון.



#### Synchronize from Server to Client

תרשים זה מפרט את ההמשך של מימוש של ה-use case הרשמה למערכת ואת המימוש של ה-use case כניסה לחשבון המשתמש. בעצם המשך ה-sequence הקודם.



#### Watch for changes sequence

תרשים זה הינו המשך ה-sequence הקודם.

תרשים זה מפרט את המימוש של ה-use case מחיקה, יצירה, שינוי שם ועדכון.



#### יציאה מהמערכת

היות והמערכת תומכת בכל התנתקות פתאומית מן השרת, לא קיימת דרישה מן הלקוח לבצע יציאה מסודרת לפי use case יציאה מחשבון משתמש.

## פירוט ותיאור המודולים

### מודולים מובנים

#### os

המודול משמש בעיקר לעיסוק בתיקיות ובקבצים כגון: יצירה, מחיקה, שינוי שם וכו'.

#### ast

המודול הופך משתני מחרוזות לסוג משתנה אחר עפ"י הנתונים במחרוזת.

#### shutil

משמש בעיקר לעיסוק בקבצים.

#### Disutils

משמש להעתקת עץ של תיקיות ממיקום אחד לאחר.

#### Errno

משמש לתפיסת exceptions.

#### Thread

משמש ליצירת threads. בפרויקט קיים שימוש ב threads כאשר יש צורך להעלות הודעה חזותית כלשהי על המסך ובמקביל יש עניין שהתוכנית תמשיך לרוץ.

#### Zipfile

משמש לעיסוק בקבצי zip. בפרויקט קיים שימוש במודול זה כאשר הלקוח מעוניין להעלות קובץ אל השרת, על מנת לאפשר תעבורת רשת מהירה יותר הקובץ נדחס. כאשר הקובץ מגיע אל השרת, הוא מורחב מחדש.

#### Zlib

המודול משמש לעיסוק בקבצים מכווצים. בפרויקט קיים שימוש במודול זה כאשר השרת שולח אל הלקוח תיקייה כלשהי. המודול אינו מסוגל לכווץ תיקייה ריקה, אך כאשר התיקייה מכילה כל דבר שהוא המודול מכווץ את כל תוכן התיקייה לקובץ אחד. השרת מכווץ את התיקייה ושומר אותה ב temp zip files server (תיקייה זמנית) ואז שולח את הקובץ הדחוס ולאחר מכן התיקייה נמחקת. כאשר הלקוח מקבל את הקובץ, הוא נשמר ב temp zip files client (תיקייה זמנית) לאחר מכן הקובץ מורחב בחזרה, ואז מועבר אל תיקיית הסנכרון, ולבסוף התיקייה הזמנית נמחקת.

#### Uuid

משמש לחישוב GUID. GUID הוא מחרוזת ארוכה שלעולם לא תחזור על עצמה. תיקיות הסנכרון השונות של כל משתמש נשמרות בשרת בשם ה GUID שלהם ולא בשמם האמתי. ל GUID שני יתרונות בולטים:

* אבטחה: קשה מאוד ואף בלתי אפשרי לנחש שם של GUID כך שאין לדעת כיצד נקראות התיקיות בשרת.
* מניעת התנגשויות: מכיוון ש GUID הוא שם ייחודי שלא חזור על עצמו פעמיים. אין סיכוי שהתיקיות השונות שמאוחסנות בשרת יתנגשו אחת עם השנייה.

#### Hashlib

משמש לחישוב hash. הסיסמאות של כל המשתמשים נשמרות רק ב HASH שחושב להם. לאחר שחושב ה-HASH השרת לא שומר את הסיסמה המקורית של המשתמש. כאשר המשתמש ירצה לבצע כניסה למערכת ה HASH של סיסמתו ייבדק ולא הסיסמה עצמה. היתרון הברור כאן הוא אבטחה, השרת אינו "יודע" מהי הסיסמה של כל משתמש ולכן אין אפשרות לגנוב אותה ממנו.

#### Socket

בצד הלקוח המודול משמש לבניית Socket לתקשורת. בצד השרת המודול משמש לתפיסת שגיאות מסוג Socket.

#### SocketServer

בצד השרת, SocketServer משמש ליצירת שרת שפותח thread נפרד לכל התקשרות עם לקוח. השיטה מונעת את קריסת השרת וכמו כן היא נוחה לשימוש.

### מודולים שאינם מובנים ושלא נכתבו על ידי

#### wx

מספק GUI על מנת שהתוכנה תהיה נוחה ונגישה ללקוח. Wx מתקדם יותר מ easygui ולעומתו אינו יוצר בעיות ב threads.

#### easy\_gui

מודול המספק GUI נוח ופשוט לשימוש.

#### Folder\_monitor

מודול שמנטר כל שינוי הנעשה בתיקייה מסוימת (גם בתת התיקיות שלה.) למודול מגבלה והוא אינו מסוגל לנתר פעולת drag.

### מודולים שאינם מובנים ונכתבו על ידי

#### Utils

מודול עזר ובו פונקציות רבות המשמשות את השרת וגם את הלקוח.

פונקציות מרכזיות ב Utils:

* בניית ההודעות שעוברות בין הלקוח לשרת עפ"י פרוטוקול הרשת.
* שינוי paths על פי פרמטרים מסוימים.
* הפיכת תיקייה לקובץ zip.

#### Folder\_manager

מודול המטפל בכל מערכת התיקיות בשרת.

#### User\_system

מודול המטפל ברישום המשתמשים בשרת.

# תיעוד הקוד

## תיעוד הפונקציות

### פונקציות עיקריות במודול Cloud\_client

|  |  |
| --- | --- |
| פעולות מחוץ למחלקה | |
| שם הפונקציה | **תיאור** |
| show\_message\_dlg | הפונקציה מציגה GUI בפני המשתמש. |
| get\_version | הפונקציה שולחת הודעת check\_version לשרת ומקבלת תגובת פרוטוקול על הבקשה. |
| sign\_up | הפונקציה קוראת ל enter\_pass\_and\_username. לאחר מכן, הפונקציה רושמת את המשתמש במערכת המשתמשים של השרת. |
| sign\_in | הפונקציה קוראת ל enter\_pass\_and\_username. לאחר מכן, הפונקציה מבצעת כניסה לחשבון משתמש הנמצא אל השרת. |
| enter\_pass\_and\_username | הפונקציה מציגה GUI לקבלת שם משתמש וסיסמא. |
| get\_main\_sync\_folders | הפונקציה מבקשת מהשרת לקבל את רשימת כל התיקיות השייכות לחשבון המשתמש שאליו בוצעה כניסה. |
| sync\_from\_server | הפונקציה מבקשת מהשרת לקבל תיקיית סנכרון ראשית מסוימת והשרת מחזיר אותה. פונקציה זאת נקראת לכל תיקיית סנכרון ראשית ששייכת למשתמש. |
| watch\_for\_changes | הפונקציה מפעילה מוניטור המנתר כל שינוי בתיקיית הסנכרון. על כל שינוי שיזוהה על ידי המערכת הפונקציה תקרא לאחת מן הפונקציות הבאות בהתאם לשינוי על מנת ליידע את השרת על השינויים שהתבצעו: sync\_to\_server\_upload\_file, sync\_to\_server\_create, sync\_to\_server\_rename, sync\_to\_server\_delete. |
| sync\_to\_server\_upload\_file | הפונקציה מודיעה לשרת כי תיקייה או קובץ כלשהו עודכנו. לפני השליחה לשרת, הקובץ מכווץ. |
| sync\_to\_server\_create | הפונקציה מודיעה לשרת כי תיקייה או קובץ חדש נוצר\ה. |
| sync\_to\_server\_rename | הפונקציה מודיעה לשרת על כך ששונה שמו\ה של קובץ\תיקייה. |
| sync\_to\_server\_delete | הפונקציה מודיעה לשרת על כך שתיקייה\קובץ נמחק\ה. |

### פונקציות ומחלקות עיקריות במודול cloud\_server:

|  |  |
| --- | --- |
| המחלקה RequestHandler | |
| שם הפונקציה | **תיאור** |
| RequestHandler | המחלקה מממשת את פעולת ה SocketServer ומנהלת את יצירת הקשר עם הלקוח .המחלקה משתמשת ב threads ולכן כל לקוח מנוהל על ידי thread משלו. |
| Handle | תקשורת השרת עם הלקוח במהלך הסנכרון מתנהלת בפונקציה זו. |
| פעולות מחוץ למחלקה | |
| handle\_message | הפונקציה מקבלת את ההודעה שהתקבלה מאת הלקוח, ומנתבת את ההודעה לפונקציה התואמת לטיפול בה. |
| get\_version | הפונקציה מחזירה הודעת פרוטוקול ללקוח על בקשת get\_version. |

### פונקציות ומחלקות עיקריות במודול Folder\_monitor

|  |  |
| --- | --- |
| המחלקה Watcher | |
| שם הפונקציה | **תיאור** |
| \_\_init\_\_ | הפונקציה הבונה של המחלקה Watcher המנתרת שינויים הנעשים בתיקייה נבחרת. |
| run | הפונקציה מפעילה את ניתור השינויים בתיקייה הנבחרת. |
| פעולות מחוץ למחלקה | |
| Watch\_path | הפונקציה מנטרת על כל השינויים הנעשים בתיקייה הנבחרת ומכניסה כל שינוי לתוך תור במחלקה Watcher. |

### פונקציות עיקריות במודול Utils

|  |  |
| --- | --- |
| פעולות מחוץ למחלקה | |
| שם הפונקציה | **תיאור** |
| zip\_folder | מכווץ תיקייה לכדי קובץ zip. |
| contains\_any | בודק האם הפרמטר שהתקבל מכיל את אחד מהתווים שהוגדרו כאסורים בפרמטר השני. אם מכיל: מחזיר True אחרת: False. |
| unzip\_folder | מרחיב קובץ zip לתיקייה. |
| build\_request | בונה הודעת בקשת לקוח בהתאם לפרוטוקול ולפרמטרים שהתקבלו. מחזיר מחרוזת שמהווה את הבקשה שתישלח לשרת. |
| build\_response | בונה הודעת תגובה ללקוח בהתאם לפרוטוקול ולפרמטרים שהתקבלו. מחזיר מחרוזת שמהווה את התגובה שתישלח ללקוח. |
| decode\_message | מקבלת הודעת פרוטוקול ומפתח. מחזירה את הערך של המפתח. |
| get\_params\_dict | מקבלת הודעת פרוטוקול. מחזירה את צמדי המפתח והערך במילון |
| get\_multiple\_params | מקבלת מילון של מפתחות וערכים, וגם רשימה של מפתחות. מחזירה את הערכים של המפתחות ברשימה על פי אותו הסדר של רשימת המפתחות. |

### פונקציות ומחלקות עיקריות במודול User\_system

|  |  |
| --- | --- |
| המחלקה User | |
| שם הפונקציה | **תיאור** |
| \_\_init\_\_ | הפעולה בונה את המשתמש הבסיסי בעל תכונות של סיסמא, שם משתמש, האם מחובר וכתובת המשתמש. |
| המחלקה User\_system | |
| \_\_init\_\_ | הפעולה בונה את User\_system שמטפל בכל הקשור לניהול חשבונות משתמשים. תכונות המחלקה: רשימת המשתמשים הרשומים, קובץ שבו רשימת המשתמשים. כאשר הפונקציה נבנית, היא טוענת מתוך הקובץ את כל המשתמשים הקיימים אל תוך הרשימה הווירטואלית. |
| Sign\_up | רושמת משתמש חדש במערכת. |
| Sign\_in | מבצעת כניסה לחשבון משתמש. |
| Log\_out | מבצעת יציאה מחשבון משתמש. |

### פונקציות ומחלקות עיקריות במודול Folder\_manager

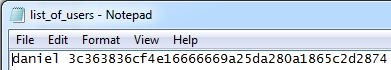
|  |  |
| --- | --- |
| המחלקה Folder | |
| \_\_init\_\_ | הפעולה בונה את התיקייה הבסיסית שיש לכל משתמש (לכל משתמש מספר תיקיות). תיקייה היא בעלת התכונות: שם אצל הלקוח, שם ה GUID של התיקייה, בעל התיקייה, רשימת האנשים המשותפים עם התיקייה. כל תיקייה נרשמת בקובץ users\_folders ביחד עם תכונותיה. |
| change\_name | משנה את השם שהלקוח נתן לתיקייה. |
| המחלקה Folder\_manager | |
| \_\_init\_\_ | הפעולה בונה את ה Folder\_manager שדואג לטפל בכל הקשור לניהול קבצים. תכונת המחלקה: רשימה של תיקיות מסוג Folder. אם קיימות כבר תיקיות בקובץ ה Folders הן נטענות לתוך הרשימה. |
| Create\_folder\_or\_file | הפעולה יוצר\ת תיקייה או קובץ בזיכרון השרת על מנת להסתנכרן עם הלקוח. קוראת ל \_create\_main\_sync\_folder במקרה הצורך. לבסוף ללקוח נשלחת הודעת אישור. |
| \_create\_main\_sync\_folder | הפונקציה יוצרת תיקיית main\_sync ובנוסף מעדכנת את הקובץ Folders בהתאם. |
| upload file | הפונקציה מקבלת קובץ ומאחסנת אותו במקום המיועד לכך על פי הפרמטרים שהתקבלו. הפרמטרים מכילים את שם בעל התיקייה ואת שם התיקייה אצל הלקוח. |
| delete\_folder\_or\_file | הפונקציה מקבלת path שעליה למחוק , על מנת להסתנכרן עם הלקוח. הפונקציה מוחקת את הקובץ\תיקייה ומחזירה הודעת אישור ללקוח. הפונקציה קוראת ל \_delete\_main\_sync\_folder במקרה הצורך. הפונקציה מחזירה ללקוח אישור שהקובץ\תיקייה נמחק\ה |
| \_delete\_main\_sync\_folder | הפונקציה מוחקת תיקיית main\_sync ועל כן יש גם ליצור את התיקייה וגם לעדכן את קובץ ה Folders. |
| rename\_folder\_or\_file | הפונקציה מקבלת path שעליה לשנות את שמו, על מנת להסתנכרן עם הלקוח. הפונקציה משנה את הקובץ\תיקייה ומחזירה הודעת אישור ללקוח. הפונקציה קוראת ל\_rename\_main\_sync\_folder במקרה הצורך. הפונקציה מחזירה ללקוח אישור ששם הקובץ\תיקייה שונה\תה |
| \_rename\_main\_sync\_folder | הפונקציה משנה את שם הלקוח של תיקיית main\_sync ועל כן מתריאה יש גם ליצור את התיקייה וגם לעדכן את קובץ ה Folders. |
| download\_file | הפונקציה מקבלת path של קובץ שאותו הלקוח מעוניין להוריד. הפונקציה מחזירה את הקובץ בהודעה התואמת את הפרוטוקול (לפני השליחה הקובץ נדחס). |
| download\_folder | הפונקציה מקבלת path של תיקייה שאותה הלקוח מעוניין להוריד. הפונקציה מחזירה את התיקייה ואת כל התכולה שלה בהודעה התואמת את הפרוטוקול (לפני השליחה התיקייה נדחסת). |
| get\_folders\_names\_by\_owner | הפונקציה מקבלת שם משתמש ומחזירה ללקוח בהודעה עפ"י הפרוטוקול את כל התיקיות השייכות לו במילון כאשר המפתח הוא שם התיקייה אצל הלקוח והערך הוא השם אצל השרת. |

### פירוט מאגרי המידע של השרת

השרת משתמש בשני מאגרי מידע:

* users\_list
* folders

בעת הרצת השרת במקרה שמאגרי מידע אלו לא קיימים, השרת יצור אותם בעצמו. לאחר מכן, השרת טוען את המאגרים הללו לתוך משתנים. כל שינוי שיתבצע על המשתנה יתבצע על מאגר המידע המתאים לו. השרת זקוק למאגרים אלו כי אם הוא ישמור אותם בתור משתנים הם יימחקו בכל הפעלה מחדש.

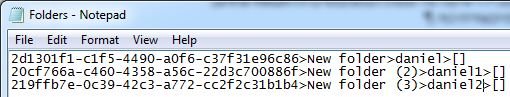
users\_list הוא קובץ שמכיל רשימה של שמות המשתמשים וה-HASH שחושב לסיסמאותיהם, בצמדים ובצורה כזו:

על מנת להבין את קובץ folders יש צורך בהסבר קצר על מבנה שמירת הקבצים בשרת:

לכל לקוח קיימת "תיקיית סנכרון ראשית" (ראה: תיאור הממשק למשתמש; תחת לתיקייה זו מותר לשים רק תיקיות(. זאת מכיוון שהמערכת נבנתה כאופטימלית לאופציית הרחבה לשיתוף קבצים בין משתמשים. לאחר הרחבה זו כל תיקייה שנמצאת מתחת לתיקיית הסנכרון, תהיה בעלת הגדרות ומאפיינים אחרים שעל פיהם יקבע עם אילו משתמשים תשותף כל תיקייה. מתחת לתיקיות הללו ניתן לאחסן כל קובץ\תיקייה כלשהם והם ישותפו עם משתמשים אחרים עפ"י הגדרות התיקייה שנמצאת מתחת לתיקיית הסנכרון.

כל תיקיית סנכרון ראשית לא נשמרת אצל השרת בשם שהלקוח נתן לה, זאת מכיוון שאז ייווצרו התנגשויות בין תיקיות של משתמשים שונים. על כן לכל תיקייה מחושב GUID ומכיוון ש GUID אינו יכול לחזור על עצמו פעמיים אין סיכוי שתיקרה התנגשות בין תיקיות שונות. אך על מנת להחזיר את התיקייה אל הלקוח יש צורך לשמור את שם התיקייה שניתן על ידי הלקוח.

לאחר הבנת עקרונות אלו ניתן להבין את מבנה קובץfolders . הקובץ שומר את: שם המשתמש שלו שייכת התיקייה, רשימה של שמות המשתמשים המשותפים עם התיקייה (כרגע הרשימה תמיד ריקה כי אין אפשרות של שיתוף קבצים בין משתמשים),שם ה GUID של התיקייה כפי שנשמרה בשרת ולבסוף את שם התיקייה האמתי. במקום רווח בין כל אחד מן המשתנים קיים הסימן "<" זאת מכיוון ששימוש ברווח יוצר באגים ומורכבויות מיותרות.

בתמונה הבאה ניתן לראות כיצד נשמרים הנתונים לגבי כל תיקיית סנכרון ראשית:

# האלגוריתמים המרכזיים בפרויקט

## סנכרון

### תיאור וניסוח הבעיה האלגוריתמית

השאלה היא כיצד ניתן לסנכרן מהלקוח לשרת כל שינוי שנעשה בתיקיות הסנכרון של הלקוח. למשל: כאשר הלקוח מעדכן קובץ כלשהו כיצד ניתן לדעת שבצד השרת הקבוץ פחות עדכני מאשר בצד הלקוח?

### תיאור אלגוריתמים קיימים לפתרון הבעיה

קיימות שלושה אפשרויות לפתרון הבעיה:

* להשוות בין תאריכי השינוי של התיקיות בשרת ובלקוח.

החיסרון הוא שפתרון זה דורש המון תעבורת רשת מכל משתמש מכיוון שהוא נאלץ לשלוח כל הזמן לשרת הודעת בדיקה כדי "לדעת" אם השרת סנכרן את הקובץ העדכני ביותר.

* כאשר עולה קובץ לשרת, לחשב לו HASH ולשמור את ה-HASH בשרת. אחרי כן, לבקש מהשרת כל זמן קבוע את ה-HASH של כל הקבצים ולחשב האם יש HASH שהשתנה. החיסרון הוא שגם כאן קיימת תעבורת רשת גבוהה.
* מוניטור שמעדכן בזמן אמת על כל שינוי בתיקייה נבחרת. היתרון הבולט של המוניטור הוא שהוא אינו מעמיס על תעבורת רשת, ושהוא מודיע על שינוי רק כאשר הוא באמת נוצר, ולא כל זמן קבוע ללא קשר האם נוצר שינוי שכזה. בנוסף בזכות המוניטור הלקוח אינו צריך כלל להתערב בתהליך הסנכרון, מכיוון שכל הסנכרון יתבצע ברקע. החיסרון הוא שאם בצד הלקוח התוכנה איננה רצה והלקוח משנה אצלו את קבציו. אז המוניטור לא מסוגל לנטר על השינוי מכיוון שהוא רץ כחלק מהלקוח. נוסף על כך, המוניטור אינו מסוגל לזהות פעולת drag כראוי.

### הפתרון הנבחר

הפתרון הנבחר הוא המוניטור ,ואכן הוא מגביל במעט את מערכת הענן.

## סנכרון בדגש סנכרון תיקיות

### תיאור וניסוח הבעיה האלגוריתמית

קיימת בעיה כאשר יש צורך בשליחת תיקייה שלמה על כל התוכן שלה (אין זה משנה אם השליחה מתבצעת אל צד השרת או אל צד הלקוח). אין שום אפשרות לשלוח תיקייה שלמה ביחד עם כל התוכן שלה דרך הרשת.

### תיאור אלגוריתמים קיימים לפתרון הבעיה

קיימים שני אלגוריתמים לפתרון הבעיה:

* קיימת אפשרות לשליחת כל הקבצים אחד אחרי השני בעזרת פונקציה רקורסיבית או בעזרת לולאה. החיסרון הוא ששיטה זו היא מאוד מסורבלת וקשה ליישום.
* אפשרות נוספת היא למפות את עץ התיקיות בעזרת פונקציה כדוגמת map ולאחר מכן לטעון כל קובץ ותיקייה שנמצא\ת בעץ התיקיות ולשלוח כל אחד מהם\ן באופן פרטני. החיסרון הוא שגם שיטה זו מסורבלת ובנוסף עד לשליחת כל הקבצים יכול להיווצר מצב שבו עץ הקבצים עצמו כבר השתנה.

### הפתרון הנבחר

לאחר עיון רב בבעיה נראה כי הפתרון הפשוט ביותר הוא לכווץ את התיקייה לכדי קובץ zip . כך גם חוסכים בתעבורת רשת וגם ניתן לשלוח את התיקייה בשלמותה ללא כל בעיה. נראה כי הפתרון מושלם לצרכיה של המערכת.

## שליחת קבצים ומשתנים על גבי הרשת

### תיאור וניסוח הבעיה האלגוריתמית

בפרוטוקול שכתבתי אין אפשרות לשלוח משתנים אלא strings בלבד נשלחים. כיצד אפשר לשלוח משתנים וקבצים על גבי הפרוטוקול?

### הפתרון הנבחר

הפתרון פשוט למדי בעת שליחת המשתנה התוכנית משתמשת בפונקציית str על מנת להפוך את המשתנה למחרוזת. לאחר שהמשתנה נשלח יש להשתמש במודול ast הנ"ל (ראה פירוט ותיאור המודלים) על מנת לשלוח את הקבצים ליעדם.