## Project Overview & Architecture Summary

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

The system is built from two independent microservices that work together:

1. **Watcher Service** – Monitors a local folder in real time, detects new files, extracts metadata, and sends a secure HTTP request with this data to the Logger Service.
2. **Logger Service** – Receives the POST request containing the metadata, validates the JWT token, verifies the information, and writes the result into a log file in text format.

### Main Goal

The project demonstrates secure communication between two independent services using:

- **JWT Authentication**
- **JSON-based data transfer**
- **Structured and secure log writing**

The entire process runs automatically:
Once a new file appears in the watched folder, it is detected, processed, and logged — without any manual user interaction.

### Architecture

Each service is fully independent, with its own runtime, configuration, and responsibility.

- **WatcherService** – Responsible for file monitoring and sending metadata.
- **LoggerService** – Responsible for receiving data and writing logs.

Communication between them is handled through an **HTTP REST API** secured with **JWT authentication**.

### Core Architectural Principles

**1. RESTful Communication**
The services communicate via HTTP (POST /logs) using consistent request structures, standard status codes, and JSON payloads.
Each service can run independently, following microservices best practices.

**2. Dependency Injection (DI)**
All dependencies are injected through constructors.
This approach improves testability, ensures consistent lifetime management, and allows for easy replacement of components.

**3. Separation of Concerns**
Clear separation between logical layers:

- **Services** – Contain business logic and main workflows.
- **Models / DTOs** – Represent internal and external data structures.
- **Controllers** – Define API endpoints.

**4. Error Handling**
Errors are handled locally within each service.
For example, a communication error in the Watcher does not crash the Logger service.

**5. Configuration Management**
Uses configuration files and environment variables (ENV) for flexibility and security.

## Internal Architecture per Service

**WatcherService**

- **FileSystemWatcher** – Listens for changes inside the watched/ directory.
- **JwtTokenGenerator** – Creates a JWT token (HS256) valid for 5 minutes.
- **ApiManageService** – Handles HTTP requests to the Logger service.
- **BaseService** – Provides a generic implementation for sending and receiving HTTP requests.
- **ConfigService** – Reads configuration values (paths, URLs, secret keys, etc.).
- **FileMover** – Moves processed files into the processed/ directory after success.
- **MappingConfig (AutoMapper)** – Maps internal models to DTOs.

**LoggerService**

- **JwtMiddleware** – Verifies the JWT token signature and expiration.
- **LoggerController** – Main API endpoint (POST /logs).
- **LogFileCreator** – Creates and appends new log entries to the log file.
- **APIResponse** – Unified response model for all requests (status code, data, and error messages).

## Technologies Used

- **.NET 8 (C#)** – Core framework for both services (API + BackgroundService).
- **Dependency Injection** – Built-in .NET DI for structured dependency management.
- **JWT Authentication** – Implemented with System.IdentityModel.Tokens.Jwt.
- **HTTP Client Factory** – IHttpClientFactory for efficient and secure HTTP requests.
- **File Monitoring** – FileSystemWatcher for real-time folder monitoring.
- **AutoMapper** – Object mapping between models and DTOs.
- **Logging** – Serilog for structured logging.

## WatcherService

### WatchFolderService

**Role:**
This is the main background service that monitors the watched/ folder and initiates the entire process.
It inherits from **BackgroundService** to run continuously in the background for the entire lifetime of the application.

I chose to implement WatchFolderService as a **BackgroundService** because it provides a dedicated .NET infrastructure for long-running background tasks
starting automatically when the application runs and stopping gracefully when it shuts down.

Inside the service, the **ExecuteAsync()** method runs the **FileSystemWatcher**,
which listens in real time for file system changes such as file creation, modification, or deletion.

The combination of **BackgroundService** and **FileSystemWatcher** allows continuous monitoring and ensures an immediate response to every new file event,
starting the processing workflow reliably and efficiently.

**Main Workflow:**

- Detects new files in the watched folder.
- Reads file metadata (name, size, creation time).
- Computes a **SHA-256 hash** for file uniqueness.
- Generates a **JWT token** using JwtTokenGenerator.
- Sends the metadata to the Logger via ApiManageService.
- On successful response (HTTP 200 OK), moves the file to the processed/ folder.

**JwtTokenGenerator**

**Role:**
Responsible for creating a symmetric **JWT token (HS256)** with a short 5-minute lifetime.

**Workflow:**

- Builds claims (iss, exp).
- Signs the token with a secret key from configuration or environment variable (JWT_SECRET).
- Returns a valid, signed JWT string.

**ApiManageService**

**Role:**
Handles sending the metadata to the **LoggerService** via an HTTP POST request.
Uses IHttpClientFactory to manage HTTP clients safely and efficiently.

**Process:**

- Adds an Authorization: Bearer <JWT> header.
- Serializes the metadata into JSON.
- Sends the POST request to /logs.
- Returns a unified APIResponse object containing status and result.

**BaseService**

**Role:**
Provides a generic HTTP request layer for other services.

**What it does:**

- Builds HttpRequestMessage for the desired method (POST/GET/PUT/DELETE).
- Serializes the payload into JSON (POST only in this project).
- Returns a standardized APIResponse object with server response.

**ConfigService**

**Role:**
Manages reading and updating configuration files.
Retrieves paths, URLs, secrets, and other environment settings.

**FileMover**

**Role:**
Moves processed files to the processed/ directory.

**What it does:**

- Checks if the file exists.
- Moves it to the target folder.
- Deletes the oldest files if the folder exceeds the maximum count (5 files).

In the processed folder I check if there are up to 5 files.

If a file is entered, it will delete the oldest file that has not been used.

The sorting is done by signing the name I added a date to each transfer of the file

So later if we want to change or control the contents of the folder we have the option.

**LoggerService**

**LoggerController**

**Role:**
Main entry point of the Logger service handles POST requests from the Watcher.

**What it does:**

- Receives the request with an Authorization header containing the JWT token.
- Validates the metadata.
- Calls LogFileCreator to write the data.
- Returns a 200 OK response if successful.

**LogFileCreator**

**Role:**
Creates and manages the text log files containing file metadata.

**What it does:**

- Opens or creates the log file.
- Appends a new line with the received metadata.
- Automatically creates the directory if it doesn't exist.

## How to Run the Application

Before running, make sure the **JWT secret key** is set.
The system first checks for an environment variable named JWT_SECRET.
If not found, it will use the key defined in the WatcherSettings.json configuration file.

Then, run both services:

- **WatcherService**
- **LoggerService**

Inside the **WatcherService** folder, there is a directory named watched/.
To test the system, simply drag any file (e.g., .pdf, .txt, .pptx) into that folder.

Once added, the service detects it automatically, processes it, and moves it to the processed/ folder if successful.

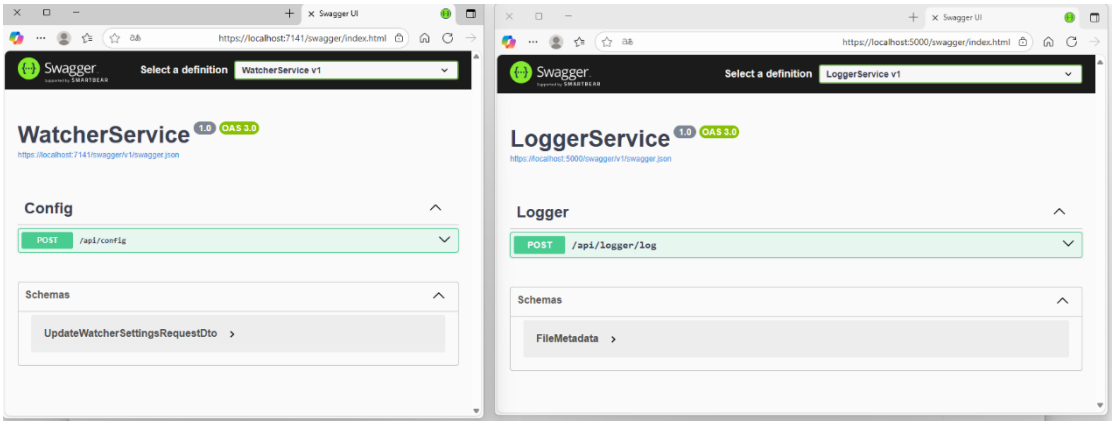
You can then check the **LoggerService/logs/** directory

a new folder will be created with the current date, containing a log file with the metadata of the processed file.

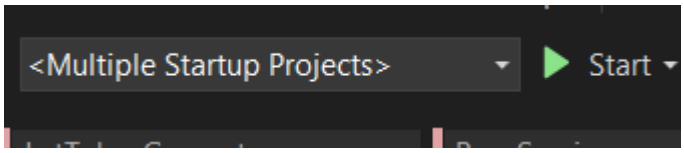
To modify configuration settings through the **UI**, open the Swagger page in your browser.
You can update parameters directly there or by editing the configuration file.

### *Example for running*

When the project loads we will see two SWAGGERs, one for each controller.

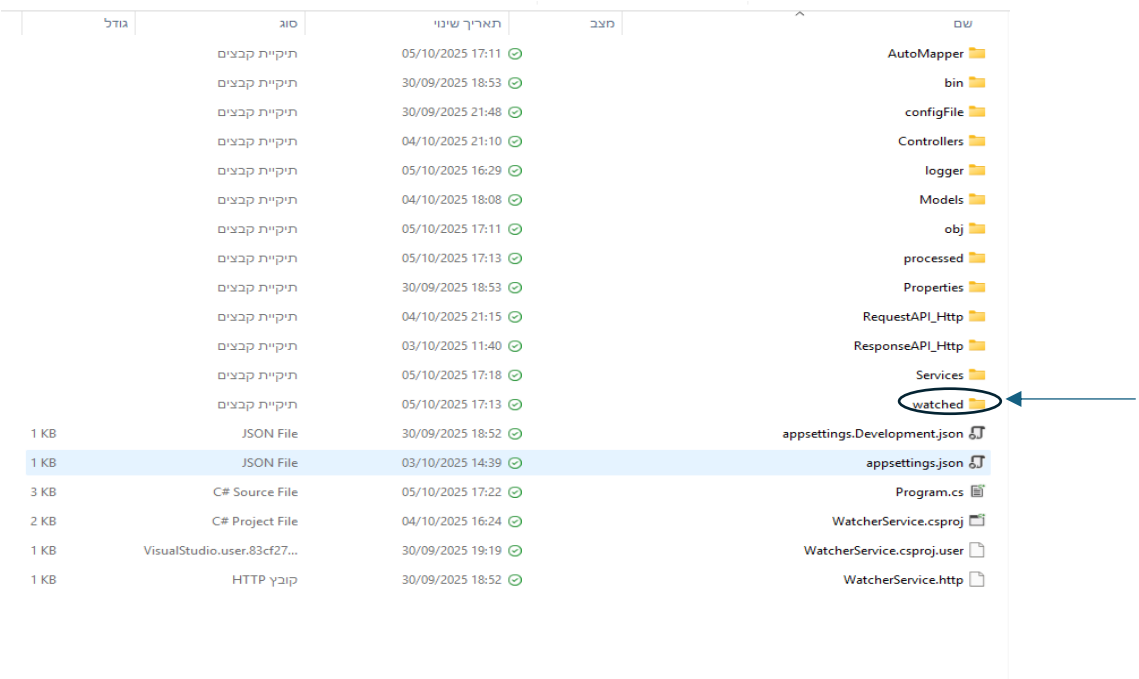


If it does not appear, note that you are in Multiple StartUp mode.



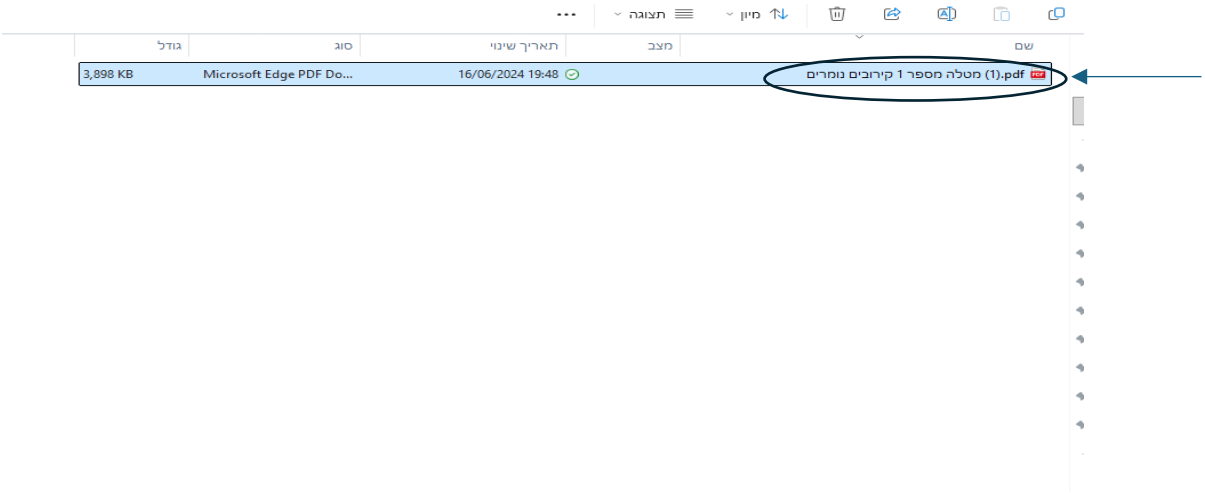

We will enter the folder:

WatcherService\WatcherService



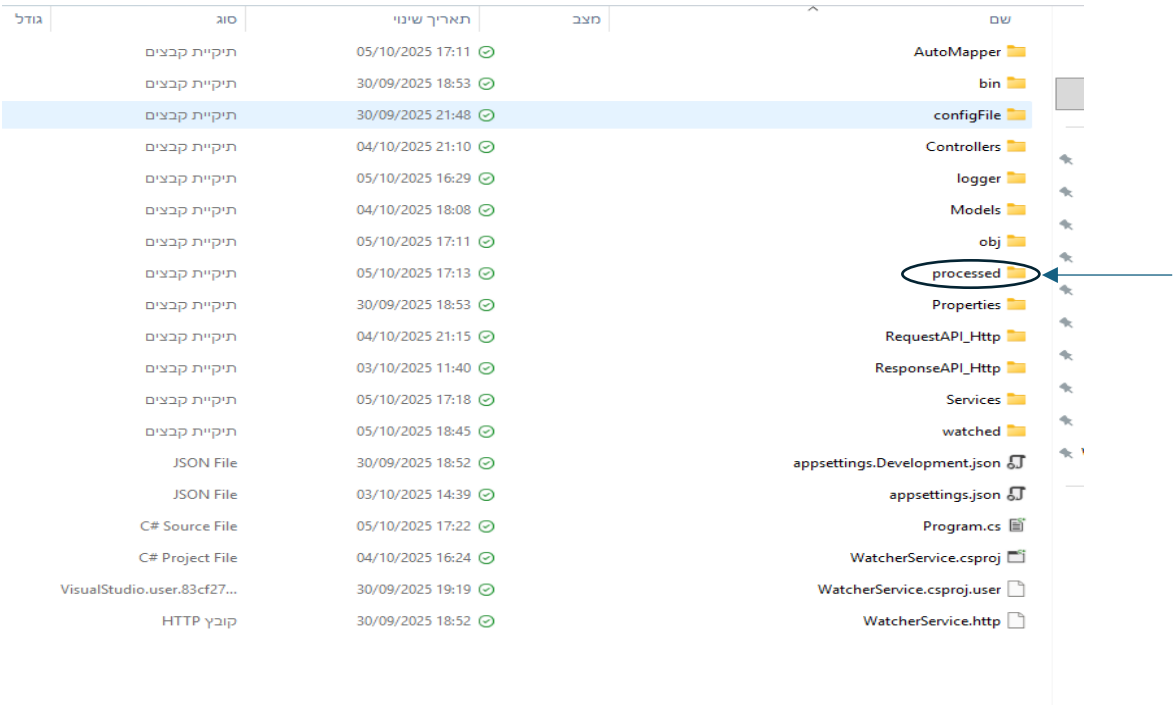
| גודל | סוג | תאריך שינוי | מצב | שם |
|---|---|---|---|---|
| | תיקיית קבצים | 05/10/2025 17:11 | | AutoMapper |
| | תיקיית קבצים | 30/09/2025 18:53 | | bin |
| | תיקיית קבצים | 30/09/2025 21:48 | | configFile |
| | תיקיית קבצים | 04/10/2025 21:10 | | Controllers |
| | תיקיית קבצים | 05/10/2025 16:29 | | logger |
| | תיקיית קבצים | 04/10/2025 18:08 | | Models |
| | תיקיית קבצים | 05/10/2025 17:11 | | obj |
| | תיקיית קבצים | 05/10/2025 17:13 | | processed |
| | תיקיית קבצים | 30/09/2025 18:53 | | Properties |
| | תיקיית קבצים | 04/10/2025 21:15 | | RequestAPI_Http |
| | תיקיית קבצים | 03/10/2025 11:40 | | ResponseAPI_Http |
| | תיקיית קבצים | 05/10/2025 17:18 | | Services |
| | תיקיית קבצים | 05/10/2025 17:13 | | watched |
| 1 KB | JSON File | 30/09/2025 18:52 | | appsettings.Development.json |
| 1 KB | JSON File | 03/10/2025 14:39 | | appsettings.json |
| 3 KB | C# Source File | 05/10/2025 17:22 | | Program.cs |
| 2 KB | C# Project File | 04/10/2025 16:24 | | WatcherService.csproj |
| 1 KB | VisualStudio.user.83cf27... | 30/09/2025 19:19 | | WatcherService.csproj.user |
| 1 KB | קובץ HTTP | 30/09/2025 18:52 | | WatcherService.http |

After entering the folder, we will drag a file into it. PDF, TXT, PPTX ...



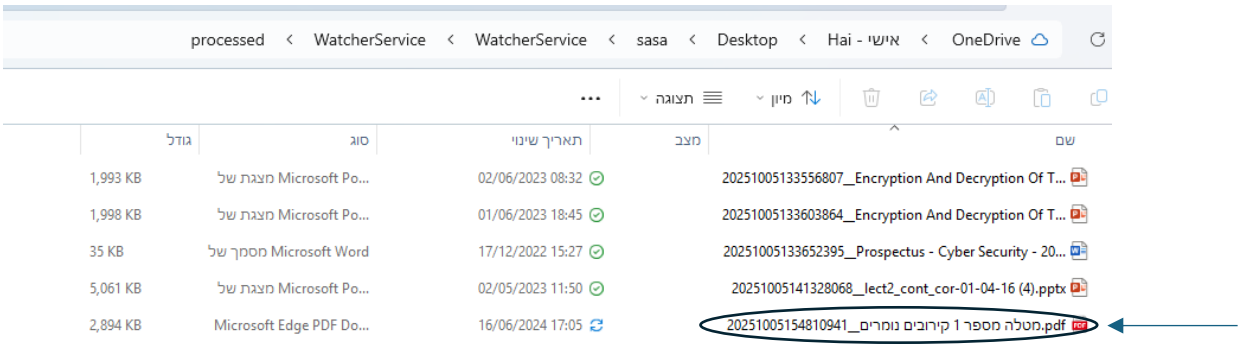
| גודל | סוג | תאריך שינוי | מצב | שם |
|---|---|---|---|---|
| 3,898 KB | Microsoft Edge PDF Do... | 16/06/2024 19:48 | | מטלה מספר 1 קירובים נומרים (1).pdf |

Once we have dragged the file to the folder, the listener will start working. If everything is correct, the file will be moved to another folder.



| גודל | סוג | תאריך שינוי | מצב | שם |
|---|---|---|---|---|
| | תיקיית קבצים | 05/10/2025 17:11 | | AutoMapper |
| | תיקיית קבצים | 30/09/2025 18:53 | | bin |
| | תיקיית קבצים | 30/09/2025 21:48 | | configFile |
| | תיקיית קבצים | 04/10/2025 21:10 | | Controllers |
| | תיקיית קבצים | 05/10/2025 16:29 | | logger |
| | תיקיית קבצים | 04/10/2025 18:08 | | Models |
| | תיקיית קבצים | 05/10/2025 17:11 | | obj |
| | תיקיית קבצים | 05/10/2025 17:13 | | processed |
| | תיקיית קבצים | 30/09/2025 18:53 | | Properties |
| | תיקיית קבצים | 04/10/2025 21:15 | | RequestAPI_Http |
| | תיקיית קבצים | 03/10/2025 11:40 | | ResponseAPI_Http |
| | תיקיית קבצים | 05/10/2025 17:18 | | Services |
| | תיקיית קבצים | 05/10/2025 18:45 | | watched |
| | JSON File | 30/09/2025 18:52 | | appsettings.Development.json |
| | JSON File | 03/10/2025 14:39 | | appsettings.json |
| | C# Source File | 05/10/2025 17:22 | | Program.cs |
| | C# Project File | 04/10/2025 16:24 | | WatcherService.csproj |
| | VisualStudio.user.83cf27... | 30/09/2025 19:19 | | WatcherService.csproj.user |
| | HTTP קובץ | 30/09/2025 18:52 | | WatcherService.http |

We will enter the folder and there you will be able to see the file's path.



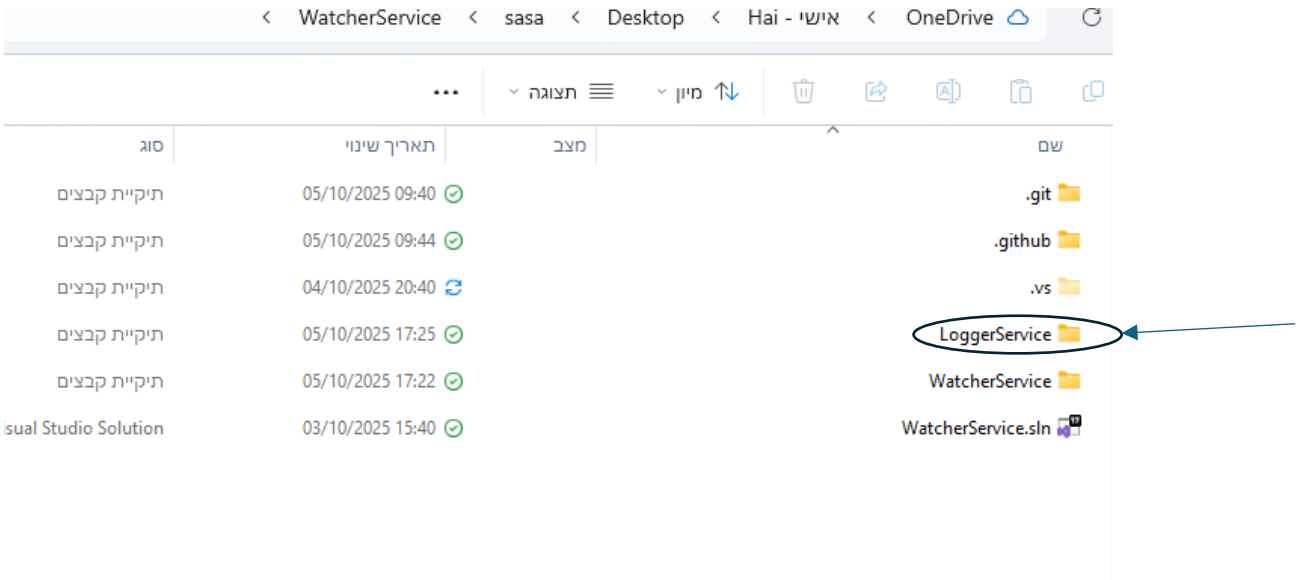
processed < WatcherService < WatcherService < sasa < Desktop < Hai - אישי < OneDrive

תצוגה מיון

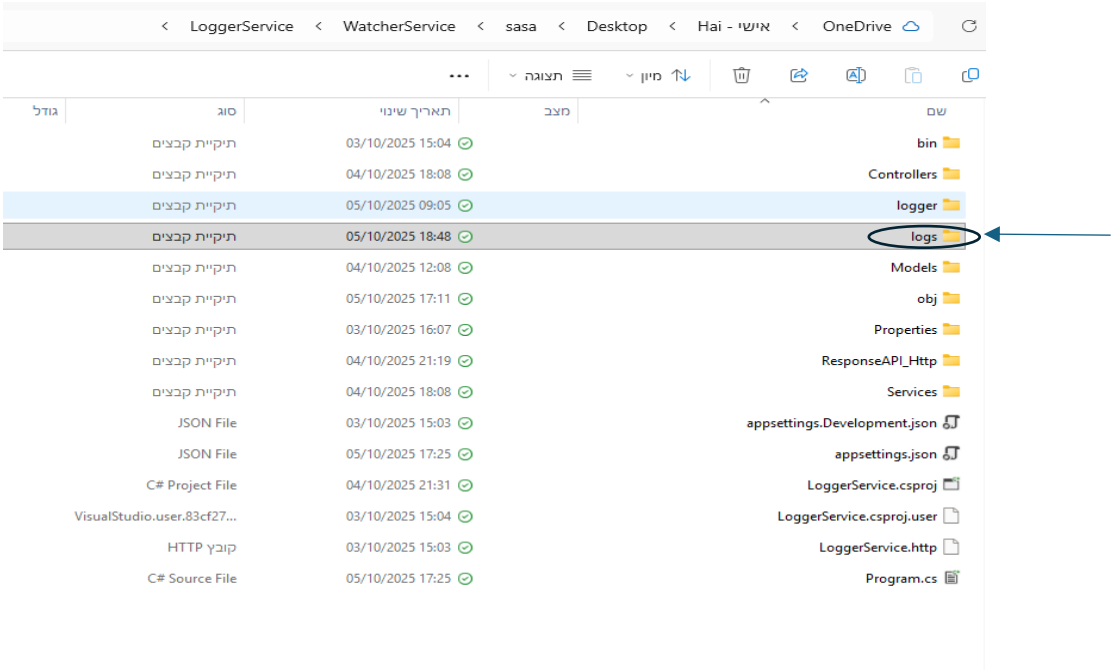
| גודל | סוג | תאריך שינוי | מצב | שם |
|---|---|---|---|---|
| 1,993 KB | מצגת של Microsoft Po... | 02/06/2023 08:32 | | 20251005133556807__Encryption And Decryption Of T... |
| 1,998 KB | מצגת של Microsoft Po... | 01/06/2023 18:45 | | 20251005133603864__Encryption And Decryption Of T... |
| 35 KB | מסמך של Microsoft Word | 17/12/2022 15:27 | | 20251005133652395__Prospectus - Cyber Security - 20... |
| 5,061 KB | מצגת של Microsoft Po... | 02/05/2023 11:50 | | 20251005141328068__lect2_cont_cor-01-04-16 (4).pptx |
| 2,894 KB | Microsoft Edge PDF Do... | 16/06/2024 17:05 | | 20251005154810941__מטלה מספר 1 קירובים נומרים.pdf |

Now we will check the second service to which we sent the metadata to see if everything is correct.

We will enter the LoggerService folder

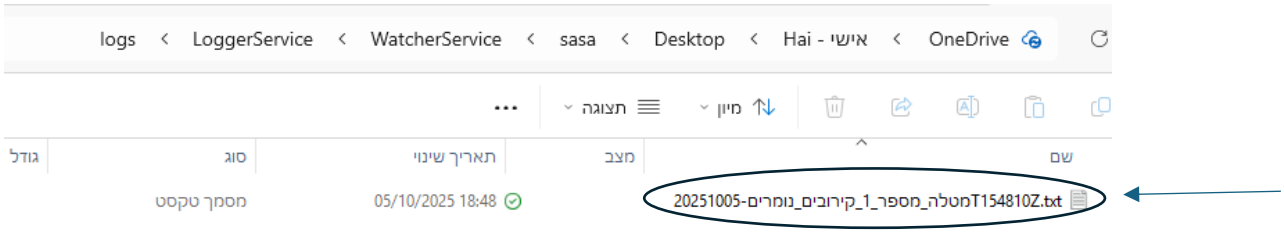



When we enter the folder we will see the logs folder, enter it

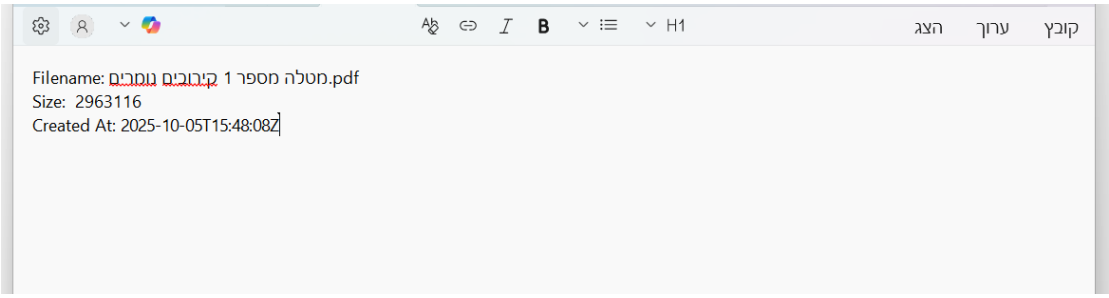


Inside the folder we can see .txt files

as required.




Open the file.




We see all the metadata that we sent from the WatcherService service from the original file using the HTTP communication protocol.