BAB 3
PELAKSANAAN KERJA MAGANG

Kegiatan magang ini dilaksanakan di PT Adicipta Inovasi Teknologi dengan penempatan sebagai *Quality Engineer Intern* pada Departemen *Software Testing*. Selama pelaksanaan magang dilakukan pengembangan serta eksekusi *automation testing* untuk menjaga dan memastikan kualitas perangkat lunak internal perusahaan agar sesuai dengan standar dan spesifikasi yang telah ditentukan.

Sepanjang periode kegiatan, dilaksanakan berbagai aktivitas pengujian otomatis, khususnya pada *track* kedua program magang. Pengujian difokuskan pada layanan aplikasi sisi *backend* melalui API. Seluruh proses pengujian dilakukan menggunakan *Katalon Studio* sebagai alat utama, dengan penerapan skenario uji yang merepresentasikan alur bisnis secara menyeluruh (*end-to-end*), termasuk verifikasi data pada *database*.

Sebagian informasi teknis terkait sistem internal maupun data perusahaan tidak dapat dijelaskan secara detail dalam laporan ini karena bersifat rahasia dan dilindungi oleh kebijakan internal perusahaan. Oleh karena itu, penyusunan laporan disajikan secara umum agar tetap menggambarkan pengalaman dan ruang lingkup pekerjaan selama magang tanpa melanggar ketentuan kerahasiaan yang berlaku.

3.1 Kedudukan dan Koordinasi

Sebagai *Quality Engineer* selama proses magang, penempatan dilakukan pada Departemen *Software Testing* dengan peran utama dalam memastikan kualitas perangkat lunak berfungsi sesuai dengan kebutuhan dan ketentuan yang ditetapkan.

Dalam pelaksanaannya, posisi ini berada dalam bimbingan langsung *Senior Automation Engineer* yang berperan memberikan arahan teknis serta melakukan penilaian terhadap tugas-tugas yang dikerjakan. Selain itu, kegiatan magang juga berada dalam pengawasan dan koordinasi *Section Head Quality Engineering* serta *Head of Software Testing Department*.

Koordinasi pekerjaan dilaksanakan melalui rapat mingguan (*weekly meeting*) menggunakan platform Microsoft Teams. Rapat ini bertujuan untuk menyampaikan pembaruan progres pekerjaan, mendiskusikan kendala yang dihadapi, serta menyusun rencana kerja dan menentukan prioritas tugas untuk periode selanjutnya.

3.2 Tugas yang Dilakukan

Selama pelaksanaan magang *track 2*, tanggung jawab utama yang dijalankan meliputi pengembangan lanjutan skrip *automation testing* yang telah dikembangkan pada periode pertama. Adapun tugas yang dilaksanakan adalah sebagai berikut:

1. Mengembangkan skrip berbahasa Groovy untuk *automation testing* melalui *platform* Katalon Studio yang difokuskan pada pengujian *core multifinance system* (CONFINS). Pengembangan ini meliputi penyusunan skenario uji otomatis yang merepresentasikan alur proses bisnis aplikasi secara nyata melalui pengujian API atau sisi *backend* secara *end-to-end*.

2. Secara langsung melaksanakan validasi data pada *database* menggunakan SQL sebagai bagian dari pengujian *backend* melalui PostgreSQL, khususnya pada pengujian layanan API. Proses validasi dilakukan untuk memastikan kesesuaian data yang diproses oleh API dengan data yang tersimpan di *database*, dari seluruh aspek seperti nilai, hasil perhitungan, serta keterkaitan antara entitas data.

3. Membangun skrip *automation testing* dengan fokus pada pengujian bersifat *end-to-end*, yang mencakup integrasi antar modul sistem, verifikasi hasil keluaran pada setiap tahap proses, serta pengecekan kesesuaian data antara API dan *database*. Pendekatan ini memastikan seluruh alur bisnis berjalan sesuai dengan ketentuan fungsional yang telah ditetapkan.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan magang dilakukan pada periode Juli 2025 hingga Februari 2025 dengan pembatasan cangkuman terakhir pada Desember 2025. Pada periode kedua ini, proses kegiatan magang mencakup lanjutan dari proyek utama dengan rincian pelaksanaan kerja magang pada Tabel 3.1.

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

Minggu Ke -	Pekerjaan yang dilakukan
1-4	Melanjutkan Pengerjaan Skrip Automation pada sistem CONFINS meliputi pembuatan API Master untuk digunakan pada flow e2e.
5-6	Membangun skrip pada format e2e mencakup pemanggilan test case, validasi database, dan validasi pada journal.
7-9	Melakukan pembangunan skrip e2e testing serta enhancement validasi journal, dan melakukan testing / running skrip automation testing.
10 - 11	Melakukan *enhancement* pada skrip *automation testing* yang sudah ada serta berkontribusi dalam beberapa proyek pengujian lain di bawah tim *Software Testing*, sambil menunggu dimulainya proses pengembangan lanjutan untuk sistem CONFINS LMS Konventional.
12 - 13	Mempersiapkan dan melanjutkan pengembangan lanjutan CONFINS LMS Konventional, pengembangan skrip master dan e2e tambahan untuk skenario baru.
14 - 20	Melanjutkan pengembangan lanjutan CONFINS LMS Konventional, pengembangan skrip e2e sekanrio baru dan testing untuk skrip tersebut, disertai proses raise dan retest issue yang ditemukan saat proses development.

3.3.1 Konsep dan Struktur Pengujian End-to-End pada Automation Testing

Software testing merupakan proses berkelanjutan dan iteratif yang dilakukan pada setiap tahap pengembangan perangkat lunak, mulai dari analisis kebutuhan, perancangan, pengembangan, rilis, hingga pemeliharaan. Tahapan ini memiliki peran penting dalam memastikan kualitas, keandalan, dan stabilitas sistem secara menyeluruh [5]. Dalam praktiknya, saat ini *automation testing* menjadi salah satu pendekatan yang banyak digunakan karena memungkinkan eksekusi kasus uji secara otomatis melalui skrip yang ditulis menggunakan bahasa pemrograman seperti Groovy, Python, Java, atau JavaScript. Pendekatan ini mampu mengurangi intervensi manual, mempercepat proses pengujian, serta meningkatkan konsistensi hasil uji [6].

A Automation Testing

Automation testing adalah metode untuk pengujian *software* yang memanfaatkan skrip dan (*testing tools*) untuk mengeksekusi kasus uji secara otomatis tanpa ketergantungan tinggi pada intervensi pengujian secara manual [7]. Pendekatan ini bertujuan untuk meningkatkan efisiensi, akurasi, serta konsistensi dalam proses pengujian, terutama pada skenario yang bersifat berulang atau memiliki kompleksitas tinggi sehingga dapat dijalankan secara regresi [8]. Dalam lingkungan pengembangan modern, automation testing menjadi krusial karena mampu mempercepat siklus pengujian, mendeteksi isu lebih dini, serta mendukung praktik pengembangan berkelanjutan seperti *Continuous Integration* dan *Continuous Delivery* (CI/CD). Pemanfaatan bahasa pemrograman dan framework pengujian memungkinkan otomatisasi untuk memodelkan alur proses bisnis secara terstruktur dan konsisten, sehingga kualitas perangkat lunak dapat dipertahankan pada setiap tahap pengembangan.

Pada konteks pengujian untuk sistem produk CONFINS, pada track kedua magang ini, menggunakan tools utama untuk proses development serta testing adalah sebagai berikut.

1. Katalon

   Katalon merupakan platform *automation testing* yang dimanfaatkan untuk melakukan pengujian terhadap aplikasi pada produk AdIns. Versi Katalon yang digunakan dalam kegiatan magang ini adalah versi 10.0.0. Platform ini merupakan antarmuka yang mendukung pengujian berbasis skrip, baik untuk API maupun UI.

2. PostgreSQL

   Sebagai sistem manajemen basis data relasional berbasis SQL, PostgreSQL digunakan oleh *software tester* dalam proses pengecekan dan verifikasi data. Perangkat ini berfungsi untuk menyimpan, mengelola, serta mengakses data menggunakan bahasa SQL. Dalam implementasi *automation testing*, PostgreSQL dimanfaatkan untuk memvalidasi data pada sisi *backend*, menjalankan *query* guna memastikan konsistensi data, serta mendukung penerapan pengujian berbasis data (*data-driven testing*) yang terintegrasi dengan Katalon.

3. Postman

   Merupakan *tools* yang digunakan untuk melakukan pengujian *Application Programming Interface (API)* secara manual. Postman berfungsi untuk memverifikasi akurasi payload sebelum diimplementasikan ke dalam skrip automation yang bersifat reusable. Pengujian menggunakan Postman umumnya dilakukan sebelum proses pengembangan test case master, untuk memastikan kesesuaian response yang akan divalidasi, baik untuk skenario positif maupun negatif.

4. VSCode

   Visual Studio Code (VSCode) dimanfaatkan sebagai alat bantu dalam penyelesaian *file* yang mengalami konflik saat proses *Pull Request (PR)*. Konflik umumnya muncul akibat perbedaan perubahan antara *branch* pengembangan dan *branch master*. VSCode secara otomatis menandai bagian kode yang bertentangan, termasuk perubahan dari *branch lokal* (*current change*) dan perubahan dari *branch* tujuan penggabungan (*incoming change*). Untuk menyelesaikan konflik tersebut, tersedia beberapa opsi seperti *Accept Current Change*, *Accept Incoming Change*, *Accept Both Changes*, atau melakukan pengeditan manual sesuai kebutuhan. *File* disimpan dan ditandai sebagai *resolved* saat konflik diselesaikan dan kemudian dilanjutkan dengan proses *commit* lalu *push*.

5. JIRA

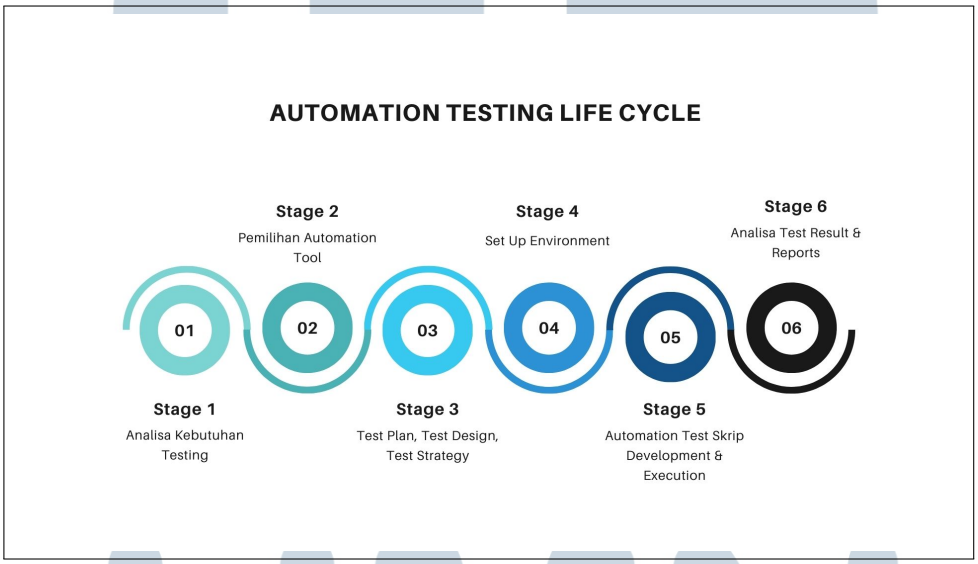
   Digunakan oleh tim *Quality Engineer* sebagai sarana perencanaan, pemantauan, dan pengelolaan tugas maupun laporan *bug* selama pelaksanaan *sprint* pengembangan perangkat lunak. Melalui JIRA, tim dapat mencatat *bug*, mengatur *backlog*, serta mengaitkan hasil pengujian dengan *ticket* yang relevan. Untuk mendukung manajemen pengujian, tim QE memanfaatkan *plugin* Zephyr Scale yang menyediakan fitur pembuatan *test case*, pelaksanaan *test run*, serta untuk melaporkan hasil pengujian Katalon yang sudah terintegrasi.

6. Bitbucket

   Bitbucket adalah platform *source code management* berbasis Git yang dikembangkan oleh Atlassian. Bitbucket digunakan untuk pengelolaan *repository*, manajemen *branch*, serta proses *pull request*, dan terintegrasi dengan *tools* lain seperti JIRA. Dalam kegiatan *testing*, Bitbucket berfungsi

sebagai repositori penyimpanan skrip *automation testing* sekaligus sebagai media pemantauan perubahan kode. Selain itu, Bitbucket digunakan oleh *Senior Automation Engineer* pada proses *code review* pada saat *pull request* diajukan, guna memastikan tidak terjadi konflik dengan *branch* lain sebelum skrip digabungkan ke dalam *branch master*.

Penggunaan tools berikut diterapkan dalam *automation testing life cycle (ATLC)* yang merupakan dasar dari proses automation testing yang digambarkan seperti berikut pada Gambar 3.1.



Gambar 3.1. Automation testing life cycle

Sumber: [9]

Tahapan *automation* dimulai dari analisa kebutuhan pengujian untuk mengidentifikasi modul, skenario, serta area aplikasi yang dibutuhkan dan cocok untuk diotomatisasi. Tahap ini memastikan bahwa upaya otomasi sesuai dengan tujuan kualitas dan prioritas dalam alur bisnis. Selanjutnya, tahapan kedua berfokus pada pemilihan *automation tool* yang paling sesuai, dengan mempertimbangkan aspek seperti kompatibilitas teknologi untuk aplikasi, dukungan integrasi, kemudahan pemeliharaan, serta efisiensi eksekusi. Sesuai dengan penjelasan *tools* yang digunakan di atas, produk yang dipakai adalah *Katalon*.

Pada tahap ketiga, dilakukan penyusunan *test plan*, desain skenario uji, serta strategi otomasi yang menentukan cakupan pengujian, struktur skrip, dan pendekatan validasi yang akan digunakan. Selanjutnya, pada tahap keempat mencakup *environment setup*, yaitu proses menyiapkan lingkungan pengujian otomatis, termasuk persiapan *server* yang akan digunakan, pengaturan format data, integrasi dengan sistem pendukung, serta penyiapan *test repository*. Setelah lingkungan siap, tahapan kelima melibatkan pembangunan skrip pengujian otomatis (*automation test script development*) serta eksekusi awal untuk memastikan bahwa skrip berjalan stabil dan sesuai harapan. Tahapan ini biasanya mencakup pembuatan fungsi *reusable*, struktur kontrol, serta validasi terhadap *response*, *database*, dan komponen pendukung lainnya.

Terakhir, pada tahap keenam dilakukan analisis hasil pengujian dalam format *test suite report*. Pada tahap ini dilakukan pemeriksaan mendalam terhadap *log* eksekusi, kegagalan yang terjadi, serta proses yang berhasil. Keseluruhan tahapan *ATLC* tersebut memastikan bahwa proses otomasi berjalan secara terukur, terstandarisasi, serta mampu memberikan kontribusi langsung terhadap peningkatan kualitas perangkat lunak. Selain meningkatkan efisiensi pengujian, *ATLC* juga menjadi fondasi penting untuk memastikan bahwa setiap komponen otomatisasi dapat saling terintegrasi dengan baik. Hal ini sangat krusial sebagai dasar dalam membangun pengujian *End-to-End*, yang akan dibahas lebih lanjut pada subbab berikutnya.

B Konsep End-to-End Testing

Dalam *automation testing*, penerapan *end-to-end (E2E) testing* merupakan pendekatan pengujian yang memverifikasi bahwa seluruh alur bisnis aplikasi berjalan sesuai harapan dari sisi *user*. Dalam konteks produk yang diuji, pengujian dimulai dari pemanggilan *API* hingga verifikasi pemrosesan data di sisi *backend* dari seluruh titik *endpoint* [10]. Dalam konteks *automation testing*, *E2E testing* sangat penting karena menguji integrasi penuh antar komponen sistem, termasuk interaksi antar layanan, alur otentikasi, pemrosesan data, serta penyimpanan di basis data untuk memastikan sistem sebagai satu kesatuan berfungsi dengan benar.

E2E testing juga berperan utama dalam mengidentifikasi isu atau *bug* integrasi yang mungkin tidak terdeteksi dalam pengujian *unit* atau dalam konteks *development*—pada penelitian ini berada pada level *test case master*—seperti kesalahan alur bisnis, masalah dependensi antar layanan, atau inkonsistensi data.

Selain itu, *E2E testing* mendukung praktik *continuous testing* atau *regression testing* dalam *pipeline CI/CD*, karena memberikan dasar pengujian bahwa alur aplikasi yang kompleks tetap valid saat dilakukan *deployment*.

Struktur *E2E test case* biasanya mencakup skenario *end-to-end* yang mencerminkan proses nyata pengguna dalam produk CONFINS. Pengujian dimulai dari awal penggunaan API yang membentuk data hingga validasi jurnal keuangan. Dalam kerangka otomatisasi, *E2E test suite* bersifat lebih menyeluruh dibandingkan *functional test*, dengan format yang harus diorganisasi dengan baik agar mudah dipelihara, mudah dijalankan kembali, serta mampu memberikan cakupan lengkap terhadap fitur utama dalam aplikasi. Berikut merupakan perbandingan *E2E test* dan *functional test* berdasarkan Katalon.

Aspek	**Pengujian Fungsional**	**Pengujian End-to-End**
Lingkup	Pengujian terbatas pada satu bagian kode atau satu aplikasi.	Pengujian mencakup banyak aplikasi dan kelompok pengguna.
Tujuan	Memastikan perangkat lunak memenuhi kriteria penerimaan.	Memastikan proses tetap berjalan setelah adanya perubahan.
Metode Pengujian	Menguji bagaimana satu pengguna berinteraksi dengan aplikasi.	Menguji bagaimana beberapa pengguna bekerja lintas aplikasi.
Yang Divalidasi	Validasi hasil dari setiap pengujian berdasarkan input dan output.	Validasi bahwa setiap langkah dalam proses telah diselesaikan.

Tabel 3.2. Perbandingan Pengujian Fungsional dan Pengujian End-to-End

Sumber: [11]

Struktur pengujian perangkat lunak umumnya digambarkan melalui *test automation pyramid*, yang membagi jenis pengujian berdasarkan cakupan, tujuan, serta tingkatannya. Setiap lapisan memiliki peran yang berbeda dalam memastikan kualitas sistem secara menyeluruh. Secara umum, pengujian dapat dikelompokkan menjadi beberapa kategori sebagai berikut:

1. ***Unit Tests***
   Pengujian unit/*master* berfokus pada komponen individu, seperti fungsi atau metode, untuk memastikan bahwa setiap bagian kecil dari sistem bekerja dengan benar secara terisolasi. Contohnya adalah pengujian terhadap satu *API* yang berdiri sendiri tanpa ketergantungan pada modul lain.

2. ***Integration Tests***

   Setelah komponen individual berfungsi dengan baik, langkah berikutnya adalah menguji bagaimana komponen-komponen tersebut bekerja ketika digabungkan. Komponen yang telah lulus *unit test* tetap dapat mengalami kegagalan saat diintegrasikan, biasanya akibat kesalahan komunikasi data. *Integration test* memastikan aliran data antar modul berjalan dengan baik serta antarmuka antar komponen berfungsi sebagaimana mestinya.

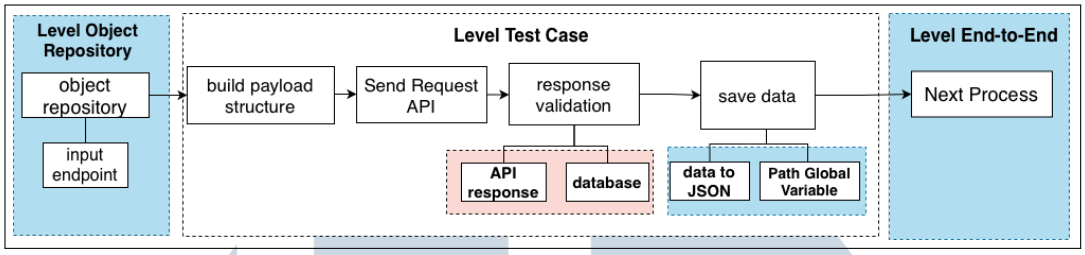
3. ***End-to-End (E2E) Tests***

   End-to-end test memvalidasi satu alur secara keseluruhan atau skenario bisnis aplikasi, mulai dari *API* untuk antarmuka pengguna hingga proses di sisi *back-end*, untuk memastikan sistem bekerja sebagai satu kesatuan. Pengujian ini memberikan keyakinan tinggi terhadap pemenuhan kebutuhan bisnis, tetapi membutuhkan waktu lebih lama dan bersifat kompleks. Oleh karena itu, jumlah *E2E test* biasanya dibatasi pada alur yang benar-benar kritikal dan umum digunakan.

C Transisi Master Test Case ke End-to-End

Master test case atau *unit test* merupakan pengujian komponen secara individu yang berfokus pada satu fungsi, contohnya seperti *API* untuk penambahan *user* yang berdiri sendiri tanpa bantuan *API* lain untuk menjalankan prosesnya. Transisi dari *master test case* menuju pengujian *end-to-end* dilakukan ketika beberapa *API* atau layanan telah saling terhubung untuk membentuk satu alur proses bisnis yang utuh. Pada tahap ini, setiap unit yang sebelumnya diuji secara terpisah mulai dikombinasikan dan dijalankan dalam urutan yang menyerupai kondisi nyata di sistem. Pengujian *end-to-end* bertujuan untuk memvalidasi bahwa seluruh rangkaian proses—mulai dari *input* awal, pemanggilan *API* secara berantai, pengolahan data, hingga penyimpanan ke *database*—berjalan secara konsisten dan menghasilkan *output* yang sesuai.

Selain itu, penggunaan *global variable*, *dynamic variables*, serta penyimpanan *response* antar *step* menjadi bagian penting dalam membangun skenario *E2E* secara efisien. Hal ini memastikan bahwa setiap *API* yang saling bergantung dapat dijalankan dalam alur yang konsisten tanpa memerlukan *input* manual. Dalam penerapan ini, penyimpanan *response* dilakukan ke dalam file *JSON*. Berikut merupakan proses penyimpanan data untuk integrasi *End-to-End*.



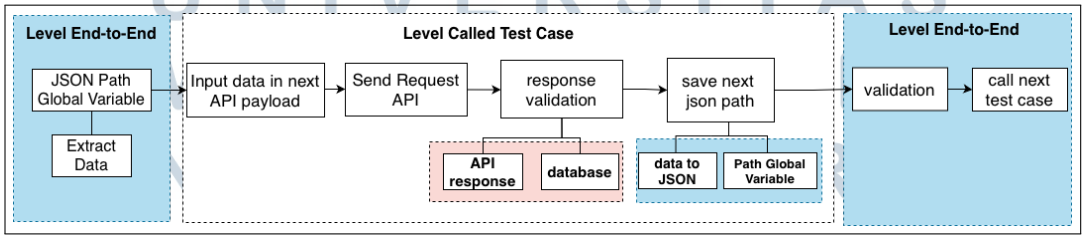
Gambar 3.2. Alur pembangunan skrip API sampai penyimpanan data

Pada Gambar 3.2, alur pengujian dimulai dari *Object Repository* yang berisi *endpoint* API yang disimpan dan dikelola. Pada tahap ini, *endpoint* API diinput bersama dengan *global variable* untuk pengaturan *header* yang berisi *API key* dan *base URL* yang akan digunakan sebagai titik awal proses pengujian, termasuk *payload* yang bersifat dinamis dan dapat diisi di *test case* sesuai kebutuhan.

Selanjutnya, proses pengujian berlanjut ke tahap pembangunan struktur *payload* pada level *test case* sesuai kebutuhan *API*, termasuk parameter yang bersifat *mandatory* serta variabel *default* yang akan digunakan untuk pengujian awal sebelum disusun menjadi *end-to-end*. Setelah *payload* tersusun, *request API* dikirim menggunakan *endpoint* dengan data yang telah disiapkan.

Kemudian, proses masuk ke tahap validasi *response*, di mana hasil *response API* diverifikasi untuk memastikan kesesuaiannya dengan ekspektasi, baik dari sisi *status code* maupun isi *response*. Pada saat yang sama, dilakukan pula validasi ke *database* untuk memastikan data yang masuk sesuai dengan hasil pemrosesan *API* dan benar-benar tersimpan di sistem *backend*.

Dari hasil *response* yang berhasil divalidasi, proses berlanjut ke tahap penyimpanan data ke dalam file *JSON* dengan *timestamp* tertentu. *Path* file tersebut kemudian disimpan ke dalam *global variable* untuk digunakan kembali pada proses berikutnya dalam rangkaian skenario *end-to-end*. Berikut merupakan gambaran alur penggunaan *path* dalam *global variable* untuk menghubungkan data antar alur pada struktur *End-to-End*.



Gambar 3.3. Tahapan selanjutnya untuk menyambungkan antar data di End-to-End

Alur pada proses Gambar 3.3 ini menggambarkan bagaimana setiap *called*

test case terintegrasi di dalam skenario *end-to-end* sehingga membentuk rangkaian pengujian yang utuh dan berkesinambungan. Pada level *end-to-end*, data dari *JSON* proses sebelumnya atau *test case* yang pertama dijalankan akan digunakan kembali melalui *JSON Path Global Variable* yang telah disimpan dari proses sebelumnya. Data ini kemudian dimasukkan sebagai *input* ke dalam *payload API* berikutnya untuk memastikan setiap layanan menerima nilai yang relevan dan konsisten, contohnya seperti nomor konsumen atau nomor dokumen yang dibuat dari hasil *response API* sebelumnya.

Dari data tersebut, proses berpindah ke level *called test case* selanjutnya sesuai kebutuhan, yaitu bagian di mana permintaan *API* dikirimkan dan responsnya divalidasi, baik melalui pengecekan nilai pada *API response* maupun verifikasi terhadap data di *database*. Jika seluruh validasi berhasil, informasi penting akan disimpan kembali sesuai kebutuhan dalam format *JSON* serta diperbarui ke dalam *Global Variable Path* agar dapat digunakan oleh proses selanjutnya.

Proses ini berjalan secara berulang sesuai kebutuhan skenario dan jumlah *test case* yang harus dipanggil dalam satu rangkaian *end-to-end*. Pada file level *end-to-end*, dilakukan juga validasi lanjutan seperti pengecekan *journal* maupun pengambilan data tambahan dari *database* untuk memastikan integritas proses. Setelah seluruh validasi pada tahap tersebut selesai, sistem kemudian memanggil *test case* berikutnya dan mengulangi pola yang sama. Siklus ini terus berlanjut hingga seluruh rangkaian proses bisnis berhasil diverifikasi, sehingga memastikan bahwa setiap langkah berjalan konsisten dan sesuai harapan dari awal hingga akhir skenario *end-to-end*.

3.3.2 Development End-to-End API Automation Testing pada Produk Core Multifinance System (CONFINS)

Dalam penerapan produk CONFINS, konsep *End-to-End* bertujuan untuk melakukan validasi berdasarkan skenario yang dibentuk. Skenario tersebut mencakup pemanggilan *test case master API*, validasi data, proses *end of day (EOD)*, serta validasi ke dalam data *journal engine*. Proses pengujian ini berfokus pada dua level utama, yaitu pembangunan *test case master* serta pembangunan alur untuk *end-to-end*.

A API Master Level Test Case

Dalam master test case, setiap API yang dibutuhkan dari suatu modul disusun dalam folder masing-masing. Setiap API umumnya memiliki *positive case* dan *negative case*, serta dirancang bersifat dinamis pada bagian input agar dapat digunakan berulang kali dalam berbagai kebutuhan skenario end-to-end. Struktur ini memungkinkan respons dan data yang diambil untuk disimpan ke dalam global variable secara efisien dan terstandarisasi, sehingga setiap elemen dapat digunakan kembali (*reusable*) tanpa perlu membangun ulang logika atau payload pada setiap test case.

1. **Positive Test Case**

   Positive test case merupakan skenario pengujian yang digunakan untuk memastikan bahwa sistem berfungsi sesuai harapan ketika diberikan *input* yang valid. Dalam konteks *end-to-end*, tipe *test case* ini lebih sering digunakan karena menghasilkan data yang dapat dimanfaatkan kembali pada proses bisnis berikutnya. Variasi yang digunakan biasanya bergantung pada jenis data yang diperlukan oleh *API* dan umumnya menghasilkan respons `200 OK` sebagai indikasi bahwa proses berhasil dijalankan. Berikut adalah contoh *pseudocode* pada Kode 3.1 untuk *case* positif.

```
1
2  TEST CASE: Payment Process
3  DEPENDENCIES:
4      - Call Master Test Case: Receipt From Regist
5      - Call Master Test Case: Receipt From Proses
6      - Utility Classes: DB, DateUtils, CommonAction,
       CheckStatus, ValidateSchemaJson
7
8  BEGIN
9      LOAD required global variables and request payloads
10     EXTRACT important fields (NoXX, ReceiptFormNo, Date)
11
12     PREPARE dynamic values:
13         - Generate No
14         - Determine Request Date
15         - Build allocation list (only include non-empty
       fields)
16         - Set auto when allocation list is empty
17
18     SEND Payment API REQUEST with all parameters
19
20     VERIFY response.statusCode == 200
21
22     IF response is valid THEN
23         PARSE response
24         ASSERT response header fields are correct
25
26         WAIT until payment status is processed
27         VALIDATE Prepaid Matching when allocation code
       requires it
28
```

```
        QUERY database for payment transaction info
        ASSERT critical fields match the input
        dbData = data from database

        assert dbData.trxNo == trxNo
        assert dbData.payNo == json.PayNo
        assert dbData.receive == RcvAmt

        VALIDATE allocation amount logic (scenario based)

        SAVE final data into JSON file
        finalData = {
        trxNo, dbData.payCode, trxAmt,
        RefallocCode, allocAmt, voucherNo
    }

    filePath = "JSON/payXX_Code.json"
    jsonTool.save(finalData, filePath)
    GlobalVariable.payment_XX = filePath
        STORE file path into Global Variables
    ELSE
        LOG "API Payment Proses Failed"
    ENDIF
END
```

Kode 3.1: Pseudocode API positif proses payment

Pseudocode di atas menggambarkan alur utama proses *master* dalam skenario *end-to-end* yang paling sering digunakan, yaitu proses *payment*, dimulai dari pemanggilan *test case* untuk pembuatan nomor transaksi. Setelah itu, sistem melakukan inisialisasi berbagai utilitas dan kelas *helper* seperti *database handler*, *date utility*, *action utility*, hingga *JSON parser* untuk memuat data dari *global variable* yang dihasilkan oleh *test case* sebelumnya. Data penting seperti *payment number*, *value date*, *reference number*, dan nomor formulir penerimaan kemudian diekstraksi dan dipersiapkan sebagai parameter *request* sesuai yang nantinya dipanggil di *end-to-end*.

Selanjutnya, sistem membangun struktur *payload* secara dinamis melalui *mapping* yang hanya memasukkan *field* yang memiliki nilai valid. *Payload* tersebut kemudian dikirimkan ke *API* untuk *payment*, dan respons yang diterima diverifikasi untuk memastikan status 200 *Success*. Jika respons valid, data *JSON* diparsing untuk mendapatkan nomor transaksi pembayaran, yang kemudian digunakan untuk proses verifikasi lanjutan ke *database*.

Pada tahap akhir, hasil akhir transaksi dikumpulkan ke dalam struktur data, disimpan dalam file *JSON* untuk keperluan dokumentasi atau kebutuhan *test case* berikutnya, dan kemudian disimpan sebagai *global variable*. Alur ini merupakan alur positif pada level *unit case* untuk digunakan nantinya dalam karakteristik pengujian *end-to-end*, di mana satu proses bisnis tidak hanya menguji *API*, tetapi juga memastikan integritas data antar *modul*, validasi

database, hingga menghasilkan *output* yang dapat digunakan kembali pada tahapan selanjutnya.

2. **Negative Test Case**

   Negative test case digunakan untuk menguji bagaimana sistem menangani *input* yang tidak valid, kondisi *error*, atau skenario yang tidak sesuai dengan aturan bisnis. Pengujian ini memastikan bahwa sistem mampu memberikan respons *error* yang tepat sesuai dengan apa yang tidak sesuai ketentuan *payload*, serta menampilkan pesan kesalahan yang informatif. Meskipun tidak selalu digunakan dalam alur *end-to-end*, *negative test case* sangat penting untuk memverifikasi ketahanan sistem dan memastikan bahwa mekanisme validasi telah diterapkan dengan benar, dan jika digunakan dalam *end-to-end* yang diperlukan adalah validasi *input* yang salah sebelum melakukan validasi kasus positifnya.

```
1
2  TEST CASE: Payment Transaction dengan Empty Mandatory Field
3  BEGIN
4      PREPARE request payload for "PayProses"
5          SET ValDt = empty
6          SET other parameters = valid values
7          INCLUDE mandatory allocation list
8
9      SEND request to Payment API
10
11     VERIFY response.statusCode == 200
12
13     IF response is received THEN
14         PARSE response JSON
15
16         ASSERT HeaderObj.StatusCode == 400
17         ASSERT HeaderObj.Message == "Validation Failed"
18         ASSERT HeaderObj.ErrorMessages is not empty
19
20         EXTRACT first error detail
21         ASSERT error.Field == "ValDt"
22         ASSERT error.Message indicates invalid or null
       mandatory input
23
24         LOG "Negative case validated: Mandatory Field empty
       triggers correct validation error"
25     ELSE
26         LOG "Unexpected response: No response received"
27     ENDIF
28 END
```

Kode 3.2: Pseudocode kasus negatif mandatory field kosong

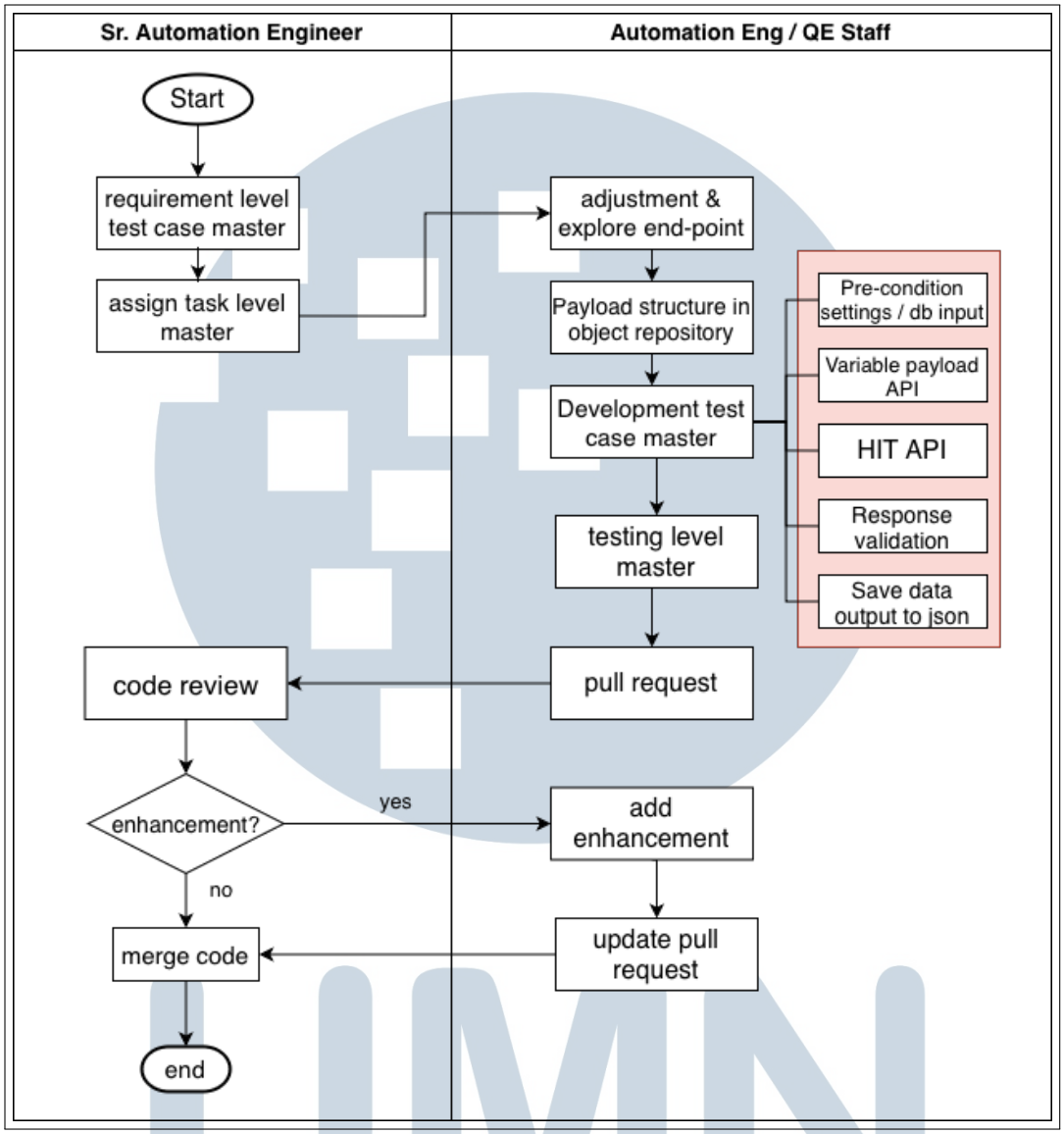
Pada Kode 3.2, skenario *negative test case* ini menguji *API* untuk memastikan bahwa validasi berjalan dengan benar ketika terdapat *mandatory field* yang tidak diisi pada saat pengiriman *request*. Ketika *API* dipanggil dengan parameter yang tidak lengkap, server tetap merespons permintaan tersebut,

namun hasil verifikasi menunjukkan bahwa status yang diterima adalah *Validation Failed*. Respons ini berisi informasi yang menjelaskan bahwa terdapat kolom wajib yang tidak diberikan nilainya, sehingga proses tidak dapat dilanjutkan.

Pendekatan ini digunakan pada kasus *end-to-end* apabila skenario yang diberikan mencakup validasi kasus negatif. Selain itu, *negative test case* umumnya juga digunakan untuk keperluan *regression testing* maupun pada level *unit testing*.

Master test case, yang terdiri dari skenario positif maupun negatif, merupakan tahapan pembangunan skenario *end-to-end*. Seluruh skenario ini dikembangkan terlebih dahulu agar setiap alur API dapat tervalidasi secara individual. Setelah rangkaian skenario pada level *master* selesai dibangun, proses integrasi kemudian dilakukan pada *test case end-to-end* dengan cara memanggil atau menggabungkan *test case* tersebut ke dalam satu alur pengujian yang utuh.

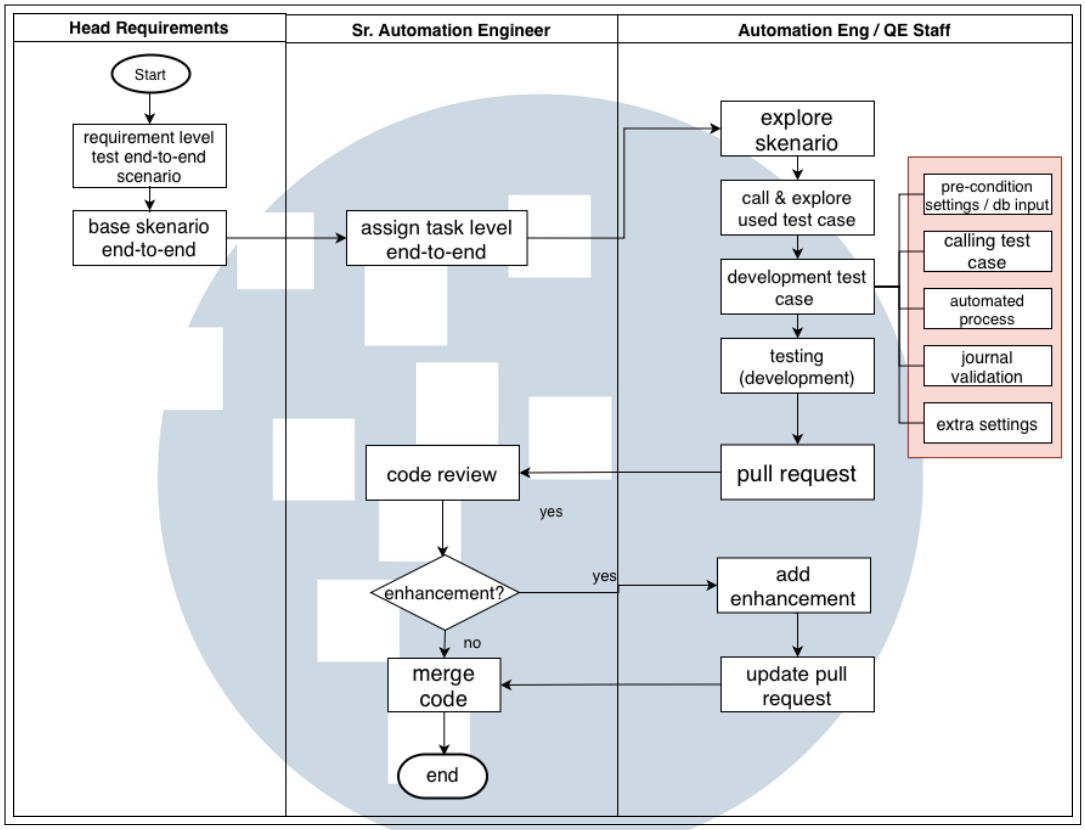
Penyusunan *master test case* ini juga mengikuti alur kerja yang terstruktur, dimulai dari tahap analisis dan desain oleh *Senior Automation Engineer* hingga tahap implementasi dan eksekusi oleh *QE Staff*. Dengan adanya proses yang sistematis ini, setiap komponen dapat diuji secara konsisten serta siap digunakan kembali pada level pengujian selanjutnya. Alur pada Gambar 3.4 berikut memberikan gambaran mengenai proses pembangunan *test case* pada level *master* sebelum digunakan dalam skenario *end-to-end*.



Gambar 3.4. Proses pembangunan test case master pada automasi testing CONFINS

B End-to-End Level Test Case

Setelah *test case master* sudah terbangun sesuai kebutuhan yang sudah diterapkan, melalui *test case* tersebut dibangunlah alur *end-to-end* sesuai kebutuhan alur bisnis tertentu. Proses dari pembangunan *case end-to-end* ditampilkan pada Gambar 3.5 sebagai berikut.



Gambar 3.5. Proses pembangunan test case end-to-end pada automasi testing CONFINS

Alur pada Gambar 3.5 di atas menggambarkan proses kerja antara tiga peran utama dalam proses pembangunan *end-to-end (e2e)*, dimulai dari *requirements* atau *Head Requirements*, kemudian *Senior Automation Engineer* yang ditulis sebagai *Sr. Automation Engineer* pada gambar, serta *Automation Engineer/QE Staff*. Proses dimulai dari sisi *Head Requirements* yang bertanggung jawab dalam menyusun *requirement-level test end-to-end scenario*. Skenario ini berfungsi sebagai dasar dalam pembuatan *base scenario* yang selanjutnya menjadi acuan bagi tim *Automation* dalam membangun *test case*.

Setelah kebutuhan dan skenario dasar selesai disusun, *Sr. Automation Engineer* akan membuat tiket *task test case end-to-end* berdasarkan *requirement* tersebut. Pada tahap ini, *Sr. Automation Engineer* melakukan *task assignment* serta memastikan bahwa arah pengembangan *test case* telah sesuai dengan kebutuhan yang ditetapkan. Pada sisi *Automation Engineer/QE Staff*, proses dimulai dari memahami alur bisnis dan *requirement* yang telah diberikan. Selanjutnya dilakukan pembangunan serta eksplorasi *existing test case* yang akan digunakan atau digabungkan untuk membentuk satu alur skenario secara utuh.

Secara lebih rinci, di dalam satu *end-to-end case* terdapat berbagai aspek

atau struktur isi yang harus diperhatikan sesuai dengan Gambar 3.5. Setiap aspek tersebut membentuk alur pengujian yang memastikan bahwa proses bisnis dapat berjalan dari awal hingga akhir tanpa hambatan.

1. Pre-condition dan pengaturan DB input
   Melakukan inisialisasi data awal, setup database, dan konfigurasi prasyarat sebelum skenario dijalankan yang ditampilkan pada Kode 3.3.

```
BEGIN
    INITIALIZE helper objects
        CREATE jsonAction AS ValidateSchemaJson
        CREATE checkStat AS CheckStatus
        CREATE db connection
        CREATE dbCheck AS Check
        CREATE dateUtils AS DateUtils

    PREPARE business date
        CALL db.getBDate() -> businessDMap
        CONVERT businessDMap INTO formatted bDate
            USING DateUtils.convertToBDt()

    LOG "Initialization completed: All helper
   components loaded."

END
```

Kode 3.3: Pseudocode Tahap Inisialisasi Komponen Test E2E

2. Pemanggilan test case
   Kode 3.4 adalah eksekusi test case level unit master sebagai bagian dari rangkaian pengujian.

```
BEGIN

    CALL TestCase "Activation Process"
        PARAMS = {}
        FAILURE_HANDLING = STOP_ON_FAILURE

    CALL TestCase "Submit NonTaXX"
        PARAMS = {}
        FAILURE_HANDLING = STOP_ON_FAILURE

    CALL TestCase "Request Deactivate"
        PARAMS:
            ProcessNo      = ProcessNo
            requestDtTime = businessDate
        FAILURE_HANDLING = STOP_ON_FAILURE

    CALL TestCase "Recov - EndDeal Total"
```

```
        PARAMS:
            dealxx = dealxx
        FAILURE_HANDLING = STOP_ON_FAILURE

    CALL TestCase "Recov"
        PARAMS:
            ProcessNo      = ProcessNo
            requestDtTime = businessDate
        FAILURE_HANDLING = STOP_ON_FAILURE

    CALL TestCase "Update Contract Status to EXP"
        PARAMS:
            AgrmntNo = ""
            Code     = GlobalVariable.Key
        FAILURE_HANDLING = STOP_ON_FAILURE

END
```

Kode 3.4: Pseudocode contoh tahap pemanggilan test case E2E

3. Pelaksanaan *Automated Process* (jika diperlukan)
   Menjalankan proses siklus automatisasi sesuai kebutuhan logika bisnis pada skenario tertentu. Proses pada Kode 3.5 ini adalah proses otomasi secara API sesuai kebutuhan skenario. Berikut adalah contoh cara pemanggilannya.

```

BEGIN
    # Menunggu proses berjalan
    CALL checkStat.waitIsProcesRun("1")

    # Menunggu proses selesai
    CALL checkStat.waitIsProcessRun("0")

    # Memanggil Test Case untuk API Otomatisasi
    CALL TestCase "API otomatisasi proses"
        PARAMETERS = {}
        FAILURE_HANDLING = STOP_ON_FAILURE

    # Memanggil Test Case untuk API lanjutan
   otomastisasi proses
    CALL TestCase "API otomatisasi proses"
        PARAMETERS = {}
        FAILURE_HANDLING = STOP_ON_FAILURE

END
```

Kode 3.5: Pseudocode pemanggilan automated process

4. Validasi komponen terkait
   Memverifikasi hasil proses pada sisi proses, transaksi, atau modul pendukung

lainnya untuk memastikan keakuratan data. Berikut adalah contoh format validasi jurnal pada Kode 3.6.

```
BEGIN
    LOAD json data from GlobalVariable.payment
        EXTRACT PaXX
        EXTRACT RcvXX and AllocXX as decimal values

    WAIT until validation appears for PayRcvXX

    RETRIEVE actual validation rows by PayRcvXX

    COMPUTE valueDate and postingDate using
   business date

    PREPARE expected journal rows:
        ROW 1:
            payAllocCode = "-"
            amount = RcvXX
            DebOrCred = "DEBET"
            officeCode = Global.offCode
            postingDate = computed postDate
            valueDate = computed valueDate
            originCurrCode = ""

        ROW 2:
            payAllocCode = "PRE_XX"
            amount = AllocationXX
            DebOrCred = "CREDIT"
            officeCode = Global.offCode
            postingDate = computed postDate
            valueDate = computed valueDate
            originCurrCode = ""

    ASSERT actual rows match expected rows

END
```

Kode 3.6: Pseudocode validasi proses dari hasil API payment

5. Penambahan konfigurasi atau settings tambahan
   Menambahkan konfigurasi atau pengaturan tambahan apabila dibutuhkan untuk penyelesaian skenario pengujian. Proses ini biasanya berupa pengaturan tambahan selama proses testing berlangsung seperti pada Kode 3.7.

```

BEGIN
    LOG "Initialize additional configuration
   settings"

```

```
    // Setting 1: Update Option
    CALL db.updateSetting("X", "XX_XX")
        EXPECT configuration updated successfully

    // Setting 2: Disable Auto Allocation Feature
    CALL db.updateSetting("0", "AUTO_XX")
        EXPECT flag correctly stored in system

    // Setting 3: Set Maximum Retry for Automation
   Process
    CALL db.updateSetting("1", "MAX_RETRY")
        VERIFY configuration applies

    LOG "All additional settings applied
   successfully"

END
```

Kode 3.7: Pseudocode tambahan pengaturan konfigurasi

Setelah test case dibuat, dilakukan proses testing dalam tahap development untuk memastikan script berjalan sesuai desain. Jika hasilnya sudah valid, engineer membuat pull request untuk dilakukan review dan integrasi. Bila ditemukan kebutuhan perbaikan atau optimasi, dilakukan proses enhancement sebelum disetujui untuk masuk ke tahap selanjutnya. Berikut adalah contoh salah satu test case end-to-end secara lengkap pada Kode 3.8 dari generate data hingga tahap akhir agreement.

```

TEST CASE: End-to-End Financial Flow

BEGIN

    INITIALIZE helper classes:
        jsonAction = ValidateSchemaJson
        checkStat = CheckStatus
        db = db
        assertJournal = JournalAssertion
        dbJournal = Journal
        dateUtils = DateUtils

    FETCH business date from DB
    CONVERT business date into required format

    CALL TestCase "Activation Data"
        params = {}

    APPLY additional system settings
        db.updateGeneralSetting("x", "xx_XX")
```

```
22
23    RUN Automated multiple cycles:
24        WAIT until process starts (flag = '1')
25        WAIT until process ends   (flag = '0')
26
27
28    LOAD number activation result from Global Variable
29    EXTRACT UniqueNo
30
31    WAIT until UniqueNo data appears in mirror table
32
33    FETCH recognized amounts
34    CALCULATE:
35        xxAfter  = xxInterest - totalRecog
36        xxGross  = xxInterest + xxPrincipal
37
38    CALL TestCase "Submit Non Process"
39
40    LOAD NonAccrual response
41    EXTRACT nonAccrualXX
42
43    WAIT for validation posting for nonACNo
44
45    FETCH actual validation rows
46    PREPARE expected journal rows (ARXX, ARX, UCI,
    OTHER)
47    ASSERT actual vs expected using assertJournal
48
49    CALL TestCase "Request Deactivation"
50        params = { UniqueNo, requestDtTime =
    businessDate }
51
52    LOAD cutOff response
53    EXTRACT Trx
54
55    WAIT for posting for Trx
56
57    FETCH DB cut off amounts (Amt, xxAmt, xxxAmt)
58    PREPARE expected journal rows
59    ASSERT actual vs expected
60
61    CALL TestCase "Payment Trx"
62        params = { RcvAmt, UniqueNo,
    RefPaymentAllocCode = PRE_XX, AllocAmt }
63
64    LOAD payment result
65    EXTRACT PayRcvNo
66
67    WAIT for journal posting for PayRcvNo
```

```
68
69      PREPARE expected journal rows:
70          DEBET "-" for RAmt
71          CREDIT "PRE_XX" for AllAmt
72      ASSERT actual vs expected
73
74      WAIT for UniqueNo mirror replication to Recovery
75
76      CALL TestCase "Deal Amount"
77          params = { dealAmt = ntfAmt / 2 }
78
79      LOAD deal amount response
80      EXTRACT trxNoDealXX
81
82      WAIT for journal posting
83
84      PREPARE expected journal rows:
85          DEBET PREPAID
86          CREDIT OTHALXX
87      ASSERT actual vs expected
88
89      CALL TestCase "Debt Forgiveness"
90          params = { uniqueNo, requestDtTime =
    businessDate }
91
92      CALL TestCase "Update Contract Stat Code to Exp"
93          params = { uniqueNo, Code = Key }
94
95  END
```

Kode 3.8: Pseudocode End-to-End Test Flow

Pseudocode diatas menggambarkan salah satu alur lengkap pengujian end-to-end pada proses finansial di sistem CONFINS, dimulai dari inisialisasi seluruh kebutuhan yang diperlukan untuk validasi JSON, pengecekan status, akses database, hingga verifikasi jurnal. Setelah diinisialisasi, sistem mengambil business date langsung dari database dan mengubahnya ke format yang dibutuhkan untuk proses transaksi.

Pengujian diawali dengan menjalankan test case untuk mempersiapkan data awal. Setelah itu dilakukan pengaturan tambahan jika dibutuhkan sesuai kebutuhan skenario. Tahap berikutnya adalah menjalankan siklus aktualisasi API untuk simulasi *end of day* yang dibutuhkan oleh rangkaian skenario sampai benar-benar selesai.

Ketika data aktivasi agreement sudah tersedia, nomor kontrak diambil dari global variable dan ditunggu hingga mirror table kepada seluruh database selesai.

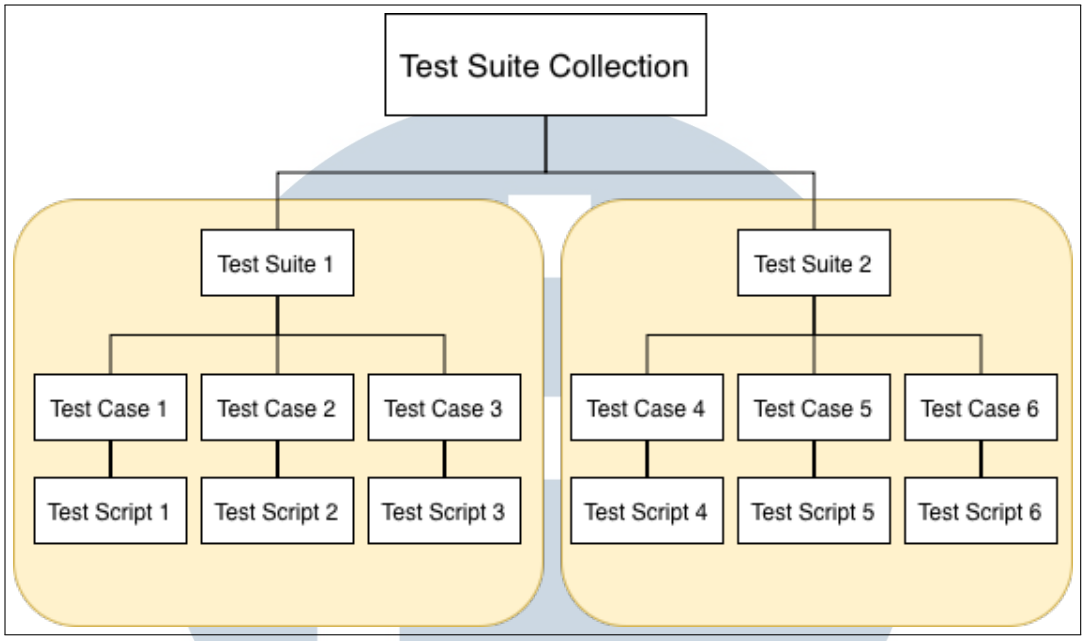
Kemudian recognized interest dihitung ulang karena menjadi dasar untuk proses API selanjutnya. Setelah API Non Accrual dipanggil, nomor transaksinya diambil lalu diverifikasi jurnalnya dengan membandingkan nilai aktual dari database dengan jurnal yang seharusnya terbentuk, seimbang antara credit dan debitnya.

Proses kemudian dilanjutkan dengan tahap request deactivation hingga pembentukan assertasi yang juga divalidasi. Setelah selesai, dilakukan proses pembayaran terkahir yang membentuk validasi penerimaan pembayaran dan kembali divalidasi komponennya. Ketika agreement sudah tercermin ke modul akhir pada *recovery*, pengujian berlanjut pada proses API Deal Amount yang juga menghasilkan jurnal. Jurnal ini dicek kembali apakah sesuai rule yang sudah ditetapkan. Setelah seluruh alur recovery selesai, dilakukan proses akhir untuk Debt Forgiveness dan akhirnya perubahan status contract menjadi expired sebagai langkah penutup dalam end-to-end flow tersebut.

Penjelasan ini mencerminkan bahwa keseluruhan alur pseudocode bertujuan memastikan setiap proses bisnis, mulai dari aktivasi, non accrual, write off, payment, recovery, hingga update status, berjalan sesuai aturan bisnis dan seluruh jurnal yang dihasilkan benar adanya dari sisi backend.

3.3.3 Proses Testing End-to-End API Automation Testing pada Produk Core Multifinance System (CONFINS)

Proses testing dalam skrip end-to-end melewati beberapa tahap dan juga beberapa proses pengecekan, dalam testing API digunakan *test suite* dan *test suite collection* untuk melakukan pengujian agar bisa menampilkan bentuk report. Struktur isi *test suite collection* adalah seperti berikut.



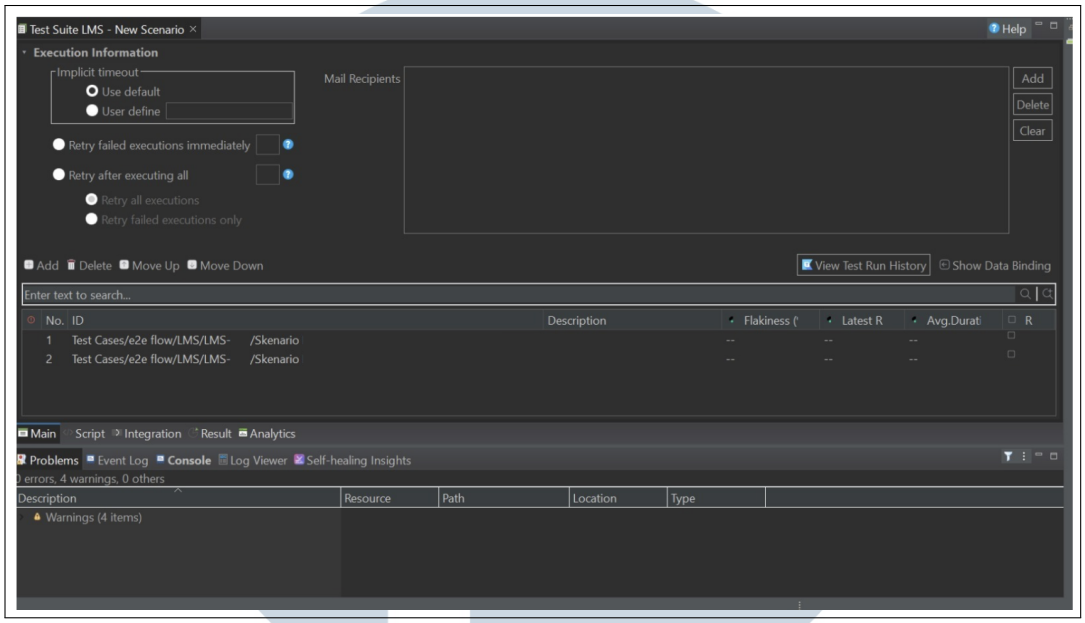
Gambar 3.6. Struktur isi test suite collection

Gambar 3.6 di atas menggambarkan struktur *test suite collection* yang digunakan dalam proses pengujian otomatis. Di dalam satu *test suite collection* terdapat beberapa *test suite*, dan setiap *test suite* terdiri dari sejumlah *test case* yang mewakili skenario pengujian tertentu. Setiap *test case* kemudian berisi *test script* yang menjalankan langkah–langkah pengujian secara otomatis. Dengan struktur bertingkat ini, pengujian dapat diatur dengan eksekusi secara paralel maupun berurutan, serta memudahkan pembuatan laporan yang komprehensif untuk keseluruhan rangkaian tes.

A Test Suite

Sesuai Gambar 3.6, di dalam *test suite* terdapat *test case* yang dapat dipanggil satu atau lebih sehingga dapat dijalankan secara bersamaan. Sebuah *test suite* pada dasarnya merupakan kumpulan dari beberapa *test case*. Setiap *test case* juga dapat menjadi bagian dari *test suite* yang berbeda, tergantung pada kebutuhan pengujian. Pengelompokan *test case* ke dalam sebuah *test suite* biasanya dilakukan berdasarkan logika tertentu, komponen tertentu, atau fitur/aturan bisnis. Selain itu, sebuah *test suite* dapat digunakan secara dinamis untuk memastikan cakupan terhadap berbagai kebutuhan atau *requirements* yang harus divalidasi dalam proses pengujian. Pada pengujian CONFINS, test suite digunakan untuk mengelompokan test case end-to-end berdasarkan skenario tertentu, agar mendapatkan report yang

bisa dianalisa setelahnya. Berikut adalah contoh tampilan *test suite* yang digunakan pada Gambar 3.7.



Gambar 3.7. Tampilan test suite collection

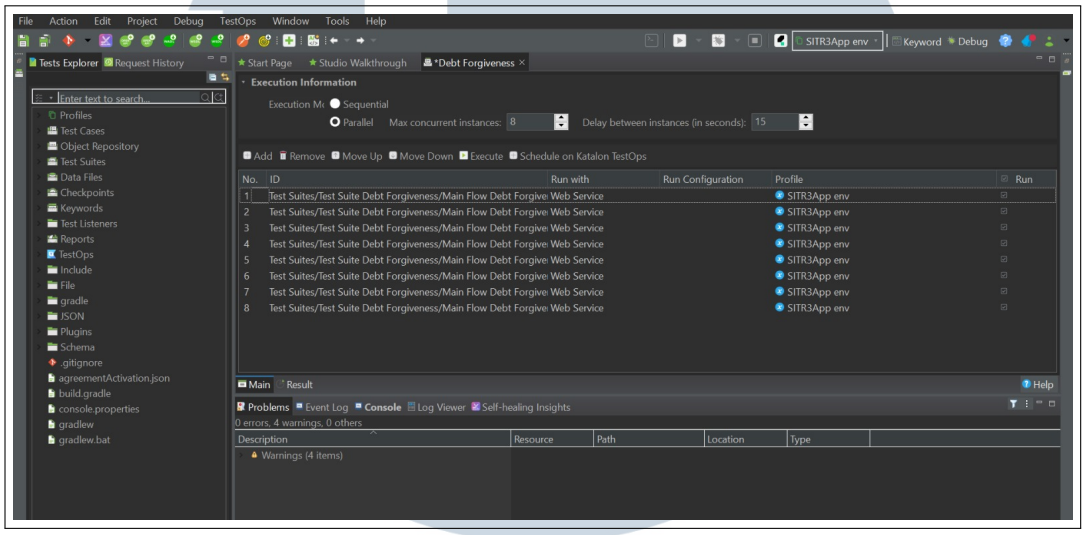
Didalam sebuah test suite, ada beberapa fitur yang dapat dilakukan untuk mempermudah pengujian dan mengelola seluruh skenario E2E dalam satu wadah eksekusi. Pada bagian atas terdapat konfigurasi *execution information*, untukmengatur implicit timeout apakah memakai default atau mengatur sendiri batas waktu tunggu element, eksekusi yang gagal juga dapat diulang secara otomatis sesuai dengan kebutuhan testing, baik langsung setelah gagal atau setelah seluruh test selesai, sehingga cocok untuk meminimalkan false fail akibat flakiness.

Panel utama di bawahnya menampilkan daftar test case yang termasuk dalam test suite beserta informasi seperti ID test case, status, hasil eksekusi terakhir, dan rata-rata durasi runtime, sehingga memudahkan monitoring performa dan stabilitas test. Secara keseluruhan, tampilan ini memungkinkan untuk mengatur urutan testing, menjalankan test secara batch, memantau stabilitas test, melakukan retry otomatis, dan melihat hasil riwayat running skrip.

B Test Suite Collection

Setelah *test case* dimasukkan ke dalam *test suite*, setiap *test suite* akan menghasilkan *report* secara individu. Namun, untuk dapat melihat hasil pengujian sebagai satu kesatuan yang terintegrasi, beberapa *test suite* kemudian digabungkan

ke dalam sebuah *test suite collection*. Dengan menggunakan *test suite collection*, proses eksekusi dapat dijalankan secara berurutan maupun parallel, sehingga seluruh alur pengujian end-to-end dapat dievaluasi secara menyeluruh. Selain itu, *test suite collection* memungkinkan pengelolaan skenario pengujian yang lebih kompleks serta mempermudah pembuatan *report* yang komprehensif, mencakup semua *test suite* yang termasuk di dalamnya formatnya seperti pada Gambar 3.8.



Gambar 3.8. Tampilan test suite collection

Di dalam sebuah *test suite collection* terdapat beberapa opsi untuk menjalankan proses eksekusi. Skrip dapat dijalankan secara *parallel* maupun *sequence* dengan melakukan pengaturan pada parameter *max concurrency instance* dan *delay between instance*. Pengaturan ini memungkinkan penyesuaian terkait jumlah proses yang dapat berjalan bersamaan serta jeda waktu antar eksekusi, sehingga alur pengujian dapat disesuaikan dengan kebutuhan dan kapasitas *environment* pengujian. Berikut adalah fitur *test suite collection* secara lebih lengkap pada Tabel 3.3.

Tabel 3.3. Fitur pada *Test Suite Collection* untuk API Testing

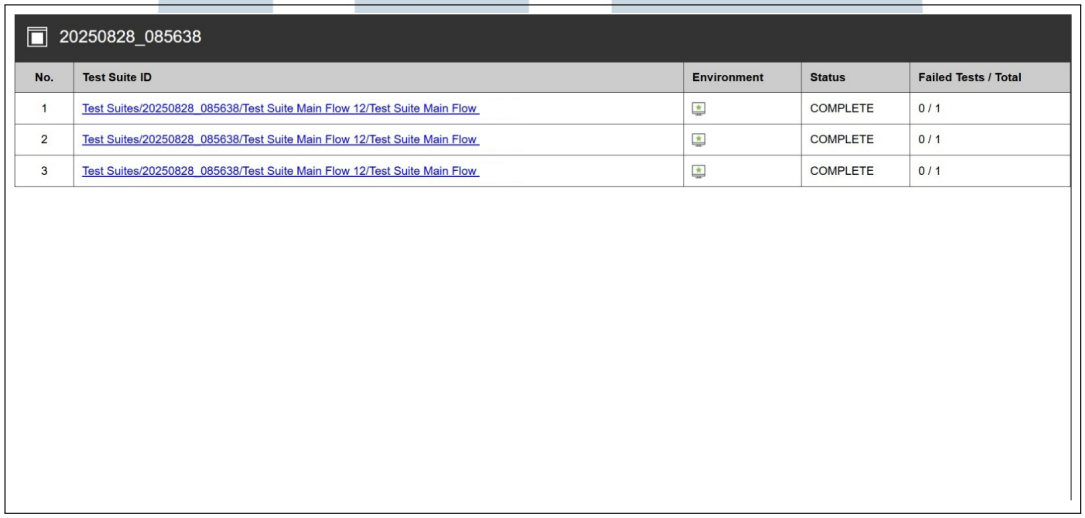
Fitur	**Deskripsi**
Run with	Digunakan untuk memilih *environment* yang akan digunakan dalam menjalankan *test suite*, seperti *Chrome*, *Firefox*, atau *API environment* yang terkait dengan jenis pengujian.
Execution Mode	Kolom ini memungkinkan pengguna menentukan cara menjalankan proses pengujian. Terdapat dua opsi utama, yaitu *sequential* dan *parallel*. Pada mode *sequential*, seluruh *test suite* dijalankan satu per satu secara berurutan. Sementara itu, pada mode *parallel*, beberapa *test suite* dapat dijalankan secara bersamaan. Mode ini juga dapat dikonfigurasi dengan parameter seperti *max concurrency instance* dan *delay between instance* untuk mengatur jumlah proses paralel dan jeda antar eksekusi.
Profile	Mengatur *execution profile* yang berisi nilai global variabel untuk setiap proses eksekusi *test suite*, sehingga lebih mudah untuk berpindah *environment* / server pengujian dengan nilai berbeda, seperti perubahan base URL. Penggunaan *execution profile* memungkinkan konsistensi data uji, pengaturan *environment*, serta fleksibilitas dalam mengeksekusi skenario yang sama dengan konfigurasi yang berbeda.
Execute	Digunakan untuk memilih *test suite* mana saja yang akan dijalankan dalam *test suite collection*. Setelah dipilih, seluruh *test suite* tersebut dapat dieksekusi sesuai konfigurasi yang telah ditentukan. Fitur ini memastikan hanya *test suite* yang relevan saja yang dijalankan dalam satu rangkaian eksekusi.

Pada implementasinya di environment CONFINS, beberapa proses pengujian dijalankan secara parallel untuk meningkatkan efisiensi waktu serta mengoptimalkan pemanggilan API EOD yang dapat mengubah tengat waktu pada kontrak. Dengan metode ini, beberapa *agreement* dapat diproses secara bersamaan dalam rentang waktu yang sama sesuai dengan *grouping* skenario yang telah ditetapkan sebelumnya. Setiap skenario atau rangkaian API yang dijalankan memiliki struktur eksekusi yang serupa sebagaimana digambarkan pada Gambar 3.8. Melalui pendekatan parallel execution ini, hasil keseluruhan proses dapat dievaluasi lebih cepat dan lebih komprehensif melalui testing report, sehingga

identifikasi keberhasilan ataupun kegagalan langkah pengujian dapat dilakukan secara lebih efektif.

C Testing Report

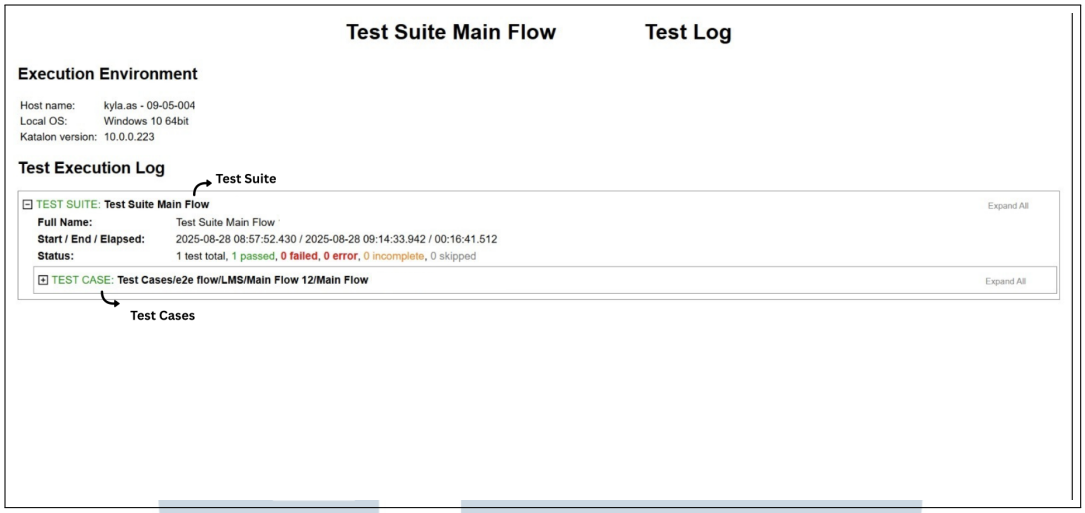
Setelah proses testing dijalankan dalam *test suite collection*, katalon akan mengerluarkan report yang dapat dianalisa atau dilihat kembali untuk memastikan alur bisnis sudah sesuai dan tidak ada proses yang terlewatkan, berikut adalah tampilan dari hasil testing report pada Gambar 3.9.


20250828_085638

No.	Test Suite ID	Environment	Status	Failed Tests / Total
1	Test Suites/20250828_085638/Test Suite Main Flow 12/Test Suite Main Flow		COMPLETE	0 / 1
2	Test Suites/20250828_085638/Test Suite Main Flow 12/Test Suite Main Flow		COMPLETE	0 / 1
3	Test Suites/20250828_085638/Test Suite Main Flow 12/Test Suite Main Flow		COMPLETE	0 / 1

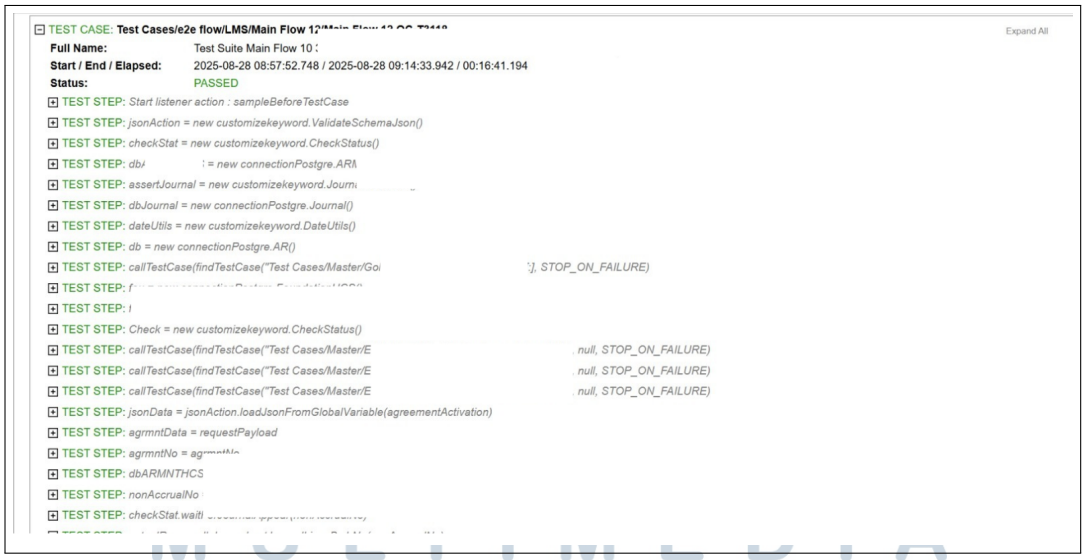
Gambar 3.9. Tampilan hasil testing report test suite collection

Pada Gambar 3.9 ditampilkan tampilan awal hasil *report* dari proses pengujian. Pada halaman ini terdapat beberapa tautan yang mengarah pada *report* untuk setiap *test suite*. Seluruh *report* tersebut dihasilkan secara otomatis oleh Katalon dan dapat dibuka melalui browser untuk melihat detail hasil pengujian pada masing-masing *test suite*. Kemudian dapat dilihat isi dari setiap hasil pengujian seperti berikut.



Gambar 3.10. Tampilan hasil testing report test suite

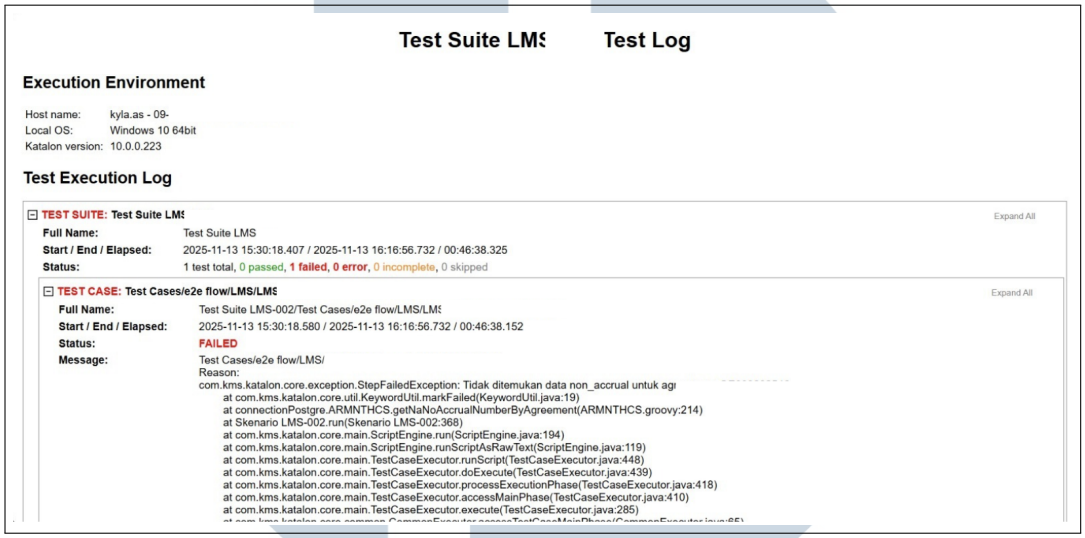
Pada Gambar 3.10 ditampilkan isi *report* pada level *test suite*. Di dalamnya terdapat daftar *test case* yang diuji, dan setiap *test case* menampilkan detail hasil pengujian API. Melalui tampilan ini, dapat terlihat dengan jelas status eksekusi setiap langkah pengujian, apakah berhasil atau mengalami kegagalan, sehingga memudahkan dalam melakukan analisis hasil *testing*. Berikut adalah tampilan secara detail isi dari laporan level *test case* pada Gambar 3.11.



Gambar 3.11. Tampilan hasil testing report test case

Didalam report yang dihasilkan, setiap langkah eksekusi dapat diamati secara detail sehingga memudahkan pengecekan apakah seluruh proses berjalan sesuai alur yang diharapkan. Informasi seperti hasil respons API, nilai yang di asertasi, serta status setiap tahapan ditampilkan secara jelas sehingga dapat

memastikan bahwa perilaku sistem sudah sesuai dengan kebutuhan skenario dan cepat mengidentifikasi apabila terjadi penyimpangan atau kegagalan. Sebagai perbandingan, berikut adalah contoh tampilan *test suite* yang memiliki kegagalan pada Gambar 3.12.



Gambar 3.12. Tampilan hasil testing report test case failed case

Test report menampilkan lima jenis hasil eksekusi, yaitu *passed*, *failed*, error (misalnya karena masalah koneksi), *incomplete*, dan *skipped*. Pada contoh di atas terlihat bahwa beberapa tahap berstatus gagal karena terdapat ketidaksesuaian pada proses validasi di sisi backend. Melalui report ini, QE dapat langsung mengidentifikasi langkah mana yang tidak memenuhi kebutuhan skenario, apakah karena nilai yang tidak sesuai, respons API yang salah, atau kegagalan proses pada sistem. Dengan demikian, report memudahkan proses analisis dan perbaikan karena setiap step dieksekusi secara terperinci dan ditampilkan lengkap beserta pesan kegagalannya.

3.4 Kendala dan Solusi yang Ditemukan

Selama periode kerja magang track kedua di PT Adicipta Inovasi Teknologi sebagai Quality Engineer atau QA Automation Engineer, terdapat beberapa kendala yang muncul selama proses development dan testing sebagai berikut:

1. Terjadi perubahan kebutuhan (requirement) yang cukup dinamis selama proses testing, sehingga beberapa test case harus diperbarui agar tetap sesuai dengan alur bisnis dan logika sistem terbaru.

2. Ditemukan kebutuhan untuk melakukan eksplorasi tambahan pada pengujian skenario tertentu, terutama ketika diperlukan mekanisme khusus seperti perulangan (*looping*), penyimpanan data JSON dengan penanda waktu (*timestamp*), atau trik tertentu agar data antar proses tidak saling tertukar, khususnya saat menjalankan beberapa skenario yang bersifat pararel.

Melalui kendala tersebut, dilakukan beberapa solusi yang efektif sebagai berikut:

1. Untuk menghadapi perubahan requirement, setiap pembaruan dilakukan sesuai prosedur. Pendekatan test case yang bersifat reusable juga dimanfaatkan, sehingga perubahan cukup diterapkan pada modul tertentu tanpa perlu memodifikasi keseluruhan rangkaian skenario.

2. Pada skenario yang membutuhkan penanganan khusus, digunakan pendekatan dynamic data handling, seperti menambahkan timestamp pada penyimpanan JSON, menggunakan variable global sementara, atau membangun fungsi utilitas yang mampu mengeksekusi proses berulang secara otomatis agar data tetap konsisten dan tidak tertukar antar proses.