

🕷️ Panduan Lengkap Web Scraper CI4 — Untuk Pemula

Tujuan sistem ini: Mengambil data dari website secara otomatis (scraping) berdasarkan konfigurasi yang sudah ditentukan, menyimpannya ke database, dan mengekstraknya sebagai CSV. Sistem berjalan 24/7 menggunakan Cron Job di shared hosting cPanel.

📋 Daftar Isi

1. [Gambaran Umum Sistem](#)
2. [Struktur File & Penjelasannya](#)
3. [Penjelasan Kode Per File](#)
 - [WebsiteScraperBaru.php — File Konfigurasi](#)
 - [CheckpointStore.php — Penyimpan Progress](#)
 - [RateLimiter.php — Pengatur Jeda](#)
 - [UrlQueue.php — Antrian URL](#)
 - [Fetcher.php — Pengambil HTML](#)
 - [PipelineParser.php — Pengekstrak Data](#)
 - [ClaudeCrawl.php — Otak Scraper](#)
 - [Scraper.php — Command CLI](#)
 - [ScrapeController.php — Controller Web](#)
4. [Setup Awal](#)
5. [Cara Menambah Website Baru](#)
6. [Referensi Selector Khusus](#)
7. [Setup Cron Job di cPanel](#)
8. [Troubleshooting](#)

1. Gambaran Umum Sistem

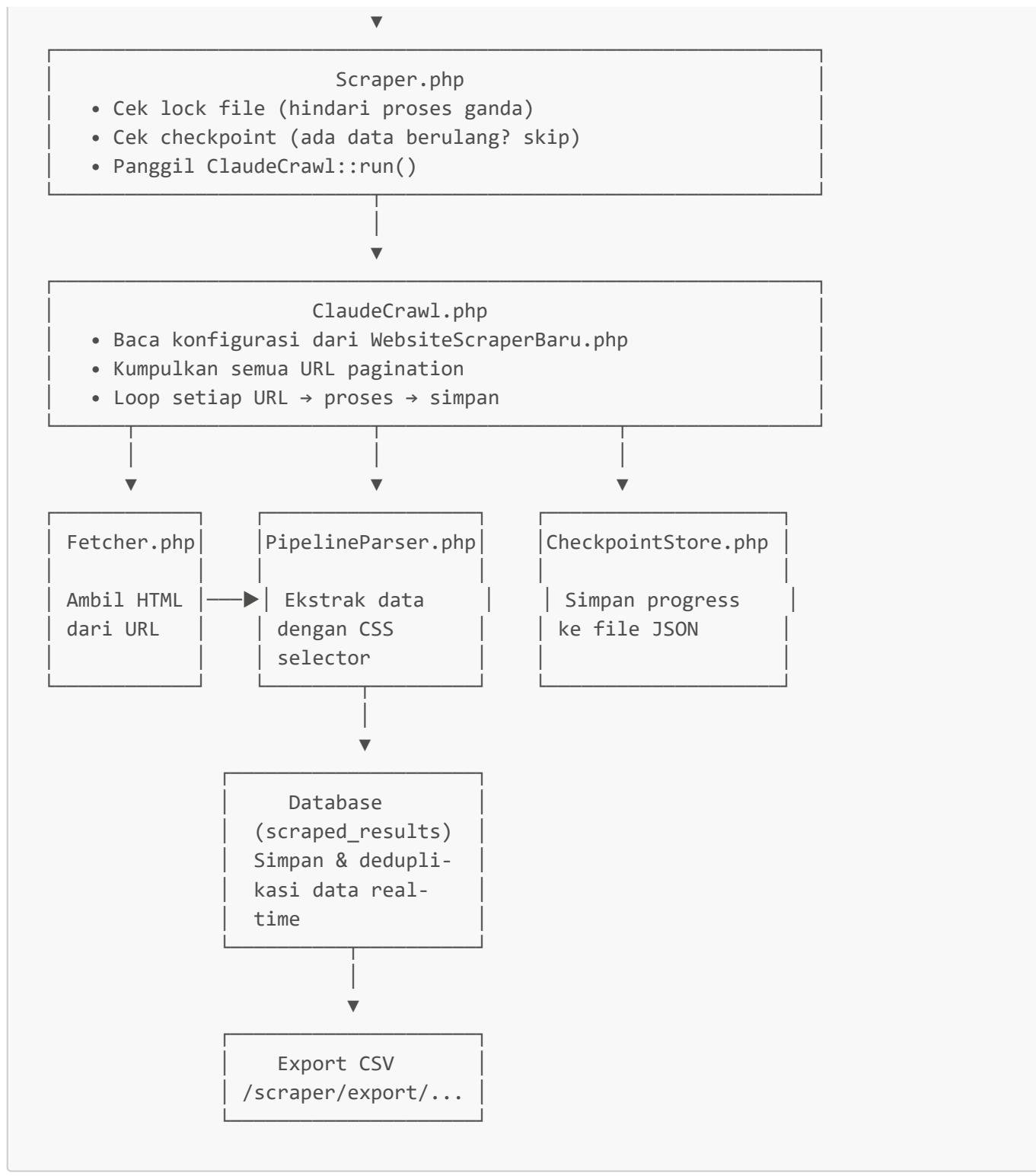
Sistem ini dibangun di atas **CodeIgniter 4 (CI4)** dengan dua mode scraping:

Mode A — Scraping URL Langsung (via Cron Job) Scraper mengunjungi URL website, membaca HTML-nya, mengekstrak data sesuai konfigurasi, dan menyimpannya ke database secara otomatis.

Mode B — Upload File HTML Kamu bisa upload file `.html` yang sudah disimpan secara manual, lalu scraper mengekstrak data dari file tersebut tanpa perlu mengakses internet.

Alur Kerja Sistem

```
CRON JOB (cPanel)
php spark scraper:run kemenperin
```



2. Struktur File & Penjelasannya

```

app/
  └── Commands/
    └── Scrape.php           ← Entry point cron job (CLI command)

  └── Controllers/
    └── ScrapeController.php   ← Halaman web: form, export, upload HTML

  └── Libraries/Crawler/
  
```

— ClaudeCrawl.php	← Otak utama: koordinasi semua komponen
— Fetcher.php	← HTTP client: unduh HTML dari internet
— PipelineParser.php	← Parser: baca HTML, ekstrak field data
— UrlQueue.php	← Antrian URL yang akan dikunjungi
— RateLimiter.php	← Jeda antar request (anti-blokir)
— CheckpointStore.php	← Simpan/muat progress scraping
— Config/	
— WebsiteScraperBaru.php	← KONFIGURASI UTAMA (edit ini untuk tambah website)
— Views/	
— scraper_form.php	← UI halaman web untuk export CSV

3. Penjelasan Kode Per File

3.1 WebsiteScraperBaru.php — File Konfigurasi Utama

File ini adalah satu-satunya file yang perlu kamu edit untuk menambah website baru. Semua pengaturan scraping ada di sini.

```
<?php
namespace Config;
use CodeIgniter\Config\BaseConfig;

class WebsiteScraperBaru extends BaseConfig
{
    public $sites = [
        // Setiap key di sini adalah "nama unik website"
        // Nama ini yang dipakai waktu jalankan: php spark scraper:run NAMA_INI

        'inaexport' => [
            // _____
            // URL halaman pertama yang akan dikunjungi scraper.
            // Biasanya halaman daftar/list dengan parameter page=1
            // _____
            'start_url' => 'https://inaexport.id/suppliers?sortek=asc&page=1',

            // _____
            // Konfigurasi pagination (perpindahan halaman).
            // Hapus bagian ini jika website tidak punya pagination.
            // _____
            'pagination' => [
                // CSS selector untuk elemen link halaman (misal: tombol 1, 2, 3,
                'selector' => 'ul.pagination li.page-item .page-link',
                // Berapa halaman maksimal yang boleh dikunjungi per satu eksekusi
                // cron.
                // Kecilkan nilai ini jika cron sering timeout.
            ]
        ]
    ]
}
```

```
        'max_page' => 1
    ],
    // _____
    // Pengaturan kecepatan request.
    // Jangan terlalu cepat agar tidak diblokir website target.
    // _____
    'rate_limit' => [
        'base_delay_ms' => 800, // Jeda dasar antar request (0.8 detik)
        'max_delay_ms'  => 5000, // Jeda maksimal saat slowdown (5 detik)
        'retry'          => 3    // Berapa kali coba ulang jika gagal
    ],
    // _____
    // Stages = tahapan scraping.
    // Jika website punya halaman list → halaman detail,
    // gunakan 2 stages. Jika hanya halaman list saja, cukup 1 stage.
    // _____
    'stages' => [
        // STAGE 0: Scrape halaman LIST → ambil link ke detail
        [
            'name' => 'list',
            // CSS selector untuk SETIAP ITEM dalam list.
            // Bayangkan ini sebagai "kotak pembungkus" satu baris data.
            'parent' => 'center',
            // Field-field yang mau diambil dari dalam 'parent'.
            // Format: 'nama_kolom' => 'css_selector'
            'fields' => [
                // Field bernama 'detail_url' bersifat SPESIAL:
                // scraper akan mengikuti link ini ke stage berikutnya.
                'detail_url' => 'a.btn[href]'
            ],
            // Beritahu scraper: setelah stage ini,
            // kunjungi URL yang ada di field 'detail_url'
            'next' => 'detail_url'
        ],
        // STAGE 1: Scrape halaman DETAIL → ambil data lengkap
        [
            'name'    => 'detail',
            'parent'  => 'div.shop_area.shop_reverse',
            'fields'  => [
                'company_name' => 'center h3 b',
                'address'       => 'center h5',
                'email'         => 'div.row:contains("Email") .col-sm-9',
                'telephone'     => 'div.row:contains("Telephone") .col-sm-9',
                // ... dst
            ]
        ]
    ]
]
```

```

        ],
        [
        // -----
        // Contoh website KEDUA: hanya satu stage (tidak ada halaman detail)
        // -----
        'kemenperin' => [
            'start_url' => 'https://kemenperin.go.id/direktori-eksportir?
what=%20&prov=32',
            'pagination' => [
                'selector' => 'div.col-md-12 center:first-of-type ul.pagination
a[href]',
                'max_page' => 1
            ],
            'rate_limit' => [
                'base_delay_ms' => 800,
                'max_delay_ms' => 5000,
                'retry' => 3
            ],
            'stages' => [
                [
                    'name' => 'list',
                    // Setiap baris tabel (tr) adalah satu record
                    'parent' => 'table#newspaper-a tbody tr[valign="top"]',
                    'fields' => [
                        // :text(1) = ambil baris teks PERTAMA dari elemen td ke-2
                        'perusahaan' => 'td:nth-child(2) :text(1)',
                        // :text(2) = ambil baris teks KEDUA
                        'alamat' => 'td:nth-child(2) :text(2)',
                        // :text(3):contains("Telp.") = baris ke-3 yang mengandung
                        // kata "Telp."
                        'telp' => 'td:nth-child(2)
:text(3):contains("Telp."'),
                        'email' => 'td:nth-child(2) :text(4):contains("e-
Mail")',
                        'komoditi' => 'td:nth-child(3)',
                        'bidang_usaha' => 'td:nth-child(4)'
                    ]
                ],
                [
                ],
            ],
        ];
    }
}

```

3.2 CheckpointStore.php — Penyimpan Progress

Bayangkan ini seperti "bookmark" — menyimpan posisi terakhir scraping agar bisa dilanjutkan jika cron berikutnya dipanggil.

```

<?php
namespace App\libraries\Crawler;

class CheckpointStore
{
    // Path file checkpoint: writable/crawler_checkpoint_NAMASITE.json
    // Contoh: writable/crawler_checkpoint_kemenperin.json
    protected $file = WRITEPATH . 'crawler_checkpoint_';

    /**
     * Simpan data checkpoint ke file JSON.
     *
     * $data berisi informasi seperti:
     * - last_url          : URL terakhir yang berhasil diproses
     * - url_repeat_count : berapa kali URL yang sama muncul berturut-turut
     * - html_repeat_count : berapa kali HTML yang sama muncul berturut-turut
     * - row_repeat_count  : berapa kali data yang sama muncul berturut-turut
     * - pagination_finished: apakah semua halaman sudah di-scrape
     */
    public function save($data, $site)
    {
        // json_encode($data) → ubah array PHP ke string JSON
        // file_put_contents → tulis ke file (overwrite jika sudah ada)
        file_put_contents($this->file . "$site.json", json_encode($data));
    }

    /**
     * Muat checkpoint dari file JSON.
     * Jika file belum ada (pertama kali run), kembalikan array kosong.
     */
    public function load($site)
    {
        // file_exists      → cek apakah file ada
        // file_get_contents → baca isi file sebagai string
        // json_decode(..., true) → ubah string JSON ke array PHP (bukan object)
        return file_exists($this->file . "$site.json")
            ? json_decode(file_get_contents($this->file . "$site.json"), true)
            : []; // Jika belum ada, kembalikan array kosong
    }
}

```

Contoh isi file crawler_checkpoint_kemenperin.json yang disimpan:

```
{
    "last_url": "https://kemenperin.go.id/direktori-eksportir?page=5",
    "url_repeat_count": 0,
    "html_repeat_count": 0,
    "row_repeat_count": 1,
    "last_html_hash": "a1b2c3d4e5f6...",
    "last_row_hash": "f6e5d4c3b2a1...",
```

```

    "last_max_page_seen": 5,
    "pagination_finished": false,
    "paginate": true
}

```

Penjelasan setiap field:

Field	Tipe	Fungsi
last_url	string	URL terakhir yang berhasil dikunjungi. Jika cron mati di tengah jalan, eksekusi berikutnya mulai dari sini , bukan dari halaman 1
url_repeat_count	int	Berapa kali URL yang sama muncul berturut-turut. Jika ≥ 3 , ini sinyal pagination sudah habis
html_repeat_count	int	Berapa kali HTML yang sama (hash identik) muncul berturut-turut. Mendeteksi server yang selalu return halaman sama
row_repeat_count	int	Berapa kali baris data terakhir (hash row terakhir) sama persis dengan sebelumnya
last_html_hash	string	MD5 dari HTML halaman terakhir yang diproses
last_row_hash	string	MD5 dari row terakhir hasil parse. Dipakai untuk deteksi konten berulang
last_max_page_seen	int	Nomor halaman terbesar yang pernah ditemukan di pagination. Dipakai agar scraper hanya mengambil halaman lebih besar dari ini di eksekusi berikutnya
pagination_finished	bool	<code>true</code> jika sistem mendeteksi pagination sudah habis (stop condition terpenuhi)
paginate	bool	Apakah site ini punya pagination, diambil langsung dari konfigurasi

Kapan checkpoint direset vs dilanjutkan:

- Dilanjutkan otomatis:** Setiap eksekusi cron berikutnya membaca checkpoint, lalu `last_url` dijadikan titik mulai, dan `last_max_page_seen` dijadikan batas bawah nomor halaman yang boleh diambil
- Direset manual:** Hapus file JSON-nya. Ini membuat scraper mulai dari halaman 1 lagi
- Tidak ada mekanisme reset otomatis** — sistem tidak pernah menghapus checkpoint sendiri

3.3 RateLimiter.php — Pengatur Jeda Request

Mencegah scraper dikira bot/spam oleh website target dengan menambahkan jeda acak antar request.

```

<?php
namespace App\libraries\Crawler;

class RateLimiter
{

```

```

// Jeda dasar dalam milidetik (default 600ms = 0.6 detik)
protected $delay;

public function __construct($base = 600)
{
    $this->delay = $base;
}

/**
 * Tunggu sebelum request berikutnya.
 *
 * usleep() menerima nilai dalam MIKRO-detik:
 *   1 detik = 1.000.000 mikro-detik
 *   Jadi (600 + 150) * 1000 = 750.000 µs = 0.75 detik
 *
 * rand(50, 300) = jeda acak 50-300ms agar tidak terlalu teratur
 * (pola yang terlalu teratur lebih mudah dideteksi sebagai bot)
 */
public function wait()
{
    usleep(($this->delay + rand(50, 300)) * 1000);
}

/**
 * Perlambat scraper jika terdeteksi masalah (misal: error HTTP).
 *
 * Kalikan delay dengan 1.5, tapi tidak melebihi 5000ms (5 detik).
 * Contoh progres: 600 → 900 → 1350 → 2025 → 3037 → 4555 → [cap] 5000
 *
 * min() = ambil nilai terkecil dari dua angka
 */
public function slowDown()
{
    $this->delay = min($this->delay * 1.5, 5000);
}
}

```

Catatan penting: Di `ClaudeCrawl`, nilai `base_delay_ms` dari konfigurasi **tidak dioper ke konstruktur RateLimiter**. `ClaudeCrawl` membuat `new RateLimiter()` tanpa argumen, sehingga delay default selalu 600ms. Yang dipakai dari config `rate_limit` hanya `retry`. Nilai `base_delay_ms` dan `max_delay_ms` di `WebsiteScraperBaru.php` saat ini bersifat dokumentasi saja dan **tidak berdampak langsung** pada jeda yang berjalan.

Ada dua fungsi rate limit berbeda di `ClaudeCrawl`:

Fungsi	Dipanggil saat	Jeda
<code>\$this->rate->wait()</code>	Setelah tiap URL diproses di queue utama	600ms + random 50-300ms
<code>\$this->rateLimit()</code> (method lokal)	Saat mengumpulkan URL pagination	Random 200ms-600ms <code>(usleep(rand(200000, 600000)))</code>

Dua fungsi ini terpisah dan independen — jeda saat discovery pagination lebih pendek dari jeda saat scraping konten.

3.4 UrlQueue.php — Antrian URL

Menyimpan daftar URL yang akan dikunjungi dan memastikan URL yang sama tidak dikunjungi dua kali (mencegah loop).

```
<?php
namespace App\libraries\Crawler;

class UrlQueue
{
    // Antrian URL yang menunggu untuk diproses.
    // Format setiap item: ['stage' => 0, 'url' => 'https://...', 'parent' =>
    [...]
    protected $queue = [];

    // Index URL yang sudah pernah dikunjungi.
    // Format: ['https://url-yang-sudah-dikunjungi' => true]
    // Menggunakan array sebagai hash map untuk pencarian O(1) yang cepat.
    protected $visited = [];

    /**
     * Tambahkan URL baru ke antrian.
     *
     * @param int    $stage  Index stage (0 = list, 1 = detail, dst)
     * @param string $url    URL yang akan dikunjungi
     * @param array   $parent Data dari stage sebelumnya (dibawa ke stage
     berikutnya)
     */
    public function push($stage, $url, $parent = [])
    {
        // Cek: apakah URL ini sudah pernah dikunjungi?
        if (!isset($this->visited[$url])) {
            $this->visited[$url] = true; // Tandai sebagai "sudah dikunjungi"

            // compact() adalah shorthand PHP untuk membuat array asosiatif.
            // compact('stage', 'url', 'parent') sama dengan:
            // ['stage' => $stage, 'url' => $url, 'parent' => $parent]
            $this->queue[] = compact('stage', 'url', 'parent');
        }
        // Jika URL sudah ada di $visited, abaikan saja (tidak masuk antrian)
    }

    /**
     * Ambil dan hapus URL pertama dari antrian (FIFO = First In, First Out).
     * Seperti antre di kasir: yang pertama masuk, pertama dilayani.
     *
     * array_shift() menghapus elemen pertama array dan menggeser sisanya.
     */
}
```

```

public function pop()
{
    return array_shift($this->queue);
}

/**
 * Cek apakah masih ada URL yang belum diproses.
 * empty() mengembalikan true jika array kosong.
 */
public function hasJobs()
{
    return !empty($this->queue);
}
}

```

Ilustrasi antrian:

```

Awalnya queue kosong: []

push(0, 'https://site.com/page1') → [{stage:0, url:'page1', parent:[]}]
push(0, 'https://site.com/page2') → [{stage:0, url:'page1'}, {stage:0,
url:'page2'}]
push(0, 'https://site.com/page1') → DIABAIKAN (sudah ada di visited!)

pop() → mengembalikan {stage:0, url:'page1'} dan menghapusnya
Queue sekarang: [{stage:0, url:'page2'}]

```

Tentang field parent:

Field **parent** dipakai untuk sistem multi-stage. Ketika stage 0 (list) menemukan **detail_url**, job untuk stage 1 (detail) di-push dengan membawa data dari stage 0 sebagai **parent**. Ketika stage 1 selesai diproses, datanya di-**array_merge** dengan **parent** — jadi satu record akhir berisi data dari kedua stage sekaligus.

Penting: **\$visited** hanya ada selama satu eksekusi. Setiap cron baru, **UrlQueue** dibuat ulang dari nol, sehingga URL yang sama bisa masuk lagi di eksekusi berikutnya. Deduplikasi permanen antar-eksekusi ditangani oleh **deduplicateResults()** via database, bukan oleh **UrlQueue**.

3.5 Fetcher.php — Pengambil HTML dari Internet

Mengunduh HTML dari sebuah URL, menyamar sebagai browser Chrome agar tidak mudah diblokir website target.

```

<?php
namespace App\libraries\Crawler;

use GuzzleHttp\Client;
use GuzzleHttp\Exception\RequestException;

```

```
class Fetcher
{
    protected $client;

    public function __construct()
    {
        // Buat HTTP client dengan konfigurasi yang menyerupai browser nyata
        $this->client = new Client([
            // Timeout 60 detik.
            // Jika server tidak merespon dalam 60 detik, lempar exception.
            'timeout' => 60,
            // Abaikan error SSL certificate.
            // Berguna untuk website dengan sertifikat expired atau self-signed.
            // Di production yang butuh keamanan, set ke true.
            'verify' => false,
            // Ikuti redirect otomatis (jika server arahkan ke URL lain).
            // Contoh: http://site.com → https://site.com (redirect ke HTTPS)
            'allow_redirects' => [
                'max' => 5, // Maks 5 kali redirect
                'protocols' => ['http', 'https'], // Izinkan HTTP dan HTTPS
            ],
            // Header HTTP – ini yang membuat scraper terlihat seperti browser nyata.
            // Website target membaca header ini untuk menentukan apakah request sah.
            'headers' => [
                // User-Agent: identitas "browser" yang dikirim ke server.
                // String panjang ini adalah User-Agent Chrome 121 di Windows 10.
                'User-Agent' => 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36',
                // Memberitahu server jenis konten yang kita terima
                'Accept' =>
                'text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8',
                // Preferensi bahasa Indonesia (terlihat lebih natural ke server Indonesia)
                'Accept-Language' => 'id-ID,id;q=0.9,en-US;q=0.8,en;q=0.7',
                // Memberitahu server kita menerima response yang dikompresi (lebih cepat)
                'Accept-Encoding' => 'gzip, deflate, br',
                // Seolah-olah ini adalah navigasi dari user (bukan API call)
                'Sec-Fetch-Dest' => 'document',
                'Sec-Fetch-Mode' => 'navigate',
            ],
            // Opsi cURL (library HTTP level rendah yang digunakan Guzzle di bawahnya)
        ]);
    }
}
```

```
'curl' => [
    // Paksa HTTP/1.1 karena banyak WAF (Web Application Firewall)
    // memblokir HTTP/2 dari library non-browser
    CURLOPT_HTTP_VERSION => CURL_HTTP_VERSION_1_1,

    // Paksa penggunaan IPv4 (hindari masalah kompatibilitas IPv6)
    CURLOPT_IPRESOLVE => CURL_IPRESOLVE_V4,

    // Hapus header 'Expect: 100-continue' yang sering menyebabkan
error
    CURLOPT_HTTPHEADER => ['Expect:'],

    // Beberapa server Indonesia butuh SSL Cipher yang lebih lama
    CURLOPT_SSL_CIPHER_LIST => 'DEFAULT@SECLEVEL=1'
],
]);
}

/** 
 * Ambil HTML dari URL tertentu.
 *
 * @param string $url URL yang ingin diambil HTML-nya
 * @return string Konten HTML halaman sebagai string
 * @throws \Exception Jika gagal mengambil HTML (timeout, error HTTP, dll)
 */
public function get($url)
{
    try {
        // Kirim HTTP GET request ke $url
        $response = $this->client->request('GET', $url);

        // Cek status code HTTP response:
        // 200 = OK (berhasil)
        // 403 = Forbidden (diblokir)
        // 404 = Not Found
        // 500 = Internal Server Error
        if ($response->getStatusCode() !== 200) {
            throw new \Exception("HTTP Error: " . $response->getStatusCode());
        }

        // Ambil body (isi HTML) dari response dan kembalikan sebagai string
        return $response->getBody()->getContents();
    } catch (RequestException $e) {
        // RequestException = error dari Guzzle (timeout, DNS fail, connection
refused)
        $msg = $e->getMessage();
        if ($e->hasResponse()) {
            // Jika server merespon tapi dengan status error
            $msg = "Server merespon dengan status: " . $e->getResponse()->getStatusCode();
        }
        throw new \Exception("Gagal mengambil data dari $url. Detail: " .
$msg);
    }
}
```

```
    } catch (\Exception $e) {
        throw new \Exception("Gagal: " . $e->getMessage());
    }
}
```

Error handling di `get()`:

Jika server merespon (status bukan 200) → lempar Exception dengan kode status HTTP
Jika request gagal total (timeout, DNS error) → lempar Exception dengan pesan
Guzzle
Exception ini ditangkap di ClaudeCrawl → rate->slowDown() dipanggil → lanjut ke
URL berikutnya

3.6 PipelineParser.php — Pengekstrak Data dari HTML

Ini adalah "tangan" scraper. ia membaca HTML dan mengambil teks/atribut dari elemen yang ditunjuk CSS selector.

Cara PipelineParser Bekerja (Ringkasan Internal)

Setiap kali scraper memproses satu halaman, `PipelineParser::parse()` dipanggil dengan dua parameter:

- `$html` — string HTML mentah dari halaman tersebut
 - `$selectors` — array konfigurasi dari `stages` di `WebsiteScraperBaru.php`

Alur di dalamnya:

```
parse($html, $selectors)
  |
  └─ filter elemen berdasarkan 'parent'
  |
  └─ jika ada 'group' config → groupElements() dulu
  |
  └─ untuk setiap elemen → resolveField()
      |
      └─ ada ':group'      → resolveGroup()
      └─ ada ':contains'   → resolveContains()
      └─ ada ':text(n)'   → resolveTextIndex()
      └─ standar          → resolveStandard()
```

Parser mendeteksi jenis selector secara otomatis berdasarkan kata kunci `:group`, `:contains`, dan `:text(n)`. Urutan prioritas ini penting: **`:group` selalu dicek pertama**.

Fungsi Utama: `parse()`

```
public function parse($html, $selectors)
{
    // Buat objek Crawler dari string HTML.
    // Crawler adalah library Symfony yang memudahkan navigasi DOM HTML,
    // mirip seperti jQuery di JavaScript.
    $crawler = new Crawler($html);
    $results = [];
    $seenLinks = [];// Index URL yang sudah diambil (cegah duplikat link)

    // 'parent' adalah CSS selector pembungkus setiap record/item.
    // Contoh: 'table tbody tr' → setiap <tr> dianggap satu record.
    $itemSelector = $selectors['parent'];

    // Mendukung array of selectors (jika perlu scrape dari beberapa parent
    berbeda)
    $parentSelectors = is_array($itemSelector) ? $itemSelector : [$itemSelector];

    foreach ($parentSelectors as $currentParent) {

        // Loop setiap elemen HTML yang cocok dengan selector 'parent'.
        // $node = satu elemen HTML (satu record/item)
        $crawler->filter($currentParent)->each(
            function (Crawler $node) use (&$results, &$seenLinks, $selectors) {
                $item = [];// Array untuk menyimpan satu record

                // Loop setiap field yang didefinisikan di konfigurasi
                foreach ($selectors['fields'] as $key => $selector) {
                    try {
                        // Delegasikan ke resolveField() berdasarkan jenis
                        selector
                        $item[$key] = $this->resolveField($node, $key, $selector,
$seenLinks);
                    } catch (\Throwable $e) {
                        $item[$key] = '';// Jika gagal, isi dengan string kosong
                    }
                }

                // Hanya simpan record jika setidaknya ADA SATU field yang tidak
                kosong.
                // array_filter() menghapus nilai falsy (null, '', 0, false).
                if (!empty(array_filter($item))) {
                    $results[] = $item;
                }
            }
        );
    }

    return $results;
}
```

Multiple Parent Selector

Kapan dipakai: Ketika satu halaman punya lebih dari satu struktur HTML untuk data yang sama (misal: dua jenis card berbeda yang keduanya harus diambil).

Cara pakai: Isi `parent` dengan array, bukan string.

```
// config di WebsiteScraperBaru.php
'stages' => [
    [
        'name' => 'list',
        'parent' => ['div.card-type-a', 'div.card-type-b'], // ← array
        'fields' => [
            'nama' => 'h2.title',
            'alamat' => 'p.address',
        ]
    ]
]
```

Cara kerjanya: Parser akan loop dua kali — sekali untuk setiap parent selector — lalu hasil keduanya digabung jadi satu array hasil. Tidak ada duplikat handling otomatis, jadi pastikan kedua selector memang menunjuk elemen yang berbeda.

Kode di `parse()` - Penjelasan Inline Kode

```
public function parse($html, $selectors)
{
    $crawler = new Crawler($html); // Bungkus HTML mentah ke dalam objek Crawler
    Symfony
    $results = []; // Array hasil akhir semua record yang
    berhasil diambil
    $seenLinks = []; // Tracker URL yang sudah dilihat, untuk
    deduplikasi detail_url

    $itemSelector = $selectors['parent']; // Ambil nilai 'parent' dari
    config stage
    $groupConfig = $selectors['group'] ?? null; // Ambil 'group' jika ada, default
    null

    // — MULTIPLE PARENT: normalisasi ke array —————
    // Jika 'parent' berupa string biasa → bungkus jadi array 1 elemen
    // Jika 'parent' sudah berupa array → pakai langsung
    // Ini membuat logika di bawah selalu bisa loop, baik 1 maupun banyak parent
    $parentSelectors = is_array($itemSelector) ? $itemSelector : [$itemSelector];

    // Loop setiap parent selector satu per satu
    foreach ($parentSelectors as $currentParent) {

        if ($groupConfig !== null) {
            // — JALUR A: Ada konfigurasi 'group' di level stage —————
```

```

// Kumpulkan dulu semua elemen, lalu kelompokkan sesuai groupConfig
$groups = $this->groupElements($crawler, $currentParent,
$groupConfig);

foreach ($groups as $groupNodes) {
    $item = [];

    // Buat HTML sementara dari semua node dalam satu grup
    // Ini perlu karena node-node dalam grup tidak punya parent
bersama
    $groupHtml = '';
    foreach ($groupNodes as $node) {
        $groupHtml .= $node->outerHtml(); // Ambil HTML lengkap tiap
node
    }

    // Bungkus HTML gabungan ke Crawler baru agar bisa di-filter
    $groupCrawler = new Crawler($groupHtml);

    foreach ($selectors['fields'] as $key => $selector) {
        // Skip key 'parent' dan 'group' jika tidak sengaja masuk ke
fields
        if ($key === 'parent' || $key === 'group') continue;

        try {
            // resolveField akan otomatis deteksi jenis selector
            // (:group, :contains, :text, atau standard)
            $item[$key] = $this->resolveField($groupCrawler, $key,
$selector, $seenLinks);
        } catch (\Throwable $e) {
            $item[$key] = ''; // Jika error apapun, isi dengan string
kosong
        }
    }

    // Hanya simpan record jika setidaknya ada 1 field yang tidak
kosong
    if (!empty(array_filter($item))) {
        $results[] = $item;
    }
}

} else {
    // — JALUR B: Tidak ada 'group' – perilaku normal ——————
    // Filter semua elemen yang cocok dengan parent selector
    // lalu proses tiap elemen sebagai 1 record langsung
    $crawler->filter($currentParent)->each(function (Crawler $node) use
(&$results, &$seenLinks, $selectors) {
        $item = [];

        foreach ($selectors['fields'] as $key => $selector) {
            if ($key === 'parent' || $key === 'group') continue;

            try {

```

```

        $item[$key] = $this->resolveField($node, $key, $selector,
$seenLinks);
    } catch (\Throwable $e) {
        $item[$key] = '';
    }
}

if (!empty(array_filter($item))) {
    $results[] = $item;
}
});

// Setelah loop selesai untuk parent ini, lanjut ke parent berikutnya
(jika ada)
// Hasilnya ditambahkan ke $results yang sama → semua parent digabung
}

return $results; // Array semua record dari semua parent selector
}

```

Poin kritis: Karena `$results` adalah satu array bersama yang diisi oleh semua iterasi `$parentSelectors`, hasil dari parent selector pertama dan kedua **langsung digabung** tanpa pemisahan. Urutan record di output mengikuti urutan parent selector di config.

Sistem Resolver: `resolveField()`

```

protected function resolveField(Crawler $node, string $key, string $selector,
array &$seenLinks)
{
    // Sistem ini memeriksa "fitur khusus" dalam selector secara berurutan.
    // Prioritas: :group → :contains → :text(n) → CSS biasa

    // 1. Jika selector mengandung ":group" → gabungkan beberapa elemen
    if ($this->hasGroup($selector)) {
        return $this->resolveGroup($node, $selector);
    }

    // 2. Jika selector mengandung ":contains(...)" → cari elemen berdasarkan teks
    if ($this->hasContains($selector)) {
        return $this->resolveContains($node, $selector);
    }

    // 3. Jika selector mengandung ":text(n)" → ambil baris teks ke-n
    if ($this->hasTextIndex($selector)) {
        return $this->resolveTextIndex($node, $selector);
    }

    // 4. Tidak ada fitur khusus → gunakan CSS selector standar
    return $this->resolveStandard($node, $key, $selector, $seenLinks);
}

```

Kode `hasGroup()`

```

protected function hasGroup(string $selector): bool
{
    // Cari substring ':group' di mana saja dalam string selector
    // strpos() return int (posisi) jika ditemukan, false jika tidak
    // Casting ke bool: int apapun (termasuk 0) → true, false → false
    //
    // Contoh yang return true:
    //   "span.tag :group"
    //   "div.info :group :text(2)"
    //   "p :group :contains("Email")"
    //   "td :group :text(1) :contains("HP")"
    //
    // Contoh yang return false:
    //   "div.card"
    //   "td:nth-child(2)"
    //   "div.row:contains("Email") .col"      ← tidak ada :group
    //   "td :text(3)"                      ← tidak ada :group
    return strpos($selector, ':group') !== false;

    // Mengapa strpos, bukan preg_match?
    // ':group' adalah string literal yang tidak butuh pattern matching.
    // strpos() jauh lebih cepat dari regex untuk pengecekan substring sederhana.
}

```

Kode `hasContains()`

```

protected function hasContains(string $selector): bool
{
    // Cari substring ':contains' di mana saja dalam string selector
    //
    // Contoh yang return true:
    //   "div.row:contains(\"Email\") .col-sm-9"
    //   "div.row:contains(\"Telephone\")"
    //   "td :text(2) :contains(\"HP\")"           ← kombinasi dengan :text(n)
    //   "p.info:contains(\"Alamat\")"
    //
    // Contoh yang return false:
    //   "div.card"
    //   "span.tag :group"
    //   "td :text(3)"
    //   "a[href]"
    //
    // CATATAN PENTING:
    // hasGroup() selalu dicek SEBELUM hasContains() di resolveField().
    // Artinya selector seperti "span :group :contains(\"x\")" tidak akan
}

```

```
// pernah sampai ke hasContains() – sudah ditangkap hasGroup() lebih dulu.
// hasContains() hanya menerima selector yang TIDAK mengandung ':group'.
return strpos($selector, ':contains') !== false;
}
```

Kode `hasTextIndex()`

```
protected function hasTextIndex(string $selector): bool
{
    // Gunakan regex karena perlu mencocokkan POLA, bukan substring literal
    // Pattern: ':text(' diikuti satu atau lebih digit, diikuti ')'
    // Flag /i = case-insensitive ("TEXT(1)" atau ":Text(1)" juga valid)
    //
    // Breakdown regex '/\:text\(\d+\)/i':
    //   \:      → karakter titik dua literal (di-escape agar tidak ambigu)
    //   text    → string literal "text"
    //   \(     → karakter '(' literal (di-escape karena ( punya makna di regex)
    //   \d+    → satu atau lebih digit (0-9) → angka indeksnya
    //   \)     → karakter ')' literal
    //
    // Contoh yang return true:
    //   "td :text(1)"
    //   "td:nth-child(2) :text(3)"
    //   "div.info :TEXT(2)"           ← case-insensitive
    //   "span :text(10)"            ← dua digit juga valid
    //
    // Contoh yang return false:
    //   "td:nth-child(2)"           ← nth-child bukan :text(n)
    //   "div.card"
    //   "span.tag :group"
    //   ":text()"                  ← tidak ada digit → \d+ tidak cocok
    //   ":text(0)"                  ← return TRUE (regex cocok) tapi :text(0)
    //                                akan jadi index -1 di resolveTextIndex,
    //                                sehingga return '' (tidak ada elemen ke-0)
    //
    // CATATAN PENTING:
    // hasGroup() dan hasContains() dicek sebelum hasTextIndex() di
    resolveField().
    // Selector "td :text(2) :contains("HP")" akan masuk ke hasContains() (true),
    // bukan ke hasTextIndex() – meski mengandung :text(n).
    // resolveContains() yang kemudian menangani :text(n) di dalam base selector-
    nya.
    return (bool) preg_match('/\:text\(\d+\)/i', $selector);
}
```

Tabel Keputusan `resolveField()` Berdasarkan Ketiga Fungsi

Selector	hasGroup	hasContains	hasTextIndex	Fungsi yang Dipanggil
"div.card"	false	false	false	resolveStandard()
"a[href]"	false	false	false	resolveStandard()
"td :text(2)"	false	false	true	resolveTextIndex()
"div.row:contains("Email") .col"	false	true	false	resolveContains()
"td :text(2) :contains("HP"))"	false	true	true	resolveContains() ← contains menang
"span :group"	true	false	false	resolveGroup()
"span :group :text(2)"	true	false	true	resolveGroup() ← group menang
"span :group :contains("x"))"	true	true	false	resolveGroup() ← group menang
"span :group :text(1) :contains("x))"	true	true	true	resolveGroup() ← group menang

Aturan sederhana: Kolom paling kiri yang **true** menentukan pemenang. **:group** selalu menang, lalu **:contains**, lalu **:text(n)**, lalu fallback standard.

CSS Selector Standar: resolveStandard()

```
protected function resolveStandard(Crawler $node, string $key, string $selector,
array &$seenLinks)
{
    // Cari elemen di dalam $node yang cocok dengan $selector
    $element = $node->filter($selector);

    // Jika tidak ditemukan elemen apapun, kembalikan string kosong
    if (!$element->count()) return '';

    // KASUS KHUSUS 1: Field bernama 'detail_url'
    // Ambil atribut href (URL link) bukan teks
    if ($key === 'detail_url') {
        $url = trim($element->attr('href') ?? '');
        // Abaikan jika URL kosong ATAU sudah pernah diambil sebelumnya
        if ($url === '' || isset($seenLinks[$url])) return '';
        $seenLinks[$url] = true; // Tandai URL ini sudah diambil
        return $url;
    }

    // KASUS KHUSUS 2: Field yang namanya mengandung kata 'image'
    // Ambil atribut src (URL gambar) bukan teks
    if (strpos($key, 'image') !== false) {
```

```

        return $element->attr('src') ?? '';
    }

    // DEFAULT: Ambil teks dari elemen, hapus spasi di awal dan akhir
    return trim($element->text());
}

```

Selector :text(n): resolveTextIndex()

```

protected function resolveTextIndex(Crawler $node, string $selector)
{
    // Contoh selector: 'td:nth-child(2) :text(3)'
    // Regex ini memisahkan dua bagian:
    //   Group 1 (base): 'td:nth-child(2)'
    //   Group 2 (index): '3'
    preg_match('/^(.+?)\s+:text\((\d+)\)$/i', $selector, $m);

    $baseSelector = trim($m[1] ?? ''); // 'td:nth-child(2)'
    $index = ((int)($m[2] ?? 1)) - 1; // 3 - 1 = 2 (array 0-indexed)

    $element = $node->filter($baseSelector);
    if (!$element->count()) return '';

    // extractTextParts() memecah isi elemen menjadi array baris teks
    // menggunakan 5 strategi berbeda: <br> split, child blocks, DOM nodes,
    // newline, fallback
    $parts = $this->extractTextParts($element);

    // Ambil baris ke-index, atau string kosong jika tidak ada
    return $parts[$index] ?? '';
}

```

Contoh cara kerja :text(n):

HTML yang di-scrape:

```

<td>
    PT. Contoh Jaya<br>
    Jl. Merdeka No. 1, Jakarta<br>
    Telp. 021-1234567<br>
    e-Mail: info@contoh.com
</td>

```

extractTextParts() menghasilkan:

```

[
    0 => "PT. Contoh Jaya",
    1 => "Jl. Merdeka No. 1, Jakarta",
    2 => "Telp. 021-1234567",
    3 => "e-Mail: info@contoh.com"
]

```

```
'td :text(1)' → index 0 → "PT. Contoh Jaya"
'td :text(2)' → index 1 → "Jl. Merdeka No. 1, Jakarta"
'td :text(3)' → index 2 → "Telp. 021-1234567"
'td :text(4)' → index 3 → "e-Mail: info@contoh.com"
```

:group — Gabungkan Beberapa Elemen Menjadi Satu Field

Kapan dipakai: Ketika data yang kamu inginkan tersebar di beberapa elemen sibling (saudara) yang terpisah, dan kamu perlu menggabungkannya.

Format dasar:

```
'field_name' => 'css_selector :group'
```

Cara kerjanya:

1. Parser mengambil semua elemen yang cocok dengan `css_selector`
2. Teks dari setiap elemen digabung dengan separator internal `<g>`
3. Separator dibersihkan (email, koma, titik dua dijaga agar tidak terpotong)
4. Hasil akhir dikembalikan sebagai satu string

Kode `resolveGroup()` - Penjelasan Inline Kode

```
protected function resolveGroup(Crawler $node, string $selector)
{
    // Selector masuk contohnya:
    //   "span.tag :group"
    //   "span.tag :group :text(2)"
    //   "span.tag :group :contains("Email")"
    //   "span.tag :group :text(1) :contains("HP")"

    // Pisahkan base selector (sebelum :group) dari modifier (setelah :group)
    // Regex: tangkap semua karakter (lazy) sebelum spasi+:group, sisanya jadi
    // modifier
    preg_match('/^(.+?)\s+:group(.*)$/i', $selector, $matches);

    $baseSelector = trim($matches[1] ?? '');
    $modifiers    = trim($matches[2] ?? '');

    if ($baseSelector === '') return '';

    // Ambil semua elemen DOM yang cocok dengan baseSelector
    $elements = $node->filter($baseSelector);
    if (!$elements->count()) return '';

    // — TAHAP 1: Kumpulkan teks semua elemen
```

```

$grouped = [];
$elements->each(function ($el) use (&$grouped) {
    $text = $this->normalizeText($el->text()); // Bersihkan Unicode, spasi,
    dll
    if ($text !== '') {
        $grouped[] = $text; // Hanya tambahkan jika teks tidak kosong setelah
        normalisasi
    }
});

// — TAHAP 2: Gabungkan dengan separator _____
// dipilih sebagai separator karena tidak mungkin muncul dalam teks HTML
normal
$groupedText = implode(' ', $grouped);

// Bersihkan separator yang "salah posisi" (misal: di tengah email)
// cleanGroupSeparators() menangani kasus edge seperti: "abc@def.com"
$groupedText = $this->cleanGroupSeparators($groupedText);

// Pecah kembali by untuk kebutuhan modifier :text(n)
// Sekarang $parts[0] = bagian pertama, $parts[1] = bagian kedua, dst
$parts = array_map('trim', explode(' ', $groupedText));

// — TAHAP 3: Proses modifier ——————
// Tidak ada modifier → return semua bagian digabung dengan spasi
if ($modifiers === '') {
    return implode(' ', $parts);
}

// Ada :text(n) → ambil bagian ke-n (1-indexed dari config, 0-indexed di
array)
if (preg_match('/:text\\((\\d+)\\)/i', $modifiers, $textMatch)) {
    $index = ((int)$textMatch[1]) - 1; // Konversi: :text(1) → index 0
    $text = $parts[$index] ?? ''; // Ambil, atau kosong jika index tidak
ada

    // Ada :contains juga setelah :text? → cari label dalam teks bagian
    tersebut
    if (preg_match('/:contains\\("(.)"\\)/i', $modifiers, $containsMatch)) {
        $label = $containsMatch[1];
        return $this->extractAfterLabel($text, $label); // Ekstrak nilai
        setelah label
    }

    return $text; // Return bagian ke-n tanpa filter label
}

// Ada :contains tapi tidak ada :text → cari label di semua bagian sekaligus
if (preg_match('/:contains\\("(.)"\\)/i', $modifiers, $containsMatch)) {
    $label = $containsMatch[1];
    $fullPart = implode(' ', $parts); // Gabungkan semua bagian dulu

    if (stripos($fullPart, $label) !== false) { // Label ditemukan?
}

```

```

        return $this->extractAfterLabel($fullPart, $label); // Ekstrak
nilainya
    }

    return ''; // Label tidak ditemukan di manapun
}

// Fallback: ada modifier yang tidak dikenali → return groupedText apa adanya
return $groupedText;
}

```

Visualisasi alur untuk '`span.tag :group :text(2)`':

HTML: Ekspor
Makanan
Sertified

Step 1: filter "span.tag" → 3 elemen
Step 2: ambil teks → ["Ekspor", "Makanan", "Sertified"]
Step 3: implode → "Ekspor <g> Makanan <g> Sertified"
Step 4: cleanGroupSeparators → (tidak ada perubahan)
Step 5: explode → ["Ekspor", "Makanan", "Sertified"]
Step 6: modifier ":text(2)" → index 1 → "Makanan"
Output: "Makanan"

Contoh:

```

// HTML:
// <div class="info">Alamat: Jl. A</div>
// <div class="info">Kota: Jakarta</div>
// <div class="info">Kode Pos: 12345</div>

'alamat_lengkap' => 'div.info :group'
// Hasil: "Alamat: Jl. A Kota: Jakarta Kode Pos: 12345"

```

Kombinasi `:group + :text(n)` — ambil elemen ke-n dari hasil group:

```

'alamat_lengkap' => 'div.info :group :text(2)'
// Hasil: "Kota: Jakarta" ← hanya bagian ke-2

```

Kombinasi `:group + :contains("label")` — cari label tertentu dalam hasil group:

```

'kode_pos' => 'div.info :group :contains("Kode Pos")'
// Hasil: "12345" ← teks setelah "Kode Pos:"

```

Kombinasi :group + :text(n) + :contains("label") — filter baris ke-n lalu cari label:

```
'nilai' => 'td.data :group :text(2) :contains("Total")'
```

Konfigurasi **group** di Level Stage — Grouping Elemen Sibling

Ini berbeda dari **:group** selector di atas. Konfigurasi **group** di level **stage** digunakan ketika **satu record data terdiri dari beberapa elemen HTML yang berurutan**, bukan terbungkus dalam satu parent yang sama.

Ada dua mode:

Mode 1: Numeric Grouping — Setiap N Elemen = 1 Record

```
'stages' => [
  [
    'name'    => 'list',
    'parent'  => 'table.data tr',    // semua baris tabel
    'group'   => 3,                  // setiap 3 baris = 1 record
    'fields'  => [
      'nama'    => 'td:first-child',
      'alamat'  => 'td:nth-child(2)',
      'telepon' => 'td:nth-child(3)'
    ]
  ]
]

// HTML:
// <tr><td>PT. A</td></tr>
// <tr><td>Jl. B</td></tr>
// <tr><td>021-xxx</td></tr>
// → 1 record: {nama: PT.A, alamat: Jl.B, telepon: 021-xxx}
```

Mode 2: Selector Range Grouping — Dari Elemen START sampai Elemen START berikutnya

```
'stages' => [
  [
    'name'    => 'list',
    'parent'  => 'div.content',        // container
    'group'   => ['h3.company', 'hr'], // [start_selector, end_selector]
    'fields'  => [
      'nama'    => 'h3.company',
      'detail'  => 'p.desc',
    ]
  ]
]
```

Cara kerjanya: Parser iterasi semua child dari `parent`. Setiap kali menemukan elemen yang cocok dengan `start_selector` (index 0), grup sebelumnya disimpan dan grup baru dimulai. Grup terus bertambah sampai `start_selector` berikutnya ditemukan.

Catatan: Parameter `end_selector` (index 1) saat ini tidak digunakan secara aktif dalam kode — pengelompokan murni berdasarkan `start_selector`. Parameter ini disiapkan untuk pengembangan ke depan.

Kode `groupElements()` — Router Mode Grouping

```
protected function groupElements(Crawler $crawler, string $itemSelector,
                                $groupConfig): array
{
    // Kumpulkan semua elemen yang cocok dengan itemSelector ke dalam array biasa
    // (Crawler tidak bisa di-index langsung, jadi harus di-collect dulu)
    $elements = [];
    $crawler->filter($itemSelector)->each(function (Crawler $node) use
    (&$elements) {
        $elements[] = $node; // Simpan setiap Crawler node ke array
    });

    if (empty($elements)) {
        return []; // Tidak ada elemen sama sekali, return kosong
    }

    $groups = [];

    // — MODE 1: Numeric grouping —————
    // Jika groupConfig adalah angka (misal: 3), maka setiap 3 elemen = 1
    // grup/record
    if (is_numeric($groupConfig)) {
        $groupSize = (int) $groupConfig;

        // array_chunk memecah array flat menjadi potongan-potongan berukuran
        // $groupSize
        // misal: 9 elemen dengan groupSize=3 → 3 grup, masing-masing isi 3 elemen
        $chunks = array_chunk($elements, $groupSize);

        return $chunks; // [[el1,el2,el3], [el4,el5,el6], [el7,el8,el9]]
    }

    // — MODE 2: Selector range grouping —————
    // Jika groupConfig adalah array 2 elemen: ['start_selector', 'end_selector']
    // Delegasikan ke fungsi grouping() yang lebih spesifik
    if (is_array($groupConfig) && count($groupConfig) === 2) {
        $res = $this->grouping($crawler, $itemSelector, $groupConfig);
        return $res;
    }

    // — Fallback —————
    // groupConfig tidak dikenali → setiap elemen jadi grup sendiri (1 elemen per
    // grup)
}
```

```

    return array_map(function ($el) {
        return [$el]; // Bungkus tiap elemen dalam array agar format tetap
    konsisten
    }, $elements);
}

```

Kode `grouping()` — Mode Selector Range

Kode ini adalah helper (lihat penggunaan pada kode di atas mode 2)

```

public function grouping(Crawler $crawler, string $itemSelector, $groupConfig)
{
    // Destruktur array config: index 0 = start selector, index 1 = end selector
    // Contoh: ['h3.company-name', 'hr.divider']
    [$startSelector, $endSelector] = $groupConfig;

    // Ambil parent container berdasarkan itemSelector
    $parent = $crawler->filter($itemSelector);
    // Lalu ambil semua child langsung dari parent tersebut
    $children = $parent->children();

    $groups = []; // Hasil akhir: array of array of Crawler nodes
    $currentGroup = []; // Grup yang sedang dibangun

    $children->each(function (Crawler $child, $i) use (&$groups, &$currentGroup,
    $endSelector, $startSelector) {

        // Apakah child ini cocok dengan startSelector?
        if ($child->matches($startSelector)) {

            // Jika ada grup yang sedang berjalan, simpan dulu sebelum mulai yang
            baru
            if (!empty($currentGroup)) {
                $groups[] = $currentGroup; // Tutup dan simpan grup lama
            }

            // Mulai grup baru, dengan elemen start ini sebagai anggota pertama
            $currentGroup = [$child];

        } elseif (!empty($currentGroup)) {
            // Bukan start selector, tapi kita sedang dalam sebuah grup
            // → tambahkan elemen ini ke grup yang sedang berjalan
            $currentGroup[] = $child;
        }
        // Jika currentGroup masih kosong dan bukan startSelector
        // → elemen ini diabaikan (belum ada grup yang aktif)

        // CATATAN: Logika end selector di bawah ini sengaja di-
        comment/nonaktifkan.
        // Artinya pengelompokan HANYA berdasarkan startSelector.
        // Grup berakhir ketika startSelector BERIKUTNYA ditemukan, bukan ketika
    })
}

```

```

        // endSelector ditemukan. endSelector di config saat ini tidak
berpengaruh.
        //
        // if ($child->matches($endSelector)) {
        //     if (!empty($currentGroup)) {
        //         $groups[] = $currentGroup;
        //         $currentGroup = [];
        //     }
        // }
    });

// Setelah loop selesai, masih ada grup terakhir yang belum disimpan
// (karena tidak ada startSelector berikutnya yang memicunya tersimpan)
if (!empty($currentGroup)) {
    $groups[] = $currentGroup;
}

return $groups;
// Format return: [[node_h3, node_p, node_p], [node_h3, node_p], ...]
//                         ↑ grup 1                           ↑ grup 2
}

```

Visualisasi alur selector range dengan 'group' => ['h3', 'hr']:

HTML children dari parent:

```

<h3>Perusahaan A</h3> ← matches startSelector 'h3' → simpan grup lama
(kosong), mulai grup baru
<p>Alamat A</p>           ← dalam grup aktif → tambahkan
<p>Telp A</p>             ← dalam grup aktif → tambahkan
<hr>                      ← BUKAN h3, dalam grup aktif → tambahkan (end selector
diabaikan!)
<h3>Perusahaan B</h3>   ← matches startSelector 'h3' → SIMPAN grup 1, mulai grup
2
<p>Alamat B</p>           ← dalam grup aktif → tambahkan

```

Selesai loop → simpan grup 2

Hasil:

```

Grup 1: [<h3>Perusahaan A</h3>, <p>Alamat A</p>, <p>Telp A</p>, <hr>]
Grup 2: [<h3>Perusahaan B</h3>, <p>Alamat B</p>]

```

Penting: Karena end selector dinonaktifkan, `<hr>` ikut masuk ke grup 1 (bukan memutus grup). Batas antar grup murni ditentukan oleh kemunculan `startSelector` berikutnya.

9.6 :contains("label") — Detail Penggunaan Lanjutan

Panduan utama menunjukkan contoh sederhana. Berikut pola-pola yang juga didukung:

Pola A — Standard (sudah di panduan):

```
'telepon' => 'div.row:contains("Telephone") .col-sm-9'
// Cari <div class="row"> yang teksnya mengandung "Telephone",
// lalu ambil teks dari child .col-sm-9
```

Pola B — Tanpa tail selector, ambil teks setelah label:

```
'telepon' => 'div.row:contains("Telephone")'
// Cari <div class="row"> yang mengandung "Telephone",
// lalu otomatis ambil teks SETELAH kata "Telephone" (dan tanda : atau -)
```

Pola C — Kombinasi dengan :text(n):

```
// HTML: <td>Nama: PT. ABC<br>Telepon: 021-123</td>
'telepon' => 'td :text(2) :contains("Telepon")'
// Ambil baris ke-2 dari td, lalu ekstrak teks setelah "Telepon:"
// Hasil: "021-123"
```

Cara extractAfterLabel bekerja:

Fungsi ini mencari baris yang mengandung label, lalu mencoba regex:

```
LABEL\s*[:-]?\s*([^\n\r]+)
```

Artinya: ambil semua teks setelah label + opsional titik dua atau dash. Jika regex gagal, semua teks sebelum label dihapus.

Kode resolveContains() — Entry Point

```
protected function resolveContains(Crawler $node, string $selector)
{
    // Parsing selector dengan regex, pisahkan 3 bagian:
    // $base = selector CSS sebelum :contains(...)
    // $label = teks di dalam tanda kutip :contains("TEKS_INI")
    // $tail = selector CSS setelah :contains(...), jika ada
    //
    // Contoh 1: 'div.row:contains("Email") .col-sm-9'
    // $base = 'div.row'
    // $label = 'Email'
    // $tail = '.col-sm-9'
    //
    // Contoh 2: 'div.row:contains("Email")'
    // $base = 'div.row'
    // $label = 'Email'
    // $tail = '' (kosong)
```

```

// Contoh 3: 'td :text(2) :contains("HP")'
// $base = 'td :text(2)' ← base mengandung :text(n)!
// $label = 'HP'
// $tail = ''
preg_match('/^(.+?):contains\("(.)"\)(.*$/i', $selector, $m);

$base = trim($m[1] ?? '');
$label = $m[2] ?? '';
$tail = trim($m[3] ?? '');

if ($base === '' || $label === '') return ''; // Guard: selector tidak lengkap

// — ROUTING: tentukan jalur berdasarkan isi $base ——————
// CASE A: base mengandung :text(n)
// Artinya kita perlu ambil baris ke-n dulu, BARU cari label di dalamnya
if ($this->hasTextIndex($base)) {
    return $this->resolveContainsFromText($node, $base, $label);
}

// CASE B: base adalah CSS selector biasa (mungkin ada $tail, mungkin tidak)
return $this->resolveContainsFromElements($node, $base, $label, $tail);
}

```

Kode `resolveContainsFromText()` — Case A: Base Ada `:text(n)`

Kode ini adalah helper (digunakan pada fungsi `resolveContains()`)

```

protected function resolveContainsFromText(Crawler $node, string $baseSelector,
string $label)
{
    // Contoh input:
    // $baseSelector = 'td :text(2)'
    // $label        = 'HP'

    // Pisahkan CSS selector asli dari angka :text(n)
    preg_match('/^(.+?)\s+:text\((\d+)\)$i', $baseSelector, $m);

    $real = trim($m[1] ?? ''); // CSS selector murni: 'td'
    $index = ((int)($m[2] ?? 1)) - 1; // :text(2) → index 1 (0-based)

    // Filter elemen dengan CSS selector asli
    $element = $node->filter($real);
    if (!$element->count()) return '';

    // Pecah konten elemen menjadi bagian-bagian teks (pakai extractTextParts)
    // yang sudah menangani , child elements, newline, dll
    $parts = $this->extractTextParts($element);

    // Ambil baris ke-n

```

```

$text = $parts[$index] ?? '';

// Dalam teks baris tersebut, cari dan ekstrak nilai setelah label
return $this->extractAfterLabel($text, $label);
// misal: "HP: 081234567" dengan label "HP" → "081234567"
}

```

Kode `resolveContainsFromElements()` — Case B: Base CSS Selector Biasa

Kode ini adalah helper (digunakan pada fungsi `resolveContains()`)

```

protected function resolveContainsFromElements(Crawler $node, string $base, string
$label, string $tail)
{
    // Contoh input A (ada tail):
    //   $base  = 'div.row'
    //   $label = 'Email'
    //   $tail  = '.col-sm-9'
    //
    // Contoh input B (tanpa tail):
    //   $base  = 'p.info'
    //   $label = 'Telepon'
    //   $tail  = ''

    $value = ''; // Nilai yang akan dikembalikan, default kosong

    // Filter semua elemen yang cocok dengan $base selector
    $node->filter($base)->each(function ($el) use ($label, $tail, &$value) {

        // Periksa apakah elemen ini mengandung teks label (case-insensitive)
        // strpos() → false jika tidak ditemukan, int >= 0 jika ditemukan
        if (strpos($el->text(), $label) === false) return; // Tidak ada label?

        Skip

        // Label ditemukan di elemen ini. Sekarang tentukan cara ambil nilainya:

        if ($tail) {
            // — Ada tail selector ——————
            // Cari child element sesuai $tail di dalam elemen yang mengandung
            label
            // Contoh: di dalam yang ada kata "Email", ambil teks dari .col-sm-9
            $target = $el->filter($tail);
            if ($target->count()) {
                $value = trim($target->text()); // Ambil teks dari child tersebut
            }
            // Jika child $tail tidak ditemukan, $value tetap '' atau nilai
            sebelumnya
        } else {
            // — Tidak ada tail, ekstrak dari teks elemen langsung ——————
            // Gunakan extractAfterLabel untuk ambil teks setelah "label: ..."
            // Contoh: "Telepon: 021-123" → "021-123"
        }
    });
}

```

```

    $value = $this->extractAfterLabel($el->text(), $label);
}

// CATATAN: Loop .each() tidak berhenti walau sudah menemukan nilai.
// Jika ada lebih dari 1 elemen yang mengandung label yang sama,
// $value akan di-overwrite oleh elemen terakhir yang cocok.
});

return $value;
}

```

Perbandingan Case A vs Case B secara visual:

CASE A – 'td :text(2) :contains("HP")'

HTML: <td>

Nama: PT. ABC	← baris 1 (:text(1))
HP: 081234567	← baris 2 (:text(2))
Email: a@b.com	← baris 3

</td>

Langkah: filter('td') → extractTextParts → ambil index 1
→ "HP: 081234567" → extractAfterLabel("HP") → "081234567"

CASE B dengan tail – 'div.row:contains("Email") .col-sm-9'

HTML: <div class="row"><div class="col">Nama</div><div class="col-sm-9">PT.
X</div></div>
<div class="row"><div class="col">Email</div><div class="col-sm-9">x@x.com</div></div>

Langkah: filter('div.row') → cek tiap elemen → yang ada "Email"
→ filter('.col-sm-9') di dalamnya → trim text → "x@x.com"

CASE B tanpa tail – 'p.info:contains("Telepon")'

HTML: <p class="info">Telepon: 021-999</p>

Langkah: filter('p.info') → cek ada "Telepon" → extractAfterLabel → "021-999"

Fungsi `extractTextParts()` — Penjelasan Inline Kode Lengkap

Fungsi ini bertugas memecah konten satu elemen HTML menjadi **array bagian-bagian teks**. Hasilnya dipakai oleh `:text(n)` untuk mengambil bagian ke-n. Tantangannya: struktur HTML di dunia nyata sangat beragam — ada yang pakai `
`, ada yang pakai `<p>`, ada yang pakai newline biasa. Fungsi ini menangani semua kemungkinan itu lewat **5 strategi berurutan**.

Kode Lengkap dengan Penjelasan Inline

```
protected function extractTextParts(Crawler $crawler): array
{
    // Guard: jika elemen tidak ditemukan sama sekali → return array kosong
    if (!$crawler->count()) return [];

    // Ambil DOM node pertama (object DOMNode native PHP)
    // Dipakai di Strategy 3 untuk akses langsung ke childNodes
    $node = $crawler->getNode(0);

    $texts = [];

    // Ambil HTML inner dari elemen (bukan teks, tapi HTML mentah termasuk tag)
    // Dipakai di Strategy 1 untuk mendeteksi keberadaan <br> atau <hr>
    $html = $crawler->html();

    // =====
    // STRATEGY 1 – Split by <br> atau <hr>
    // =====
    // Kondisi: HTML mengandung tag <br> atau <hr> (dalam bentuk apapun)
    // Ini adalah pola paling umum untuk konten multi-baris di HTML lama/tabel
    //
    // Contoh HTML yang cocok:
    //   <td>PT. ABC<br>Jl. Merdeka 1<br>Telp. 021-111</td>
    //   <td>Nama<br/>Alamat<hr>Kota</td>
    //   <p>Baris satu<BR>Baris dua</p>
    if (preg_match('/<(br|hr)\b/i', $html)) {

        // Pecah HTML mentah berdasarkan tag <br> atau <hr> (semua variannya)
        // /<(:br|hr)\b[^>]*>/i mencocokkan:
        //   <br>  <br/>  <br />  <BR>  <hr>  <HR class="x"> dll
        $parts = preg_split('/<(:br|hr)\b[^>]*>/i', $html);

        // Bersihkan setiap bagian:
        // strip_tags() → hapus semua tag HTML yang tersisa (misal <span> dalam
<td>)
        // trim()      → hapus whitespace di awal/akhir
        // array_filter() → buang elemen kosong (string "" atau "0")
        // array_values() → reindex array agar mulai dari 0
        $clean = array_values(array_filter(
            array_map(fn($p) => trim(strip_tags($p)), $parts)
        ));

        // Jika berhasil menghasilkan setidaknya 1 bagian → return, selesai
        // Tidak perlu lanjut ke strategy berikutnya
        if ($clean) return $clean;

        // Jika $clean kosong (semua bagian ternyata kosong setelah dibersihkan)
        // → lanjut ke Strategy 2
    }
}
```

```
// =====
// STRATEGY 2 – Child block elements
// =====
// Kondisi: elemen punya lebih dari 1 child element langsung
// Cocok untuk struktur seperti:
//   <td><p>Baris 1</p><p>Baris 2</p><p>Baris 3</p></td>
//   <div><span>A</span><span>B</span><span>C</span></div>
//
// Mengapa harus > 1? Karena jika hanya 1 child, berarti seluruh konten
// ada dalam satu elemen saja – tidak ada yang perlu dipecah
$children = $crawler->children();

if ($children->count() > 1) {
    $children->each(function ($child) use (&$texts) {
        // Ambil teks dari setiap child element
        $t = trim($child->text());

        // Hanya tambahkan jika teks tidak kosong
        if ($t !== '') $texts[] = $t;
    });

    // Validasi: harus menghasilkan lebih dari 1 bagian
    // Jika hanya 1 (semua child ternyata kosong kecuali satu) → tidak
    // berguna,
    // lanjut ke strategy berikutnya
    if (count($texts) > 1) return $texts;
}

// =====
// STRATEGY 3 – DOM text nodes
// =====
// Kondisi: konten elemen adalah campuran teks dan elemen inline
// Cocok untuk struktur seperti:
//   <td>Teks langsung <strong>bold</strong> lanjutan teks</td>
//   <td>
//     Baris satu
//     Baris dua
//   </td>
//
// Menggunakan DOMNode native PHP (bukan Crawler) untuk akses ke text node
// yang tidak terdeteksi oleh children() Symfony
$texts = []; // Reset $texts dari Strategy 2 yang mungkin sudah terisi
// sebagian

foreach ($node->childNodes as $child) {

    // Hanya proses dua tipe node:
    // XML_TEXT_NODE (tipe 3) → teks murni di antara tag
    // XML_ELEMENT_NODE (tipe 1) → tag HTML (ambil textContent-nya)
    //
    // Node lain seperti XML_COMMENT_NODE (komentar HTML) diabaikan
    if ($child->nodeType === XML_TEXT_NODE || $child->nodeType ===
        XML_ELEMENT_NODE) {
```

```
// normalizeText() membersihkan:  
//   - Unicode tersembunyi (zero-width space, BOM, dll)  
//   - Variasi tanda baca Unicode → ASCII  
//   - Spasi berlebih, newline → single space  
//   - HTML entities  
$t = $this->normalizeText($child->textContent);  
  
if ($t !== '') $texts[] = $t;  
}  
}  
  
// Validasi: harus lebih dari 1 bagian agar bermakna sebagai "split"  
if (count($texts) > 1) return $texts;  
  
=====  
// STRATEGY 4 – Newline split  
=====  
// Kondisi: teks mengandung newline (\n, \r\n, atau \r) sebagai pemisah baris  
// Cocok untuk konten yang di-render dari template atau data yang sudah  
// mengandung line break secara eksplisit  
//  
// Contoh:  
//  <td>  
//    PT. ABC  
//    Jl. Sudirman  
//    Jakarta  
//  </td>  
  
// normalizeText() dipanggil sebelum split untuk membersihkan  
// karakter Unicode yang bisa mengganggu pendeksiian newline  
$lines = preg_split('/\r\n|\r|\n/', $this->normalizeText($crawler->text()));  
  
// Bersihkan dan filter:  
//  array_map('trim', ...) → trim setiap baris  
//  array_filter(...)      → buang baris kosong  
//  array_values(...)     → reindex  
$lines = array_values(array_filter(array_map('trim', $lines)));  
  
// Validasi: harus lebih dari 1 baris  
if (count($lines) > 1) return $lines;  
  
=====  
// STRATEGY 5 – Fallback: seluruh teks sebagai satu bagian  
=====  
// Sampai di sini berarti semua strategy di atas gagal menghasilkan  
// lebih dari 1 bagian. Artinya elemen ini hanya punya satu konten tunggal.  
//  
// Return sebagai array 1 elemen agar format tetap konsisten  
// (caller selalu mengharapkan array, bukan string)  
//  
// Contoh kasus yang sampai ke sini:  
//  <td>PT. ABC</td>      → ["PT. ABC"]  
//  <span>satu teks saja</span> → ["satu teks saja"]  
$full = $this->normalizeText($crawler->text());
```

```

    // Jika bahkan teks penuh pun kosong → return []
    // Jika ada teks → return sebagai array 1 elemen
    return $full !== '' ? [$full] : [];
}

```

Visualisasi: Strategi Mana yang Dipakai untuk HTML Apa

HTML: <td>PT. ABC
Jl. Merdeka
021-111</td>

- Strategy 1: ada
? → YA
 - split by
 → ["PT. ABC", "Jl. Merdeka", "021-111"]
- RETURN ✓ (tidak lanjut ke strategy berikutnya)

HTML: <td><p>PT. ABC</p><p>Jl. Merdeka</p><p>021-111</p></td>

- Strategy 1: ada
? → TIDAK
- Strategy 2: children > 1? → YA (3 elemen <p>)
 - each child → ["PT. ABC", "Jl. Merdeka", "021-111"]
- RETURN ✓

HTML: <td>PT. ABC Tbk – Jl. Merdeka</td>

- Strategy 1: ada
? → TIDAK
- Strategy 2: children > 1? → TIDAK (hanya 1 child:)
- Strategy 3: DOM text nodes
 - childNodes: ["PT. ABC ", , " – Jl. Merdeka"]
 - normalizeText tiap node → ["PT. ABC Tbk", "Jl. Merdeka"]
 - count > 1? → YA
- RETURN ✓

HTML: <td>

```

    PT. ABC
    Jl. Merdeka
    021-111
  </td>

```

- Strategy 1: ada
? → TIDAK
- Strategy 2: children > 1? → TIDAK (tidak ada child element)
- Strategy 3: DOM text nodes → hanya 1 text node (whitespace digabung)
 - count > 1? → TIDAK
- Strategy 4: split by \n
 - ["PT. ABC", "Jl. Merdeka", "021-111"]
 - count > 1? → YA

└ RETURN ✓

```
HTML: <td>PT. ABC</td>
      |
      └─ Strategy 1: ada <br>? → TIDAK
      └─ Strategy 2: children > 1? → TIDAK
      └─ Strategy 3: DOM text nodes → 1 node, count > 1? → TIDAK
      └─ Strategy 4: split by \n → 1 baris, count > 1? → TIDAK
      └─ Strategy 5: fallback
          normalizeText("PT. ABC") → "PT. ABC"
      └─ RETURN ["PT. ABC"] ✓
```

Tabel Ringkasan 5 Strategi

Strategy	Kondisi Aktif	Teknik	Kasus HTML
1	Ada atau <hr> di HTML	preg_split pada tag /<hr>	Tabel lama, konten legacy
2	Lebih dari 1 child element	Crawler->children()->each()	<p>, , <div> bersusun
3	Ada text node campuran di DOM	DOMNode->childNodes native PHP	Inline element + teks murni
4	Ada newline di teks	preg_split pada \r\n, \r, \n	Template/data dengan line break
5	Semua gagal (fallback)	normalizeText + return 1 elemen	Elemen sederhana satu baris

Normalisasi Teks: normalizeText()

```
protected function normalizeText(string $text): string
{
    // Fungsi ini membersihkan teks dari berbagai "sampah" karakter unicode
    // yang sering muncul di website dan menyebabkan masalah di CSV/database.

    // Decode HTML entities: &amp; → &, &lt; → <, &nbsp; → spasi, dll
    $text = html_entity_decode($text, ENT_QUOTES | ENT_HTML5, 'UTF-8');

    // Unicode normalize ke FORM_C (NFC) – standarisasi representasi karakter
    if (class_exists('Normalizer')) {
        $text = Normalizer::normalize($text, Normalizer::FORM_C);
    }

    // Ganti berbagai varian karakter Unicode ke ASCII standar.
```

```

// Ini penting karena banyak website menggunakan karakter "fancy"
// yang terlihat sama tapi berbeda secara encoding.
$map = [
    // Berbagai macam tanda dash → tanda minus biasa (-)
    "\u{2013}" => "-", // En dash
    "\u{2014}" => "--", // Em dash

    // Berbagai macam tanda kutip → kutip standar
    "\u{2018}" => '"', // Left single quotation mark
    "\u{2019}" => "'", // Right single quotation mark
    "\u{201C}" => '“', // Left double quotation mark
    "\u{201D}" => '”', // Right double quotation mark

    // Non-breaking space dan berbagai spasi unicode → spasi biasa
    "\u{00A0}" => " ", // No-Break Space (yang paling sering muncul)
    "\u{2003}" => " ", // Em Space

    // Zero-width characters (tidak terlihat tapi mengganggu hash/matching)
    "\u{200B}" => "", // Zero Width Space
    "\u{FEFF}" => "", // BOM (Byte Order Mark)
];
$text = strtr($text, $map);

// Hapus karakter kontrol (invisible characters)
$text = preg_replace('/[\p{Cf}\p{Cc}]/u', '', $text);

// Ganti newline dengan spasi (karena scraping mengambil satu baris)
$text = preg_replace("/\r\n|\r|\n/u", " ", $text);

// Collapse whitespace berganda menjadi satu spasi
// Contoh: "PT. Contoh Jaya" → "PT. Contoh Jaya"
$text = preg_replace('/\s+/u', ' ', $text);

return trim($text); // Hapus spasi di awal dan akhir
}

```

Field Spesial yang Dikenali Otomatis

Parser mengenali beberapa nama field secara khusus tanpa perlu konfigurasi tambahan:

Nama Field	Perilaku Otomatis
detail_url	Mengambil atribut href (bukan teks), dan deduplication otomatis — URL yang sama tidak akan diambil dua kali dalam satu sesi parse
Field yang namanya mengandung kata image	Mengambil atribut src (bukan teks)

Contoh:

```
'fields' => [
  'detail_url' => 'a.btn-detail', // otomatis ambil href, skip duplikat
  'image_produk' => 'img.product-img', // otomatis ambil src
  'image' => 'img.logo', // otomatis ambil src (mengandung
  "image")
]
```

Selector: Urutan Prioritas dan Konflik

Jika selector mengandung lebih dari satu kata kunci khusus, prioritas penanganannya adalah:

1. :group ← paling tinggi, cek ini dulu
2. :contains ← jika tidak ada :group
3. :text(n) ← jika tidak ada :group dan :contains
4. standard ← fallback

Namun :group bisa **dikombinasi** dengan :text(n) dan :contains sekaligus (diproses di dalam resolveGroup). Sementara :contains bisa dikombinasi dengan :text(n) di base selector-nya.

Yang tidak bisa dikombinasi: :contains + :text(n) tanpa :group jika keduanya berada di level yang sama (bukan di dalam :group).

Tabel Ringkasan Semua Selector Khusus

Selector / Config	Lokasi	Contoh	Fungsi
:text(n)	fields	'td :text(2)'	Ambil baris ke-n dari elemen dengan konten campuran
:contains("label")	fields	'div:contains("Email") .val'	Filter elemen berdasarkan teks, ekstrak nilai setelah label
:contains("label") tanpa tail	fields	'p:contains("HP")'	Ekstrak teks setelah label otomatis
:group	fields	'span.tag :group'	Gabungkan semua elemen matching menjadi satu string
:group :text(n)	fields	'span :group :text(2)'	Gabungkan lalu ambil bagian ke-n
:group :contains("x")	fields	'span :group :contains("Email")'	Gabungkan lalu cari label
parent sebagai array	stage	'parent' => ['div.a', 'div.b']	Proses lebih dari satu parent selector
group numeric	stage	'group' => 3	Setiap N elemen sibling = 1 record

Selector / Config	Lokasi	Contoh	Fungsi
group selector range	stage	'group' => ['h3', 'hr']	Kelompokkan dari START selector ke START selector berikutnya
detail_url (nama field)	fields	'detail_url' => 'a[href]'	Ambil href, deduplikasi otomatis, trigger multi-stage
image* (nama field)	fields	'image_cover' => 'img'	Ambil atribut src otomatis

3.7 ClaudeCrawl.php — Otak Utama Scraper

File ini mengkoordinasikan semua komponen. Ia yang menentukan URL mana dikunjungi, kapan berhenti, dan bagaimana data digabungkan.

Fungsi run() — Entry Point

```
public function run($siteKey)
{
    // Ambil konfigurasi website dari WebsiteScraperBaru.php
    // $siteKey misalnya: 'kemenperin' atau 'inaexport'
    $config      = $this->config->sites[$siteKey];
    $stages      = $config['stages'];

    // Muat checkpoint terakhir (progress dari cron sebelumnya)
    $checkpoint = $this->checkpoint->load($siteKey);

    // Apakah konfigurasi website ini punya pagination?
    $hasPagination = !empty($config['pagination']['selector']);

    // Kumpulkan semua URL halaman yang perlu dikunjungi
    // (termasuk halaman 2, 3, 4... dari pagination)
    $paginationUrls = $this->collectPaginationUrls($config, $checkpoint,
$hasPagination);

    // Masukkan semua URL tersebut ke antrian (stage 0 = halaman list)
    foreach ($paginationUrls as $url) {
        $this->queue->push(0, $url);
    }

    // Proses antrian URL satu per satu
    $results = $this->processQueue($stages, $checkpoint, $hasPagination,
$siteKey);

    return $results;
}
```

Fungsi collectPaginationUrls() — Penjelasan Inline Kode Lengkap

Fungsi ini bertanggung jawab mengumpulkan semua URL halaman pagination **sebelum** scraping konten dimulai. Hasilnya adalah array URL yang langsung di-push ke queue.

```
protected function collectPaginationUrls($config, $checkpoint, $hasPagination)
{
    // Tentukan URL awal:
    // - Jika ada checkpoint['last_url'] → lanjut dari halaman terakhir yang diproses
    // - Jika tidak ada → mulai dari start_url di config
    // Ini yang membuat scraper tidak mengulang dari halaman 1 setiap kali cron jalan
    $startUrl = $checkpoint['last_url'] ?? $config['start_url'];

    // Inisialisasi array URL dengan startUrl sebagai elemen pertama
    // Array ini yang akan terus bertambah saat pagination ditemukan
    $urls = [$startUrl];

    // Jika site tidak punya pagination → langsung return array dengan 1 URL saja
    if (!$hasPagination) {
        $this->lastUrlProcessed = $startUrl;
        return $urls;
    }

    // — INISIALISASI STATE DARI CHECKPOINT ——————
    // maxPageFound = nomor halaman terbesar yang pernah ditemukan sebelumnya
    // Ini jadi "batas bawah" – hanya halaman dengan nomor LEBIH BESAR dari ini
    // yang akan diambil di eksekusi ini
    $this->maxPageFound = $checkpoint['last_max_page_seen'] ?? 0;

    // isFirstFetch = true jika belum pernah ada pagination yang ditemukan sebelumnya
    // (maxPageFound masih 0 = eksekusi pertama kali)
    // Dipakai untuk logika penanganan ellipsis "..."
    $this->isFirstFetch = ($this->maxPageFound == 0);

    $currentUrl = $startUrl;

    // Batas maksimal iterasi dari config: 'max_page'
    // Ini BUKAN jumlah total halaman, tapi berapa kali boleh "loncat" ke URL berikutnya
    // dalam satu eksekusi untuk mencari URL pagination baru
    $maxIterations = $config['pagination']['max_page'];
    $iteration = 0;

    // — LOOP DISCOVERY PAGINATION ——————
    while ($iteration < $maxIterations) {

        // Fetch HTML dari currentUrl untuk membaca link pagination di dalamnya
        $html = $this->fetcher->get($currentUrl);
```

```
// Jeda singkat setelah setiap fetch (0.2-0.6 detik, berbeda dari rate->wait())
$this->rateLimit();

// Jika fetch gagal / HTML kosong → hentikan discovery
if (!$html) break;

// Cari semua link pagination di halaman ini
// menggunakan CSS selector dari config['pagination']['selector']
$pageData = $this->detectPaginationLinks(
    $html,
    $config['pagination']['selector']
);

// detectPaginationLinks return:
// [
//   'links' => ['/page=4', '/page=5', '/page=6'], // href yang ditemukan
    // 'maxPage' => 6                                // angka terbesar yang ditemukan
]
// Jika tidak ada link baru yang lebih besar dari maxPageFound → array links kosong

// Tidak ada halaman baru yang ditemukan → pagination sudah habis
if (empty($pageData['links'])) {
    break;
}

//Tambahkan URL baru ke $urls, dengan konversi relatif → absolut
foreach ($pageData['links'] as $link) {
    if ($link !== null) {
        // makeAbsoluteUrl() menangani 3 kasus:
        // 1. URL sudah absolut (http/https) → pastikan pakai https
        // 2. Path absolut (/page=2)           → tambahkan domain dari startUrl
        // 3. Path relatif (page=2)          → tambahkan domain + slash
        $absoluteUrl = $this->makeAbsoluteUrl($link, $startUrl);

        // Cegah URL duplikat masuk ke array
        if (!in_array($absoluteUrl, $urls)) {
            $urls[] = $absoluteUrl;
        }
    }
}

// Update maxPageFound dengan angka terbesar yang ditemukan di iterasi ini
// max() memastikan nilai tidak pernah turun dari yang sudah ada sebelumnya
$this->maxPageFound = max($pageData['maxPage'], $this->maxPageFound);

// Ambil URL terakhir yang baru saja ditambahkan → jadikan currentUrl berikutnya
// Logika: kita "loncat" ke halaman terjauh untuk mencari link pagination
```

```

berikutnya
$nextUrl = end($urls);

// Jika URL terjauh sama dengan currentUrl → tidak ada kemajuan → stop
if ($nextUrl === $currentUrl) break;

$currentUrl = $nextUrl;

// Catat URL terakhir yang diproses untuk disimpan ke checkpoint nantinya
$this->lastUrlProcessed = $nextUrl;

$iteration++;
}

// Return semua URL yang berhasil dikumpulkan
// Contoh hasil: ['https://contoh.com/page=3', 'https://contoh.com/page=4',
...]
// (dimulai dari last_url checkpoint, bukan dari halaman 1)
return $urls;
}

```

Kode `detectPaginationLinks()` — Dipanggil di Dalam Loop

```

protected function detectPaginationLinks($html, $selector)
{
    $crawler = new Crawler($html);

    // Nilai awal maxPage diambil dari state saat ini (bukan 0)
    // agar tidak pernah lebih kecil dari yang sudah pernah ditemukan
    $result = [
        'maxPage' => $this->maxPageFound,
        'links'     => []
    ];

    // Flag untuk menghentikan iterasi .each() lebih awal
    // (Symfony Crawler tidak punya break native, jadi pakai flag)
    $shouldStop = false;

    $crawler->filter($selector)->each(function (Crawler $node) use (&$result,
    &$shouldStop) {

        // Jika sudah ada sinyal stop dari iterasi sebelumnya → skip semua elemen
        berikutnya
        if ($shouldStop) return;

        $text = trim($node->text()); // Teks yang terlihat user: "3", "Next",
        "...", dll
        $href = $node->attr('href'); // Nilai atribut href dari elemen

        // — FILTER 1: Skip anchor link —
    });
}

```

```
// isAnchorLink() → true jika href dimulai dengan '#' atau diakhiri '#'
// Anchor link adalah navigasi dalam halaman yang sama, bukan halaman lain
if ($this->isAnchorLink($href)) {
    return; // Skip, lanjut ke elemen berikutnya
}

// — EKSTRAK NOMOR HALAMAN dari teks —————
// extractPageNumber() pakai regex: /^[1-9][0-9]*(?:\D{2,3})?$/
// Cocok: "5", "12", "100"
// Tidak cocok: "Next", "»", "Berikutnya", "..." → return 0
$pageNumber = $this->extractPageNumber($text);

// — FILTER 2: Tangani Ellipsis ..." —————
// isEllipsis() → true jika teks adalah "..." atau mengandung "..."
if ($this->isEllipsis($text)) {

    if ($this->isFirstFetch) {
        // EKSEKUSI PERTAMA: Langsung stop saat pertama kali ketemu "..."
        // Alasan: kita tidak tahu berapa halaman yang ada, ambil dulu
yang kelihatan
        // Contoh pagination: 1 2 3 [...] 50
        //                                ↑ STOP di sini
        $shouldStop = true;
        return;
    }

    // EKSEKUSI BERIKUTNYA: Perilaku berbeda tergantung kondisi
    if (!empty($result['links'])) {
        // Sudah ada link yang dikumpulkan sebelum "..." ini
        // → Stop, jangan ambil halaman yang ada di balik "..."
        // Contoh: 1 2 [...] 4 5 6 [...] 50
        //                                ↑sudah diambil   ↑ STOP di sini
        $shouldStop = true;
        return;
    }

    // Belum ada link yang dikumpulkan dan ini bukan fetch pertama
    // → "..." ini ada di awal (halaman-halaman sebelumnya sudah pernah
diamambil)
    // → Skip "..." dan lanjut cari link di sebelah kanannya
    // Contoh: 1 2 [...] 4 5 6
    //                                ↑ skip, lanjut ke 4, 5, 6
    return;
}

// — FILTER 3: Hanya ambil halaman yang LEBIH BESAR dari maxPageFound —
// Ini mencegah scraper mengunjungi ulang halaman yang sudah pernah
diproses
if ($pageNumber > $this->maxPageFound) {

    // Update maxPage jika angka ini lebih besar dari yang sebelumnya
    if ($pageNumber > $result['maxPage']) {
        $result['maxPage'] = $pageNumber;
    }
}
```

```

        // Tambahkan href ke daftar link yang akan diproses
        $result['links'][] = $href;
    }
    // Jika pageNumber <= maxPageFound → skip (sudah pernah diambil)
});

return $result;
}

```

Helper yang Dipakai di Kedua Fungsi

```

protected function isAnchorLink($href)
{
    // True jika href dimulai dengan '#' → link ke bagian dalam halaman yang sama
    // True jika href diakhiri '#' → sama, tapi ditulis di akhir URL
    return (strpos($href, '#') === 0 || substr($href, -1) === '#');
}

protected function extractPageNumber($text)
{
    // Regex mencocokkan teks yang:
    // - Dimulai dengan angka 1-9 (bukan 0)
    // - Dikuti angka opsional lainnya
    // - Boleh diikuti 2-3 karakter non-digit di akhir (toleransi untuk "5." atau
    "5 ")
    if (preg_match('/^([1-9][0-9]*)((?:\D{2,3})?)$/', $text, $matches)) {
        return (int)$matches[1];
    }
    // Tidak cocok → return 0 (bukan angka halaman)
    return 0;
}

protected function isEllipsis($text)
{
    // True jika teks HANYA terdiri dari 3+ titik: "...", "....", dst
    // True jika teks MENGANDUNG "..." di mana saja
    return (preg_match('/^(?:\.{3,})$/i', $text) || str_contains($text, '...'));
}

protected function makeAbsoluteUrl($url, $baseUrl)
{
    // Kasus 1: Sudah ada protokol http/https → sudah absolut
    if (preg_match('/^https?:\/\/\//', $url)) {
        // Normalisasi: paksa semua URL ke HTTPS
        return str_replace('http://', 'https://', $url);
    }

    // Ekstrak domain dari baseUrl
    // Contoh: "https://kemenperin.go.id/direktori?page=1"
}

```

```

//   split → ["https:", "", "kemenperin.go.id", "direktori?page=1"]
//   index 0 = "https:"
//   index 2 = "kemenperin.go.id"
$parts = explode('/', $baseUrl);
$domain = $parts[0] . '//' . $parts[2]; // → "https://kemenperin.go.id"

// Kasus 2: URL dimulai dengan '/' → path absolut dari root domain
if (strpos($url, '/') === 0) {
    return $domain . $url; // "https://kemenperin.go.id" + "/direktori?page=2"
}

// Kasus 3: URL relatif tanpa slash di depan
return $domain . '/' . $url; // "https://kemenperin.go.id/" + "page=2"
}

protected function rateLimit()
{
    // Jeda lokal khusus untuk fase discovery pagination
    // Lebih pendek dari $this->rate->wait() yang dipakai saat scraping konten
    // rand(200000, 600000) microsecond = 0.2 - 0.6 detik
    usleep(rand(200000, 600000));
}

```

Visualisasi Alur Lengkap

```

collectPaginationUrls() dipanggil
└── Tentukan startUrl (dari checkpoint atau config)
    └── $urls = [startUrl]

    └── [Tidak ada pagination?] → return [startUrl] langsung

    └── [Ada pagination] → masuk loop:
        └── Iterasi 1: fetch startUrl → detectPaginationLinks()
            └── Eksekusi pertama (maxPageFound=0, isFirstFetch=true):
                ┌─────────────────────────────────────────────────────────────────────────┐
                │ Pagination: 1 2 3 ... 50                                │
                │          ok ok ok STOP                                │
                └─────────────────────────────────────────────────────────────────┘
                ┌─────────────────────────────────────────────────────────────────┐
                │ $urls = [startUrl, /page=2, /page=3]                  │
                │ maxPageFound = 3                                     │
                │ currentUrl = /page=3                                │
                └─────────────────────────────────────────────────────────────────┘

            └── Eksekusi kedua (maxPageFound=3, isFirstFetch=false):
                ┌─────────────────────────────────────────────────────────────────┐
                │ Pagination: 1 2 ... 4 5 6 ... 50                      │
                │          skip skip ↑ ok ok ok STOP                      │
                │          skip (belum ada link dikumpulkan)           │
                └─────────────────────────────────────────────────────────────────┘
                ┌─────────────────────────────────────────────────────────────────┐
                │ $urls = [startUrl, /page=4, /page=5, /page=6]          │
                │ maxPageFound = 6                                       │
                │ currentUrl = /page=6                                  │
                └─────────────────────────────────────────────────────────────────┘

```

```

    └─ Iterasi 2: fetch /page=3 (atau /page=6)
        → Cari link baru yang > maxPageFound
        → Jika ada: tambah ke $urls, update currentUrl
        → Jika tidak ada: break

    └─ Return $urls → semua URL langsung di-push ke UrlQueue

```

Fungsi `processQueue()` — Proses Semua URL di Antrian

Ini adalah **jantung utama scraper**. Semua URL yang sudah dikumpulkan oleh `collectPaginationUrls()` diproses satu per satu di sini — fetch HTML, parse data, deduplikasi, simpan ke DB, cek stop condition, dan simpan checkpoint.

```

protected function processQueue($stages, $checkpoint, $hasPagination, $siteKey)
{
    $results = [];

    // — INISIALISASI STATE DARI CHECKPOINT —————
    // State adalah "memori" antar iterasi dalam loop ini.
    // Nilainya diambil dari checkpoint file (hasil eksekusi sebelumnya).
    // Jika belum ada checkpoint → semua default ke 0 atau null.
    //
    // State ini dipakai untuk:
    // 1. Mendeteksi apakah konten sudah berulang (stop condition)
    // 2. Disimpan kembali ke checkpoint setiap iterasi
    $state = [
        'urlRepeat'      => $checkpoint['url_repeat_count'] ?? 0,      // Berapa
        kali URL yang sama muncul berturut-turut
        'htmlRepeat'     => $checkpoint['html_repeat_count'] ?? 0,      // Berapa
        kali HTML identik (hash sama) berturut-turut
        'rowRepeat'      => $checkpoint['row_repeat_count'] ?? 0,      // Berapa
        kali row terakhir identik berturut-turut
        'lastUrl'        => $checkpoint['last_url']                ?? null, // URL yang
        diproses di iterasi sebelumnya
        'lastHtmlHash'   => $checkpoint['last_html_hash']          ?? null, // MD5 HTML
        dari iterasi sebelumnya
        'lastRowHash'    => $checkpoint['last_row_hash']           ?? null, // MD5 row
        terakhir dari iterasi sebelumnya
        'maxPageSeen'    => $checkpoint['last_max_page_seen'] ?? 0      // Nomor
        halaman terbesar yang pernah terdeteksi
    ];

    // — LOOP UTAMA —————
    // Terus jalan selama masih ada job di dalam UrlQueue
    // Job ditambahkan sebelum loop ini oleh collectPaginationUrls() (stage 0)
    // dan bisa bertambah di tengah loop oleh processRows() (stage 1, 2, dst)
    while ($this->queue->hasJobs()) {

        // Ambil job berikutnya dari depan antrian (FIFO)
        // Struktur job: ['stage' => int, 'url' => string, 'parent' => array]
        $job = $this->queue->pop();
    }
}

```

```
// Tentukan stage mana yang sedang diproses
// stageIndex 0 = halaman list, stageIndex 1 = halaman detail, dst
$stageIndex = $job['stage'];
$stage      = $stages[$stageIndex]; // Ambil config stage dari
WebsiteScraperBaru

// — STEP 1: FETCH HTML —————
$html = $this->fetcher->get($job['url']);

if (!$html) {
    // Fetch gagal (timeout, error server, dll)
    // slowDown() menaikkan delay 1.5x (maks 5000ms) untuk request
berikutnya
    // continue → skip sisa loop, langsung ambil job berikutnya dari queue
    $this->rate->slowDown();
    continue;
}

// Hitung MD5 dari HTML yang baru di-fetch
// Dipakai untuk mendeteksi apakah halaman ini identik dengan halaman
sebelumnya
$htmlHash = md5($html);

// — STEP 2: TRACK REPETISI URL & HTML —————
// Hanya dilakukan jika site punya pagination
// Site tanpa pagination tidak perlu stop condition → tidak perlu tracking
if ($hasPagination) {
    $state = $this->trackRepetitions($state, $job['url'], $htmlHash);
}

// — STEP 3: PARSE HTML → ROWS —————
// parser->parse() membaca HTML dan mengekstrak data berdasarkan
// config 'parent' dan 'fields' dari stage yang sedang aktif
// Return: array of associative array, tiap elemen = 1 record
$rows = $this->parser->parse($html, $stage);

// — STEP 4: TRACK REPETISI ROW —————
// getLastRowHash() mengambil row TERAKHIR dari hasil parse, lalu MD5-nya
// Logika: jika halaman sudah tidak punya data baru, row terakhir
// akan sama terus (data recycle atau halaman sama dikunjungi lagi)
$currentRowHash = $this->getLastRowHash($rows);

if ($hasPagination && $currentRowHash) {
    // Bandingkan hash row terakhir sekarang dengan iterasi sebelumnya
    $state['rowRepeat'] = ($state['lastRowHash'] === $currentRowHash)
        ? $state['rowRepeat'] + 1 // Sama → naikkan counter
        : 0;                      // Berbeda → reset counter ke 0
}

// — STEP 5: PROSES ROWS (STAGE ROUTING) —————
// processRows() melakukan dua hal berbeda tergantung apakah stage punya
'next':
//
```

```

    // Jika ada 'next' (misal stage list yang punya detail):
    //   → Ambil URL dari field 'next', push ke queue sebagai job stage
berikutnya
    //   → $currentJobResults akan KOSONG (data belum final, masih menunggu
detail)
    //
    // Jika tidak ada 'next' (stage terakhir / satu-satunya):
    //   → Merge data parent (dari stage sebelumnya) + data row sekarang
    //   → Masukkan ke $currentJobResults sebagai record final
    //
    // Parameter [] di akhir = inisialisasi $results kosong di dalam
processRows()
$currentJobResults = $this->processRows($rows, $stage, $stageIndex, $job,
[]);

// — STEP 6: AMBIL DATA LAMA DARI DATABASE ——————
// Setiap iterasi selalu baca DB terbaru, bukan variabel lokal.
// Ini penting karena data bisa bertambah dari iterasi-iterasi sebelumnya
// dalam eksekusi yang sama.
$existing = $this->scrapeModel->where('website', $siteKey)->first();
$oldResults = [];

if ($existing && !empty($existing['data'])) {
    // Decode JSON dari kolom 'data' → array PHP
    $oldResults = json_decode($existing['data'], true);
}

// — STEP 7: DEDUPLIKASI ——————
// Bandingkan data baru ($currentJobResults) dengan data lama
($oldResults)
    // Hanya data yang benar-benar baru (hash belum ada) yang ditambahkan
    // Hasil = $oldResults + record baru yang unik
    $finalResults = $this->deduplicateResults(
        $oldResults,
        $currentJobResults,
        $checkpoint['duplicate_count'] ?? [] // Histori duplikat dari
checkpoint
    );

// — STEP 8: SIMPAN KE DATABASE (REAL-TIME) ——————
$jsonData = json_encode($finalResults);

if ($existing) {
    // Record website ini sudah ada di DB → UPDATE kolom 'data' saja
    $this->scrapeModel->update($existing['id'], ['data' => $jsonData]);
} else {
    // Belum ada sama sekali → INSERT baru dengan status 'crawling'
    // Status 'crawling' menandakan proses belum selesai
    $this->scrapeModel->insert([
        'website' => $siteKey,
        'data'     => $jsonData,
        'status'   => 'crawling'
    ]);
}

```

```
// Simpan ke variabel lokal juga, untuk return di akhir fungsi
// $results selalu di-overwrite dengan data terlengkap di setiap iterasi
$results = $finalResults;

// — STEP 9: CEK STOP CONDITION —————
// shouldStop() return true jika SEMUA kondisi ini terpenuhi bersamaan:
//   - urlRepeat >= 3 (URL sama 3x berturut-turut)
//   - htmlRepeat >= 3 (HTML identik 3x berturut-turut)
//   - rowRepeat >= 3 (row terakhir identik 3x berturut-turut)
//   - maxPageFound <= maxPageSeen (tidak ada halaman baru)
if ($hasPagination && $this->shouldStop($state)) {
    // Simpan checkpoint dengan flag isFinished = true
    // lalu BREAK → keluar dari while loop
    $checkpoint = $this->saveCheckpoint(
        $checkpoint, $state, $currentRowHash, $htmlHash,
        $finalResults, $siteKey, $hasPagination,
        true // isFinished = true → pagination_finished: true di JSON
    );
    break;
}

// — STEP 10: SIMPAN CHECKPOINT NORMAL —————
// Disimpan setiap akhir iterasi (selama belum stop)
// Jika cron mati di tengah jalan, eksekusi berikutnya bisa lanjut dari
sini
$checkpoint = $this->saveCheckpoint(
    $checkpoint, $state, $currentRowHash, $htmlHash,
    $finalResults, $siteKey, $hasPagination,
    false // isFinished = false → scraping masih berjalan
);

// — STEP 11: UPDATE STATE UNTUK ITERASI BERIKUTNYA —————
// Nilai "sekarang" menjadi nilai "sebelumnya" untuk iterasi berikutnya
$diproses = [
    $state['lastUrl']      = $job['url'];           // URL yang baru saja diproses
    $state['lastHtmlHash'] = $htmlHash;             // Hash HTML yang baru saja
];
$diproses
    $state['lastRowHash'] = $currentRowHash; // Hash row terakhir yang baru
    saja diparse
    // maxPageSeen diambil nilai terbesar antara yang ada di state vs yang
    ditemukan sekarang
    $state['maxPageSeen'] = max($state['maxPageSeen'], $this->maxPageFound);

    // Jeda sebelum fetch berikutnya (600ms + random 50-300ms)
    // Berbeda dari rateLimit() di collectPaginationUrls yang hanya 200-600ms
    $this->rate->wait();
}

return $results;
}
```

Kode `trackRepetitions()` — Dipanggil di Step 2

```

protected function trackRepetitions($state, $currentUrl, $htmlHash)
{
    // — Cek URL REPEAT ——————
    // Bandingkan URL job sekarang dengan URL yang diproses di iterasi sebelumnya
    $state['urlRepeat'] = ($state['lastUrl'] === $currentUrl)
        ? $state['urlRepeat'] + 1 // URL sama → increment
        : 0;                      // URL beda → reset ke 0

    // — Cek HTML REPEAT ——————
    // Bandingkan MD5 HTML sekarang dengan MD5 HTML iterasi sebelumnya
    // Dua halaman berbeda bisa punya HTML identik jika server return konten yang
    // sama
    // (misal: halaman terakhir pagination yang dikunjungi berulang)
    $state['htmlRepeat'] = ($state['lastHtmlHash'] === $htmlHash)
        ? $state['htmlRepeat'] + 1
        : 0;

    // state['lastUrl'] dan state['lastHtmlHash'] BELUM diupdate di sini
    // Update dilakukan di Step 11 (akhir iterasi loop utama)
    // Ini penting: trackRepetitions() hanya BACA, bukan TULIS ke
    lastUrl/lastHtmlHash

    return $state;
}

```

Kode `getLastRowHash()` — Dipanggil di Step 4

```

protected function getLastRowHash($rows)
{
    // Jika parse tidak menghasilkan apa-apa → return null
    // null tidak akan memicu perubahan rowRepeat di loop utama
    if (empty($rows)) return null;

    // Ambil elemen TERAKHIR dari array $rows
    // end() memindahkan internal pointer ke elemen terakhir dan return nilainya
    $lastRow = end($rows);

    // Encode row terakhir ke JSON lalu hash dengan MD5
    // JSON dipakai agar seluruh isi array (semua field) ikut di-hash,
    // bukan hanya satu field tertentu
    return md5(json_encode($lastRow));
}

```

Mengapa hanya row TERAKHIR, bukan semua row?

Karena row terakhir adalah indikator paling sensitif terhadap "halaman yang sama dikunjungi lagi". Jika scraper terjebak di halaman yang sama, semua row akan sama persis — termasuk row terakhir. Menghash seluruh halaman lebih mahal secara komputasi dan tidak memberikan manfaat tambahan untuk deteksi ini.

Kode `processRows()` — Dipanggil di Step 5

```
protected function processRows($rows, $stage, $stageIndex, $job, $results)
{
    foreach ($rows as $row) {

        // — APAKAH STAGEINI PUNYA 'next'? ——————
        // 'next' ada di config stage jika ini bukan stage terakhir
        // Contoh: stage 0 (list) punya 'next' => 'detail_url'
        //          stage 1 (detail) tidak punya 'next'
        if (!empty($stage['next'])) {

            $nextField = $stage['next']; // Nama field yang berisi URL → misal
            'detail_url'

            if (!empty($row[$nextField])) {
                // Push job baru ke queue untuk stage BERIKUTNYA
                $this->queue->push(
                    $stageIndex + 1, // stage 0 → push sebagai stage 1

                    $row[$nextField], // URL yang diambil dari field 'detail_url'

                    // 'parent' = gabungan data parent job sekarang + data row ini
                    // Data ini dibawa ke stage berikutnya dan di-merge dengan
                    hasil detail
                    // Contoh: job stage 0 tidak punya parent → [] + row stage 0
                    //          job stage 1 punya parent dari stage 0
                    array_merge($job['parent'] ?? [], $row)
                );
            }
            // Jika $row[$nextField] kosong (URL tidak ditemukan) → skip row ini
        } else {
            // — STAGE TERAKHIR / TIDAK ADA 'next' ——————
            // Ini adalah data final yang siap disimpan

            // Merge data parent (dari stage sebelumnya) dengan data row sekarang
            // Contoh multi-stage:
            //   $job['parent'] = ['detail_url' => 'https://...'] ← dari stage 0
            //   $row           = ['nama' => 'PT. X', 'email' => 'x@x.com'] ←
            dari stage 1
            //   hasil merge    = ['detail_url' => '...', 'nama' => 'PT. X',
            'email' => '...']
            //
            // Contoh single-stage:
            //   $job['parent'] = [] (kosong, tidak ada parent)
            //   $row           = ['nama' => 'PT. X', 'alamat' => 'Jl. A']
        }
    }
}
```

```

        // hasil merge = ['nama' => 'PT. X', 'alamat' => 'Jl. A']
        $results[] = array_merge($job['parent'] ?? [], $row);
    }

}

return $results;
// Catatan: jika stage punya 'next', $results selalu KOSONG karena
// semua row hanya di-push ke queue, tidak ke $results
// Data finalnya baru muncul ketika stage berikutnya diproses
}

```

Kode `deduplicateResults()` — Dipanggil di Step 7

```

protected function deduplicateResults($oldResults, $newResults, $duplicateCount)
{
    $hashIndex = []; // Lookup table: hash → true/false
    $finalResults = $oldResults; // Mulai dari data lama, data baru akan
    ditambahkan

    // — FASE 1: INDEX SEMUA DATA LAMA ——————
    // Buat hash untuk setiap record yang sudah ada di DB
    // Hash ini jadi "sidik jari" unik setiap record
    foreach ($oldResults as $row) {

        ksort($row);
        // ksort() mengurutkan key array secara alfabetis
        // Kenapa perlu? Karena json_encode(['b'=>1, 'a'=>2]) ≠
        json_encode(['a'=>2, 'b'=>1])
        // Tanpa ksort, dua record yang SAMA bisa punya hash BERBEDA hanya karena
        urutan key

        $normalizedRow = array_map(function($v) {
            return is_string($v) ? trim($v) : $v;
        }, $row);
        // array_map + trim: bersihkan spasi di awal/akhir semua string
        // "PT. ABC" dan "PT. ABC" seharusnya dianggap sama → trim()
        menyamakannya
        // Non-string (int, null) dibiarkan apa adanya

        $hashIndex[md5(json_encode($normalizedRow))] = true;
        // Simpan hash ke lookup table
        // Menggunakan hash sebagai KEY (bukan value) → lookup O(1), sangat cepat
    }

    // — FASE 2: FILTER DATA BARU ——————
    foreach ($newResults as $row) {

        // Normalisasi yang sama persis seperti data lama
        // Wajib konsisten agar hash bisa dibandingkan dengan akurat
        ksort($row);
    }
}

```

```

$normalizedNewRow = array_map(function($v) {
    return is_string($v) ? trim($v) : $v;
}, $row);

$hash = md5(json_encode($normalizedNewRow));

if (!isset($hashIndex[$hash])) {
    // Hash belum ada di index → ini record baru yang unik
    $finalResults[] = $row;           // Tambahkan ke hasil (row
ORIGINAL, bukan normalized)
    $hashIndex[$hash] = true;         // Daftarkan hash agar tidak
masuk lagi
    $duplicateCount[$hash] = 0;       // Inisialisasi counter duplikat
untuk record ini
} else {
    // Hash sudah ada → ini duplikat
    // Increment counter duplikat (untuk tracking/debugging)
    $duplicateCount[$hash] = ($duplicateCount[$hash] ?? 0) + 1;
    // Record ini TIDAK ditambahkan ke $finalResults
}
}

return $finalResults;
// Catatan: $duplicateCount tidak di-return dan tidak disimpan ke checkpoint
// di versi kode ini – hanya dipakai secara internal untuk tracking
}

```

Kode `shouldStop()` — Dipanggil di Step 9

```

protected function shouldStop($state)
{
    // Cek apakah tidak ada halaman baru yang ditemukan
    // maxPageFound = halaman terbesar yang terdeteksi di pagination saat ini
    // maxPageSeen = halaman terbesar yang pernah dilihat (dari checkpoint
sebelumnya)
    // Jika maxPageFound <= maxPageSeen → tidak ada halaman baru → pagination
stagnan
    $pageStagnant = ($this->maxPageFound <= $state['maxPageSeen']);

    // SEMUA 4 kondisi harus true secara bersamaan untuk stop
    // Satu saja false → scraper terus jalan
    return (
        $state['urlRepeat'] >= 3 && // URL yang sama dikunjungi 3x berturut-
turut
        $state['htmlRepeat'] >= 3 && // HTML identik 3x berturut-turut
        $state['rowRepeat'] >= 3 && // Row terakhir identik 3x berturut-turut
        $pageStagnant           // Tidak ada halaman baru yang belum pernah
dilihat
    );
}

```

```
// Mengapa threshold 3, bukan 1 atau 2?  
// Nilai 1 terlalu agresif → bisa stop prematur karena anomali sesaat  
// Nilai 3 memberi toleransi: scraper akan coba 3 kali dulu sebelum menyerah  
}
```

Kode `saveCheckpoint()` — Dipanggil di Step 9 & 10

```
protected function saveCheckpoint($checkpoint, $state, $rowHash, $htmlHash,  
$finalResults, $siteKey, $hasPagination, $isFinished)  
{  
    // Bangun ulang array checkpoint dari state terkini  
    // Setiap key di sini adalah field yang akan tersimpan di JSON file  
    $checkpoint = [  
  
        // URL terakhir yang diproses:  
        // Prioritas ke $this->lastUrlProcessed (URL terakhir dari fase discovery  
        // pagination)  
        // Fallback ke checkpoint['last_url'] yang sudah ada jika lastUrlProcessed  
        null  
        'last_url' => $this->lastUrlProcessed ?? $checkpoint['last_url'],  
  
        // Simpan semua counter repetisi ke checkpoint  
        // Dipakai di eksekusi berikutnya untuk melanjutkan tracking  
        'url_repeat_count' => $state['urlRepeat'],  
        'html_repeat_count' => $state['htmlRepeat'],  
        'row_repeat_count' => $state['rowRepeat'],  
  
        // Simpan hash terakhir sebagai referensi untuk iterasi berikutnya  
        'last_html_hash' => $htmlHash,  
        'last_row_hash' => $rowHash,  
  
        // Simpan nomor halaman terbesar yang pernah ditemukan  
        // max() memastikan nilainya tidak pernah turun  
        'last_max_page_seen' => max($state['maxPageSeen'], $this->maxPageFound),  
  
        // Flag apakah pagination sudah selesai sepenuhnya  
        // true → dikirim dari shouldStop() → eksekusi berikutnya tahu sudah  
        // habis  
        // false → masih ada kemungkinan halaman baru di eksekusi berikutnya  
        'pagination_finished' => $isFinished,  
  
        // Simpan apakah site ini punya pagination (dari config)  
        'paginate' => $hasPagination,  
    ];  
  
    // Tulis ke file JSON: writable/crawler_checkpoint_{siteKey}.json  
    $this->checkpoint->save($checkpoint, $siteKey);  
  
    // Return checkpoint yang baru saja disimpan  
    // Dipakai untuk update variabel $checkpoint di loop utama
```

```

    return $checkpoint;
}

```

Visualisasi Alur Satu Iterasi Loop

```

queue->pop() → job: {stage:0, url:'https://x.com/page=4', parent:[]}

— STEP 1: fetcher->get(url)
  └─ Sukses → $html = "<html>...</html>", $htmlHash = "a1b2c3..."
    └─ Gagal → rate->slowDown(), continue (skip sisa step)

— STEP 2: trackRepetitions()
  └─ lastUrl == currentUrl? → urlRepeat++ atau reset ke 0
    └─ lastHtmlHash == htmlHash? → htmlRepeat++ atau reset ke 0

— STEP 3: parser->parse($html, $stage)
  └─ $rows = [{nama:'PT.A', detail_url:'https://x.com/1'}, ...]

— STEP 4: getLastRowHash($rows)
  └─ lastRowHash == currentRowHash? → rowRepeat++ atau reset ke 0
    └─ $currentRowHash = "d4e5f6..."

— STEP 5: processRows()
  └─ stage punya 'next' (stage list):
    → queue->push(stage:1, url:'https://x.com/1', parent:[...row])
    → $currentJobResults = [] (kosong)
  └─ stage tidak punya 'next' (stage detail):
    → $currentJobResults = [{nama:'PT.A', alamat:'Jl.B', ...}]

— STEP 6: scrapeModel->where('website', siteKey)->first()
  └─ $oldResults = [record lama dari DB...]

— STEP 7: deduplicateResults($oldResults, $currentJobResults)
  └─ Index semua oldResults → hashIndex
  └─ Cek setiap currentJobResults:
    └─ Hash baru? → tambah ke finalResults
      └─ Hash lama? → skip (duplikat)
    └─ $finalResults = oldResults + record baru yang unik

— STEP 8: scrapeModel->update() atau insert()
  └─ DB diupdate dengan $finalResults terbaru

— STEP 9: shouldStop()?
  └─ Ya (urlRepeat≥3 & htmlRepeat≥3 & rowRepeat≥3 & pageStagnant)
    → saveCheckpoint(isFinished=true) → BREAK
  └─ Tidak → lanjut

— STEP 10: saveCheckpoint(isFinished=false)
  └─ writable/crawler_checkpoint_siteKey.json diperbarui

```

```

└─ STEP 11: update $state
    └ lastUrl, lastHtmlHash, lastRowHash, maxPageSeen ← nilai iterasi ini

└ rate->wait() → jeda 600ms + random 50-300ms
    └ kembali ke atas while loop → pop job berikutnya

```

3.8 Scraper.php — Command CLI untuk Cron Job

File ini adalah "pintu masuk" yang dipanggil oleh cron job. Ia mengatur lock file (agar tidak ada dua proses bersamaan) dan memanggil ClaudeCrawl.

```

<?php
namespace App\Commands;

class Scraper extends BaseCommand
{
    // Metadata command – digunakan oleh CI4 untuk mendaftarkan command
    protected $group      = 'App';
    protected $name        = 'scraper:run';           // Nama command di CLI
    protected $description = 'Run scraping process.';
    protected $usage       = 'scraper:run siteKey'; // Cara pakai

    public function run(array $params)
    {
        // Ambil argument pertama dari CLI.
        // Contoh: "php spark scraper:run kemenperin" → $siteKey = 'kemenperin'
        $siteKey = $params[0] ?? null;

        if (!$siteKey) {
            CLI::error('Error: siteKey argument is required.');
            return;
        }

        // —— LOCK FILE: Cegah dua proses berjalan bersamaan ———
        // Bayangkan lock file seperti kartu "Sedang Digunakan" di toilet.
        // Proses pertama memasang kartu, proses kedua melihat kartu dan pergi.

        // Path file lock: /tmp/scraper_kemenperin.lock
        $lockFile = sys_get_temp_dir() . "/scraper_$siteKey.lock";
        $fp = fopen($lockFile, 'c'); // Buka atau buat file lock

        // flock() = kunci file
        // LOCK_EX = exclusive lock (hanya satu proses yang bisa punya lock ini)
        // LOCK_NB = non-blocking (tidak menunggu, langsung return false jika gagal)
        if (!flock($fp, LOCK_EX | LOCK_NB)) {
            // File sudah dikunci oleh proses lain = ada scraping yang berjalan
            echo "Process already running, exiting.\n";
            return; // Keluar tanpa error
        }
    }
}

```

```
// Jika sampai sini: berhasil mendapat lock, lanjut proses

// — CEK CHECKPOINT: Perlu scraping atau tidak? —————
$checkpoint = $this->checkpoint->load($siteKey);

$html_repeat_count    = $checkpoint['html_repeat_count']    ?? 0;
$url_repeat_count     = $checkpoint['url_repeat_count']     ?? 0;
$row_repeat_count     = $checkpoint['row_repeat_count']     ?? 0;
$paginate             = $checkpoint['paginate']             ?? false;
$pagination_finished = $checkpoint['pagination_finished'] ?? false;

// Tentukan kapan harus skip scraping (data sudah tidak berubah)
$shouldPause = $paginate
    // Untuk website DENGAN pagination: tunggu semua halaman selesai dulu
    ? ($pagination_finished && $html_repeat_count >= 3 &&
$url_repeat_count >= 3 && $row_repeat_count >= 3)
    // Untuk website TANPA pagination: langsung cek repeat count
    : ($html_repeat_count >= 3 && $url_repeat_count >= 3 &&
$row_repeat_count >= 3);

if ($shouldPause) {
    echo "Repeated content detected, skipping.\n";
    flock($fp, LOCK_UN); // Lepas lock
    fclose($fp);
    return;
}

// — JALANKAN SCRAPING —————
try {
    $data = $this->scraperNew->run($siteKey);

    // Reconnect database (koneksi bisa timeout jika scraping terlalu
lama)
    db_connect()->reconnect();

    $existing = $this->scrapeModel->where('website', $siteKey)->first();

    if ($existing) {
        // Website sudah ada di database → GABUNGKAN data lama + baru
        $dataAsli = json_decode($existing['data'], true);
        $dataOlah = array_merge($dataAsli, $data);
        $this->scrapeModel->update($existing['id'], [
            'data'      => json_encode($dataOlah),
            'website'   => $siteKey,
            'status'    => 'running'
        ]);
    } else {
        // Website baru → INSERT record baru ke database
        $this->scrapeModel->insert([
            'data'      => json_encode($data),
            'website'   => $siteKey,
            'status'    => 'running'
        ]);
    }
}
```

```

} catch (\Exception $e) {
    CLI::write("Error: " . $e->getMessage());
}

// Update status menjadi 'stop' setelah selesai
$existing = $this->scrapeModel->where('website', $siteKey)->first();
if ($existing) {
    $this->scrapeModel->update($existing['id'], ['status' => 'stop']);
}

// — LEPAS LOCK FILE —
flock($fp, LOCK_UN); // Buka kunci
fclose($fp); // Tutup file
unlink($lockFile); // Hapus file lock (bersih-bersih)
}
}

```

Ilustrasi sistem lock file:

Timeline:

- 00:00 → Cron 1 mulai: buat lock file /tmp/scraping_kemenperin.lock
- 00:30 → Cron 2 mulai: coba buat lock → GAGAL (sudah ada) → langsung keluar
- 00:45 → Cron 1 selesai: hapus lock file
- 01:00 → Cron 3 mulai: buat lock file → BERHASIL → mulai scraping

3.9 ScrapeController.php — Controller Web

Mengatur halaman web: form untuk export CSV dan endpoint untuk upload file HTML.

```

<?php
namespace App\Controllers;

class ScrapeController extends BaseController
{
    /**
     * GET /scraper
     * Halaman utama: tampilkan form pilih website untuk di-export.
     */
    public function index()
    {
        // Ambil semua nama website yang punya data di database
        // select('website') = hanya ambil kolom 'website', tidak perlu kolom lain
        $isExist = $this->scrapeModel->select('website')->findAll();

        // Kirim ke view sebagai variabel $list_website
        // View akan loop ini untuk membuat dropdown option
        return view('scraper_form', ['list_website' => $isExist]);
    }
}

```

```
}

/**
 * POST /scraper/export atau GET /scraper/export/{siteKey}
 * Export data website tertentu sebagai file CSV yang langsung terdownload.
 */
public function export($siteKey = null)
{
    // Ambil nama website dari:
    // - Form POST (ketika submit dari halaman web), ATAU
    // - URL parameter (ketika akses langsung via URL)
    $website = $this->request->getPost('website') ?? $siteKey;

    $isExist = $this->scrapeModel->where('website', $website)->select('data')->first();

    try {
        // Decode JSON string dari database → array PHP
        $data = json_decode($isExist['data'], true);

        if (empty($data)) {
            return redirect()->back()->with('error', 'Tidak ada data!');
        }

        // Nama file yang akan didownload
        // Contoh: kemenperin_2024-01-15_10-30-00.csv
        $filename = $website . '_' . date('Y-m-d_H-i-s') . '.csv';

        // Set header HTTP agar browser mendownload file, bukan menampilkan
        header('Content-Type: text/csv; charset=UTF-8');
        header('Content-Disposition: attachment; filename="' . $filename .
        '"');
        header('Cache-Control: no-store, no-cache');

        // php://output = stream output langsung ke browser (tidak perlu
        simpan ke file dulu)
        $output = fopen('php://output', 'w');

        // BOM (Byte Order Mark) UTF-8: tiga byte khusus di awal file CSV.
        // Ini WAJIB agar Microsoft Excel membaca encoding dengan benar.
        // Tanpa BOM, huruf Indonesia (ä, é, dll) akan tampil rusak di Excel.
        fprintf($output, chr(0xEF) . chr(0xBB) . chr(0xBF));

        // Tulis baris header CSV (nama kolom)
        // Diambil dari key array record pertama
        // Contoh: ['perusahaan', 'alamat', 'telp', 'email']
        $headers = array_keys($data[0]);
        fputcsv($output, $headers);

        // Tulis setiap baris data
        foreach ($data as $row) {
            $cleanRow = [];
            foreach ($headers as $key) {
                // Jika kolom tidak ada di record ini, isi dengan string

```

```

kosong
    $cleanRow[ ] = $row[$key] ?? '';
}
// fputcsv otomatis handle:
// - Menambahkan koma antar kolom
// - Memberi tanda kutip jika nilai mengandung koma
// - Escaping karakter khusus
fputcsv($output, $cleanRow);
}

fclose($output);
exit; // PENTING! Hentikan eksekusi PHP setelah file dikirim

} catch (\Exception $e) {
    return redirect()->back()->with('error', $e->getMessage());
}
}

<**
 * POST /scraper/file
 * Endpoint untuk upload dan proses file HTML secara offline.
 */
public function file()
{
    try {
        // Ambil konfigurasi dari form (dikirim dalam format Base64)
        // Mengapa Base64? Untuk menghindari masalah encoding saat kirim
        // konfigurasi JSON yang kompleks via multipart/form-data.
        $configRaw = $this->request->getPost('data') ?? '';

        if (empty($configRaw)) {
            return $this->response->setJSON([
                'status' => 'error',
                'message' => 'Data kosong.'
            ])->setStatusCode(400);
        }

        // Langkah decode: Base64 → JSON string → array PHP
        $decodedJson = base64_decode($configRaw);
        $configs     = json_decode($decodedJson, true);

        if (json_last_error() !== JSON_ERROR_NONE) {
            throw new \Exception("Gagal decode JSON: " .
json_last_error_msg());
        }
    }

    $files    = $this->request->getFiles();
    $results  = [];

    if (isset($files['html_files'])) {
        foreach ($files['html_files'] as $index => $file) {
            if ($file->isValid() && !$file->hasMoved()) {
                // getTempName() = path file sementara yang diupload PHP
                // File ini ada di /tmp/phpXXXXXX dan akan otomatis

```

```

dihapus
    $html = file_get_contents($file->getTempName());

    // Ambil konfigurasi untuk file ini
    // Jika hanya ada satu config, pakai untuk semua file
    $configPilihan = $configs[$index] ?? $configs[0];

    // Proses HTML dan dapatkan URL untuk download hasil
    $results[] = $this->scraperNew->runFileOnce($html,
$configPilihan);
}
}

return $this->response->setJSON([
    'status' => 'success',
    'message' => 'Data diproses',
    'link' => $results // Array URL download CSV
]);

} catch (\Exception $e) {
    return $this->response->setJSON([
        'status' => 'error',
        'message' => $e->getMessage()
    ])->setStatusCode(500);
}
}
}
}

```

4. Setup Awal

Persyaratan

- PHP 8.0+ dengan ekstensi: curl, mbstring, intl, json
- CodeIgniter 4.x
- Composer
- Database MySQL/MariaDB

Langkah 1: Install Dependensi via Composer

```
composer require guzzlehttp/guzzle symfony/dom-crawler symfony/css-selector
```

Langkah 2: Buat Tabel Database

```

CREATE TABLE scraped_results (
    id      INT AUTO_INCREMENT PRIMARY KEY,
    website VARCHAR(255) NOT NULL,

```

```

    data      LONGTEXT,
    status    VARCHAR(50) DEFAULT 'running',
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP,
    updated_at DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);

```

Langkah 3: Buat Model

Buat file `app/Models/ScrapedResult.php`:

```

<?php
namespace App\Models;
use CodeIgniter\Model;

class ScrapedResult extends Model
{
    protected $table      = 'scraped_results';
    protected $primaryKey = 'id';
    protected $allowedFields = ['website', 'data', 'status'];
    protected $useTimestamps = true;
}

```

Langkah 4: Daftarkan Route

Di `app/Config/Routes.php`:

```

$route->get('scraper',           'ScrapeController::index');
$route->post('scraper/export',   'ScrapeController::export');
$route->get('scraper/export/(:any)', 'ScrapeController::export/$1');
$route->get('scraper/file',       'ScrapeController::indexFile');
$route->post('scraper/file',      'ScrapeController::file');

```

Langkah 5: Permission Folder

```
chmod -R 755 writable/
```

5. Cara Menambah Website Baru

Edit `app/Config/WebsiteScraperBaru.php` dan tambahkan entry baru:

Template 1: Satu Halaman, Tanpa Pagination

```
'nama_unik_website' => [
    'start_url' => 'https://contoh.com/daftar',
    'rate_limit' => [
        'base_delay_ms' => 800,
        'max_delay_ms' => 5000,
        'retry' => 3
    ],
    'stages' => [
        [
            'name' => 'list',
            'parent' => 'div.card-item', // Satu <div class="card-item"> = satu
            record
            'fields' => [
                'nama' => 'h2.title', // Teks dari <h2 class="title">
                'alamat' => 'p.address',
                'telepon' => 'span.phone',
            ]
        ]
    ]
],
],
```

Template 2: Dengan Pagination

```
'nama_unik_website' => [
    'start_url' => 'https://contoh.com/daftar?page=1',
    'pagination' => [
        'selector' => 'ul.pagination li a', // Link angka halaman
        'max_page' => 50 // Maks 50 halaman per eksekusi
    ],
    'rate_limit' => [
        'base_delay_ms' => 800,
        'max_delay_ms' => 5000,
        'retry' => 3
    ],
    'stages' => [
        [
            'name' => 'list',
            'parent' => 'table tbody tr',
            'fields' => [
                'nama' => 'td:first-child',
                'alamat' => 'td:nth-child(2)',
                'telepon' => 'td:nth-child(3)'
            ]
        ]
    ]
],
```

```
        ]
    ],
],
```

Template 3: Multi-Stage (List → Detail)

```
'nama_unik_website' => [
    'start_url' => 'https://contoh.com/daftar',
    'pagination' => [
        'selector' => 'a.page-next',
        'max_page' => 100
    ],
    'rate_limit' => [
        'base_delay_ms' => 1200,
        'max_delay_ms' => 5000,
        'retry' => 3
    ],
    'stages' => [
        // STAGE 0: Ambil link ke halaman detail
        [
            'name' => 'list',
            'parent' => 'div.company-card',
            'fields' => [
                'detail_url' => 'a.btn-detail[href]', // HARUS bernama
                'detail_url'
            ],
            'next' => 'detail_url' // Beritahu scraper untuk follow link ini
        ],
        // STAGE 1: Ambil data dari halaman detail
        [
            'name' => 'detail',
            'parent' => 'div.company-profile',
            'fields' => [
                'nama_perusahaan' => 'h1.company-name',
                'alamat' => 'p.address',
                'email' => 'a[href^="mailto:"]',
                'produk_utama' => '.main-product',
            ]
        ]
    ],
]
```

6. Referensi Selector Khusus

CSS Selector Standar

Selector	HTML yang Cocok	Keterangan
div.card	<div class="card">	Elemen dengan class
#main	<h1 id="main">	Elemen dengan ID
table tbody tr	Baris tabel	Descendant selector
td:first-child	Kolom pertama	Pseudo-class
td:nth-child(2)	Kolom kedua	Pseudo-class dengan angka
a[href]		Atribut selector
a[href^="http"]	Link yang mulai dengan "http"	Prefix atribut

:text(n) — Ambil Baris Teks ke-n

```
// HTML:
// <td>
//   PT. ABC<br>
//   Jl. Merdeka 1<br>
//   Telp. 021-111<br>
//   Email: abc@abc.com
// </td>

'perusahaan' => 'td:nth-child(2) :text(1)', // "PT. ABC"
'alamat'      => 'td:nth-child(2) :text(2)', // "Jl. Merdeka 1"
'telepon'     => 'td:nth-child(2) :text(3)', // "Telp. 021-111"
'email'       => 'td:nth-child(2) :text(4)', // "Email: abc@abc.com"
```

:contains("label") — Filter Elemen Berdasarkan Teks

```
// HTML: banyak <div class="row"> dengan label berbeda
// <div class="row"><div class="col">Telephone</div><div class="col-sm-9">021-123</div></div>
// <div class="row"><div class="col">Email</div><div class="col-sm-9">a@b.com</div></div>

'telepon' => 'div.row:contains("Telephone") .col-sm-9', // Cari row yang ada kata "Telephone"
'email'   => 'div.row:contains("Email") .col-sm-9',      // Cari row yang ada kata "Email"
```

7. Setup Cron Job di cPanel (24/7)

1. Login ke cPanel → menu "Cron Jobs"

2. Isi kolom **Command:**

```
/usr/local/bin/php /home/CPANELUSERNAME/public_html/spark scraper:run kemenperin
>> /home/CPANELUSERNAME/logs/scraping.log 2>&1
```

Keterangan setiap bagian:

/usr/local/bin/php	→ Path PHP di server (tanya support hosting jika berbeda)
/home/USERNAME/public_html/	→ Root folder project CI4
spark	→ File command CI4 (ada di root project)
scraper:run	→ Command yang dijalankan
kemenperin	→ Nama site key
>> .../logs/scraping.log	→ Simpan output ke file log
2>&1	→ Arahkan error ke file log yang sama

3. Atur jadwal:

Jadwal	Minute	Hour	Day	Month	Weekday
Setiap 30 menit	*/30	*	*	*	*
Setiap 1 jam	0	*	*	*	*
Setiap 2 jam	0	*/2	*	*	*

4. Buat cron terpisah untuk setiap website yang ingin di-scrape.

8. Troubleshooting

✗ Scrapper tidak mengambil data

CSS selector salah. Cara debug:

1. Buka website di browser → klik kanan → "Inspect"
2. Coba selector di Console: `document.querySelectorAll('selector-kamu')`
3. Jika hasilnya kosong, selector perlu diperbaiki

✗ Scraping berhenti dan tidak mau lanjut

Sistem mendeteksi konten berulang (normal jika semua data sudah terkumpul). Reset checkpoint:

```
rm /home/USERNAME/public_html/writable/crawler_checkpoint_kemenperin.json
```

✗ Error "Process already running"

Hapus lock file yang tersisa:

```
rm /tmp/scraping_kemenperin.lock
```

✗ Cron job tidak berjalan

Debug dengan jalankan command manual via SSH:

```
/usr/local/bin/php /home/USERNAME/public_html/spark scraping:run kemenperin  
# Cek log  
tail -f /home/USERNAME/logs/scraping.log
```

✗ CSV rusak di Excel

Buka via: Excel → Data → From Text/CSV → pilih encoding **UTF-8**.

❖ Ringkasan Cara Pakai

1. Edit WebsiteScraperBaru.php → tambah konfigurasi website
2. Test manual via SSH:
`php spark scraping:run nama_website`
3. Cek hasilnya di browser:
`http://domain.com/scraping` → Export CSV
4. Setup cron di cPanel:
`*/30 * * * * /usr/local/bin/php .../spark scraping:run nama_website`
5. Sistem berjalan otomatis 24/7!