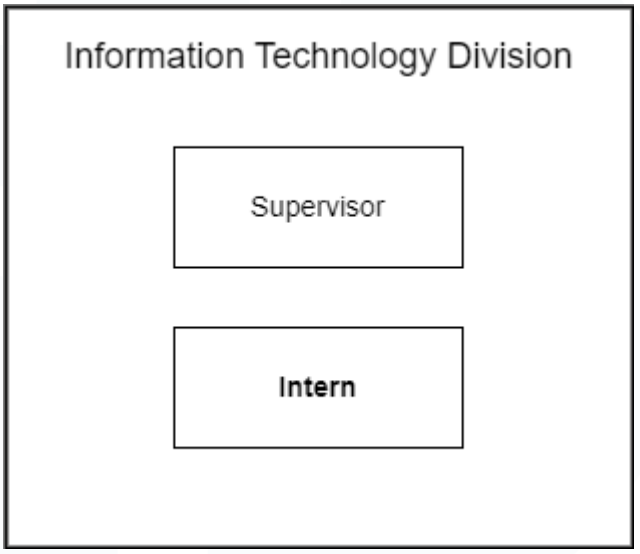
BAB III

PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

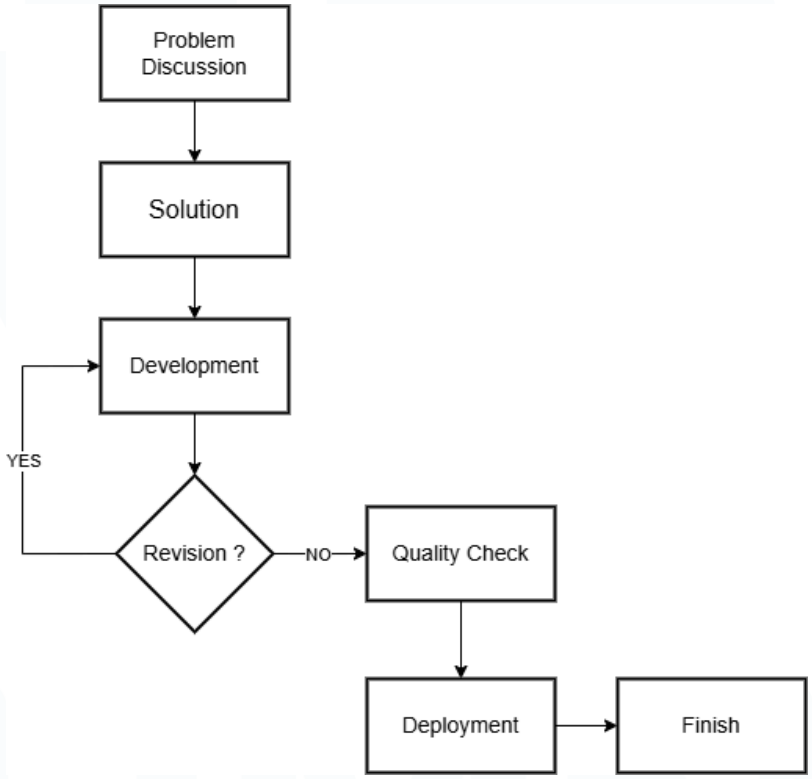


Gambar 3.1 Struktur bagian *information technology*

Pada program magang di PT. Mitra Makmur Multindo, penulis ditempatkan pada bagian IT sebagai *full-stack developer intern*. Bagian IT memiliki tanggung jawab utama dalam pemeliharaan serta pengembangan sistem dan aplikasi yang digunakan dalam operasional perusahaan. Selama pelaksanaan magang, penulis berada di bawah koordinasi langsung dari seorang supervisor dan juga mentor bagian IT Hansen Wangsawardhana, seperti yang terlihat pada gambar 3.1. Dalam struktur organisasi, bagian IT memiliki salah satu fungsi sebagai unit pendukung lintas departemen, di mana fokus utamanya adalah menyusun solusi teknologi yang sesuai dengan kebutuhan *internal*, baik dalam bentuk aplikasi berbasis web maupun perangkat lunak desktop. Seluruh kegiatan kerja dilakukan secara terarah dengan mengacu pada prioritas proyek yang diberikan oleh pihak perusahaan, dan dipantau secara berkala.

Alur kerja bagian IT bersifat kolaboratif dan responsif terhadap kebutuhan tiap bagian lain, seperti bagian *accounting*, dan *purchasing*. Setiap permintaan atau permasalahan yang memerlukan solusi digital akan dikomunikasikan melalui

koordinasi antar bagian, yang kemudian dikaji oleh tim IT untuk dikembangkan menjadi sebuah sistem atau aplikasi yang sesuai. Dalam konteks ini, penulis tidak hanya menjalankan peran teknis dalam membangun sistem, tetapi juga berpartisipasi dalam diskusi kebutuhan pengguna (*user requirement*), pengujian *internal*, serta penyempurnaan antarmuka pengguna (UI). Alur kerja dapat dilihat pada gambar 3.2.



Gambar 3.2 Alur kerja bagian *information technology*

Seluruh proyek yang dikerjakan selama masa magang disampaikan langsung kepada *intern* melalui koordinasi *internal* bagian IT. Alur kerja dimulai dari pembahasan permasalahan yang dihadapi oleh bagian pengguna, dilanjutkan dengan perumusan solusi teknis yang sesuai untuk menjawab kebutuhan tersebut. Setelah solusi disepakati, proyek masuk ke tahap pengembangan sistem. Pada fase ini, *intern* diberi tanggung jawab untuk mengembangkan aplikasi berdasarkan spesifikasi yang telah ditentukan, sambil tetap berkoordinasi secara aktif, terutama apabila terdapat informasi yang belum jelas atau terjadi perubahan kebutuhan dari pihak pengguna.

Selama proses pengembangan berlangsung, supervisi melakukan monitoring secara berkala untuk meninjau perkembangan proyek dan memberikan *feedback* terhadap hasil sementara. Revisi dapat terjadi di berbagai aspek, seperti fungsionalitas, alur logika, maupun tampilan antarmuka pengguna (UI). Proses pengembangan ini bersifat iteratif, artinya dilakukan secara bertahap dan memungkinkan adanya penyesuaian ulang hingga sistem mencapai versi akhir yang stabil dan sesuai dengan kebutuhan pengguna.

Setelah sistem dianggap siap secara fungsional dan visual, *intern* bersama supervisi akan melakukan tahap *Quality Check* (QC) untuk menguji keseluruhan alur kerja, keakuratan pengolahan data, serta konsistensi tampilan dan interaksi sistem. Sebelum masuk ke tahap *deployment*, biasanya dilakukan *meeting final* bersama *user* untuk melakukan demo aplikasi secara langsung. Pertemuan ini menjadi tahap validasi akhir yang penting untuk memastikan sistem telah sesuai ekspektasi *user*, sekaligus memberikan ruang untuk menyampaikan masukan terakhir jika masih diperlukan. Apabila sistem telah disetujui, proyek dilanjutkan ke tahap *deployment*, yaitu proses penyebaran aplikasi ke lingkungan operasional agar dapat digunakan secara aktif oleh *user* sesuai tujuan awal pengembangannya.

3.2 Tugas dan Uraian Kerja Magang

Sebagai *full-stack developer intern*, penulis memiliki tanggung jawab utama untuk mengembangkan aplikasi berdasarkan kebutuhan *internal* perusahaan yang telah ditentukan sebelumnya melalui tahap perumusan solusi (*solution*). Pada tahap tersebut, setiap aplikasi telah ditetapkan fungsi dan spesifikasinya secara garis besar, sehingga proses implementasi dapat langsung diarahkan pada pemenuhan kebutuhan tersebut secara teknis.

Selama masa magang, penulis diberikan tiga proyek utama yang dikerjakan secara *paralel*, masing-masing berasal dari kebutuhan bagian perusahaan yang berbeda. Proyek pertama adalah pengembangan aplikasi e-Catalogue W3M, yang ditujukan untuk mendukung kegiatan tim *sales* dalam mengakses informasi produk secara daring. Proyek kedua adalah Tax Namer aplikasi pengubah format penamaan *file invoice* otomatis, yang dibuat untuk mendukung efisiensi kerja di

bagian *accounting* khususnya perpajakan. Sedangkan proyek ketiga adalah pengembangan aplikasi Order-helper, yaitu sistem berbasis desktop yang digunakan oleh bagian *purchasing* untuk menyusun *file order* bulanan secara lebih cepat dan efisien. Rincian pekerjaan yang dilakukan pada masing-masing proyek dapat dilihat pada tabel 3.1.

Tabel 3.1 *Timeline* Kerja Magang

No.	Pekerjaan	Bulan	Minggu ke-
1.	*Onboarding*	Maret	1
2.	Persiapan Teknis	Maret	1
Project 1: e-Catalogue W3M			
4.	Membuat Schema Models dan Backend	Maret-April	3-2
5.	Membuat Desain Tampilan	April	3
6.	Membuat Tampilan (UI)	April-Mei	3-1
7.	Penambahan Fitur “Spare parts Relations” dan Revisi Tampilan (UI)	Juni	1-2
8.	Demo dan Deployment	Juni	2
Project 2: Tax Namer			
9.	Membuat Route Backend	Maret	1-2
10.	Membuat Tampilan (UI)	Maret	2-3
Project 3: Order-helper			
11.	Membuat Backend dan Inter-Process Communication	Mei	1-2
12.	Membuat Tampilan (UI)	Mei	3-4
13.	Demo	Mei	4
14.	Revisi Output File	Juni	1

3.2.1 Onboarding

Kegiatan magang dimulai dengan sesi *onboarding* yang dilaksanakan pada tanggal 3 Maret 2025. *Onboarding* diawali pada pukul 08.00 WIB, dan dihitung sebagai hari pertama menjalani tugas sebagai *intern* di PT. Mitra Makmur Multindo. Dalam sesi ini, penulis mendapatkan penjelasan mengenai profil umum perusahaan, ruang lingkup usaha, serta struktur organisasi yang mencakup sektor *office* dan sektor produksi. Disampaikan pula deskripsi tugas dari masing-masing bagian seperti IT, *purchasing*, dan *accounting*, termasuk peraturan kerja serta ketentuan jam kerja yang berlaku bagi seluruh karyawan. Pada kesempatan ini juga *intern* dibuatkan presensi yang menggunakan alat absensi berbasis sidik jari (*fingerprint scanner*) untuk mencatat jam masuk dan pulang secara otomatis.

Setelah sesi pemaparan materi selesai, kegiatan *onboarding* dilanjutkan dengan tur keliling area kerja, mencakup kunjungan ke sektor *office* dan sektor produksi. *Intern* diajak untuk melihat langsung kondisi lingkungan kerja serta alur aktivitas di masing-masing bagian, sehingga mendapatkan gambaran menyeluruh terkait proses operasional yang berlangsung di perusahaan.

3.2.2 Persiapan Teknis

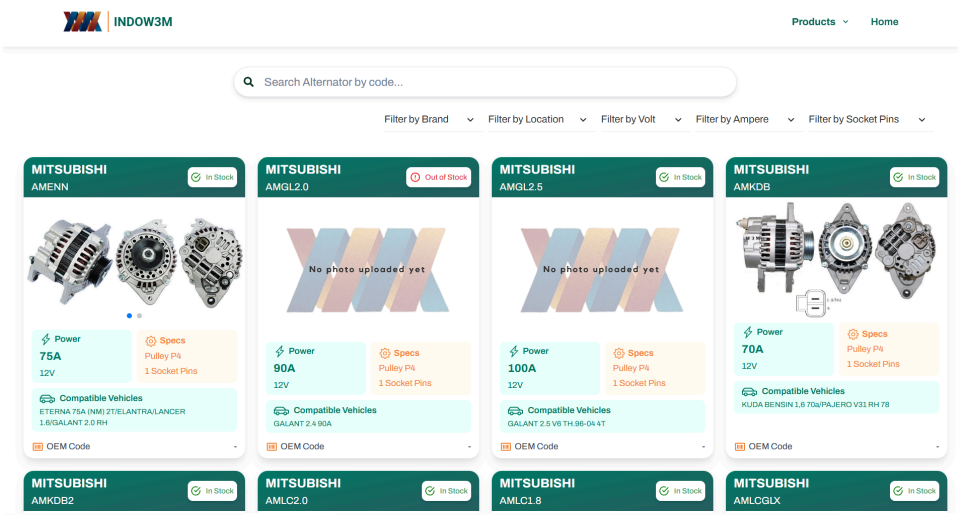
Setelah proses onboarding selesai dilaksanakan, penulis mengikuti rapat awal bersama supervisi dari bagian IT untuk membahas secara umum proyek-proyek yang akan dikerjakan selama masa magang. Pada pertemuan tersebut, penulis diberikan gambaran mengenai teknologi dan *framework* yang digunakan oleh perusahaan. Sebagai *intern full-stack developer*, penulis diminta untuk mempersiapkan diri dan menguasai penggunaan *framework* React, yang merupakan standar utama dalam pengembangan antarmuka pengguna (UI) di lingkungan internal perusahaan.

Dalam proses adaptasi awal ini, penulis diarahkan untuk memahami struktur proyek React yang digunakan di perusahaan, termasuk penerapan TypeScript untuk memastikan *type safety* dan keterbacaan kode yang lebih baik. Kegiatan pembelajaran dilakukan secara mandiri selama dua hari.

3.2.3 Project 1: e-Catalogue W3M

Pada proyek pertama ini, penulis ditugaskan untuk mengembangkan sebuah aplikasi *e-Catalogue* berbasis web yang dirancang khusus untuk produk-produk W3M, yaitu lini produk suku cadang yang dipasarkan oleh PT. Mitra Makmur Multindo. Tujuan utama dari pengembangan aplikasi ini adalah untuk meningkatkan efisiensi kerja tim *sales* lapangan, yang selama ini harus membawa katalog fisik yang tebal untuk mencari informasi spesifikasi produk, serta harus menghubungi staf kantor secara langsung hanya untuk mengetahui ketersediaan stok.

Aplikasi *e-Catalogue* W3M terdiri dari dua sisi utama, yaitu sisi *client* dan sisi *admin*. Pada sisi *client*, pengguna dapat mengakses katalog produk secara terbuka, melihat gambar, detail spesifikasi, serta informasi umum terkait ketersediaan barang. Sementara itu, sisi *admin* merupakan *panel* CMS (*Content Management System*) yang memungkinkan administrator atau staf *internal* untuk melakukan CRUD (*Create, Read, Update, Delete*) terhadap data produk. *Panel* ini memungkinkan perusahaan untuk memperbarui katalog sesuai dengan perubahan produk, baik dari segi spesifikasi, gambar, harga, maupun status ketersediaan secara berkala. Tampilan halaman e-Catalogue W3M dapat dilihat pada gambar 3.3.



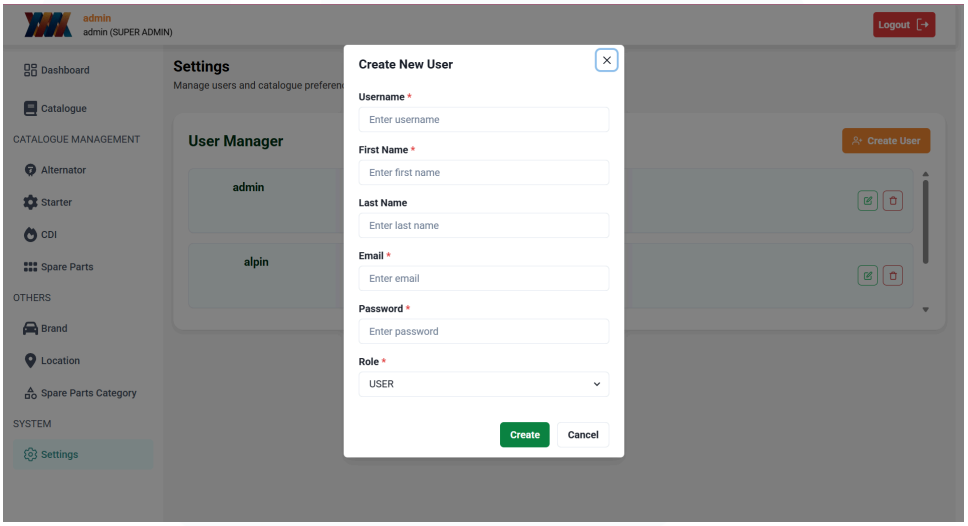
Gambar 3.3 Tampilan halaman *e-Catalogue* W3M

Aplikasi ini dibangun menggunakan teknologi *modern* yang berfokus pada efisiensi dan keamanan sistem. Pada sisi *backend*, sistem dikembangkan dengan Express.js yang dijalankan melalui *runtime* Bun, yang memberikan performa *server* yang cepat dan efisien. Untuk pengelolaan *database*, digunakan Prisma ORM sebagai alat utama dalam menangani operasi CRUD. Setiap data yang masuk ke dalam sistem divalidasi menggunakan Zod.

Sementara itu, pada sisi frontend, aplikasi dibangun dengan *framework* React, dipadukan dengan Chakra UI untuk pembuatan antarmuka pengguna (UI) yang responsif, dan konsisten di berbagai perangkat. Dari sisi keamanan, sistem menerapkan autentikasi berbasis JSON Web Token (JWT) yang dikombinasikan dengan cookies untuk menjaga sesi pengguna selama penggunaan aplikasi.

Untuk memastikan setiap pengguna hanya dapat mengakses fitur yang sesuai dengan tanggung jawabnya, diterapkan pula sistem *Role-Based Access Control* (RBAC). Melalui mekanisme ini, setiap *role* seperti *Super Admin*, *Admin*, *Sales*, dan *Guest* memiliki tingkat akses yang berbeda. *Super Admin* memiliki kendali penuh terhadap data dan pengaturan sistem, *Admin* memiliki akses terbatas hanya pada pengelolaan

produk melalui CMS, sementara *Sales* atau pengguna publik hanya dapat mengakses informasi katalog tanpa kemampuan untuk melakukan perubahan. RBAC diterapkan untuk menjaga integritas data dan mencegah akses yang tidak sah terhadap fitur-fitur sensitif dalam aplikasi, tampilan *feature* RBAC dapat dilihat pada gambar 3.4.



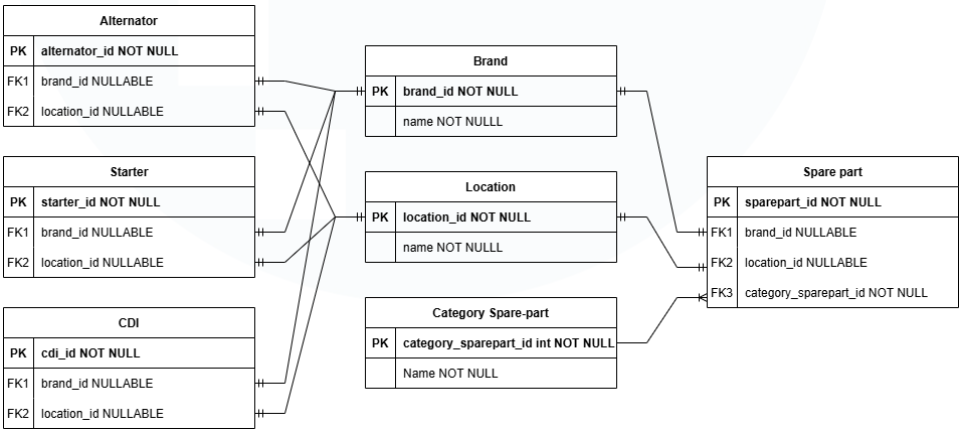
Gambar 3.4 Tampilan *feature* RBAC

Berbeda dengan pendekatan umum yang menggunakan *token* berisi informasi *role*, sistem ini tidak menggunakan *token* secara eksplisit untuk mengatur hak akses. Sebagai gantinya, *role* pengguna seperti *Super Admin*, *Admin*, *Sales*, dan *Guest* disimpan dalam bentuk enum pada basis data, dan secara otomatis di-assign ke setiap entitas pengguna. Penerapan sistem *Role-Based Access Control* (RBAC) menggunakan *middleware* khusus bernama verifyRole. *Middleware* ini bertugas memverifikasi apakah *role* pengguna termasuk dalam daftar yang diperbolehkan untuk mengakses *route* tertentu. Jika tidak sesuai, permintaan akan langsung ditolak dengan respons *forbidden*.

Dalam pembuatan skema model pada *database*, struktur *database* dirancang dengan memisahkan produk ke dalam empat tipe utama, yaitu Alternator, Starter, CDI, dan Spare Part. Pemisahan ini dilakukan berdasarkan karakteristik dan atribut khusus yang dimiliki oleh

masing-masing jenis produk. Setiap tipe memiliki atribut atau *field* yang berbeda, sehingga pemisahan model mempermudah dalam proses validasi, pencarian, serta pengolahan data.

Selain keempat skema utama tersebut, terdapat juga model pendukung seperti ‘brand’, ‘location’, dan ‘category spare part’. Model brand dan location memiliki relasi *one-to-one* dengan setiap tipe produk, artinya satu produk hanya terhubung dengan satu merek dan satu lokasi penyimpanan. Sementara itu, model ‘category spare part’ memiliki relasi *one-to-many* terhadap produk, di mana satu kategori dapat mencakup banyak produk Spare Part. Relasi antar model tergambar pada gambar 3.5.



Gambar 3.5 Relasi skema model produk

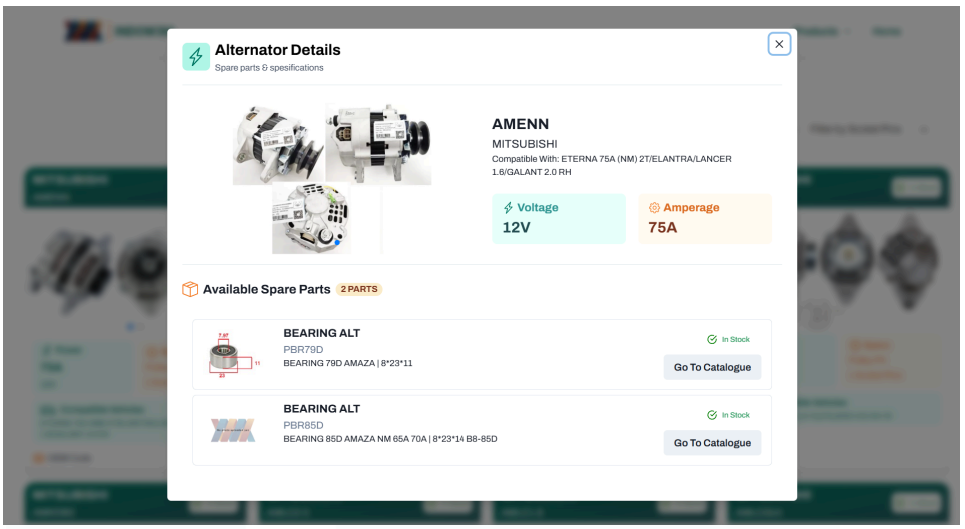
Setelah penyusunan model *database* selesai, tahap selanjutnya adalah pengembangan sisi *backend* yang mencakup pembuatan *controller* dan *route* untuk menangani operasi CRUD pada setiap tipe produk, seperti Alternator, Starter, CDI, dan Spare Part. Setiap *controller* bertanggung jawab dalam memproses *request* dari sisi *frontend*, melakukan validasi data, dan berinteraksi dengan database melalui Prisma ORM. Selain itu, dibuat pula *controller* khusus untuk proses autentikasi login, yang dilengkapi dengan *middleware* JWT dan cookies untuk mengelola sesi pengguna. Serta *middleware Role Checking*, untuk memastikan bahwa setiap permintaan ke *route* yang di *protect* hanya dapat diakses oleh pengguna yang telah terverifikasi, sesuai dengan sistem *Role-Based*

Access Control (RBAC) yang diterapkan sebelumnya. Implementasi *role checking* untuk *protected route* dapat dilihat pada gambar 3.6.

```
const router = Router();
router.get("/", getAllAlternator);
router.post("/code", getAlternatorWithCode);
router.post("/id", getAlternatorWithId);
router.post(
  "/create",
  verifyJWT,
  verifyRole(["SUPER_ADMIN", "ADMIN"]),
  createProduct
);
router.put(
  "/update",
  verifyJWT,
  verifyRole(["SUPER_ADMIN", "ADMIN"]),
  updateProduct
);

router.delete(
  "/delete",
  verifyJWT,
  verifyRole(["SUPER_ADMIN", "ADMIN"]),
  deleteProduct
);
```

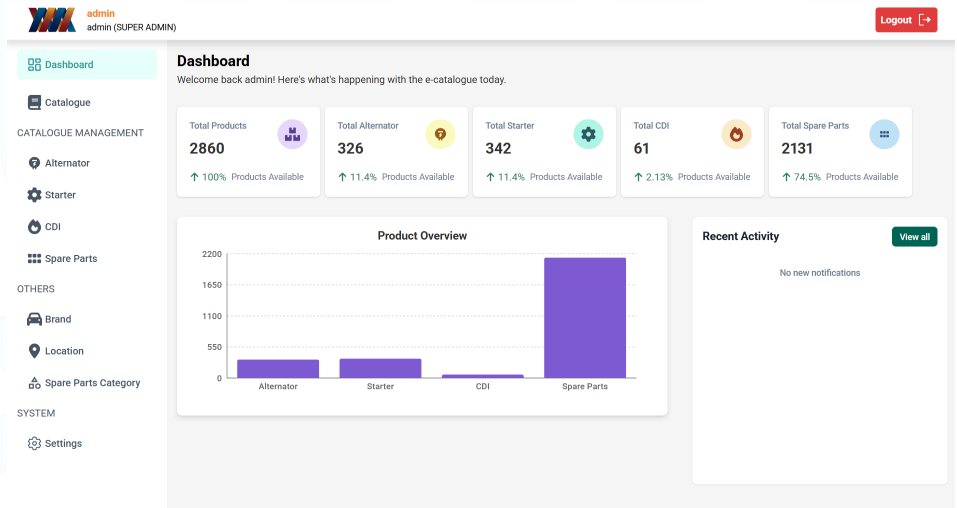
Gambar 3.6 Implementasi *role checking* pada *protected route*

Pada sisi *frontend*, tampilan aplikasi dibagi menjadi dua *panel* utama, yaitu sisi *client* dan sisi *admin*. Sisi *client* ditujukan untuk *guest* atau tim *sales* agar dapat mengakses katalog produk secara terbuka. Pada *panel* ini, pengguna dapat melihat informasi produk seperti Alternator, Starter, dan CDI, serta relasinya dengan komponen suku cadang yang terkait. Tampilan antarmuka detail produk katalog dapat dilihat pada gambar 3.7.



Gambar 3.7 Detail produk pada *e-Catalogue*

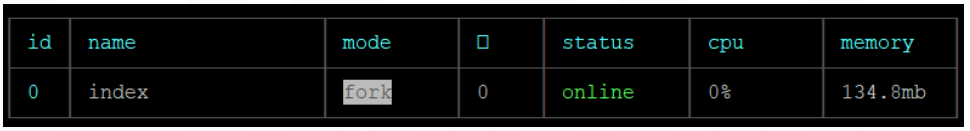
Sementara itu, sisi *admin* dikembangkan sebagai sebuah *Content Management System* (CMS) yang memungkinkan *admin* untuk melakukan CRUD terhadap produk, sekaligus mengelola relasi antara produk dengan suku cadang terkait. Melalui *panel* ini, *admin* dapat melihat berapa banyak barang melalui presentasi angka dan dapat memperbarui data seperti gambar, deskripsi, dan keterkaitan antar produk agar katalog selalu sesuai dengan kondisi inventaris terkini. Tampilan antarmuka CMS *admin* dapat dilihat pada gambar 3.8.



Gambar 3.8 Tampilan CMS pada panel *admin*

Setelah proses pengembangan selesai, tahap selanjutnya adalah *deployment* agar aplikasi dapat diakses dan digunakan secara luas oleh pengguna dan tim *sales*. Pada sisi *frontend*, penulis menggunakan layanan Cloudflare Pages sebagai platform untuk melakukan *deployment*. Pemilihan Cloudflare Pages didasarkan pada kemudahan integrasi dengan repositori Git, kecepatan dalam proses *build* dan distribusi, serta dukungan otomatis terhadap konten statis seperti aplikasi React. Selain itu, Cloudflare Pages juga menyediakan fitur CDN (*Content Delivery Network*) bawaan yang memungkinkan akses *frontend* menjadi lebih cepat dan stabil dari berbagai lokasi.

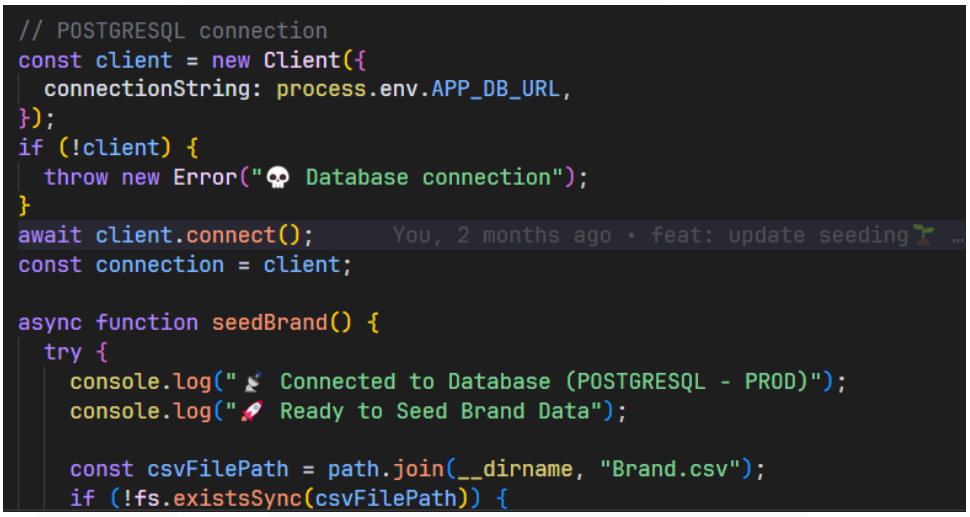
Untuk sisi *backend*, *deployment* dilakukan menggunakan *Virtual Machine* (VM) dari Amazon Web Services (AWS), seperti pada gambar 3.9. Aplikasi backend dijalankan dengan metode git *pull* langsung dari *repository*, kemudian dijalankan di latar belakang menggunakan PM2, sebuah *process manager* untuk aplikasi Node.js yang memungkinkan proses tetap berjalan. Untuk mengatur akses dan koneksi dari *domain* publik ke *server backend*, menggunakan Caddy sebagai *reverse proxy*.



Gambar 3.9 Status backend menggunakan PM2

Setelah tahap *deployment*, langkah terakhir yang dilakukan adalah proses *seeding* data ke dalam *database production* menggunakan *file* CSV berisi data produk yang telah difilter dari informasi sensitif. Proses ini dilakukan dengan bantuan *script custom* berbasis TypeScript, yang dapat dilihat pada gambar 3.10, untuk memastikan struktur data sesuai dengan skema yang ada. Setelah semua data dan sistem siap, dilakukan demo aplikasi bersama tim *marketing* dan *sales* sebagai pengguna utama, untuk

memastikan bahwa aplikasi berjalan sesuai kebutuhan dan dapat digunakan secara optimal di lapangan.



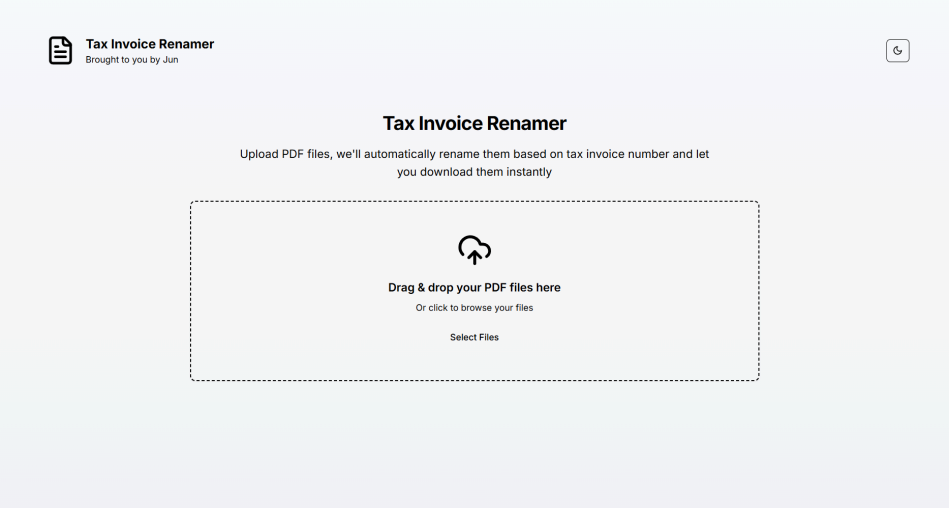
Gambar 3.10 Potongan kode *Script custom* untuk seeding *database*

3.2.4 Project 2: Tax Namer

Tax Namer adalah aplikasi berbasis web yang dikembangkan untuk menjawab permasalahan efisiensi kerja di bagian *accounting*, khususnya pada perpajakan. Aplikasi ini hadir sebagai solusi atas permasalahan yang dihadapi staf *accounting* dalam mengelola *file invoice* pajak yang dihasilkan oleh sistem eksternal, Coretax. Format penamaan *file* yang dihasilkan secara default sering kali tidak sistematis dan sulit dikenali, sehingga menyulitkan proses pencarian, pengarsipan, dan pelacakan dokumen.

Sebagai proyek kedua selama masa magang, pengembangan Tax Namer dimulai pada tanggal 6 Maret 2025. Tujuan utama dari aplikasi ini adalah untuk meningkatkan efisiensi dan akurasi dalam proses penamaan *file*, dengan mengotomatiskan proses konversi nama *file* ke format yang lebih informatif dan seragam. Aplikasi ini dikembangkan dengan antarmuka sederhana namun fungsional seperti yang terlihat pada gambar 3.11, yang memungkinkan pengguna untuk mengunggah *file*, dan langsung

mengunduh *file* yang telah diperbarui tanpa perlu instalasi atau login tambahan. Dengan solusi ini, bagian perpajakan dapat menghemat waktu kerja secara signifikan dan meminimalkan kesalahan dalam pengelolaan dokumen.



Gambar 3.11 Tampilan halaman Tax Namer

Pada awalnya, proses penamaan *file invoice* dilakukan secara manual oleh staf di bagian *accounting*. Proses ini memerlukan pembukaan dua *tab* secara bersamaan, yaitu *tab* untuk melihat isi dokumen dan *tab folder* penyimpanan *file*. Staf harus secara teliti mencari nomor referensi, nomor invoice, dan nomor NPWP (jika tersedia) dari isi dokumen tersebut, kemudian menyusunnya secara manual sebagai nama *file*. Format penamaan yang digunakan mengikuti pola: 'Referensi-No.Invoice-No.NPWP'. Proses ini memakan waktu cukup lama, terutama ketika harus dilakukan terhadap puluhan *file* dalam satu waktu, dan sangat rentan terhadap kesalahan penamaan maupun ketidakkonsistenan format. Hasil format dapat dilihat pada gambar 3.12.

Gambar 3.12 Hasil format *file* invoice Tax Namer

Aplikasi Tax Namer dikembangkan menggunakan Next.js, sebuah *framework full-stack* berbasis React yang mendukung pengembangan *frontend* dan *backend* dalam satu *environment*. Next.js dikenal karena kemampuannya dalam *rendering* sisi *server* (*Server-Side Rendering* atau SSR), routing otomatis, serta dukungan bawaan terhadap API *route* yang memudahkan pembuatan *endpoint backend* tanpa perlu *separate configuration*. *Framework* ini dipilih karena kesederhanaannya dalam struktur proyek, efisiensi dalam pengelolaan logika *server* dan *client*, serta kompatibilitas tinggi dengan berbagai *library modern* yang dibutuhkan dalam pengembangan aplikasi berbasis web.

Meskipun bersifat *full-stack*, struktur aplikasi tetap dapat dijelaskan dalam dua bagian utama seperti aplikasi web pada umumnya. Pada sisi *frontend*, antarmuka pengguna dibangun menggunakan Tailwind CSS untuk *styling* dan komponen UI dari ShadCN, yang memberikan tampilan profesional dan konsisten.

Sedangkan pada sisi *backend*, aplikasi memanfaatkan beberapa dependensi utama, salah satunya pdf-lib (pdf-ts), untuk membaca dan memproses isi *file* PDF *invoice*. Proses ini dilakukan untuk mengekstrak data penting seperti nomor referensi, nomor *invoice*, dan NPWP yang kemudian digunakan sebagai dasar dalam penamaan *file* secara otomatis.

Dalam proses otomatisasi penamaan *file*, aplikasi Tax Namer menerima *file* PDF dari sisi *frontend* melalui fungsi bernama ‘handleFiles’. Fungsi ini bertugas untuk memproses *file-file* yang dipilih oleh pengguna dan memfilter hanya *file* dengan tipe application/pdf. Setiap *file* yang valid kemudian dikemas ke dalam struktur ‘FileItem’, yang berisi informasi penting seperti nama, ukuran, tipe, dan status pemrosesan *file*. Struktur ini juga menyertakan ID unik untuk keperluan pelacakan dan manajemen status tiap *file* dalam daftar. Setelah *file* di filter dan disiapkan, seluruh *file*

tersebut dimasukkan ke dalam *array* 'FileItem'. Selanjutnya, array ini dikirimkan ke fungsi berikutnya yaitu 'processFiles'. Potongan kode fungsi 'handleFiles' dapat dilihat pada gambar 3.13.

```
const handleFiles = async (fileList: FileList) => {
  const newFiles: FileItem[] = [];

  for (let i = 0; i < fileList.length; i++) {
    const file = fileList[i];
    if (file.type === "application/pdf") {
      const fileItem: FileItem = {
        id: Math.random().toString(36).substring(2, 9),
        name: file.name,
        size: file.size,
        type: file.type,
        file,
        progress: 0,
        status: "uploading",
      };
      newFiles.push(fileItem);
    }
  }

  if (newFiles.length > 0) {
    setFiles((prev) => [...prev, ...newFiles]);
    await processFiles(newFiles);
  }
};
```

Gambar 3.13 Potongan kode fungsi handleFiles

Fungsi processFiles bertanggung jawab untuk mengirimkan *file* ke sisi *backend* melalui API *endpoint*. Pada setiap iterasi, fungsi akan memperbarui status *file* secara bertahap untuk menampilkan progres pemrosesan kepada pengguna, dimulai dari status *uploading* hingga *complete*. *File* dikirim ke *server* menggunakan objek FormData, dan dikirimkan melalui permintaan POST ke *endpoint* '/api/upload'. Setelah dikirim, *file* akan diproses di sisi server untuk mengekstrak data penting seperti nomor referensi dan nomor *invoice*. Jika berhasil, sistem akan menerima respons berupa nama baru *file* yang telah diproses dan mengubah statusnya menjadi complete. Namun jika terjadi kesalahan dalam proses pengiriman atau pengolahan, status *file* akan ditandai sebagai error dan pesan kesalahan akan ditampilkan kepada pengguna. Potongan kode fungsi 'processFiles' dapat dilihat pada gambar 3.14.

```
const processFiles = async (filesToProcess: FileItem[]) => {
  for (const fileItem of filesToProcess) {
    // Update status to uploading with progress
    updateFileStatus(fileItem.id, "uploading", 30);

    try {
      // Create a FormData object to send the file
      const formData = new FormData();
      formData.append("file", fileItem.file);

      updateFileStatus(fileItem.id, "uploading", 60);

      // Send the file to our API endpoint
      const response = await fetch("/api/upload", {
        method: "POST",
        body: formData,
      });

      updateFileStatus(fileItem.id, "processing", 80);

      if (!response.ok) {
        const errorData = await response.json();
        throw new Error(errorData.error || "Failed to process file");
      }

      const data = await response.json();
```

Gambar 3.14 Potongan kode fungsi processFiles

Pada sisi *backend*, *file* PDF yang dikirim dari fungsi 'processFiles' akan diterima dan diproses oleh *endpoint* POST dalam API *handler*. Pertama-tama, *file* diekstraksi dari objek FormData dan dikonversi menjadi *buffer* agar dapat dianalisis. Sistem secara eksplisit memeriksa validitas nama *file* dengan menolak *file* yang mengandung pola seperti "(1)" atau "- Copy" seperti pada gambar 3.15, untuk menghindari duplikasi dan kesalahan penamaan. Selanjutnya, isi *file* PDF dibaca menggunakan 'pdfToText', lalu dilakukan pencarian nomor referensi menggunakan pola *regular expression* (*regex*) untuk dijadikan bagian utama dari nama *file* baru.

Gambar 3.15 Penolakan nama *file* yang tidak sesuai

Untuk mengekstrak nomor referensi dari dokumen PDF, sistem terlebih dahulu mengubah isi *file* menjadi teks menggunakan *library* 'pdfToText', kemudian melakukan pencarian pola teks menggunakan *regular expression* (*regex*) '/\(Referensi:\s*([A-Za-z0-9-]{5,10})/' yang dapat dilihat pada gambar 3.16. *Regex* ini bekerja dengan mencocokkan *string* yang diawali dengan '(Referensi:' diikuti oleh kombinasi huruf, angka, atau tanda hubung sepanjang 5 hingga 10 karakter. Jika pola berhasil ditemukan, maka *substring* dari hasil pencocokan tersebut akan dipotong menggunakan fungsi '.slice(4)' untuk menghapus karakter awal yang tidak relevan dan menyisakan hanya kode referensinya.

```
try {
  const pdfText = await pdfToText(buffer);

  const match = pdfText.match(/\(Referensi:\s*([A-Za-z0-9-]{5,10})/);

  if (match && match[1]) {
    extractedText = match[1].slice(4);
  }
} catch (pdfError) {
  console.error("PDF parsing error:", pdfError);
}
```

Gambar 3.16 Potongan kode fungsi mencari nomor referensi

Sementara itu, untuk mendapatkan informasi nomor *invoice* dan nomor NPWP, sistem melakukan pengecekan terhadap panjang nama *file* asli menggunakan *if statement*. Jika panjang nama *file* lebih dari 94 karakter, maka diasumsikan bahwa bagian akhir dari nama *file* tersebut

mengandung NPWP, dan potongan tersebut diambil menggunakan '*slice(-38, -4)*', yaitu dari karakter ke-38 dari belakang hingga 4 karakter sebelum akhir nama *file* (menghapus ekstensi .pdf). Sebaliknya, jika panjang nama *file* lebih pendek, maka dianggap hanya mengandung nomor *invoice*, dan diambil menggunakan 'slice(-23, -6)'. Potongan kode fungsi untuk mencari nomor NPWP dan nomor *invoice* dapat dilihat pada gambar 3.17.

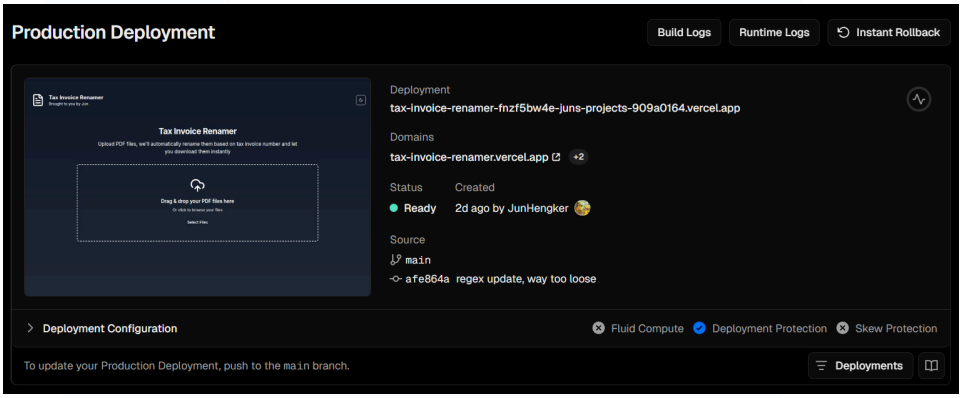
```
if (existingFileName.length > 94) {
  numWithoutPDF = existingFileName.slice(-38, -4);
} else {
  numWithoutPDF = existingFileName.slice(-23, -6);
}

const newFileName = `${extractedText}-${numWithoutPDF}.pdf`;
```

Gambar 3.17 Potongan kode fungsi mencari nomor NPWP dan nomor *invoice*

Setelah proses ekstraksi nomor referensi dan nomor *invoice* serta NPWP (jika ada) berhasil dilakukan, sistem menyusun nama *file* baru menggunakan format '${extractedText}-${numWithoutPDF}.pdf'. Format ini menggabungkan dua elemen penting, kode referensi dan nomor *invoice* serta NPWP dari nama *file*. Nama *file* baru ini kemudian dikirimkan kembali ke sisi *frontend* dalam bentuk respons JSON menggunakan NextResponse.json, dengan properti 'newName' berisi nama *file* hasil pemrosesan, dan status HTTP 200 sebagai penanda bahwa proses berhasil diselesaikan tanpa error. Langkah ini menutup seluruh alur kerja pemrosesan *file* PDF dari *upload* hingga pengembalian nama baru yang telah terstruktur secara otomatis.

Untuk proyek ini, tidak dilakukan sesi demo aplikasi secara formal karena pihak *accounting*, menilai bahwa tampilan dan penggunaan aplikasi Tax Namer sudah cukup sederhana dan *straight-forward*. Selain itu, proses *deployment* dilakukan menggunakan *platform* Vercel, yang merupakan layanan *cloud* yang secara otomatis terintegrasi dengan

Next.js. Proses deployment menjadi sangat efisien karena cukup dengan menghubungkan *repository* GitHub ke akun Vercel dan memberikan izin akses, sehingga setiap pembaruan pada kode secara otomatis ter-*deploy* tanpa perlu konfigurasi tambahan. Tampilan Vercel dapat dilihat pada gambar 3.18.



Gambar 3.18 Tampilan vercel

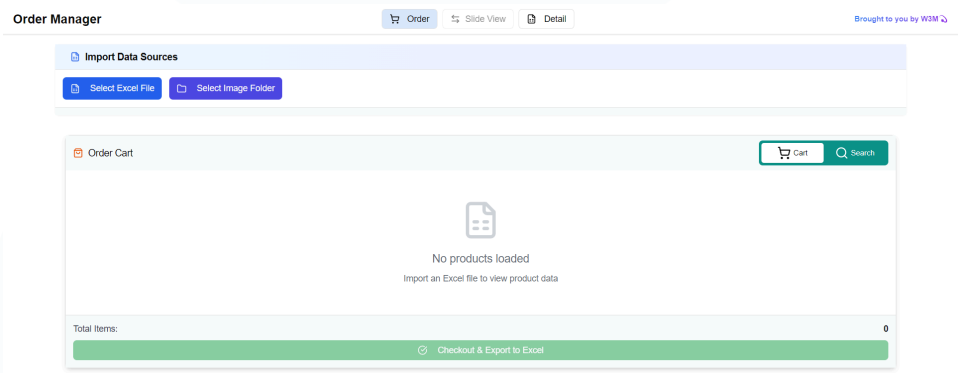
3.2.5 **Project 3: Order-helper**

Proyek Order-helper dikembangkan untuk menjawab permasalahan efisiensi waktu dan beban kerja yang dihadapi bagian *purchasing* di PT. Mitra Makmur Multindo. Proyek ini dimulai pada tanggal 8 Mei 2025 untuk menggantikan proses manual yang selama ini dilakukan oleh staf dalam menyusun *file order* bulanan. Tujuan utama dari aplikasi ini adalah menyederhanakan alur kerja dan mempersingkat waktu yang dibutuhkan dalam proses pembuatan *file order* bahan baku suku cadang.

Secara umum, setiap bulannya bagian *purchasing* menyusun *file order* sebagai dasar untuk melakukan *restock* bahan baku. Pada proses sebelumnya, staf perlu membuka beberapa sumber data terpisah secara manual, yaitu *database internal*, *file* Excel, serta *folder* gambar produk. Setelah menentukan barang-barang yang perlu di *order*, staf akan mulai menyusun *file* Excel secara manual. Langkah ini diawali dengan pencarian *data* barang berdasarkan ID produk pada sistem *internal*, kemudian mencocokkannya dengan riwayat pembelian sebelumnya. Melalui

informasi tersebut, staf dapat mengevaluasi kenaikan harga dari *supplier* dan mempertimbangkan keputusan pembelian.

Jika informasi harga sudah sesuai, staf akan memasukkan data barang ke dalam *file* Excel, lengkap dengan berbagai atribut penting seperti *part number* perusahaan, *part number* dari *supplier*, kuantitas pesanan, harga satuan, mata uang yang digunakan, serta nama *supplier*. Selain data numerik, staf juga perlu menambahkan gambar produk, yang diperoleh secara manual dengan membuka *folder* gambar, mencari *file* yang sesuai, lalu menyisipkannya ke dalam *file* Excel. Proses ini sering kali memakan waktu lama dan rentan terhadap kesalahan, terutama ketika jumlah *item* yang dipesan cukup banyak. Oleh karena itu, pengembangan aplikasi Order-helper menjadi solusi penting untuk menyatukan seluruh proses ini dalam satu sistem yang lebih efisien dan terstruktur. Tampilan aplikasi Order-helper dapat dilihat pada gambar 3.19.

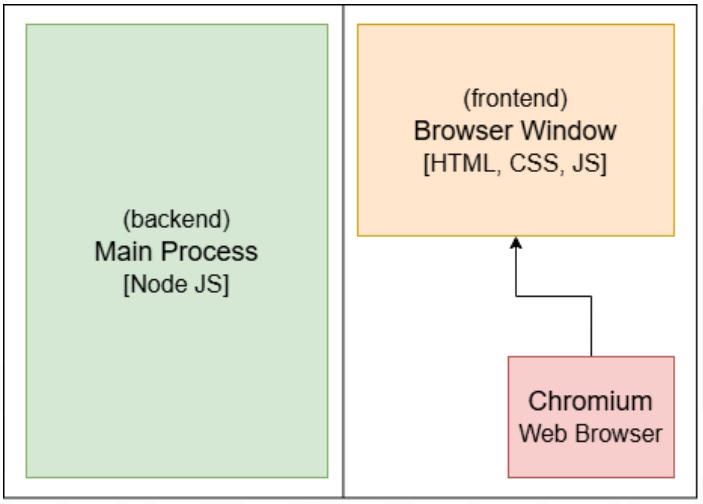


Gambar 3.19 Tampilan Order-helper

Khusus untuk proyek ini, Order-helper diminta untuk dikembangkan sebagai sebuah aplikasi desktop (*offline*) yang tidak diakses melalui internet dan tidak di *deploy* secara online. Aplikasi ini harus dapat dijalankan secara lokal dalam bentuk *file* .exe agar dapat digunakan secara langsung oleh staf *purchasing* tanpa bergantung pada koneksi jaringan,

serta menghasilkan output *file order* berupa Excel. Dengan ketentuan tersebut, penulis diberikan kebebasan dalam menentukan *framework* yang sesuai, asalkan hasil akhirnya sesuai dengan syarat dan permintaan sebelumnya.

Penulis memilih untuk menggunakan ElectronJS, yaitu sebuah *framework open-source* yang memungkinkan pengembangan aplikasi desktop menggunakan teknologi web seperti HTML, CSS, dan JavaScript. Electron menggabungkan Chromium sebagai *engine* untuk menampilkan antarmuka pengguna dan Node.js sebagai *runtime* untuk mengakses sistem *file* dan fitur *backend* secara lokal, seperti yang terlihat pada gambar 3.20. Dengan pendekatan ini, penulis dapat membangun aplikasi desktop dengan arsitektur dan gaya yang sama seperti aplikasi web, namun tetap berjalan sepenuhnya di sisi pengguna tanpa memerlukan *server*. Oleh karena itu, ElectronJS menjadi kandidat yang sangat tepat untuk proyek Order-helper karena mampu memenuhi kebutuhan perusahaan akan aplikasi desktop yang tidak bergantung pada jaringan internet.



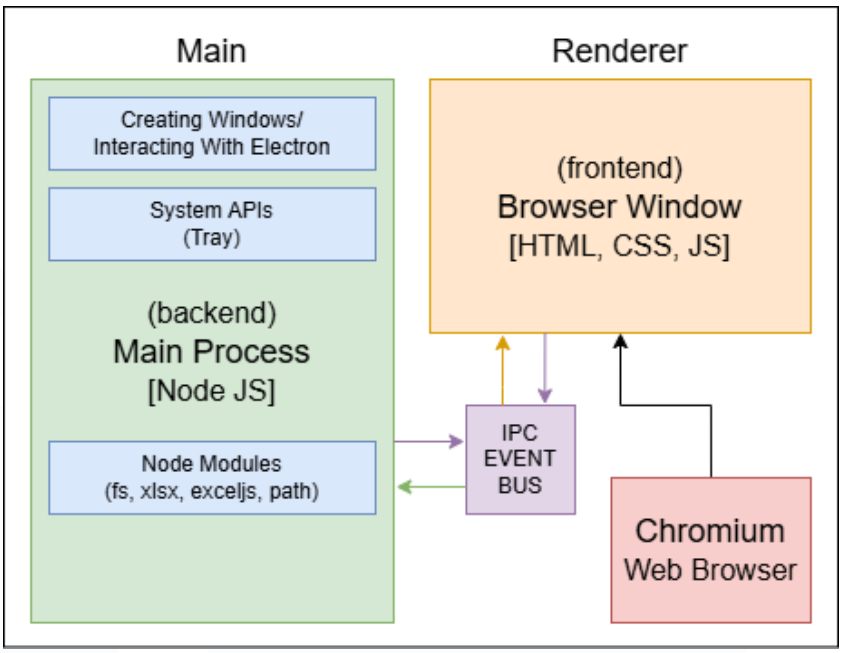
Gambar 3.20 Struktur aplikasi ElectronJS

Sebelum memulai proses pengembangan aplikasi, penulis perlu terlebih dahulu mempelajari dasar-dasar penggunaan ElectronJS. Proses pembelajaran dilakukan secara mandiri melalui berbagai sumber, di

antaranya dengan membaca dokumentasi resmi ElectronJS serta menonton beberapa video *tutorial* di YouTube yang membahas pengenalan konsep Electron, struktur proyek, hingga praktik pengembangan aplikasi desktop menggunakan ElectronJS.

Arsitektur ElectronJS terbagi menjadi dua proses utama, yaitu *Browser Window* (*Renderer Process*) yang dijalankan oleh Chromium, dan *Main Process* yang dijalankan oleh Node.js. Kedua proses ini bekerja secara *paralel* namun terpisah, sehingga tidak dapat berkomunikasi secara langsung. Proses *renderer* bertugas menangani tampilan antarmuka pengguna (UI), sedangkan *main process* bertanggung jawab atas fungsi-fungsi inti seperti mengelola *tab* aplikasi, akses sistem *file*, serta *logic backend*. Karena keduanya berada dalam ruang eksekusi yang berbeda, diperlukan mekanisme khusus agar mereka dapat saling bertukar informasi.

Untuk menangani komunikasi antar proses tersebut, ElectronJS menyediakan IPC (*Inter-Process Communication*) Event Bus. IPC memungkinkan proses *renderer* dan *main* untuk saling berkomunikasi melalui *channel* yang telah ditentukan. Misalnya, saat pengguna menekan tombol di antarmuka (*renderer*), informasi akan dikirim melalui IPC ke *main process* untuk melakukan tindakan tertentu seperti membaca *file*, kemudian hasilnya dikembalikan ke *renderer* untuk ditampilkan. Mekanisme ini menjembatani interaksi antara UI dan *logic* aplikasi secara terstruktur dan terisolasi. Diagram struktur komunikasi ElectronJS dapat dilihat pada gambar 3.21.

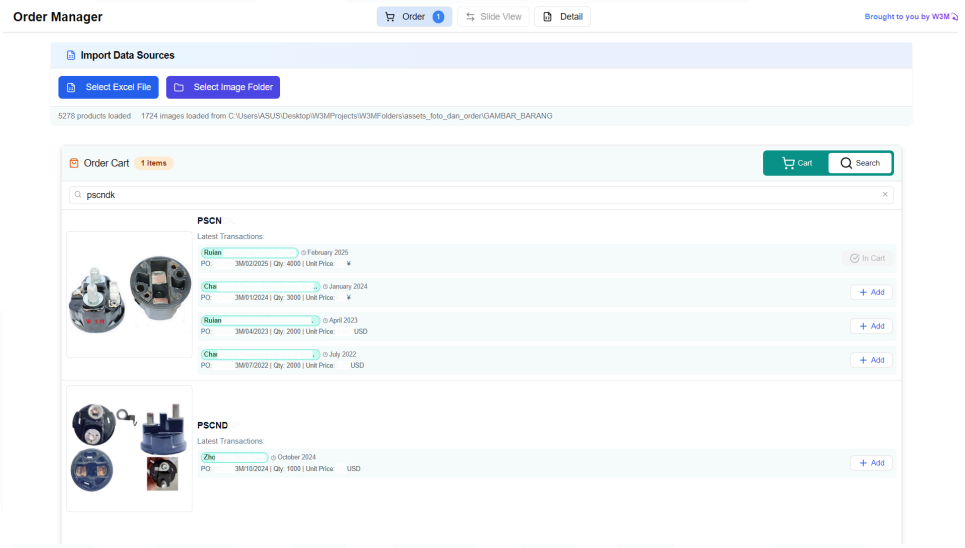


Gambar 3.21 Struktur komunikasi aplikasi ElectronJS

Proses pengembangan aplikasi Order-helper dibagi menjadi dua bagian utama, yaitu *backend* dan *frontend*. Pada sisi *backend*, aplikasi dikembangkan menggunakan Node.js dan memanfaatkan beberapa *library* (*node modules*) utama. *Library* 'fs' digunakan untuk mengakses dan memanipulasi sistem *file* lokal, seperti membaca *file* Excel serta mengambil gambar produk dari direktori yang ditentukan. *Library* 'xlsx' digunakan untuk membaca *file* Excel yang berisi riwayat pembelian sebelumnya, sedangkan 'exceljs' dimanfaatkan untuk menyusun dan menghasilkan *file* Excel baru sebagai *output* akhir yang akan digunakan staf *purchasing*. Selain itu, modul 'path' digunakan untuk mengelola dan menggabungkan *path file* dan *folder* yang nantinya digunakan untuk memanggil foto dari *folder* yang ditentukan.

Pada sisi *frontend*, aplikasi dikembangkan menggunakan React, dan untuk *styling* digunakan Tailwind CSS. Selain itu, penulis juga menggunakan ShadCN UI, sebuah *library* komponen UI berbasis Radix dan Tailwind yang memberikan tampilan modern sekaligus menjaga fleksibilitas dalam desain.

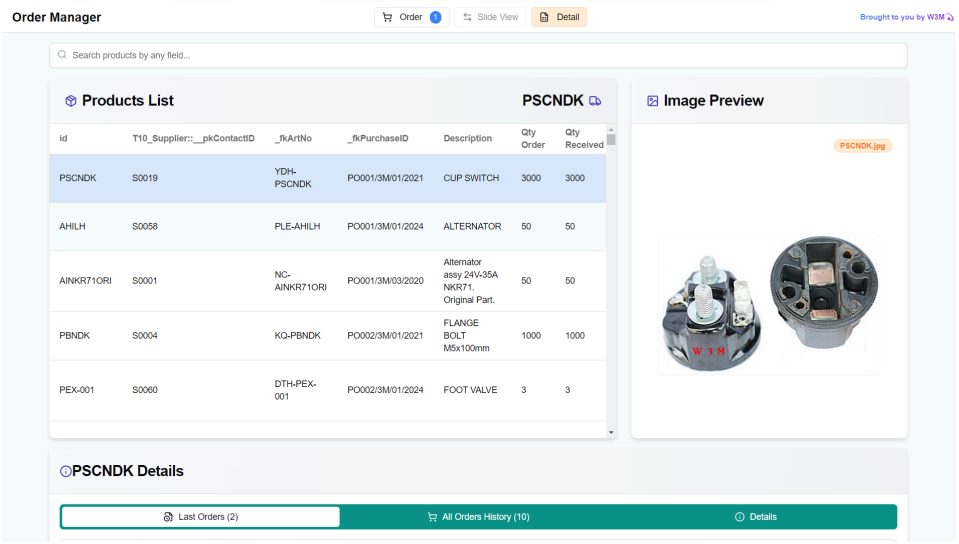
Pada bagian frontend, antarmuka aplikasi Order-helper dibagi menjadi dua panel utama, yaitu *panel* 'Order' dan *panel* 'Detail'. Pembagian ini merupakan hasil dari permintaan langsung staf *purchasing*, yang menginginkan tampilan antarmuka yang lebih terstruktur dan tidak membingungkan. Dengan adanya dua *panel* yang terpisah secara visual dan fungsional, informasi tidak tercampur dalam satu area, sehingga membantu mengurangi stres atau kebingungan saat digunakan dalam situasi kerja yang sibuk. *Panel* 'Order' digunakan untuk mencari produk yang akan dipesan dan menambahkannya ke daftar pesanan, sedangkan *panel* 'Detail' menampilkan informasi lengkap produk yang sedang dipilih, seperti riwayat pembelian, riwayat harga, mata uang yang dipakai, *part number*, hingga gambar produk. Tampilan panel *Order* dapat dilihat pada gambar 3.22.



Gambar 3.22 Tampilan *panel* Order

Sebelum aplikasi dapat digunakan, pengguna perlu memuat dua sumber data utama: *file* Excel riwayat pembelian dan *folder* gambar produk. Hal ini dilakukan karena kedua sumber data tersebut bersifat dinamis dan selalu diperbarui oleh *staf* di luar aplikasi. Riwayat pembelian berisi data harga dan transaksi sebelumnya yang penting untuk proses evaluasi harga, sedangkan *folder* gambar menyimpan dokumentasi visual

setiap produk. Daripada menyimpan ulang data tersebut dalam sistem aplikasi yang memerlukan *maintain* tambahan, aplikasi Order-helper justru mengakses langsung *file* dan *folder* yang telah diperbarui, sehingga lebih praktis dan efisien. Tampilan *panel* Detail dapat dilihat pada gambar 3.23



Gambar 3.23 Tampilan *panel* Detail

Pada bagian backend, terdapat beberapa fungsi utama, pertama , fungsi ‘selectExcelFile’ yang dapat dilihat pada gambar 3.24, fungsi ini bertugas untuk membuka dialog *file explorer* pada jendela utama (*mainWindow*) guna memilih *file* dengan ekstensi khusus .xlsx, .xls, atau .csv. Setelah pengguna memilih *file*, sistem akan membaca kontennya menggunakan *library* XLSX, kemudian mengonversi data pada *sheet* pertama menjadi *format* JSON. Hasil dari *file* Excel kemudian disimpan kedalam variabel ‘excelData’, serta *path file* disimpan secara lokal menggunakan modul *Store* agar dapat diakses kembali jika diperlukan.

```
ipcMain.handle("selectExcelFile", async () => {
  try {
    const result = await dialog.showOpenDialog(mainWindow, {
      properties: ["openFile"],
      filters: [{ name: "Excel Files", extensions: ["xlsx", "xls", "csv"] }],
    });

    if (result.canceled) {
      return "";
    }

    const fileContent = fs.readFileSync(result.filePaths[0]);
    const workbook = XLSX.read(fileContent, { type: "buffer" });
    const sheetName = workbook.SheetNames[0];
    const sheet = workbook.Sheets[sheetName];
    const rows = XLSX.utils.sheet_to_json(sheet);
```

Gambar 3.24 Potongan kode fungsi selectExcelFile

Fungsi ‘selectImageFiles’ dan ‘getImageFiles’ pada sisi *backend* berperan untuk memuat gambar produk. Fungsi ‘selectImageFiles’ digunakan untuk membuka *dialog* pemilihan folder, di mana pengguna dapat memilih direktori yang berisi *file* gambar produk. Setelah *folder* dipilih, seluruh *file* gambar dibaca dan dimasukkan ke dalam struktur ‘imageMap’, lengkap dengan atribut seperti id, *path*, dan data gambar. *Path folder* tersebut juga disimpan secara lokal menggunakan modul *Store*. Sementara itu, fungsi ‘getImageFiles’ yang dapat dilihat pada gambar 3.25, digunakan untuk membaca ulang *file* gambar dari *folder* yang telah dipilih, memfilter *file* berdasarkan ekstensi gambar (seperti .jpg, .png), lalu mengubahnya ke *format* base64 agar dapat ditampilkan di sisi *frontend* dalam bentuk preview gambar.

```
ipcMain.handle("getImageFiles", async (_, folderPath: string) => {
  try {
    const files = fs.readdirSync(folderPath);
    const imageFiles = files.filter((file) =>
      /\.(png|jpe?g|gif|bmp)$/i.test(file)
    );

    const images = imageFiles.map((file) => {
      const filePath = path.join(folderPath, file);
      const imageBuffer = fs.readFileSync(filePath);
      const base64Image = `${imageBuffer.toString("base64")}`;

      return {
        id: path.basename(file, path.extname(file)),
        name: file,
        dataUrl: base64Image,
      };
    });

    return images;
  } catch (error) {
    console.error("Error reading image files:", error);
    return [];
  }
});
```

Gambar 3.25 Potongan kode fungsi getImageFiles

Fungsi ‘exportExcelFile’ merupakan bagian akhir dari alur kerja aplikasi Order-helper, yang bertugas untuk menyusun dan menghasilkan *file* Excel output sesuai ketentuan *format order*. Fungsi ini menerima data dari keranjang pesanan (*cart*) dan memanfaatkan *library* exceljs untuk membuat *worksheet* baru. Data dari setiap *item* akan dimasukkan sebagai baris baru, mencakup informasi seperti *part number* perusahaan, *part number* dari supplier, kuantitas pesanan, harga satuan, mata uang yang digunakan, serta nama supplier. Kolom gambar juga disiapkan dan diisi secara otomatis. Pengguna akan diminta memilih lokasi penyimpanan file melalui *dialog save*, dan jika disetujui, *file* Excel akan langsung ditulis ke *path* tersebut. Potongan kode fungsi ‘exportExcelFile’ dapat dilihat pada gambar 3.26.

```
const filePath = await dialog.showSaveDialog({
  filters: [{ name: "Excel Files", extensions: ["xlsx"] }],
});

if (filePath.canceled || !filePath.filePath) return;

await workbook.xlsx.writeFile(filePath.filePath);

// ✅
return { status: 200, message: "Excel file exported successfully" };
```

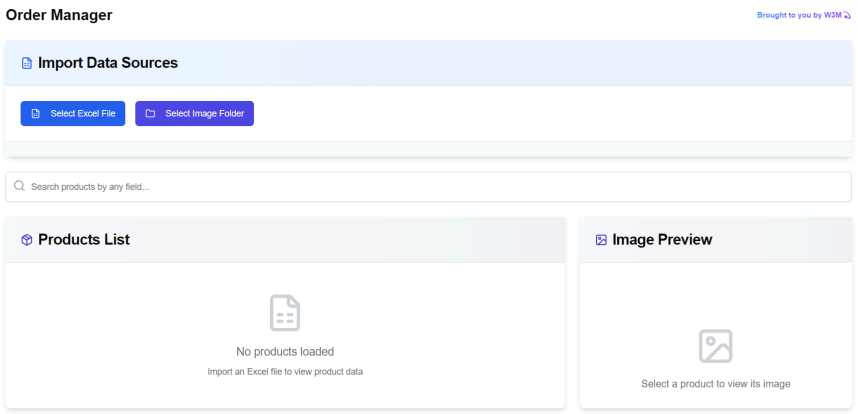
Gambar 3.26 Potongan kode fungsi exportExcelFile

Seluruh fungsi *backend* seperti 'selectExcelFile', 'selectImageFiles', 'getImageFiles', dan 'exportExcelFile' di-*handle* oleh 'ipcMain', yang berfungsi sebagai jembatan komunikasi antarproses (*inter-process communication*/IPC) di sisi *main process*. Karena dalam arsitektur ElectronJS proses *renderer* (yang menjalankan UI di dalam Chromium) tidak dapat langsung mengakses modul bawaan Node.js seperti fs, path, atau dialog, maka semua logika dan interaksi sistem dilakukan melalui 'ipcMain'. Agar proses *renderer* dapat mengakses fungsi-fungsi ini secara aman, mereka diekspos melalui 'contextBridge' pada saat *preload script* dijalankan, penggunaan 'contextBridge' dapat dilihat pada gambar 3.27. *Context bridge* ini bertugas untuk menyediakan API kepada *renderer* tanpa membuka akses langsung ke seluruh modul Node.js, sehingga menjaga keamanan aplikasi.

```
const { ipcRenderer, contextBridge, webFrame } = require("electron");

//Expose electron to the window object
contextBridge.exposeInMainWorld("electron", {
  selectExcelFile: (...args: Parameters<selectExcelFile>) =>
    ipcRenderer.invoke("selectExcelFile", ...args),
  selectImageFiles: (...args: Parameters<selectImageFiles>) =>
    ipcRenderer.invoke("selectImageFiles", ...args),
  getImageFiles: (...args: Parameters<getImageFiles>) =>
    ipcRenderer.invoke("getImageFiles", ...args),
  exportToExcel: (...args: Parameters<exportToExcel>) =>
    ipcRenderer.invoke("exportExcelFile", ...args),

  // retreive paths
  getExcelFilePath: (...args: Parameters<getExcelFilePath>) =>
    ipcRenderer.invoke("getExcelFilePath", ...args),
  getImageFolderPath: (...args: Parameters<getImageFolderPath>) =>
    ipcRenderer.invoke("getImageFolderPath", ...args),
} as Window["electron"]);
```

Gambar 3.27 *Exposing* fungsi menggunakan *contextBridge*

Selama proses pengembangan, terjadi beberapa revisi, khususnya pada tampilan antarmuka pengguna (UI) dan *layout output file* Excel. Revisi ini dilakukan untuk menyesuaikan dengan preferensi staf *purchasing* serta memastikan hasil akhir sesuai dengan standar dokumen yang biasa digunakan di perusahaan. Tampilan Order-helper sebelum revisi dapat dilihat pada gambar 2.8.



Gambar 3.28 Tampilan Order-helper sebelum revisi

Demo aplikasi dilakukan dalam satu hari bertepatan dengan *meeting final* bersama staf *purchasing*. Pada sesi ini, staf langsung dapat menggunakan aplikasi tanpa kesulitan, karena alur kerja dan tampilan antarmuka telah dirancang sesuai dengan permintaan dan kebiasaan kerja mereka sejak awal pengembangan, sehingga aplikasi terasa “ngalir” dan mudah digunakan.

3.3 Kendala yang Ditemukan

Selama menjalani kegiatan magang di PT. Mitra Makmur Multindo, penulis menghadapi sejumlah kendala yang muncul, kendala-kendala ini antara lain:

- **Dokumentasi yang kurang memadai**

  Seluruh proyek yang dikerjakan tidak didukung oleh dokumentasi teknis atau panduan pengembangan resmi dari perusahaan. Ketiadaan *design guideline* dan standar alur kerja (*workflow*) memungkinkan terjadinya perbedaan desain antarmuka maupun logika antar aplikasi di masa depan.

- **Data yang kurang memadai**

  Data produk yang tersedia pada awal pengembangan proyek belum sepenuhnya lengkap atau siap pakai. *Intern* perlu melakukan proses filterisasi dan penyesuaian manual terhadap data yang akan digunakan, seperti menghapus informasi sensitif, menyusun ulang format, dan mengelompokkan produk berdasarkan kebutuhan aplikasi. Proses ini menambah beban kerja dan memakan waktu tambahan di luar tugas pengembangan sistem, yang seharusnya dapat diminimalkan apabila data sudah disiapkan dalam format yang sesuai sejak awal.

- **Keterbatasan sumber daya manusia**

  Seluruh proses pengembangan proyek dilakukan oleh *intern* secara mandiri sebagai *full-stack developer*, mulai dari perancangan *frontend* hingga pembuatan *logic backend*. Meskipun sesuai dengan cakupan pekerjaan *full-stack*, tidak adanya dukungan khusus dari tim UI/UX menyebabkan proses perancangan UI memerlukan waktu lebih lama dan meningkatkan beban kerja.

- **Penambahan feature mendadak**

  Selama proses pengembangan, seringkali terjadi penambahan fitur secara mendadak berdasarkan kebutuhan dari pengguna. Meskipun penambahan fitur merupakan hal positif untuk meningkatkan fungsi aplikasi, beberapa fitur yang diajukan bersifat cukup besar dan berdampak pada arsitektur utama aplikasi. Hal ini mengharuskan *intern* untuk melakukan peninjauan ulang dan penyesuaian menyeluruh, yang pada akhirnya menambah waktu

pengerjaan dan risiko terjadinya *bug* di bagian lain yang sebelumnya telah selesai.

3.4 Solusi atas Kendala yang Ditemukan

Untuk mengatasi kendala-kendala yang muncul selama proses magang, penulis menerapkan beberapa solusi yang bertujuan untuk meminimalisir dampak dari masing-masing permasalahan. Solusi ini bersifat praktis dan disesuaikan dengan kondisi kerja di lapangan, sehingga tetap mendukung kelancaran proyek yang sedang dikembangkan. Berikut adalah beberapa solusi atas kendala yang ditemukan:

- **Membuat guideline sendiri**

  Untuk mengatasi dokumentasi yang kurang memadai, *intern* mengambil inisiatif dengan menyusun *guideline internal* sendiri. *Guideline* ini mencakup struktur folder, pola desain UI, dan alur *logic* aplikasi. Dengan adanya panduan ini, proses pengembangan menjadi lebih terarah dan konsisten. Selain itu, guideline ini juga dapat digunakan sebagai acuan apabila pengembangan proyek dilanjutkan oleh pihak lain di masa depan, sehingga meminimalisir terjadinya ketidaksimetrisan antar aplikasi *internal*.

- **Filterisasi data mandiri**

  Untuk mengatasi keterbatasan data produk, *intern* membuat beberapa *script* Python khusus untuk melakukan filterisasi dan reformasi data secara otomatis. Data yang awalnya tidak terstruktur diformat ulang menjadi *file* CSV yang lebih bersih dan konsisten. Proses ini juga bertujuan untuk memastikan bahwa data yang digunakan bersifat publik, tanpa menyertakan informasi sensitif perusahaan. Selain mendukung keamanan data, hasil filterisasi ini juga digunakan sebagai dasar pengembangan pada aplikasi lain yang membutuhkan sumber data produk yang sudah siap pakai.

- **Inisiatif desain UI mandiri**

  Untuk mengatasi keterbatasan tidak adanya tim UI/UX, *intern* mengambil inisiatif untuk membuat desain antarmuka secara mandiri menggunakan Figma. Desain komponen UI direferensikan dari berbagai sumber di internet, kemudian dimodifikasi agar sesuai dengan kebutuhan dan konteks penggunaan di lingkungan perusahaan. Pendekatan ini membantu mempercepat proses pengembangan *frontend* serta menjaga konsistensi tampilan aplikasi meskipun tidak didampingi oleh tim desain.

- **Konsep modular pada pengembangan aplikasi**

  Untuk menghadapi kemungkinan penambahan fitur mendadak berskala besar, *intern* menerapkan konsep modularitas dalam pengembangan aplikasi. Struktur kode dibagi berdasarkan fungsi dan kegunaan, mulai dari pemisahan *controller*, *route*, hingga komponen-komponen *frontend*. Dengan pendekatan ini, setiap bagian aplikasi dikelola secara terpisah dan terstruktur, sehingga memudahkan proses peninjauan dan pengembangan lanjutan tanpa harus mengubah keseluruhan sistem. Modularitas juga membantu *intern* dalam melakukan integrasi fitur baru tanpa mengganggu logika yang telah berjalan.