

# TASK DOCUMENTATION

***Web Scraping Data DAPODIK Kota Balikpapan, Makassar, dan Palembang***

by Michael Vincent (*Senin, 01/12/2025*)

Program ini dirancang untuk melakukan *web scraping* data detail sekolah dari website Dapodik. Program mengutamakan **kecepatan** melalui pemrosesan paralel (*multithreading*, yang seakan kita memiliki beberapa pekerja) dan **ketepatan** melalui mekanisme *retry* yang adaptif.

## Fungsi Program (*dapodik\_utils.py*)

Source Code: [https://github.com/michaelvincentsebastian/Scraping-Dapodik-Data/blob/main/part2/dapodik\\_utils.py](https://github.com/michaelvincentsebastian/Scraping-Dapodik-Data/blob/main/part2/dapodik_utils.py)

(berisi fungsi yang akan dipanggil di main program)

Program ini berisi semua fungsi *low-level* yang menangani komunikasi jaringan, *parsing* data, dan manajemen file.

### 1. Setup Variabel Global

Variabel-variabel ini menentukan target dan batasan *request*.

```
BASE_URL = 'https://dapo.kemendikdasmen.go.id'  
SEMESTER_ID = '20251'  
REQUEST_TIMEOUT = 10 # detik
```

- **SEMESTER\_ID**: Kode kunci yang menentukan data semester mana yang akan diambil. Kesalahan pada nilai ini (misalnya data belum tersedia) akan menghasilkan *Error 404*.
- **REQUEST\_TIMEOUT**: Batas waktu yang diberikan untuk server merespons (10 detik) sebelum koneksi diputus (*timeout*).

### 2. Fungsi Komunikasi Jaringan (Infinite Retry)

Kedua fungsi *request* ini mengimplementasikan **loop while True** untuk memastikan *request* akan terus dicoba ulang hingga berhasil atau program dihentikan.

#### A. **request\_api()**: Mengambil Data JSON

Fungsi ini mengambil data wilayah atau rekapitulasi sekolah yang formatnya **JSON**.

```
def request_api(...) -> dict:
```

```

while True:
    try:
        # ... (Logika GET request) ...
        if res.status_code != 200:
            print(f"[API] {res.status_code} untuk {url},"
retry dalam {backoff}s...")
            time.sleep(backoff)
            continue # Mencoba kembali
        # ... (Cek balasan HTML/Anti-Bot) ...
        return res.json()
    except Exception as e:
        # ... (Retry jika error jaringan) ...

```

- **Penanganan Error Status Code:** Jika *status code* bukan 200 (termasuk 404 dan 429), program akan jeda (`time.sleep`) lalu menggunakan `continue` untuk mengulang permintaan.

## B. `parse_html()`: Parsing HTML dan Integrasi API

Fungsi ini bertugas menganalisis konten halaman profil sekolah (*HTML Parsing*) dan menggabungkannya dengan data rekapitulasi dinamis (*API Request*).

```

def parse_html(url: str) -> dict:
    req = request_html(url)
    soup = BeautifulSoup(req, 'html.parser')
    # ... (Logika parsing menggunakan BeautifulSoup) ...

    sekolah_id = url.split('/')[-1].strip()
    # Integrasi: Ambil data rekapitulasi menggunakan API
    recapitulation = request_api(sekolah_id=sekolah_id,
backoff=5)
    if recapitulation and isinstance(recapitulation, list):
        school_data["recapitulation"] = recapitulation[0]

    return school_data

```

- Setelah mendapatkan data statis dari HTML, fungsi ini langsung memanggil `request_api()` dengan parameter `sekolah_id` untuk mendapatkan data rekapitulasi (jumlah guru, murid, rombel) dan menyatukannya dalam satu *dictionary*.

### 3. Fungsi Manajemen CSV

Bagian ini memastikan data yang berhasil diambil tersimpan dengan aman dan memungkinkan fitur *resume* (melanjutkan pekerjaan).

#### A. `load_processed_ids()`: Fitur Resume

```
def load_processed_ids(filename: str) -> set:
    processed = set()
    # ...
    # Membaca kolom 'sekolah_id_enkrip' dari file CSV yang sudah ada
    # ...
    return processed
```

- Fungsi ini membaca ID sekolah yang telah tersimpan dan mengembalikannya dalam objek `set`. Objek set memastikan pengecekan ID (*lookup*) di program utama berjalan sangat cepat.

#### B. `append_to_csv()`: Penyimpanan Data Aman (Flush & Fsync)

```
def append_to_csv(filename: str, ...):
    # ...
    try:
        with open(filename, 'a', newline='', encoding='utf-8') as csvfile:
            # ... (Menulis baris data) ...
            csvfile.flush()
            os.fsync(csvfile.fileno())
    return True
```

- `csvfile.flush()` dan `os.fsync()`: Dalam lingkungan *multithreading*, dua perintah ini **wajib** digunakan. Fungsinya adalah memaksa data yang baru ditulis untuk **segera disimpan secara fisik** ke *hard disk*, meminimalkan risiko kehilangan data jika terjadi *crash* program.

# Main Program

Source Code:

[kotaBalikpapan.py](#), [kotaMakassar.py](#), dan [kotaPalembang.py](#)

Program utama mengimplementasikan alur navigasi wilayah dan mengaplikasikan teknik peningkatan kinerja dan keandalan.

## 1. Inisialisasi dan Konfigurasi Kinerja

```
import concurrent.futures # Digunakan untuk Multithreading

MAX_WORKERS = 5           # Jumlah thread maksimum
RETRY_DELAY = 10          # Jeda awal untuk Exponential Backoff
SCHOOL_DETAIL_DELAY = 3   # Jeda per peluncuran thread
```

- **concurrent.futures**: Pustaka standar Python untuk menjalankan tugas secara paralel melalui ThreadPoolExecutor.
- **MAX\_WORKERS**: Membatasi hingga **5 thread** berjalan bersamaan saat mengambil detail sekolah.
- **SCHOOL\_DETAIL\_DELAY**: Digunakan untuk **mengontrol kecepatan (throttling)** peluncuran *thread* baru, mencegah lonjakan *request* instan yang dapat memicu *Error 429*.

## 2. process\_school(): Jaminan Pengambilan Data

Fungsi ini adalah yang dijalankan oleh setiap *thread* dan memiliki fitur keandalan tinggi.

### A. Pengecekan Skip & Loop Tak Terbatas

```
def process_school(...):
    sekolah_id_enkrip = sekolah['sekolah_id_enkrip'].strip()

    # 1. Pengecekan ID yang sudah diproses (Fitur Resume)
    if sekolah_id_enkrip in processed_ids:
        # ... (return True untuk skip) ...
        return True

    retry_count = 0
    while True: # << Loop Coba Ulang Tak Terbatas
        retry_count += 1
        try:
```

```
# ... (Ambil dan simpan data) ...
return sekolah_id_enkrip # KELUAR DARI LOOP jika
sukses
```

```
except Exception as e:
    # ... (Mekanisme Exponential Backoff) ...
```

- **if sekolah\_id\_enkrip in processed\_ids:** Memanfaatkan *set* yang dimuat dari CSV untuk melewati sekolah yang sudah selesai diproses.
- **while True:** Menjamin bahwa jika terjadi kegagalan, *thread tidak akan mengabaikan* sekolah tersebut, melainkan akan terus mencoba.

## B. Mekanisme Exponential Backoff

Bagian ini mengatur waktu tunggu (*delay*) yang bertambah panjang setiap kali terjadi kegagalan.

```
except Exception as e:
    #
    delay_time = RETRY_DELAY * (2 ** (retry_count - 1))
    if delay_time > 120: delay_time = 120 # Batasi jeda
maksimal 2 menit

    print(f"⌚ Menunggu {delay_time:.1f} detik
sebelum mencoba ulang...")
    time.sleep(delay_time)
```

- **Exponential Backoff Formula:** Delay =  $\text{RETRY\_DELAY} * (2^{**(\text{retry\_count} - 1)})$ . Waktu tunggu awal (10 detik) akan berlipat ganda pada setiap percobaan gagal berikutnya (10s, 20s, 40s, dst.).
- **Tujuan:** Strategi ini sangat efektif untuk *web scraping* karena memberikan waktu yang cukup bagi server untuk pulih atau bagi masalah jaringan untuk selesai, **mengurangi beban retry**, dan pada akhirnya **menjamin kelengkapan data** sekolah.

## 3. Eksekusi Paralel

Aplikasi *multithreading* terjadi di *loop* Kecamatan.

```
With
concurrent.futures.ThreadPoolExecutor(max_workers=MAX_WORKERS)
as executor:
```

```
# ...
for sekolah in sekolah_to_process:
    future = executor.submit(process_school, sekolah, ...)
    # ...
    #💡 KONTROL KECEPATAN: Throttling
    time.sleep(SCHOOL_DETAIL_DELAY / MAX_WORKERS)

    # Kumpulkan hasil (menunggu semua thread selesai)
for future in concurrent.futures.as_completed(futures):
    # ... (Perbarui processed_ids) ...
```

- **executor.submit:** Meluncurkan fungsi `process_school` (sebagai tugas independen) ke *thread pool*.
- **Kontrol Kecepatan:** Jeda `time.sleep(SCHOOL_DETAIL_DELAY / MAX_WORKERS)` di antara setiap peluncuran tugas adalah *rate limiting* yang disengaja di sisi *scraper* untuk menjaga kestabilan *request* ke server.