

Fileee.com Upload manager

Fileee.com hat im kostenfreien Service eine Uploadgrenze von 10 Dokumenten, wir müssen also einen Watchdog haben der Fileee nach und nach je nach Erlaubnis füttert.

Setup

Fileee schaut auf WebDav von Nextcloud, daher brauche ich keine Spielereien mit Webcrawlern. Dieses WebDav Folder ist natürlich synchronisiert.

Generator: Perplexity Pro - 2025-02-03

pip install watchdog

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Fileee Upload Manager - Windows Background Service
Verwaltet die Warteschlange für Fileee.com Uploads mit monatlichem Limit
"""

import os
import sys
import time
import json
import shutil
from datetime import datetime
from pathlib import Path
from watchdog.observers import Observer
from watchdog.events import FileSystemEventHandler

class FileeeUploadManager:
    def __init__(self, queue_folder, target_folder, monthly_limit=10):
        self.queue_folder = Path(queue_folder)
        self.target_folder = Path(target_folder)
        self.monthly_limit = monthly_limit
        self.state_file = self.queue_folder / '.fileee_state.json'
```

```

self.queue = []
self.monthly_distribution = {}

# Erstelle Ordner falls nicht vorhanden
self.queue_folder.mkdir(parents=True, exist_ok=True)
self.target_folder.mkdir(parents=True, exist_ok=True)

self.load_state()

def get_current_month_key(self):
    """Gibt aktuellen Monat im Format MMY zurück"""
    now = datetime.now()
    return f"{now.month:02d}{str(now.year)[-2:]}"

def get_month_with_offset(self, offset):
    """Berechnet Monat mit Offset im Format MMY"""
    now = datetime.now()
    month = now.month + offset
    year = now.year

    while month > 12:
        month -= 12
        year += 1

    return f"{month:02d}{str(year)[-2:]}"

def count_existing_files_in_month(self, month_key):
    """Zählt vorhandene PDFs im Monatsordner"""
    month_folder = self.target_folder / month_key
    if not month_folder.exists():
        return 0
    pdf_files = list(month_folder.glob('*.pdf'))
    return len(pdf_files)

def scan_queue(self):
    """Scannt Warteschlangen-Ordner nach PDFs"""
    pdf_files = list(self.queue_folder.glob('*.pdf'))

    for pdf_file in pdf_files:
        if not any(f['path'] == str(pdf_file) for f in self.queue):

```

```

        self.queue.append({
            'path': str(pdf_file),
            'name': pdf_file.name,
            'added': datetime.now().isoformat(),
            'status': 'pending',
            'scheduled_month': None
        })

    self.distribute_files()
    self.save_state()
    print(f"[INFO] {len(self.queue)} Dateien in Warteschlange")

def distribute_files(self):
    """Verteilt Dateien auf Monate basierend auf Limit"""
    self.monthly_distribution = {}
    month_offset = 0

    # Sortiere nach Hinzufügedatum
    self.queue.sort(key=lambda x: x['added'])

    for file in self.queue:
        target_month = self.get_month_with_offset(month_offset)

        if target_month not in self.monthly_distribution:
            self.monthly_distribution[target_month] = []

        # Wenn Monat voll, gehe zum nächsten
        if len(self.monthly_distribution[target_month]) >= self.monthly_limit:
            month_offset += 1
            target_month = self.get_month_with_offset(month_offset)
            if target_month not in self.monthly_distribution:
                self.monthly_distribution[target_month] = []

        self.monthly_distribution[target_month].append(file)
        file['scheduled_month'] = target_month

    current_month = self.get_current_month_key()
    file['status'] = 'scheduled' if target_month == current_month else 'pending'

def process_current_month(self):

```

```

        """Verschiebt Dateien für aktuellen Monat in Zielordner"""
        current_month = self.get_current_month_key()

        if current_month not in self.monthly_distribution:
            return

        month_folder = self.target_folder / current_month
        month_folder.mkdir(exist_ok=True)

        for file in self.monthly_distribution[current_month]:
            if file['status'] == 'scheduled':
                source = Path(file['path'])
                if source.exists():
                    dest = month_folder / source.name
                    try:
                        shutil.move(str(source), str(dest))
                        print(f"[SUCCESS] Verschoben: {source.name} → {current_month}/")
                        self.queue.remove(file)
                    except Exception as e:
                        print(f"[ERROR] Fehler beim Verschieben von {source.name}: {e}")

        self.distribute_files()
        self.save_state()

def save_state(self):
    """Speichert aktuellen Zustand"""
    state = {
        'queue': self.queue,
        'monthly_distribution': self.monthly_distribution,
        'last_update': datetime.now().isoformat()
    }
    with open(self.state_file, 'w', encoding='utf-8') as f:
        json.dump(state, f, indent=2, ensure_ascii=False)

def load_state(self):
    """Lädt gespeicherten Zustand"""
    if self.state_file.exists():
        try:
            with open(self.state_file, 'r', encoding='utf-8') as f:
                state = json.load(f)

```

```

        self.queue = state.get('queue', [])
        self.monthly_distribution = state.get('monthly_distribution', {})
        print(f"[INFO] Zustand geladen: {len(self.queue)} Dateien")
    except Exception as e:
        print(f"[WARNING] Fehler beim Laden des Zustands: {e}")

def run(self):
    """Hauptschleife"""
    print("[INFO] Fileee Upload Manager gestartet")
    print(f"[INFO] Warteschlange: {self.queue_folder}")
    print(f"[INFO] Zielordner: {self.target_folder}")
    print(f"[INFO] Monatslimit: {self.monthly_limit}")

    self.scan_queue()

    # Verarbeite initial
    self.process_current_month()

    last_check_day = datetime.now().day

    while True:
        try:
            # Prüfe auf neuen Monat
            current_day = datetime.now().day
            if current_day != last_check_day and current_day == 1:
                print("[INFO] Neuer Monat erkannt, verarbeite Warteschlange...")
                self.scan_queue()
                self.process_current_month()
                last_check_day = current_day

            # Scanne alle 5 Minuten
            time.sleep(300)
            self.scan_queue()
            self.process_current_month()

        except KeyboardInterrupt:
            print("\n[INFO] Service wird beendet...")
            self.save_state()
            break
    except Exception as e:

```

```

        print(f"[ERROR] Unerwarteter Fehler: {e}")
        time.sleep(60)

if __name__ == '__main__':
    # Konfiguration
    QUEUE_FOLDER = r'WEBDAV-QUEUE'
    TARGET_FOLDER = r'WEBDAV-FOLDER'
    MONTHLY_LIMIT = 10

    # Starte Manager
    manager = FileeeUploadManager(QUEUE_FOLDER, TARGET_FOLDER, MONTHLY_LIMIT)
    manager.run()

```

Das ganze schreibt ihr in einen Ordner ohne Leerzeichen und könnt es dann als Windows Service definieren.

Zum Beispiel mit dem [Non-Sucking-Service-Manager](#) - geht sicher auch irgendwie ohne... Aber wie der Name schon sagt, der Weg würde sicher nicht so geil sein...

```

nssm install FileeeUploadManager PYTHON.EXE SCRIPT.PY
nssm set FileeeUploadManager AppDirectory "SCRIPTORDNER"
nssm set FileeeUploadManager DisplayName "Fileee Upload Manager"
nssm set FileeeUploadManager Description "Verwaltet Fileee.com Upload-Warteschlange"
nssm set FileeeUploadManager Start SERVICE_AUTO_START

```

Bei Fehlern braucht ihr

```

nssm set FileeeUploadManager AppStderr LOGFILE
nssm set FileeeUploadManager AppStdout LOGFILE

```

Darauf seht ihr den neuen Dienst im Service Manager (services.msc) und ihr habt einen Hintergrundprozess der monatlich 10 Files von einem anderen Ordner in euren Webdav Ordner schiebt...

Ob man es brauchen kann oder nicht... War ein kleines Feierabendprojekt bei mir...

Version #1

Erstellt: 2026-02-03 19:43:34 UTC von Konstantin

Zuletzt aktualisiert: 2026-02-03 19:57:50 UTC von Konstantin