BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Dalam program *Humanity Project*, penulis ditempatkan pada divisi *Data & Technology* dengan peran sebagai *Full-stack Developer Intern*. Tanggung jawab utama posisi ini meliputi pengembangan sistem berbasis web untuk pemantauan distribusi logistik bagi pengungsi pascabencana. Selain itu, penulis juga berkontribusi dalam perancangan dan pengembangan *backend* aplikasi RUinRISK, yang berfungsi sebagai media edukasi guna meningkatkan kewaspadaan masyarakat terhadap potensi bencana, khususnya di wilayah Lebak Selatan.

Pelaksanaan kerja magang menerapkan sistem *hybrid*, dengan rincian jadwal kunjungan yang dipaparkan pada Bagian 1.3. Seluruh kegiatan dilaksanakan di bawah supervisi langsung Direktur GMLS, Anis Faisal Reza. Koordinasi dengan *supervisor* dilakukan secara intensif, baik melalui pertemuan tatap muka saat kunjungan ke Lebak Selatan maupun komunikasi daring menggunakan platform WhatsApp.

3.2 Tugas yang Dilakukan

Tugas yang dilaksanakan meliputi pengembangan lanjutan *User Interface* (UI) dan *database* pada sistem berbasis web GMLS SYS. Sistem ini berfungsi sebagai sarana pemantauan distribusi logistik pascabencana, guna memastikan pemerataan dan ketepatan alokasi bantuan di setiap lokasi pengungsian.

Selain itu, dilakukan pula perancangan serta pengembangan struktur *database* dan *backend* untuk aplikasi *mobile* RUinRISK. Aplikasi ini berperan sebagai media edukasi untuk meningkatkan *awareness* masyarakat terhadap potensi bahaya dan risiko kebencanaan di lingkungan sekitar.

3.2.1 Uraian Kerja Magang

Rincian fitur sistem GMLS SYS yang dikembangkan selama periode kerja magang disajikan dalam Tabel 3.1.

Tabel 3.1. Penjelasan Tugas Singkat GMLS SYS

Fitur	Fungsi
Dashboard Logistik Admin	Sebagai antarmuka bagi admin GMLS untuk mengatur parameter logistik serta mengelola akun pengguna.
Form Input Pengungsi	Untuk mendaftarkan pengungsi bencana pada setiap tempat evakuasi akhir (TEA) yang ada, data yang didaftarkan mencakup, nama, jenis kelamin, kategori usia, NIK (jika ada), nomor telepon (jika ada), serta kondisi fisik. Nantinya kebutuhan logistik setiap orang pada suatu TEA akan dikalkulasikan secara otomatis berdasarkan jenis kelamin, usia, dan kondisi fisik yang sebelumnya telah didaftarkan.
Dashboard Relawan	Sebagai antarmuka bagi relawan untuk memantau jumlah pengungsi, kebutuhan total logistik, serta stok logistik yang tersedia pada lokasi TEA yang ada.
Database GMLS SYS	Menyimpan data-data *user* seperti admin dan relawan serta menyimpan data logistik dan data pengungsi.

Selanjutnya, uraian fitur aplikasi RUinRISK yang dikembangkan selama pelaksanaan magang dapat dilihat pada Tabel 3.2.

Tabel 3.2. Penjelasan Tugas Aplikasi RUinRISK

Fitur	Fungsi
Database Aplikasi	Menyimpan data *user* serta data-data lain dari fitur yang ada pada aplikasi.
Autentikasi	Fungsi *login* (masuk), *register* (pendaftaran akun), serta autentikasi berbasis Google.

3.3 Uraian Pelaksanaan Magang

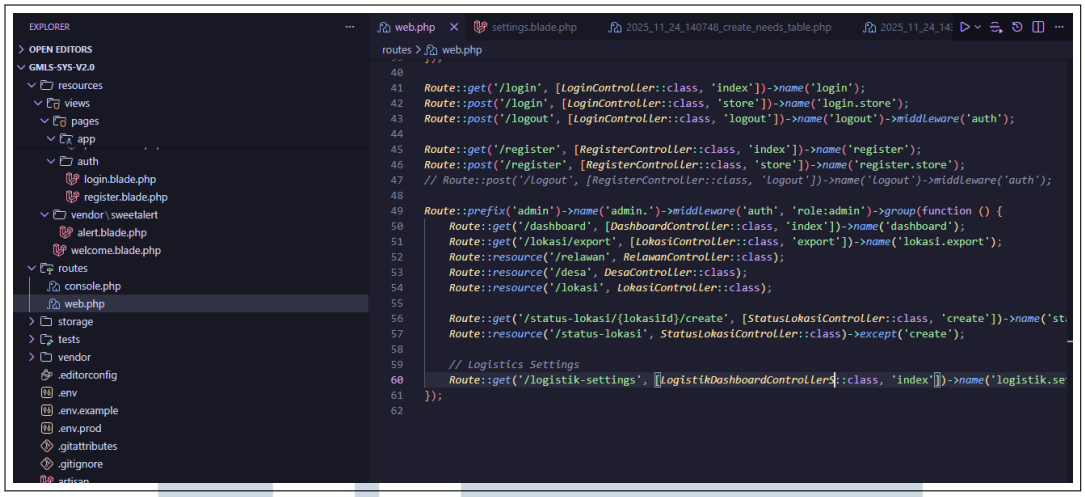
Pelaksanaan kerja magang ini berlangsung dalam kurun waktu 17 minggu (setara dengan empat bulan). Adapun rincian aktivitas yang dilakukan setiap minggunya disajikan dalam Tabel 3.3.

Tabel 3.3. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke-	Pekerjaan yang dilakukan
1 – 2	Mempelajari dan memahami *source code* situs GMLS SYS sebagai dasar pengembangan sistem.
3 – 4	Menganalisis kebutuhan sistem *backend* dan struktur *database* situs GMLS SYS yang sesuai dengan fungsionalitas aplikasi, meliputi pengelolaan data, alur proses bisnis, serta kebutuhan keamanan dan kinerja sistem.
5 – 7	Merancang sistem *backend* dan struktur *database* situs GMLS SYS berdasarkan hasil analisis kebutuhan yang telah dilakukan.
8 – 9	Mengimplementasikan sistem *backend* dan *database* ke dalam sistem situs GMLS SYS sesuai dengan rancangan yang telah dibuat.
10	Melakukan *testing* untuk memastikan sistem *backend* dan *database* situs GMLS SYS berjalan sesuai dengan kebutuhan yang telah ditentukan.
11 – 12	Menganalisis kebutuhan sistem *backend* dan struktur *database* pada aplikasi RUinRISK sebagai dasar perancangan sistem.
13 – 14	Merancang sistem *backend* dan struktur *database* aplikasi RUinRISK berdasarkan hasil analisis kebutuhan.
15 – 16	Mengimplementasikan sistem *backend* dan *database* aplikasi RUinRISK sesuai dengan rancangan yang telah dibuat.
17	Melakukan *testing* untuk memastikan sistem *backend* dan *database* aplikasi RUinRISK berjalan sesuai dengan kebutuhan dan rancangan sistem.

3.3.1 Pemahaman Sistem

Pada minggu ke-1 hingga minggu ke-2, kegiatan yang dilakukan adalah mempelajari dan memahami *source code* situs GMLS SYS yang telah ada seperti pada Gambar 3.1.



Gambar 3.1. *Source code* situs GMLS SYS.

Tahap ini bertujuan untuk memahami struktur proyek, alur sistem, serta keterkaitan antara sistem *backend* dan *database* sehingga dapat menjadi dasar dalam proses pengembangan selanjutnya.

3.3.2 Analisis Kebutuhan Sistem

Pada tahap analisis kebutuhan sistem, dilakukan identifikasi terhadap kebutuhan sistem *backend* dan struktur *database* yang akan dikembangkan. Analisis ini bertujuan untuk memastikan sistem yang dibangun mampu mendukung fungsionalitas aplikasi, alur *business logic*, serta kebutuhan keamanan dan kinerja sistem.

A Sistem GMLS SYS

Hasil analisis kebutuhan sistem *backend* dan *database* pada situs GMLS SYS meliputi:

- Kebutuhan penggunaan *framework* Laravel sebagai *backend framework* untuk mengelola *business logic* dan pengembangan *API*.
- Kebutuhan *database* PostgreSQL untuk menyimpan data secara terstruktur dan mendukung relasi antar tabel.
- Kebutuhan pengelolaan data pengguna, data transaksi, serta data pendukung lainnya yang tersimpan dalam *database*.

- Kebutuhan *API endpoint* untuk mendukung operasi *Create, Read, Update, Delete* (CRUD) pada setiap entitas data.
- Kebutuhan mekanisme autentikasi dan otorisasi pengguna untuk membatasi akses terhadap fitur tertentu.

B Sistem Aplikasi RUinRISK

Selanjutnya, hasil analisis kebutuhan sistem *backend* dan *database* pada aplikasi RUinRISK meliputi:

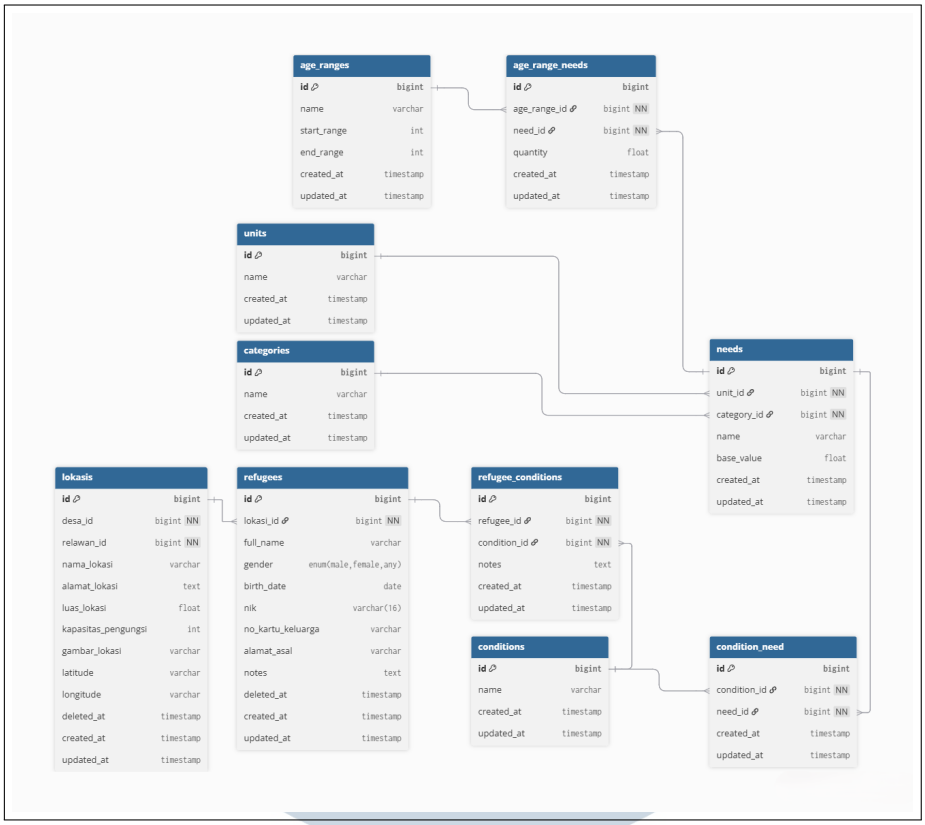
- Kebutuhan penggunaan *backend framework* untuk mengelola *business logic* aplikasi.
- Kebutuhan *database* untuk menyimpan dan mengelola data utama aplikasi secara terstruktur.
- Kebutuhan *API endpoint* sebagai penghubung antara *backend* dan *client-side* aplikasi.
- Kebutuhan perancangan struktur *database* yang mendukung relasi antar data.
- Kebutuhan validasi data untuk menjaga konsistensi dan integritas data.
- Kebutuhan sistem autentikasi untuk pengelolaan data *user* pada aplikasi.

3.3.3 Perancangan Sistem

A situs GMLS SYS

Sistem GMLS SYS dirancang menggunakan arsitektur *monolith* berbasis *framework* Laravel.

1. **Perancangan Basis Data Sistem GMLS SYS**
   Basis data diimplementasikan menggunakan PostgreSQL untuk menangani relasi kompleks antara data demografis pengungsi, manajemen lokasi, dan kalkulasi kebutuhan logistik. Struktur relasi antar entitas digambarkan dalam *Entity Relationship Diagram* (ERD) sebagaimana ditunjukkan pada Gambar 3.2.



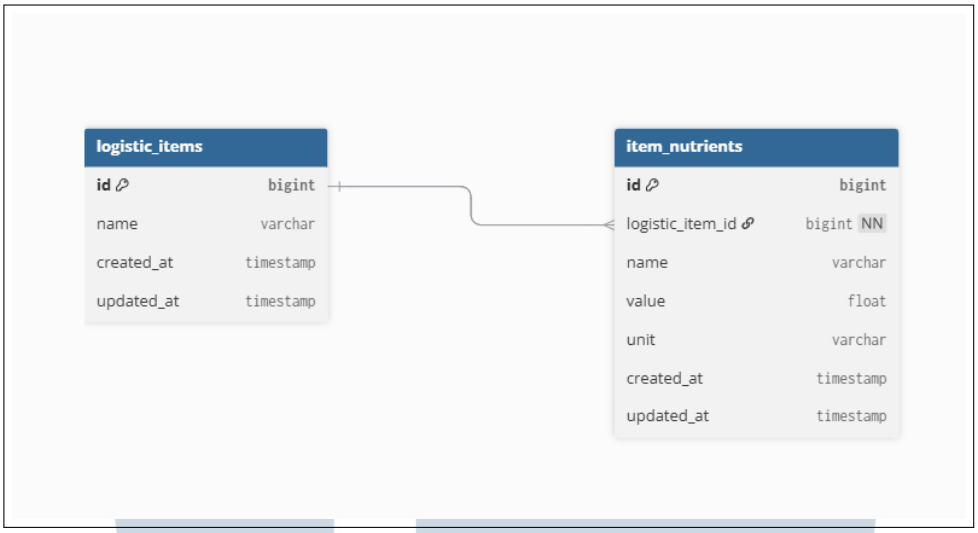
Gambar 3.2. *Entity Relationship Diagram* (ERD) 1 GMLS SYS.

Berdasarkan Gambar 3.2, berikut adalah deskripsi fungsional dari setiap entitas yang membentuk skema basis data:

- **Entitas *refugees*:** Berfungsi sebagai penyimpan data utama individu pengungsi. Atribut yang disimpan meliputi identitas demografis (Nama Lengkap, NIK, No. Kartu Keluarga), jenis kelamin, tanggal lahir, dan alamat asal. Data ini esensial untuk validasi penerima bantuan dan analisis statistik populasi terdampak.
- **Entitas *lokasis*:** Merepresentasikan titik-titik posko pengungsian atau tempat evakuasi. Entitas ini menyimpan atribut geografis (*latitude* dan *longitude*) untuk keperluan visualisasi pemetaan, serta atribut kapasitas (*kapasitas_pengungsi*) dan penanggung jawab lokasi (*relawan_id*).
- **Entitas *categories*:** Merupakan tabel referensi yang digunakan untuk mengklasifikasikan jenis kebutuhan logistik (misalnya: Pangan, Sandang, Obat-obatan). Entitas ini bertujuan untuk memudahkan pengelompokan dan manajemen inventaris pada entitas *needs*.

- **Entitas *units*:** Menyediakan standarisasi satuan ukur (misalnya: kg, liter, paket) yang berelasi dengan entitas *needs*. Penggunaan entitas ini memastikan konsistensi data kuantitas dalam perhitungan stok logistik.
- **Entitas *needs*:** Menyimpan katalog master kebutuhan logistik yang diperlukan di tempat evakuasi. Setiap *record* kebutuhan terhubung dengan satuan ukur (*unit_id*) dan kategori (*category_id*) untuk spesifikasi yang lebih detail.
- **Entitas *conditions*:** Mendefinisikan daftar kondisi khusus atau status kerentanan yang mungkin dialami oleh pengungsi (misalnya: Ibu Hamil, Sakit, Disabilitas) yang mempengaruhi jenis bantuan yang dibutuhkan.
- **Entitas *refugee_conditions*:** Merupakan entitas asosiatif (*pivot table*) yang menghubungkan data pengungsi (*refugees*) dengan kondisi yang dimilikinya (*conditions*). Relasi ini bersifat *many-to-many*, memungkinkan satu pengungsi memiliki lebih dari satu kondisi kesehatan yang tercatat.
- **Entitas *condition_need*:** Berfungsi memetakan kebutuhan logistik tambahan yang wajib dipenuhi berdasarkan kondisi tertentu. Entitas ini menghubungkan *conditions* dan *needs* dalam relasi *many-to-many*.
- **Entitas *age_ranges*:** Mendefinisikan klasifikasi kelompok usia (misalnya: Balita, Dewasa, Lansia) menggunakan atribut batas bawah (*start_range*) dan batas atas (*end_range*) untuk keperluan segmentasi distribusi bantuan.
- **Entitas *age_range_needs*:** Menghubungkan kelompok usia (*age_ranges*) dengan kebutuhan logistik (*needs*). Entitas ini memiliki atribut *quantity* yang menentukan standar jatah kebutuhan spesifik untuk setiap kelompok umur, memungkinkan sistem melakukan kalkulasi kebutuhan secara otomatis dan presisi.

Selain pemetaan kebutuhan dasar pengungsi, sistem GMLS SYS juga dirancang untuk memiliki modul khusus untuk manajemen inventaris yang lebih mendetail, khususnya terkait nilai gizi dari bantuan logistik. Struktur data untuk modul ini digambarkan pada Gambar 3.3.



Gambar 3.3. *Entity Relation Diagram* (ERD) 2 GML SYS

Modul ini dirancang untuk memberikan granularitas data yang lebih tinggi mengenai kandungan nutrisi dari setiap barang bantuan yang masuk. Penjelasan mengenai entitas-entitas yang terlibat adalah sebagai berikut:

- **Entitas *logistic_items*:** Berfungsi sebagai tabel induk untuk mendata jenis barang fisik atau produk logistik yang tersedia di gudang penyimpanan (misalnya: ”Beras 5kg”, ”Susu Bubuk Sachet”). Entitas ini menyimpan atribut *name* sebagai identitas utama barang, serta cap waktu (*timestamps*) untuk pelacakan data.
- **Entitas *item_nutrients*:** Merupakan entitas yang menyimpan rincian kandungan gizi yang terdapat dalam satu unit *logistic_item*. Entitas ini memiliki relasi *one-to-many* terhadap *logistic_items*, di mana satu jenis barang dapat memiliki berbagai komponen nutrisi (seperti Karbohidrat, Protein, Lemak).

  Tabel ini menyimpan atribut:

  – *name*: Nama nutrisi (misalnya: ”Kalori”, ”Protein”).
  – *value*: Nilai kuantitatif kandungan nutrisi.
  – *unit*: Satuan ukur nutrisi (misalnya: ”kkal”, ”mg”, ”gram”).

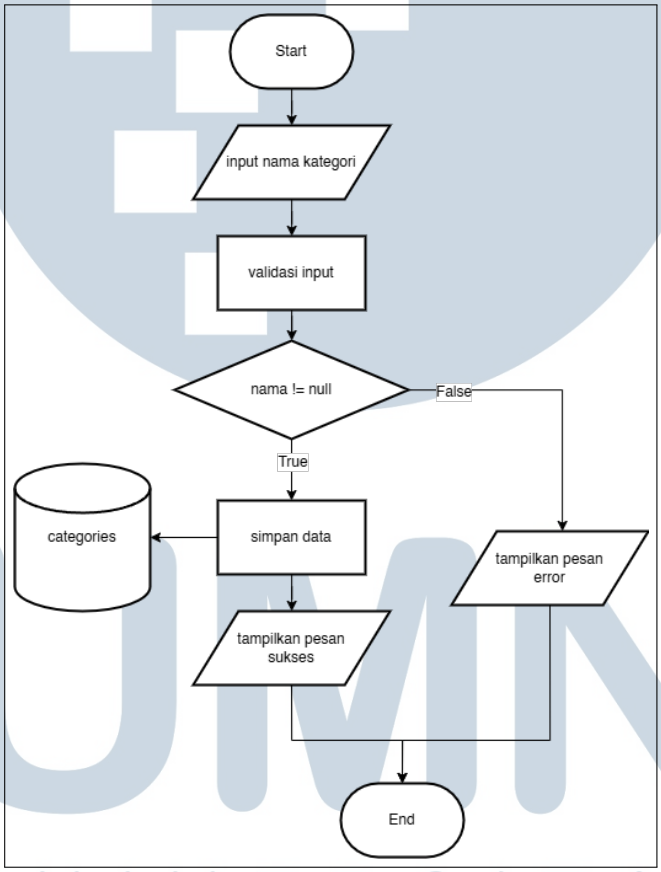
  Keberadaan entitas ini memungkinkan sistem untuk melakukan kalkulasi otomatis terkait total asupan gizi yang diterima oleh pengungsi berdasarkan barang yang didistribusikan.

2. **Perancangan Alur Logika dan Fitur *Back-End***

   Perancangan alur dan fitur ini disesuaikan dengan permintaan dari organisasi GMLS.

   Beberapa alur logikanya antara lain:

   - **Menambahkan Kategori Kebutuhan** Selain pengelolaan data kebutuhan spesifik, sistem memfasilitasi pengguna untuk mendefinisikan kategori logistik baru. Alur logika untuk fitur ini diilustrasikan pada Gambar 3.4.



Gambar 3.4. *Flowchart* proses penambahan kategori baru.

Berdasarkan diagram di atas, sistem memulai proses dengan menerima *input* nama kategori, kemudian melakukan validasi untuk memeriksa apakah data tersebut kosong atau tidak. Logika percabangan yang terjadi adalah sebagai berikut:

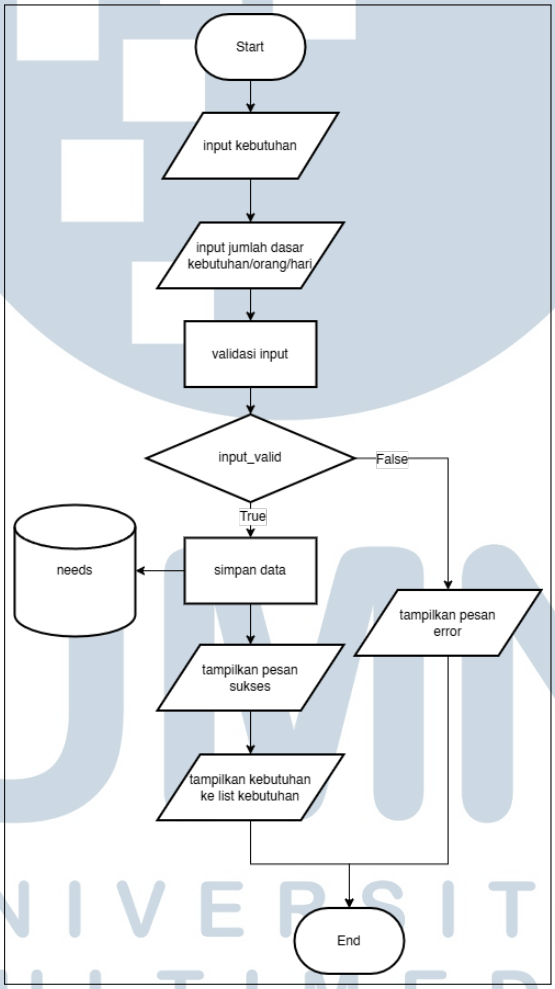
(a) **Jika kondisi terpenuhi (*True*):** Apabila *input* nama tidak bernilai *null*, sistem akan menyimpan data ke dalam tabel `categories` dan

menampilkan pesan sukses (*success message*) kepada pengguna.

(b) **Jika kondisi tidak terpenuhi (*False*):** Apabila *input* kosong, sistem tidak akan memproses penyimpanan data dan langsung menampilkan pesan kesalahan (*error message*) sebagai peringatan.

- **Menambahkan Data Kebutuhan**

  Fitur ini berfungsi untuk mendaftarkan jenis kebutuhan logistik baru beserta parameter standar distribusinya. Alur logika penambahan data ini digambarkan pada Gambar 3.5.



Gambar 3.5. *Flowchart* alur penambahan data kebutuhan logistik.

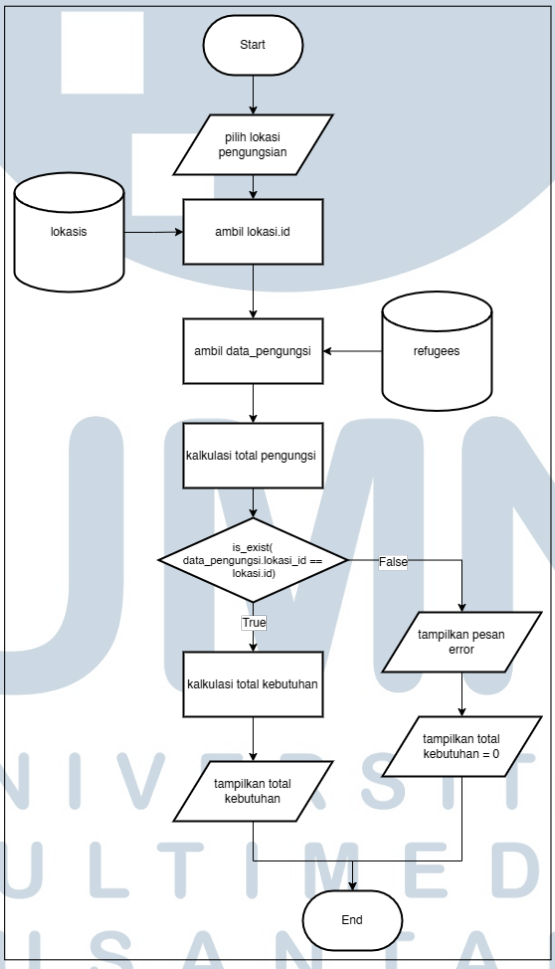
Berdasarkan diagram di atas, proses dimulai dengan pengguna memasukkan nama kebutuhan dan jumlah dasar kebutuhan per orang per hari (*base amount/person/day*). Sistem kemudian melakukan validasi terhadap *input* tersebut. Mekanisme percabangan yang terjadi adalah:

(a) **Jika *input* valid (*True*):** Sistem menyimpan data ke dalam tabel `needs` di basis data, menampilkan pesan sukses, dan secara otomatis memperbarui daftar kebutuhan (*list kebutuhan*) pada antarmuka pengguna.

(b) **Jika *input* tidak valid (*False*):** Sistem menolak penyimpanan data dan menampilkan pesan *error* sebagai peringatan agar pengguna memperbaiki data yang dimasukkan.

- **Kalkulasi Logistik Otomatis**

  Fitur unggulan sistem ini adalah kemampuan menghitung estimasi kebutuhan logistik secara dinamis per lokasi. Alur komputasi logistik ini diilustrasikan pada Gambar 3.6.



Gambar 3.6. *Flowchart* logika kalkulasi kebutuhan logistik.

Berdasarkan diagram alur di atas, proses dimulai ketika pengguna memilih lokasi pengungsian. Sistem kemudian mengambil `lokasi.id`

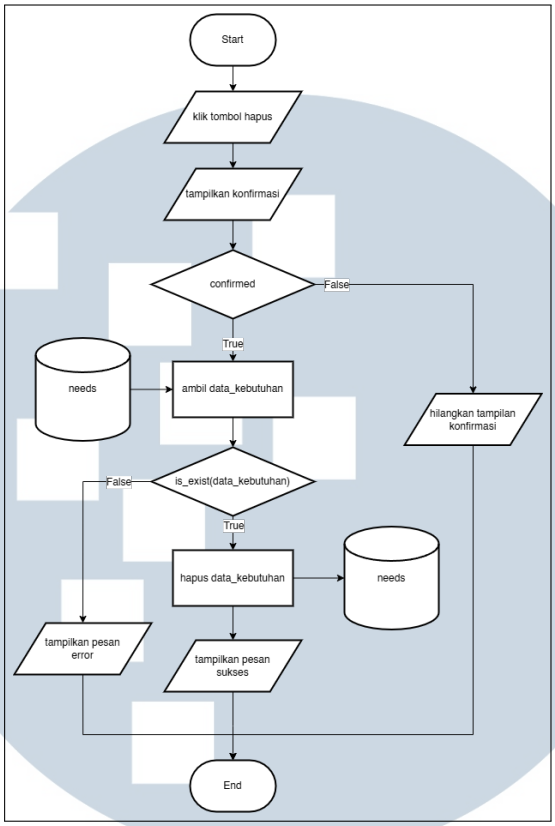
dari tabel `lokasis` untuk dijadikan kunci pencarian data pengungsi pada tabel `refugees`.

Setelah melakukan kalkulasi total pengungsi, sistem memvalidasi ketersediaan data dengan logika `is_exist`. Percabangan yang terjadi adalah sebagai berikut:

(a) **Kondisi *True*:** Jika data pengungsi ditemukan (`data_pengungsi.lokasi_id == lokasi.id`), sistem akan menghitung total kebutuhan logistik dan menampilkan hasilnya pada antarmuka.

(b) **Kondisi *False*:** Jika data tidak ditemukan, sistem akan menampilkan pesan *error* dan secara otomatis menetapkan nilai total kebutuhan menjadi 0 (*nol*).

- **Penghapusan Data Kebutuhan** Untuk menjaga relevansi data, sistem menyediakan fitur penghapusan kebutuhan logistik yang sudah tidak diperlukan. Mekanisme ini menerapkan validasi berlapis untuk mencegah penghapusan yang tidak disengaja, sebagaimana ditunjukkan pada Gambar 3.7.



Gambar 3.7. *Flowchart* mekanisme penghapusan data.

Berdasarkan diagram alur di atas, proses dimulai ketika pengguna menekan tombol hapus, yang kemudian memicu munculnya dialog konfirmasi. Selanjutnya, sistem menjalankan dua tahapan logika percabangan:

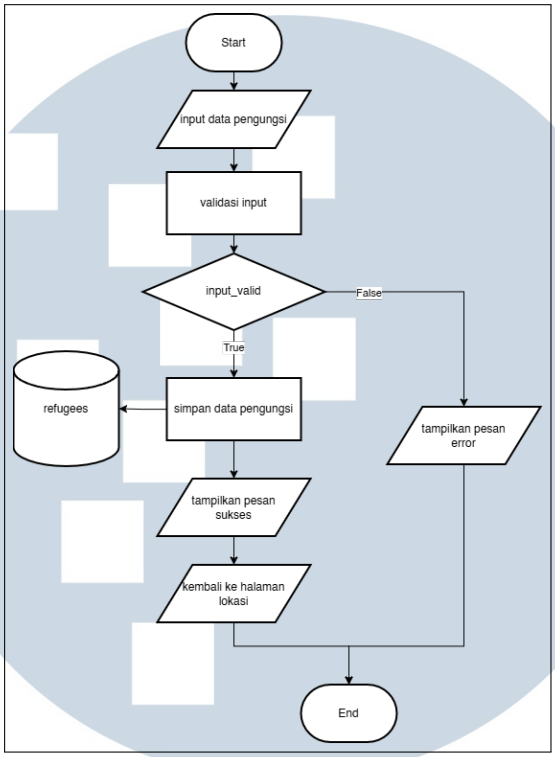
(a) **Konfirmasi Pengguna (*User Confirmation*):** Sistem menunggu respon pengguna. Jika pengguna membatalkan (*False*), sistem hanya akan menutup dialog konfirmasi. Jika pengguna menyetujui (*True*), sistem akan mengambil data terkait dari tabel `needs`.

(b) **Validasi Data (*Data Existence*):** Setelah konfirmasi, sistem memeriksa keberadaan data (`is_exist`). Jika data valid (*True*), sistem menghapus data tersebut dari tabel `needs` dan menampilkan pesan sukses. Namun, jika data tidak ditemukan (*False*), sistem akan menampilkan pesan *error*.

- **Menambahkan Data Pengungsi**

  Sistem menyediakan fitur pencatatan data pengungsi untuk memetakan populasi terdampak di setiap titik lokasi. Alur logika proses input data

ini digambarkan pada Gambar 3.8.



Gambar 3.8. *Flowchart* proses penambahan data pengungsi.

Berdasarkan diagram alur di atas, proses dimulai dengan pengisian formulir data pengungsi oleh pengguna. Sistem kemudian melakukan validasi terhadap *input* tersebut. Mekanisme percabangan yang terjadi adalah sebagai berikut:

(a) **Kondisi Valid (*True*):** Jika data memenuhi syarat, sistem akan menyimpan informasi tersebut ke dalam tabel `refugees` di basis data dan menampilkan pesan sukses. Setelah itu, sistem secara otomatis mengarahkan pengguna kembali ke halaman detail lokasi (*redirect to location page*).

(b) **Kondisi Tidak Valid (*False*):** Jika terdapat kesalahan pada *input*, sistem membatalkan penyimpanan dan menampilkan pesan *error* kepada pengguna.

B Aplikasi RUinRISK

Sistem *back-end* pada aplikasi RUinRISK menggunakan *tech-stack* Node.js dan Express.js sebagai kerangka kerja pengembangan layanan server, Prisma sebagai *Object Relational Mapping* (ORM) untuk pengelolaan skema database, serta PostgreSQL sebagai basis data utama.

1. **Perancangan Basis Data**
   Perancangan basis data dilaksanakan guna mengakomodasi kebutuhan penyimpanan dan pengelolaan informasi aplikasi secara terstruktur. Dalam implementasinya, penulis memanfaatkan Prisma sebagai *Object-Relational Mapping* (ORM) untuk efisiensi pemodelan skema, manajemen relasi antar-tabel, serta integrasi dengan sistem manajemen basis data PostgreSQL. Struktur data dirancang secara sistematis untuk mendukung fungsionalitas utama aplikasi RUinRISK, khususnya pada manajemen autentikasi dan pengelolaan profil pengguna.

2. **Perancangan Alur Logika *Back-End* Aplikasi**
   Pengembangan logika *back-end* difokuskan pada optimalisasi penanganan permintaan pengguna (*user request*) yang aman dan efisien. Lingkup utama modul ini mencakup implementasi mekanisme autentikasi dan registrasi pengguna, yang di dalamnya menerapkan validasi *input* yang ketat, enkripsi data sensitif, serta manajemen sesi (*session management*) untuk menjaga keamanan akses.

   - **Alur Autentikasi Pengguna (*Login*)**
     Mekanisme *login* dirancang sebagai gerbang keamanan utama untuk memverifikasi identitas pengguna sebelum memberikan hak akses. Alur logika autentikasi ini diilustrasikan pada Gambar 3.9.

Gambar 3.9. *Flowchart* alur *login* aplikasi RUinRISK.

Berdasarkan diagram alur di atas, proses autentikasi melibatkan beberapa tahapan validasi:

(a) **Validasi Format Input:** Sistem pertama-tama memeriksa format data yang dimasukkan (`input_valid`). Jika format tidak sesuai, sistem langsung menampilkan pesan *error* tanpa melakukan akses ke basis data.

(b) **Verifikasi Kredensial:** Jika format valid (*True*), sistem mencari data akun pada tabel `users` di basis data. Selanjutnya, dilakukan pengecekan logika ganda: apakah akun tersedia (`is_exist`) **dan** apakah kata sandi sesuai (`password_correct`).

(c) **Hasil Autentikasi:**

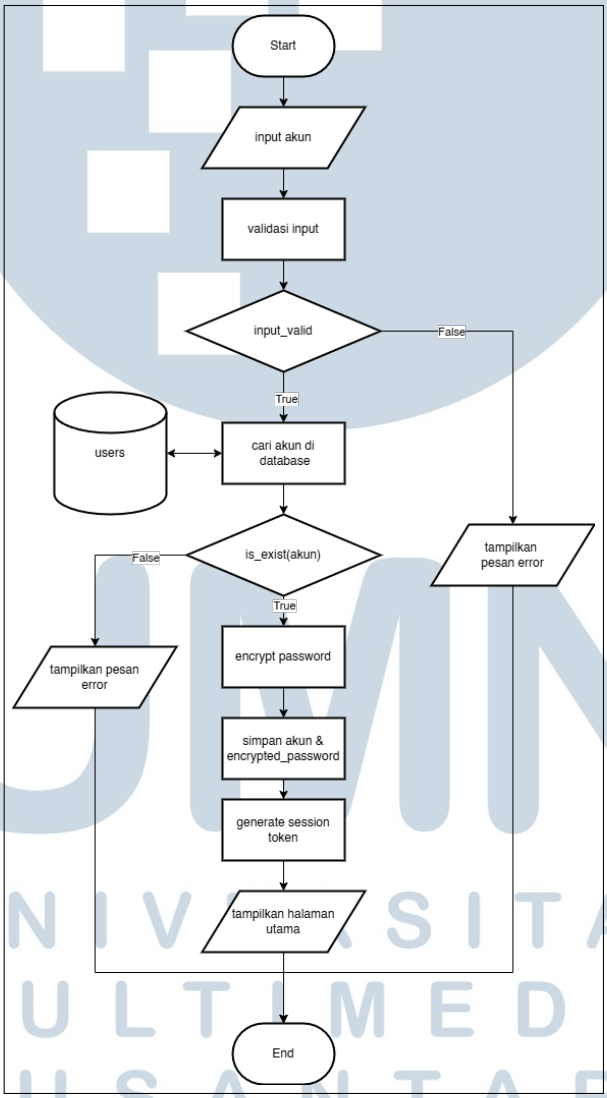
- Jika kedua syarat terpenuhi (*True*), sistem akan

membangkitkan *session token* sebagai tanda otorisasi dan mengarahkan pengguna ke halaman utama.

– Jika salah satu syarat tidak terpenuhi (*False*), sistem akan menolak akses dan menampilkan pesan kesalahan.

- **Alur Pendaftaran Pengguna (*Register*)**

  Selain autentikasi, sistem menyediakan fitur registrasi untuk mendaftarkan pengguna baru. Alur kerja teknis pendaftaran akun ini digambarkan pada Gambar 3.10.



Gambar 3.10. *Flowchart* alur *register* aplikasi RUinRISK.

Berdasarkan diagram tersebut, proses pendaftaran melibatkan serangkaian prosedur keamanan data:

(a) **Validasi Awal:** Proses dimulai dengan input data akun. Sistem memvalidasi kelengkapan format input. Jika tidak valid (*False*), sistem langsung menampilkan pesan *error*.

(b) **Verifikasi Basis Data:** Jika input valid (*True*), sistem melakukan pencarian dan verifikasi status akun pada tabel `users`.

(c) **Enkripsi dan Penyimpanan:** Setelah verifikasi akun terpenuhi (*True*), sistem melakukan enkripsi kata sandi (*encrypt password*) untuk keamanan, kemudian menyimpan kredensial baru tersebut ke dalam basis data.

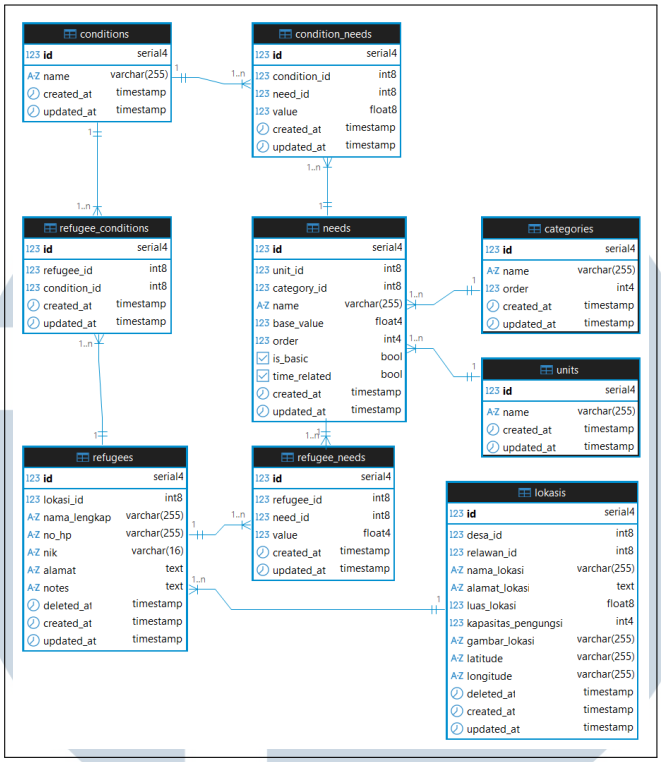
(d) **Otorisasi Otomatis:** Setelah penyimpanan berhasil, sistem langsung men-*generate session token* dan mengarahkan pengguna ke halaman utama, sehingga pengguna tidak perlu melakukan *login* ulang.

3.3.4 Implementasi

Pada tahap implementasi ini semua rancangan yang telah dibuat sebelumnya, direalisasikan menjadi basis data dan *source code*.

A Situs GMLS SYS

1. **Implementasi Basis Data**
   Implementasi struktur basis data dibangun menggunakan fitur *Migration* pada framework Laravel. Gambar 3.11 menampilkan *Entity Relationship Diagram* (ERD) dari skema yang terbentuk.



Gambar 3.11. Skema relasi tabel (*ERD*) hasil implementasi.

Berikut adalah penjabaran *source code* migrasi untuk entitas-entitas utama:

- **Tabel Master Logistik**

  Kelompok tabel ini berfungsi sebagai referensi utama sistem yang mendefinisikan standar satuan, kategori, dan jenis kebutuhan logistik, sebagaimana ditunjukkan pada Kode Program 3.1.

```
        // Tabel: categories
        Schema::create('categories', function (Blueprint
    $table) {
            $table->id();
            $table->string('name');
            $table->integer('order')->default(0);
            $table->timestamps();
        });

        // Tabel: units
        Schema::create('units', function (Blueprint
    $table) {
            $table->id();
            $table->string('name');
```

```
            $table->timestamps();
        });

        // Tabel: needs
        Schema::create('needs', function (Blueprint
    $table) {
            $table->id();
            $table->foreignId('unit_id')->constrained()->
    cascadeOnDelete();
            $table->foreignId('category_id')->constrained
    ()->cascadeOnDelete();
            $table->string('name');
            $table->float('base_value', 8, 2)->default(0)
    ;
            $table->boolean('is_basic')->default(false);
            $table->boolean('time_related')->default(true
    );
            $table->timestamps();
        });

```

Kode 3.1: Migrasi Tabel Master Logistik

- **Tabel Data Wilayah dan Pengungsi**
  Tabel ini menyimpan data administratif posko pengungsian (`lokasis`) serta biodata lengkap para pengungsi (`refugees`). Implementasi skema tabel ini dapat dilihat pada Kode Program 3.2.

```
        // Tabel: lokasis
        Schema::create('lokasis', function (Blueprint
    $table) {
            $table->id();
            $table->foreignId('desa_id')->constrained()->
    cascadeOnDelete();
            $table->foreignId('relawan_id')->constrained
    ()->cascadeOnDelete();
            $table->string('nama_lokasi');
            $table->text('alamat_lokasi');
            $table->float('luas_lokasi');
            $table->integer('kapasitas_pengungsi');
            $table->string('latitude');
            $table->string('longitude');
            $table->timestamps();
        });
```

```
14
15          // Tabel: refugees
16          Schema::create('refugees', function (Blueprint
    $table) {
17              $table->id();
18              $table->foreignId('lokasi_id')->constrained('
    lokasis')->cascadeOnDelete();
19              $table->string('nama_lengkap');
20              $table->string('nik', 16)->nullable();
21              $table->text('alamat')->nullable();
22              $table->text('notes')->nullable();
23              $table->softDeletes();
24              $table->timestamps();
25          });
26
```

Kode 3.2: Migrasi Tabel Lokasi dan Pengungsi

- **Tabel Logika Kondisi**

  Bagian ini menangani logika pemetaan antara kondisi khusus (seperti kerentanan tertentu) dengan standar kebutuhan spesifik yang menyertainya. Struktur tabel untuk logika ini terdapat pada Kode Program 3.3.

```
1           // Tabel: conditions
2           Schema::create('conditions', function (Blueprint
    $table) {
3               $table->id();
4               $table->string('name');
5               $table->timestamps();
6           });
7
8           // Tabel: condition_needs
9           Schema::create('condition_needs', function (
    Blueprint $table) {
10              $table->id();
11              $table->foreignId('condition_id')->
    constrained()->cascadeOnDelete();
12              $table->foreignId('need_id')->constrained()->
    cascadeOnDelete();
13              $table->float('value')->default(0);
14              $table->timestamps();
15          });
```

```
16
```

Kode 3.3: Migrasi Tabel Logika Kondisi

- **Tabel Relasi Personal**

  Tabel ini berfungsi sebagai pencatat hubungan *many-to-many* antara data pengungsi dengan kondisi yang dialami. Implementasi relasi tersebut dapat dilihat pada Kode Program 3.4.

```
1         // Tabel: refugee_conditions
2         Schema::create('refugee_conditions', function (
    Blueprint $table) {
3             $table->id();
4             $table->foreignId('refugee_id')->constrained
    ()->cascadeOnDelete();
5             $table->foreignId('condition_id')->
    constrained()->cascadeOnDelete();
6             $table->timestamps();
7         });
8
9         // Tabel: refugee_needs
10        Schema::create('refugee_needs', function (
    Blueprint $table) {
11            $table->id();
12            $table->foreignId('refugee_id')->constrained
    ()->cascadeOnDelete();
13            $table->foreignId('need_id')->constrained()->
    cascadeOnDelete();
14            $table->float('value', 8, 2)->nullable();
15            $table->timestamps();
16        });
17
```

Kode 3.4: Migrasi Tabel Relasi Pengungsi

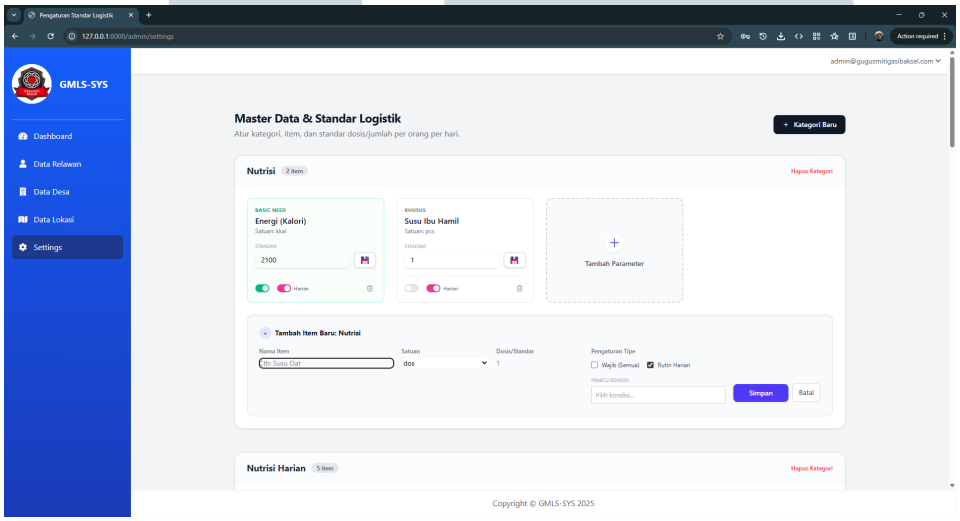
2. **Implementasi Alur Logika Sistem Web (GMLS-SYS)**

   Pengembangan logika *back-end* pada sistem GMLS-SYS difokuskan pada pengelolaan data logistik dan algoritma kalkulasi kebutuhan pengungsi. Berikut adalah rincian implementasi logika sistem beserta antarmuka yang dihasilkan:

   - **Manajemen Master Data & Standar Logistik**

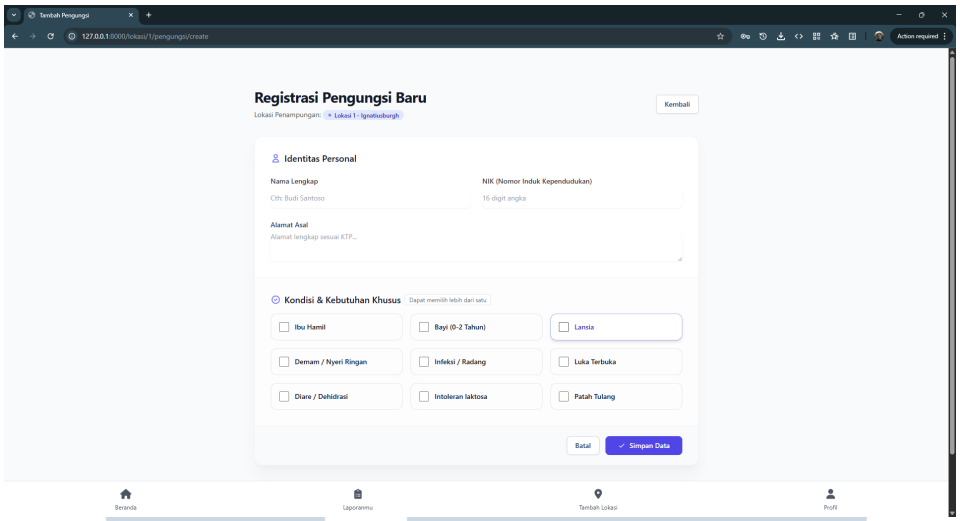
     Sistem menyediakan antarmuka bagi admin untuk mendefinisikan

standar logistik, mulai dari kategori hingga item spesifik. Secara logika, sistem menerapkan validasi ketat pada saat *input* data. Ketika admin menambahkan item baru, sistem mewajibkan pengisian parameter jumlah dasar kebutuhan per orang per hari. Jika validasi terpenuhi, data disimpan ke tabel `needs` dan sistem secara otomatis mengaktifkan perhitungan logistik untuk item tersebut. Tampilan antarmuka pengaturan standar logistik dapat dilihat pada Gambar 3.12.



Gambar 3.12. Implementasi antarmuka *input* standar logistik.

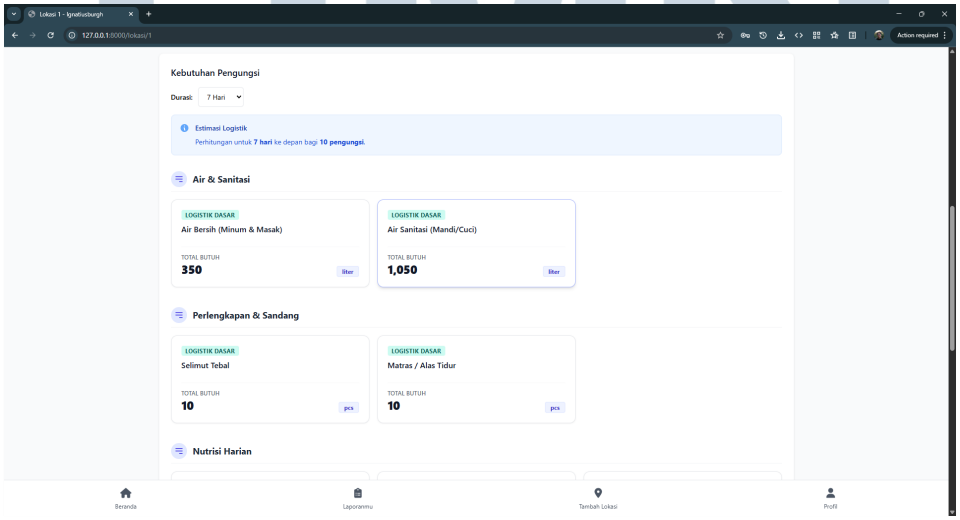
- **Manajemen & Registrasi Data Pengungsi**
  Pencatatan data pengungsi dilakukan melalui formulir digital untuk memetakan populasi terdampak. Logika *back-end* memproses data yang dimasukkan melalui tahap validasi integritas data. Setelah data dinyatakan valid dan berhasil disimpan ke tabel `refugees`, sistem menerapkan mekanisme *redirect* otomatis yang mengembalikan pengguna ke halaman detail lokasi untuk melihat pembaruan data secara *real-time*. Formulir registrasi pengungsi ditampilkan pada Gambar 3.13.



Gambar 3.13. Antarmuka formulir registrasi pengungsi baru.

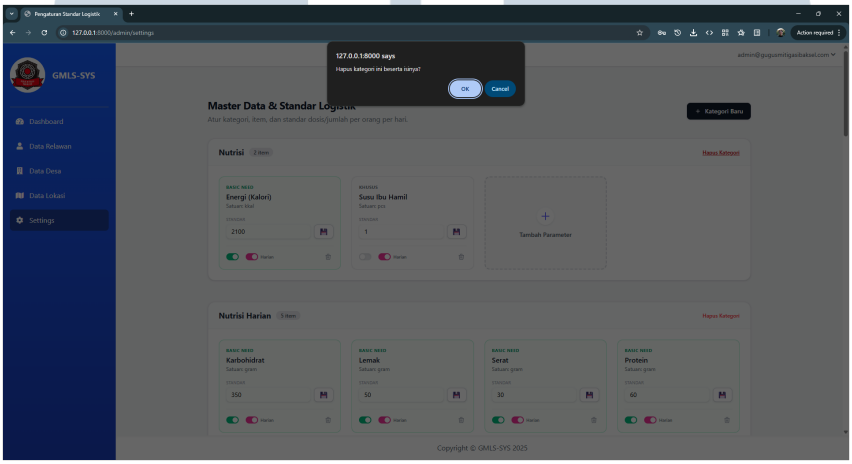
- **Algoritma Kalkulasi Logistik Otomatis**

  Fitur utama sistem adalah kalkulasi otomatis total kebutuhan logistik. Algoritma bekerja dengan cara mengambil ID lokasi yang dipilih, menghitung total populasi pengungsi yang terdaftar pada lokasi tersebut, kemudian mengalikannya dengan nilai dasar kebutuhan (*base value*) yang telah diatur di master data. Hasil kalkulasi ini disajikan secara dinamis pada antarmuka pengguna untuk membantu relawan menyiapkan stok logistik yang presisi. Hasil implementasi fitur ini terlihat pada Gambar 3.14.



Gambar 3.14. Tampilan hasil estimasi kebutuhan logistik per lokasi.

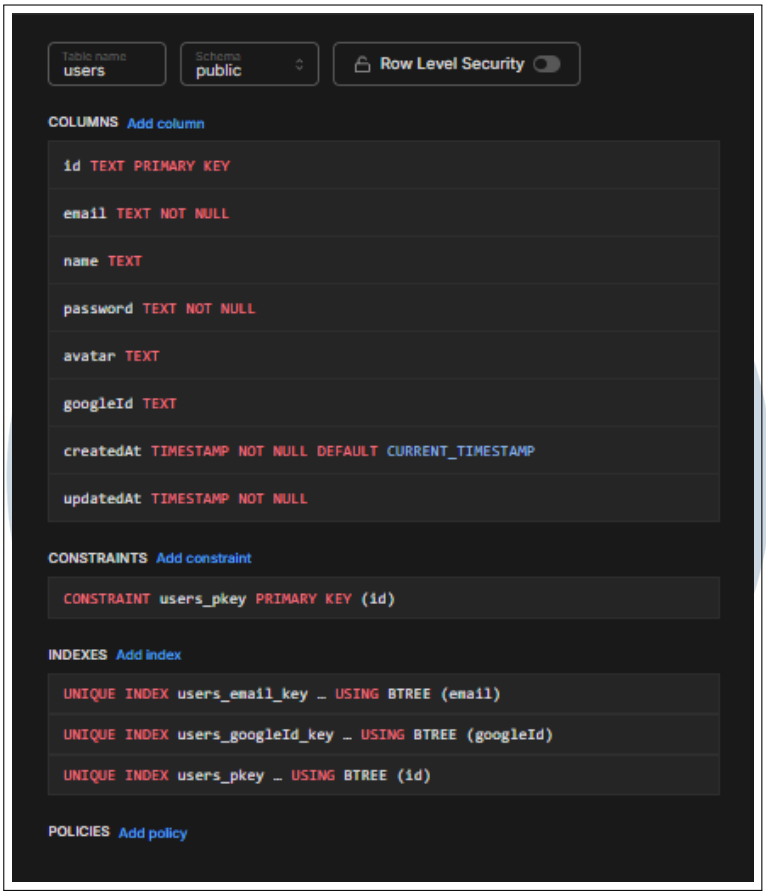
- **Mekanisme Validasi Penghapusan Data**
  Untuk menjaga keamanan data dari kesalahan operasi, sistem menerapkan logika validasi bertingkat pada fitur penghapusan. Sistem tidak langsung menghapus data saat tombol ditekan, melainkan memunculkan dialog konfirmasi terlebih dahulu. Penghapusan permanen dari basis data hanya dieksekusi setelah pengguna memberikan konfirmasi eksplisit (*confirmed*). Implementasi dialog konfirmasi ini dapat dilihat pada Gambar 3.15.



Gambar 3.15. Implementasi modal konfirmasi untuk keamanan penghapusan.

B Aplikasi RUinRISK

1. **Implementasi Basis Data (Aplikasi RUinRISK)**
   Pada pengembangan aplikasi *mobile* RUinRISK, manajemen basis data dilakukan menggunakan *Object-Relational Mapping* (ORM) Prisma dengan basis data PostgreSQL yang di-*hosting* pada layanan NeonDB. Gambar 3.16 menampilkan struktur tabel yang telah terimplementasi di server.



Gambar 3.16. Struktur tabel `users` pada dashboard NeonDB.

Kode definisi skema (*Prisma Schema*) yang digunakan untuk men-*generate* tabel tersebut dapat dilihat pada Kode Program 3.5. Tabel ini dirancang untuk menangani otentikasi pengguna, termasuk dukungan untuk *login* konvensional (*email/password*) dan OAuth (Google ID).

```
model User {
    // Primary Key menggunakan UUID
    id         String   @id @default(uuid())

    // Data akun utama
    email      String   @unique
    name       String?
    password   String
    avatar     String?

    // Dukungan OAuth (Optional)
    googleId   String? @unique

```

```
    // Audit trails
    createdAt DateTime @default(now())
    updatedAt DateTime @updatedAt

    // Mapping ke nama tabel di database
    @@map("users")
}
```

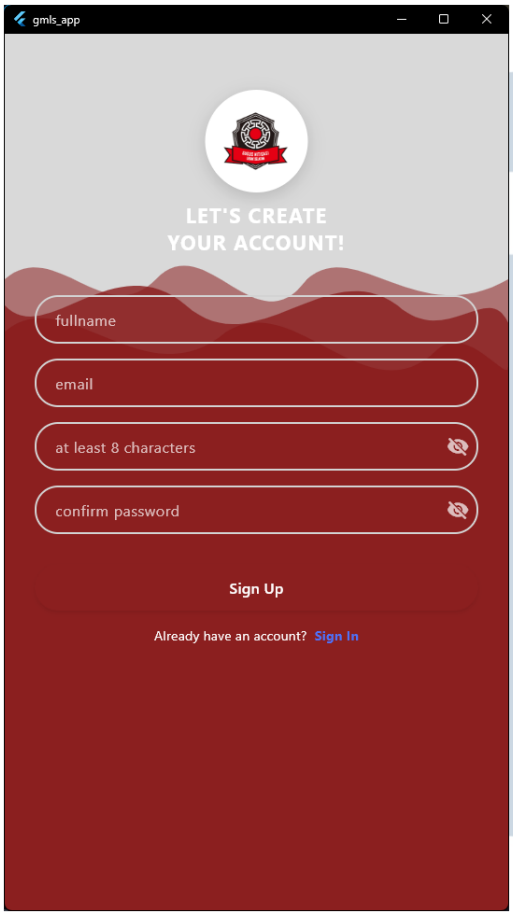
Kode 3.5: Definisi Model User (Prisma Schema)

2. **Implementasi *Back-End***

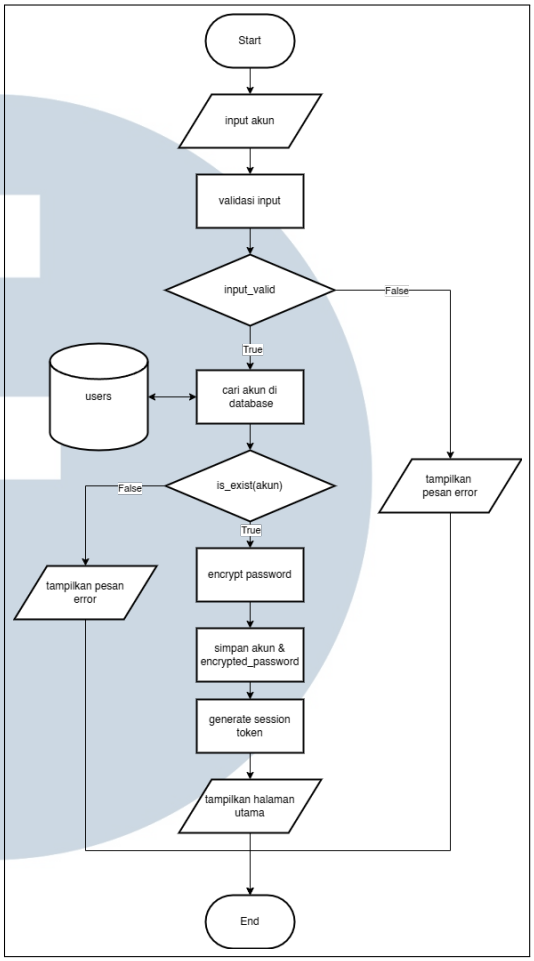
   Implementasi sisi server (*server-side*) dibangun menggunakan lingkungan Node.js dengan framework Express.js. Logika bisnis utama mencakup manajemen otentikasi pengguna dan keamanan data menggunakan enkripsi.

   - **Implementasi Registrasi Akun**

     Proses pendaftaran pengguna diawali dengan pengisian identitas pada antarmuka aplikasi sebagaimana terlihat pada Gambar 3.17. Data yang dimasukkan kemudian diproses oleh sistem melalui tahapan validasi input, enkripsi kata sandi menggunakan algoritma *hashing* (Bcrypt), dan penyimpanan ke basis data. Alur logika teknis ini digambarkan pada Gambar 3.18.



Gambar 3.17. Antarmuka halaman registrasi akun pengguna.



Gambar 3.18. *Flowchart* alur registrasi pengguna.

Implementasi teknis dari alur tersebut dapat dilihat pada Kode Program 3.6. Pada kode ini, sistem melakukan pengecekan duplikasi *email* terlebih dahulu sebelum melakukan *hashing password* dan penyimpanan data.

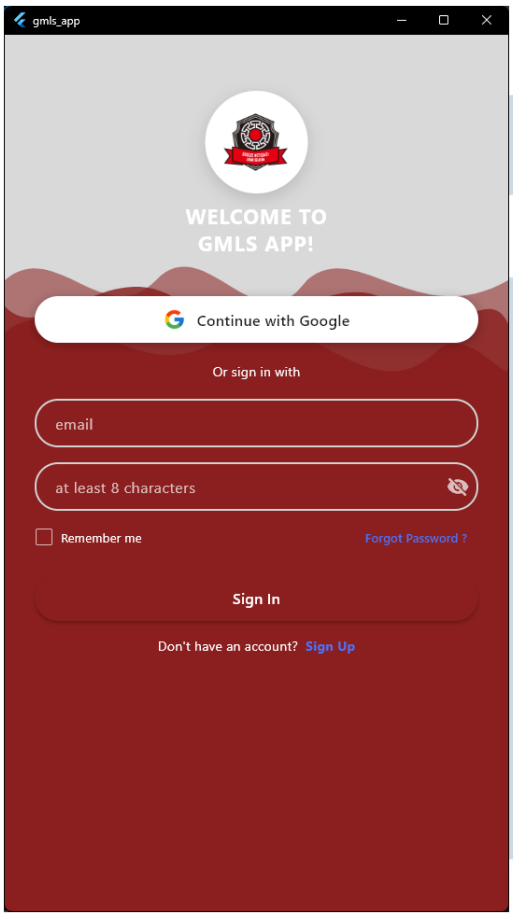
```
        register: async (req, res) => {
            // Cek apakah user sudah ada
            const userExists = await prisma.user.
    findUnique({ where: { email } });
            if (userExists) {
                return res.status(400).json({ message: "
    User already exists..." });
            }

            // Enkripsi Password
```

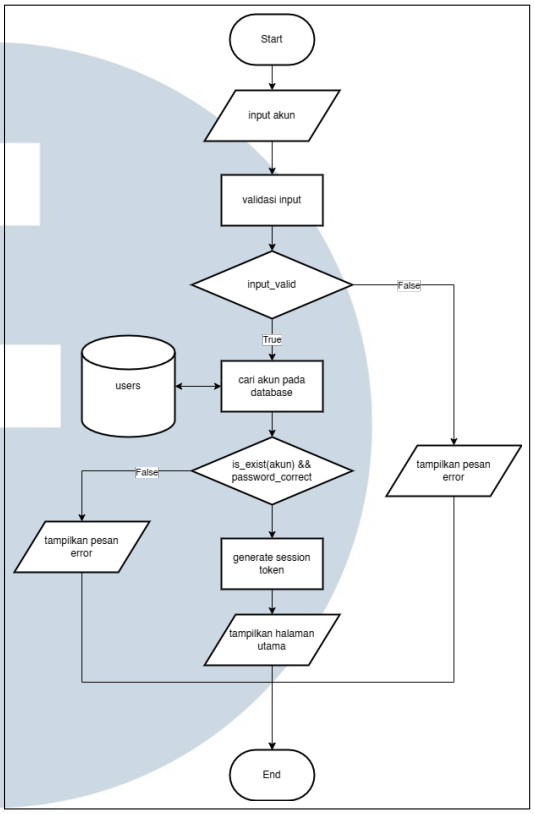
```
9             const salt = await bcrypt.genSalt(10);
10            const hashedPassword = await bcrypt.hash(
    password, salt);
11
12            // Simpan ke Database via Prisma
13            const user = await prisma.user.create({
14                data: { name, email, password:
    hashedPassword },
15            });
16
17            // Generate Token
18            const token = generateToken(user.id);
19            // ... return response json
20        },
21
```

Kode 3.6: Logika Registrasi (authController.js)

- **Implementasi Login Konvensional**
  Sistem autentikasi memverifikasi identitas pengguna melalui formulir *login* yang ditampilkan pada Gambar 3.19. Sistem membandingkan *plain-text password* yang dikirimkan klien dengan *hashed password* yang tersimpan di *database*. Jika validasi berhasil, sistem akan menerbitkan *JSON Web Token* (JWT) sebagai sesi akses. Alur logika ini diilustrasikan pada Gambar 3.20.



Gambar 3.19. Tampilan antarmuka halaman *login*.



Gambar 3.20. *Flowchart* alur *login* pengguna.

Implementasi kode untuk verifikasi kata sandi dan pembuatan token sesi ditunjukkan pada Kode Program 3.7. Kode tersebut menunjukkan penggunaan fungsi `bcrypt.compare` untuk mencocokkan kredensial secara aman.

```
1       login: async (req, res) => {
2           // Cari user berdasarkan email
3           const user = await prisma.user.findUnique({
    where: { email } });
4
5           // Verifikasi Password menggunakan Bcrypt
6           const isPasswordValid = await bcrypt.compare(
    password, user.password);
7
8           if (!isPasswordValid) {
9               return res.status(401).json({ message: "
    Invalid email or password" });
```

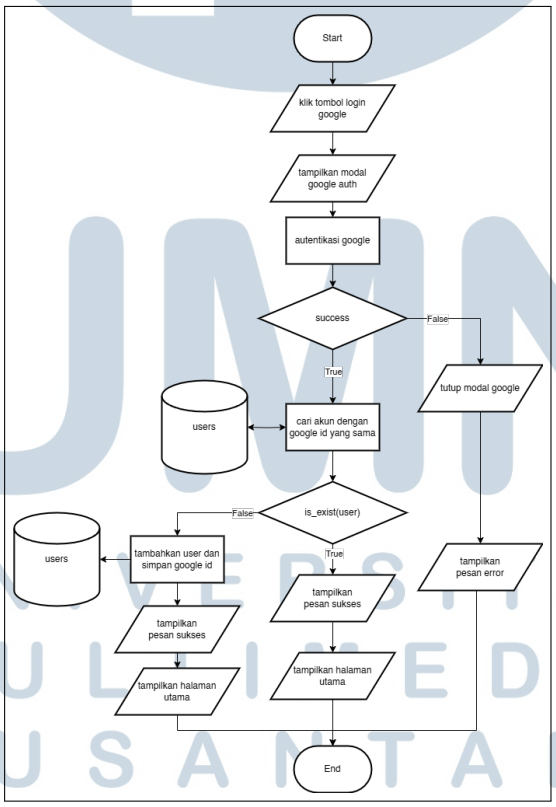
```
            }

            // Generate Session Token
            const token = generateToken(user.id);

            res.status(200).json({ status: "success", token, ... });
        },
```

Kode 3.7: Logika Login (authController.js)

- **Implementasi Logika Google OAuth (*Backend*)**
  Selain metode konvensional, dikembangkan pula logika autentikasi menggunakan standar OAuth 2.0 melalui layanan Google. Fitur ini dirancang untuk mempermudah akses pengguna tanpa perlu mengisi formulir registrasi manual. Alur logika pemrosesan data autentikasi Google ini digambarkan pada Gambar 3.21.



Gambar 3.21. *Flowchart* alur logika autentikasi Google OAuth.

Implementasi di sisi *back-end* memanfaatkan pustaka

`passport-google-oauth20`. Sebagaimana ditunjukkan pada Kode Program 3.8, sistem menerapkan strategi *Find or Create*. Artinya, sistem akan menerima profil dari Google, kemudian mengecek apakah `googleId` pengguna sudah terdaftar. Jika sudah ada, pengguna langsung masuk; jika belum, sistem akan membuatkan akun baru secara otomatis menggunakan data dari Google.

```
    passport.use(
        new GoogleStrategy({
            clientID: process.env.GOOGLE_CLIENT_ID,
            clientSecret: process.env.GOOGLE_CLIENT_SECRET,
            callbackURL: "http://localhost:5005/auth/google/callback",
        },
        async (accessToken, refreshToken, profile, done) => {
            try {
                // Logika Find or Create: Cari user berdasarkan Google ID,
                // jika tidak ada, buat user baru menggunakan data profil Google.
                const [user] = await User.findOrCreate({
                    where: { googleId: profile.id },
                    defaults: {
                        name: profile.displayName,
                        email: profile.emails?.[0]?.value,
                    },
                });
                return done(null, user);
            } catch (error) { return done(error, null); }
        })
    );

```

Kode 3.8: Strategi Google OAuth (googleOAuth.js)

3.4 Kendala dan Solusi yang Ditemukan

3.4.1 Kendala Pengembangan

Selama proses pelaksanaan proyek *Humanity Project*, tim pengembang menghadapi sejumlah hambatan teknis maupun manajerial yang memengaruhi efektivitas alur kerja *back-end*. Kendala-kendala tersebut diidentifikasi sebagai berikut:

- **Ambiguitas Spesifikasi Kebutuhan Sistem**
  Pada tahap inisiasi, cakupan spesifikasi sistem masih bersifat general dan belum terdefinisi secara granular. Hal ini menyulitkan tim dalam menentukan prioritas fitur utama yang harus dikembangkan terlebih dahulu.

- **Kompleksitas Arsitektur *Back-End* Awal**
  Struktur kode *back-end* warisan (*legacy code*) dirancang dengan tingkat kompleksitas yang tinggi tanpa pola desain yang konsisten. Kondisi ini memperlambat proses pengembangan fitur baru serta mempersulit pemeliharaan (*maintenance*) ketika terjadi *bug*.

- **Keterbatasan Alokasi Sumber Daya Manusia**
  Terbatasnya jumlah personil dalam tim menyebabkan beban kerja tidak terdistribusi secara ideal. Akibatnya, satu pengembang sering kali harus menangani peran ganda (*multitasking*) yang berpotensi menurunkan fokus dan kualitas kode.

- **Restriksi Durasi Pengembangan**
  Waktu pelaksanaan magang yang terbatas membuat tidak seluruh rencana fitur dapat diselesaikan dalam satu siklus pengembangan, sehingga berisiko meninggalkan fitur yang belum matang (*half-baked features*).

- **Ketiadaan Dokumentasi Teknis Legasi**
  Khusus pada pengembangan situs GMLS, tidak ditemukan dokumentasi teknis yang memadai dari pengembang periode sebelumnya. Hal ini menghambat proses *reverse engineering* untuk memahami logika bisnis dan struktur sistem yang sudah berjalan.

3.4.2 Solusi dan Tindak Lanjut

Guna memitigasi kendala di atas dan memastikan keberlanjutan proyek, diterapkan sejumlah strategi penyelesaian yang dipetakan sebagai berikut:

- **Analisis Ulang dan Prioritisasi Kebutuhan**
  (Solusi untuk Ambiguitas Spesifikasi)
  Dilakukan peninjauan kembali (*requirement review*) bersama pemangku kepentingan untuk mempertegas ruang lingkup sistem. Pengembangan kemudian difokuskan pada fitur-fitur esensial (*Minimum Viable Product*) yang memiliki urgensi tinggi.

- **Refaktorisasi dan Penyederhanaan Struktur**
  (Solusi untuk Kompleksitas Arsitektur)
  Arsitektur *back-end* disusun ulang (*refactoring*) menggunakan pendekatan yang lebih modular dan terorganisir. Tujuannya adalah untuk menurunkan utang teknis (*technical debt*) sehingga sistem lebih mudah dikembangkan di masa depan.

- **Optimalisasi Manajemen Tugas**
  (Solusi untuk Keterbatasan SDM)
  Menerapkan manajemen tugas yang ketat menggunakan skala prioritas. Pembagian beban kerja difokuskan pada keahlian spesifik masing-masing anggota untuk menjaga efisiensi, meskipun dengan sumber daya terbatas.

- **Perencanaan Keberlanjutan Sistem (*Handover Plan*)**
  (Solusi untuk Restriksi Waktu)
  Mengantisipasi keterbatasan waktu, pengembangan dirancang dengan prinsip keberlanjutan. Fitur yang belum selesai dicatat dalam *backlog* pengembangan untuk dilanjutkan oleh tim pengembang periode berikutnya.

- **Penyusunan Dokumentasi Teknis Komprehensif**
  (Solusi untuk Ketiadaan Dokumentasi)
  Penulis menyusun dokumentasi teknis yang lengkap, mencakup skema basis data, alur API, dan petunjuk instalasi. Dokumentasi ini bertujuan menjadi acuan standar (*blueprint*) bagi pengembang selanjutnya agar kendala analisis sistem tidak terulang kembali.