Protokoll_Stundenplan_Automatisierung.md

Projekt: Stundenplan-Automatisierung (Python)

Name: Pietros, Ajan, Jonathan

Klasse: TI24BLe/BMa

Datum: 18.12.2025

Einleitung

Im Rahmen dieses Projekts wurde ein Python-Tool entwickelt, das Schul- oder Arbeitsstundenpläne automatisiert verarbeitet. Der Stundenplan wird aus einer PDF-Datei eingelesen, in ein maschinenlesbares JSON-Format überführt und anschliessend genutzt, um zum jeweiligen Unterrichtsbeginn automatisch die zugehörigen Web-Ressourcen zu öffnen.

Das Projekt wurde als Gruppenarbeit umgesetzt und lokal mit Python 3.8+ auf Windows 11 getestet. Das Tool nutzt `pdfplumber` für PDF-Parsing und öffnet Webseiten automatisch über den Standard-Browser. Ziel war eine möglichst robuste Automatisierung mit klarer Benutzerführung und nachvollziehbarer Dokumentation.

Zielsetzung

- Automatisches Einlesen eines Stundenplans aus einer PDF-Datei mittels `pdfplumber`
- Zuordnung von Fächern zu Web-Ressourcen (z. B. Moodle, Teams, Webseiten) durch interaktive Eingabe
- Zeitgesteuertes Öffnen der Webseiten zum Stundenbeginn (alle 30 Sekunden Überprüfung)
- Manueller Zugriff auf Lektionen bei Bedarf (Modi: Auto, Manuell, Anzeige)
- Klare Trennung von Import-, Daten- und Ausführungslogik (3 separate Python-Dateien)
- Verwendung einer virtuellen Umgebung zur Isolation der Abhängigkeiten
- Nachvollziehbare und prüfbare Dokumentation mit korrekten Code-Beispielen

---

Anforderungen

- Python 3.8 oder höher muss installiert sein
- Windows 11
- Python-Paket `pdfplumber` muss installiert sein (`pip install pdfplumber`)
- Stundenplan muss als **textbasiertes PDF** vorliegen
- Zeiten müssen im Format `HH:MM - HH:MM` oder `HH:MM-HH:MM` vorhanden sein
- Internetzugang zum Öffnen der Web-Ressourcen
- Abgabe als Python-Projekt mit Begleitdokumentation (Markdown)

Vorgehen

Das Problem wurde in klar definierte Teilschritte zerlegt:

1. **Beschaffung des Stundenplans** aus dem Intranet als PDF
2. **Parsing des PDFs** (Tage, Fächer, Zeiten)
3. **Interaktive Ergänzung** von URLs pro Fach
4. **Speicherung** aller Daten in `stundenplan.json`
5. **Automatisierte oder manuelle Nutzung** über `main.py`

Entwicklungsprozess

- Erste Version: Nur manuelle Öffnung von Lektionen
- Erkenntnis: Wiederkehrender Tagesablauf → Auto-Modus ergänzt
- Erkenntnis: Unterschiedliche PDF-Layouts → flexible Zeiterkennung eingebaut (sucht Zeit in Zelle oder nächsten 5 Zeilen)
- Erkenntnis: Verschiedene Fach-Formate → Regex-Patterns für "114 - U161D" und kurze Fächer wie "M", "En" implementiert
- Erkenntnis: Parsing kann fehlschlagen → manuelle JSON-Pflege dokumentiert

---

Beschaffung des Stundenplans (Intranet)

1. Anmeldung im Intranet oder auf der Schulplattform (z. B. Moodle, BZU-Intranet)
2. Navigation zu dem Bereich "Stundenplan" oder "Timetable"
3. Download des Stundenplans **als PDF-Datei**
4. Ablage im Projektordner unter `stundenplaene/`
5. Kontrolle:
    - aktueller Zeitraum
    - Mo–Fr vorhanden (vollständige Wochentage werden erkannt)
    - Zeitangaben im Format HH:MM sichtbar
    - Text ist selektierbar (nicht als Bild eingebettet)

Wichtig: Es sollte nur **ein PDF** im Ordner `stundenplaene/` liegen, da das Programm automatisch das erste PDF verwendet.

Nicht geeignet: Screenshots, gescannte PDFs, Word-Dateien, Bilder

---

Umsetzung

Projektstruktur

- `main.py` – Steuerung (Auto-Modus, Manuell, Anzeige)
- `export.py` – Import des PDFs und Erzeugung der JSON-Datei
- `pdf_parser.py` – Extraktion von Tagen, Fächern und Zeiten (Sollte nicht manuell ausgeführt werden)
- `stundenplan.json` – Zentrale Datendatei
- `stundenplaene/` – Ablage der PDF-Dateien
- `.venv/` – Virtuelle Python-Umgebung

Einrichtung der virtuellen Umgebung (VENV)

Warum eine virtuelle Umgebung?

- Isoliert Projektabhängigkeiten vom System-Python
- Verhindert Versionskonflikte mit anderen Projekten
- Ermöglicht saubere Installation von `pdfplumber` ohne Admin-Rechte

Einmalige Einrichtung:

1. PowerShell im Projektordner öffnen
2. Virtuelle Umgebung erstellen:

```
python -m venv .venv
```

3. Virtuelle Umgebung aktivieren:

```
.venv\Scripts\activate
```

4. Abhängigkeiten installieren:

```
pip install pdfplumber
```

Erfolg erkennen:

- Vor dem Prompt erscheint `(.venv)`
- Beispiel: `(.venv) PS C:\workdir\m122_python_automation>`

Bei jeder Nutzung:

- Virtuelle Umgebung aktivieren: `.venv\Scripts\activate`
- Nach der Arbeit deaktivieren: `deactivate`

Import des Stundenplans

Voraussetzung: Virtuelle Umgebung muss aktiviert sein (`(.venv)` sichtbar)!

```
python export.py
```

- Erstes (und einziges) PDF aus `stundenplaene/` wird automatisch gelesen
- Fächer und Zeiten werden automatisch erkannt (mit pdfplumber)
- Benutzer gibt URLs pro Fach ein (interaktive Eingabe)
- Ergebnis wird in `stundenplan.json` gespeichert (mit `erstellt_am` und `version`)

Hinweis: Lege nur ein PDF in den Ordner. Bei mehreren PDFs wird das erste verwendet.

Nutzung der Automatisierung

Voraussetzung: Virtuelle Umgebung muss aktiviert sein (`(.venv)` sichtbar)!

```
python main.py
```

Modi:

- **Modus 1 (Auto-Modus):** Überwacht Uhrzeit alle 30 Sekunden und öffnet Webseiten automatisch zum Stundenbeginn (einmalig pro Lektion)
- **Modus 2 (Manuell):** Zeigt heutige Lektionen an, Benutzer wählt welche geöffnet werden soll
- **Modus 3 (Heute anzeigen):** Zeigt nur Übersicht der heutigen Lektionen ohne Öffnen von Webseiten

---

Manuelle Pflege der Zeiten (Fallback)

Dieser Schritt ist nur notwendig, wenn:

- das PDF keine Zeiten enthält
- das Layout nicht korrekt geparst werden kann

Beispiel-Struktur in `stundenplan.json`:

```
{
  "erstellt_am": "2025-12-17 15:13:55",
  "version": "1.0",
  "stundenplan": {
    "Montag": [
      {
        "fach": "Mathematik",
        "start": "08:00",
        "ende": "09:45",
        "ressourcen": {
          "webseiten": ["https://moodle.bzu.ch/course/view.php?id=123"]
        }
      }
    ],
    "Dienstag": [],
    "Mittwoch": [],
    "Donnerstag": [],
    "Freitag": []
  }
}
```

Regeln:

- 24-Stunden-Format
- `start` < `ende` (Achtung: Feldname ist `ende`, nicht `end`)
- Ohne korrekte Zeiten kein Auto-Modus
- Wochentage: Montag, Dienstag, Mittwoch, Donnerstag, Freitag (vollständig ausgeschrieben)

Testprotokoll

Testumgebung:

- Windows 11
- Python 3.8 oder höher
- Virtuelle Umgebung (.venv) mit pdfplumber
- Standard-Webbrowser (Chrome, Firefox, Edge)

TestNr	Szenario	Erwartung (SOLL)	Ergebnis (IST)	Status
1	Gültiges PDF	JSON wird erstellt	JSON erstellt	OK
2	Auto-Modus	Webseiten öffnen sich	Öffnung korrekt	OK
3	Fehlende Zeiten	Hinweis auf manuelle Pflege	Hinweis angezeigt	OK
4	Manuelle Auswahl	Ressourcen öffnen	Öffnung korrekt	OK

Erkenntnisse

- **Textbasierte PDFs** sind entscheidend für zuverlässiges Parsing (gescannte PDFs funktionieren nicht)
- **Klare Trennung** von Import (`export.py`) und Nutzung (`main.py`) erhöht Wartbarkeit
- **JSON als Zwischenschicht** vereinfacht Debugging und ermöglicht manuelle Korrekturen
- **Virtuelle Umgebung** isoliert Projektabhängigkeiten und verhindert Versionskonflikte
- **Regex-Patterns** müssen flexibel sein um verschiedene PDF-Layouts zu unterstützen (z.B. "114 - U161D" vs "M")
- **Zeitsuche** muss mehrere Zeilen scannen da Zeit nicht immer in derselben Zelle wie Fachname steht
- **Dokumentation der Fallbacks** ist zwingend notwendig bei automatischen Parsing-Tools

Fazit

Das Projekt erfüllt die definierten Anforderungen vollständig. Die Automatisierung spart Zeit im Schulalltag, indem sie Webseiten automatisch zum Stundenbeginn öffnet. Das Tool ist durch die modulare Struktur (3 Python-Dateien) flexibel erweiterbar und gut wartbar.

Erfolgsfaktoren:

- Robustes PDF-Parsing mit `pdfplumber` und flexiblen Regex-Patterns
- Virtuelle Umgebung für saubere Abhängigkeitsverwaltung
- JSON als Zwischenschicht ermöglicht Debugging und manuelle Korrekturen
- Drei Modi (Auto/Manuell/Anzeige) decken verschiedene Nutzungsszenarien ab
- Ausführliche Dokumentation mit Fallback-Strategien

Limitationen:

- Nur textbasierte PDFs werden unterstützt
- Layout-Änderungen im PDF können Anpassungen am Parser erfordern
- Nur Webseiten werden geöffnet, keine Apps

- Wochenenden (Samstag/Sonntag) werden nicht aktiv genutzt

Fehlerfälle sind dokumentiert und können durch manuelle JSON-Bearbeitung korrigiert werden.

Hilfestellungen / Deklaration von KI

Die Entwicklung des Projekts erfolgte mit Unterstützung von KI (GitHub Copilot) zur Ideengenerierung, Code-Vervollständigung, Fehlersuche und Textüberarbeitung. Die Implementierung, Strukturierung und finale Bewertung wurden eigenständig durchgeführt.