Project Overview & Architecture Summary

**Project Overview — Secure File Metadata Logger (Microservices)**

המערכת בנויה משני **Microservices עצמאיים** העובדים יחד:

1. **Watcher Service** עוקב בזמן אמת אחרי תיקייה מקומית, מזהה קבצים חדשים, מחלץ מהם נתוני מטא־דאטה, ויוצר בקשת HTTP מאובטחת ל־Logger.
2. **Logger Service** מקבל את בקשת ה־POST עם המטא־דאטה, מאמת את ה־ JWT, מבצע ולידציה למידע, וכותב את התוצאה לקובץ לוג בפורמט טקסט

מטרה מרכזית

המערכת מדגימה תקשורת בין שני שירותים נפרדים בעזרת:

* אימות באמצעות **JWT Token**
* העברת מידע בפורמט **JSON**
* כתיבת לוגים בצורה מאורגנת ומאובטחת

כל התהליך מתרחש באופן אוטומטי   
ברגע שקובץ חדש נוצר בתיקייה הנצפית, הוא נשלח ללוג באופן שקוף וללא התערבות המשתמש.

**Architecture**

כל שירות עצמאי, עם סביבת ריצה, תצורה וה־ Responsibility שלו:

* **WatcherService** אחראי על איתור קבצים ותקשורת החוצה.
* **LoggerService** אחראי על קבלת המידע וכתיבת לוגים.

התקשורת ביניהם מתבצעת באמצעות  **HTTP REST API** עם אימות JWT

**עקרונות מרכזיים**

* **RESTful Communication:**  
  התקשורת בין השירותים מתבצעת באמצעות HTTP (POST /logs)   
  עם מבנה בקשה אחיד, תגובות בקוד סטטוס תקני, ופורמט JSON להעברת מידע.  
  כל שירות מנותק מהשני וניתן לפריסה והרצה עצמאית, בהתאם לעקרון Microservices
* **Dependency Injection**   
  כל מחלקה מקבלת את התלויות שלה דרך ה־Constructor   
  מה שמאפשר ניהול חיים אחיד, בדיקות יחידה, והחלפת מימושים בקלות
* **Separation of Concerns:**  
  חלוקה ברורה לשכבות לוגיות:
  + **Services** לוגיקת עסקים ותהליכים.
  + **Models / DTOs** ייצוגי מידע פנימיים וחיצוניים.
  + **Controllers** נקודות גישה (Endpoints).
* **Error**   
  כל שגיאה מטופלת בשירות שבו היא מתרחשת )למשל שגיאת תקשורת ב־ Watcher אינה מפילה את ה־Logger.)
* **Configuration Management:**  
  שימוש בקובץ קונפיגורציה ובמשתני סביבה (ENV)

**Internal Architecture per Service**

**WatcherService**

* FileSystemWatcher מאזין לשינויים בתיקייה (watched/).
* JwtTokenGenerator מייצר טוקן JWT תקף ל־5 דקות עם חתימה סימטרית (HS256).
* ApiManageService שולט על בקשות ה־HTTP ל־Logger.
* BaseService מטפל בשליחת בקשות וקליטת תגובות באופן גנרי.
* ConfigService קורא נתוני קונפיגורציה secret) נתיב watched, וכו')
* FileMover מעביר קבצים לתיקיית processed/ לאחר הצלחה.
* MappingConfig (AutoMapper) ממפה בין FileMetadata ל־ FileMetadataDto.

**LoggerService**

* JwtMiddleware בודק את תקפות החתימה של ה־ JWT ומוודא תוקף וזמן.
* LoggerController Endpoint ראשי (POST /logs).
* LogFileCreator יוצר רשומת לוג ומוסיף אותה לקובץ metadata.log.
* APIResponse מחלקת תגובה אחידה לכל קריאה (קוד סטטוס, תוצאה, הודעת שגיאה).

**Technologies**

.NET 8 (C#) - פיתוח שני השירותים (API + Background Service)

Dependency Injection -Built-in .NET DI – הזרקת תלויות בין שירותים ומחלקות.

JWT Authentication - System.IdentityModel.Tokens.Jwt יצירה ואימות טוקנים.

HTTP Client Factory – IhttpClientFactory קריאות API מאובטחות ויעילות.

File Monitoring – FileSystemWatcher האזנה לקבצים חדשים בתיקייה.

AutoMapper- מיפוי אובייקטים בין DTO ל־Model.

Logging – Serilog כתיבת קובצי לוג

**WatcherService**

WatchFolderService

**תפקיד:**  
השירות המרכזי שמאזין לתיקיית watched ומתחיל את כל התהליך.  
נכתב כיורש של BackgroundService כדי לרוץ ברקע באופן מתמשך

השירות המרכזי במערכת, שאחראי על ניטור התיקייה watched/ והפעלת תהליך העיבוד,  
נכתב כיורש של BackgroundService כדי לאפשר ריצה רציפה ברקע לאורך כל חיי האפליקציה.

בחרתי לממש את המחלקה WatchFolderService כ ־BackgroundService משום שזו התשתית הייעודית ב־.NET להפעלת תהליכים ארוכי־טווח ברקע החל מרגע עליית הפרויקט ועד לסגירתו.  
השירות מופעל אוטומטית בעת אתחול המערכת ומריץ את המתודה ExecuteAsync שבתוכה מופעל הרכיב FileSystemWatcher

ה־ FileSystemWatcher אחראי להאזין בזמן אמת לשינויים במערכת הקבצים, כגון יצירת קובץ חדש, שינוי תוכן או מחיקה בתיקייה המוגדרת.  
שילוב שני הרכיבים BackgroundService ו- FileSystemWatcher מאפשר האזנה מתמשכת ורציפה, שמבטיחה תגובה מיידית לכל אירוע בתיקייה ותחילת תהליך העיבוד בצורה יציבה ויעילה.

**תהליכים עיקריים:**

* משתמש ב־ FileSystemWatcher כדי לזהות קבצים חדשים.
* כאשר קובץ מתווסף קורא את הנתונים (שם, גודל, תאריך יצירה).
* מחשב ערך  **SHA-256 Hash** לקובץ כדי לזהות ייחודיות.
* יוצר  **JWT Token** בעזרת JwtTokenGenerator.
* שולח את המטא־דאטה בעזרת ApiManageService
* לאחר תשובה מוצלחת (200 OK) מעביר את הקובץ לתיקיית processed

**JwtTokenGenerator**

**תפקיד:**  
אחראי על יצירת  **JWT Token** סימטרי (HS256) עם זמן תוקף קצר (5 דקות).

**מה היא עושה:**

* בונה Claims (iss, exp)
* חותם עם מפתח סודי (SecretKey) שמגיע מקובץ הקונפיגורציה (או מ ENV)
* מחזיר מחרוזת JWT חתומה.

**ApiManageService**

**תפקיד:**  
מבצע את השליחה של המטא־דאטה ל־LoggerService דרך קריאת HTTP   
משתמש ב־ IHttpClientFactory כדי ליצור HttpClient בצורה מבוקרת.

**תהליך:**

* מוסיף Header מסוג Authorization: Bearer <JWT>
* ממיר את האובייקט ל־JSON
* שולח את הבקשה ל־POST /logs/. לקונטרולר
* מחזיר אובייקט APIResponse עם הסטטוס והתוצאה.

**BaseService**

**תפקיד:**  
שכבת עזר גנרית לשירותי HTTP   
מגדירה פונקציה אחת עיקרית לשליחה אסינכרונית של בקשות.

**מה היא עושה:**

* בונה אובייקט HttpRequestMessage בהתאם לסוג (POST/GET/PUT/DELETE).

(אצלינו רק POST)

* ממירה את הנתונים ל־ JSON .
* מחזירה את תגובת השרת בצורה אחידה (APIResponse)

**ConfigService**

תפקיד:

מטפל בקריאה ועדכון של קובץ ההגדרות (Config file).

משם נשלפים נתיבים, URL, סוד ה־ JWT, וכו'.

**FileMover**

**תפקיד:**  
אחראי להזיז את הקובץ לאחר העיבוד לתיקיית processed/

**מה היא עושה:**

* בודקת אם הקובץ קיים.
* מעתיקה או מזיזה לתיקיית processed/.
* מוחקת קבצים ישנים אם הושג מספר קבצים מקסימלי.

**הוספה שלי:**

בתיקיה processed אני בודק אם יש עד 5 קבצים.

במידה וקובץ ניכנס הוא ימחוק את הקובץ הכי ישן שלא נעשה בו שימוש.

המיון מתבצע על ידי החתימת השם הוספתי תאריך בכל העברה של הקובץ

ככה בהמשך אם נרצה לשנות או לישלוט על תוכן התיקיה יש לנו אפשרות.

**LoggerService**

**LoggerController**

**תפקיד:**  
נקודת הכניסה לשירות — מקבלת את בקשת ה־POST מה־WatcherService

מה היא עושה:

* מקבלת את הבקשה עם Authorization Header (JWT).
* בודקת שהמטא־דאטה תקין.
* קוראת לשירות LogFileCreator כדי לרשום את הנתונים.
* מחזירה תגובה 200 OK אם הכול תקין**.**

**LogFileCreator**

**תפקיד:**  
יוצר קובץ לוג טקסט ומוסיף רשומות חדשות עם פרטי הקובץ.

מה היא עושה:

פותחת את הקובץ.

כותבת טקסט עם המאטה דאטה שנשלח.

יוצרת את התיקייה אם אינה קיימת.

**איך מריצים את התוכנית**

לפני שמריצים את התוכנית יש לוודא שהמפתח הסודי של ה־ JWT מוגדר.  
המערכת בודקת תחילה האם קיים משתנה סביבה בשם **JWT\_SECRET**  
אם הוא קיים הוא ישמש כמפתח לחתימה ולאימות.  
אם לא, המערכת תשתמש במפתח שמוגדר בקובץ ההגדרות החיצוני (WatcherSettings.json).

לאחר מכן מריצים את שני השירותים יחד:  
**WatcherService** ו־**LoggerService**

בתוך התיקייה של WatcherService קיימת תיקייה בשם **watched**  
כדי לבדוק שהמערכת עובדת, גוררים לשם כל קובץ שתרצו.  
ברגע שהקובץ נוסף, השירות מזהה אותו אוטומטית ומתחיל את העיבוד.  
אם הכול תקין, הקובץ יועבר לתיקייה **processed**.

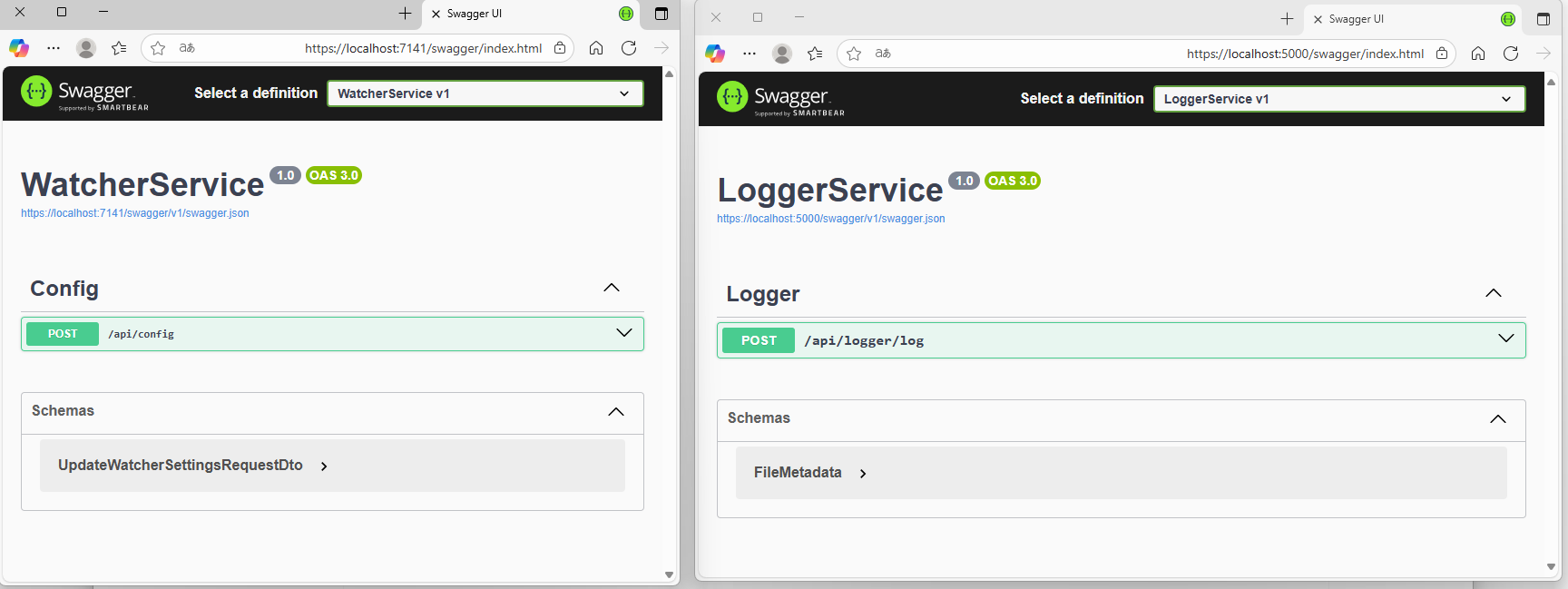
לאחר מכן ניתן לראות את תוצאת הפעולה בשירות השני **LoggerService**.  
בתוך התיקייה LoggerService/logs תיווצר תיקייה חדשה לפי תאריך היום, ובתוכה יופיע קובץ לוג חדש המכיל את כל המידע (metadata) על הקובץ שעובד.

כדי לשנות קונפיגורציה מה (UI) ניתן לפתוח בדפדפן את ה־**Swagger**:

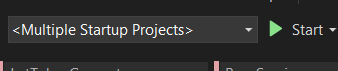
שם ניתן לשנות את הקונפיגורציה לפי הנוחיות או בקובץ עצמו.

**דוגמא להרצה**

בעת עליית הפרויקט אנחנו ניראה שני SWAGGER אחד לכל קונטרולר

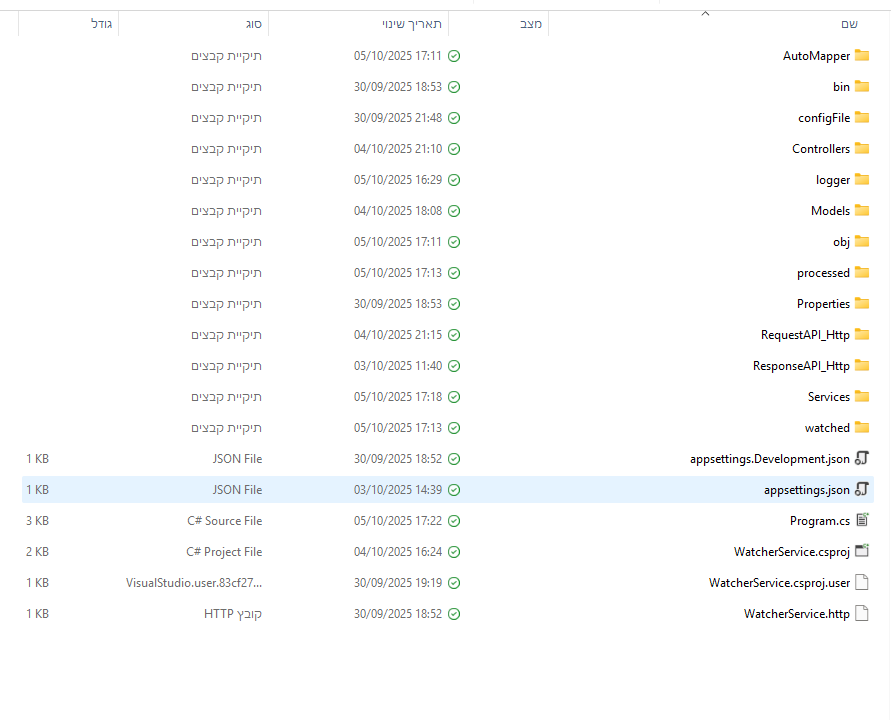


אם לא מופיע שים לב שאתה במצב Multiple StartUp



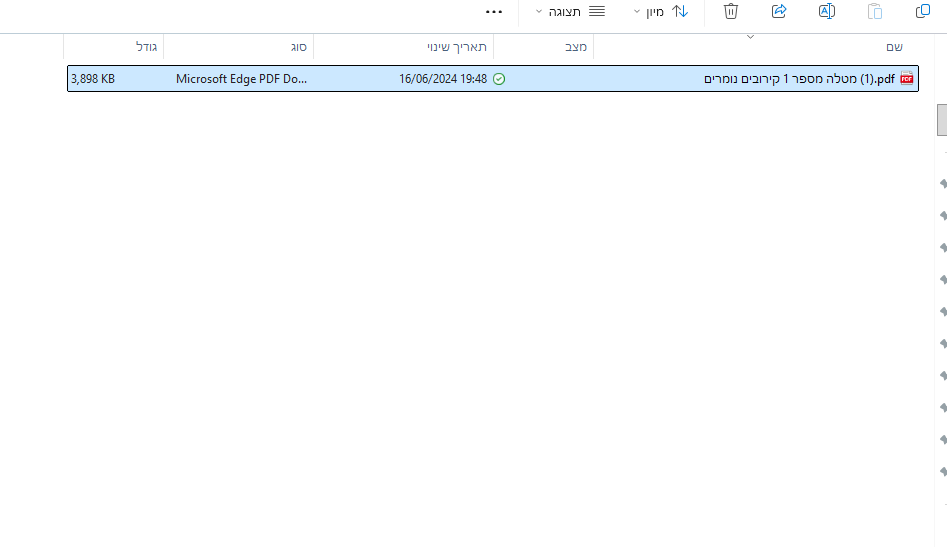
ניכנס לתיקיה:

WatcherService\WatcherService

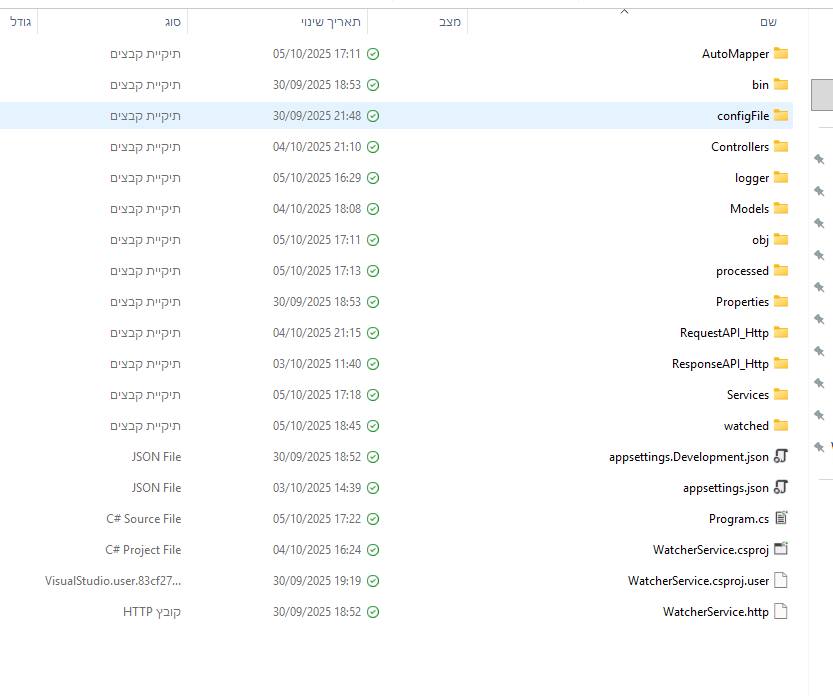


לאחר שניכנסנו לתיקיה ניגרור לתוכה קובץ.

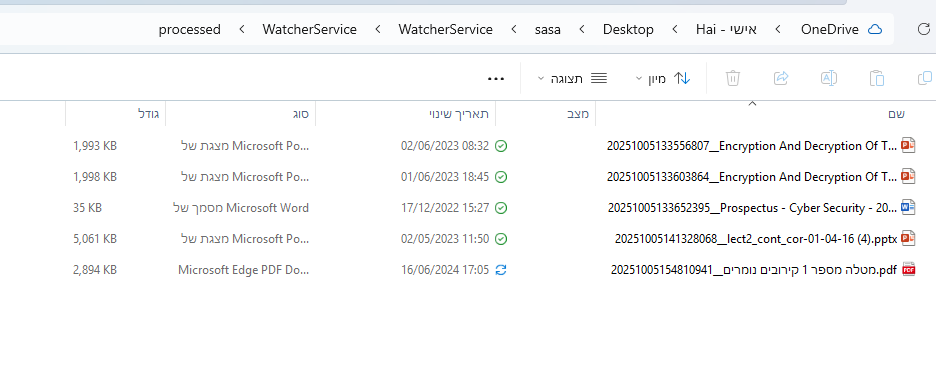
PDF, TXT, PPTX …



ברגע שגררנו לתיקיה קובץ המאזין יתחיל לעבוד

אם הכל תקין הקובץ יעבור לתיקיה אחרת

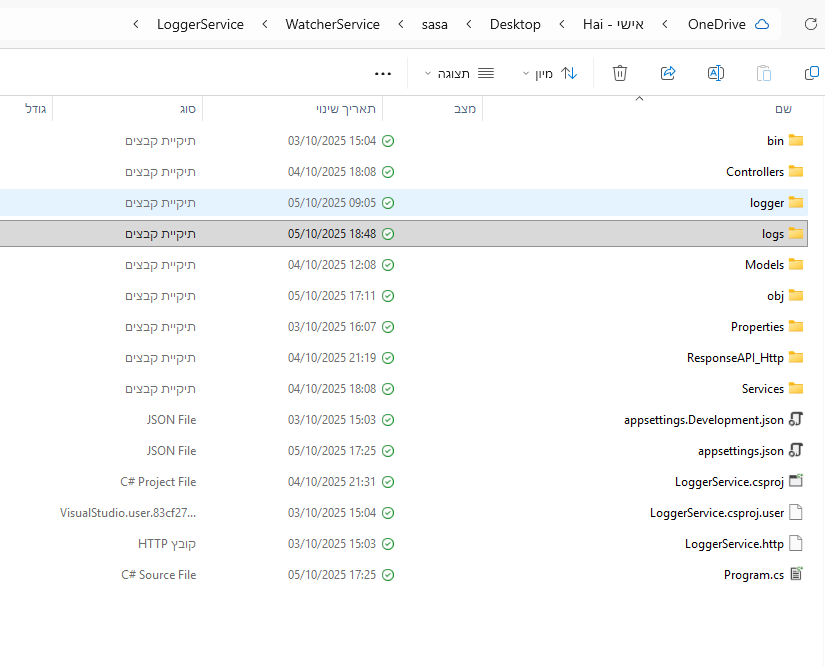
ניכנס לתיקיה ושם תוכל לראות את המעבר של הקובץ

עכשיו ניבדוק בסרוויס השני אליו שלחנו את המאטה דאטה לראות שהכל תקין

תמונה שמכילה טקסט, צילום מסך, תוכנה, סמל מחשב

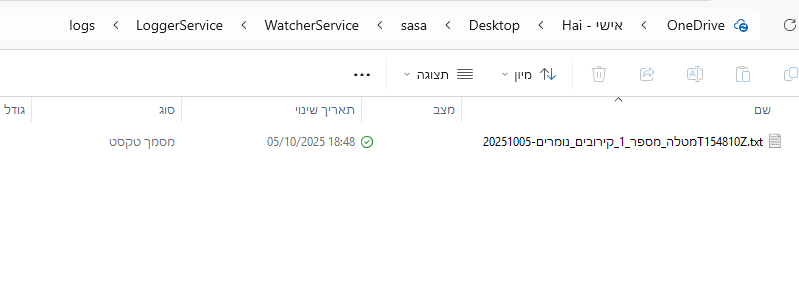
תוכן בינה מלאכותית גנרטיבית עשוי להיות שגוי.ניכנס לתיקיית LoggerService

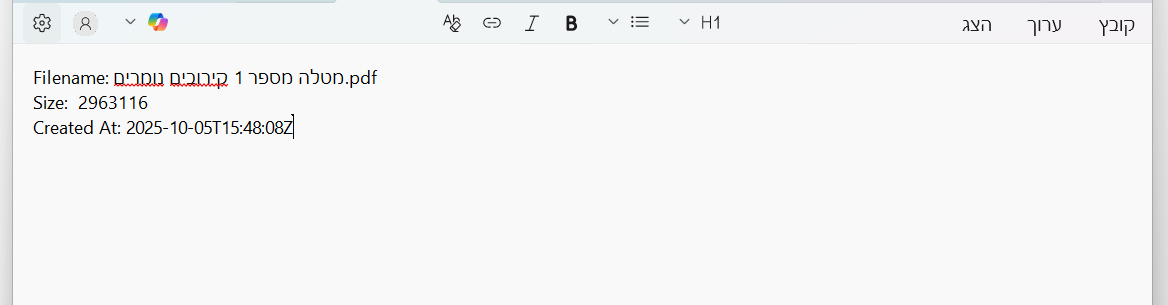
כאשר נכנסנו לתיקיה נראה את תיקיית logs ניכנס אליה



...

בתוך התיקיה נוכל לראות קבצי .txt

לפי הדרישה .

נפתח את הקובץ

אנו רואים את כל המאטה דאטה ששלחנו מסרוויס **WatcherService** מהקובץ המקורי בעזרת פרוטוקול תקשורת HTTP