BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kegiatan magang di PT Braincode Digital Teknologi, kegiatan kerja magang ditempatkan pada posisi sebagai *Software Backend Engineer*. Dalam menjalankan peran tersebut, kegiatan magang bekerja secara langsung di bawah arahan Technical Lead yang juga berperan sebagai pembimbing lapangan selama kegiatan magang berlangsung. Posisi ini berada dalam struktur divisi Backend yang berfokus pada pengembangan sisi server aplikasi.

Berikut merupakan rekan-rekan yang turut terlibat dalam kegiatan magang pada periode yang sama:

a) **Nama:** Muhammad Tristan Ajibrilyan Nandipinto
Asal Kampus: Universitas Multimedia Nusantara
Jurusan: Informatika
Angkatan: 2022

b) **Nama:** Reinhard Javera Maheswara
Asal Kampus: Universitas Multimedia Nusantara
Jurusan: Informatika
Angkatan: 2022

Salah satu proyek utama yang dikerjakan selama kegiatan magang di PT Braincode Digital Teknologi adalah Relogica, yaitu sebuah aplikasi manajemen aset berbasis web yang dirancang untuk membantu perusahaan dalam proses inventarisasi, pemantauan kondisi aset, analisis potensi permasalahan, serta penentuan langkah penanganan atau *countermeasures*. Aplikasi ini dikembangkan untuk meningkatkan efisiensi pengelolaan aset melalui pendekatan digital yang terintegrasi dan berbasis data.

Dalam proyek tersebut, mahasiswa magang tergabung dalam tim pengembangan sisi *backend* yang bertanggung jawab atas pengembangan dan pemeliharaan API, pengelolaan basis data, serta pelaksanaan proses *deployment* layanan ke server produksi. Proses pengembangan dilakukan menggunakan bahasa pemrograman Rust dengan dukungan sistem basis data PostgreSQL untuk kebutuhan operasional dan analitik.

Tim pengembang aplikasi Relogica terdiri dari beberapa anggota dengan pembagian peran yang jelas. Pengembangan sisi *frontend* dikerjakan oleh Ade Esarrendy Intris, sedangkan pengembangan sisi *backend* ditangani oleh Muhammad Ukasah Hayata, S.Si., dan Muhammad Tristan Ajibrilyan Nandipinto.

3.2 Tugas yang Dilakukan

Selama menjalani kegiatan magang di PT Braincode Digital Teknologi, kegiatan kerja magang dilaksanakan dalam berbagai tugas yang berkaitan dengan pengembangan dan pemeliharaan sistem backend pada proyek Relogica. Tugas-tugas tersebut mencakup perancangan basis data, pengembangan REST API menggunakan bahasa pemrograman Rust, serta pelaksanaan proses deployment layanan ke server. Adapun uraian tugas yang dilaksanakan adalah sebagai berikut:

a) **Perancangan Basis Data:** Melakukan perancangan struktur tabel serta membangun relasi antar tabel pada sistem basis data PostgreSQL[5]. Proses ini didukung oleh penggunaan aplikasi DBeaver sebagai alat bantu visualisasi dan pengelolaan data. Perancangan skema dilakukan dengan mempertimbangkan efisiensi penyimpanan, integritas data, dan performa query, khususnya untuk kebutuhan pengelolaan aset perusahaan.

b) **Pengembangan API:** Melakukan pengembangan REST API menggunakan bahasa pemrograman Rust dengan dukungan manajemen paket melalui *Cargo*[6]. API dirancang untuk menangani berbagai proses pada sistem manajemen aset, seperti autentikasi pengguna, pengelolaan data aset, serta pengaturan hak akses pengguna berdasarkan peran (*role*). Selain itu, pengujian endpoint dilakukan menggunakan Postman[7] untuk memastikan keandalan layanan dan konsistensi pertukaran data antar sistem.

c) **Deployment Layanan:** Melaksanakan proses deployment layanan backend melalui terminal berbasis Linux agar API dapat diakses oleh tim frontend. Proses ini mencakup konfigurasi server, pengelolaan *environment variable*, serta pemantauan performa layanan pada server pengujian maupun server produksi.

Proyek utama yang dikerjakan selama kegiatan magang adalah pengembangan sistem *Relogica Asset Management System* (Relogica AMS), yaitu sebuah platform manajemen aset berbasis web yang dirancang untuk

mendukung proses inventarisasi, pemantauan, dan pengendalian aset secara terpusat.

Relogica AMS bertujuan untuk menyediakan sistem yang mampu memfasilitasi pencatatan dan pengelolaan aset secara digital serta mendukung proses pengambilan keputusan melalui penyajian data yang terintegrasi. Sistem ini juga dilengkapi dengan fitur untuk mendeteksi potensi permasalahan pada aset dan memberikan rekomendasi tindakan penanggulangan atau *countermeasures*.

Dalam pengembangan sistem tersebut, kegiatan kerja magang sebagai *Backend Developer* yang bertanggung jawab dalam proses pembuatan, pengujian, serta pemeliharaan API yang menghubungkan sistem frontend dengan basis data. Backend dikembangkan menggunakan bahasa pemrograman Rust yang dikenal memiliki performa tinggi dan tingkat keamanan memori yang baik, sehingga sesuai untuk diterapkan pada aplikasi berskala besar seperti sistem manajemen aset.

Relogica AMS memiliki dua jenis peran pengguna utama, yaitu **Admin** dan **User Biasa**, yang masing-masing memiliki hak akses berbeda:

a) **Admin:** Peran ini diperuntukkan bagi pengguna dengan tanggung jawab manajerial, seperti supervisor atau tim IT internal. Admin memiliki kewenangan untuk mengelola data aset secara menyeluruh, melakukan manajemen pengguna (*CRUD user*), serta memantau kondisi aset dan menganalisis langkah penanggulangan (*countermeasure*) terhadap aset yang bermasalah. Admin memiliki seluruh hak akses, termasuk `can_delete`, `can_insert`, `can_edit`, dan `can_upload`.

b) **User Biasa:** Peran ini ditujukan bagi karyawan umum yang memiliki hak akses terbatas sesuai dengan izin (*permission*) yang diberikan oleh admin. Pengguna pada peran ini hanya dapat melakukan pengelolaan data aset sesuai dengan hak akses yang dimiliki, seperti menambah atau mengedit data tertentu, tanpa memiliki izin untuk menghapus atau mengunggah dokumen aset. Pengaturan hak akses dilakukan secara dinamis oleh admin guna menjaga keamanan dan integritas data sistem.

Pengembangan sistem Relogica AMS dilakukan secara kolaboratif antara tim backend dan frontend dengan pendekatan kerja berbasis *sprint*[8]. Seluruh proses pengembangan dikelola menggunakan GitLab[9] sebagai media manajemen repositori dan pembagian tugas, sedangkan komunikasi antar anggota tim dilakukan melalui platform internal perusahaan. Proses deployment dilakukan ke server

internal dan server publik untuk keperluan pengujian sebelum sistem dirilis ke lingkungan produksi.

3.3 Uraian Pelaksanaan Magang

Selama pelaksanaan magang, tugas-tugas yang telah dikerjakan dirangkum pada Tabel 3.1 berikut.

Tabel 3.1. Pekerjaan yang dilakukan tiap minggu selama pelaksanaan kerja magang

Minggu Ke -	Pekerjaan yang dilakukan
1	a) Melakukan pembaruan masa berlaku token menjadi satu minggu dan memastikan validasi sesi pengguna. b) Memperbaiki sistem perizinan berbasis token untuk mencegah kesalahan “no permission”. c) Memeriksa logika penghapusan item rekomendasi pada backend API. d) Menyelaraskan proses unggah data *AMS Analysis Pack* dengan tampilan FMdb agar kolom *Task Type* konsisten. e) Memperbaiki fungsi *Add/Delete Recommendation* serta memastikan sinkronisasi database setelah operasi berhasil. f) Menelusuri dan memperbaiki logika pemrosesan *Asset Hierarchy* terutama pada fitur *drag & drop*. g) Mengubah seluruh notifikasi menjadi bahasa Inggris dan memastikan data rekomendasi muncul sesuai hierarki induk. h) Memperbaiki proses unggah hierarki aset agar data berhasil masuk ke database tanpa error.

Minggu Ke -	Pekerjaan yang dilakukan
2	a) Membuat layanan penyimpanan konfigurasi tabel (*Save Table Configuration*) untuk modul *Standard Task* dan *FM Database*. b) Menambahkan validasi unggah file pada *AMS Analysis Pack* agar hanya menerima *asset_id* terdaftar. c) Menambahkan logika error handling dan pesan umpan balik ketika proses unggah gagal. d) Memperbaiki pemetaan kolom *Task Type* pada file unggahan *AMS Analysis Pack*. e) Membuat layanan penyimpanan pengaturan level hierarki aset per pengguna. f) Membatasi hak akses unggah data agar hanya dapat dilakukan oleh pengguna dengan peran Admin. g) Memperbaiki tampilan halaman detail *AMS Analysis Pack* agar seluruh data dapat dimuat dengan benar.
3	a) Menambahkan kolom baru pada tabel *task* seperti *Useful Life*, *PF Interval*, *Failure Finding Interval*, dan *EACH UOM*. b) Menyesuaikan API `PUT /fm` agar mendukung pembaruan kolom tambahan tersebut. c) Memperbaiki urutan data (*order by*) pada konfigurasi tabel AMS agar tampil ascending. d) Memperbaiki bug pada halaman *table configuration edit* (AMS Pack) untuk metode `ADD` dan `PUT`.

Minggu Ke -	Pekerjaan yang dilakukan
4	a) Memperbaiki bug pada konfigurasi tabel AMS Pack untuk metode `PUT`. b) Mendesain tabel baru untuk menyimpan kolom komentar dan membuat API untuk menambah serta mengambil komentar berdasarkan data terkait. c) Memperbaiki bug pada fitur *Edit FM* yang belum mengimplementasikan kolom komentar. d) Memperbaiki proses unggah hierarki aset yang salah menampilkan pesan sukses padahal gagal tersimpan. e) Menghapus data AMS yang terhubung dengan FM Data untuk pembersihan database.
5	a) Menghapus data AMS yang mencakup tabel *rcmd_task* dan *task*. b) Memperbaiki bug duplikasi pada kolom *Frequency*, *Preparation*, dan *Tools* di unggahan *Standard Task*. c) Menyesuaikan API *Task Delivered List* pada proyek LMS agar menghitung keterlambatan dengan menambahkan satu hari pada *Due Date*. d) Menyesuaikan API *Attendance List* agar sesuai spesifikasi, termasuk penghapusan offset waktu dan penambahan tanggal. e) Membuat tabel *bc_core_rebalancing_simulation_recalculate* dan API untuk pengelolaan data *rebalancing simulation*.

Minggu Ke -	Pekerjaan yang dilakukan
6	a) Menyesuaikan respons JSON untuk endpoint `/statistics/project_enrolled`. b) Membuat API untuk mengedit dan menghapus data *rebalancing simulation*. c) Memperbaiki payload dan menambahkan kolom baru seperti *incidentid* pada tabel dan API terkait.
7	a) Menyesuaikan API penyimpanan dan pengeditan *rebalancing simulation* untuk mendukung kolom baru *incidenceId*. b) Menambahkan kolom *incidenceId* dalam hasil query berbagai endpoint seperti `/json-api-result-info-rebalancing-simulation` dan `/view-detail-rebalancing-simulation`. c) Memastikan fungsi `ch_query::Update/Delete` berjalan pada cluster ClickHouse.
8	a) Melakukan pengujian model deteksi keselamatan kerja (*workers-best.pt*) untuk mendeteksi pekerja dengan/ tanpa helm dan rompi. b) Mengonfigurasi CUDA agar proses pelatihan model menggunakan GPU. c) Mengumpulkan dataset tambahan melalui pengambilan foto dan proses pelabelan menggunakan *Labeling Lab*. d) Mengekspor dan melatih model menggunakan YOLOv5 serta menguji performa hasil pelatihan.

Minggu Ke -	Pekerjaan yang dilakukan
9	a) Membuat beberapa API untuk proyek Telkommetra ULO, yaitu: • `GET /dashboard/trx_success_rate_card` • `POST /dashboard/trx_success_rate_trend` • `POST /dashboard/trx_success_rate_summary` • `GET /dashboard/failed_section_distribution` • `POST /dashboard/failed_transaction_list` b) Melakukan debugging terhadap hitungan *success_count* dan *failed_count*.
10	a) Memperbaiki logika *filter_search* pada API `/dashboard/trx_monitoring_list`. b) Membuat API ekspor data ke Excel untuk beberapa layanan, antara lain: • `/dashboard/trx_monitoring_list` • `/dashboard/failed_transaction_list` • `/dashboard/trx_success_rate_summary`
11	a) Menyesuaikan pemetaan *origin* dan *destination* pada API `/dashboard/trx_monitoring_popup_log_table`. b) Menambahkan kolom *status* ke dalam respons JSON. c) Mengubah metode beberapa API dari `GET` menjadi `POST` serta menambahkan dukungan filter *start_date* dan *end_date* di berbagai layanan analitik.

Minggu Ke -	Pekerjaan yang dilakukan
12	a) fix bug API `trx_montioring_list_export` where table staging is not included`/dashboard/trx_monitoring_popup_log_table`. b) fix bug API `trx_success_rate_summary_export` where `date_range` doesn't work c) fix API `failed_section_distribution` where mapping "Telkomsel" not included
13	a) fix API `trx_sucess_rate_summary` where percentage still shows 100% when data success and failed data is 000000/0 b) fix API `trx_sucess_rate_card` where percentage still shows 100% when data success and failed data is 000000/0
14	a) adjust API POST `/dashboard/trx_monitoring_popup_log_table`: tambah field path dan adjust kolom origin dan destination `/dashboard/trx_monitoring_popup_log_table`. b) mencoba reduksi query time API trx_monitoring_list dengan menambahkan index di kolom yang terkena

Minggu Ke -	Pekerjaan yang dilakukan
15	a) Melanjutkan optimasi query `trx_monitoring_list` dengan me-mappingkan table `revenue_share` agar mengurangi beban query dari db langsung (50 sec -¿ 22 sec) b) Melanjutkan optimasi query `trx_monitoring_list` dengan me-mappingkan query sesuai dengan filter search (platform, package, status; masing-masing diberi querynya sendiri-sendiri) (22 sec -¿ 11 sec / 6 sec)
16	a) Melanjutkan optimasi query `trx_monitoring_list` dengan menambahkan select mapping table `dashboard_portal_staging`, `dashboard_portal`, `history_hit_api` b) Melanjutkan optimasi query `trx_monitoring_list` dengan membuat function untuk mengambil select query mapping dari table `dashboard_portal_staging` dan `dashboard_portal`

Selama masa magang di PT Braincode Digital Teknologi, kegiatan difokuskan pada pengembangan *backend service* berbagai proyek seperti Relogica, LMS Braincode, SafeVision, dan Telkommetra ULO. Tugas utama mencakup perancangan tabel database, pembuatan dan pengujian API, integrasi dengan sistem eksternal.

3.3.1 Infrastruktur Backend API

Perancangan *backend API* pada sistem *Relogica Asset Management System (AMS)* menggunakan bahasa pemrograman *Rust* dengan manajer paket *Cargo* untuk pengembangan layanan berbasis REST API. Sistem ini dirancang untuk menyediakan layanan pengelolaan data aset yang efisien, aman, dan mudah diintegrasikan dengan antarmuka pengguna berbasis web.

Seluruh proses pengembangan dilakukan menggunakan *Visual Studio Code*

sebagai *Integrated Development Environment (IDE)*, dengan *PostgreSQL* sebagai basis data utama. Pengujian endpoint dilakukan melalui *Postman*, sedangkan pengelolaan dan visualisasi struktur basis data dilakukan menggunakan *DBeaver*.

Berikut merupakan spesifikasi dari infrastruktur dan *framework* yang digunakan dalam perancangan *backend* sistem *Relogica AMS*:

a) *Cargo* versi 1.88.0

b) *rustc* versi 1.88.0

c) *Visual Studio Code* versi 1.105.1

d) *PostgreSQL* versi 17

e) *DBeaver* versi 24.3.4.202502021521

f) *Postman* versi 11.64.4

g) *OS Linux* versi 9.4 (Seafoam Ocelot)

Arsitektur *backend* dibangun dengan pendekatan *modular service*, di mana setiap endpoint dikelola melalui *controller* terpisah berdasarkan fungsionalitasnya, seperti pengelolaan aset, pengguna, dan log aktivitas.

Fungsi *handler* kemudian menjalankan logika bisnis seperti autentikasi pengguna, validasi data, serta interaksi dengan basis data melalui *library SQLx*. Setiap operasi basis data dilakukan secara asinkron untuk meningkatkan performa sistem. Setelah logika selesai diproses, *handler* mengembalikan respons dalam format *JSON* kepada klien.

Dengan arsitektur ini, *backend API* sistem *Relogica AMS* dapat dengan mudah diintegrasikan dengan *frontend* berbasis web maupun aplikasi *mobile*, serta mendukung skalabilitas dan keamanan data dalam lingkungan produksi perusahaan.

3.3.2 Autentikasi dan Manajemen Sistem

Pada tahap pengembangan sistem autentikasi dan manajemen token dalam proyek Relogica, dilakukan beberapa perbaikan penting untuk memastikan keamanan sekaligus kestabilan proses login dan validasi sesi pengguna. Salah satu langkah utama adalah memperpanjang masa hidup token autentikasi (JWT) menjadi 1 minggu, agar pengguna tidak perlu terlalu sering melakukan login ulang selama periode penggunaan yang aktif.

Selain itu, logika verifikasi token melalui middleware diperkuat agar proses validasi dapat mendeteksi token secara akurat dan menghindari kasus false negative, di mana pengguna dengan hak akses valid justru terblokir akibat perbedaan kecil pada format atau casing role. Dengan demikian, sistem autentikasi menjadi lebih andal dan toleran terhadap variasi data yang tidak kritis.

Pseudocode Sign In 3.1.

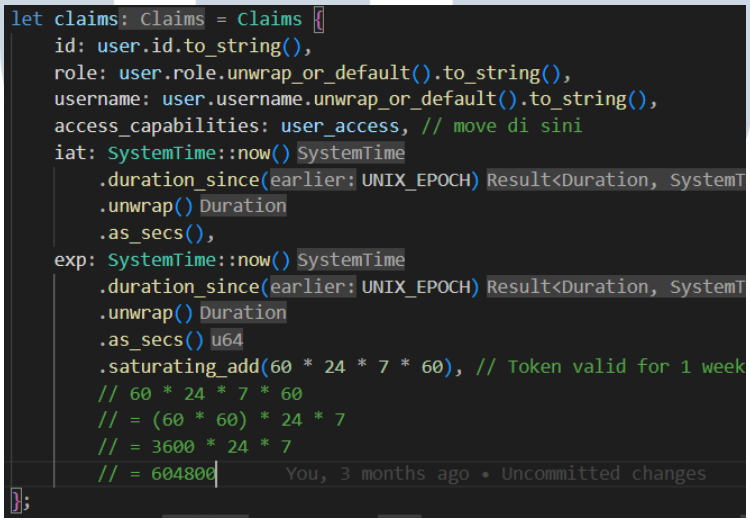
```
function signin(request):

    # 1. Baca body JSON
    body = parse_json(request.body)
    username = body.username or ""
    password_input = body.pass

    # 2. Query database: Ambil user berdasarkan username
    user = DB.query("
        SELECT id, username, pass_hash, role, company_id
        FROM user
        WHERE username = ?", username)

    if user NOT FOUND:
        return error("Username Not Found")

    # 3. Ambil hash password
    stored_hash = user.pass_hash

    if stored_hash is NULL:
        return error("Password Not Found")

    # 4. Verifikasi password
    if verify_hash(password_input, stored_hash) == false:
        return error("Wrong Password")

    # 5. Ambil akses user
    access_list = DB.query("
        SELECT
            user.id, user.username, user.role, user.company_id,
            access.can_delete, access.can_edit, access.can_insert,
    access.can_upload, access.module, company_name
        FROM user
        JOIN user_access_capabilities access ON user.id = access.
   id_user
```

```
        JOIN company ON user.company_id = company.id
        WHERE user.username = ?", username
    )

    # 6. Ambil data untuk response
    role = access_list[0].role
    company_id = access_list[0].company_id
    company_name = access_list[0].company_name
    username_resp = access_list[0].username

    # 7. Buat JWT Claims
    claims = {
        id: user.id,
        role: user.role,
        username: user.username,
        access_capabilities: access_list,
        iat: current_timestamp(),
        exp: current_timestamp() + 1 week
    }

    # 8. Generate JWT token
    token = jwt_encode(claims, SECRET_KEY)

    # 9. Buat response sukses
    response_data = {
        status: "Ok",
        info: "Login Successfully",
        role: role,
        company: company_id,
        company_name: company_name,
        username: username_resp
    }

    # 10. Set cookie "token"
    cookie = create_cookie(
        name = "token",
        value = token,
        http_only = false,
        same_site = Strict,
        expires = now + 7 days
    )

    response = create_response(200)
```

```
78      response.set_cookie(cookie)
79      response.set_json_body(response_data)
80
81      return response
```

Kode 3.1: Pseudocode Sign In

Secara teknis, penambahan masa aktif token diimplementasikan pada saat pembuatan *claims JWT*. Nilai exp (*expiration*) dihitung berdasarkan waktu saat ini (*SystemTime::now()*) yang dikonversi menjadi Unix *timestamp*, lalu ditambahkan dengan durasi satu minggu menggunakan metode *saturating_add*.



Gambar 3.1. Token Expiration Period

Pada Gambar 3.1. Perhitungan waktu 1 minggu dilakukan melalui ekspresi 60 * 24 * 7 * 60, yang jika diuraikan berarti:

a) 60 detik per menit × 60 menit per jam = 3600 detik per jam.

b) 3600 detik × 24 jam per hari = 86.400 detik per hari.

c) 86.400 × 7 hari = 604.800 detik total (durasi satu minggu).

Fungsi *saturating_add* digunakan untuk menambahkan nilai waktu ini ke *timestamp* saat ini dengan cara yang aman, yaitu mencegah terjadinya overflow pada tipe data numerik yang digunakan. Dengan pendekatan ini, sistem dapat menjaga stabilitas perhitungan waktu dan memastikan token akan kadaluarsa tepat setelah satu minggu tanpa risiko kesalahan perhitungan.

Melalui penerapan perbaikan ini, mekanisme autentikasi dalam Relogica kini berfungsi lebih konsisten dan tahan terhadap kesalahan minor pada sisi *backend* maupun *frontend*, sekaligus meningkatkan kenyamanan pengguna dalam sesi penggunaan aplikasi.

Pengujian performa dan kualitas *service* diuji menggunakan *Whitebox Testing* dan *Blackbox Testing* dan diuji oleh dua personel *Backend*.

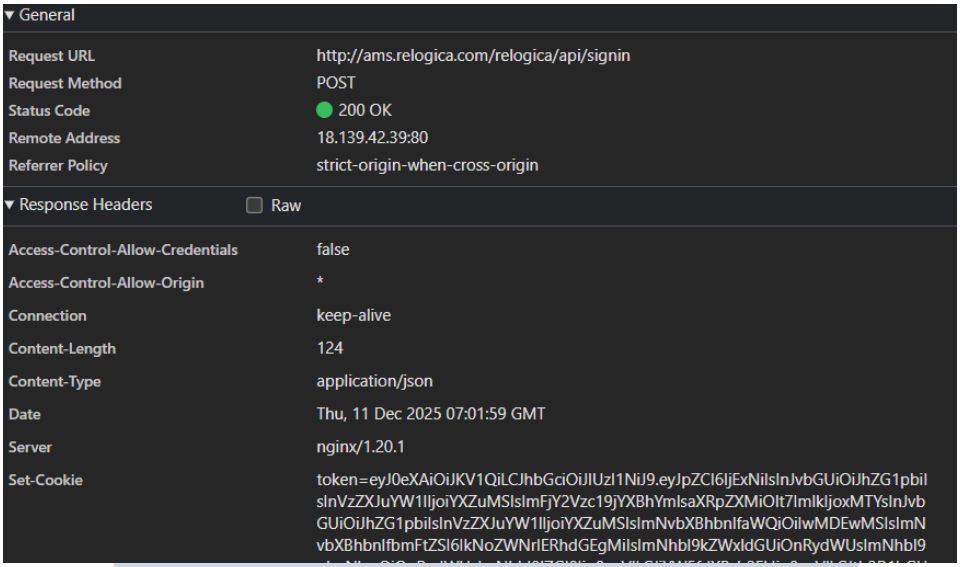
A Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada seluruh endpoint autentikasi, dilakukan pengujian dengan berbagai skenario input (data valid, data tidak valid, token kedaluwarsa, token salah, dan sebagainya) dan memeriksa apakah response yang dihasilkan sesuai ekspektasi.

Tabel 3.2. Tabel Pengujian API Autentikasi (`signin.rs`, `auth.rs`) - *Blackbox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /signin`	Login gagal dengan data tidak valid, password salah, username tidak terdaftar	Lulus
2	`POST /signin`	Login gagal dengan body request tidak lengkap (missing field)	Lulus
3	`GET /signout`	Logout, cek cookie token terhapus	Lulus
4	Protected endpoint	Request dengan token valid, expired, invalid, dan tanpa token	Lulus
5	Protected endpoint	Request ke `/signin` dan `/forgotpass` tanpa token tetap diterima	Lulus
6	Protected endpoint	Token dengan role berbeda, cek akses dan response sesuai hak akses	Lulus

Pada Gambar 3.2. token berhasil di-*generate* oleh JWT yang menyimpan expiration date dan created_at yang digunakan untuk mengetahui waktu kedaluwarsa token login user, beserta berfungsi untuk validasi kebenaran token.



Gambar 3.2. Service successfully generated JWT - Token Expiration Period

B Whitebox Testing

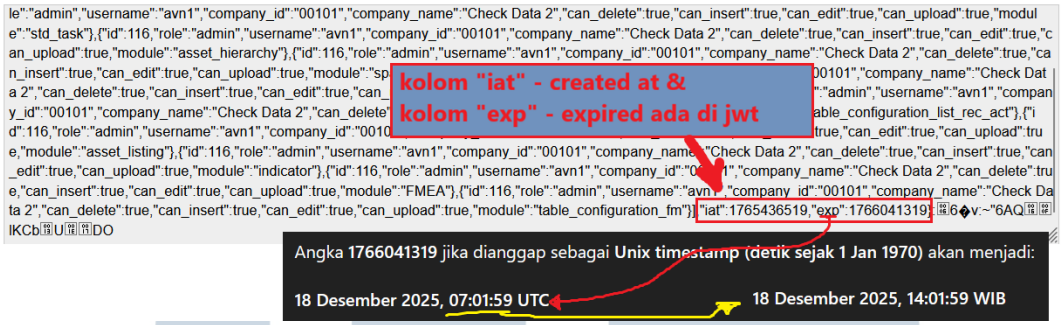
Pengujian dilakukan dengan mengetahui struktur internal kode, logika autentikasi, proses enkripsi/dekripsi, validasi token, serta integrasi dengan database. Pada API seperti `/signin`, `/signout`, dan middleware validasi token, pengujian whitebox dilakukan untuk memastikan seluruh proses autentikasi, validasi, dan keamanan berjalan sesuai logika yang diharapkan.

Tabel 3.3. Tabel Pengujian API Autentikasi (`signin.rs`, `auth.rs`) - *Whitebox*

No	Endpoint/Fungsi	Skenario Pengujian	Status
1	`POST /signin`	Validasi username dan password, verifikasi hash bcrypt, pembuatan JWT dengan expiry 1 minggu, set cookie token	Lulus
2	`POST /signin`	Penanganan password salah, username tidak ditemukan, dan user tanpa password	Lulus
3	`GET /signout`	Penghapusan cookie token, response logout sukses	Lulus
4	Middleware `check_user`	Decode JWT, validasi expiry, penanganan error token (expired, invalid, signature), inject claims ke request	Lulus
5	Middleware `check_user`	Bypass path untuk `/signin` dan `/forgotpass` tanpa validasi token	Lulus
6	Middleware `check_user`	Penanganan claims dan role, serta *saturating_add* pada expiry untuk mencegah overflow	Lulus

Pada Gambar 3.3. hasil *decoding* JWT pada field ”iat” dan ”exp”

menunjukkan bahwa token JWT yang dihasilkan valid dan mengandung durasi kedaluwarsa token login user.



Gambar 3.3. Decoded Base64 JWT - Token Expiration Period

3.3.3 Access Control & Role Management

Pada tahap ini, pengembangan sistem difokuskan pada penerapan dan penyempurnaan fitur *Access Control & Role Management* untuk menjaga keamanan dan konsistensi data pada aplikasi Relogica. Sistem ini bertujuan untuk memastikan bahwa setiap operasi yang dilakukan oleh pengguna, terutama pada proses unggah (*upload*) data atau berkas, hanya dapat dilakukan oleh pengguna yang memiliki hak akses sesuai dengan perannya.

Langkah pertama yang dilakukan adalah melakukan audit menyeluruh terhadap seluruh modul yang memiliki fitur unggah data, termasuk modul *Asset Management*, *Recommendation*, dan *FM Database*. Dari hasil audit tersebut ditemukan bahwa beberapa endpoint belum memiliki pembatasan akses yang spesifik berdasarkan peran (*role*) pengguna. Untuk itu, diterapkan validasi tambahan pada setiap proses *upload* dengan memeriksa nilai `role` yang terdapat di dalam objek `claims` hasil dekripsi token JWT.

Pseudocode Sign In 3.2.

```
1
2  function main_request_handler(request):
3      path = request.get_path()
4
5      # 1. Proses pengecekan jika endpoint adalah /signin atau /
       forgotpass
6      if path.starts_with("/signin") or path.starts_with("/
       forgotpass"):
7          return continue_request(request)
8
```

```
    # 2. Mengambil token dari cookie
    token = request.get_cookie("token")

    if token is empty:
        return response_error("No Token", "Authentication required. Please login first")

    # 3. Mengambil secret JWT dari environment
    jwt_secret = read_env("JWT_SECRET", default="secret")

    # 4. Decode token JWT
    token_data = decode_jwt(token, jwt_secret)

    if token_data is invalid:
        return response_error("Invalid Token", "Please login again")

    # 5. Mengecek apakaha token sudah kadaluwarsa
    if token_data.claims.exp < current_timestamp();
        return response_error("Token Expired", "Your session has expired. Please login again")

    # 6. Menyimpan claims ke dalam request
    request.set_claims(token_data.claims)

    # 7. Menyimpan request ke endpoint berikutnya
    return continue_request(request)
```

Kode 3.2: Pseudocode Role Access

Pengecekan dilakukan menggunakan kondisi sederhana yang memastikan hanya pengguna dengan peran `admin` yang dapat melanjutkan proses *upload*. Jika pengguna tidak memiliki hak akses yang sesuai, sistem akan mengembalikan respons dengan kode status 403 disertai pesan kesalahan `"Forbidden: admin only"`. Pada Gambar 3.4. penerapan validasi ini bertujuan untuk mencegah manipulasi data dan menjaga agar operasi penting seperti unggah aset, konfigurasi, maupun impor data hanya dilakukan oleh pengguna yang berwenang.

```
let claims = req.ext::<Claims>().unwrap();
if claims.role != "admin" {
    return ws_response("Error", "Forbidden: admin only");
}
```

Gambar 3.4. User Access Control

Dengan adanya penerapan *Access Control* ini, struktur keamanan aplikasi menjadi lebih terjamin, dan risiko kesalahan operasional akibat akses tidak sah dapat diminimalkan. Selain itu, penerapan sistem peran (*role-based access control*) juga memberikan fleksibilitas dalam pengelolaan hak akses pengguna, memungkinkan pengembang untuk menambahkan atau memodifikasi peran di masa mendatang tanpa mengubah arsitektur utama sistem.

Secara keseluruhan, fitur ini memperkuat lapisan keamanan backend Relogica dan memastikan bahwa setiap interaksi pengguna dengan sistem berlangsung sesuai prinsip otorisasi yang ketat dan terukur.

Pengujian performa dan kualitas *service* diuji menggunakan *Whitebox Testing* dan *Blackbox Testing* dan diuji oleh dua personel *Backend*.

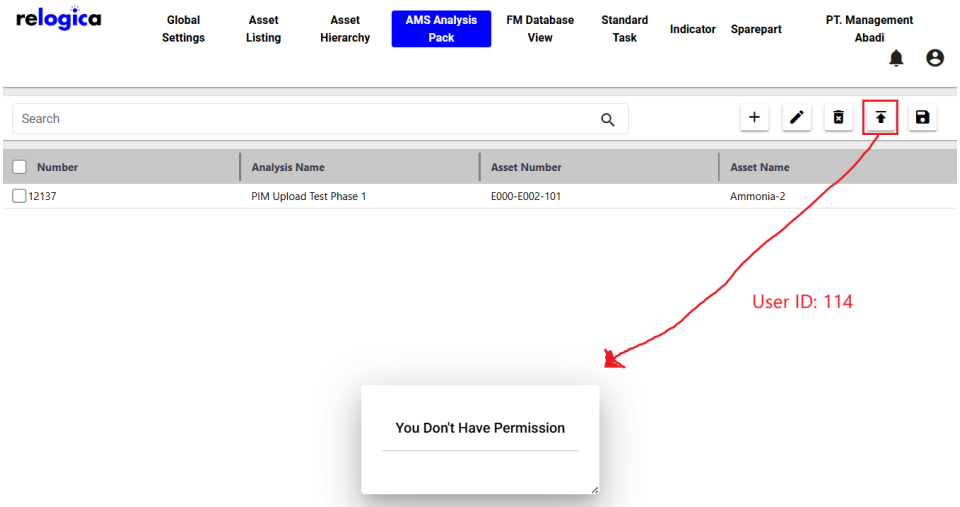
A Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada seluruh endpoint upload, dilakukan pengujian dengan berbagai skenario role (admin, non-admin, role tidak valid) dan memeriksa apakah response yang dihasilkan sesuai ekspektasi.

Tabel 3.4. Tabel Pengujian Access Control (`auth.rs`) - *Blackbox*

No	Endpoint	Skenario Pengujian	Status
1	Endpoint upload (Asset, Recommendation, FM)	Request dengan token role admin, upload berhasil	Lulus
2	Endpoint upload (Asset, Recommendation, FM)	Request dengan token role non-admin, response 403 Forbidden: admin only	Lulus
3	Endpoint upload	Request tanpa token atau token tanpa field role, response error otorisasi	Lulus
4	Endpoint upload	Request dengan token role tidak valid (random string), response error otorisasi	Lulus
5	Endpoint upload	Uji perubahan role di token, cek fleksibilitas akses sesuai role	Lulus

Pada Gambar 3.5. *User* yang terlogin sebagai *user* biasa, tidak dapat menyelesaikan proses *upload* di menu AMS Analysis Pack dikarenakan hanya *user* dengan role *Admin* saja yang bisa melakukan proses tersebut.



Gambar 3.5. Non-Admin User Upload Permission Denied - Access Control

B Whitebox Testing

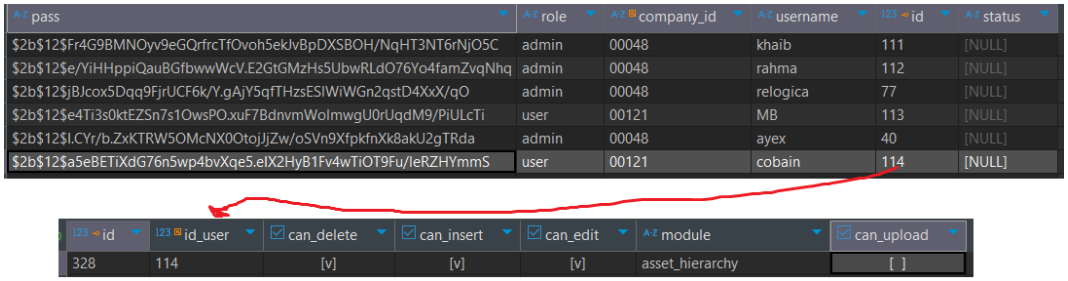
Pengujian dilakukan dengan mengetahui struktur internal kode, logika validasi role, pengecekan claims JWT, serta integrasi dengan endpoint upload dan modul lain. Pada middleware dan endpoint upload, pengujian whitebox memastikan proses otorisasi berjalan sesuai logika role-based access control.

Tabel 3.5. Tabel Pengujian Access Control (`auth.rs`) - *Whitebox*

No	Endpoint/Fungsi	Skenario Pengujian	Status
1	Middleware `check_user`	Validasi claims.role pada setiap request upload, hanya admin yang lolos	Lulus
2	Middleware `check_user`	Penanganan claims.role non-admin, response 403 Forbidden: admin only	Lulus
3	Middleware `check_user`	Penanganan claims.role hilang atau kosong, response error otorisasi	Lulus
4	Middleware `check_user`	Integrasi dengan endpoint upload di modul Asset, Recommendation, FM Database	Lulus
5	Middleware `check_user`	Penambahan/ubah role baru, cek fleksibilitas logika role-based access	Lulus
6	Middleware `check_user`	Penanganan claims injection dan validasi tipe data role	Lulus

Pada Gambar 3.6. *user* dengan id 114 adalah *user* yang sedang aktif di page tersebut, terlihat dari *database* bahwa ”can_upload” bernilai *false* yang

menyebabkan keterbatasan aktivitas *upload*.



pass	role	company_id	username	id	status
$2b$12$Fr4G9BMNOyv9eGQrfrcTfOvoh5eklvBpDXSBOH/NqHT3NT6rNjO5C	admin	00048	khaib	111	[NULL]
$2b$12$e/YiHHppiQauBGfbwwWcV.E2GtGMzHs5UbwRLdO76Yo4famZvqNhq	admin	00048	rahma	112	[NULL]
$2b$12$jBJcox5Dqq9FjrUCF6k/Y.gAjY5qfTHzsESIWiWGn2qstD4XxX/qO	admin	00048	relogica	77	[NULL]
$2b$12$e4Ti3s0ktEZSn7s1OwsPO.xuF7BdnvmWolmwgU0rUqdM9/PiULcTi	user	00121	MB	113	[NULL]
$2b$12$I.CYr/b.ZxKTRW5OMcNX0OtojJjZw/oSVn9XfpkfnXk8akU2gTRda	admin	00048	ayex	40	[NULL]
$2b$12$a5eBETiXdG76n5wp4bvXqe5.elX2HyB1Fv4wTiOT9Fu/IeRZHYmmS	user	00121	cobain	114	[NULL]

id	id_user	can_delete	can_insert	can_edit	module	can_upload
328	114	[v]	[v]	[v]	asset_hierarchy	[ ]

Gambar 3.6. Role and Permission Database View - Access Control

3.3.4 CRUD & Validasi Data (Recommendation, Task, Hierarchy)

Pada tahap ini, fokus utama pengembangan sistem Relogica adalah memperkuat keandalan proses *CRUD*[10] (*Create, Read, Update, Delete*) serta validasi data pada modul *Recommendation* dan *Hierarchy*. Tujuan utamanya adalah memastikan setiap perubahan data benar-benar tercatat di basis data, serta mencegah terjadinya operasi yang tidak sah berdasarkan konteks perusahaan (`company_id`) dan ruang lingkup pengguna yang sedang aktif.

Langkah pertama yang dilakukan adalah memeriksa serta memperbaiki endpoint `add` dan `delete` pada modul *Recommendation* (terutama pada entitas `rcmd_task` dan `standard_task`). Sebelumnya, ditemukan bahwa beberapa operasi penghapusan tidak benar-benar memengaruhi data di basis data karena tidak dilakukan verifikasi terhadap jumlah baris yang terpengaruh. Untuk mengatasi hal tersebut, dilakukan pemeriksaan terhadap hasil eksekusi query dengan memanfaatkan fungsi `rows_affected()`. Jika nilai yang dikembalikan adalah nol, sistem akan mengembalikan respons bahwa data tidak ditemukan atau pengguna tidak memiliki izin untuk melakukan operasi tersebut. Dengan demikian, hanya data yang relevan dengan perusahaan dan pengguna yang benar-benar dapat diubah atau dihapus.

Pseudocode Insert dan Add Rcmd Task 3.3.

```
1
2 function add_rcmd_task(request):
3
4     user = username from JWT
5     param = parse JSON body into RcmdTask
6     pool = DB
7
```

```
    inserted_id = insert_rcmd_task(param.fm_id, param.task_id,
    param.is_primary_plan, pool)

    if inserted_id is not None:
        return ws_response("OK", "Successfully inserted rcmd task"
    )
    else:
        return ws_response("Error", "Failed to insert rcmd task")

function insert_rcmd_task(fm_id, task_id, is_primary_plan, pool):

    SQL INSERT INTO rcmd_task(fm_id, task_id, is_primary_plan)
        RETURNING id

    if query success:
        return new id
    else:
        log error
        return None
```

Kode 3.3: Pseudocode Insert dan Add Rcmd Task

Pseudocode Delete Rcmd Task 3.4.

```

function delete_rcmd_task(request):

    user = username from JWT
    param = parse query -> DelParam(id)
    pool = DB

    Execute SQL:
        DELETE FROM rcmd_task WHERE id = param.id

    if success:
        return ws_response("OK", "Successfully deleted")
    else:
        return ws_response("Error", "Failed to delete")

function delete_rcmd_task_by_fm_id(fm_id, pool):

    Execute SQL:
        DELETE FROM rcmd WHERE fm_id = fm_id

    if success:
```

```
        return OK response
    else:
        return Error response
```

Kode 3.4: Pseudocode Delete Rcmd Task

Selanjutnya, dilakukan penguatan pada proses pembaruan posisi dan relasi `parent-child` pada modul *asset hierarchy*, khususnya ketika pengguna melakukan aksi *drag and drop*. Saat posisi atau hierarki aset diperbarui, sistem secara otomatis menjalankan perintah `UPDATE` untuk memperbarui nilai `parent_id` dan `position` di basis data sesuai dengan urutan terbaru. Pendekatan ini memastikan struktur hierarki tetap konsisten setelah setiap perubahan posisi dilakukan melalui antarmuka pengguna.

Pseudocode Get Asset Hierarchy 3.5.

```

function get_asset_hierarchy(request):
    user = get username from JWT
    param = parse query parmeters
    pool = DB connection

    # Ambil nama perusahaan
    site = query "SELECT company_name FROM company WHERE id =
   param.company_id"

    # Ambil semua asset milik perusahaan
    asset_list = get_assets(pool, param.company_id)

    # Tentukan parent awal
    if param.parent_asset exists:
        parents = asset_list filtered by parent_asset == param.
   parent_asset
    else:
        parents = asset_list filtered by parent_asset == NULL

    reseult = []

    for each parent in parents:
        file_path = [company_name, "id - asset_name"]
        create Tree object for parent
        append to result

        # Rekursi: tambahkan seluruh children
```

```
        call add_child_tree(company_name, file_path, parent.id,
    asset_list, result)

    return response JSON result
```

Kode 3.5: Pseudocode Get Asset Hierarchy

Pseudocode Update Asset Hierarchy 3.6.

```

function change_asset_parent(request):

    user_id = get from JWT
    body = parse JSON
    pool = DB

    # 1. Ambil informasi asset lama
    existing = SELECT asset fields, parent FROM asset_hierarchy
   WHERE id = body.asset_id
    if not found: return error

    # 2. Validasi parent yang baru
    if new parent exists:
        check that new_parent_asset exists in DB
        check new_parent_asset != asset_id

    # 3. Validasi hierarchy_type dan asset_type_id
    ensure hierarchy_type_id != asset_type_id

    # 4. Gunakan body value jika ada, jika tidak pakai existing
    final values = merged values

    # 5. Update Asset
    UPDATE asset_hierarchy SET updated fields WHERE id = asset_id

    # 6. Log Perubahan
    INSERT INTO user_asset_hierarchy_log

    return OK
```

Kode 3.6: Pseudocode Update Asset Hierarchy

Selain itu, validasi data juga diterapkan pada proses *upload* berkas untuk modul *asset hierarchy*. Sistem memeriksa keberadaan header penting seperti `asset_id` dan memastikan bahwa setiap aset yang diunggah sudah terdaftar di dalam sistem sebelum dilakukan operasi `INSERT`. Jika aset tidak ditemukan,

sistem akan menolak permintaan dengan pesan kesalahan yang sesuai. Proses ini membantu mencegah inkonsistensi data dan memastikan hanya aset yang valid yang dapat ditambahkan ke dalam hierarki.

Pseudocode Add Child or Parent to Tree 3.7.

```

function add_child_tree(company, file_path, asset_id, all_assets, result):
    children = all_assets filtered where parent_asset == asset_id

    if children empty: return

    for each child in children:
        new_path = file_path + ["id - asset_name"]

        create Tree Object
        append to result

        # Rekursi ke level berikutnya
        add_child_tree(company, new path, child.id, all_assetts, result)

function add_child_parent(pool, asset_id, assets, result):

    if asset_id is None: return

    children = assets filtereed where parent_asset == asset_id

    add asset_id to result

    for each child in children:
        add_child_parent(pool, child.id, assets, result)
```

Kode 3.7: Pseudocode Add Child or Parent to Tree

Secara keseluruhan, pembenahan pada tahap ini meningkatkan integritas data dalam sistem Relogica dengan cara menggabungkan verifikasi berbasis hasil eksekusi query, kontrol akses berbasis `company_id`, serta validasi input yang ketat sebelum proses penyimpanan ke basis data. Dengan demikian, setiap perubahan yang dilakukan oleh pengguna kini lebih aman, transparan, dan terhindar dari manipulasi yang tidak sah.

Pengujian performa dan kualitas *service* diuji menggunakan *Whitebox Testing* dan *Blackbox Testing* dan diuji oleh dua personel *Backend*.

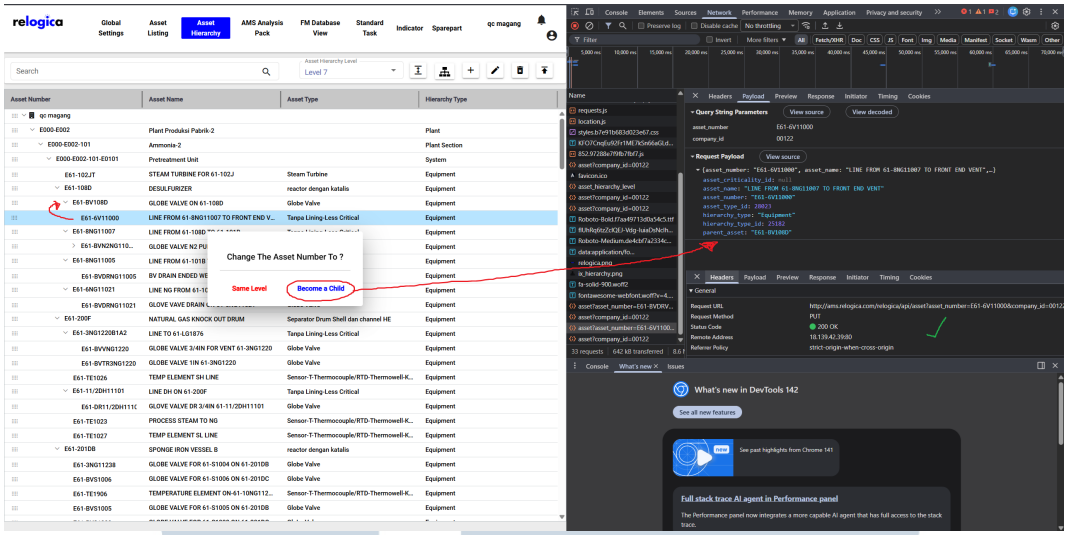
A Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada seluruh endpoint CRUD, dilakukan pengujian dengan berbagai skenario input (data valid, data tidak valid, company_id salah, asset tidak ditemukan, dsb) dan memeriksa apakah response yang dihasilkan sesuai ekspektasi.

Tabel 3.6. Tabel Pengujian CRUD & Validasi Data (`rcmd_task.rs`, `asset_hierarchy.rs`) - *Blackbox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /add_rcmd_task`	Input valid, data berhasil ditambah; input fm_id/task_id salah, response error	Lulus
2	`DELETE /delete_rcmd_task`	Data ada, berhasil dihapus; data tidak ada, response error ”not found”	Lulus
3	`PUT /update_rcmd_task`	Update data valid, data berubah; update id salah, response error	Lulus
4	`GET /get_rcmd_task`	Query dengan fm_id/id valid, data sesuai; query dengan id/fm_id salah, data kosong	Lulus
5	`POST /add_asset`	Asset_id dan company_id valid, asset ditambah; asset_id tidak terdaftar, response error	Lulus
6	`DELETE /delete_asset`	Asset ada, berhasil dihapus; asset tidak ada, response error	Lulus
7	`PUT /update_asset`	Update asset valid, parent-child konsisten; update parent asset tidak valid, response error	Lulus
8	`POST /change_asset_parent`	Parent asset valid, perubahan berhasil; parent asset tidak valid, response error	Lulus
9	`POST /upload_asset_hierarchy`	Upload file dengan asset_id valid, data masuk; asset_id tidak terdaftar, response error	Lulus
10	`GET /get_asset_hierarchy`	Query dengan company_id valid, struktur tree benar; company_id salah, data kosong	Lulus

Pada Gambar 3.7. *asset* dengan nama E61-6V11000 dijadikan menjadi *child asset* E61-BV108D. Proses perpindahan level dari level yang sama ke level child dari suatu *asset* berhasil terproses.



Gambar 3.7. Same Level to Child Transition - Asset Hierarchy Update

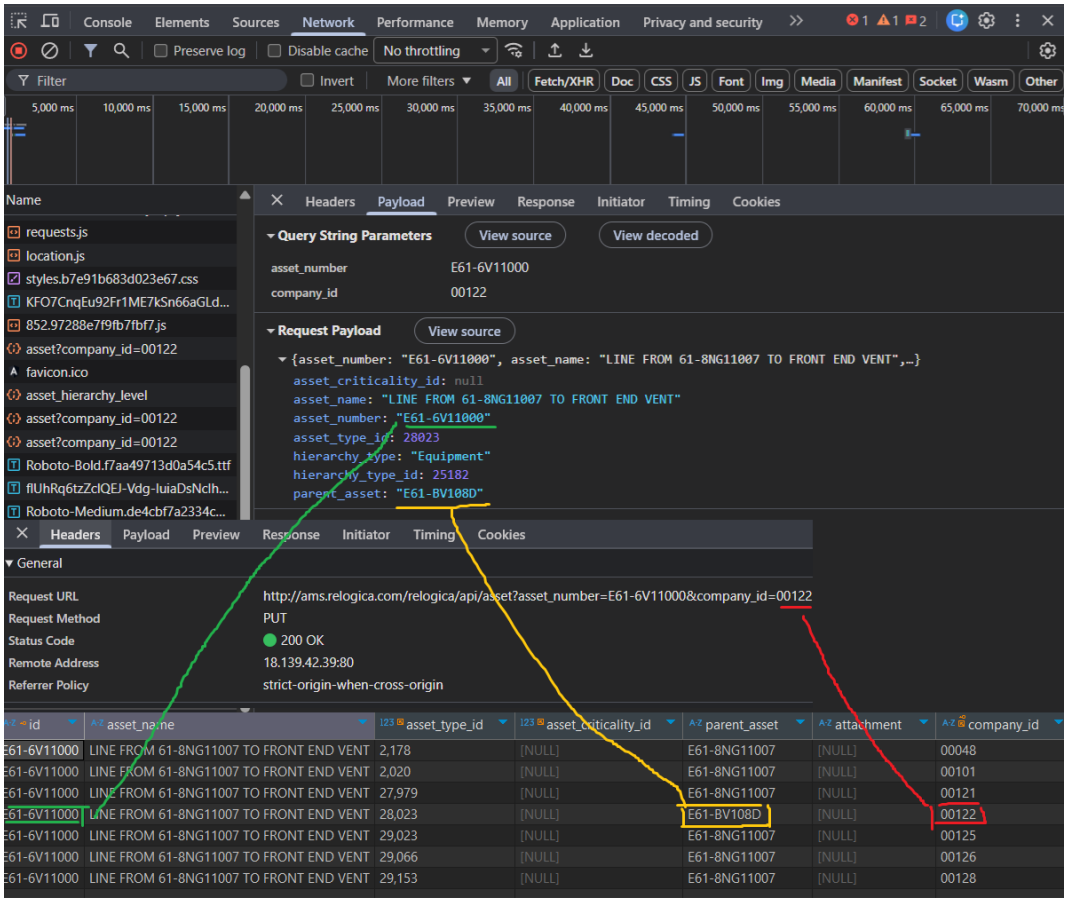
B Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika query, validasi input, pengecekan rows_affected, serta integrasi dengan database dan kontrol akses company_id.

Tabel 3.7. Tabel Pengujian CRUD & Validasi Data (`rcmd_task.rs`, `asset_hierarchy.rs`) - *Whitebox*

No	Endpoint/Fungsi	Skenario Pengujian	Status
1	`POST /add_rcmd_task`	Validasi input, insert rcmd_task, cek hasil query dan id yang dikembalikan	Lulus
2	`DELETE /delete_rcmd_task`	Eksekusi delete, cek rows_affected, response error jika data tidak ditemukan	Lulus
3	`PUT /update_rcmd_task`	Validasi input, update rcmd_task, cek hasil query	Lulus
4	`GET /get_rcmd_task`	Query rcmd_task dengan filter fm_id dan id, pastikan data sesuai scope user	Lulus
5	`POST /add_asset`	Validasi asset_id, parent_asset, company_id, insert asset baru, cek hasil query	Lulus
6	`DELETE /delete_asset`	Delete asset dengan filter asset_number dan company_id, cek rows_affected	Lulus
7	`PUT /update_asset`	Update asset, validasi parent-child, cascade update ke child dan log perubahan	Lulus
8	`POST /change_asset_parent`	Validasi parent asset, update parent_asset, log histori perubahan	Lulus
9	`POST /upload_asset_hierarchy`	Validasi header asset_id, cek keberadaan asset sebelum insert, response error jika tidak ditemukan	Lulus
10	`GET /get_asset_hierarchy`	Query asset dengan filter company_id, parent_asset, pastikan struktur tree konsisten	Lulus

Pada Gambar 3.8. terlihat dari *database view* bahwa *asset* E61-6V11000 berubah nilai ”parent_asset” menjadi E61-BV108D, terlihat juga ada ”company_id” yang digunakan untuk mengeksklusifkan suatu *asset* dari *asset* lainnya.



Gambar 3.8. Database View Post-Transition - Asset Hierarchy Update

3.3.5 AMS Analysis Pack & FM Database

Tahap pengembangan berikutnya difokuskan pada integrasi antara modul *AMS Analysis Pack* dan *FM Database* di dalam sistem Relogica. Tujuan utama dari tahap ini adalah untuk memastikan proses pemetaan data antara *AMS* (Asset Management System) dan *FM Database* berjalan secara konsisten, terutama dalam hal kesesuaian struktur dan isi data yang diunggah ke dalam sistem.

Salah satu permasalahan yang ditemukan pada tahap awal adalah adanya ketidaksesuaian antara kolom yang berasal dari *AMS* dengan kolom yang ada pada tabel di *FM Database*, khususnya pada bidang `Task Type` dan `Recommended Strategy`. Untuk mengatasi hal tersebut, dilakukan perbaikan pada proses pemetaan (*field mapping*) agar setiap entri data dari *AMS* dapat diterjemahkan dengan benar ke dalam format yang digunakan oleh *FM Database*. Proses pemetaan

ini melibatkan konversi nilai teks secara terstandardisasi, seperti melakukan operasi `TRIM()` dan `LOWER()` agar tidak terjadi kesalahan akibat perbedaan kapitalisasi atau spasi tambahan yang terdapat pada Gambar 3.9.

```
let rec_strategy: String = row.get("Recommended Strategy");
let task_type_name = rec_strategy.trim();
let task_type_id: Option<i32> = sqlx::query_scalar(
    "SELECT id FROM task_type WHERE LOWER(TRIM(task_type)) = LOWER(TRIM($1)) AND company_id = $2"
)
.bind(&task_type_name).bind(&company_id).fetch_optional(pool).await?;
// jika tidak ada, insert atau beri alert
```

Gambar 3.9. AMS Analysis Pack Insert - Task Type Validation

Selain itu, sistem juga dilengkapi dengan mekanisme validasi yang lebih ketat terhadap nilai `asset_id`. Setiap kali pengguna melakukan proses *upload* data, sistem akan memeriksa terlebih dahulu apakah `asset_id` tersebut sudah terdaftar di tabel `asset_hierarchy`. Jika data tidak ditemukan, maka sistem akan menampilkan pesan kesalahan yang bersifat informatif, misalnya *"Upload failed: asset_id 1234 not registered"* pada Gambar 3.10. Pesan ini memudahkan pengguna untuk segera memahami sumber kesalahan dan memperbaikinya tanpa harus menelusuri log sistem secara manual.

```
if asset_not_found {
    return ws_response("Error", format!("Upload failed: asset_id {} not registered", asset_id).as_str());
}
```

Gambar 3.10. Error Alert for Unknown Asset

Pseudocode Upload/Import AMS Analysis Pack and Asset Validation **??**.

```

function import_fmea_fm(request):

    user = extract username from JWT or "Unknown"
    log "start import_fmea_fm"

    company_id = request.query.company_id
    body_json = request.body_string

    records = parse JSON -> Vec<ImportFmeaFM>
    if parsing fails:
        return JSON client error with specific message

    log number of records parsed

    # Validation
```

```
    validation_errors = []
    for each record (index i):
        if failure_mode empty -> push error with row number
        if asset_id empty -> push error
    if any validation errors:
        return JSON error with list

    pool = DB connection
    error_data = []

    # Process Each Row
    for each import (index row):

        failure_mode = trimmed string
        asset_id = trimmed import.asset_id
        asset_id_fm = import.asset_id_fm

        # 1. Check asset existence
        asset_exists = SQL EXISTS(asset_id, company_id)
        if not exists:
            log error, push error_data with row info
            continue

        # 2. Resolve reference tables (upsert or lookup)
        fc_id = get_or_insert_fc(...)
        sym_id = get_or_inser_symptopms(...)
        tech_id = get_id_tech_by_value(...)
        task_type_id = get_or_insert_task_type(...)

        useful_life_uom_id = get_or_insert_uom
        pf_interval_uom_id = get_or_insert_uom
        ff_interval_uom_id = get_or_insert_uom

        sparepart_ids = parse sparepart list -> Vec<String>
        spare_ids = get_id_sparepart_by_value(sparepart_ids)

        # 3. Determine task_id (task master table)
        task_id = get_id_task_by_value(
            task_description,
            indicator,
            frequency,
            op_condition,
            trade,
```

```
            duration,
            uom,
            company_id,
            asset_id_fm,
            pf_interval, pf_uom,
            useful_life, useful_life_uom,
            ff_interval, ff_uom
        )

        # 4. Upsert FMEA
        try INSERT fmea(analysis_name, asset_id, company_id)
        if conflict -> fetch existing fmea.id
        fmea_id = id

        # 5. Delete fm that conflicts by failure_mode + asset_id + company_id
        SQL DELETE FROM fm WHERE lower(failure_mode) = lower($1)
                             AND asset_id = $2
                             AND company_id = $3

        # 6. Insert new FM
        try INSERT INTO fm(...) RETURNING id
        if fails:
            log error, push error_data
            continue
        fm_id = returned_id
        log "FM inserted/updated"

        # 7. Insert rcmd_task
        DELETE existing rcmd_task for fm_id
        if task_id not empty:
            INSERT rcmd_task(fm_id, task_id, is_primary_plan=true)

        # 8. Insert credible_sparepart
        DELETE existing credible_sparepart for fm_id
        for each part id in spare_ids:
            INSERT credible_sparepart(fm_id, part_id)

    return ok_with_data("Import Done", "error_data", error_data)
```

Kode 3.8: Pseudocode Upload/Import AMS Analysis Pack and Asset Validation

Dalam proses pengolahan data *Recommended Strategy*, sistem juga melakukan pencocokan terhadap tabel `task_type`. Jika jenis tugas yang dimaksud

belum tersedia di basis data, maka sistem akan melakukan dua kemungkinan tindakan, yaitu melakukan penyisipan data baru (`INSERT`) atau memberikan peringatan kepada pengguna agar meninjau kembali data yang diunggah. Langkah ini menjaga agar referensi data tetap konsisten di seluruh modul aplikasi.

Terakhir, endpoint pada modul *FM* disesuaikan untuk dapat menampilkan kolom `Task Type` berdasarkan nilai yang diambil dari kolom `Recommended Strategy`. Dengan cara ini, data yang bersumber dari *AMS* dapat disajikan secara komprehensif di antarmuka *FM*, memastikan keterhubungan antara hasil analisis dan konteks aset di lapangan tetap terjaga.

Pseudocode Get FM Data and Task Type Validation **??**.

```

function getFMData():

    # 1. Ambil data FM dari database
    fm_list = query("
        SELECT id, faillure_mode recommended_strategy
        FROM fm
    ")

    result = []

    # 2. Untuk setiap baris FM
    for each fm in fm_list:

        recommended = TRIM(LOWER(fm.recommended_strategy))

        # 3. Lakukan pencarian task type berdasarkan recommended strategy
        task_type = query("
            SELECT task_type
            FROM task_type
            WHERE LOWER(name) = ?
        ", recommended)

        # 4. Jika tidak ditemukan -> gunakan nilai default atau beri peringatan
        if task_type is null:
            task_type = "UNDEFINED"

        # 5. Susun kembali objek FM untuk output endpoint
        output_item = {
```

```
            id: fm.id,
            failure_mmode: fm.failure_mode,
            recommended_strategy: fm.recommended_strategy,
            task_type: task_type
        }

        append output_item to result

    return response_json(result)
```

Kode 3.9: Pseudocode Get FM Data and Task Type Validation

Secara keseluruhan, perbaikan di tahap ini meningkatkan integritas dan keterpaduan data antara sistem *AMS* dan *FM Database*. Melalui penerapan validasi `asset_id`, standarisasi pemetaan kolom, serta pemberian pesan kesalahan yang ramah pengguna, proses integrasi data kini berjalan lebih andal dan transparan di dalam ekosistem Relogica.

Pengujian performa dan kualitas *service* diuji menggunakan *Whitebox Testing* dan *Blackbox Testing* dan diuji oleh dua personel *Backend*.

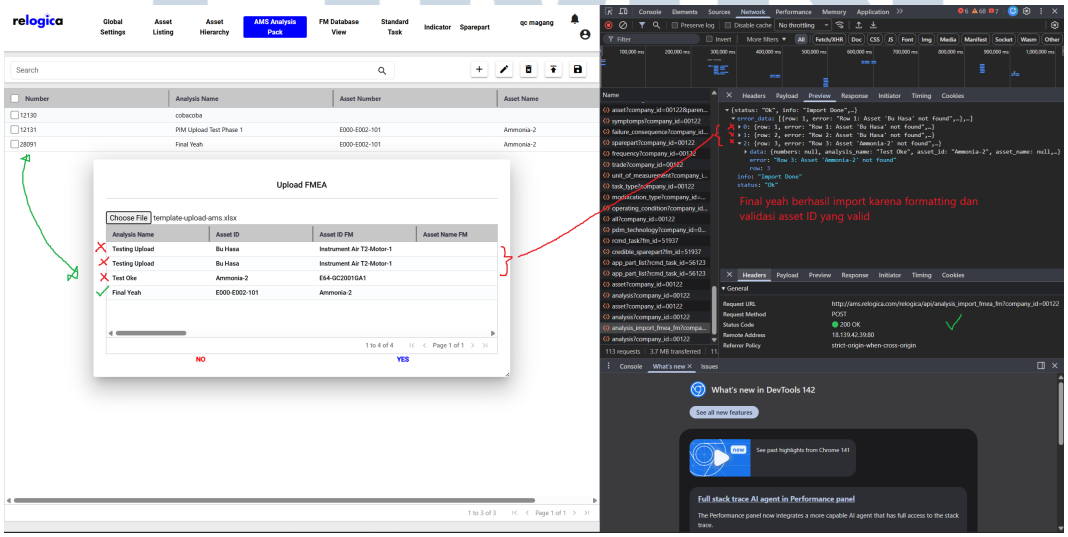
A Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada seluruh proses upload dan query FM, dilakukan pengujian dengan berbagai skenario input (data valid, asset id tidak terdaftar, task type baru, recommended strategy berbeda format, dsb) dan memeriksa apakah response yang dihasilkan sesuai ekspektasi.

Tabel 3.8. Tabel Pengujian Integrasi AMS Analysis Pack & FM Database - *Blackbox*

No	Endpoint/Proses	Skenario Pengujian	Status
1	Upload AMS Analysis Pack	Upload data dengan asset id valid, data masuk ke FM Database	Lulus
2	Upload AMS Analysis Pack	Upload data dengan asset id tidak terdaftar, response error “Upload failed: asset id ... not registered”	Lulus
3	Upload AMS Analysis Pack	Upload data dengan Task Type baru, data task type otomatis ditambah ke database	Lulus
4	Upload AMS Analysis Pack	Upload data dengan Task Type berbeda format/kasus kapitalisasi, data tetap terstandardisasi	Lulus
5	Upload AMS Analysis Pack	Upload data dengan Recommended Strategy berbeda format, data tetap terstandardisasi	Lulus
6	Endpoint FM Database	Query FM, kolom Task Type tampil sesuai Recommended Strategy dari AMS	Lulus
7	Upload AMS Analysis Pack	Upload data dengan asset id kosong, response error validasi	Lulus
8	Upload AMS Analysis Pack	Upload data dengan field mapping salah, response error/warning sesuai	Lulus

Pada Gambar 3.11. proses *upload* file excel dengan 4 row yang memiliki Analysis Name yang bervariasi dicoba untuk keberhasilan proses *upload* ke database. Setelah proses selesai, terlihat bahwa *row* dengan nama ”Final Yeah” tidak masuk ke dalam payload ”error_data” karena row tersebut berhasil masuk ke daftar AMS Analysis Pack. *Row* yang lain gagal dikarenakan sudah adanya Analysis Name yang sama dan juga karena Analysis Name yang ingin di-*upload* tidak memiliki *asset* yang terdaftar di database dengan company_id 00122.



Gambar 3.11. New Data Existence - AMS Upload

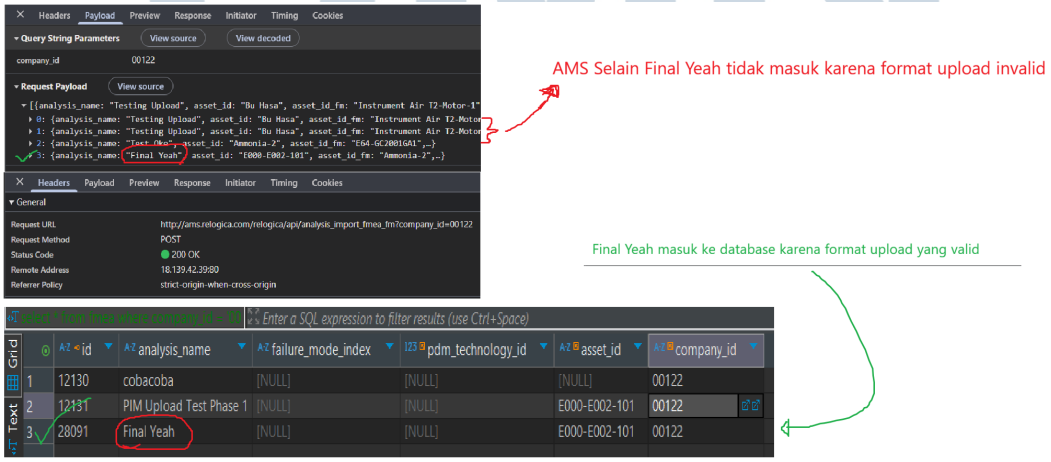
B Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika pemetaan field, validasi asset id, pencocokan dan insert task type, serta penyesuaian endpoint FM agar menampilkan data Recommended Strategy secara konsisten.

Tabel 3.9. Tabel Pengujian Integrasi AMS Analysis Pack & FM Database - *Whitebox*

No	Fungsi/Endpoint	Skenario Pengujian	Status
1	Proses upload AMS Analysis Pack	Validasi asset id, cek asset sudah terdaftar di asset hierarchy sebelum insert	Lulus
2	Proses upload AMS Analysis Pack	Pemetaan field Task Type dan Recommended Strategy, konversi TRIM() dan LOWER() sebelum insert	Lulus
3	Proses upload AMS Analysis Pack	Pencocokan task type, insert jika belum ada, atau warning jika data tidak konsisten	Lulus
4	Proses upload AMS Analysis Pack	Penanganan error upload, response informatif jika asset id tidak ditemukan	Lulus
5	Endpoint FM Database	Penyesuaian query agar kolom Task Type diambil dari Recommended Strategy	Lulus
6	Endpoint FM Database	Validasi dan mapping data dari AMS agar tampil konsisten di FM Database	Lulus
7	Proses upload AMS Analysis Pack	Penanganan spasi/kasus kapitalisasi pada field mapping, pastikan data tidak duplikat	Lulus
8	Proses upload AMS Analysis Pack	Penanganan upload data dengan asset id tidak valid, response error sesuai	Lulus

Gambar 3.12. menunjukkan keberadaan Analysis Name ”Final Yeah” di database setelah proses *upload* berhasil.



Gambar 3.12. Database View New Data Existence - AMS Upload

3.3.6 Table Configuration & User Preferences

Pada tahap ini, fokus pengembangan diarahkan pada pembuatan dan penyempurnaan fitur *Table Configuration & User Preferences*, yaitu layanan yang memungkinkan setiap pengguna menyimpan preferensi tampilan tabel secara individual di modul *Standard Task* dan *FM Database*. Fitur ini memberikan fleksibilitas bagi pengguna untuk mengatur kolom mana saja yang ingin ditampilkan atau disembunyikan, serta mengatur lebar kolom dan urutan tampilan sesuai kebutuhan mereka.

Implementasi fitur ini dilakukan dengan pendekatan *per-user configuration*, di mana setiap entri konfigurasi tabel disimpan di tabel `table_configuration` dengan relasi langsung terhadap `user_id`. Hal ini memastikan bahwa preferensi setiap pengguna bersifat personal dan tidak memengaruhi tampilan pengguna lain. Untuk menjamin integritas data dan menghindari duplikasi entri, digunakan mekanisme `UPSERT` dengan perintah `ON CONFLICT` yang memanfaatkan kombinasi unik dari kolom `field`, `table_name`, dan `user_id`. Dengan cara ini, sistem dapat melakukan penyisipan (`INSERT`) data baru atau pembaruan otomatis (`UPDATE`) jika data konfigurasi untuk kombinasi tersebut sudah ada sebelumnya.

Pseudocode Table Confguration Per User **??**.

```

function getTableConfiguration(userId):

    # Ambil konfigurasi milik user
    userConfig = QUERY:
        SELECT * FROM table_configuration
        WHERE user_id = userId
        ORDER BY column_index

    # Jika belum punya konfigurasi, gunakan default (user_id IS
    NULL)
    if userConfig is empty:
        defaultConfig = QUERY:
            SELECCT * FROM table_configuration
            WHERE user_id IS NULL
            ORDER BY column_index
        return defaultConfig

    return userConfig
end function
```

```
20
21 function addOrUpdateTableConfiguration(userId, configList):
22
23     for each configItem in configList:
24
25         # Validasi wajib
26         if configItem.field is NULL or configItem.table_name is
   NULL:
27             return error("Missing required field or table name")
28
29         # Mekanisme UPSERT
30         EXECUTE:
31             INSERT INTO table_configuration (...)
32             VALUES (configItem..., userId)
33             ON CONFLICT (field, table_name, user_id)
34             DO UPDATE SET
35                 header_name = EXCLUDED.header_name,
36                 checkbox_selection = EXCLUDED.checkbox_selection,
37                 width = EXCLUDED.width,
38                 hide = EXCLUDED.hide,
39                 column_index = EXCLUDED.column_index
40
41             # Sinkronkan sequence ID
42             UPDATE sequence using last-inserted-id
43
44     return success("Configuration saved")
45 end function
```

Kode 3.10: Pseudocode Table Confguration Per User

Selain itu, perbaikan juga dilakukan pada logika `UPDATE` untuk menangani kasus di mana kolom `user_id` bernilai `NULL`. Nilai `NULL` ini menandakan konfigurasi global (yang berlaku untuk seluruh pengguna) sehingga diperlukan kondisi `WHERE` yang aman dan fleksibel agar pembaruan tidak menimpa konfigurasi pengguna lain. Kondisi tersebut memanfaatkan ekspresi:

```
(user_id = $8 OR (user_id IS NULL AND $8 IS NULL))
```

yang memungkinkan sistem membedakan antara konfigurasi personal dan global dengan tepat pada Gambar 3.13. berikut.

```rust
let result: Result<PgRow, Error> = sqlx::query(      You, 3 months ago • table conf
    sql: r#"
    INSERT INTO table_configuration (
        field, header_name, checkbox_selection, width, hide, table_name, column_index, user_id
    )
    VALUES ($1, $2, $3, $4, $5, $6, $7, $8)
    ON CONFLICT (field, table_name, user_id)
    DO UPDATE SET
        header_name = EXCLUDED.header_name,
        checkbox_selection = EXCLUDED.checkbox_selection,
        width = EXCLUDED.width,
        hide = EXCLUDED.hide,
        column_index = EXCLUDED.column_index
    RETURNING id
    "#,
) Query<'_, Postgres, PgArguments>
.bind(param.field.clone()) Query<'_, Postgres, PgArguments>
.bind(param.header_name.clone()) Query<'_, Postgres, PgArguments>
.bind(param.checkbox_selection) Query<'_, Postgres, PgArguments>
.bind(param.width) Query<'_, Postgres, PgArguments>
.bind(param.hide) Query<'_, Postgres, PgArguments>
.bind(param.table_name.clone()) Query<'_, Postgres, PgArguments>
.bind(param.column_index) Query<'_, Postgres, PgArguments>
.bind(user_id) Query<'_, Postgres, PgArguments>
.fetch_one(executor: pool) impl Future<Output = Result<…, …>>
.await;

if let Err(e: Error) = result {
    println!("error inserting/updating config: {:?}", e);
    return ws_response(status_val: "Error", info_val: "Failed to insert or update table configuration!");
}
```

Gambar 3.13. Upsert Table Configuration per User

Secara fungsional, sistem ini mendukung penyimpanan atribut konfigurasi penting seperti nama kolom (`header_name`), status visibilitas (`hide`), pemilihan kolom (`checkbox_selection`), lebar kolom (`width`), dan indeks posisi kolom (`column_index`). Dengan demikian, setiap kali pengguna mengakses halaman tabel, sistem dapat secara otomatis memuat preferensi yang telah disimpan sebelumnya, memberikan pengalaman antarmuka yang lebih personal dan efisien.

Melalui pengembangan fitur ini, aplikasi Relogica kini memiliki sistem konfigurasi tabel yang dinamis, adaptif, dan ramah pengguna. Tidak hanya meningkatkan fleksibilitas tampilan, tetapi juga memperkuat struktur manajemen data pengguna dengan penerapan mekanisme `UPSERT` yang efisien serta validasi logika pembaruan yang aman terhadap kondisi `NULL`.

Pengujian performa dan kualitas *service* diuji menggunakan *Whitebox Testing* dan *Blackbox Testing* dan diuji oleh dua personel *Backend*.

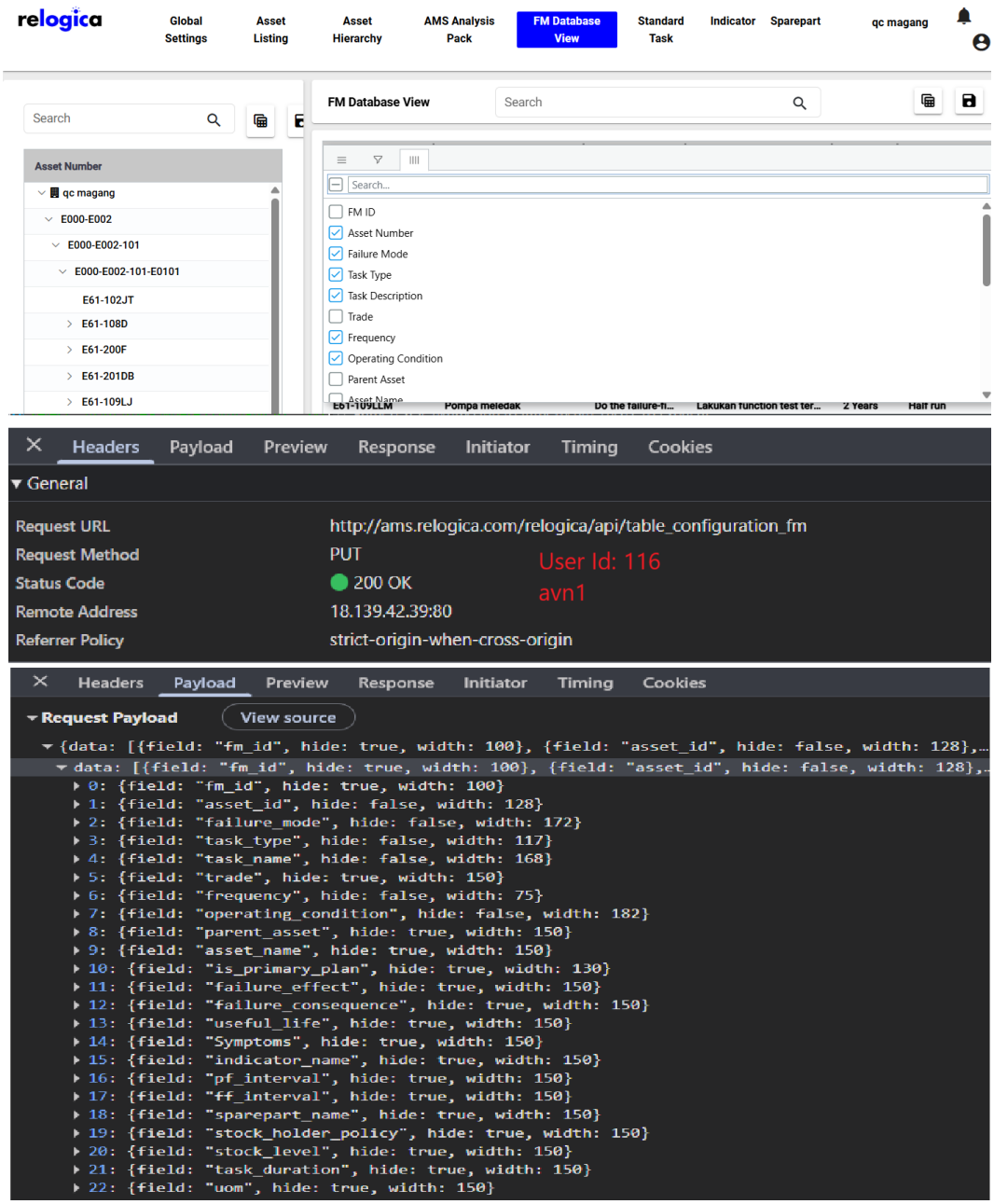
A Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan.

Tabel 3.10. Tabel Pengujian Table Configuration & User Preferences - *Blackbox*

No	Endpoint	Skenario Pengujian	Status
1	`POST /add_table_conf`	Input field dan `table_name` valid, konfigurasi tersimpan; input kosong, response error	Lulus
2	`PUT /update_table_conf`	Update konfigurasi per-user, data berubah; input duplikat diproses sebagai update	Lulus
3	`PUT /update_table_conf`	Update konfigurasi global (`user_id = NULL`), data global berubah tanpa memengaruhi user lain	Lulus
4	`DELETE /delete_table_conf`	Delete konfigurasi milik user; jika id salah, response error	Lulus
5	`GET /get_table_conf`	Query konfigurasi milik user; jika tidak ada, fallback ke konfigurasi global	Lulus
6	`POST /update_table_conf_fm`	Sinkronisasi FM Database: data user lama dihapus, lalu insert data baru	Lulus
7	`POST /update_table_conf_std_task`	Sinkronisasi Standard Task: data user lama dihapus, data baru masuk	Lulus
8	`POST /update_table_conf_list_rec_act`	Sinkronisasi List Recommendation: data user lama dihapus, data baru masuk	Lulus
9	`POST /update_table_conf_index_fm`	Sinkronisasi Index FM: data user lama dihapus, data baru masuk	Lulus
10	`POST /update_table_conf_index_std_task`	Sinkronisasi Index Standard Task: data user lama dihapus, data baru masuk	Lulus

Pada Gambar 3.14. terlihat daftar *row* yang mengandung banyak data mengenai konfigurasi table untuk setiap *user* setelah menjalankan proses *update table config*.



Gambar 3.14. Inlined Data of Config - Table Config per User

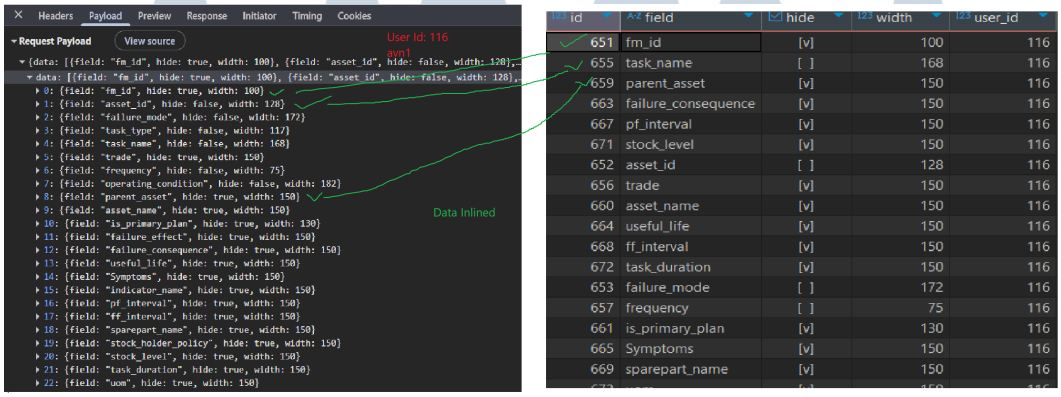
B Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika UPSERT (ON CONFLICT), validasi field dan `user_id`, penanganan konfigurasi global (`user_id = NULL`), serta integrasi dengan modul Standard Task dan FM Database.

Tabel 3.11. Tabel Pengujian Table Configuration & User Preferences - *Whitebox*

No	Endpoint/Fungsi	Skenario Pengujian	Status
1	`POST /add_table_conf`	Validasi field dan `table_name` wajib, insert konfigurasi baru dengan `user_id`, cek hasil query	Lulus
2	`PUT /update_table_conf`	UPSERT konfigurasi per-user, ON CONFLICT (`field`, `table_name`, `user_id`), update jika sudah ada	Lulus
3	`PUT /update_table_conf`	Penanganan `user_id = NULL`, kondisi WHERE aman: `(user_id = $8 OR (user_id IS NULL AND $8 IS NULL))`	Lulus
4	`DELETE /delete_table_conf`	Delete konfigurasi milik user, filter by `id` dan `user_id`, cek rows_affected	Lulus
5	`GET /get_table_conf`	Query konfigurasi milik user, fallback ke konfigurasi global jika tidak ada	Lulus
6	`POST /update_table_conf_fm`	Sinkronisasi konfigurasi FM Database per-user, hapus dan insert ulang data `user_id`	Lulus
7	`POST /update_table_conf_std_task`	Sinkronisasi konfigurasi Standard Task per-user, hapus dan insert ulang data `user_id`	Lulus
8	`POST /update_table_conf_list_rec_act`	Sinkronisasi konfigurasi List Recommendation per-user, hapus dan insert ulang data `user_id`	Lulus
9	`POST /update_table_conf_index_fm`	Sinkronisasi konfigurasi Index FM per-user, hapus dan insert ulang data `user_id`	Lulus
10	`POST /update_table_conf_index_std_task`	Sinkronisasi konfigurasi Index Standard Task per-user, hapus dan insert ulang data `user_id`	Lulus

Pada Gambar 3.15. keserasian atau data yang ada di payload sudah benar dengan data yang ada di *database*.



Gambar 3.15. Database View Inlined Data of Config - Table Config per User

3.3.7 Perbaikan Duplikasi Data pada Proses Import

Salah satu permasalahan yang muncul selama proses integrasi data pada sistem Relogica adalah terjadinya duplikasi entri di beberapa tabel pendukung seperti `tools`, `task_preparation`, dan `materials`. Duplikasi ini umumnya

disebabkan oleh variasi penulisan data (misalnya perbedaan huruf besar/kecil atau adanya spasi tambahan), yang mengakibatkan entri dengan makna sama dianggap sebagai data berbeda oleh sistem basis data.

Pseudocode Duplicated Data Handler **??**.

```
# handle tools (dedup + lookup/insert)
tools_ids = []
for each tool_name in row.tools:
    clean_tool = trim_lower(tool_name)

    tool_id = lookup_or_insert(
        table = tools,
        key = clean_tool,
        company_id
    )

    add tools_id to tools_ids

# handle task preparation (dedup + lookup/insert)
prep_ids = []
for each prep_name in row.preparation:
    clean_prep = trim_lower(prep_name)

    prep_id = lookup_or_insert(
        table = task_preparation,
        key = clean_prep,
        company_id
    )

    add prep_id to prep_ids

# handle sparepart/materials (dedup + lookup/insert)
sparepart_ids = []
for each material in row.materials:
    clean_name = trim_lower(material.name)

    spare_id = lookup_or_insert(
        table = sparepart,
        key = clean_name,
        company_id
    )

```

```
    add (spare_id, maaterial.amount) to sparepart_ids
```

Kode 3.11: Pseudocode Duplicated Data Handler

Untuk mengatasi masalah tersebut, dilakukan penerapan mekanisme *lookup* sebelum proses penyisipan data (`INSERT`) ke dalam tabel. Mekanisme ini bekerja dengan cara melakukan pemeriksaan terlebih dahulu menggunakan *query* pencarian yang bersifat *case-insensitive* dan *whitespace-tolerant*. Dalam implementasinya, setiap nilai yang akan dimasukkan terlebih dahulu diubah menjadi huruf kecil dan dihilangkan spasi berlebih menggunakan fungsi `LOWER(TRIM(...))`.

Pseudocode Lookup or Insert **??**.

```

function lookup_or_insert(table, key, company_id):

    result = query("
        SELECT id FROM table
        WHERE lOWER(TRIM(name)) = key
        AND company_id = company_id
    ")

    if result exists:
        return result.id

    else:
        new_id = insert(
            table,
            name = key,
            company_id
        ) returning id

        return new_id

function trim_lower(value):
    if value is NULL:
        return ""
    return LOWER(TRIM(value))
```

Kode 3.12: Pseudocode Lookup or Insert

Sistem kemudian melakukan pengecekan terhadap keberadaan entri dengan nama yang sama pada tabel target berdasarkan kombinasi nilai `name` dan `company_id`. Jika hasil *query* menunjukkan bahwa entri tersebut sudah ada, maka sistem tidak akan melakukan proses penyisipan baru, melainkan menggunakan

`id` dari entri yang sudah ada. Sebaliknya, jika entri belum ditemukan, sistem akan melakukan `INSERT` dan mengembalikan `id` yang baru dibuat melalui perintah `RETURNING id` pada Gambar 3.16.

```
let name = input.trim().to_lowercase();
let existing: Option<i32> = sqlx::query_scalar("SELECT id FROM tools WHERE LOWER(TRIM(tools)) = $1 AND company_id = $
    .bind(&name).bind(company_id).fetch_optional(pool).await?;
let id = match existing {
    Some(id) => id,
    None => sqlx::query("INSERT INTO tools (tools, company_id) VALUES ($1,$2) RETURNING id")
                .bind(&name).bind(company_id).fetch_one(pool).await?.get::<i32,_>("id"),
};
```

Gambar 3.16. Code Snippet LOWER(TRIM()) Validation before Insert

Pendekatan ini tidak hanya mencegah duplikasi data yang tidak diinginkan, tetapi juga meningkatkan efisiensi penyimpanan dan konsistensi data di seluruh modul yang bergantung pada entitas tersebut. Dengan adanya validasi ini, data yang dimasukkan ke dalam tabel pendukung seperti `tools`, `task_preparation`, dan `materials` menjadi lebih bersih, terstandardisasi, dan bebas dari redundansi akibat variasi format input pengguna.

Secara keseluruhan, penerapan metode *case-insensitive lookup* dan *pre-insertion validation* ini memperkuat stabilitas integrasi data sistem AMS, memastikan bahwa setiap proses impor dapat berjalan dengan akurat tanpa mengorbankan integritas struktur basis data.

Pengujian performa dan kualitas *service* diuji menggunakan *Whitebox Testing* dan *Blackbox Testing* dan diuji oleh dua personel *Backend*.

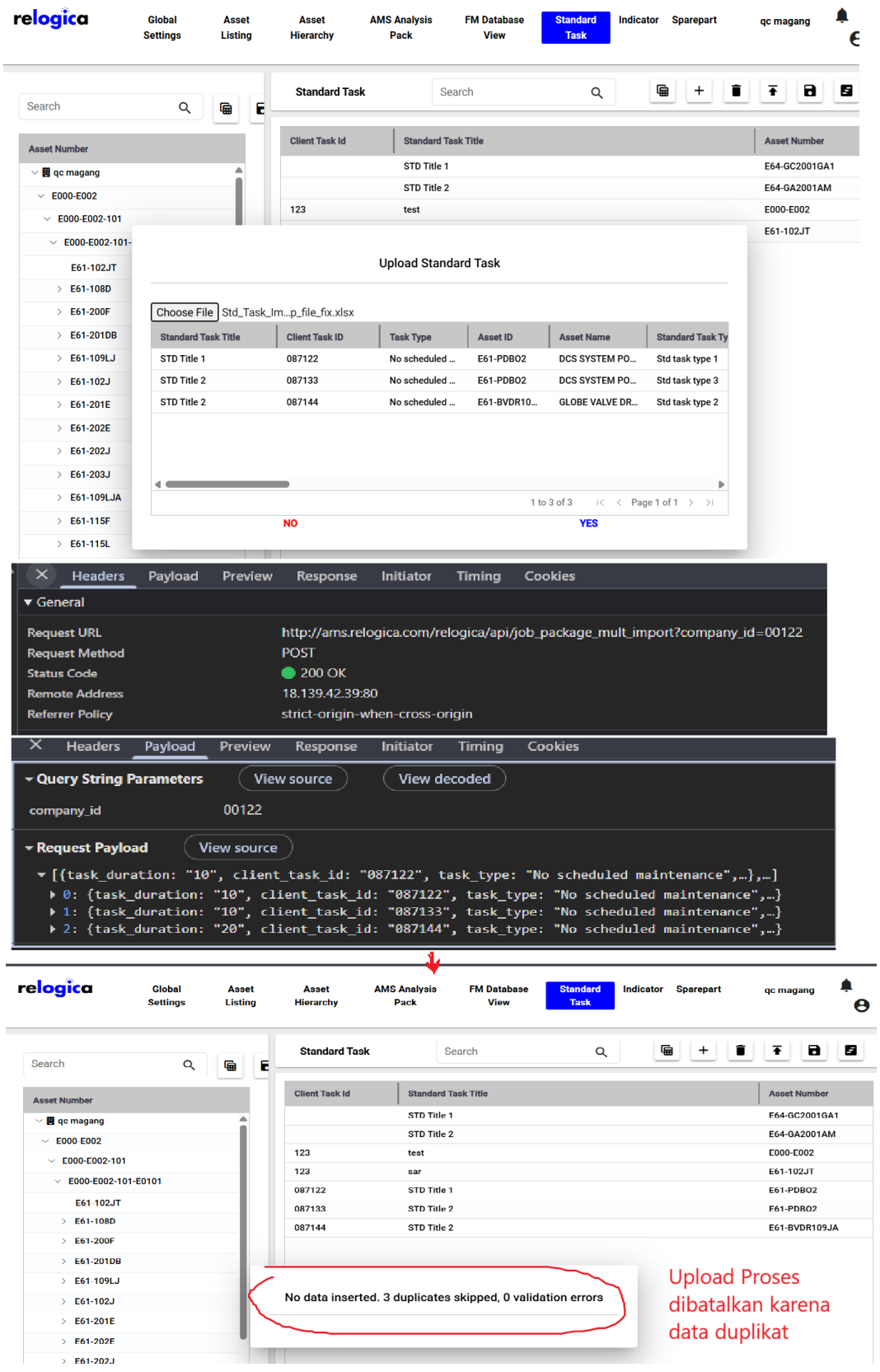
A Blackbox Testing

Pengujian dilakukan tanpa mengetahui detail implementasi kode, hanya berdasarkan input dan output yang dihasilkan. Pada proses import, dilakukan pengujian dengan berbagai variasi input (kapitalisasi berbeda, spasi tambahan, nama sama antar perusahaan, dsb) dan memeriksa apakah data yang dihasilkan bebas duplikasi.

Tabel 3.12. Tabel Pengujian Duplikasi Data pada Proses Import - *Blackbox*

No	Proses/Endpoint	Skenario Pengujian	Status
1	Import tools	Input “Obeng”, “obeng”, “ OBENG ”, hanya satu data tersimpan	Lulus
2	Import task_preparation	Input “Crane”, “crane”, “ crane ”, hanya satu data tersimpan	Lulus
3	Import materials	Input “Bolt”, “bolt”, “ bolt ”, hanya satu data tersimpan	Lulus
4	Import tools	Input nama sama pada company_id berbeda, data tersimpan terpisah	Lulus
5	Import task_preparation	Input nama sama pada company_id berbeda, data tersimpan terpisah	Lulus
6	Import materials	Input nama sama pada company_id berbeda, data tersimpan terpisah	Lulus
7	Import tools/task_preparation/materials	Input nama baru, data baru ditambah, id baru dikembalikan	Lusus
8	Import tools/task_preparation/materials	Input nama sudah ada, id lama dikembalikan, tidak ada data duplikat	Lulus
9	Import tools/task_preparation/materials	Input dengan spasi/kasus kapitalisasi berbeda, data tetap satu	Lulus
10	Import tools/task_preparation/materials	Input data kosong atau hanya spasi, response error validasi	Lulus

Pada Gambar 3.17. proses *duplicated data handling* terlihat berhasil dengan munculnya *alert box* yang menerangkan jumlah data duplikat dan diikuti oleh kegagalan masuknya data baru yang sama.



Gambar 3.17. Duped Data Validation - Job Package Upload

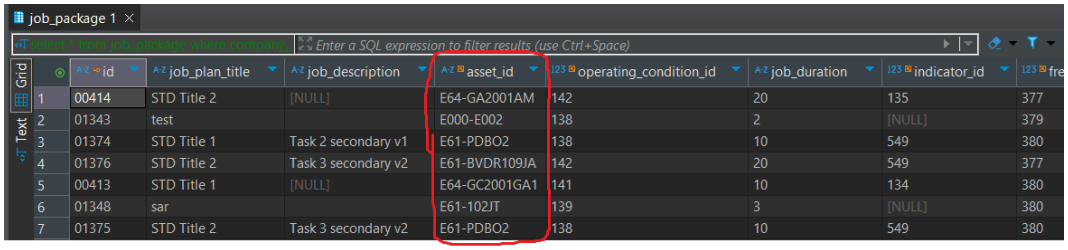
B Whitebox Testing

Pengujian dilakukan dengan mengetahui struktur internal kode, logika lookup sebelum insert, penggunaan query `LOWER(TRIM(...))`, serta validasi kombinasi `name` dan `company_id` pada proses import data.

Tabel 3.13. Tabel Pengujian Duplikasi Data pada Proses Import - *Whitebox*

No	Fungsi/Endpoint	Skenario Pengujian	Status
1	Import tools	Lookup dengan `LOWER(TRIM(tools))`, insert hanya jika belum ada, ambil id jika sudah ada	Lulus
2	Import task_preparation	Lookup dengan `LOWER(TRIM(preparation))`, insert hanya jika belum ada, ambil id jika sudah ada	Lulus
3	Import materials	Lookup dengan `LOWER(TRIM(material_name))`, insert hanya jika belum ada, ambil id jika sudah ada	Lulus
4	Import tools	Validasi kombinasi `name` dan `company_id`, mencegah duplikasi antar perusahaan	Lulus
5	Import task_preparation	Validasi kombinasi `name` dan `company_id`, mencegah duplikasi antar perusahaan	Lulus
6	Import materials	Validasi kombinasi `name` dan `company_id`, mencegah duplikasi antar perusahaan	Lulus
7	Import tools	Penanganan variasi kapitalisasi dan spasi, data “Obeng”, “obeng”, “ obeng ” dianggap sama	Lulus
8	Import task_preparation	Penanganan variasi kapitalisasi dan spasi, data “Crane”, “crane”, “ crane ” dianggap sama	Lulus
9	Import materials	Penanganan variasi kapitalisasi dan spasi, data “Bolt”, “bolt”, “ bolt ” dianggap sama	Lulus
10	Import tools/task_preparation/materials	Query lookup dan insert menggunakan `RETURNING id`, pastikan id yang digunakan konsisten	Lulus

Berdasarkan hasil pengujian dan observasi pada Gambar 3.18, dapat dilihat bahwa sistem berhasil menjaga konsistensi data pada tabel *job_package*, di mana tidak ditemukan entri dengan nilai *job_package* yang sama dalam satu *asset_id* yang identik. Hal ini menunjukkan bahwa mekanisme validasi dan pengendalian duplikasi yang diterapkan pada proses impor telah berjalan dengan baik, khususnya dalam memastikan bahwa setiap *job package* yang terasosiasi dengan suatu aset bersifat unik dan tidak redundan.



Tidak ada job_package yang sama (duplikat) dalam suatu asset yang sama

Gambar 3.18. Database View Duped Data Validation - Job Package Upload

Keberhasilan ini dicapai melalui penerapan logika pengecekan sebelum proses penyimpanan data ke basis data, sehingga sistem mampu mendeteksi keberadaan data yang sama berdasarkan kombinasi atribut kunci yang relevan. Dengan demikian, apabila ditemukan data dengan karakteristik yang identik pada *asset_id* yang sama, sistem akan menolak penyisipan data baru dan mempertahankan data yang sudah ada. Pendekatan ini tidak hanya mencegah terjadinya duplikasi data, tetapi juga menjaga integritas relasi antar tabel serta menghindari potensi inkonsistensi data pada proses analisis dan pelaporan aset.

Implementasi validasi ini berkontribusi langsung terhadap peningkatan kualitas data pada sistem Relogica AMS, terutama dalam konteks pengelolaan job package yang memiliki keterkaitan erat dengan proses penjadwalan, pemeliharaan, dan analisis risiko aset. Dengan data yang terstruktur dan bebas duplikasi, sistem mampu memberikan informasi yang lebih akurat dan andal bagi pengguna dalam mendukung pengambilan keputusan operasional.

3.4 Kendala dan Solusi yang Ditemukan

Kendala

Kendala yang ditemukan selama proses kerja magang di PT Braincode Digital Teknologi pada pengembangan proyek *Relogica Asset Management System (AMS)* adalah sebagai berikut:

1. Terjadi ketidaksinkronan antara *frontend* dan *backend* setelah proses *deployment*, terutama pada penyesuaian format payload dan struktur data yang digunakan antar layanan.

2. Munculnya bug yang bersifat berantai pada fitur tertentu, seperti perbaikan pada satu bagian kode terkadang menimbulkan masalah baru pada modul lain

yang saling terhubung.

Solusi

Beberapa solusi yang diterapkan untuk mengatasi kendala tersebut adalah sebagai berikut:

1. Melakukan diskusi dan *cross-check* secara menyeluruh antara tim *frontend* dan *backend* sebelum proses *deployment*, memastikan kesesuaian payload, penamaan atribut, serta membenahi kesalahan pengetikan (*typo*) yang dapat menyebabkan ketidaksesuaian data.
2. Berkolaborasi dengan rekan kerja dari tim terkait, baik *backend* maupun *frontend*, untuk menemukan akar permasalahan secara menyeluruh dengan melakukan proses *debugging*, *code review*, serta *testing* bertahap hingga bug dapat teratasi secara tuntas.