BAB 4. HASIL DAN PEMBAHASAN

Bab ini membahas hasil penelitian berupa implementasi dan pengujian sistem yang dikembangkan berdasarkan perancangan pada Bab III. Tahap perancangan dilakukan untuk menghasilkan desain sistem yang menjadi acuan dalam proses pengembangan, sedangkan tahap implementasi menjelaskan realisasi rancangan ke dalam bentuk perangkat lunak. Selanjutnya, pengujian dilakukan untuk memastikan bahwa seluruh fungsi dan komponen sistem telah berjalan sesuai dengan kebutuhan yang telah ditetapkan.

Pembahasan pada bab ini disusun secara bertahap, dimulai dari implementasi sistem hingga pengujian sistem menggunakan pendekatan *V-Model*. Dengan demikian, setiap hasil implementasi dapat divalidasi berdasarkan kebutuhan sistem yang telah dirumuskan pada tahap analisis.

4.1 Implementasi Lingkungan Pengembangan

Implementasi sistem merupakan tahap realisasi dari hasil perancangan yang telah dibahas pada bab sebelumnya. Pada tahap ini, seluruh komponen sistem diimplementasikan menjadi sebuah aplikasi yang mampu mendukung proses pengambilan, pengolahan, dan penyajian informasi produk dari platform 1688.com. Implementasi dilakukan dengan mengintegrasikan setiap modul sesuai arsitektur sistem sehingga seluruh proses dapat berjalan secara otomatis.

Sistem yang dikembangkan terdiri atas beberapa komponen utama, yaitu Chrome Extension sebagai media DOM scraping dan interaksi dengan halaman produk, Backend Flask sebagai pusat pengendali proses dan penyedia layanan REST API, modul OCR dan Image Processing untuk mengekstraksi, menerjemahkan, dan mengganti teks pada gambar produk, modul Gemini API untuk menghasilkan nama produk, deskripsi, dan kata kunci secara otomatis, modul Firefly Algorithm untuk mengoptimalkan konfigurasi *pipeline scheduler* pada proses scraping dan pengolahan data, serta Web Dashboard sebagai media pengelolaan dan penyajian informasi produk.

Seluruh komponen tersebut diintegrasikan melalui REST API sehingga proses pertukaran data antar modul dapat berlangsung secara otomatis, terstruktur, dan konsisten.

4.1.1 Implementasi Lingkungan Pengembangan

Implementasi sistem pada penelitian ini dilakukan menggunakan lingkungan pengembangan yang terdiri atas perangkat keras (*hardware*) dan perangkat lunak (*software*). Lingkungan pengembangan tersebut digunakan untuk mendukung proses pembangunan, implementasi, pengujian, dan integrasi seluruh

komponen sistem, meliputi Chrome Extension, Backend Flask, modul OCR dan Image Processing, Gemini API, Firefly Algorithm, serta Web Dashboard. Spesifikasi perangkat keras dan perangkat lunak yang digunakan pada penelitian ini disajikan pada Tabel 4.1 dan Tabel 4.2.

Tabel 4.1 Perangkat Keras

No	Komponen	Spesifikasi
1	Perangkat Utama	Laptop Acer Aspire 3 Spin 14
2	Prosesor (CPU)	Intel® Core™ i3-1315U (6 Core, hingga 4,5 GHz)
3	Memori (RAM)	8 GB LPDDR5
4	Penyimpanan	SSD 256 GB
5	Pengolah Grafis (GPU)	Intel® UHD Graphics (Integrated)
6	Layar	14 inci Touchscreen, resolusi 1920 × 1200 piksel (Full HD+)
7	Sistem Operasi	Windows 11 64-bit
8	Koneksi Internet	Wi-Fi 6 dengan kecepatan minimal 20 Mbps

Tabel 4.2 Perangkat Lunak

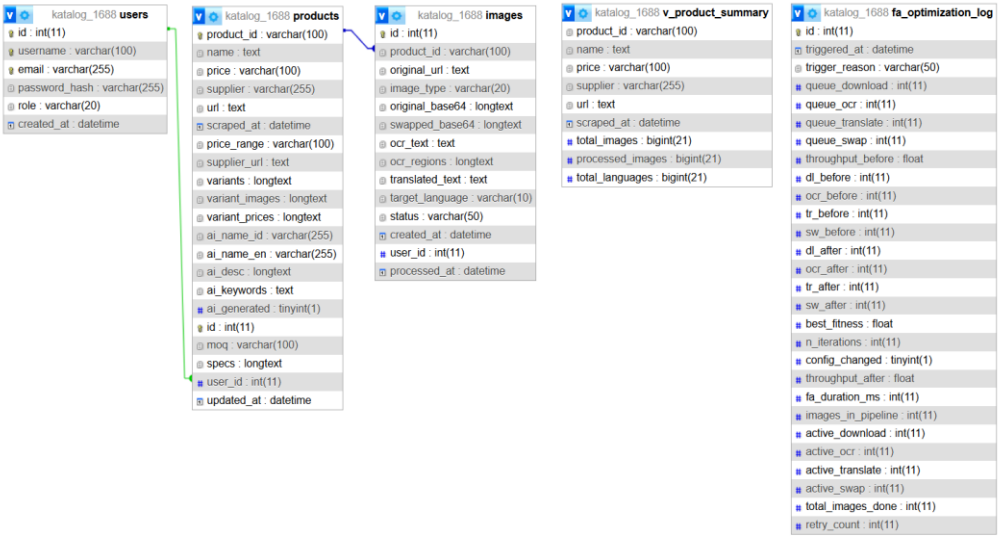
Perangkat Lunak	Versi	Fungsi
Windows 11	24H2	Sistem operasi
Visual Studio Code	Terbaru	Editor kode
Python	3.13.1	Bahasa pemrograman backend
Flask	3.1.3	Framework backend REST API

Perangkat Lunak	Versi	Fungsi
Google Chrome	150.0.7871.115	Menjalankan Chrome Extension
EasyOCR	1.7.2	Ekstraksi teks pada gambar
OpenCV	4.13.0	Pengolahan citra digital
Pillow (PIL)	12.2.0	Manipulasi gambar
Google Translate API	v2	Penerjemahan teks hasil OCR
Google Gemini API	gemini-2.5-flash	Generasi deskripsi produk berbasis AI
MySQL	5.2.1	Basis data
XAMPP	3.3.0	Lingkungan pengelolaan basis data MySQL

4.2 Implementasi Sistem

4.2.1 Implementasi Basis Data

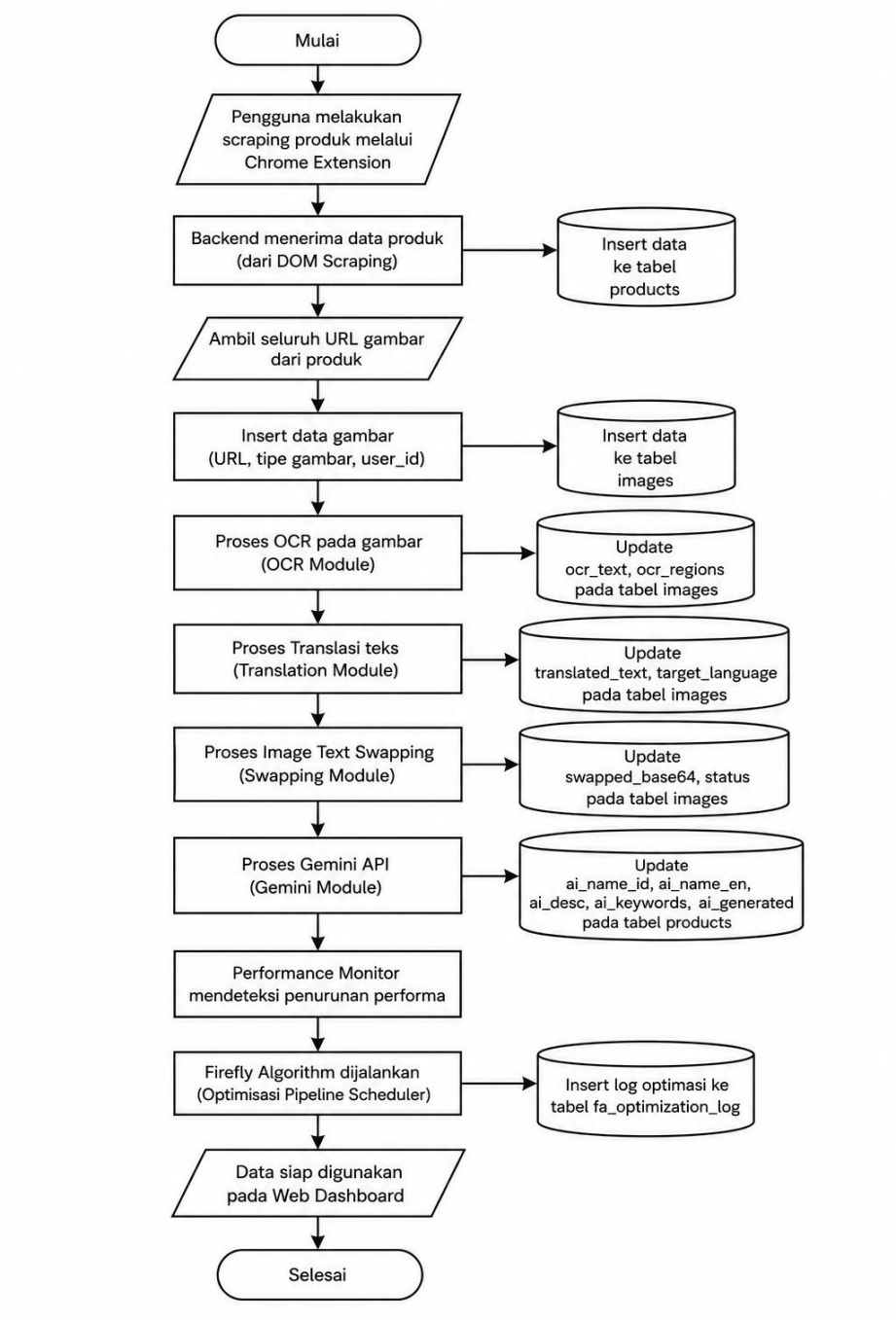
Basis data pada penelitian ini diimplementasikan menggunakan MySQL sebagai media penyimpanan utama untuk mengelola seluruh data yang dihasilkan selama proses scraping, pengolahan gambar, hingga optimasi Pipeline Scheduler. Basis data dirancang menggunakan empat tabel utama, yaitu products, images, users, dan fa_optimization_log, serta satu view yaitu v_product_summary. Relasi antar tabel dibangun menggunakan primary key dan foreign key untuk menjaga konsistensi data selama proses pengolahan berlangsung.



Gambar 4.1 Implementasi Basis Data

Berdasarkan Gambar 4.1, tabel products menjadi pusat penyimpanan data karena setiap produk yang berhasil diperoleh dari proses scraping akan direkam terlebih dahulu pada tabel ini. Setiap produk dapat memiliki lebih dari satu gambar sehingga tabel images dihubungkan dengan tabel products melalui atribut product_id. Selain itu, setiap data gambar juga dikaitkan dengan akun pengguna melalui atribut user_id sehingga proses pengolahan gambar dapat ditelusuri berdasarkan pengguna yang menjalankan sistem.

Selama proses pengolahan berlangsung, data pada setiap tabel akan diperbarui secara bertahap sesuai dengan tahapan pipeline. Alur implementasi basis data pada sistem ditunjukkan pada Gambar 4.2.



Gambar 4.2 Alur Penyimpanan Data ke Basis Data

Setelah pengguna melakukan proses scraping melalui Chrome Extension, backend menerima data produk berupa nama produk, harga, supplier, URL produk, variasi, serta informasi pendukung lainnya. Data tersebut kemudian disimpan ke dalam tabel products menggunakan product_id sebagai identitas utama setiap produk.

Selanjutnya sistem mengambil seluruh URL gambar dari produk yang berhasil diperoleh. Setiap gambar disimpan sebagai satu record pada tabel images yang dihubungkan dengan tabel products melalui atribut product_id. Pada tahap ini sistem menyimpan URL gambar asli (*original_url*), tipe gambar (*image_type*), serta identitas pengguna yang menjalankan proses scraping.

Setelah data gambar tersimpan, modul OCR memproses setiap gambar untuk mendeteksi teks yang terdapat pada gambar produk. Hasil ekstraksi teks disimpan pada atribut ocr_text, sedangkan koordinat setiap area teks disimpan pada atribut ocr_regions. Data tersebut kemudian digunakan sebagai masukan pada proses translasi.

Tahap berikutnya adalah proses translasi. Hasil translasi setiap teks disimpan pada atribut translated_text, sedangkan bahasa tujuan penyimpanan dicatat pada atribut target_language. Setelah proses translasi selesai, modul Image Text Swapping menghasilkan gambar baru yang telah berisi teks hasil translasi. Gambar hasil pengolahan tersebut kemudian disimpan dalam bentuk Base64 pada atribut swapped_base64, sedangkan atribut status diperbarui sesuai dengan kondisi pemrosesan gambar.

Selain pengolahan gambar, sistem juga mengimplementasikan modul Gemini API untuk menghasilkan nama produk, deskripsi, dan kata kunci secara otomatis berdasarkan data hasil scraping dan OCR. Hasil pemrosesan tersebut digunakan untuk memperbarui atribut ai_name_id, ai_name_en, ai_desc, ai_keywords, serta status ai_generated pada tabel products.

Selama proses pengolahan gambar berlangsung, Performance Monitor terus memantau kondisi Pipeline Scheduler. Ketika Firefly Algorithm dijalankan, sistem akan menyimpan seluruh informasi proses optimasi ke dalam tabel fa_optimization_log. Informasi yang dicatat meliputi kondisi antrean (*queue*) pada setiap tahapan pipeline, nilai throughput, jumlah *active worker*, nilai *fitness* terbaik, jumlah iterasi, durasi optimasi, serta konfigurasi Pipeline Scheduler yang diterapkan setelah proses optimasi selesai. Data tersebut digunakan sebagai dasar analisis pada tahap pengujian untuk membandingkan kondisi pipeline sebelum dan sesudah optimasi.

Untuk meningkatkan efisiensi proses pengambilan data pada Web Dashboard, sistem juga mengimplementasikan view v_product_summary. View ini menggabungkan informasi dari tabel products dan images sehingga dashboard dapat langsung menampilkan jumlah gambar, jumlah gambar yang telah diproses, serta jumlah bahasa hasil translasi tanpa perlu melakukan proses *join* dan agregasi berulang pada setiap permintaan data.

Ringkasan implementasi basis data masing-masing tabel dan *view* ditunjukkan pada Tabel 4.3.

Tabel 4.3 Implementasi Basis Data

Objek Basis Data	Implementasi dalam Sistem
products	Menyimpan data hasil scraping dan diperbarui kembali setelah proses Gemini selesai.
images	Menyimpan data setiap gambar dan diperbarui secara bertahap oleh modul OCR, Translation, dan Image Text Swapping.
users	Digunakan pada proses autentikasi serta mengidentifikasi pemilik data scraping.
fa_optimization_log	Mencatat setiap proses optimasi Firefly Algorithm sebagai data evaluasi performa Pipeline Scheduler.
v_product_summary	Menyediakan data agregat untuk Web Dashboard sehingga proses pengambilan data menjadi lebih efisien.

4.2.2 Implementasi Chrome Extension

Chrome Extension dikembangkan menggunakan HTML, CSS, dan JavaScript dengan mengadopsi arsitektur Manifest V3. Ekstensi berfungsi sebagai antarmuka utama yang menghubungkan pengguna dengan backend sehingga proses scraping, pengiriman data, serta penerimaan hasil pemrosesan dapat dilakukan secara langsung pada halaman produk platform 1688.

Implementasi Chrome Extension dibangun secara modular dengan membagi fungsi ke dalam beberapa berkas sesuai tanggung jawabnya. Struktur direktori Chrome Extension ditunjukkan pada Gambar 4.3.

Gambar 4.3 Struktur Direktori Chrome Extension

Berdasarkan Gambar 4.3, Chrome Extension terdiri atas lima komponen utama, yaitu manifest.json, content.js, background.js, popup.html, dan folder icons. Berkas manifest.json digunakan sebagai konfigurasi utama ekstensi, content.js bertugas melakukan DOM Scraping dan manipulasi halaman 1688, background.js menangani komunikasi dengan Backend Flask, popup.html menyediakan antarmuka pengguna, sedangkan folder icons berisi ikon yang digunakan sebagai identitas ekstensi pada browser Google Chrome.

Selanjutnya implementasi setiap komponen dijelaskan sebagai berikut.

4.2.2.1 Implementasi Manifest V3

Implementasi Chrome Extension diawali dengan penyusunan berkas manifest.json sebagai konfigurasi utama ekstensi. Berkas ini menentukan identitas ekstensi, izin (*permissions*), halaman yang dapat diakses (*host permissions*), *background service worker*, serta *content script* yang akan dijalankan ketika pengguna membuka halaman produk pada platform 1688.

Kode Program 4.1 Konfigurasi Manifest V3

```
{
  "manifest_version": 3,
  "name": "1688 Katalog Assistant",
  "version": "4.0",
  "permissions": ["activeTab", "scripting", "downloads", "storage",
"alarms"],
  "host_permissions": ["https://*.1688.com/*",
"http://localhost/*", "http://127.0.0.1/*"],
  "action": { "default_popup": "popup.html", "default_icon": {
"48": "icons/icon48.png" } },
  "background": { "service_worker": "background.js" },
  "content_scripts": [{
    "matches": ["https://*.1688.com/*"],
    "js": ["content.js"],
    "run_at": "document_idle"
```

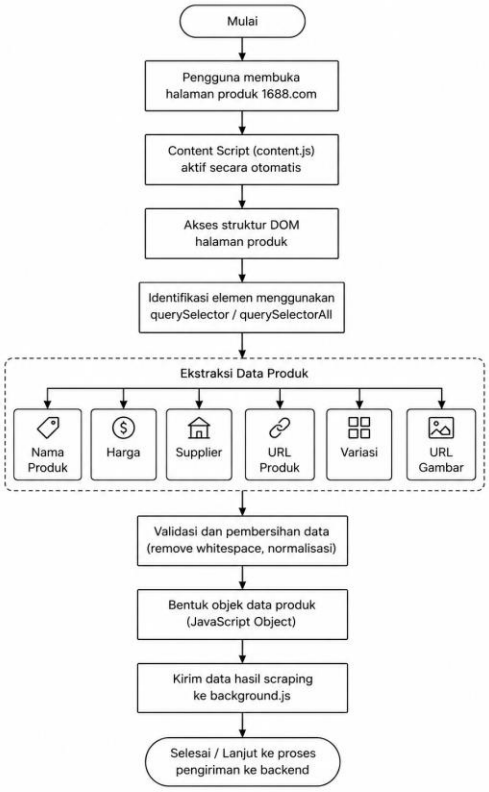
```
  }],
  "icons": { "48": "icons/icon48.png" }
}
```

Pada implementasi tersebut, properti content_scripts digunakan untuk menjalankan content.js secara otomatis ketika pengguna membuka halaman produk 1688. Selanjutnya background.js dijalankan sebagai *service worker* untuk menangani komunikasi antara Chrome Extension dengan Backend Flask. Sementara itu, *host_permissions* memberikan izin kepada ekstensi untuk mengakses halaman produk pada domain 1688 sehingga proses DOM Scraping dapat dilakukan.

4.2.2.2 Implementasi DOM Scraping

Setelah Chrome Extension aktif pada halaman produk, content.js dijalankan secara otomatis untuk mengakses struktur Document Object Model (DOM) yang telah dirender oleh browser. Implementasi DOM Scraping dilakukan menggunakan JavaScript DOM API dengan memanfaatkan fungsi querySelector() dan querySelectorAll() untuk memperoleh informasi produk dari elemen HTML.

Alur implementasi DOM Scraping ditunjukkan pada Gambar 4.4.



Gambar 4.4 Alur implementasi DOM Scraping

Implementasi ekstraksi data dilakukan dengan memilih elemen HTML sesuai struktur halaman produk menggunakan selector CSS. Contoh proses pengambilan elemen ditunjukkan pada Kode Program 4.2.

Kode Program 4.2 Pengambilan Data Menggunakan DOM Scraping

```
function scanShadow(root) {
    root.querySelectorAll('img').forEach(img => {
        let src = img.getAttribute('src')
               || img.getAttribute('data-src')
               || '';
        if (src.startsWith('//'))
            src = 'https:' + src;
        if (src.includes('cbu01.alicdn.com') ||
            src.includes('alicdn.com/img/ibank')) {
            addUrl(src, img);
        }
    });
    root.querySelectorAll('*').forEach(el => {
        if (el.shadowRoot)
            scanShadow(el.shadowRoot);
    });
}
document.querySelectorAll('img').forEach(img => {
    let src = img.getAttribute('src') || '';
    if (src.startsWith('//'))
        src = 'https:' + src;
    if (src.includes('cbu01.alicdn.com') ||
        src.includes('alicdn.com/img/ibank')) {
        addUrl(src, img);
    }
});
```

Kode Program 4.2 menunjukkan implementasi DOM Scraping yang digunakan untuk mengambil gambar produk dari halaman 1688. Sistem terlebih dahulu mencari seluruh elemen gambar (<img>) yang terdapat pada halaman menggunakan fungsi querySelectorAll(). Setelah itu, sistem mengambil alamat URL dari setiap gambar melalui atribut src atau data-src. Apabila URL masih menggunakan format relatif (//), sistem akan menambahkan protokol https: sehingga URL dapat digunakan pada proses berikutnya.

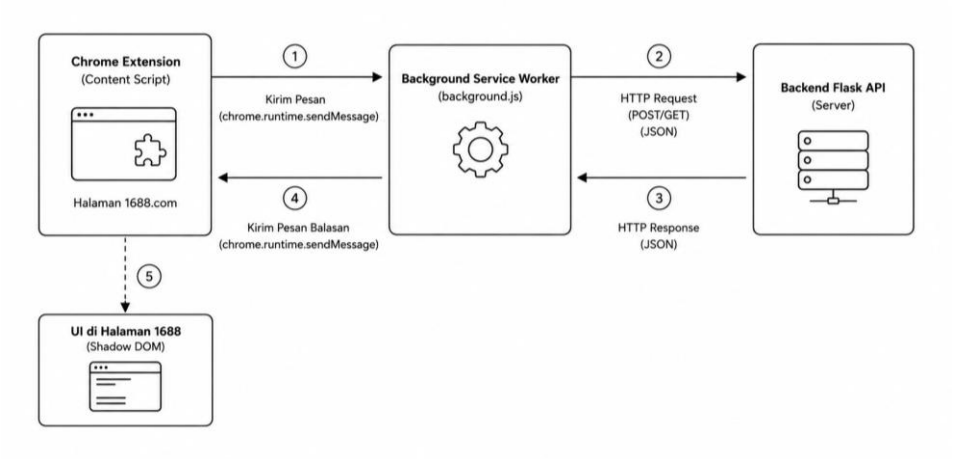
Selanjutnya, sistem melakukan penyaringan terhadap URL gambar yang diperoleh. Hanya gambar yang berasal dari domain penyimpanan gambar produk 1688, yaitu cbu01.alicdn.com dan alicdn.com/img/ibank, yang akan diproses lebih lanjut. URL yang memenuhi kriteria kemudian disimpan menggunakan fungsi

addUrl() agar tidak terjadi duplikasi data dan siap digunakan pada tahap OCR serta *Image Text Swapping*.

Selain mengambil gambar dari halaman utama, sistem juga melakukan pemindaian pada Shadow DOM melalui fungsi scanShadow(). Hal ini dilakukan karena beberapa gambar produk pada platform 1688 berada di dalam komponen web yang tidak dapat diakses melalui DOM biasa. Dengan melakukan pemindaian secara rekursif pada setiap shadowRoot, sistem dapat memperoleh gambar-gambar tersebut sehingga proses scraping menjadi lebih lengkap dan tidak hanya bergantung pada struktur DOM utama.

4.2.2.3 Implementasi Komunikasi dengan Backend

Komunikasi antara Chrome Extension dan Backend Flask diimplementasikan menggunakan Background Service Worker pada berkas background.js. Komponen ini menerima data hasil DOM Scraping dari content.js, kemudian mengirimkannya ke Backend Flask melalui REST API menggunakan metode HTTP. Alur komunikasi ditunjukkan pada Gambar 4.5.



Gambar 4.5 Alur implementasi DOM Scraping

Berdasarkan Gambar 4.5, proses komunikasi diawali ketika Content Script pada halaman 1688.com mengirimkan data hasil DOM Scraping kepada Background Service Worker melalui mekanisme *message passing* menggunakan chrome.runtime.sendMessage() (1). Data yang dikirim berisi informasi produk yang telah berhasil diekstraksi dari halaman web.

Selanjutnya, Background Service Worker meneruskan data tersebut ke Backend Flask API menggunakan *REST API* melalui permintaan HTTP Request dengan format JSON (2). Pada tahap ini, backend menerima data untuk diproses sesuai dengan layanan yang diminta, seperti penyimpanan data, proses OCR, translasi, maupun Image Text Swapping.

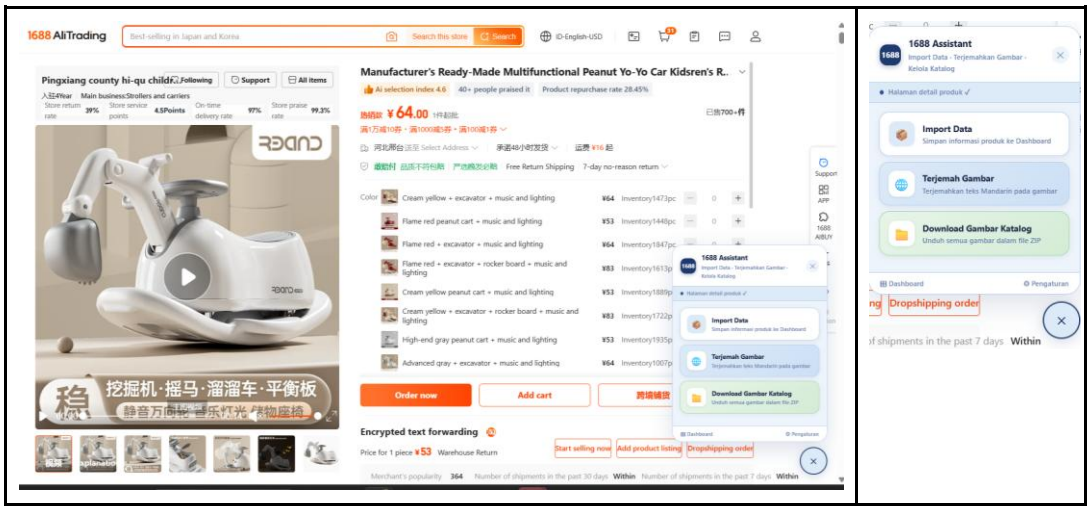
Setelah proses pada backend selesai dilakukan, server mengembalikan hasil pemrosesan dalam bentuk HTTP Response berformat JSON kepada Background Service Worker (3). Respons tersebut kemudian diteruskan kembali ke Content Script menggunakan mekanisme chrome.runtime.sendMessage() (4) sehingga hasil pemrosesan dapat diterima oleh ekstensi.

Tahap terakhir, Content Script menampilkan hasil yang diterima ke dalam antarmuka pengguna (UI) yang disisipkan pada halaman 1688.com menggunakan Shadow DOM (5). Dengan mekanisme tersebut, antarmuka ekstensi dapat ditampilkan tanpa mengubah struktur asli halaman web sehingga kompatibilitas dengan halaman 1688.com tetap terjaga.

Melalui mekanisme ini, Background Service Worker berperan sebagai penghubung antara Chrome Extension dan Backend Flask. Pendekatan tersebut membuat proses komunikasi menjadi lebih terstruktur, memisahkan logika antarmuka dengan logika pemrosesan di server, serta memudahkan pengelolaan pertukaran data antara frontend dan backend.

4.2.2.4 Implementasi Antarmuka Chrome Extension

Antarmuka pengguna Chrome Extension diimplementasikan menggunakan popup.html yang ditampilkan ketika pengguna memilih ikon ekstensi pada toolbar browser. Popup berfungsi sebagai media interaksi pengguna untuk menjalankan fitur-fitur utama sistem, seperti proses scraping, penerjemahan gambar, akses Web Dashboard, serta konfigurasi sistem.



Gambar 4.6 Implementasi Popup Chrome Extension

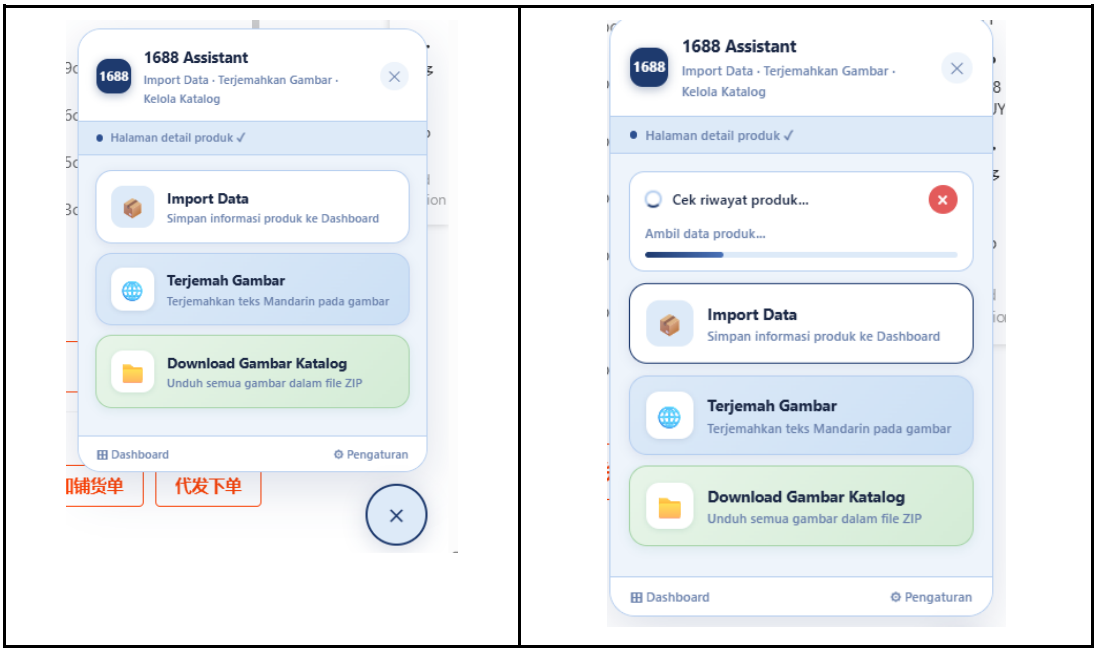
Implementasi antarmuka dibuat sederhana agar seluruh fitur utama dapat diakses tanpa mengganggu aktivitas pengguna pada halaman produk. Setiap tombol

pada popup terhubung dengan fungsi JavaScript yang akan mengirimkan perintah ke content.js atau background.js sesuai proses yang dijalankan.

4.2.2.5 Implementasi Fitur Import Data

Fitur Import Data merupakan salah satu fitur utama pada Chrome Extension yang digunakan untuk mengambil informasi produk dari platform 1688.com dan menyimpannya ke dalam sistem. Implementasi fitur ini tidak hanya mencakup proses pengambilan data dari halaman web, tetapi juga mengintegrasikan beberapa modul sistem, yaitu Content Script, Background Service Worker, Backend Flask, Basis Data, dan Web Dashboard. Integrasi tersebut memungkinkan informasi produk yang diperoleh dari halaman 1688.com dapat diproses, disimpan, dan ditampilkan kembali pada dashboard untuk mendukung pengelolaan katalog produk.

Implementasi antarmuka fitur Import Data pada Chrome Extension ditunjukkan pada Gambar 4.7. Fitur ini disediakan dalam bentuk tombol Import Data yang dapat digunakan pengguna setelah membuka halaman detail produk pada platform 1688.com. Ketika tombol tersebut dipilih, sistem akan menjalankan proses pengambilan informasi produk secara otomatis tanpa memerlukan input manual dari pengguna.

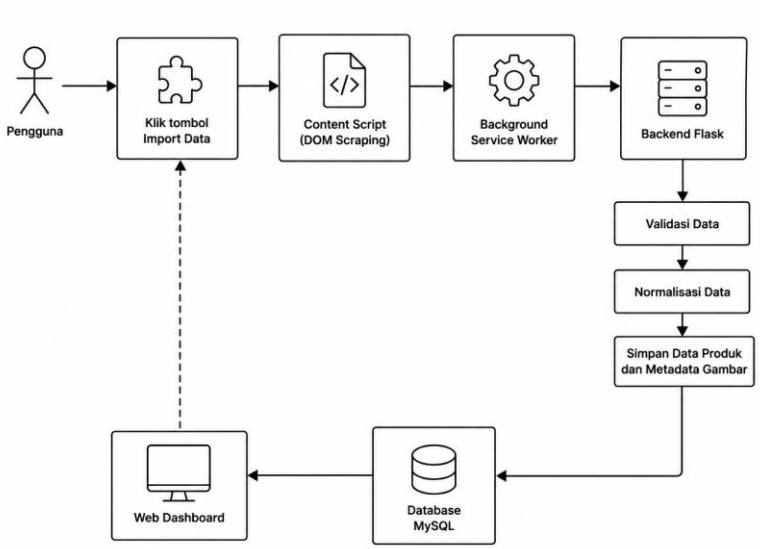


Gambar 4.7 Implementasi Fitur Import Data pada Chrome Extension

Selanjutnya, proses implementasi fitur Import Data melibatkan beberapa modul sistem yang saling berkomunikasi. Proses diawali ketika Content Script melakukan DOM Scraping untuk memperoleh informasi produk, seperti nama

produk, harga, spesifikasi, spp, URL produk, serta URL gambar. Data yang berhasil diperoleh kemudian disusun dalam format JSON sebelum diteruskan ke Background Service Worker melalui mekanisme *message passing*. Selanjutnya, Background Service Worker mengirimkan data tersebut ke Backend Flask menggunakan REST API melalui endpoint /api/scrape/save. Pada sisi backend dilakukan proses validasi dan normalisasi data sebelum informasi produk beserta metadata gambar disimpan ke dalam basis data MySQL. Setelah proses penyimpanan selesai, backend mengirimkan respons kepada Chrome Extension sebagai indikator bahwa proses impor data telah berhasil dilakukan.

Agar alur implementasi fitur lebih mudah dipahami, hubungan antar modul pada proses Import Data ditunjukkan pada Gambar 4.8.



Gambar 4.8 Alur Implementasi Fitur Import Data

Berdasarkan Gambar 4.8, implementasi fitur Import Data diawali dengan proses DOM Scraping pada *Content Script* untuk mengambil informasi produk dari halaman 1688.com. Data yang diperoleh kemudian diteruskan ke Background Service Worker dan dikirim ke Backend Flask melalui REST API. Selanjutnya, backend melakukan validasi, normalisasi, serta menyimpan data produk dan metadata gambar ke dalam basis data MySQL sebelum ditampilkan pada Web Dashboard.

Implementasi proses pengambilan informasi produk menggunakan DOM Scraping ditunjukkan pada Kode Program 4.3.

Kode Program 4.3 Implementasi Pengambilan Data Produk

```
function scrapePage() {
```

```
    let name = '';
    ...
    let price = '';
    ...
    let supplier = '';
    ...
    const specs = {};
    return {
        product_name: name,
        price,
        supplier,
        specifications: specs,
        product_url: location.href
    };
}
```

Kode Program 4.3 menunjukkan implementasi proses pengambilan data produk pada halaman detail platform 1688 menggunakan teknik DOM Scraping. Implementasi diawali dengan pencarian elemen HTML yang berisi nama produk menggunakan beberapa selector alternatif untuk menyesuaikan perbedaan struktur halaman. Selanjutnya sistem mengekstraksi informasi harga, nama pemasok (*supplier*), serta spesifikasi produk dari elemen DOM yang tersedia. Setelah seluruh informasi berhasil diperoleh, data disusun ke dalam sebuah objek yang berisi nama produk, harga, pemasok, spesifikasi, dan URL produk sebelum diteruskan ke proses pengiriman data menuju backend.

Selanjutnya, proses pengiriman data hasil scraping menuju Backend Flask diimplementasikan menggunakan Background Service Worker sebagaimana ditunjukkan pada Kode Program 4.4.

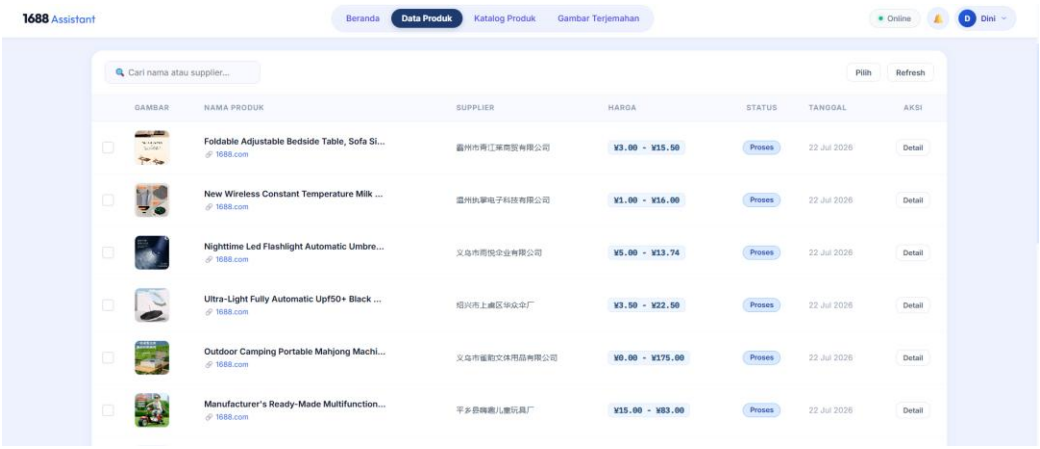
Kode Program 4.4 Implementasi Pengiriman Data ke Backend

```
case 'SAVE_SCRAPE': {
        const url = cfg.backendUrl;
        try {
          const r = await fetch(`${url}/api/scrape/save`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify(msg.data),
            signal: AbortSignal.timeout(15000)
          });
          return sendResponse(await r.json());
        } catch(e) { return sendResponse({ success: false, error:
e.message }); }
```

```
        }
```

Kode Program 4.4 menunjukkan implementasi proses pengiriman data hasil scraping dari Content Script menuju Backend Flask melalui Background Service Worker. Data produk yang telah diperoleh sebelumnya dikirim menggunakan metode HTTP POST ke endpoint /api/scrape/save dalam format JSON melalui fungsi fetch(). Pada proses ini sistem menetapkan header Content-Type: application/json agar backend dapat mengenali format data yang diterima. Selain itu, mekanisme AbortSignal.timeout(15000) diterapkan untuk membatasi waktu tunggu respons selama 15 detik sehingga permintaan yang tidak memperoleh respons dapat dihentikan secara otomatis. Setelah backend selesai memproses permintaan, sistem mengembalikan respons dalam format JSON kepada Chrome Extension sebagai informasi keberhasilan maupun kegagalan proses penyimpanan data.

Setelah seluruh proses selesai, informasi produk yang berhasil disimpan pada basis data akan ditampilkan pada Web Dashboard sebagai pusat pengelolaan katalog produk.

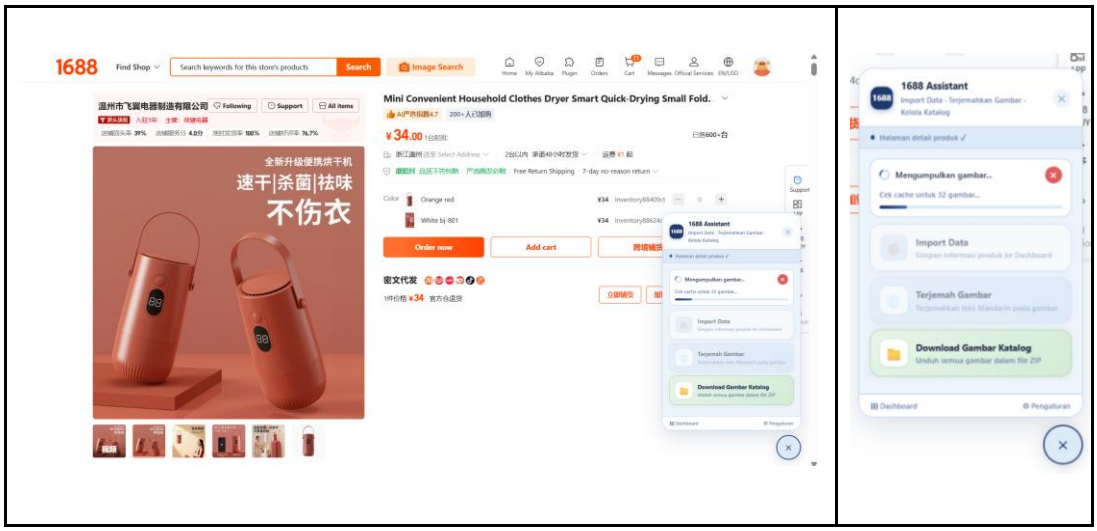


Gambar 4.9 Hasil Import Data pada Web Dashboard

4.2.2.6 Implementasi Fitur Terjemahkan Gambar

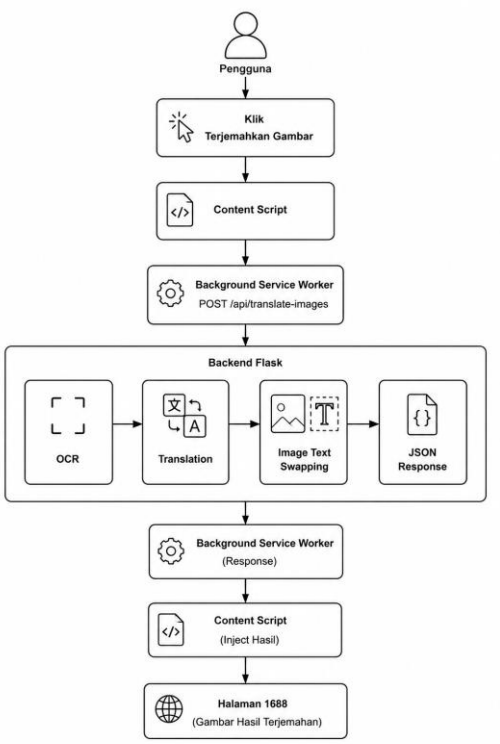
Fitur Terjemahkan Gambar merupakan salah satu fitur utama yang diimplementasikan pada Chrome Extension untuk menerjemahkan teks berbahasa Mandarin yang terdapat pada gambar produk di platform 1688 secara otomatis. Fitur ini membantu pengguna memahami informasi visual produk tanpa perlu melakukan penerjemahan secara manual maupun menggunakan aplikasi penerjemah lain.

Ketika pengguna menekan tombol Terjemahkan Gambar, Chrome Extension terlebih dahulu mengumpulkan URL seluruh gambar produk yang terdapat pada halaman. Selanjutnya, URL gambar tersebut dikirim ke Backend Flask untuk diproses melalui beberapa tahapan, yaitu ekstraksi teks menggunakan Optical Character Recognition (OCR), penerjemahan teks ke bahasa Indonesia, serta penggantian teks pada gambar melalui proses Image Text Swapping. Setelah seluruh proses selesai, gambar hasil terjemahan dikirim kembali ke Chrome Extension dan ditampilkan secara langsung pada halaman produk.



Gambar 4.10 Implementasi Fitur Terjemahkan Gambar

Berdasarkan Gambar 4.10, pengguna dapat menjalankan proses penerjemahan gambar dengan menekan tombol Terjemahkan Gambar pada Chrome Extension. Setelah tombol dipilih, sistem akan mengirimkan URL gambar produk ke backend untuk diproses melalui tahapan OCR, penerjemahan teks, dan Image Text Swapping sebelum hasilnya ditampilkan kembali pada halaman produk.



Gambar 4.11 Alur Implementasi Fitur Terjemahkan Gambar

Berdasarkan Gambar 4.11, implementasi fitur Terjemahkan Gambar diawali ketika pengguna memilih tombol Terjemahkan Gambar pada Chrome Extension. Content Script terlebih dahulu mengumpulkan URL gambar produk yang terdapat pada halaman, kemudian meneruskannya ke Background Service Worker untuk dikirim ke Backend Flask melalui REST API. Selanjutnya backend menjalankan proses OCR untuk mendeteksi teks pada gambar, menerjemahkan hasil OCR ke bahasa Indonesia, serta melakukan proses Image Text Swapping sehingga menghasilkan gambar baru dengan teks hasil terjemahan. Setelah seluruh proses selesai, backend mengirimkan hasilnya dalam format JSON kepada Chrome Extension untuk mengganti gambar asli pada halaman produk secara otomatis.

Kode Program 4.5 Implementasi Pengambilan URL Gambar

```
const quickImgs = [];
    const _qSeen = new Set();
    document.querySelectorAll('img').forEach(img => {
      let src = img.getAttribute('src') || '';
      if (src.startsWith('//')) src = 'https:' + src;
      const key = src.split('?')[0];
      if ((src.includes('cbu01.alicdn.com') ||
src.includes('alicdn.com/img/ibank'))
          && !_qSeen.has(key)) {
        _qSeen.add(key);
        quickImgs.push({index: quickImgs.length, src, width: 400, height:
```

```
400});
      }
    });
```

Kode Program 4.5 menunjukkan implementasi proses pengambilan URL gambar produk dari halaman 1688 menggunakan Content Script. Sistem melakukan pemindaian terhadap elemen gambar yang terdapat pada halaman, termasuk gambar yang baru muncul setelah proses *scrolling*. Seluruh URL gambar yang berhasil diperoleh kemudian dikumpulkan ke dalam sebuah daftar dan digunakan sebagai masukan pada proses OCR dan Image Text Swapping di backend.

Kode Program 4.6 Implementasi Pengiriman URL Gambar ke Backend

```
// Background Service Worker — meneruskan request ke backend
case 'TRANSLATE_IMAGES': {
        const url = msg.backendUrl || cfg.backendUrl;
        try {
          const r = await fetch(`${url}/api/translate-images`, {
            method: 'POST',
            headers: { 'Content-Type': 'application/json' },
            body: JSON.stringify({
              images: msg.images,
              target_language: msg.targetLang || 'id',
              product_id: msg.productId || ''
            }),
            signal: AbortSignal.timeout(120000)
          });
          return sendResponse(await r.json());
        } catch(e) { return sendResponse({ success: false, error:
e.message }); }
      }
```

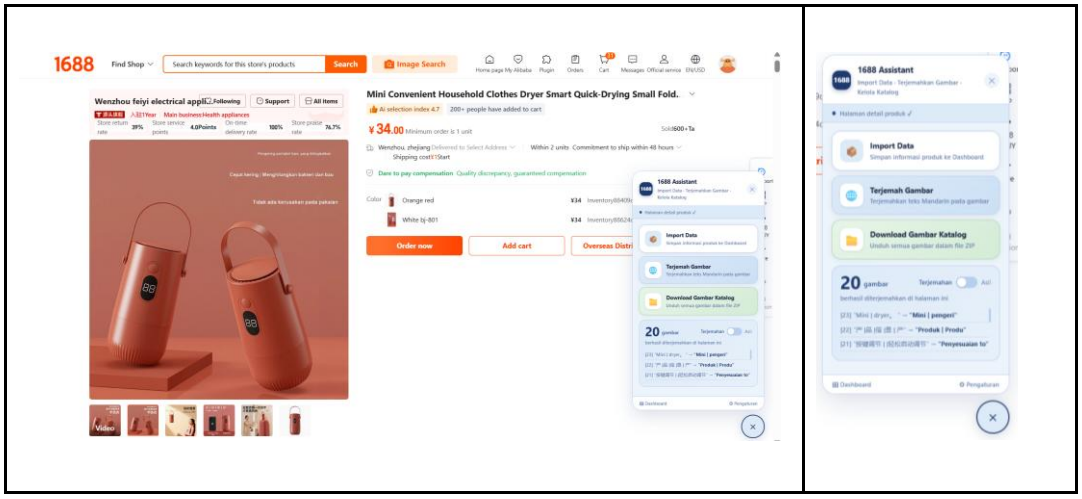
Kode Program 4.6 menunjukkan implementasi pengiriman URL gambar dari Chrome Extension menuju Backend Flask menggunakan Background Service Worker. Daftar URL gambar dikirim melalui metode HTTP POST dalam format JSON sehingga backend dapat memproses setiap gambar melalui tahapan OCR, penerjemahan, dan Image Text Swapping. Setelah seluruh proses selesai, backend mengembalikan informasi hasil translasi kepada Chrome Extension untuk ditampilkan kembali pada halaman produk.

Kode Program 4.7 Implementasi Penggantian Gambar

```
 async function injectTranslated(results) {
   let replaced = 0;
   for (const item of results) {
     if (!item.translated_image_url && !item.translated_image_base64)
continue;
```

```
      const translatedSrc = item.translated_image_url ||
`${item.translated_image_base64}`;
      const res = await sendBg({action:'INJECT_IMAGE',
originalSrc:item.original_src,
translatedB64:item.translated_image_base64||item.translated_image_url.replac
e('','')});
      if (res.success && res.replaced > 0) replaced += res.replaced;
    }
    return replaced;
  }
```

Kode Program 4.7 menunjukkan implementasi proses penggantian gambar asli dengan gambar hasil translasi pada halaman produk. Setelah menerima respons dari backend, Content Script memperbarui atribut gambar pada elemen DOM sehingga gambar yang ditampilkan kepada pengguna merupakan hasil Image Text Swapping tanpa perlu memuat ulang halaman.



Gambar 4.12 Hasil Implementasi Fitur Terjemahkan Gambar

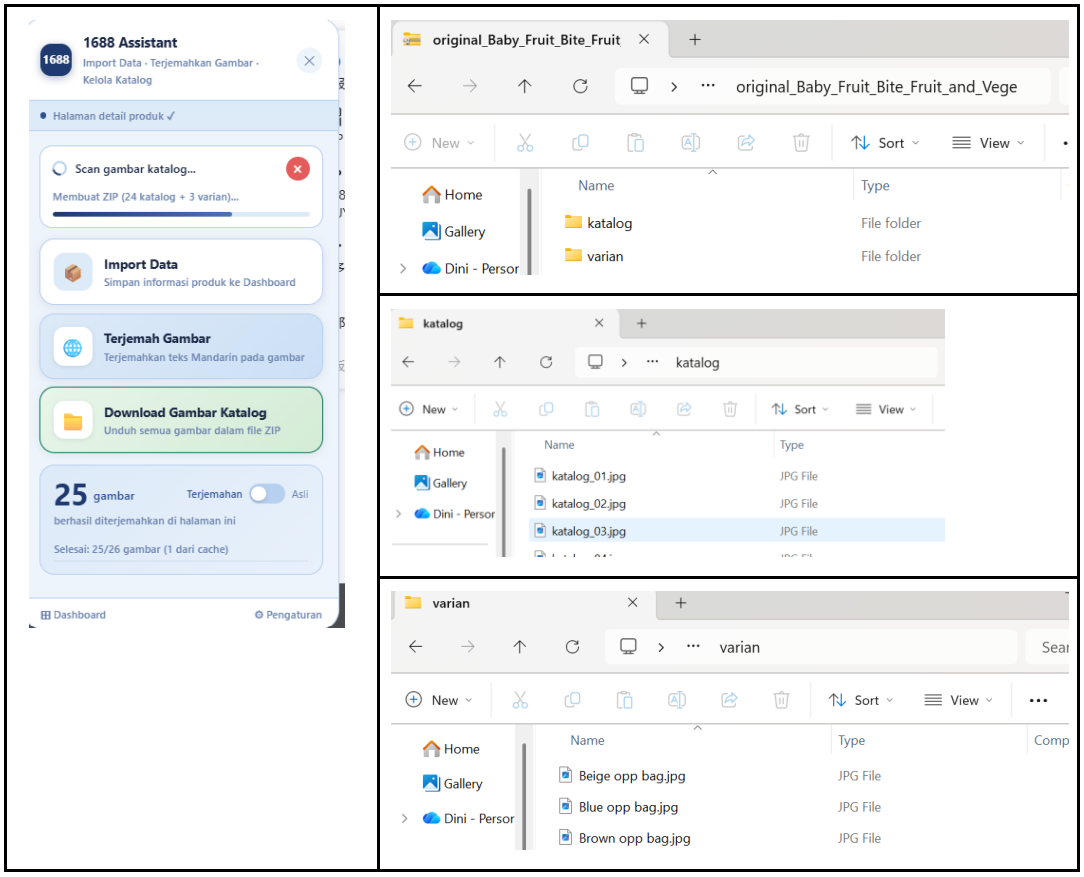
Berdasarkan Gambar 4.12, gambar produk yang semula berisi teks berbahasa Mandarin berhasil ditampilkan kembali dengan teks hasil terjemahan sesuai bahasa yang dipilih pengguna. . Hasil translasi ditampilkan secara langsung pada halaman 1688 melalui Chrome Extension tanpa mengubah tata letak halaman maupun mengganggu proses navigasi pengguna.

4.2.2.7 Implementasi Fitur Download Gambar Katalog

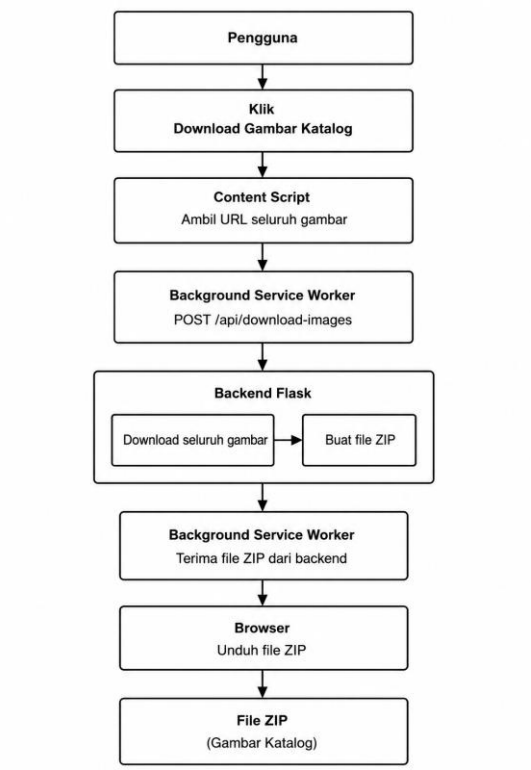
Fitur Download Gambar Katalog diimplementasikan untuk memudahkan pengguna mengunduh seluruh gambar produk yang terdapat pada halaman 1688

secara otomatis. Berbeda dengan proses penyimpanan gambar secara manual satu per satu, fitur ini memungkinkan Chrome Extension mengumpulkan seluruh URL gambar produk dan mengirimkannya ke backend untuk diproses menjadi satu berkas arsip (*ZIP*). Setelah proses selesai, berkas hasil kompresi dikirim kembali kepada pengguna sehingga seluruh gambar katalog dapat diunduh dalam satu kali proses.

Tampilan hasil implementasi fitur Download Gambar Katalog pada Chrome Extension ditunjukkan pada Gambar 4.13, sedangkan alur proses pengunduhan gambar dari Chrome Extension hingga backend menghasilkan berkas ZIP dapat dilihat pada Gambar 4.14.



Gambar 4.13 Implementasi Fitur Download Gambar Katalog



Gambar 4.14 Alur Implementasi Fitur Download Gambar Katalog

Berdasarkan Gambar 4.14, implementasi fitur Download Gambar Katalog diawali ketika pengguna menekan tombol Download Gambar pada Chrome Extension. Selanjutnya Content Script mengumpulkan URL seluruh gambar produk yang terdapat pada halaman dan meneruskannya ke Background Service Worker. Data tersebut kemudian dikirim ke Backend Flask melalui REST API untuk diproses. Backend mengunduh seluruh gambar berdasarkan URL yang diterima, kemudian menggabungkannya ke dalam satu berkas berformat ZIP. Setelah proses selesai, berkas ZIP dikirim kembali kepada Chrome Extension dan secara otomatis diunduh oleh browser.

Kode Program 4.8 Implementasi Pengiriman Permintaan Download

```
const total = catalogUrls.length + variantUrls.length;
const isTranslatedMode = !_toggleState && translationMap.size > 0;
const downloadMode = isTranslatedMode ? 'translated' : 'original';

const resp = await fetch(`${backendUrl}/api/download-zip-from-urls`, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        catalog: catalogUrls,
```

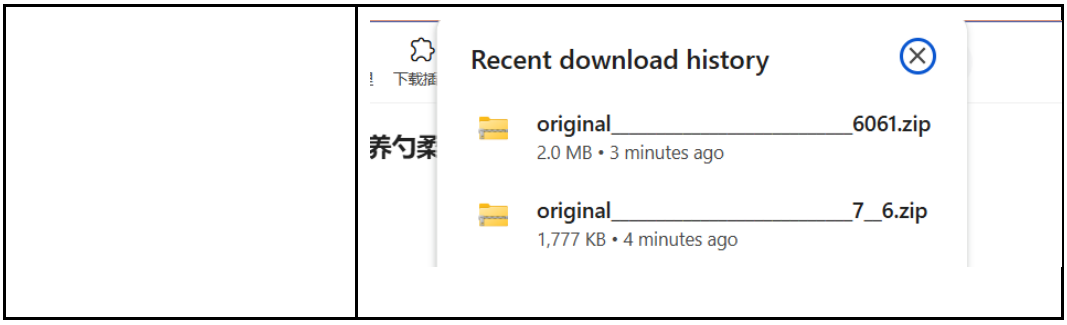
```
        variant: variantUrls,
        variant_names: variantNameMap,
        name: pageData.name || productId,
        product_id: productId,
        mode: downloadMode
    })
});
```

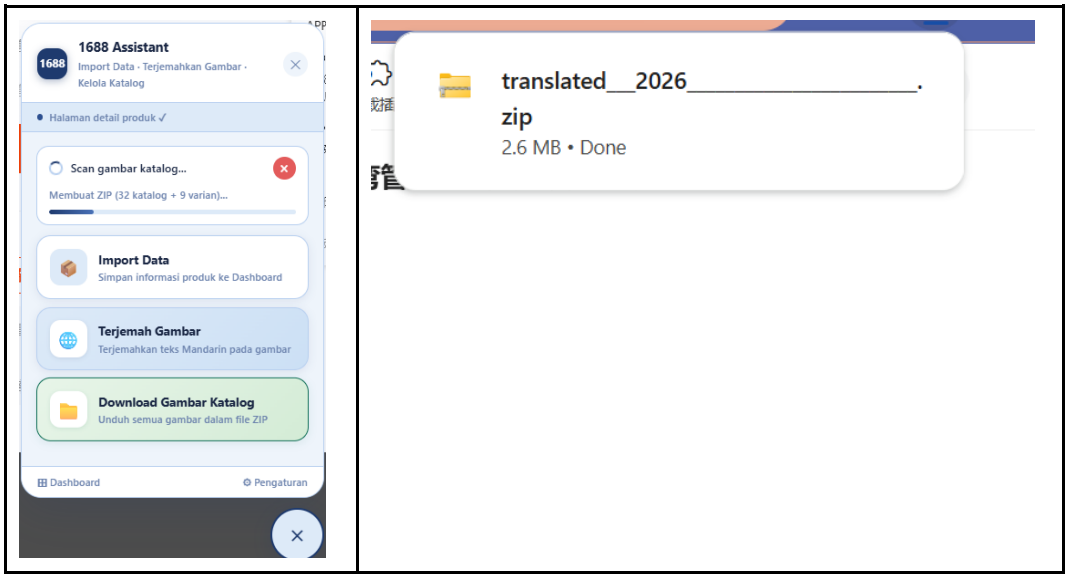
Kode Program 4.8 menunjukkan implementasi pengiriman permintaan download gambar dari Chrome Extension menuju Backend Flask melalui Background Service Worker. Daftar URL gambar produk dikirim menggunakan metode HTTP POST dalam format JSON menuju endpoint yang menangani proses download. Selanjutnya backend mengunduh seluruh gambar dan mengompresnya menjadi satu berkas ZIP sebelum mengirimkan respons kembali ke Chrome Extension.

Kode Program 4.9 Implementasi Pengunduhan Berkas

```
if (!resp.ok) throw new Error('Server error');
const blob = await resp.blob();
const url = URL.createObjectURL(blob);
const a = document.createElement('a');
a.href = url;
a.download = filename;

document.body.appendChild(a);
a.click();
document.body.removeChild(a);
URL.revokeObjectURL(url);
```

Kode Program 4.9 menunjukkan implementasi proses pengunduhan berkas hasil kompresi. Setelah menerima respons dari backend, Chrome Extension membuat tautan unduhan secara otomatis dan menjalankan proses download sehingga pengguna memperoleh seluruh gambar produk dalam satu berkas ZIP tanpa perlu mengunduh gambar satu per satu.





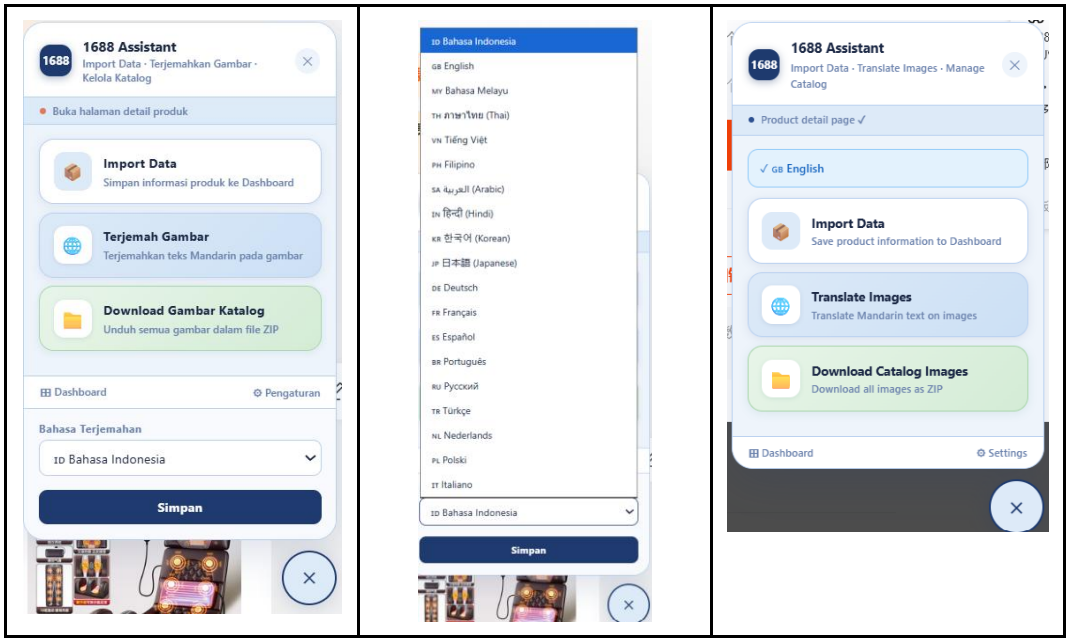
Gambar 4.15 Hasil Implementasi

Berdasarkan Gambar 4.15, sistem berhasil mengunduh seluruh gambar katalog produk dan menghasilkan satu berkas ZIP yang berisi seluruh gambar tersebut. Implementasi ini memungkinkan proses pengunduhan dilakukan secara otomatis dalam satu kali operasi sehingga pengguna tidak perlu mengunduh setiap gambar secara terpisah. Dengan demikian, proses pengambilan dan penyimpanan gambar katalog menjadi lebih efisien serta memudahkan pengguna dalam memperoleh seluruh aset gambar produk.

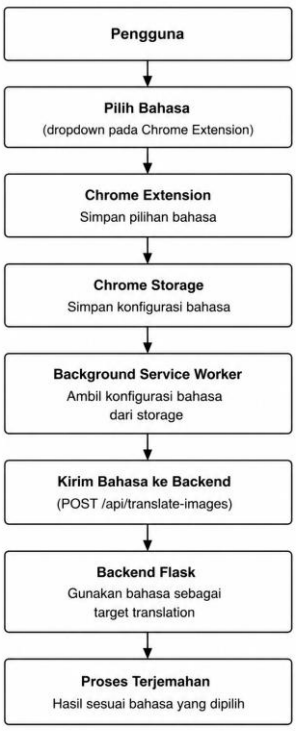
4.2.2.8 Implementasi Pengaturan Bahasa

Fitur Pengaturan Bahasa diimplementasikan untuk memberikan fleksibilitas kepada pengguna dalam menentukan bahasa hasil terjemahan yang akan digunakan pada proses penerjemahan gambar produk. Bahasa yang dipilih akan digunakan sebagai bahasa tujuan pada proses translasi sehingga hasil terjemahan dapat disesuaikan dengan kebutuhan pengguna. Pengaturan ini disimpan oleh Chrome Extension sehingga dapat digunakan kembali pada proses penerjemahan berikutnya tanpa perlu melakukan konfigurasi ulang.

Tabel 4.4 Antarmuka Pengaturan Bahasa



Berdasarkan Tabel 4.4, pengguna dapat memilih bahasa tujuan yang diinginkan melalui antarmuka Chrome Extension. Alur penyimpanan dan penggunaan pengaturan bahasa tersebut dalam proses penerjemahan ditunjukkan pada Gambar 4.16.



Gambar 4.16 Alur Implementasi Pengaturan Bahasa

Berdasarkan Gambar 4.16, implementasi fitur Pengaturan Bahasa diawali ketika pengguna memilih bahasa tujuan melalui antarmuka Chrome Extension. Selanjutnya pilihan bahasa disimpan menggunakan Chrome Storage sehingga tetap tersedia meskipun halaman dimuat ulang. Ketika pengguna menjalankan proses Terjemahkan Gambar, Background Service Worker mengambil konfigurasi bahasa yang tersimpan dan mengirimkannya ke Backend Flask sebagai parameter bahasa tujuan. Backend kemudian menggunakan parameter tersebut sebagai acuan pada proses penerjemahan teks.

Kode Program 4.10 Implementasi Penyimpanan Pengaturan Bahasa

```
// File: background.js
// Menyimpan konfigurasi bahasa ke Chrome Storage
case 'SAVE_CONFIG':
    await chrome.storage.local.set({
        backendUrl: msg.backendUrl || cfg.backendUrl,
        targetLang: msg.targetLang || cfg.targetLang
    });

    return sendResponse({
        success: true
    });
```

Kode Program 4.10 menunjukkan implementasi penyimpanan pengaturan bahasa pada file background.js. Ketika pengguna memilih bahasa tujuan melalui antarmuka Chrome Extension, nilai bahasa disimpan ke dalam Chrome Storage menggunakan chrome.storage.local.set(). Dengan mekanisme tersebut, konfigurasi bahasa tetap tersedia pada penggunaan berikutnya tanpa perlu dilakukan pengaturan ulang.

Kode Program 4.11 Implementasi Pengiriman Bahasa ke Backend

```
// File: background.js
// Mengambil konfigurasi bahasa yang tersimpan
const cfg = await getConfig();
// Mengirim bahasa tujuan ke Backend Flask
const response = await fetch(`${url}/api/translate-images`, {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json'
    },
    body: JSON.stringify({
        images: msg.images,
        target_language: msg.targetLang || cfg.targetLang,
        product_id: msg.productId || ''
    })
```

```
});
return sendResponse(await response.json());
```

Kode Program 4.11 menunjukkan implementasi pengiriman konfigurasi bahasa pada file background.js. Background Service Worker mengambil nilai bahasa yang telah tersimpan melalui fungsi getConfig(), kemudian mengirimkannya sebagai parameter target_language pada permintaan menuju Backend Flask. Parameter tersebut digunakan sebagai acuan dalam proses penerjemahan teks pada gambar.

Tabel 4.5 Hasil Implementasi Pengaturan Bahasa

Gambar original	Hasil terjemahan Bahasa Indonesia	Hasil terjemahan Bahasa Inggris
		

Berdasarkan Tabel 4.5, fitur Pengaturan Bahasa berhasil menerjemahkan ke beberapa pilihan bahasa pengguna dan dengan implementasi tersebut, hasil terjemahan dapat disesuaikan dengan bahasa tujuan yang dipilih pengguna.

4.2.3 Implementasi Modul Sistem

Modul sistem merupakan komponen inti yang bertanggung jawab dalam memproses setiap permintaan yang diterima dari Chrome Extension. Modul ini mengelola berbagai proses, seperti pengolahan data hasil scraping, penerjemahan gambar, ringkasan produk, hingga optimasi proses menggunakan Firefly Algorithm.

Implementasi modul sistem pada penelitian ini menerapkan arsitektur yang bersifat modular sehingga setiap modul memiliki tanggung jawab yang berbeda namun saling terintegrasi dalam satu alur pemrosesan. Pendekatan tersebut mempermudah proses pengembangan, pemeliharaan, serta memungkinkan setiap modul dikembangkan secara independen tanpa memengaruhi modul lainnya.

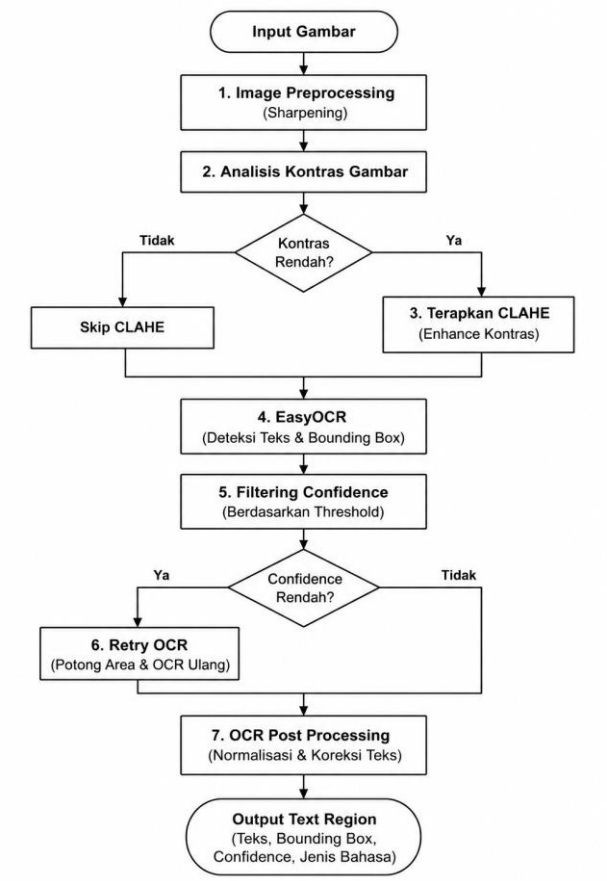
Komunikasi antar modul dilakukan melalui pemanggilan fungsi pada backend, sedangkan pertukaran data dengan Chrome Extension menggunakan layanan REST API dalam format JSON.

Implementasi modul sistem pada penelitian ini meliputi modul OCR dan Image Processing, modul Google Translate API, modul Google Gemini API, modul Firefly Algorithm, serta modul REST API dan pengelolaan basis data. Implementasi masing-masing modul dijelaskan pada subbab berikut.

4.2.3.1 Implementasi Modul OCR

Modul *Optical Character Recognition* (OCR) diimplementasikan untuk mendeteksi serta mengekstraksi teks yang terdapat pada gambar produk dari platform 1688.com. Implementasi modul ini menggunakan library EasyOCR dengan konfigurasi bahasa Mandarin Simplified (ch_sim) dan bahasa Inggris (en) sehingga mampu mengenali karakter yang umum ditemukan pada katalog produk. Sebelum proses OCR dilakukan, sistem terlebih dahulu menerapkan beberapa tahapan prapemrosesan citra (*image preprocessing*) untuk meningkatkan kualitas gambar sehingga proses pengenalan karakter dapat dilakukan dengan lebih optimal. Selain itu, modul OCR pada penelitian ini juga mengimplementasikan mekanisme Retry OCR, OCR Post Processing, serta penggunaan parameter hasil optimasi Firefly Algorithm untuk meningkatkan efisiensi proses ekstraksi teks.

Tahapan implementasi modul OCR ditunjukkan pada Gambar 4.17.



Gambar 4.17 Implementasi Modul OCR

Berdasarkan Gambar 4.17, proses OCR diawali dengan menerima masukan berupa gambar produk yang diperoleh dari proses pengunduhan gambar pada platform 1688.com. Gambar tersebut selanjutnya memasuki tahap Image Preprocessing untuk meningkatkan kualitas citra sebelum dilakukan proses pendeteksian teks. Pada implementasi ini, sistem menerapkan proses image sharpening untuk mempertajam detail citra sehingga karakter teks lebih mudah dikenali oleh modul OCR.

Setelah proses *preprocessing* selesai, sistem melakukan Analisis Kontras terhadap gambar. Tahap ini bertujuan menentukan apakah gambar memerlukan peningkatan kontras menggunakan metode Contrast Limited Adaptive Histogram Equalization (CLAHE). Apabila gambar telah memiliki tingkat kontras yang baik, proses CLAHE dilewati sehingga waktu pemrosesan dapat dikurangi. Sebaliknya, apabila kontras gambar rendah, sistem akan menerapkan CLAHE sebelum proses OCR dilakukan. Pendekatan ini memungkinkan proses peningkatan kualitas citra dilakukan secara adaptif sesuai karakteristik gambar yang diproses.

Tahap berikutnya adalah proses EasyOCR, yaitu proses pendeteksian posisi teks (*bounding box*) sekaligus pengenalan karakter pada gambar. Pada implementasinya, EasyOCR menghasilkan informasi berupa posisi teks, isi teks,

serta nilai *confidence* untuk setiap hasil deteksi. Selanjutnya sistem melakukan Filtering Confidence untuk menyaring hasil OCR berdasarkan nilai *confidence threshold* yang telah ditentukan sehingga hanya hasil deteksi yang memenuhi kriteria yang digunakan pada proses berikutnya.

Apabila sistem menemukan region teks dengan nilai *confidence* rendah, maka dijalankan mekanisme Retry OCR. Pada tahap ini sistem melakukan pemotongan ulang (*cropping*) pada area teks berdasarkan koordinat *bounding box* kemudian menjalankan proses OCR kembali pada area tersebut. Hasil OCR ulang akan digunakan apabila memiliki nilai *confidence* yang lebih tinggi dibandingkan hasil deteksi sebelumnya. Pendekatan ini diterapkan untuk meningkatkan akurasi pengenalan teks, khususnya pada teks Mandarin yang berukuran kecil atau memiliki kualitas gambar yang kurang baik.

Tahap terakhir adalah OCR Post Processing, yaitu proses normalisasi hasil OCR untuk memperbaiki karakter yang tidak sesuai, menghilangkan karakter yang tidak diperlukan, serta menghasilkan keluaran yang lebih konsisten sebelum diteruskan ke modul translasi. Hasil akhir modul OCR berupa kumpulan informasi yang terdiri atas isi teks, koordinat *bounding box*, nilai *confidence*, serta jenis bahasa pada setiap region teks. Informasi tersebut selanjutnya digunakan oleh modul Google Translate API dan modul Image Text Swapping sebagai dasar proses penerjemahan gambar.

Kode Program 4.12 Implementasi Image Preprocessing

```
class ImagePreprocessor:

    @staticmethod
    def preprocess(image: np.ndarray) -> np.ndarray:
        kernel_sharpen = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]])
        return cv2.filter2D(image, -1, kernel_sharpen)

    @staticmethod
    def enhance_for_ocr(image: np.ndarray) -> np.ndarray:
        lab = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
        l, a, b = cv2.split(lab)
        clahe = cv2.createCLAHE(clipLimit=3.0, tileGridSize=(8,8))
        enhanced_lab = cv2.merge((clahe.apply(l), a, b))
        return cv2.cvtColor(enhanced_lab, cv2.COLOR_LAB2BGR)
```

Kode Program 4.12 menunjukkan implementasi tahap *Image Preprocessing*. Sistem terlebih dahulu menerapkan metode *image sharpening* menggunakan fungsi filter2D() untuk mempertajam detail citra sehingga karakter teks lebih mudah dikenali. Selanjutnya fungsi enhance_for_ocr() menerapkan

metode *Contrast Limited Adaptive Histogram Equalization* (CLAHE) pada kanal luminansi citra untuk meningkatkan kontras gambar yang memiliki kualitas rendah sebelum diproses oleh modul OCR.

Kode Program 4.13 Implementasi EasyOCR

```
results = self.reader.readtext(
            enhanced,
            detail=1,
            paragraph=False,
            contrast_ths=0.05,
            adjust_contrast=0.7,
            text_threshold=0.6,
            low_text=0.30,
            link_threshold=0.30,
            mag_ratio=self.fa_mag_ratio,
            canvas_size=self.fa_canvas_size,
            min_size=5,
            add_margin=0.2,
            width_ths=0.3,
            height_ths=0.3,
        )
```

Kode Program 4.13 menunjukkan implementasi proses ekstraksi teks menggunakan library EasyOCR. Sistem memproses gambar hasil *preprocessing* untuk mendeteksi posisi teks (*bounding box*), mengenali karakter, serta menghasilkan nilai *confidence*. Implementasi menggunakan parameter seperti text_threshold, low_text, link_threshold, mag_ratio, dan canvas_size untuk menyesuaikan proses deteksi terhadap karakteristik gambar produk. Nilai mag_ratio dan canvas_size diperoleh dari hasil optimasi Firefly Algorithm sehingga konfigurasi OCR menjadi lebih optimal.

Kode Program 4.14 Implementasi Retry OCR

```
need_retry = (
                region.is_chinese
                and len(region.text) <= 3
                and region.confidence < 0.30
            )

            if need_retry and retry_count < MAX_RETRY:
                retry_count += 1
                t = time.perf_counter()
                retry = self._retry_small_region(image, region)
                logger.info(
                    f"[PERF] Retry OCR : {time.perf_counter()-
t:.2f}s"
                )
```

```
            if retry.confidence > region.confidence:
                region = retry
```

Kode Program 4.14 menunjukkan implementasi mekanisme *Retry OCR*. Sistem terlebih dahulu mengevaluasi hasil OCR berdasarkan panjang teks dan nilai *confidence*. Apabila region teks Mandarin memiliki *confidence* rendah, sistem menjalankan fungsi _retry_small_region() untuk melakukan proses *cropping* pada area *bounding box* dan menjalankan EasyOCR kembali pada area tersebut. Hasil OCR ulang digunakan apabila menghasilkan nilai *confidence* yang lebih tinggi dibandingkan hasil sebelumnya sehingga akurasi pengenalan karakter dapat ditingkatkan tanpa mengulang proses OCR terhadap keseluruhan gambar.

Tabel 4.6 Hasil Deteksi Bounding Box Menggunakan EasyOCR

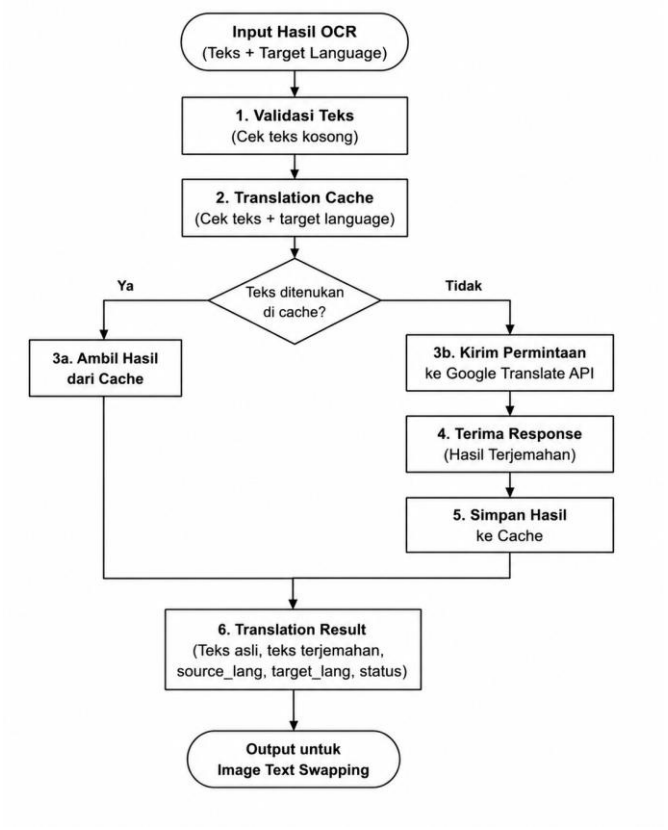
	INFO:pipeline.ocr_extractor:Ditemukan 6 region teks (6 Mandarin, 0 non-Mandarin) ========== Jumlah teks terdeteksi : 6 ========== Region 1 Teks : 活力橙 Bahasa : Mandarin BoundingBox : (62, 279) -> (170, 331) ---------- Region 2 Teks : 柠檬黄 Bahasa : Mandarin BoundingBox : (319, 279) -> (427, 330) ---------- Region 3 Teks : 馥郁红 Bahasa : Mandarin BoundingBox : (560, 279) -> (670, 330) ---------- Region 4 Teks : 墨绿色 Bahasa : Mandarin BoundingBox : (63, 609) -> (168, 659) ---------- Region 5 Teks : 巴黎蓝 Bahasa : Mandarin BoundingBox : (323, 609) -> (427, 659) ---------- Region 6 Teks : 梦幻黑 Bahasa : Mandarin BoundingBox : (560, 608) -> (667, 659) ----------

Berdasarkan hasil implementasi, modul OCR mampu mendeteksi posisi teks (*bounding box*) sekaligus mengekstraksi karakter Mandarin yang terdapat pada gambar produk. Informasi yang dihasilkan meliputi isi teks, koordinat *bounding box*, nilai *confidence*, serta jenis bahasa pada setiap region teks. Hasil tersebut selanjutnya digunakan sebagai masukan pada modul Google Translate API untuk proses penerjemahan dan modul Image Text Swapping untuk proses penggantian teks pada gambar.

4.2.3.2 Implementasi Modul Translasi

Modul Translasi diimplementasikan untuk menerjemahkan teks hasil ekstraksi dari modul OCR ke dalam bahasa tujuan yang dipilih oleh pengguna melalui fitur Pengaturan Bahasa pada Chrome Extension. Implementasi modul ini menggunakan layanan Google Translate API yang diakses melalui protokol HTTP menggunakan library Requests pada Python. Modul translasi menerima masukan berupa teks hasil OCR beserta konfigurasi bahasa tujuan, kemudian menghasilkan teks terjemahan yang selanjutnya digunakan oleh modul Image Text Swapping untuk menggantikan teks pada gambar produk.

Untuk meningkatkan efisiensi proses translasi, implementasi modul ini menerapkan mekanisme translation cache, connection pooling, batch translation, dan multithreading sehingga proses penerjemahan beberapa region teks dapat dilakukan secara lebih cepat tanpa melakukan permintaan yang berulang terhadap teks yang sama. Tahapan implementasi modul translasi ditunjukkan pada Gambar 4.18.



Gambar 4.18 Implementasi Modul Translasi

Berdasarkan Gambar 4.18, proses translasi diawali dengan menerima masukan berupa hasil ekstraksi teks dari modul OCR. Sistem terlebih dahulu melakukan Validasi Teks untuk memastikan bahwa teks yang diterima tidak kosong sebelum proses penerjemahan dilakukan. Apabila teks tidak memenuhi kriteria, sistem akan mengembalikan teks asli sebagai hasil keluaran.

Selanjutnya sistem memeriksa Translation Cache untuk mengetahui apakah teks yang sama pernah diterjemahkan sebelumnya. Apabila hasil terjemahan telah tersedia (*cache hit*), sistem langsung menggunakan hasil tersebut tanpa mengirimkan permintaan ke Google Translate API. Sebaliknya, apabila hasil translasi belum tersedia (*cache miss*), sistem mengirimkan permintaan ke Google Translate API menggunakan parameter bahasa sumber dan bahasa tujuan yang telah ditentukan.

Setelah proses translasi selesai, hasil terjemahan disimpan kembali ke dalam cache sehingga dapat digunakan pada proses berikutnya apabila ditemukan teks yang sama. Tahap terakhir menghasilkan objek Translation Result yang berisi teks asli, hasil terjemahan, bahasa sumber, bahasa tujuan, serta status proses translasi. Informasi tersebut selanjutnya diteruskan ke modul Image Text Swapping untuk menggantikan teks pada gambar produk.

Kode Program 4.15 Implementasi Google Translate API

```
def translate(self, text, target_lang="id", source_lang="zh-CN"):

    # Validasi teks
    if not text or not text.strip():
        return TranslationResult(...)
    # Pemeriksaan translation cache
    cache_key = (text, target_lang)
    with self.cache_lock:
        if cache_key in self.cache:
            return self.cache[cache_key]
    # Parameter Google Translate API
    params = {"client":"gtx", "sl":source_lang, "tl":target_lang, "dt":"t",
"q":text}
    # Mengirim permintaan ke Google Translate API
    response = self.session.get(self.BASE_URL, params=params, timeout=3)
    translated = "".join(segment[0] for segment in response.json()[0] if
segment[0])
    result = TranslationResult(
        original_text=text, translated_text=translated,
        source_lang=source_lang, target_lang=target_lang,
        method="google", success=True
    )
    # Menyimpan hasil translasi ke cache
    self.cache[cache_key] = result

    return result
```

Kode Program 4.15 menunjukkan implementasi proses translasi menggunakan Google Translate API pada file translator.py. Sistem terlebih dahulu melakukan validasi terhadap teks masukan kemudian memeriksa translation cache untuk mengetahui apakah hasil terjemahan telah tersedia. Apabila hasil translasi belum ditemukan pada cache, sistem membentuk parameter permintaan yang terdiri atas bahasa sumber (sl), bahasa tujuan (tl), serta teks yang akan diterjemahkan (q). Selanjutnya sistem mengirimkan permintaan HTTP menggunakan metode GET melalui objek Session sehingga koneksi dapat digunakan kembali (*connection pooling*) pada proses translasi berikutnya. Setelah menerima respons dari Google Translate API, sistem mengekstraksi hasil terjemahan dari data JSON kemudian membentuk objek TranslationResult yang berisi teks asli, hasil terjemahan, bahasa sumber, bahasa tujuan, metode translasi, serta status keberhasilan proses. Hasil translasi tersebut kemudian disimpan ke dalam translation cache agar dapat digunakan kembali apabila ditemukan teks yang sama pada proses berikutnya. Selain itu, sistem juga menerapkan mekanisme *fallback*, yaitu mengembalikan teks asli apabila terjadi kesalahan komunikasi dengan layanan Google Translate API sehingga proses selanjutnya tetap dapat dijalankan.

Kode Program 4.16 Implementasi Batch Translation

```
def translate_batch(self, texts, target_lang="id", source_lang="zh-CN"):

    # Menghilangkan teks yang sama
    unique_texts = list(dict.fromkeys(texts))
    def worker(text):
        return self.translate(text=text, target_lang=target_lang,
                              source_lang=source_lang,
session=self.session)
    # Menjalankan translasi secara paralel
    with ThreadPoolExecutor(max_workers=min(4, len(unique_texts))) as
executor:
        results = list(executor.map(worker, unique_texts))
    # Memetakan hasil translasi
    translation_map = {r.original_text: r for r in results}
    # Mengembalikan hasil sesuai urutan OCR
    return [translation_map[text] for text in texts]
```

Kode Program 4.16 menunjukkan implementasi proses Batch Translation. Sebelum proses translasi dilakukan, sistem terlebih dahulu menghilangkan teks yang memiliki nilai sama sehingga hanya teks unik yang dikirimkan ke Google Translate API. Pendekatan ini bertujuan mengurangi jumlah permintaan ke layanan translasi dan meningkatkan efisiensi proses. Selanjutnya sistem menggunakan ThreadPoolExecutor untuk menjalankan beberapa proses translasi secara paralel (*multithreading*) sehingga waktu pemrosesan dapat dipersingkat ketika gambar mengandung banyak region teks. Setelah seluruh proses translasi selesai, hasil terjemahan dipetakan kembali sesuai urutan hasil OCR sehingga hubungan antara teks, posisi *bounding box*, dan hasil terjemahan tetap terjaga sebelum diteruskan ke modul Image Text Swapping.

Selanjutnya sistem menggunakan ThreadPoolExecutor untuk menjalankan beberapa proses translasi secara paralel sehingga waktu pemrosesan dapat dipersingkat ketika gambar mengandung banyak region teks. Setelah seluruh proses translasi selesai, hasil terjemahan dipetakan kembali sesuai urutan hasil OCR sehingga hubungan antara teks, posisi *bounding box*, dan hasil terjemahan tetap terjaga sebelum diteruskan ke modul Image Text Swapping.

Tabel 4.7 Hasil Implementasi Modul Translasi

	Ditemukan 7 region teks (7 Mandarin, 0 non-Mandarin) Ditemukan 7 teks ========== GOOGLE TRANSLATE OCR : 低功耗强续航 Translated : Konsumsi daya rendah dan masa pakai baterai yang lama ========== GOOGLE TRANSLATE OCR : 7天畅快用 Translated : Nikmati selama 7 hari ========== GOOGLE TRANSLATE OCR : 蓝牙低功耗芯片。耗能低， Translated : Chip Bluetooth hemat energi. Konsumsi energi rendah, ========== GOOGLE TRANSLATE OCR : 性能稳定； Translated : Kinerja yang stabil; ========== GOOGLE TRANSLATE OCR : 一次充电 Translated : Satu biaya ========== GOOGLE TRANSLATE OCR : 可连续听歌12小时7天长待机， Translated : Dapat mendengarkan musik terus menerus selama 12 jam 7 hari dengan waktu standby yang lama. ========== GOOGLE TRANSLATE OCR : 满足长途出行 Translated : Cocok untuk perjalanan jarak jauh

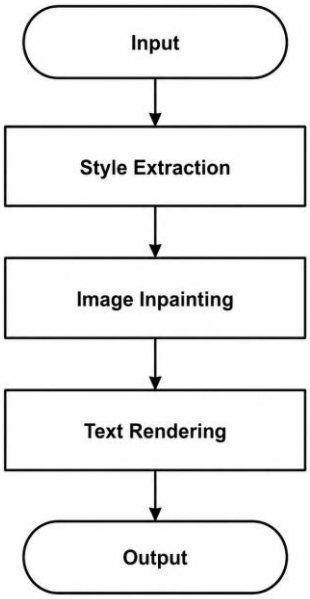
Berdasarkan hasil implementasi, modul translasi mampu menerjemahkan teks Mandarin ke dalam bahasa tujuan secara otomatis dengan tetap mempertahankan urutan hasil OCR. Penerapan mekanisme cache, batch translation, dan multithreading membantu mengurangi permintaan yang berulang ke Google Translate API sehingga proses translasi menjadi lebih efisien. Hasil terjemahan

yang diperoleh selanjutnya digunakan sebagai masukan bagi modul Image Text Swapping untuk menggantikan teks Mandarin pada gambar produk.

4.2.3.3 Implementasi Image Text Swapping

Modul *Image Text Swapping* diimplementasikan untuk menggantikan teks berbahasa Mandarin pada gambar produk dengan hasil terjemahan yang diperoleh dari modul Google Translate API. Implementasi modul ini dikembangkan menggunakan pustaka OpenCV dan Pillow (PIL) dengan menerapkan tiga tahapan utama, yaitu Style Extraction, Image Inpainting, dan Text Rendering. Pendekatan tersebut bertujuan agar hasil terjemahan dapat ditempatkan pada posisi yang sama dengan teks asli serta mempertahankan karakteristik visual gambar produk.

Selain melakukan penggantian teks, modul ini juga menyesuaikan warna tulisan, ukuran huruf, ketebalan teks, serta posisi penempatan berdasarkan karakteristik visual yang diekstraksi dari gambar asli. Dengan demikian, gambar hasil translasi tetap memiliki tata letak yang menyerupai katalog produk asli. Tahapan implementasi modul *Image Text Swapping* ditunjukkan pada Gambar 4.19.



Gambar 4.19 Alur Implementasi *Image Text Swapping*

Berdasarkan Gambar 4.19, proses *Image Text Swapping* diawali dengan menerima masukan berupa gambar asli, hasil OCR yang berisi koordinat *bounding box*, serta hasil translasi dari modul Google Translate API. Informasi tersebut digunakan sebagai dasar dalam proses penggantian teks pada gambar produk.

Tahap pertama adalah Style Extraction, yaitu proses mengekstraksi karakteristik visual dari area teks asli berdasarkan koordinat *bounding box*. Pada tahap ini sistem menganalisis warna teks, warna latar belakang, ukuran huruf, serta ketebalan tulisan sehingga karakteristik tersebut dapat digunakan kembali ketika proses penulisan teks hasil terjemahan dilakukan.

Tahap berikutnya adalah Image Inpainting, yaitu proses menghapus teks Mandarin pada gambar menggunakan metode Telea Inpainting yang tersedia pada pustaka OpenCV. Sistem membentuk *mask* berdasarkan koordinat *bounding box*, kemudian merekonstruksi area tersebut menggunakan informasi piksel di sekitarnya sehingga menghasilkan latar belakang yang siap digunakan untuk proses penempatan teks baru.

Tahap terakhir adalah Text Rendering, yaitu proses menggambar hasil terjemahan pada area yang telah dibersihkan. Implementasi tahap ini menggunakan pustaka Pillow (PIL) karena mendukung pengaturan ukuran huruf, warna teks, serta proses *font fitting* secara otomatis. Sistem menyesuaikan posisi dan ukuran teks berdasarkan dimensi *bounding box* sehingga hasil terjemahan dapat ditempatkan secara proporsional pada gambar produk. Hasil akhir dari modul ini berupa gambar katalog yang telah menggantikan teks Mandarin dengan hasil terjemahan sesuai bahasa tujuan.

Kode Program 4.17 Implementasi Style Extraction

```
class StyleExtractor:
    def extract(self, image_bgr, bbox):

        xs = [p[0] for p in bbox]; ys = [p[1] for p in bbox]
        x1, y1 = max(0, min(xs)), max(0, min(ys))
        x2, y2 = min(image_bgr.shape[1], max(xs)), min(image_bgr.shape[0],
max(ys))

        region = image_bgr[y1:y2, x1:x2]
        bg_color = np.median(region.reshape(-1,3), axis=0)
        gray = cv2.cvtColor(region, cv2.COLOR_BGR2GRAY)
        mask = cv2.adaptiveThreshold(
            gray, 255,
            cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
            cv2.THRESH_BINARY_INV,
            25, 10
        )
        return TextStyle(
            font_color=(int(font[2]), int(font[1]), int(font[0])),
            bg_color=(int(bg_color[2]), int(bg_color[1]),
int(bg_color[0])),
            font_size=max(12, int((y2-y1)*0.8)),
            is_bold=is_bold
```

```
        )
```

Kode Program 4.17 menunjukkan implementasi proses Style Extraction pada file image_swapper.py. Sistem memanfaatkan koordinat *bounding box* hasil OCR untuk mengambil area teks kemudian melakukan analisis terhadap warna latar belakang, warna huruf, ukuran teks, serta ketebalan tulisan. Informasi tersebut selanjutnya disimpan dalam objek TextStyle dan digunakan kembali pada proses Text Rendering agar karakteristik visual gambar tetap dipertahankan.

Kode Program 4.18 Implementasi Image Inpainting

```
def clean_text_areas(self, image, instructions):
    result = image.copy()
    for instr in instructions:
        region = result[y:y2, x:x2]
        gray = cv2.cvtColor(region, cv2.COLOR_BGR2GRAY)
        mask = cv2.threshold(
            cv2.absdiff(gray, int(np.median(gray))), 25, 255,
cv2.THRESH_BINARY
        )[1]

        mask = cv2.dilate(mask, np.ones((3,3), np.uint8), iterations=2)
        result[y:y2, x:x2] = cv2.inpaint(
            region, mask, 3, cv2.INPAINT_TELEA
        )
    return result
```

Kode Program 4.18 menunjukkan implementasi proses Image Inpainting pada file image_swapper.py. Sistem membentuk *mask* berdasarkan area *bounding box* kemudian menerapkan metode Telea Inpainting yang tersedia pada pustaka OpenCV untuk merekonstruksi area yang sebelumnya berisi teks Mandarin. Hasil dari proses ini berupa latar belakang gambar yang telah dibersihkan sehingga siap digunakan pada proses penempatan teks hasil terjemahan.

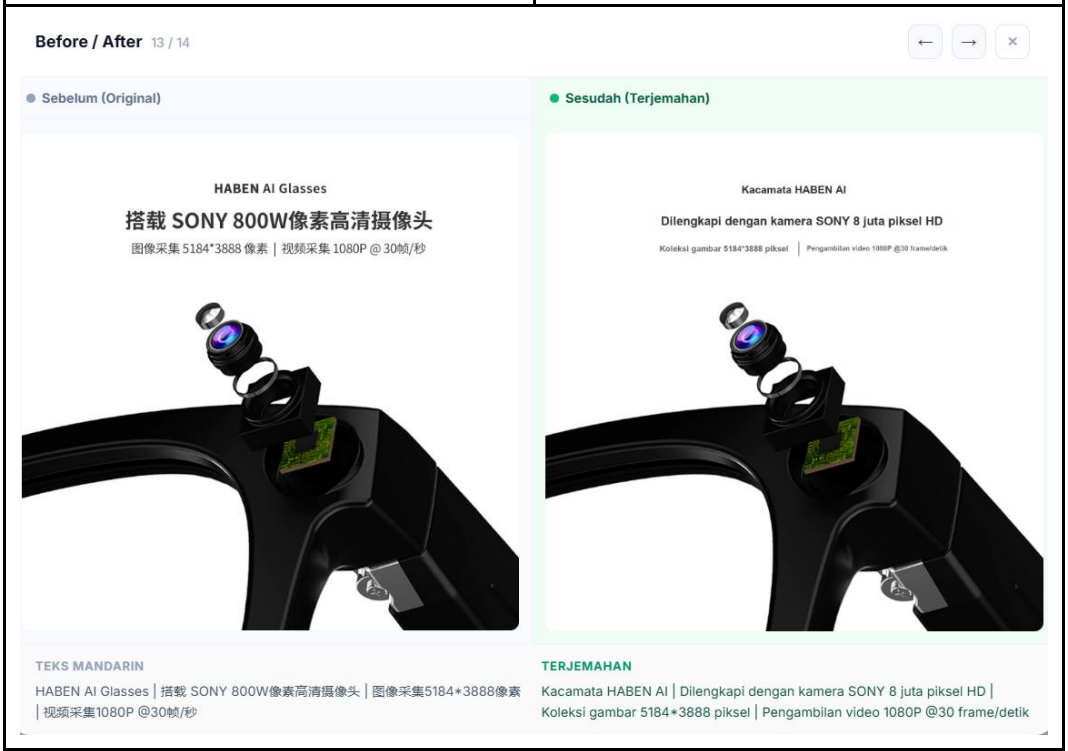
Kode Program 4.19 Implementasi Text Rendering

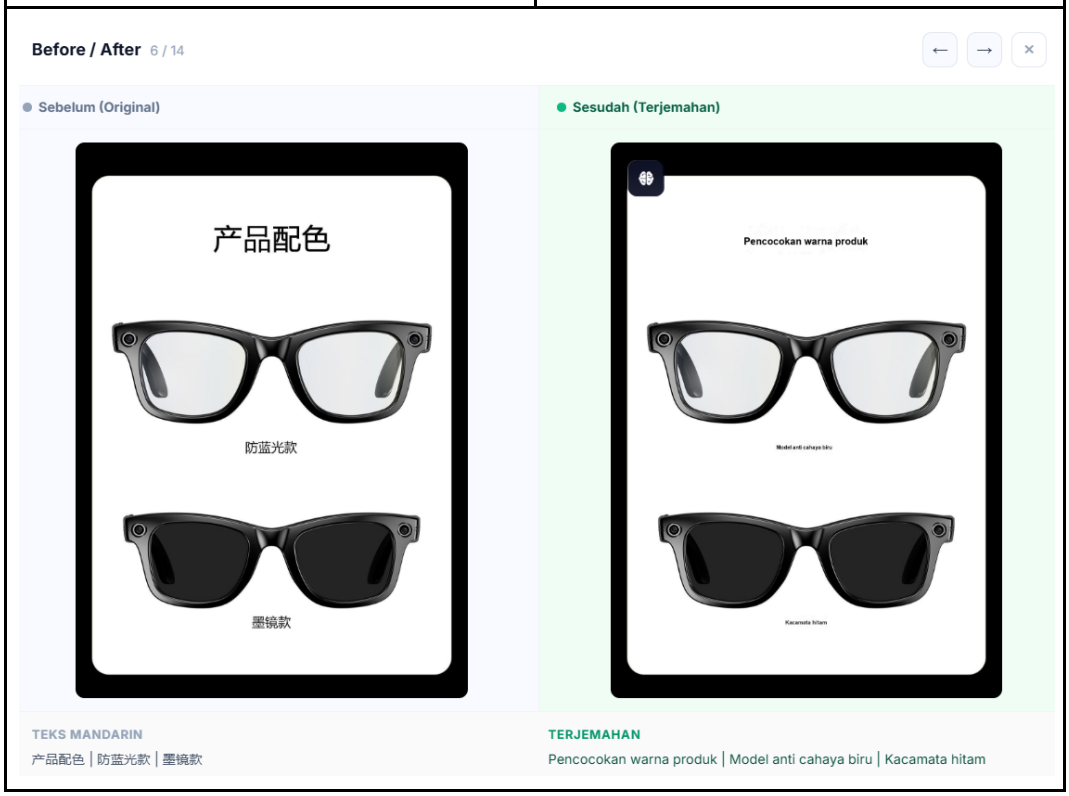
```
def render(self, image_cv, instructions, styles):

    img_pil = Image.fromarray(
        cv2.cvtColor(image_cv, cv2.COLOR_BGR2RGB)
    ).convert("RGBA")

    overlay = Image.new("RGBA", img_pil.size, (0,0,0,0))
    draw = ImageDraw.Draw(overlay)

    for instr, style in zip(instructions, styles):
        size, font = self._fit_font(
```

```
            instr.translated_text, w, h, style.is_bold
        )
        draw.text(
            (x+(w-tw)//2, y+(h-th)//2), instr.translated_text,
             fill=(*style.font_color,255), font=font
        )

    final_img = Image.alpha_composite(img_pil, overlay)

    return cv2.cvtColor(
        np.array(final_img.convert("RGB")), cv2.COLOR_RGB2BGR
    )
```

Kode Program 4.19 menunjukkan implementasi proses Text Rendering pada file image_swapper.py menggunakan pustaka Pillow (PIL). Sistem terlebih dahulu menyesuaikan ukuran huruf melalui mekanisme *font fitting* agar sesuai dengan dimensi *bounding box*. Selanjutnya hasil terjemahan digambar pada posisi yang sesuai menggunakan warna, ukuran, dan ketebalan huruf yang diperoleh dari proses Style Extraction. Setelah seluruh teks berhasil dirender, sistem menggabungkan lapisan teks dengan gambar hasil Image Inpainting menggunakan Image.alpha_composite() sehingga menghasilkan gambar katalog yang telah diterjemahkan.

Tabel 4.8 Hasil Implementasi Modul Image Text Swapping

Sebelum	Sesudah
	

Sebelum	Sesudah
	

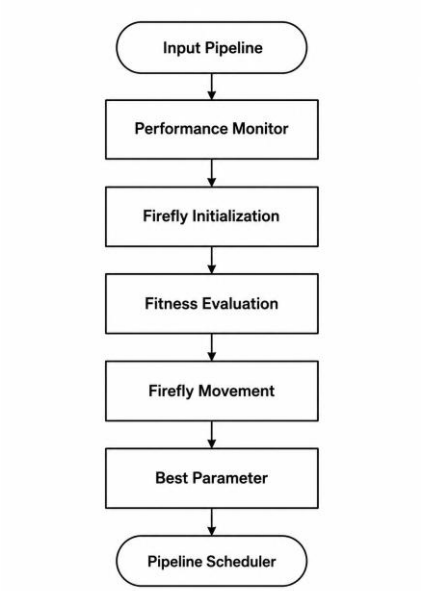
Berdasarkan hasil implementasi, modul *Image Text Swapping* mampu menghasilkan gambar katalog dengan teks berbahasa Indonesia tanpa mengubah tata letak utama pada gambar produk. Karakteristik visual seperti posisi teks, ukuran huruf, dan warna tulisan tetap dipertahankan melalui proses Style Extraction, Image Inpainting, dan Text Rendering, sehingga hasil translasi dapat dibaca dengan lebih baik oleh pengguna.

4.2.3.4 Implementasi Firefly Algorithm

Modul Firefly Algorithm diimplementasikan untuk mengoptimasi konfigurasi *pipeline scheduler* yang mengatur jumlah proses yang dapat berjalan secara bersamaan pada setiap tahapan pemrosesan gambar. Berbeda dengan implementasi Firefly Algorithm yang digunakan untuk optimasi pencarian solusi secara langsung, pada penelitian ini Firefly Algorithm digunakan untuk menentukan konfigurasi *scheduler* yang paling sesuai berdasarkan kondisi pemrosesan yang sedang berlangsung. Dengan pendekatan tersebut, proses pengunduhan gambar, OCR, translasi, dan *Image Text Swapping* dapat berjalan dengan penggunaan sumber daya yang lebih seimbang.

Implementasi Firefly Algorithm dijalankan secara adaptif ketika sistem mendeteksi perubahan kondisi pipeline melalui *Performance Monitor*. Selanjutnya algoritma mengevaluasi beberapa kandidat konfigurasi menggunakan fungsi *fitness*

untuk memperoleh konfigurasi dengan nilai *fitness* terbaik. Konfigurasi tersebut kemudian diterapkan pada *pipeline scheduler* tanpa menghentikan proses yang sedang berjalan. Tahapan implementasi Firefly Algorithm ditunjukkan pada Gambar 4.20.



Gambar 4.20 Diagram Implementasi Firefly Algorithm

Berdasarkan Gambar 4.20, implementasi Firefly Algorithm diawali dengan menerima kondisi terkini dari pipeline scheduler. Informasi tersebut diperoleh melalui Performance Monitor yang memantau kondisi setiap tahapan pemrosesan, seperti jumlah antrean (queue), jumlah proses aktif, throughput, serta jumlah proses yang mengalami retry. Informasi tersebut digunakan sebagai dasar untuk menentukan apakah proses optimasi perlu dijalankan.

Apabila proses optimasi diperlukan, sistem melakukan Firefly Initialization dengan membangkitkan sejumlah kandidat konfigurasi *scheduler*. Setiap kandidat merepresentasikan kombinasi nilai parameter yang terdiri atas download_limit, ocr_limit, translate_limit, dan swap_limit. Parameter tersebut menentukan jumlah proses yang diizinkan berjalan secara bersamaan pada setiap tahapan pipeline.

Tahap berikutnya adalah Fitness Evaluation, yaitu proses menghitung nilai *fitness* dari setiap kandidat konfigurasi. Pada implementasi ini, fungsi *fitness* mempertimbangkan beberapa indikator kinerja pipeline, yaitu *throughput*, kesesuaian konfigurasi terhadap kondisi antrean (*parameter matching*), efisiensi penggunaan sumber daya (*resource efficiency*), serta tingkat *retry* selama proses berlangsung. Nilai *fitness* tersebut digunakan sebagai dasar dalam menentukan tingkat kecerahan (*brightness*) setiap *firefly*.

Selanjutnya dilakukan proses Firefly Movement, yaitu perpindahan posisi *firefly* menuju kandidat dengan nilai *fitness* yang lebih baik. Proses perpindahan dilakukan secara iteratif menggunakan parameter alpha, beta, dan gamma hingga diperoleh konfigurasi terbaik atau memenuhi kondisi *early stopping*. Hasil optimasi kemudian menghasilkan Best Parameter yang digunakan untuk memperbarui konfigurasi *pipeline scheduler*. Dengan pendekatan tersebut, sistem dapat menyesuaikan jumlah proses yang berjalan secara bersamaan sesuai kondisi pipeline tanpa menghentikan proses pemrosesan gambar yang sedang berlangsung.

Kode Program 4.20 Implementasi Parameter Firefly

```
@dataclass
class FAParameter:

    download_limit: int = 3
    ocr_limit: int = 2
    translate_limit: int = 3
    swap_limit: int = 2

    BOUNDS = {'download_limit':(1,5), 'ocr_limit':(1,2),
'translate_limit':(1,5), 'swap_limit':(1,3)}

    def to_vector(self):
        return [self.download_limit, self.ocr_limit, self.translate_limit,
self.swap_limit]

    @classmethod
    def random(cls):
        return cls(random.randint(1,5), random.randint(1,2),
random.randint(1,5), random.randint(1,3))
```

Kode Program 4.20 menunjukkan implementasi parameter yang dioptimasi oleh Firefly Algorithm. Setiap objek FAParameter merepresentasikan satu kandidat solusi berupa konfigurasi *Pipeline Scheduler* yang terdiri atas empat parameter. Masing-masing parameter menunjukkan jumlah maksimum *worker* yang dapat dijalankan secara bersamaan pada setiap tahapan *pipeline*. Deskripsi parameter yang digunakan dalam proses optimasi ditunjukkan pada Tabel 4.9.

Tabel 4.9 Parameter yang Dioptimasi Firefly Algorithm

Parameter	Tahapan Pipeline	Rentang Nilai
download_limit	Jumlah maksimum Download Worker yang dapat berjalan secara bersamaan.	1–5

Parameter	Tahapan Pipeline	Rentang Nilai
ocr_limit	Jumlah maksimum OCR Worker yang dapat berjalan secara bersamaan.	1–2
translate_limit	Jumlah maksimum Translation Worker yang dapat berjalan secara bersamaan.	1–5
swap_limit	Jumlah maksimum Image Text Swapping Worker yang dapat berjalan secara bersamaan.	1–3

Selain menyimpan nilai setiap parameter, kelas FAParameter juga mendefinisikan batas nilai (*bounds*) yang digunakan sebagai ruang pencarian (*search space*) dalam proses optimasi. Fungsi to_vector() digunakan untuk mengubah konfigurasi parameter ke dalam bentuk vektor sebagai masukan Firefly Algorithm, sedangkan fungsi random() digunakan untuk membangkitkan konfigurasi awal secara acak pada tahap inisialisasi populasi *firefly*.

Kode Program 4.21 Implementasi Fitness Function

```
def compute_fitness(snapshot, param):

    throughput = snapshot.get("throughput", 0.0)
    f_throughput = min(1.0, throughput / 2.0)
    f_param_match = ...
    f_resource = ...

    total_retry = sum(stages.get(s, {}).get("retry", 0) for s in
stage_keys)
    total_done = sum(stages.get(s, {}).get("completed", 0) for s in
stage_keys)

    retry_rate = total_retry / max(total_done, 1)
    f_retry = max(0.0, 1.0 - retry_rate)

    fitness = 0.25*f_throughput + 0.40*f_param_match + 0.20*f_resource +
0.15*f_retry

    return round(fitness, 6)
```

Kode Program 4.21 menunjukkan implementasi fungsi *fitness* pada file firefly_optimizer.py yang digunakan untuk mengevaluasi kualitas setiap kandidat konfigurasi. Nilai *fitness* dihitung berdasarkan empat indikator utama, yaitu *throughput*, *parameter matching*, *resource efficiency*, dan *retry rate*. Setiap

indikator memiliki bobot yang berbeda sehingga menghasilkan nilai *fitness* sebagai dasar dalam menentukan tingkat kecerahan (*brightness*) setiap firefly. Semakin tinggi nilai *fitness*, semakin baik konfigurasi tersebut dalam mengoptimalkan proses pemrosesan gambar.

Kode Program 4.22 Implementasi Adaptive Firefly Optimizer

```
def _firefly_run(self, snapshot):
    pop = [FAParameter.random() for _ in range(self.n_fireflies)]
    fits = [compute_fitness(snapshot, p) for p in pop]

    for iteration in range(self.max_iter):
        for i in range(self.n_fireflies):
            for j in range(self.n_fireflies):
                if i == j or fits[j] <= fits[i]:
                    continue
                beta = self.beta_0 * math.exp(-self.gamma * r * r)
                v_i[d] = v_i[d] + beta * (v_j[d] - v_i[d]) + alpha * eps

        new_p = FAParameter.from_vector(v_i)
        new_f = compute_fitness(snapshot, new_p)

    return best_p, best_f
```

Kode Program 4.22 menunjukkan implementasi proses optimasi menggunakan Adaptive Firefly Optimizer. Secara konseptual, proses optimasi yang dilakukan terdiri atas beberapa tahapan sebagaimana ditunjukkan pada Tabel 4.10.

Tabel 4.10 Tahapan Adaptive Firefly Optimizer

Tahapan	Deskripsi
Inisialisasi Populasi	Membentuk sejumlah konfigurasi *worker* secara acak sebagai kandidat solusi awal menggunakan FAParameter.random().
Evaluasi Fitness	Menghitung nilai *fitness* setiap kandidat berdasarkan kondisi *pipeline* menggunakan fungsi compute_fitness().
Perbandingan Firefly	Membandingkan nilai *fitness* antar *firefly* untuk menentukan kandidat dengan kualitas solusi yang lebih baik.
Perpindahan Firefly	Memperbarui konfigurasi *worker* menuju *firefly* dengan nilai *fitness* yang lebih tinggi menggunakan persamaan Firefly Algorithm.

Tahapan	Deskripsi
Evaluasi Ulang	Menghitung kembali nilai *fitness* setelah konfigurasi diperbarui.
Seleksi Solusi Terbaik	Memilih konfigurasi dengan nilai *fitness* tertinggi sebagai hasil optimasi untuk diterapkan pada *Pipeline Scheduler*.

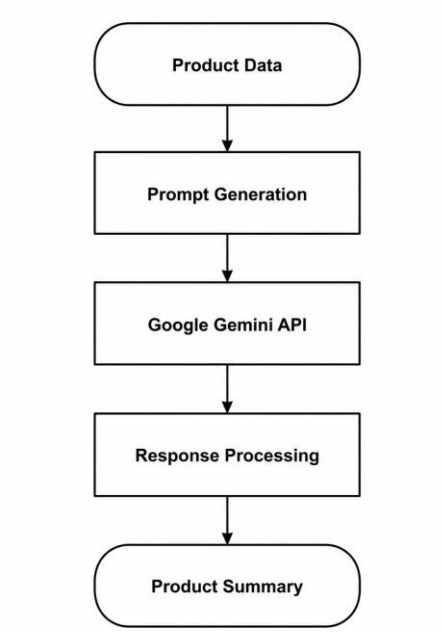
Berdasarkan tahapan tersebut, proses optimasi dilakukan secara iteratif hingga mencapai jumlah iterasi maksimum (*max_iter*). Setelah seluruh iterasi selesai, konfigurasi dengan nilai *fitness* tertinggi dipilih sebagai konfigurasi terbaik untuk memperbarui jumlah *Download Worker*, *OCR Worker*, *Translation Worker*, dan *Image Text Swapping Worker* pada *Pipeline Scheduler*.

Berdasarkan implementasi tersebut, Firefly Algorithm diintegrasikan ke dalam sistem sebagai modul yang bekerja bersama *Performance Monitor* dan *Pipeline Scheduler*. *Performance Monitor* menyediakan informasi kondisi *pipeline* sebagai masukan bagi proses optimasi, sedangkan hasil optimasi berupa konfigurasi *worker* terbaik diterapkan oleh *Pipeline Scheduler* untuk mengatur jumlah *Download Worker*, *OCR Worker*, *Translation Worker*, dan *Image Text Swapping Worker* selama proses berlangsung. Dengan mekanisme tersebut, sistem mampu menyesuaikan konfigurasi *worker* secara adaptif sesuai perubahan kondisi pemrosesan. Hasil pengujian terhadap modul Firefly Algorithm dibahas lebih lanjut pada Subbab 4.3.1.5.

4.2.3.5 Implementasi Modul Artificial Intelligence

Pada penelitian ini, Modul Artificial Intelligence diimplementasikan menggunakan Google Gemini API untuk menghasilkan ringkasan produk secara otomatis berdasarkan data hasil scraping dan OCR. Modul ini dijalankan pada sisi backend menggunakan framework Flask sehingga proses komunikasi dengan layanan Google Gemini dilakukan secara terpusat tanpa mengekspos API Key kepada Chrome Extension.

Pada implementasinya, modul menerima Product ID sebagai masukan, kemudian mengambil informasi produk dari basis data yang meliputi nama produk, harga, supplier, serta hasil OCR yang telah diterjemahkan. Informasi tersebut selanjutnya disusun menjadi sebuah prompt sebelum dikirimkan ke Google Gemini API. Hasil respons yang diterima kemudian diproses dan dikembalikan ke Chrome Extension dalam format JSON untuk ditampilkan kepada pengguna. Tahapan implementasi modul Google Gemini API ditunjukkan pada Gambar 4.21.



Gambar 4.21 Implementasi Modul Artificial Intelligence

Berdasarkan Gambar 4.21, proses diawali dengan mengambil data produk dari basis data menggunakan *Product ID* yang diterima oleh backend. Data yang diambil meliputi nama produk, harga, supplier, serta hasil OCR yang sebelumnya diperoleh dari proses ekstraksi teks pada gambar produk. Informasi tersebut digunakan sebagai sumber data untuk membangun permintaan kepada Google Gemini API.

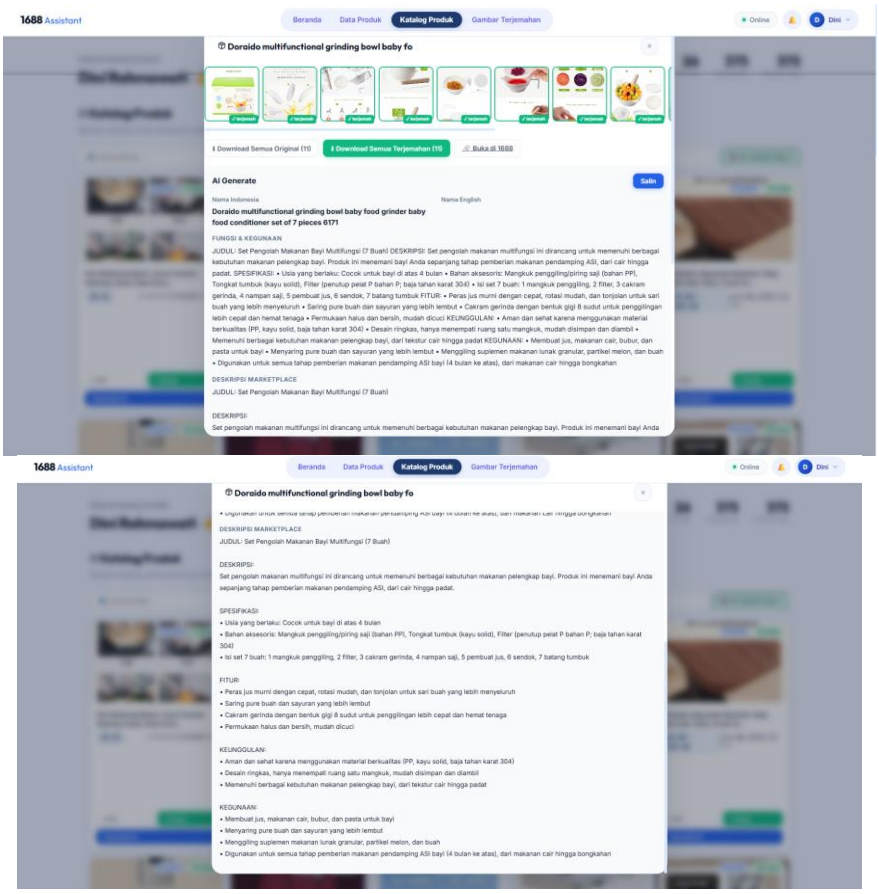
Selanjutnya sistem melakukan Prompt Generation, yaitu menyusun instruksi yang berisi informasi produk beserta format keluaran yang diharapkan. Prompt tersebut meminta Google Gemini menghasilkan judul produk, deskripsi, spesifikasi, fitur, keunggulan, dan kegunaan produk sesuai bahasa yang dipilih pengguna. Setelah prompt selesai disusun, backend mengirimkan permintaan ke layanan Google Gemini 2.5 Flash melalui metode HTTP POST menggunakan API Key yang tersimpan pada sisi server.

Tahap berikutnya adalah Response Processing, yaitu proses membaca respons dalam format JSON yang dikembalikan oleh Google Gemini API. Backend mengekstraksi hasil deskripsi yang dihasilkan model kemudian menyusunnya ke dalam objek JSON agar dapat digunakan oleh Chrome Extension. Tahap terakhir menghasilkan Product Summary yang berisi ringkasan produk dan ditampilkan kepada pengguna sebagai informasi tambahan mengenai produk yang sedang dibuka.

Kode Program 4.22 Implementasi Google Gemini API

```
api_key = os.getenv("GEMINI_API_KEY", "")

ocr_prompt = (
    "Data produk dari 1688.com:\n"
    + "Harga: " + price_range + "\n"
    + "Teks OCR:\n" + ocr_formatted
)

gemini_body = _json.dumps({
    "contents":[{"parts":[{"text":ocr_prompt}]}],
    "generationConfig":{"temperature":0.7,"maxOutputTokens":4000}
}).encode()

gemini_req = urllib.request.Request(
    gemini_url,
    data=gemini_body,
    headers={"Content-Type":"application/json","x-goog-api-key":api_key}
)
with urllib.request.urlopen(gemini_req, timeout=45) as r:
    resp = _json.loads(r.read())

raw_text = resp["candidates"][0]["content"]["parts"][0]["text"].strip()

desc = {
    "nama_produk_id": raw_name,
    "fungsi": raw_text,
    "deskripsi_marketplace": raw_text
}
return jsonify({"success":True, "description":desc})
```

Kode Program 4.22 menunjukkan implementasi proses pembuatan ringkasan produk menggunakan Google Gemini API pada file app.py. Sistem terlebih dahulu membaca API Key yang disimpan pada konfigurasi backend untuk menjaga keamanan akses layanan. Selanjutnya backend mengambil data produk dari basis data, menggabungkan informasi hasil OCR, harga, dan informasi pendukung lainnya, kemudian menyusun data tersebut menjadi sebuah prompt yang berisi instruksi bagi model Gemini untuk menghasilkan ringkasan produk. Setelah prompt selesai disusun, backend mengirimkan permintaan HTTP POST ke layanan Google Gemini 2.5 Flash menggunakan format JSON. Permintaan tersebut memuat isi prompt serta konfigurasi model seperti temperature dan maxOutputTokens. Setelah respons diterima, sistem mengekstraksi hasil keluaran dari objek JSON kemudian membentuk objek Product Summary yang dikembalikan kepada Chrome Extension dalam format JSON. Pendekatan ini memungkinkan proses pembuatan ringkasan dilakukan secara otomatis tanpa memerlukan interaksi langsung pengguna dengan layanan Gemini API.



Gambar 4.22 Hasil *Product Summary* Pada Dashboard

Berdasarkan hasil implementasi, modul Google Gemini API berhasil menghasilkan ringkasan produk secara otomatis berdasarkan informasi hasil scraping dan OCR. Ringkasan yang dihasilkan meliputi judul produk, deskripsi, spesifikasi, fitur, keunggulan, dan kegunaan produk sesuai bahasa yang dipilih pengguna. Hasil tersebut ditampilkan pada antarmuka sistem sebagai informasi tambahan yang membantu pengguna memahami karakteristik produk tanpa harus membaca seluruh informasi asli pada halaman produk.

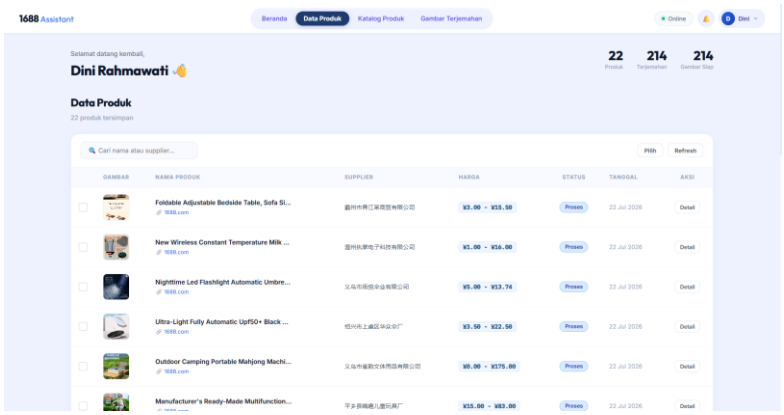
4.2.4 Implementasi Web Dashboard

Web Dashboard diimplementasikan sebagai media bagi pengguna untuk mengelola hasil scraping, hasil penerjemahan gambar, serta penyusunan katalog produk. Selain berfungsi sebagai penyimpanan data, dashboard juga menyediakan berbagai fitur yang mendukung proses penyusunan katalog produk multibahasa. Fitur-fitur tersebut dikelompokkan ke dalam empat menu utama pada *navigation bar*, yaitu Home, Data Produk, Katalog Produk, dan Gambar Terjemahan.

4.2.4.1 Menu Data Produk

Menu Data Produk digunakan untuk menampilkan seluruh data produk yang diperoleh melalui proses *scraping* pada Chrome Extension. Halaman ini berfungsi sebagai pusat pengelolaan data produk yang telah diproses oleh sistem. Informasi yang ditampilkan meliputi gambar utama produk, nama produk, status pemrosesan, tanggal pengolahan, serta informasi pendukung lainnya yang berhasil diperoleh dari platform 1688.

Selain menampilkan daftar produk, halaman ini juga menyediakan fitur pencarian untuk memudahkan pengguna menemukan produk berdasarkan nama atau kata kunci tertentu. Setiap produk juga dilengkapi dengan tombol Detail yang dapat digunakan untuk melihat informasi produk secara lebih lengkap, termasuk hasil pengolahan gambar dan informasi produk yang telah dihasilkan oleh sistem.



Gambar 4.23 Halaman Data Produk

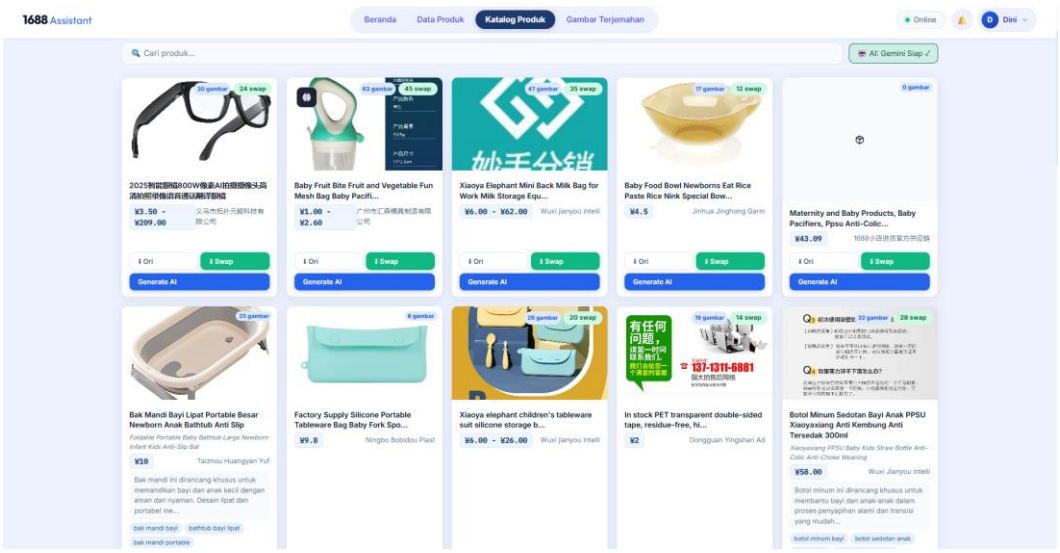
4.2.4.2 Menu Katalog Produk

Menu Katalog Produk merupakan salah satu utama pada Web Dashboard yang digunakan untuk mengelola hasil penerjemahan produk sebelum disusun menjadi katalog. Halaman ini menampilkan informasi gambar produk, hasil gambar dengan translasi, serta berbagai fasilitas pendukung penyusunan katalog.

Salah satu fitur yang tersedia pada halaman ini adalah Generate Deskripsi yang memanfaatkan Google Gemini AI. Setelah pengguna memilih produk dan menekan tombol Generate Deskripsi, sistem akan mengirimkan informasi produk yang tersimpan ke layanan Gemini AI untuk menghasilkan deskripsi produk secara otomatis. Deskripsi tersebut membantu pengguna memperoleh ringkasan informasi produk dengan cepat sehingga dapat digunakan sebagai bahan penyusunan katalog tanpa harus menulis deskripsi secara manual.

Selain itu, halaman ini juga menyediakan fitur Download Gambar Original dan Download Gambar Terjemahan sehingga pengguna dapat memperoleh gambar asli maupun gambar hasil *Image Text Swapping* secara langsung sesuai kebutuhan penyusunan katalog produk.

Tampilan halaman Katalog Produk yang memuat seluruh fitur tersebut ditunjukkan pada Gambar 4.24.

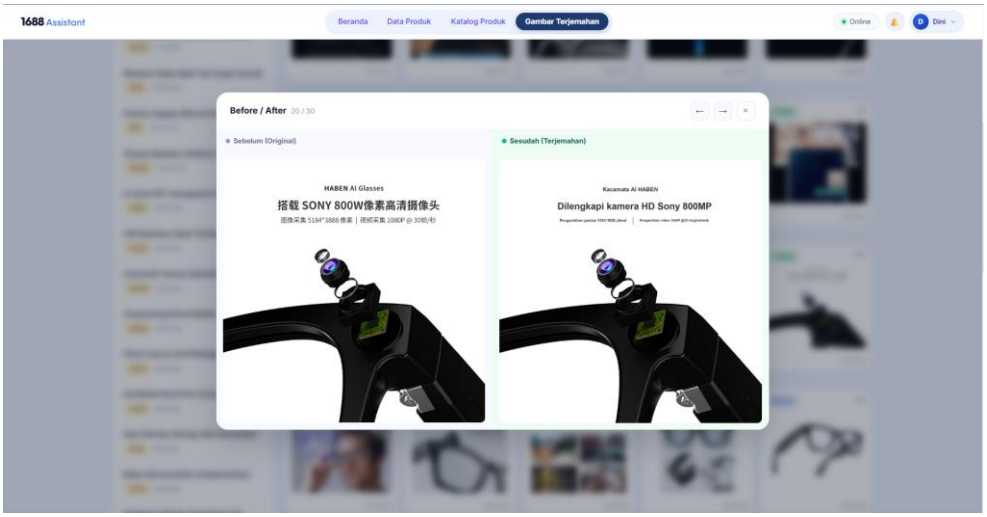


Gambar 4.24 Halaman Katalog Produk

4.2.4.3 Menu Gambar Terjemahan

Menu Gambar Terjemahan digunakan untuk menampilkan seluruh gambar yang telah berhasil diproses menggunakan modul *Image Text Swapping*. Halaman ini memudahkan pengguna untuk melihat hasil penerjemahan gambar tanpa harus membuka setiap produk secara satu per satu.

Setiap data menampilkan pasangan gambar sebelum dan sesudah proses penerjemahan sehingga pengguna dapat membandingkan hasil implementasi secara langsung. Dengan adanya halaman ini, pengguna dapat melakukan pemeriksaan visual terhadap kualitas hasil penerjemahan sekaligus memastikan bahwa tata letak dan tampilan gambar tetap terjaga setelah proses penggantian teks. Tampilan halaman Gambar Terjemahan yang menampilkan hasil sebelum dan sesudah proses penerjemahan ditunjukkan pada Gambar 4.25.



Gambar 4.25 Halaman Gambar Terjemahan

4.3 Pengujian Sistem

Pengujian sistem dilakukan untuk memastikan bahwa seluruh fungsi dan komponen yang telah diimplementasikan pada penelitian ini dapat berjalan sesuai dengan kebutuhan fungsional serta tujuan penelitian. Tahapan pengujian mengacu pada pendekatan V-Model yang telah dijelaskan pada Bab III, di mana setiap tahapan implementasi memiliki tahapan verifikasi dan validasi yang saling berpasangan. Pendekatan ini bertujuan untuk memastikan bahwa setiap modul yang dikembangkan telah memenuhi rancangan sistem sebelum diintegrasikan menjadi satu kesatuan sistem yang utuh.

Pada penelitian ini, proses pengujian dilakukan secara bertahap yang terdiri atas Unit Testing, Integration Testing, System Testing, dan User Acceptance Testing (UAT). Unit Testing digunakan untuk memverifikasi fungsi dari setiap modul sistem secara individual, meliputi modul OCR, modul translasi, modul *Image Text Swapping*, dan Firefly Algorithm. Selanjutnya, Integration Testing dilakukan untuk memastikan komunikasi dan pertukaran data antar modul dapat

berjalan dengan baik. Setelah seluruh modul berhasil diintegrasikan, System Testing dilakukan untuk mengevaluasi fungsi sistem secara keseluruhan berdasarkan kebutuhan pengguna. Tahap terakhir adalah User Acceptance Testing (UAT) yang bertujuan untuk mengetahui tingkat penerimaan pengguna terhadap sistem yang telah dikembangkan.

4.3.1 Unit Testing

Unit Testing dilakukan untuk memverifikasi bahwa setiap modul penyusun sistem telah berfungsi sesuai dengan kebutuhan fungsional sebelum dilakukan pengujian integrasi. Pengujian dilakukan secara terpisah terhadap setiap modul utama, yaitu modul DOM Scraping, OCR, Translation, Image Text Swapping, Artificial Intelligence, dan Firefly Algorithm. Setiap modul diuji menggunakan sejumlah data uji yang disesuaikan dengan karakteristik modul. Daftar lengkap data uji yang digunakan pada setiap pengujian disajikan pada Lampiran.

4.3.1.1 Pengujian DOM Scraping

Pengujian modul DOM Scraping dilakukan untuk memverifikasi kemampuan modul dalam mengekstraksi informasi produk dari halaman website 1688 berdasarkan struktur *Document Object Model* (DOM). Informasi yang diuji meliputi nama produk, harga, URL gambar produk, dan informasi penjual yang selanjutnya diteruskan ke backend untuk disimpan ke dalam basis data. Pengujian dilakukan menggunakan 10 halaman produk pada platform 1688 dengan kategori produk yang beragam. Daftar lengkap URL produk yang digunakan sebagai data uji disajikan pada Lampiran 2.

Keberhasilan pengujian diukur berdasarkan kemampuan modul dalam mengekstraksi seluruh atribut produk tanpa terjadi kehilangan data (*missing value*) maupun kesalahan pengambilan atribut (*attribute mismatch*). Tingkat keberhasilan (*Success Rate*) dihitung menggunakan Persamaan (4.1).

$$Success\ Rate\ = \frac{Jumlah\ Halaman\ Berhasil\ Diekstraksi}{Jumlah\ Seluruh\ Halaman\ Uji} \times\ 100\% \quad (4.1)$$

Tabel 4.11 Hasil Pengujian Modul DOM Scraping

Parameter	Nilai
Jumlah halaman produk diuji	10
Halaman berhasil diekstraksi	10

Halaman gagal diekstraksi	0
Success Rate	100%

Berdasarkan hasil pengujian tersebut, 10 data produk yang berhasil diekstraksi dari halaman 1688 ditampilkan pada Web Dashboard. Informasi yang berhasil diperoleh meliputi nama produk, harga produk, URL halaman, informasi penjual serta URL gambar pada Detail. Hasil tersebut menunjukkan bahwa proses ekstraksi data menggunakan Document Object Model (DOM) berhasil dilakukan dan seluruh data dapat diteruskan ke backend serta disimpan ke dalam basis data tanpa kehilangan atribut yang diperlukan. Selain mengukur tingkat keberhasilan ekstraksi halaman, dilakukan pula verifikasi terhadap setiap atribut data yang berhasil diperoleh. Hasil verifikasi atribut ditunjukkan pada Tabel 4.11.

Tabel 4.12 Verifikasi Hasil Ekstraksi Data DOM

Data yang Diekstraksi	Hasil
Nama produk	Berhasil
Harga produk	Berhasil
URL gambar	Berhasil
Informasi penjual	Berhasil

Selanjutnya dilakukan pengujian relevansi URL gambar yang berhasil diekstraksi untuk memastikan bahwa seluruh gambar yang diperoleh merupakan gambar katalog produk. Hasil pengujian relevansi gambar ditunjukkan pada Tabel 4.12.

Tabel 4.13 Hasil Pengujian Relevansi Gambar

Parameter	Nilai
Jumlah halaman produk diuji	10
Total URL gambar yang diekstraksi	302

URL gambar katalog	290
URL gambar non-katalog (gambar profil toko)	12

Berdasarkan hasil pengujian relevansi gambar pada Tabel 4.11, modul DOM Scraping berhasil mengekstraksi 302 URL gambar dari 10 halaman produk yang diuji. Dari seluruh URL gambar yang diperoleh, 290 URL merupakan gambar katalog produk, sedangkan 12 URL lainnya merupakan gambar non-katalog. Gambar non-katalog tersebut terdiri atas gambar profil toko yang terdapat pada halaman produk serta beberapa gambar lain yang tidak termasuk ke dalam katalog produk. Hal ini menunjukkan bahwa mekanisme DOM Scraping telah berhasil mengambil seluruh elemen gambar sesuai dengan *selector* yang digunakan, namun proses penyaringan (*filtering*) terhadap gambar yang bukan bagian dari katalog produk masih perlu disempurnakan agar hanya mengekstraksi gambar yang relevan.

Berdasarkan hasil pengujian, modul DOM Scraping berhasil mengekstraksi seluruh atribut utama produk yang dibutuhkan dengan tingkat keberhasilan sebesar 100%. Nama produk, harga, URL gambar, dan informasi penjual berhasil diperoleh pada seluruh halaman produk yang diuji. Meskipun demikian, hasil pengujian relevansi gambar menunjukkan bahwa masih terdapat beberapa URL gambar non-katalog yang ikut terekstraksi sehingga mekanisme *filtering* gambar masih memerlukan penyempurnaan. Temuan tersebut tidak memengaruhi proses ekstraksi atribut utama yang digunakan pada tahapan OCR, Translation, Image Text Swapping, Artificial Intelligence, dan penyimpanan ke basis data. Dengan demikian, modul DOM Scraping dinyatakan telah berfungsi sesuai dengan kebutuhan sistem dan layak digunakan pada tahap pengujian integrasi.

4.3.1.2 Pengujian Modul OCR

Pengujian modul Optical Character Recognition (OCR) dilakukan untuk memverifikasi kemampuan modul dalam mendeteksi dan mengenali teks pada gambar produk yang diperoleh dari hasil DOM Scraping. Pengujian menggunakan 10 gambar produk dari platform 1688 yang memiliki variasi ukuran gambar, jumlah teks, warna, dan kompleksitas latar belakang. Setiap gambar diproses menggunakan EasyOCR sehingga menghasilkan beberapa *text region*. Hasil pengenalan pada setiap *text region* kemudian dibandingkan dengan teks referensi (*ground truth*) untuk menghitung tingkat akurasi OCR. Daftar lengkap data uji beserta hasil pengujian disajikan pada Lampiran 3.

Tingkat akurasi OCR dihitung menggunakan Persamaan (4.2).

$$Accuracy = \frac{Jumlah\ Text\ Region\ yang\ Dikenali\ Benar}{Jumlah\ Seluruh\ text\ Region} \times 100\%$$

(4.2)

Hasil pengujian modul OCR ditunjukkan pada Tabel 4.12.

Tabel 4.14 Hasil Pengujian Modul OCR

Parameter	Nilai
Jumlah gambar uji	10
Jumlah *text region*	81
Text region dikenali dengan benar	70
Text region tidak dikenali/salah	11
Akurasi OCR	86,41%

Untuk memberikan gambaran hasil pengujian, beberapa contoh hasil ekstraksi OCR ditunjukkan pada Tabel 4.13, sedangkan hasil pengujian secara lengkap disajikan pada Lampiran 3.

Tabel 4.15 Hasil Pengujian OCR

No	*Ground Truth*	Hasil OCR	Status
1	产品包装	产品包装	Benar
2	食物调理器组7件套	食物调理器组7件套	Benar
3	哆拉哆布	#@黟布	Salah
4	— (Bukan Teks)	二	Salah
5	肉	— (tidak terdeteksi)	Salah

No	*Ground Truth*	Hasil OCR	Status
6	— (Teks Vertikal)	— (tidak terdeteksi)	Salah

Berdasarkan hasil pengujian, modul OCR memperoleh tingkat akurasi sebesar 86,41%. Hasil tersebut menunjukkan bahwa modul OCR mampu mengenali sebagian besar teks pada gambar produk dengan baik sehingga hasil ekstraksi dapat digunakan sebagai masukan bagi modul Translation dan Image Text Swapping. Kesalahan pengenalan masih ditemukan pada beberapa *text region* dengan kualitas gambar yang rendah, kontras yang kurang baik, atau karakter yang saling berhimpitan, namun secara keseluruhan tidak memengaruhi proses pemrosesan data pada sistem.

4.3.1.3 Pengujian Pengujian Modul *Translasi*

Pengujian modul Translation dilakukan untuk memverifikasi kemampuan modul dalam menerjemahkan teks hasil OCR ke bahasa target menggunakan Google Translate API. Pengujian dilakukan menggunakan hasil OCR yang diperoleh dari 10 gambar produk pada platform 1688. Setiap *text region* hasil OCR dikirimkan ke Google Translate API untuk diterjemahkan ke dalam Bahasa Indonesia. Daftar lengkap data uji beserta hasil translasi disajikan pada Lampiran 4.

Pengujian dilakukan dalam dua aspek, yaitu keberhasilan proses translasi dan akurasi hasil translasi. Keberhasilan proses translasi diukur berdasarkan kemampuan sistem dalam mengirimkan permintaan (*request*) ke Google Translate API dan menerima hasil translasi tanpa mengalami kegagalan (*request failed*). Tingkat keberhasilan proses dihitung menggunakan Persamaan (4.3).

$$Success\ Rate = \frac{Jumlah\ Request\ Berhasil}{Jumlah\ Seluruh\ Request} \times 100\% \quad (4.3)$$

Tabel 4.16 Hasil Pengujian Modul Translation

Parameter	Nilai
Jumlah request translasi	76
Request berhasil	76

Request gagal	0
Success Rate	100%
Rata-rata Response Time	0.398 detik

Selain mengukur keberhasilan proses translasi, dilakukan pula evaluasi terhadap kualitas hasil translasi dengan membandingkan makna hasil terjemahan terhadap teks sumber. Evaluasi dilakukan terhadap seluruh hasil translasi untuk memastikan bahwa makna hasil terjemahan tetap sesuai dengan konteks informasi produk. Tingkat akurasi translasi dihitung menggunakan Persamaan (4.4).

$$Akurasi\ Translasi\ = \frac{Jumlah\ Hasil\ Translasi\ Sesuai}{Jumlah\ Seluruh\ Hasil\ Translasi} \times\ 100\% \quad (4.4)$$

Tabel 4.17 Hasil Evaluasi Translasi

Parameter	Nilai
Jumlah hasil translasi dievaluasi	76
Hasil translasi sesuai	74
Hasil translasi kurang sesuai	2
Akurasi Translasi	97,36%

Berdasarkan hasil pengujian pada Tabel 4.17, seluruh 76 permintaan translasi berhasil diproses oleh Google Translate API sehingga diperoleh Success Rate sebesar 100% dengan rata-rata *response time* sebesar 0,398 detik. Hasil tersebut menunjukkan bahwa modul Translation mampu melakukan komunikasi dengan layanan penerjemahan secara konsisten tanpa mengalami kegagalan proses.

Tabel 4.18 Contoh Hasil Translasi

No	Teks OCR	Hasil Translasi	Evaluasi
1	产品包装	Kemasan produk	Sesuai

2	食物调理器组**7**件套	Set pengolah makanan 7 buah	Sesuai
3	滤出更细腻的果蔬泥	Saring puree buah dan sayuran yang lebih lembut	Sesuai
4	品名：食物调理器组	Nama Produk: Set Kondisioner Makanan	Kurang sesuai
5	只占一个碗的位置	Itu hanya menempati ruang mangkuk	Kurang sesuai

Selanjutnya, berdasarkan hasil evaluasi kualitas translasi pada Tabel 4.18, modul Translation memperoleh akurasi translasi sebesar 97,6%. Sebagian besar hasil terjemahan telah sesuai dengan makna teks sumber sehingga dapat digunakan sebagai masukan pada proses Image Text Swapping. Namun, masih terdapat beberapa hasil translasi yang kurang sesuai dalam pemilihan diksi atau konteks kalimat. Perbedaan tersebut menunjukkan bahwa kualitas hasil translasi dipengaruhi oleh kemampuan layanan penerjemahan dalam memahami konteks kalimat serta ketepatan hasil OCR pada tahap sebelumnya.

4.3.1.4 Pengujian Modul *Image Text Swapping*

Pengujian modul *Image Text Swapping* dilakukan untuk memverifikasi kemampuan modul dalam menghapus teks asli pada gambar produk menggunakan metode *image inpainting* dan menggantinya dengan teks hasil translasi. Pengujian dilakukan menggunakan 10 gambar produk yang telah melalui proses OCR dan *Translation*. Setiap gambar diproses hingga menghasilkan gambar baru dengan teks berbahasa Indonesia. Daftar lengkap data uji beserta hasil pengujian disajikan pada Lampiran 5.

Pengujian dilakukan berdasarkan empat aspek, yaitu keberhasilan penghapusan area teks asli (*image inpainting*), penempatan teks hasil translasi, kesesuaian posisi teks terhadap area semula, dan penyimpanan gambar hasil. Tingkat keberhasilan setiap aspek dihitung menggunakan Persamaan (4.5).

Tingkat keberhasilan (*Success Rate*) dihitung menggunakan Persamaan (4.5).

$$Persentase\ Keberhasilan\ Aspek = \frac{Jumlah\ Gambar\ Berhasil\ Pada\ Aspek}{Jumlah\ Seluruh\ Gambar\ Uji} \times 100\% \quad (4.5)$$

Tabel 4.19 Hasil Pengujian Modul Image Text Swapping

Aspek Pengujian	Jumlah Berhasil	Persentase
Penghapusan area teks asli (*Image Inpainting*)	8 dari 10 gambar berhasil	80%
Penempatan teks hasil translasi	9 dari 10 gambar berhasil	90%
Posisi teks sesuai area semula	10 dari 10 gambar berhasil	100%
Penyimpanan gambar hasil	10 dari 10 gambar berhasil	100%

Berdasarkan hasil pengujian pada Tabel 4.19, aspek penghapusan area teks asli (*Image Inpainting*) memperoleh tingkat keberhasilan sebesar 80%, Pada 2 gambar masih ditemukan sisa teks asli yang belum terhapus secara sempurna serta warna background yang belum sesuai.

Aspek penempatan teks hasil translasi memperoleh tingkat keberhasilan sebesar 90%. Pada 1 gambar, penempatan teks kurang optimal sehingga ukuran atau tata letak teks belum sepenuhnya menyesuaikan dengan area teks asli.

Sementara itu, aspek kesesuaian posisi teks terhadap area semula dan penyimpanan gambar hasil masing-masing memperoleh tingkat keberhasilan sebesar 100%. Hal ini menunjukkan bahwa sistem mampu mempertahankan posisi teks pada area yang sesuai dengan gambar asli serta berhasil menghasilkan dan menyimpan gambar akhir tanpa mengalami kegagalan proses.

Secara keseluruhan, hasil pengujian menunjukkan bahwa modul *Image Text Swapping* memperoleh rata-rata tingkat keberhasilan sebesar 92,5% berdasarkan empat aspek pengujian, yaitu penghapusan area teks asli (*image inpainting*), penempatan teks hasil translasi, kesesuaian posisi teks terhadap area semula, dan penyimpanan gambar hasil. Hasil tersebut menunjukkan bahwa modul telah mampu menghasilkan gambar dengan teks berbahasa Indonesia yang dapat digunakan sebagai katalog produk multibahasa. Meskipun masih ditemukan beberapa keterbatasan pada proses *image inpainting* dan penempatan teks hasil translasi, hasil akhir tetap dapat dipahami dan memenuhi kebutuhan sistem.

4.3.1.5 Pengujian Modul Firefly Algorithm

Pengujian modul Firefly Algorithm bertujuan untuk mengevaluasi kemampuan algoritma dalam mengoptimalkan konfigurasi Pipeline Scheduler

sehingga waktu pemrosesan satu halaman produk dapat diminimalkan. Pengujian dilakukan menggunakan satu halaman produk pada platform 1688 yang terdiri dari delapan gambar produk. Waktu pemrosesan dihitung sejak proses pengunduhan gambar dimulai hingga seluruh gambar selesai melalui tahapan Download Image, OCR, Translation, dan Image Text Swapping.

Pada pengujian ini, Firefly Algorithm melakukan optimasi berdasarkan fungsi *fitness* yang telah dirancang pada Bab III. Fungsi *fitness* tersebut mengevaluasi kondisi *pipeline* menggunakan empat komponen, yaitu Throughput, Parameter Match, Resource Efficiency, dan Retry Rate, sehingga konfigurasi *worker* yang dipilih merupakan konfigurasi dengan nilai *fitness* terbaik.

Pengujian dilakukan dalam dua skenario. Skenario pertama menggunakan Sequential Processing sebagai kondisi awal (*baseline*) dengan konfigurasi *worker* statis. Skenario kedua menggunakan Firefly Algorithm dengan konfigurasi awal yang sama, kemudian Firefly Algorithm menyesuaikan jumlah *worker* secara dinamis berdasarkan hasil evaluasi fungsi *fitness* terhadap kondisi *pipeline* selama proses berlangsung. Perbandingan kedua skenario digunakan untuk mengetahui pengaruh optimasi Firefly Algorithm terhadap waktu penyelesaian satu halaman produk.

4.3.1.5.1 Pengujian Sequential Processing (Baseline)

Pengujian *Sequential Processing* dilakukan sebagai kondisi awal (*baseline*) sebelum diterapkannya optimasi menggunakan Firefly Algorithm. Pada pengujian ini, jumlah *worker* pada setiap tahapan *pipeline* ditetapkan masing-masing sebanyak satu *worker* sehingga konfigurasi yang digunakan adalah (1,1,1,1).

Konfigurasi tersebut menyebabkan proses berjalan secara *sequential* karena setiap tahapan hanya memiliki kapasitas untuk menangani satu *task* pada satu waktu. Dengan kapasitas *worker* yang terbatas, gambar berikutnya tidak dapat memasuki tahapan pemrosesan sebelum gambar sebelumnya menyelesaikan seluruh rangkaian proses, yaitu *Download Image*, *OCR*, *Translation*, dan *Image Text Swapping*. Akibatnya, setiap gambar diproses secara berurutan tanpa adanya pemrosesan paralel (*concurrent processing*) antar gambar.

Pendekatan tersebut digunakan sebagai kondisi dasar untuk menggambarkan proses pengolahan gambar tanpa optimasi penjadwalan. Meskipun alur pemrosesan menjadi lebih sederhana dan mudah dikendalikan, pendekatan ini belum mampu memanfaatkan sumber daya komputasi secara optimal karena setiap tahapan harus menunggu tahapan sebelumnya selesai. Akibatnya, ketika salah satu tahapan membutuhkan waktu lebih lama, tahapan lainnya tidak dapat berjalan secara bersamaan sehingga meningkatkan waktu penyelesaian keseluruhan.

Pengujian dilakukan menggunakan satu halaman produk pada web 1688.com. Hasil rata-rata waktu pemrosesan *Sequential Processing* ditunjukkan pada Tabel 4.20.

Tabel 4.20 Rata-rata Waktu Pemrosesan Sequential Processing

Tahapan	Total (detik)	Rata-rata Waktu (detik)
Download Image	6,950	0,188
OCR	188,283	5,089
Translation	4,246	0,212
Image Text Swapping	2,657	0,133
Total Keseluruhan	202,140	5,463

Berdasarkan hasil pengujian tersebut, sistem membutuhkan waktu 202,140 detik untuk menyelesaikan pemrosesan satu halaman produk. Tahapan OCR memiliki waktu pemrosesan paling tinggi sehingga menjadi proses yang paling mendominasi waktu penyelesaian keseluruhan.

4.3.1.5.2 Pengujian *Firefly Algorithm*

Setelah diperoleh hasil pengujian menggunakan *Sequential Processing* sebagai kondisi *baseline*, selanjutnya dilakukan pengujian menggunakan *Firefly Algorithm*. Pengujian ini bertujuan untuk mengetahui kemampuan *Firefly Algorithm* dalam mengoptimalkan konfigurasi *worker* pada *Pipeline Scheduler* berdasarkan kondisi pipeline yang diamati selama proses berlangsung.

Pada pengujian ini, konfigurasi awal worker dibuat sama dengan kondisi Sequential Processing, yaitu (1,1,1,1) yang masing-masing merepresentasikan jumlah Download Worker, OCR Worker, Translation Worker, dan Image Text Swapping Worker. Penggunaan konfigurasi awal yang sama bertujuan agar proses optimasi dimulai dari kondisi yang identik dengan baseline sehingga perubahan konfigurasi yang dihasilkan sepenuhnya merupakan hasil evaluasi Firefly Algorithm.

Selama proses pemrosesan gambar berlangsung, sistem melakukan *monitoring* terhadap kondisi pipeline secara berkala melalui Performance

Monitoring. Informasi yang diamati meliputi panjang antrean (*queue*), jumlah worker aktif, throughput, serta penggunaan sumber daya. Berdasarkan informasi tersebut, Firefly Algorithm melakukan evaluasi terhadap kondisi pipeline dan menyesuaikan konfigurasi worker apabila terdeteksi adanya bottleneck pada salah satu tahapan proses.

Hasil perubahan konfigurasi worker yang dilakukan Firefly Algorithm selama proses optimasi ditunjukkan pada Tabel 4.21.

Tabel 4.21 Hasil Optimasi Firefly Algorithm

Tahap Optimasi	Kondisi yang Terdeteksi	Konfigurasi Sebelum	Konfigurasi Sesudah	Penjelasan
Kondisi awal	Inisialisasi Pipeline	(1,1,1,1)	(1,1,1,1)	Seluruh worker menggunakan satu proses sebagai kondisi awal pengujian.
Trigger 1	Bottleneck pada Download Queue (8 antrean)	(1,1,1,1)	(5,1,1,1)	Firefly meningkatkan Download Worker untuk mempercepat proses pengambilan gambar.
Trigger 2	OCR mulai menjadi bottleneck	(5,1,1,1)	(5,2,1,1)	OCR Worker ditambah karena antrean OCR mulai meningkat.
Trigger 3	Antrean OCR tetap tinggi, Translation mulai menunggu	(5,2,1,1)	(2,2,2,1)	Download dikurangi, Translation ditambah agar pipeline lebih seimbang.
Trigger 4	Download tidak lagi menjadi bottleneck	(2,2,2,1)	(1,2,2,1)	Download Worker dikurangi sehingga sumber daya dialihkan ke proses berikutnya.
Trigger 7	Translation relatif ringan	(1,2,2,1)	(1,2,1,1)	Translation Worker dikurangi karena tidak lagi menjadi bottleneck.

Tahap Optimasi	Kondisi yang Terdeteksi	Konfigurasi Sebelum	Konfigurasi Sesudah	Penjelasan
Konfigurasi akhir	Pipeline selesai	(1,2,1,1)	(1,2,1,1)	Konfigurasi akhir yang dipilih Firefly hingga seluruh gambar selesai diproses.

Berdasarkan Tabel 4.19 dapat diketahui bahwa Firefly Algorithm memulai proses optimasi dari konfigurasi awal (1,1,1,1) yang sama dengan konfigurasi pada Sequential Processing. Pada tahap awal pemrosesan, seluruh gambar masih berada pada antrean Download sehingga Performance Monitoring mendeteksi bottleneck pada tahap tersebut. Berdasarkan hasil evaluasi Firefly Algorithm, jumlah Download Worker kemudian ditingkatkan dari 1 menjadi 5 agar proses pengunduhan dapat dilakukan secara paralel dan antrean Download dapat dikurangi.

Setelah proses pengunduhan menjadi lebih cepat, bottleneck berpindah ke tahap OCR karena jumlah gambar yang masuk ke proses OCR meningkat lebih cepat dibandingkan kemampuan OCR dalam memproses gambar. Berdasarkan kondisi tersebut Firefly kembali mengevaluasi konfigurasi worker hingga diperoleh konfigurasi akhir (1,2,1,1). Hasil ini menunjukkan bahwa Firefly mampu menyesuaikan jumlah worker secara dinamis sesuai perubahan beban kerja pada setiap tahapan pipeline.

Setelah proses optimasi selesai, diperoleh hasil pengujian sebagaimana ditunjukkan pada Tabel 4.22.

Tabel 4.22 Hasil Pengujian Firefly Algorithm

Parameter	**Nilai**
Konfigurasi Awal	(1,1,1,1)
Konfigurasi Akhir	(1,2,1,1)
Jumlah Trigger Firefly	11
Total Waktu 1 Halaman Produk	20,63 detik

Berdasarkan Tabel 4.22, seluruh gambar berhasil diproses dengan konfigurasi akhir (1,2,1,1). Firefly Algorithm melakukan sebelas kali proses evaluasi hingga diperoleh konfigurasi yang mampu menyelesaikan pemrosesan satu halaman produk dalam waktu 20,63 detik.

4.3.1.5.3 Perbandingan Sequential Processing dan Firefly Algorithm

Setelah dilakukan pengujian menggunakan Sequential Processing dan Firefly Algorithm, selanjutnya dilakukan perbandingan terhadap waktu pemrosesan satu halaman produk untuk mengetahui efektivitas optimasi yang diberikan oleh Firefly Algorithm. Tingkat pengurangan waktu pemrosesan dihitung menggunakan Persamaan (4.7).

$$Pengurangan\ Waktu\ (\%)\ = \frac{T_{Sequential} - T_{Firefly}}{T_{Sequential}} \times 100\% \quad (4.6)$$

dengan:

$T_{Sequential}$ = total waktu pemrosesan menggunakan Sequential Processing.

$T_{Firefly}$ = total waktu pemrosesan menggunakan Firefly Algorithm.

Hasil perbandingan kedua metode ditunjukkan pada Tabel 4.21.

Tabel 4.23 Perbandingan Sequential Processing dan Firefly Algorithm

Metode	Total Waktu 1 Halaman Produk (detik)
Sequential Processing	202,140
Firefly Algorithm	20,63

Berdasarkan hasil pengujian, diperoleh:

$$Pengurangan\ Waktu\ (\%)\ = \frac{202{,}140 - 20{,}63}{202{,}140} \times 100\%$$

$$= \frac{181{,}510}{202{,}140} \times 100\%$$

$$= 89{,}75\%$$

Berdasarkan hasil perhitungan menggunakan Persamaan (4.6), penerapan Firefly Algorithm mampu mengurangi waktu pemrosesan sebesar 89,79% dibandingkan dengan Sequential Processing. Penurunan waktu tersebut menunjukkan bahwa mekanisme penyesuaian jumlah worker secara dinamis mampu mengurangi bottleneck pada setiap tahapan pipeline sehingga proses pemrosesan menjadi lebih efisien. Sebaliknya, pada Sequential Processing konfigurasi worker bersifat tetap sehingga sistem tidak dapat menyesuaikan diri terhadap perubahan beban kerja yang terjadi selama proses berlangsung. Oleh karena itu, Firefly Algorithm terbukti lebih efektif dalam meningkatkan efisiensi Pipeline Scheduler pada sistem yang dikembangkan.

4.3.2 Integration Testing

Integration Testing dilakukan untuk memastikan bahwa setiap modul yang telah lolos Unit Testing dapat saling berinteraksi dan bertukar data dengan benar setelah diintegrasikan menjadi satu kesatuan sistem. Pengujian dilakukan pada alur proses pengolahan informasi produk mulai dari proses pengambilan data hingga penyajian hasil pada Web Dashboard. Fokus pengujian berada pada kesesuaian aliran data (*data flow*) dan komunikasi antar modul.

Hasil pengujian integrasi ditunjukkan pada Tabel 4.24.

Tabel 4.24 Hasil Integration Testing

No	Modul yang Diintegrasikan	Skenario Pengujian	Hasil yang Diharapkan	Hasil
1	DOM Scraping → OCR	Melakukan scraping produk yang memiliki gambar berisi teks Mandarin	Gambar hasil scraping berhasil diteruskan ke modul OCR	Valid
2	OCR → Google Translate	Hasil OCR diterjemahkan	Teks hasil OCR berhasil diteruskan ke Google Translate API	Valid
3	Google Translate → *Image Text Swapping*	Hasil terjemahan digunakan untuk mengganti teks pada gambar	Gambar hasil terjemahan berhasil dihasilkan	Valid

No	Modul yang Diintegrasikan	Skenario Pengujian	Hasil yang Diharapkan	Hasil
4	Image Text Swapping → Gemini	Informasi produk diproses setelah translasi selesai	Gemini berhasil menghasilkan nama produk, deskripsi, dan kata kunci	Valid
5	Pipeline Scheduler → Firefly Algorithm	Pipeline dijalankan pada beberapa produk	Firefly berhasil mengoptimalkan konfigurasi scheduler	Valid
6	Seluruh Pipeline → Database	Seluruh proses pengolahan selesai	Data produk dan gambar berhasil disimpan ke basis data	Valid
7	Database → Web Dashboard	Dashboard dibuka	Informasi produk berhasil ditampilkan pada Web Dashboard	Valid

Berdasarkan hasil Integration Testing, seluruh modul berhasil saling berinteraksi sesuai dengan alur proses yang telah dirancang. Data dapat mengalir dari proses *DOM scraping*, OCR, penerjemahan, *image text swapping*, optimasi *Pipeline Scheduler* menggunakan Firefly Algorithm, hingga penyimpanan ke basis data dan penyajian melalui Web Dashboard tanpa ditemukan kesalahan integrasi.

4.3.3 System Testing

System Testing dilakukan untuk memverifikasi bahwa sistem yang telah diimplementasikan mampu memenuhi seluruh kebutuhan sistem yang telah dirumuskan pada tahap analisis kebutuhan. Pengujian dilakukan terhadap sistem secara menyeluruh sebagai satu kesatuan setelah seluruh modul berhasil diintegrasikan. Pada penelitian ini, *System Testing* mencakup pengujian fungsional menggunakan metode *Black Box Testing* serta pengujian non-fungsional untuk memastikan sistem memenuhi karakteristik operasional yang telah ditetapkan.

Pengujian fungsional dilakukan untuk memastikan bahwa setiap fungsi sistem telah berjalan sesuai dengan kebutuhan fungsional yang telah dirumuskan pada Bab III. Hasil pengujian fungsional ditunjukkan pada Tabel 4.25.

Tabel 4.25 Hasil Functional Testing

Kode	Fitur yang Diuji	Skenario Pengujian	Hasil yang Diharapkan	Hasil
F-01	Pengambilan gambar produk	Pengguna melakukan import data produk dari halaman 1688	Sistem berhasil mengambil gambar produk secara otomatis	Valid
F-02	DOM Scraping	Pengguna melakukan proses scraping halaman produk	Informasi produk berhasil diperoleh dari DOM halaman	Valid
F-03	OCR	Pengguna menjalankan fitur Translate Image	Sistem berhasil mendeteksi teks Mandarin pada gambar	Valid
F-04	Google Translate	Proses translasi dijalankan	Teks hasil OCR berhasil diterjemahkan ke Bahasa Indonesia	Valid
F-05	Image Text Swapping	Proses translasi selesai	Sistem berhasil menampilkan gambar dengan teks hasil terjemahan	Valid
F-06	Gemini AI	Pengguna memilih Generate Summary	Sistem menghasilkan nama produk, deskripsi, dan kata kunci	Valid
F-07	Firefly Algorithm	Pipeline dijalankan	Sistem berhasil menjalankan optimasi konfigurasi Pipeline Scheduler	Valid
F-08	Penyimpanan Data	Proses pengolahan selesai	Data berhasil disimpan ke basis data	Valid
F-09	Web Dashboard	Pengguna membuka dashboard	Informasi produk berhasil ditampilkan	Valid
F-10	Pencarian & Filter	Pengguna mencari produk	Data berhasil ditampilkan sesuai kata kunci atau filter	Valid
F-11	Download Gambar	Pengguna memilih Download	Gambar hasil terjemahan berhasil diunduh	Valid

Berdasarkan hasil pengujian fungsional, seluruh fungsi utama sistem telah berjalan sesuai dengan kebutuhan yang telah ditentukan. Sistem mampu melakukan proses *scraping*, OCR, penerjemahan, *image text swapping*, optimasi *Pipeline Scheduler* menggunakan Firefly Algorithm, penyimpanan data, serta penyajian informasi melalui *Web Dashboard.*

Selanjutnya, pengujian non-fungsional dilakukan untuk memastikan bahwa sistem memenuhi kebutuhan non-fungsional yang berkaitan dengan kompatibilitas, kemudahan penggunaan, integritas data, konektivitas layanan eksternal, dan kebutuhan operasional sistem. Hasil pengujian non-fungsional ditunjukkan pada Tabel 4.24.

Tabel 4.26 Hasil Non-Functional Testing

Kode	Aspek	Skenario Pengujian	Hasil yang Diharapkan	Hasil
NF-01	Compatibility	Sistem dijalankan menggunakan Google Chrome	Seluruh fitur dapat berjalan dengan baik pada Google Chrome	Valid
NF-02	Usability	Pengguna mencoba seluruh fitur sistem	Antarmuka mudah dipahami dan digunakan	Valid
NF-03	Data Integrity	Pengguna melakukan scraping dan penyimpanan data	Seluruh data tersimpan secara konsisten tanpa kehilangan informasi	Valid
NF-04	API Integration	Sistem mengakses Google Translate API dan Gemini API	Sistem berhasil memperoleh respons dari kedua layanan	Valid
NF-05	Network	Sistem dijalankan dengan koneksi internet aktif	Seluruh proses scraping, translasi, dan pengolahan data berjalan dengan baik	Valid

Berdasarkan hasil pengujian non-fungsional, sistem telah memenuhi karakteristik operasional yang telah ditetapkan. Sistem dapat berjalan dengan baik pada browser Google Chrome, mampu menjaga konsistensi data selama proses pengolahan berlangsung, berhasil terintegrasi dengan Google Translate API dan Gemini API, serta dapat menjalankan seluruh proses pengolahan informasi produk dengan baik selama koneksi internet tersedia.

4.3.4 *User Acceptance Testing (UAT)*

User Acceptance Testing (UAT) dilakukan untuk mengevaluasi tingkat penerimaan pengguna terhadap sistem yang telah dikembangkan berdasarkan

kebutuhan proses bisnis yang sebenarnya. Berbeda dengan pengujian fungsional yang berfokus pada kinerja sistem, UAT bertujuan untuk mengetahui apakah sistem mampu membantu pengguna dalam melakukan pengambilan data produk, penerjemahan gambar, serta penyusunan katalog produk multibahasa.

Pada penelitian ini, responden dipilih menggunakan teknik purposive sampling, yaitu teknik pemilihan responden berdasarkan kesesuaian peran dan pengalaman terhadap sistem yang dikembangkan. Pengujian melibatkan empat responden yang terdiri atas satu *Owner* bisnis impor, satu Staf Purchasing, satu Admin *Marketplace*, dan satu Pengguna Umum (*General User*). Tiga responden pertama dipilih karena mewakili pengguna utama (*key users*) yang secara langsung terlibat dalam proses pencarian produk, pengelolaan informasi produk, serta penyusunan katalog dari platform 1688. Sementara itu, responden Pengguna Umum dilibatkan untuk mengevaluasi kemudahan penggunaan (*usability*) sistem dari sudut pandang pengguna yang tidak memiliki pengalaman maupun latar belakang khusus pada proses bisnis impor.

Setelah mencoba seluruh fitur sistem, setiap responden diminta mengisi kuesioner menggunakan skala Likert lima tingkat, yaitu Sangat Tidak Setuju (1), Tidak Setuju (2), Netral (3), Setuju (4), dan Sangat Setuju (5). Hasil penilaian kemudian dihitung untuk mengetahui tingkat penerimaan pengguna terhadap sistem berdasarkan pengalaman penggunaan secara langsung.

Tabel 4.27 Karakteristik Responden

No	Responden	Peran	Tanggung Jawab
1	Responden 1	Owner Bisnis Impor	Mengelola proses bisnis impor hingga distribusi barang.
2	Responden 2	Staf Purchasing	Melakukan pencarian dan pengadaan produk dari platform 1688.
3	Responden 3	Admin Marketplace	Mengelola katalog produk dan informasi produk pada marketplace.
4	Responden 4	Pengguna Umum (*General User*)	Mengevaluasi kemudahan penggunaan sistem dari sudut pandang pengguna non-spesialis.

Pengujian User Acceptance Testing (UAT) dilakukan menggunakan delapan pernyataan yang disusun untuk mengevaluasi kemudahan penggunaan, fungsionalitas, dan manfaat sistem terhadap kebutuhan pengguna.

Tabel 4.28 Pernyataan User Acceptance Testing

No	Pernyataan
1	Sistem mempermudah proses pengambilan data produk dari platform 1688.
2	Fitur penerjemahan gambar membantu memahami informasi produk yang berbahasa Mandarin.
3	Hasil *Image Text Swapping* membuat informasi pada gambar lebih mudah dipahami.
4	Fitur *Generate* Deskripsi membantu penyusunan informasi produk untuk katalog.
5	Web *Dashboard* memudahkan pengelolaan data produk dan hasil pengolahan gambar.
6	Sistem membantu mengurangi pekerjaan manual dalam proses penyusunan katalog produk.
7	Sistem sesuai dengan kebutuhan proses bisnis impor yang dijalankan.
8	Secara keseluruhan sistem mudah digunakan dan layak diterapkan.

Persentase tingkat penerimaan pengguna dihitung menggunakan rumus berikut:

$$Persentase\ UAT\ = \frac{Total\ Skor\ yang\ Diperoleh}{Skor\ Maksimum} \times 100\% \quad (4.7)$$

Keterangan:

$Total\ Skor\ yang\ Diperoleh$ = jumlah seluruh skor jawaban responden.

$Skor\ Maksimum$ = jumlah responden × jumlah pertanyaan × skor tertinggi (5).

Hasil perhitungan persentase *User Acceptance Testing* (UAT) kemudian diinterpretasikan berdasarkan kategori tingkat penerimaan pengguna. Interpretasi persentase UAT yang digunakan dalam penelitian ini disajikan pada Tabel 4.27.

Tabel 4.29 Interpretasi Persentase UAT

Persentase	Kategori
81% – 100%	Sangat Baik
61% – 80%	Baik
41% – 60%	Cukup
21% – 40%	Kurang
0% – 20%	Sangat Kurang

Berdasarkan instrumen dan metode perhitungan yang telah ditetapkan, hasil pengujian User Acceptance Testing (UAT) dari responden disajikan pada Tabel 4.28.

Tabel 4.30 Hasil User Acceptance Testing

No	Pernyataan	Owner	Purchasing	Admin	General User	Total	Persentase	Kategori
1	Sistem mempermudah proses pengambilan data produk dari platform 1688.	4	5	5	5	19	95,00%	Sangat Baik
2	Fitur penerjemahan gambar membantu memahami informasi produk yang berbahasa Mandarin.	3	4	3	3	13	65,00%	Baik

No	Pernyataan	Owner	Purchasing	Admin	General User	Total	Persentase	Kategori
3	Hasil *Image Text Swapping* membuat informasi pada gambar lebih mudah dipahami.	4	5	5	5	19	95,00%	Sangat Baik
4	Fitur Generate Deskripsi membantu penyusunan informasi produk untuk katalog.	5	3	5	5	18	90,00%	Sangat Baik
5	Web Dashboard memudahkan pengelolaan data produk dan hasil pengolahan gambar.	3	3	5	4	15	75,00%	Baik
6	Sistem membantu mengurangi pekerjaan manual dalam proses penyusunan katalog produk.	4	5	5	5	19	95,00%	Sangat Baik
7	Sistem sesuai dengan kebutuhan proses bisnis impor yang dijalankan.	3	4	4	3	14	70,00%	Baik
8	Secara keseluruhan sistem mudah digunakan dan	4	5	5	4	18	90,00%	Sangat Baik

No	Pernyataan	Owner	Purchasing	Admin	General User	Total	Persentase	Kategori
	layak diterapkan.							
	Total	30	34	37	34	135	84,37%	Sangat Baik

Berdasarkan hasil User Acceptance Testing, sistem memperoleh tingkat penerimaan pengguna sebesar 84.37% dengan kategori Sangat Baik. Hasil tersebut menunjukkan bahwa sistem mampu membantu proses pengambilan data produk, penerjemahan gambar, penyusunan katalog, serta pengelolaan informasi produk melalui Web Dashboard. Selain memperoleh penilaian kuantitatif, responden juga memberikan masukan terhadap sistem yang dikembangkan sebagai bahan evaluasi untuk pengembangan pada penelitian selanjutnya.