BAB 3
PELAKSANAAN KERJA MAGANG

Pelaksanaan kerja magang dilakukan di PT Adicipta Inovasi Teknologi pada posisi sebagai *Quality Engineer* di bawah Departemen *Software Testing*. Kegiatan magang difokuskan pada pengembangan dan pelaksanaan *automation testing* untuk memastikan kualitas perangkat lunak internal perusahaan berjalan sesuai dengan ketentuan fungsional yang telah ditetapkan.

Selama masa magang, terlibat secara langsung dalam berbagai kegiatan pengujian otomatis terhadap sistem, baik pada sisi antarmuka pengguna (*frontend*) maupun layanan aplikasi (*backend*) melalui *API*. Proses pengujian dilakukan menggunakan *tools* yang umum digunakan di industri, seperti Katalon Studio dan PostgreSQL, dengan pendekatan skenario uji yang mencerminkan alur proses bisnis aktual.

Beberapa informasi teknis yang berkaitan dengan sistem internal dan data perusahaan tidak dapat ditampilkan secara rinci dalam laporan ini karena bersifat rahasia dan dilindungi oleh kebijakan perusahaan. Penyajian isi laporan ini telah disesuaikan agar tetap mencerminkan pengalaman kerja secara umum tanpa melanggar ketentuan kerahasiaan yang berlaku.

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kegiatan magang sebagai *Quality Engineer*, posisi yang dijalankan berada di bawah Departemen Software Testing dengan fokus utama untuk memastikan kualitas perangkat lunak berjalan sebagaimana mestinya.

Posisi ini berada langsung di bawah supervisi Senior Automation Engineer, yang bertanggung jawab memberikan arahan teknis serta melakukan evaluasi terhadap setiap tugas yang dilaksanakan. Pengawasan ini juga berada di bawah koordinasi Section Head dan Head of Software Testing Department.

Koordinasi kerja dilakukan melalui rapat mingguan (*weekly meeting*) secara luring maupun daring menggunakan aplikasi Microsoft Teams, rapat harian kemajuan tim Automation Testing (*daily meeting progress*) melalui Teams, serta komunikasi rutin melalui grup koordinasi di platform yang sama.

3.2 Tugas yang Dilakukan

Selama masa magang pada PT Adicipta Inovasi Teknologi, tugas utama yang dilakukan mencakup melanjutkan skrip untuk automation testing yang sedang berlangsung sebelum dijalankan oleh tim testing serta tim produk. Berikut adalah tugas yang dilakukan:

1. Melakukan pembangunan skrip *automation testing* menggunakan Katalon Studio dengan bahasa pemrograman Groovy, yang ditujukan untuk pengujian sistem auto signing dan core system multifinance. Proses ini mencakup pembuatan skenario uji otomatis yang mereplikasi proses bisnis aktual pada aplikasi, mulai dari login, input data, hingga proses persetujuan atau *approval* alur kerja, dengan tujuan untuk memastikan bahwa setiap fungsionalitas sistem berjalan sesuai dengan yang diharapkan.

2. Melakukan proses validasi data secara langsung terhadap database menggunakan query SQL untuk mendukung pengujian pada sisi *backend* melalui postgre, khususnya dalam konteks pengujian API. Validasi ini diperlukan guna memastikan bahwa data yang dikirimkan dan diterima melalui layanan API sesuai dengan yang tersimpan di database, baik dari segi nilai, perhitungan, maupun relasi antar entitas data.

3. Melakukan pembangunan skrip *automation testing* untuk pengujian sisi antarmuka pengguna (*frontend*). Pengujian ini bertujuan untuk memastikan bahwa elemen-elemen UI berfungsi dengan baik dan sesuai dengan desain yang telah ditentukan. Pengujian ini juga mencakup verifikasi terhadap responsivitas tampilan dan konsistensi alur website hingga verifikasi langsung ke database.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan magang untuk kebutuhan Internship Track 1 dilaksanakan pada periode Februari 2025 hingga Juni 2025, sesuai dengan waktu kontrak yang berlaku dari Februari 2025 hingga Februari 2026. Pada semester pertama ini, kegiatan magang mencakup dua 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
1-2	*Onboarding*, mengenal lingkungan perusahaan, pemberian perangkat untuk *development*, serta *training* awal khusus untuk peserta magang
3-4	*On the job training* untuk *task automation*, mengenal alur untuk membangun skrip *automation* serta aturan *commit* skrip *automation testing* dan *code review*
5-11	Melakukan pembangunan skrip *automation testing* sesuai dengan tiket *task* dari Jira untuk perangkat lunak pertama AdIns, proyek *web app auto signing* untuk dokumen keuangan
12	*Enhancements* serta *retesting* skrip *automation* proyek pertama *app auto signing* sebelum berpindah ke proyek kedua
13 - 14	Persiapan awal berpindah ke proyek *automation* kedua, meliputi *weekly meeting*, eksplorasi *API* serta *frontend*, *sprint meeting*
15 - 16	Melakukan pembangunan skrip *automation testing* untuk *backend API automation* proyek kedua, *multifinance core system*
16 -	Melakukan pembangunan skrip *automation testing* untuk *backend API automation multifinance core system*, beserta *review* dan *enhancements*

3.3.1 *Automation Testing Onboarding*

Automation testing melibatkan pembuatan skrip menggunakan bahasa pemrograman, seperti Python, Java, Groovy dan JavaScript, sehingga kasus uji dapat dijalankan dengan intervensi dan perhatian manusia yang lebih minimal [5]. *Automation testing* memiliki berbagai manfaat yang signifikan jika dibandingkan dengan *manual testing*. Proses ini lebih cepat karena dijalankan secara otomatis oleh sistem serta bersifat dinamis [6]. Selain itu, penggunaan *tools automation* memungkinkan pengurangan jumlah penguji yang dibutuhkan, sehingga dapat menghemat biaya. Dalam praktiknya, *automation testing* umumnya diterapkan pada pengujian *black-box testing*, di mana fokusnya adalah menguji fungsionalitas aplikasi tanpa perlu mengetahui struktur internal kode [6]. Hal ini memungkinkan

pengujian dilakukan secara menyeluruh dari sisi pengguna, termasuk pengujian antarmuka (UI), *API*, dan alur aplikasi, sesuai dengan kebutuhan pengguna akhir.

Automation testing juga memungkinkan pengujian yang berulang dengan mudah, karena skrip yang telah dibuat dapat dijalankan kembali kapan saja. Skrip bersifat *reusable*, sehingga dapat digunakan kembali pada versi rilis aplikasi yang berbeda tanpa harus membuat ulang pengujian dari awal. Kemampuan untuk memprogram pengujian secara kompleks juga memberikan fleksibilitas bagi penguji untuk menemukan informasi tersembunyi dalam sistem. *Automation testing* mendukung cakupan pengujian yang luas dan menyeluruh terhadap berbagai fitur aplikasi, serta memberikan hasil yang konsisten setiap kali dijalankan, sehingga menjadikannya metode unggul dalam proses *testing* [7].

Pengujian otomatis dan pengujian manual memiliki keunggulan dan prioritas masing-masing. Pengujian manual umumnya digunakan untuk skenario yang lebih banyak melibatkan perilaku pengguna (*user usage*), atau untuk pengujian eksploratif yang membutuhkan intuisi dan penilaian manusia. Sementara itu, pengujian otomatis sangat efektif untuk pengujian regresi, pengujian berulang dalam jumlah besar, serta memastikan konsistensi dan efisiensi dalam jangka panjang. Kombinasi keduanya sering kali digunakan untuk mencapai cakupan pengujian yang optimal dan meningkatkan kualitas perangkat lunak secara keseluruhan. Sumber [6] melampirkan perbandingan mengenai *automation testing* yang kemudian dialami selama proses pelaksanaan magang.

Tabel 3.2. Perbandingan pengujian manual dan pengujian otomatis

No.	Pengujian Manual	Pengujian Otomatis
1.	Kecepatan eksekusi bergantung pada manusia, sehingga memerlukan waktu lebih lama.	Pengujian dilakukan menggunakan skrip yang telah dibangun sebelumnya, sehingga proses berjalan lebih cepat dibandingkan dengan tenaga manusia.
2.	Karena dilakukan seperti layaknya pengguna pada umumnya, pengujian manual membutuhkan lebih banyak sumber daya manusia.	Menggunakan alat otomatis, sehingga jumlah penguji yang dibutuhkan lebih sedikit dan lebih efisien dalam penggunaan sumber daya manusia.
3.	Tidak memerlukan pemrograman, tetapi pengujian manual tidak cocok untuk skenario kompleks yang membutuhkan analisis mendalam.	Pengujian otomatis memerlukan pembuatan skrip terlebih dahulu, namun mampu mencakup pengujian yang lebih luas seperti kombinasi database, API, dan frontend.
4.	Rentan terhadap human error yang dapat memengaruhi konsistensi hasil pengujian.	Lebih akurat dan minim kesalahan karena dijalankan oleh sistem setelah skrip disiapkan.
5.	Pengujian dapat dilakukan secara menyeluruh	Memiliki beberapa keterbatasan dalam pengujian otomatis yang bersangkutan dengan *human approval*, sehingga memerlukan keterlibatan pengujian manual.

Pada awal tahap magang, saat proses *onboarding*, karyawan magang diberikan *training* sesuai dengan *jobdesk* nantinya. Dalam posisi *quality engineer intern*, terdapat pelatihan SQL, pelatihan dasar produk Adins, pelatihan materi finansial, dan yang paling utama adalah pelatihan *automation testing* mencakup penjelasan *tools*, *automation testing backend API*, serta *automation testing frontend UI* dengan dasar sebagai berikut.

A Tools

1. Katalon

   Katalon adalah platform *automation testing* yang digunakan untuk menguji aplikasi web dan API produk Adins. Katalon yang digunakan adalah versi 10.0.0. Katalon menyediakan antarmuka pengguna yang mendukung pengujian berbasis skrip dan tanpa skrip baik untuk pengujian API maupun *frontend UI* menggunakan *Behaviour Driven Development (BDD)*. Selain itu, Katalon mendukung integrasi dengan berbagai *tools* lain yang digunakan perusahaan seperti Git dan JIRA untuk mendukung proses *Continuous Integration* dan *Continuous Testing*.

2. PostgreSQL

   PostgreSQL adalah sistem manajemen basis data SQL yang digunakan untuk melakukan pengecekan data oleh *software tester*. *Tools* ini digunakan untuk menyimpan, mengelola, dan mengakses data dengan menggunakan bahasa SQL. Dalam konteks pengujian perangkat lunak, PostgreSQL digunakan untuk memverifikasi data *backend*, menjalankan *query* untuk validasi data, serta mendukung pengujian berbasis data (*data-driven testing*) menggunakan Katalon.

3. Postman

   Postman adalah *tools* yang digunakan untuk pengujian *Application Programming Interface (API)* secara manual untuk melakukan verifikasi akurasi *payload* sebelum dijadikan skrip *automation* yang *reusable*. Postman digunakan untuk mengirim permintaan (*request*) HTTP ke *server*, menerima *response*, serta melakukan validasi terhadap data yang diterima.

4. VSCode

   VSCode digunakan sebagai *tools* untuk memperbaiki *conflicted file* saat melakukan *PR (Pull Request)*. Konflik ini biasanya terjadi ketika terdapat perbedaan antara *branch* dan *master*. VSCode secara otomatis akan mendeteksi *file* yang mengalami konflik dengan menunjukkan bagian-bagian yang saling bertentangan, seperti perubahan dari *branch lokal (current change)* dan perubahan dari *branch* yang akan digabung (*incoming change*). Konflik memiliki pilihan untuk ”*Accept Current Change*”, ”*Accept Incoming*

Change", "*Accept Both Changes*", atau memungkinkan pengeditan manual untuk menyelesaikan konflik sesuai kebutuhan. Setelah konflik diselesaikan, *file* dapat disimpan dan ditandai sebagai *resolved* menggunakan perintah seperti git add, lalu dilanjutkan dengan *commit* dan *push*.

5. JIRA

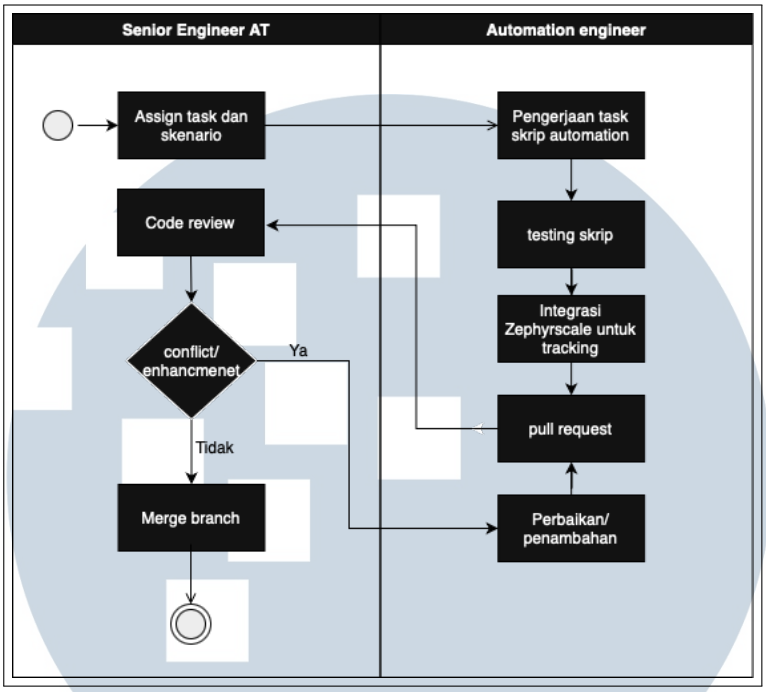
   JIRA adalah *tools* manajemen proyek yang dikembangkan oleh Atlassian dan banyak digunakan dalam pengembangan perangkat lunak berbasis *Agile*. JIRA digunakan oleh tim QE untuk merencanakan, melacak, dan mengelola *task* atau *bug* selama *sprint* pengembangan perangkat lunak. JIRA digunakan untuk mencatat dan memantau *bug*, mengelola *backlog*, serta menghubungkan hasil pengujian dengan *ticket* yang relevan. *Plugin* yang utama digunakan tim QE adalah Zephyrscale yang digunakan khusus untuk manajemen pengujian, termasuk pembuatan *test case*, pelaksanaan *test run*, dan pelaporan hasil pengujian yang terintegrasi langsung dengan Katalon untuk proses pengembangan.

6. Bitbucket

   Bitbucket merupakan salah satu produk dari Atlassian yang berfungsi sebagai platform berbasis Git untuk manajemen kode sumber (*source code management*). Platform ini mendukung kolaborasi tim dalam pengembangan perangkat lunak melalui berbagai fitur, seperti *pull request*, *branch management*, serta integrasi dengan *tools* lain seperti JIRA dan *CI/CD pipelines*. Dalam konteks *testing*, Bitbucket digunakan untuk menyimpan skrip *automation testing* serta memantau setiap perubahan kode yang dapat memicu proses pengujian otomatis. Seluruh skrip yang dikembangkan oleh tim QE wajib melalui proses *code review* oleh *senior automation engineer*. Proses ini dilakukan saat *pull request* diajukan, di mana Bitbucket akan menjadi tempat pengecekan awal untuk memastikan tidak terjadi *conflict* dengan *branch* lain sebelum skrip digabungkan ke dalam repositori utama.

B Alur Prosedur Pengembangan Skrip *Automation*

Sistem alur pembangunan skrip automation testing pada tim AT atau tim *Automation Testing* di PT Adicipta Inovasi Teknologi mencakup proses berikut



Gambar 3.1. Alur pembangunan skrip automation testing

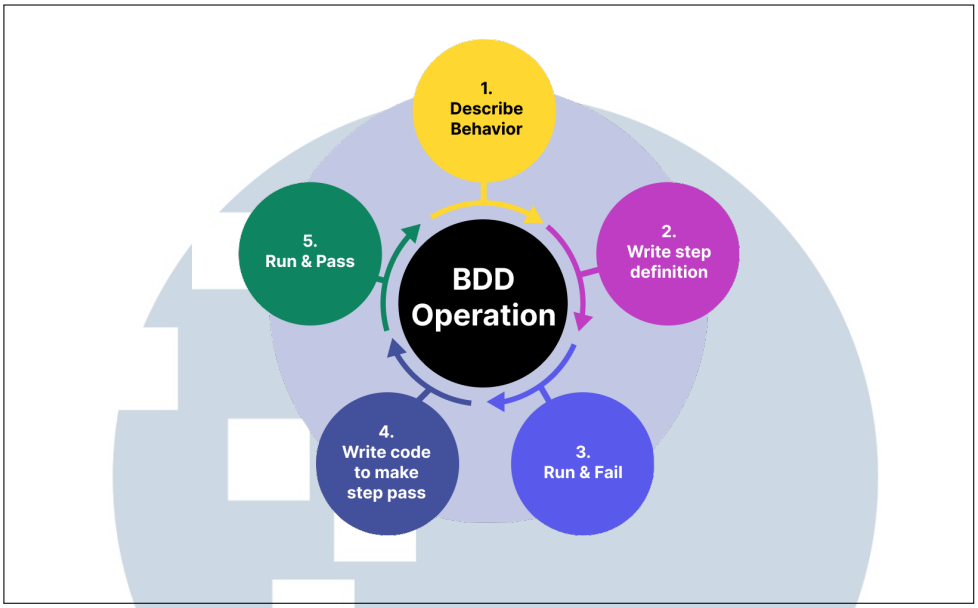
Proses kerja dimulai dari *Senior Engineer AT* yang bertugas untuk melakukan *assign task* dan skenario pengujian kepada *automation engineer*. Setelah menerima *task*, *automation engineer* akan mulai mengerjakan skrip *automation* sesuai dengan skenario yang diberikan. Setelah skrip selesai dibuat, langkah selanjutnya adalah melakukan pengujian terhadap skrip tersebut untuk memastikan bahwa fungsinya berjalan sesuai harapan. Jika pengujian berhasil, skrip akan diintegrasikan ke dalam Zephyrscale sebagai alat untuk melakukan pelacakan dan dokumentasi hasil pengujian.

Setelah proses integrasi selesai, *automation engineer* mengajukan *pull request* agar skrip dapat direview oleh *senior engineer AT*. *Senior engineer* kemudian melakukan *code review* terhadap skrip yang diajukan. Jika ditemukan adanya konflik atau kebutuhan untuk *enhancement*, skrip akan dikembalikan kepada *automation engineer* untuk dilakukan perbaikan atau penambahan. Setelah revisi dilakukan, *pull request* akan diajukan kembali dan proses *review* diulang. Jika skrip dinilai sudah sesuai dan tidak terdapat konflik, maka *branch* akan di-*merge* ke dalam *repositori* utama sebagai tahap akhir dari proses ini.

C *Frontend UI Automation Testing*

Pengembangan web front-end memiliki perubahan yang cepat, yang menuntut pembaruan UX dan peluncuran fitur secara berkala. Lingkungan yang dinamis seperti ini menimbulkan tantangan bagi metode pengujian manual, karena sering menghadapi beban pemeliharaan yang tinggi [8]. Sehingga, prosedur pembangunan automation skrip untuk frontend atau web APP berbeda dari pengujian API, pengujian web otomatis harus memastikan bahwa fungsionalitas umum dari aplikasi web berjalan dengan benar, serta memungkinkan pengujian dapat digunakan kembali dan diperluas ke berbagai browser, platform, database, dan server. Selain itu, pengujian otomatis juga harus menjamin bahwa semua pengguna dapat mengakses aplikasi web dengan waktu respons yang dapat diterima.

Dalam realisasinya frontend testing di PT Adicipta Inovasi Teknologi menggunakan teknik BDD atau *behavior-driven development*, teknik ini mendukung sistem agile dalam software engineering. Penerapan kerangka kerja automation testing yang memanfaatkan Cucumber-BDD dan Groovy memberikan efisiensi serta efektivitas proses pengujian perangkat lunak [9]. Cucumber-BDD memungkinkan keterlibatan aktif dari pemangku kepentingan non-teknis melalui skenario pengujian. Penggunakan BDD juga didasari oleh bahasa pemograman Groovy berbasis java yang memiliki fleksibilitas tinggi melalui pustaka yang kaya dan dukungan lintas platform serta penerapan *object oriented programming*, yang sangat bermanfaat dalam membangun solusi otomasi yang dapat diskalakan dan disesuaikan seiring perkembangan proyek. Kombinasi keduanya mendukung pembentukan kerangka kerja pengujian yang modular, mudah dirawat, dan dapat digunakan kembali. Berikut adalah alur proses BDD yang digunakan secara langsung di katalon.

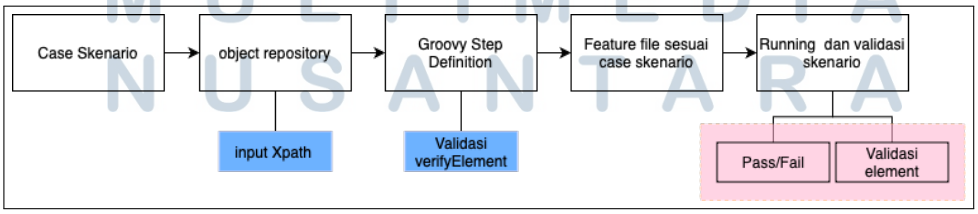


Gambar 3.2. Alur penerapan BDD dalam alur pengujian

Sumber: [10]

Proses ini dimulai dengan mendefinisikan perilaku sistem menggunakan pendekatan Behavior-Driven Development (BDD) dalam format Gherkin, seperti Given, When, And, Then, untuk menggambarkan skenario dari sudut pandang pengguna. Langkah selanjutnya adalah menulis kode untuk membangun pengujian pertama yang pada awalnya akan mengalami *run fail*, lalu memperbaiki setiap langkahnya hingga pengujian tersebut berhasil dijalankan. Setelah semua pengujian lulus, tahap akhir adalah menyempurnakan struktur dan kualitas kode tanpa mengubah perilaku fungsional yang telah terverifikasi melalui pengujian.

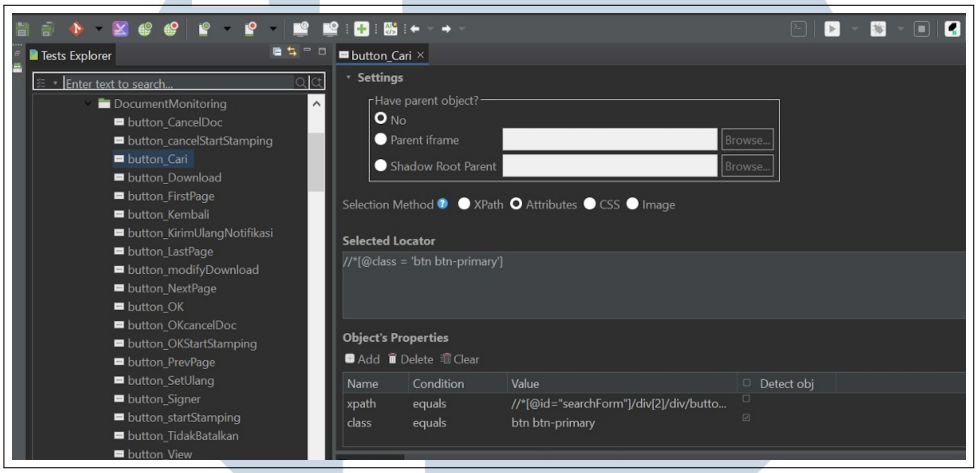
Frontend atau web app testing mencakup proses pembuatan object repository, skrip cucumber, serta file feature untuk menjalankan BDD menggunakan format Given, When, Then, sehingga sifat testing skrip dapat digunakan kembali untuk setiap step yang sudah di deklarasi di skrip. Berikut adalah alur pengujian *frontend* menggunakan skrip otomasi berbasis BDD, yang digunakan untuk memvalidasi alur UI pada aplikasi web.



Gambar 3.3. Alur pembangunan skrip *frontend* menggunakan *automation* BDD

1. Object Repository

   Setelah melakukan definisi skenario yang diberikan dari tiket untuk *development*, tahapan pertama mencangkup pembuatan object repository yang didapatkan secara manual menggunakan XPath langung dari *inspect* pada browser.



Gambar 3.4. Contoh Object Repository dalam skrip frontend automation

Object Repository digunakan untuk menyimpan elemen-elemen antarmuka pengguna seperti tombol (*button*), label, kotak teks (*text box*), dan daftar pilihan (*drop-down list*) yang akan digunakan dalam skrip *automation testing*. Dengan adanya *object repository*, skrip dapat langsung mengidentifikasi keberadaan objek yang dimaksud saat dijalankan.

Penggunaan XPath sangat membantu dalam mendeteksi elemen yang bersifat dinamis. *XML Path Language* (XPath) adalah bahasa *query* yang digunakan untuk menavigasi dan memilih *node* atau elemen dalam dokumen XML atau struktur *Document Object Model* (DOM) pada halaman web atau aplikasi.

Dengan menggunakan XPath, objek dalam aplikasi tetap dapat teridentifikasi meskipun terjadi perubahan versi atau pembaruan antarmuka aplikasi, sehingga skrip *automation* menjadi lebih fleksibel dan dapat digunakan kembali tanpa perlu banyak penyesuaian.

2. Script Groovy File

   Skrip ini digunakan untuk melakukan pengujian *frontend* menggunakan bahasa pemrograman Groovy, yang merupakan pendekatan berbasis bahasa

Java untuk *automation testing*. Di dalam skrip ini, dideklarasikan langkah-langkah pengujian dengan format *Given*, *When*, dan *Then*.

Pada bagian ini, skrip memanfaatkan berbagai fungsi verifikasi untuk memastikan elemen-elemen pada antarmuka tampil dan berperilaku sesuai ekspektasi. Fungsi yang umum digunakan antara lain *verifyElementPresent* untuk memastikan keberadaan elemen di halaman, *verifyElementText* untuk memverifikasi kesesuaian teks yang ditampilkan, serta *click*, *setText*, dan *waitForElementVisible* untuk berinteraksi dengan elemen dan memastikan UI merespons sesuai alur pengujian.

Pendekatan ini mendukung prinsip *reusability* dan *maintainability* dalam otomatisasi pengujian berbasis BDD, sehingga ketika ada perubahan pada aplikasi, kita cukup memperbarui definisi langkah (*step definition*) tanpa harus memodifikasi seluruh skenario pengujian. Berikut adalah contoh *pseudocode* untuk *step* ini.

```
1
2  Class LoginTest
3
4      @Given "Pengguna membuka halaman utama"
5      Function bukaHalamanUtama()
6          Buka browser kosong
7          Arahkan ke alamat GlobalVariabel.URL
8          Maksimalkan tampilan browser
9
10     @Given "Pengguna mengisi email {email}"
11     Function isiEmail(email)
12         Temukan elemen input email
13         Masukkan nilai email ke dalam kolom tersebut
14
15     @Given "Pengguna mengisi kata sandi {password}"
16     Function isiPassword(password)
17         Temukan elemen input password
18         Masukkan nilai password ke dalam kolom tersebut
19
20     @When "Pengguna menekan tombol login"
21     Function klikTombolLogin()
22         Fokuskan pada tombol login
23         Klik tombol login
24
25     @When "Pengguna memilih peran {peran} jika tersedia"
26     Function pilihPeranJikaAda(peran)
27         Tunggu elemen input peran muncul (opsional)
28         Jika elemen input peran ditemukan:
29             Isi kolom peran dengan nilai dari parameter
30             Tekan tombol ENTER untuk mengonfirmasi
31             Jika tombol 'Pilih Peran' tersedia:
32                 Klik tombol tersebut
33                 Catat bahwa tombol berhasil diklik
34             Jika tidak tersedia:
35                 Tampilkan catatan bahwa tombol tidak
   ditemukan
36
37     @Then "Verifikasi bahwa login berhasil"
38     Function verifikasiLoginBerhasil()
```

```
        Pastikan elemen dropdown akun muncul dalam waktu yang
    ditentukan

EndClass
```

Kode 3.1: Pseudocode skenario login pengguna

3. Script Feature File

   Setelah membangun skrip Groovy yang berisi langkah-langkah pengujian, tahap selanjutnya adalah memanggil langkah-langkah tersebut ke dalam file feature untuk dieksekusi. File feature berfungsi sebagai tempat mendefinisikan skenario pengujian menggunakan format Gherkin, yang bersifat deskriptif dan mudah dipahami.

```

Scenario Outline: Login Pengguna dengan Kredensial Valid
   untuk Semua Peran

    When Pengguna mengisi email <email>
    And Pengguna mengisi kata sandi <password>
    When Pengguna menekan tombol login
    And Pengguna memilih peran <role> jika tersedia
    Then Sistem memverifikasi bahwa login berhasil

Examples (Skenario Positif):
    | email                | password     | role         |
    | user.admin@mail.com| admin          | Admin        |
    | user.user@mail.com | user           | user       |

Examples (Skenario Negatif):
    | email                | password     | role         |
    | user.invalid@mail.com | wrongpass   | Admin        |
    | user.blank@mail.com   |             | Viewer       |
    |                       | pass      | user       |
```

Kode 3.2: Pseudocode skenario login pengguna dengan berbagai peran

   Dalam sebuah feature file, skenario pengujian yang ditulis dapat terdiri dari lebih dari satu skenario. Hal ini memberikan kemudahan dalam proses pengujian karena memungkinkan beberapa skenario diuji sekaligus dalam satu eksekusi.

4. Running dan Validasi

   Setelah seluruh skenario pengujian ditulis dan langkah-langkah (*step definitions*) didefinisikan dalam skrip Groovy, proses selanjutnya adalah menjalankan pengujian secara otomatis melalui fitur *Test Suite* atau *Test Runner*. Setiap skenario akan dijalankan sesuai urutan langkah yang telah ditentukan di dalam file *feature*.

Selama proses *running*, sistem akan mengeksekusi perintah seperti *verifyElementPresent*, *verifyElementText*, *click*, dan *waitForElementVisible* untuk memverifikasi keberadaan dan fungsionalitas elemen-elemen antarmuka, seperti tombol, label, input form, dan komponen lainnya. Jika semua elemen yang diuji tampil sesuai ekspektasi dan dapat diakses dengan benar, maka langkah pengujian akan dianggap *pass*.

Sebaliknya, jika elemen tidak ditemukan, teks tidak sesuai, tombol tidak dapat diklik, atau UI tidak merespons seperti yang diharapkan, maka sistem akan mencatat status *fail* pada langkah tersebut. Informasi ini kemudian ditampilkan dalam laporan hasil pengujian otomatis, yang dapat digunakan untuk melakukan analisis lebih lanjut terhadap defect atau error yang terjadi.

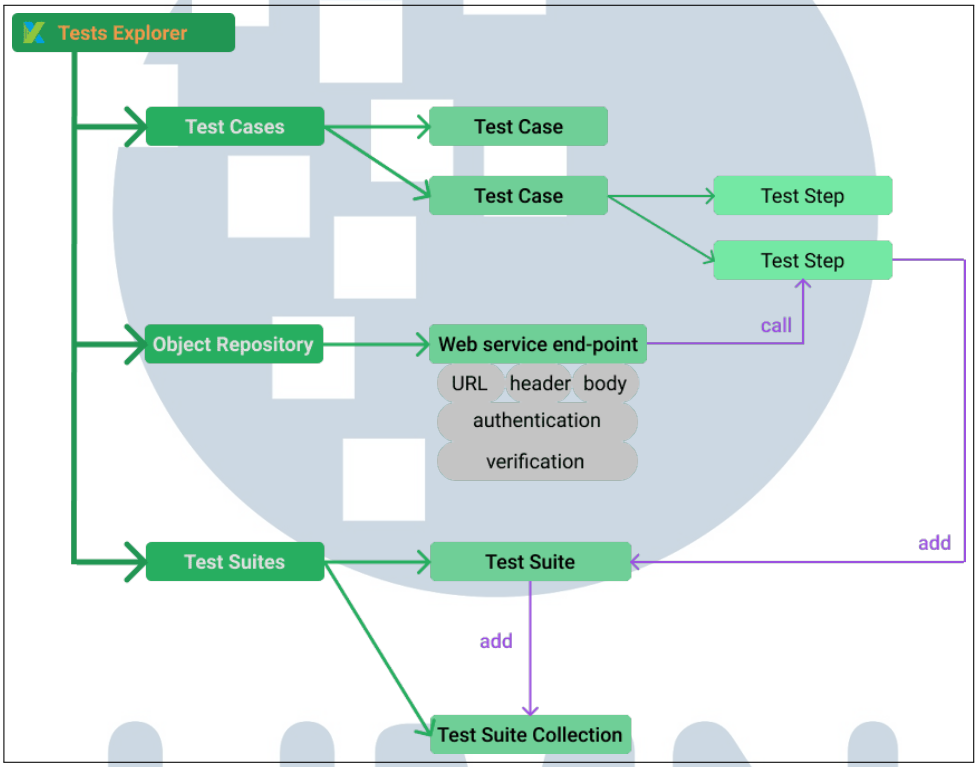
Dengan adanya beberapa skenario dalam satu file, proses pengujian menjadi lebih efisien dan konsisten, serta meminimalkan ketergantungan pada intervensi manual dari manusia. Proses ini sangat berguna terutama dalam pengujian berulang atau regresi, di mana seluruh skenario dapat dijalankan secara otomatis untuk memastikan bahwa semua fungsionalitas tetap berjalan dengan baik setelah ada perubahan pada sistem. Pemahaman diatas merupakan dasar utama dalam praktik testing automation *frontend*, khususnya saat proses onboarding ini. Penguasaan konsep ini menjadi dasar untuk memahami alur kerja pengujian otomatis, mulai dari penulisan skenario hingga eksekusi pengujian secara efisien dan konsisten dalam ranah UI yang digunakan pada projek pertama Esign.

D *API Backend Automation Testing*

Penggunaan *automation* dalam pengujian API menjadi sangat penting dalam siklus pengembangan perangkat lunak modern, terutama ketika sistem semakin kompleks dan dinamis [11]. Dengan menerapkan *automation testing*, pengujian API dapat dilakukan secara konsisten, cepat, dan berulang. *Automation* juga memungkinkan integrasi pengujian ke dalam pipeline *Continuous Integration/Continuous Deployment* (CI/CD), sehingga setiap perubahan pada kode dapat langsung diuji dan divalidasi. Selain itu, *automation testing* mendukung pengujian skenario negatif, validasi skema respons, *load testing*, dan regresi secara efisien.

API automation testing meliputi pembuatan skrip pengujian yang mencakup *Object Repository* untuk API, *Test Case* untuk menjalankan skrip pengujian,

validasi *backend* melalui basis data, serta validasi *response* untuk memastikan kesesuaian alur kerja API. Pengujian ini mencakup API internal dan eksternal, di mana API internal merupakan API yang terintegrasi langsung dengan aplikasi web. Struktur pengujian API memiliki alur tersendiri dalam penggunaan *framework* Katalon seperti ditunjukkan pada Gambar 3.4.



Gambar 3.5. Alur API testing melalui Katalon

Sumber: [12]

1. *Object Repository*
   Object Repository dalam konteks pengujian *API* menyimpan semua *endpoint* layanan web beserta informasi lengkap seperti metode *request*, URL, body API, *header*, dan autentikasi.

2. *Test Case*
   Test case berisi skenario pengujian yang terdiri dari beberapa langkah untuk merepresentasikan alur uji tertentu. Setiap *test case* dikelompokkan dalam satu folder dan dapat dijalankan secara individu menggunakan *execution profile* yang telah ditentukan.

3. *Test Suite*
   Test suite digunakan untuk kumpulan dari beberapa *test case* yang digunakan

untuk menguji fungsionalitas tertentu secara menyeluruh jika tidak ingin melalui pembuatan *test case* satu per satu. Selain itu, hasil pengujian juga dihasilkan secara otomatis pada level *test suite* dengan semua *API test case* yang dipanggil.

Berikut ini adalah contoh alur serta pseudocode untuk pengujian API yang diterapkan.

1. Skrip Test Case

   Dalam proses pengujian API, setiap skenario umumnya dibagi menjadi dua jenis test case, yaitu test case positif dan test case negatif. Test case positif dirancang untuk memastikan bahwa sistem bekerja sebagaimana mestinya ketika diberikan input yang valid atau sesuai ekspektasi. Sebaliknya, test case negatif digunakan untuk menguji bagaimana sistem merespons ketika menerima input yang tidak valid, tidak lengkap, atau tidak sesuai, dengan tujuan untuk memverifikasi terhadap kesalahan atau penyalahgunaan. Alur sebuah test case diterapkan seperti pada pseudocode berikut.

   (a) Case Positif: Login Sukses

```
1
2  SET login_endpoint TO "https://api.url.com/auth/login"
3  SET username TO "validUser"
4  SET password TO "correctPassword"
5
6  SEND POST request TO login_endpoint WITH username AND
       password
7  RECEIVE response
8
9  IF response.status_code IS 200 THEN
10     IF response.message CONTAINS "Login successful" THEN
11         PRINT "Login berhasil"
12     ELSE
13         PRINT "Status 200, tapi pesan tidak sesuai"
14     ENDIF
15 ELSE
16     PRINT "Login gagal, status code:", response.
       status_code
17 ENDIF
```

Kode 3.3: Pseudocode kasus positif login sukses

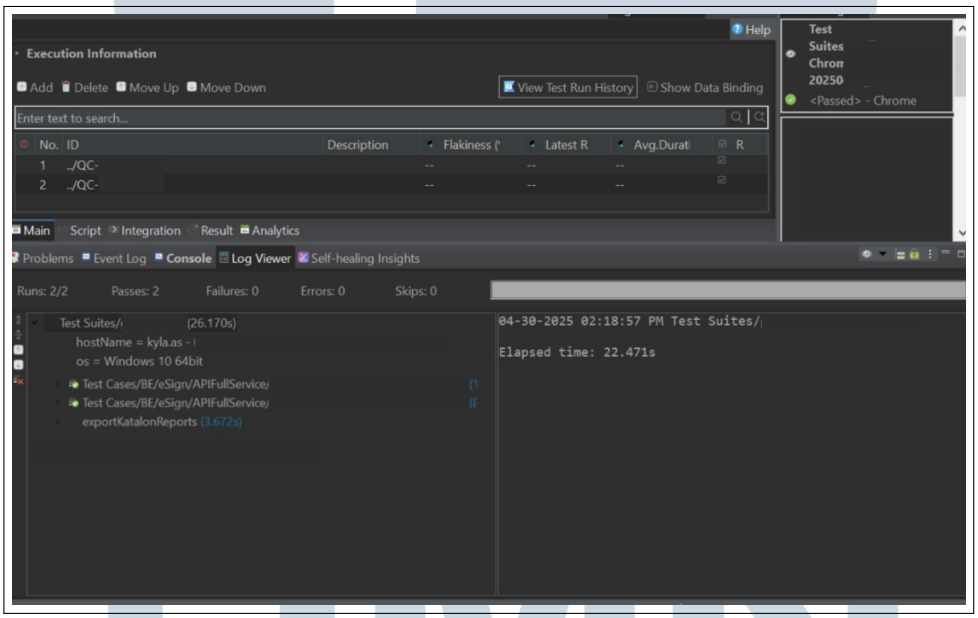
   (b) Case Negatif: Password Salah

```
1
2  SET login_endpoint TO "https://api.url.com/auth/login"
3  SET username TO "validUser"
4  SET password TO "wrongPassword"
5
6  SEND POST request TO login_endpoint WITH username AND
       password
7  RECEIVE response
```

```
8
9  IF response.status_code IS 401 THEN
10     IF response.message CONTAINS "Invalid username or
       password" THEN
11         PRINT "Pesan error sesuai"
12     ELSE
13         PRINT "Status 401, tapi pesan tidak sesuai"
14     ENDIF
15 ELSE
16     PRINT "Status code tidak sesuai:", response.
       status_code
17 ENDIF
```

Kode 3.4: Pseudocode kasus negatif *password* salah

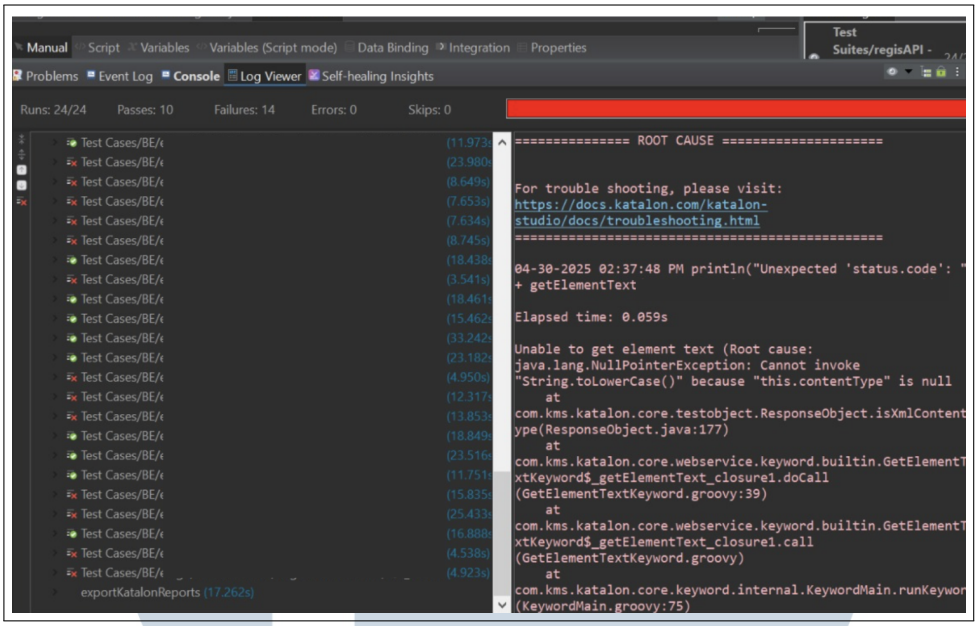
2. Tampilan Test Suite Positif



Gambar 3.6. Test Suite Positive

Pada Gambar 3.5 ditampilkan hasil eksekusi *test suite* yang menunjukkan status sukses. Seluruh *test case* yang dijalankan pada proses tersebut berhasil dieksekusi tanpa adanya *error*, serta menghasilkan *output* yang sesuai dengan ekspektasi. Setiap hasil dari *test case* juga telah melewati proses verifikasi, baik terhadap *response* dari *API* maupun terhadap data yang tersimpan di *database*, sehingga dapat dipastikan bahwa fungsionalitas yang diuji berjalan sesuai dengan ketentuan yang telah ditentukan. Keberhasilan ini menunjukkan bahwa alur pengujian berjalan dengan stabil dan dapat dijadikan acuan dalam pengujian selanjutnya.

3. Tampilan Test Suite Negatif



Gambar 3.7. Test Suite Negative

Pada Gambar 3.6 ditampilkan contoh hasil eksekusi dengan status gagal (*failed case*), di mana beberapa *test suite* yang dijalankan menghasilkan *error*. Katalon akan secara otomatis menampilkan detail letak kesalahan melalui berkas *HTTP Archive (HAR)*, yang berisi rekaman lengkap dari aktivitas permintaan dan respons selama pengujian. Selain itu, tampilan laporan pada Katalon juga memperlihatkan secara jelas *test case* mana yang berhasil *pass* dan mana yang gagal (*fail*), sehingga memudahkan untuk mengidentifikasi dan memprioritaskan perbaikan pada *test case* yang mengalami kegagalan terlebih dahulu secara spesifik.

3.3.2 Pembangunan Skrip Automation untuk Sistem Tanda Tangan Otomatis (Esign)

Proyek pertama yang dikerjakan selama masa magang adalah pembuatan sistem *automation testing* untuk aplikasi web (*Web App*) sistem tanda tangan otomatis milik PT Adicipta Inovasi Teknologi, Esign. Proses pembangunan skrip pengujian mencakup tiga area eksplorasi utama, yaitu pengujian *database* menggunakan PostgreSQL untuk memastikan integritas dan konsistensi data, pengujian *User Interface (UI)* untuk memverifikasi tampilan serta interaksi pengguna pada antarmuka aplikasi, dan pengujian API untuk memastikan konektivitas serta keakuratan *response* dari layanan yang digunakan. Pendekatan

ini dilakukan agar pengujian dapat mencakup seluruh lapisan sistem secara menyeluruh.

A Verifikasi Database

Keterlibatan *database* digunakan untuk melakukan perubahan pengaturan serta untuk memverifikasi data yang tersimpan atau mengalami perubahan. Dalam Katalon Studio, proses validasi terhadap data di *database* dapat dilakukan melalui pemanggilan fungsi yang ditulis di dalam berkas *Keywords*. Fungsi ini kemudian dapat digunakan dalam setiap *test case* untuk memvalidasi hasil dari masing-masing API yang diuji. Berikut adalah beberapa *keywords* yang digunakan untuk kebutuhan *API testing*.

1. Pseudocode validasi struktur dan kesesuaian nilai dengan data pada API

```
function verifyColumnStructureAndValuesMatchBetweenTables():
    connect to database
    define list of key columns to validate

    for each column in list:
        check if column exists in table_A
        if not exist:
            mark test failed and return false

        check if column exists in table_B
        if not exist:
            mark test failed and return false

    log: all columns exist

    query latest joined record from table_A and table_B
    if no data found:
        mark warning and exit

    for each row in result:
        compare values of each column between table_A and table_B
        if mismatch:
            log warning
        else:
            log info match successful

    close all connections
    return true if all matched
```

Kode 3.5: Pseudocode validasi struktur dan kesesuaian nilai kolom

Pseudocode 3.5 bertujuan untuk melakukan validasi struktur kolom serta kesesuaian nilai data antara dua tabel dalam basis data. Proses dimulai dengan melakukan koneksi ke *database* dan mendefinisikan daftar kolom yang akan divalidasi. Setiap kolom akan diperiksa apakah keberadaannya ada pada kedua tabel. Jika salah satu kolom tidak ditemukan pada salah satu tabel, proses akan dihentikan dan dianggap gagal. Setelah memastikan bahwa semua kolom tersedia,

sistem akan mengambil satu baris data terbaru dari hasil join antara kedua tabel. Selanjutnya, nilai dari setiap kolom dibandingkan antara tabel pertama dan tabel kedua. Jika semua nilai cocok, maka data dianggap valid, namun jika ditemukan ketidaksesuaian, akan ditampilkan peringatan. Seluruh koneksi ke database ditutup di akhir proses. Prosedur ini dapat digunakan dalam testcase untuk menjamin integritas data dan konsistensi antar entitas yang saling terhubung.

2. Pseudocode untuk mencari data yang belum terverifikasi

```
function getTotalPendingDocuments(userEmail):
    connect to database

    prepare query:
        count documents
        where user email matches
        and document not yet signed
        and active and valid tenant

    execute query with userEmail as parameter

    if result exists:
        return total count
    else:
        log: no result found

    close all resources
    return count
```

Kode 3.6: Pseudocode menghitung mencari dokumen belum diverfikasi

Pseudocode 3.6 digunakan untuk menghitung jumlah dokumen yang belum terverifikasi oleh seorang pengguna berdasarkan email. Langkah awal dilakukan dengan menghubungkan sistem ke *database*, kemudian menyusun dan menjalankan *query SQL* yang menghitung total dokumen dengan beberapa kondisi, seperti belum ditandatangani, milik pengguna tertentu, aktif, dan belum diperbarui. Parameter email digunakan untuk memfilter data agar hanya mencakup milik pengguna yang bersangkutan. Hasil *query* akan dibaca dan, jika terdapat data, jumlah total dokumen yang belum ditandatangani akan dikembalikan. Jika tidak ditemukan hasil, sistem akan memberikan informasi bahwa tidak ada data. Pada akhir proses, semua koneksi dan objek terkait *database* ditutup. Prosedur ini berguna untuk memastikan bahwa jumlah dokumen tertunda yang ditampilkan kepada pengguna sudah sesuai de

B Test Case Frontend

Pada projek pertama, proses pembangunan skrip testing lebih dominan pada pengujian *frontend*, penerapannya sesuai dengan tahapan yang dilampirkan pada

gambar 3.3. Berikut adalah contoh skrip groovy untuk pengecekan menu fitur management dalam pengecekan sistem aplikasi.

```
Class PengaturanPeranTest

    Function muatKonfigurasiJson(namaBerkas)
        lokasiFile = "JSON/" + namaBerkas
        return parsingJsonDariBerkas(lokasiFile)

    @Given "Pengguna berada di halaman utama"
    Function bukaHalamanUtama()
        Arahkan ke URL utama aplikasi

    @Then "Pengguna membuka menu Pengaturan Peran"
    Function aksesMenuPengaturanPeran()
        Klik elemen menu "pengaturan_peran"

    @Then "Label Pengaturan Peran ditampilkan"
    Function verifikasiLabelHalaman()
        Verifikasi bahwa label "Pengaturan Peran" tampil di
   halaman

    @Then "Pengguna memverifikasi komponen UI pada menu Pengaturan
     Peran"
    Function verifikasiKomponenHalaman()
        Verifikasi label Nama Peran, Status, dan Grup muncul
        Verifikasi tombol "Tambah", "Cari", dan "Atur Ulang" dapat
     diklik

    @Then "Pengguna melakukan pencarian dengan kondisi {kondisi}"
    Function pencarianBerdasarkanKondisi(kondisi)
        switch kondisi:
            case "Tidak Ada Hasil":
                Isi kolom grup dengan nilai tidak valid
                Klik tombol cari
                Verifikasi pesan "Tidak ditemukan" tampil

            case "Grup Tidak Terdaftar":
                Isi kolom grup dengan nilai acak
                Tekan Enter
                Klik tombol cari
                Verifikasi pesan "Grup tidak terdaftar" tampil
                Klik tombol atur ulang

            case "Grup Tidak Aktif":
                Isi grup dengan data valid
                Pilih status "Tidak Aktif"
                Klik tombol cari
                Klik tombol atur ulang

            case "Pencarian Valid":
                Isi grup dengan data valid
                Klik tombol cari
                Klik tombol atur ulang

            case "Peran Ganda":
                data = muatKonfigurasiJson("dataPengaturanPeran.
   json")
                for entri in data:
                    Isi input nama peran dengan entri.nama
                    Klik tombol cari
                    jika hasil tidak ditemukan:
                        Cetak "Tidak ditemukan"
                    jika hasil ditemukan:
                        Verifikasi bahwa hasil sesuai nama
```

```
                    Klik tombol atur ulang

    @When "Pengguna menambahkan akses ke modul"
    Function tambahModul()
        Pilih modul A, B, dan C sesuai kebutuhan

    @When "Pengguna menambahkan hak akses dinamis"
    Function tambahHakAksesDinamis()
        Centang hak akses berdasarkan data dinamis

End Class
```

Kode 3.7: Pseudocode pengujian pengaturan peran dalam format BDD

Pseudocode ini merupakan representasi logis dari langkah-langkah pengujian sistem manajemen yang dibutuhkan dalam sebuah aplikasi. Alur ini juga menggambarkan pola umum yang digunakan untuk fitur-fitur lain yang terdapat di dalam sistem. Tujuan utama dari pengujian ini adalah untuk memastikan bahwa antarmuka pengguna pada modul manajemen dapat diakses dengan baik, seluruh komponen tampil sesuai yang diharapkan, fitur pencarian berjalan sesuai dengan skenario yang ditentukan, serta sistem mampu mengelola peran, termasuk dalam hal penambahan akses menu.

Langkah awal yang umumnya digunakan dalam berbagai skenario adalah memuat data dari berkas konfigurasi JSON. Langkah ini penting untuk mendukung skenario dinamis, seperti pengecekan terhadap peran ganda. Setelah itu, pengguna diarahkan untuk membuka menu yang relevan, lalu dilakukan verifikasi terhadap label, tombol, serta elemen-elemen kunci lainnya untuk memastikan semuanya dapat diakses dan berfungsi.

Selanjutnya dilakukan pengujian berdasarkan berbagai kondisi, seperti data yang tidak tersedia, *tenant* yang tidak terdaftar, *tenant* yang tidak aktif, kondisi normal, hingga kasus dengan nama peran yang sama. Setiap kondisi diuji melalui pengisian formulir, interaksi dengan tombol-tombol pada antarmuka, dan validasi hasil yang ditampilkan pada halaman.

Pseudocode ini juga mencakup skenario pengelolaan hak akses peran, baik melalui penambahan menu secara statis (dengan memilih opsi yang telah ditentukan) maupun secara dinamis (dengan menggunakan nilai input yang bervariasi). Dengan demikian, pengujian ini tidak hanya fokus pada aspek tampilan, namun juga mencakup logika bisnis dan integrasi data dalam sistem.

Skrip dirancang menggunakan pendekatan *modular*, dengan setiap langkah pengujian direpresentasikan dalam bentuk fungsi atau step terpisah yang dipanggil dari file *feature*. Setiap skenario pengujian dimulai dengan pemanggilan data dari file konfigurasi berformat JSON, yang berisi parameter uji seperti nama

tenant, peran pengguna, dan status aktivasi. Setelah data dimuat, script akan membuka antarmuka yang relevan melalui perintah navigasi, lalu memverifikasi tampilan komponen seperti label, tombol aksi, dan form input. Verifikasi ini dilakukan menggunakan perintah seperti *verifyElementPresent*, *verifyElementText*, atau *verifyMatch*, tergantung jenis elemen yang diuji.

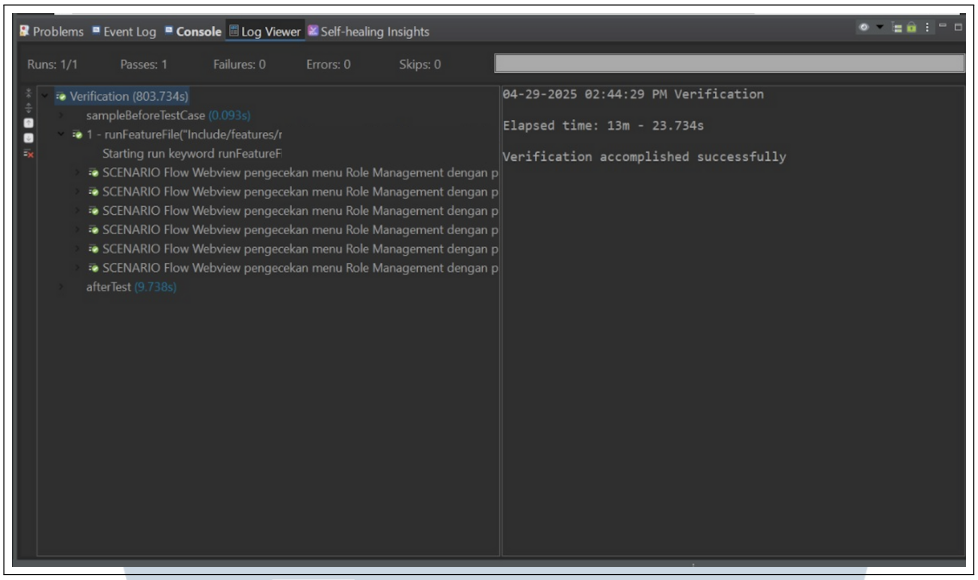
Kemudian *test case* yang telah digunakan dipanggil ke dalam *feature file* sebagai berikut.

```
1
2  Scenario 1() {
3      Given Pengguna masuk dengan akun "admin"
4      Then Pengguna membuka menu "Pengelolaan"
5      Then Pengguna memverifikasi komponen UI pada menu Pengaturan
       Peran
6
7      Then Pengguna melakukan pencarian dengan kondisi "Tidak Ada"
8      Then Pengguna melakukan pencarian dengan kondisi "Grup Tidak
       Terdaftar"
9      Then Pengguna melakukan pencarian dengan kondisi "Tidak Aktif"
10     Then Pengguna melakukan pencarian dengan kondisi "Normal"
11
12     Then Pengguna mengambil nama peran ganda dari sistem
13     Then Pengguna melakukan pencarian dengan kondisi "Peran Sama"
14     Then Pengguna melakukan pencarian tanpa kondisi khusus
15
16     When Pengguna masuk sebagai "klien"
17     Then Pengguna memverifikasi tidak ada akses ke menu Pengaturan
        Peran
18 }
19
20 Scenario 2() {
21     Given Pengguna masuk dengan akun "admin"
22     Then Pengguna membuka menu "Pengaturan Peran"
23     Then Pengguna memverifikasi komponen UI pada menu Pengaturan
       Peran
24
25     Then Pengguna mengatur status grup menjadi tidak aktif
26     Then Pengguna memverifikasi grup tidak aktif muncul dalam
       daftar
27
28     Then Pengguna mengatur status grup menjadi aktif
29     Then Pengguna memverifikasi grup tidak aktif tidak muncul
30
31     Then Pengguna melakukan pencarian dengan kondisi "Normal"
32
33     Then Pengguna menambahkan peran baru tanpa mengisi data
34     Then Pengguna membatalkan penambahan peran baru
35     Then Pengguna menambahkan peran baru dengan data valid
36
37     Then Pengguna memverifikasi peran muncul dalam daftar
38     Then Pengguna mengatur ulang filter pencarian
39
40     Then Pengguna mengubah peran lalu membatalkan
41     Then Pengguna menyimpan peran tanpa perubahan
42     Then Pengguna memverifikasi peran tetap ada
43
44     Then Pengguna mengubah nama peran lalu membatalkan
45     Then Pengguna mengubah nama peran dan menyimpan
46     Then Pengguna memverifikasi nama peran "Peran Baru" dalam
       sistem
47 }
48
49 Scenario 3() {
```

```
    Given Pengguna masuk dengan akun "admin"
    Then Pengguna membuka menu "Pengelolaan"
    Then Pengguna memverifikasi komponen UI pada menu Pengaturan
   Peran

    Then Pengguna melakukan pencarian dengan kondisi "Kelola Peran
   "
    Then Pengguna memverifikasi tampilan menu umpan balik terlihat

    Then Pengguna menonaktifkan salah satu menu
    Then Pengguna melakukan pencarian dengan kondisi "Kelola Peran
   "
    Then Pengguna memverifikasi menu umpan balik tidak terlihat

    Then Pengguna mengaktifkan kembali menu tersebut
    Then Pengguna menonaktifkan opsi kelola untuk menu tersebut
    Then Pengguna melakukan pencarian dengan kondisi "Kelola Peran
   "
    Then Pengguna memverifikasi menu umpan balik tidak terlihat

    Then Pengguna mengaktifkan kembali opsi kelola
    Then Pengguna melakukan pencarian dengan kondisi "Kelola Peran
   "
    Then Pengguna memverifikasi menu umpan balik terlihat

    Then Pengguna melakukan pencarian dengan kondisi "Kelola Peran
   "
    Then Pengguna memverifikasi tidak ada menu yang ditampilkan

    Then Pengguna melakukan pencarian dengan kondisi "Admin Klien"
    Then Pengguna memverifikasi menu untuk Admin Klien ditampilkan
}
```

Kode 3.8: Pseudocode *feature file* pengaturan peran

Pemanggilan steps dari feature file memungkinkan kode bersifat dinamis dan mudah untuk digunakan kembali di masa mendatang, bahkan jika terdapat versi baru. Hal ini meningkatkan fleksibilitas dan efisiensi dalam pengembangan skenario pengujian. Pada kode 3.8, ditunjukkan contoh penerapan langkah dinamis yang digunakan dalam tiga skenario berbeda. Meskipun skenarionya berbeda, langkah pengujiannya tetap menggunakan step yang sama, hanya nilai input-nya saja yang berbeda. Pendekatan ini sangat bermanfaat untuk menjaga konsistensi dan mengurangi duplikasi kode dalam pengujian otomatis. Tampilan passed atau sukses ditampilan pada gambar berikut.



Gambar 3.8. Hasil testing passed API *frontend*

Setiap langkah pengujian diikuti dengan proses validasi untuk memastikan sistem merespons sesuai dengan ekspektasi agar menampilkan pass testcase. Dalam kasus ketika pengguna memasukkan data peran yang tidak tersedia, script akan memverifikasi bahwa pesan error yang sesuai ditampilkan. Validasi dilakukan menggunakan *assert* dan *verify* dari framework Katalon, yang membandingkan hasil aktual di antarmuka dengan nilai yang diharapkan. Untuk kasus pengelolaan hak akses, skrip melakukan validasi terhadap status checkbox, label menu yang ditampilkan, serta struktur hierarki menu yang muncul setelah perubahan dilakukan. Hasil validasi ini dicatat dalam laporan otomatis yang dihasilkan oleh framework melalui console.

C Backend API Testing

Dalam pembangunan skrip, skenario dibagi ke dalam dua kategori utama, yaitu *positive case* dan *negative case*. Pengujian ini mencakup validasi menyeluruh yang tidak hanya dilakukan pada sisi *backend*, tetapi juga melibatkan dua jenis API, yakni API internal dan API eksternal. *Positive case* digunakan untuk memastikan bahwa sistem memberikan respons yang sesuai dengan ekspektasi, yaitu *response code 200* yang menandakan keberhasilan. Sementara itu, *negative case* difokuskan pada pengujian terhadap kondisi yang tidak valid atau *error*, untuk memastikan sistem memberikan *response code* selain 200, sebagai bentuk penanganan kesalahan mendatang.

```
def requestInvalidURL = findTestObject('Path/To/APIObject', [
    config.param1,
    config.param2
])
requestInvalidURL.setRestUrl('https://invalid.url.example.com/
    service/checkStatus')

def response1 = WS.sendRequest(requestInvalidURL)
int expectedStatus1 = 404
String expectedMessage1 = "404 Not Found"

if (WS.verifyResponseStatusCode(response1, expectedStatus1)) {
    println("Expected response received: ${expectedMessage1}")
} else {
    println("Unexpected status code: " + response1.getStatusCode()
    )
}
```

Kode 3.9: Pseudocode kasus negatif

Kode 3.9 merupakan salah satu penerapan yang digunakan secara umum untuk negative case. Langkah pertama dimulai dengan mendefinisikan request object yang merepresentasikan API yang akan dipanggil. Setelah objek tersebut terbentuk, URL endpoint dari permintaan diubah secara manual menjadi alamat yang tidak benar. Selanjutnya, permintaan dikirim menggunakan fungsi WS.sendRequest, dan hasilnya disimpan ke dalam variabel response1. Dalam pengujian ini, expected response code yang diharapkan adalah 404 yang berarti Not Found, mengindikasikan bahwa alamat tujuan tidak ditemukan. Tujuan dari pengujian ini adalah untuk memastikan sistem dapat menangani kesalahan URL dengan tepat dan memberikan respon yang sesuai.

```

def requestInvalidRef = findTestObject('Path/To/APIObject', [
    config.param1,
    config.invalidParam1
])
def response = WS.sendRequest(requestInvalidRef)

if (response.getStatusCode() == 200) {
  WS.verifyElementPropertyValue(response, 'status.code', 4001)
  WS.verifyElementPropertyValue(response, 'status.message', '
    Document not found')
  println("Invalid Ref Number test PASSED")
} else {
  println("Unexpected status code: " + response.getStatusCode())
  assert response.getStatusCode() == 200
}
```

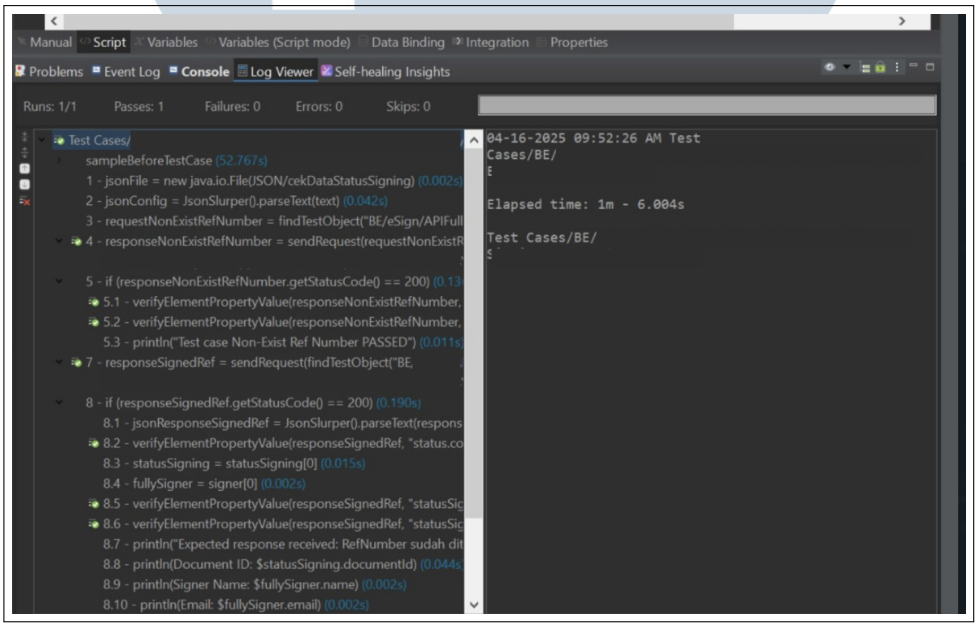
Kode 3.10: Pseudocode kasus negatif *invalid reference number*

Pada skenario kedua ini, pada Kode 3.10 dilakukan pengujian *negative case* yang lebih spesifik terhadap referensi dokumen yang tidak valid. Pengujian dimulai dengan pembuatan objek permintaan API menggunakan *findTestObject*, di mana parameter yang digunakan adalah kombinasi dari nilai yang valid dan nilai referensi yang tidak valid. Permintaan tersebut kemudian dikirim ke *endpoint* menggunakan

sendRequest, dan responsnya disimpan dalam variabel *response*.

Meskipun nilai referensi salah, sistem tetap memberikan *status code* 200, yang berarti permintaan diproses dengan sukses, namun isi dari respons menunjukkan bahwa dokumen tidak ditemukan. Oleh karena itu, dilakukan verifikasi terhadap isi respons, dengan memeriksa apakah *status.code* bernilai 4001 dan *status.message* menyatakan ”Document not found”. Bila kedua nilai ini sesuai, maka pengujian dianggap berhasil menangani kasus referensi tidak valid. Jika *status code* yang diterima tidak sesuai (bukan 200), maka akan dicetak pesan bahwa status yang diterima tidak sesuai dan dilakukan *assert* untuk memastikan nilai status memang seharusnya 200.

Pengujian ini digunakan untuk memastikan sistem memberikan respons logis dan pesan kesalahan yang tepat ketika diberikan input yang tidak dikenal. Berikut adalah tampilan jika *test case* API sukses dijalankan.



Gambar 3.9. Hasil testing *passed* API *backend*

Penyusunan skrip dilakukan secara bertahap melalui proses iteratif dan pengembangan berkelanjutan (*enhancement*) setiap kali terdapat rilis baru. Setiap perubahan atau penyempurnaan yang dilakukan bertujuan untuk meningkatkan kualitas dan cakupan pengujian. Setelah skrip mencapai tingkat stabilitas dan kelengkapan yang memadai, hasil akhirnya akan diintegrasikan ke tahap pengembangan produk (*product development*) sebagai bagian dari *pipeline* otomatisasi yang siap digunakan dalam lingkungan produksi.

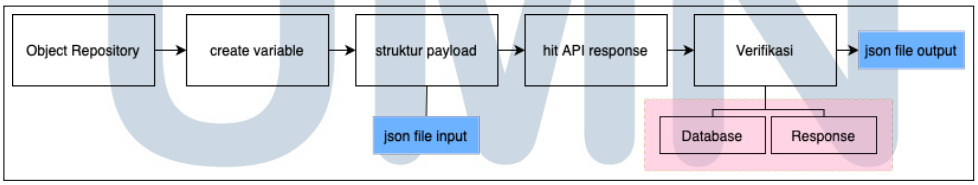
3.3.3 Pembangunan Skrip Automation untuk Core Multifinance System (CONFINS)

Proyek kedua mencakup pembangunan skrip otomasi untuk pengujian backend API pada produk *core multifinance system* milik Adins (CONFINS). Berbeda dengan proyek pertama yang berfokus pada pengujian produk secara menyeluruh, pengujian dalam proyek ini dikhususkan untuk API. Pada periode ini, *test case* dibagi menjadi dua kategori, yaitu *test case* per API dan *test case* untuk *main flow*.

Sistem *automation testing* untuk *test case* API memungkinkan fleksibilitas dan sifat dinamis berkat adanya *object repository* serta variabel yang dapat diubah sesuai kebutuhan. Karena pengujian ini mencakup berbagai *server* dan *base URL* yang berbeda, pergantian *server* dapat dilakukan dengan mudah melalui Katalon. Dengan memanfaatkan *global variable* yang telah disetting, pergantian server ini secara otomatis akan menyesuaikan nilai variabel yang digunakan dalam pengujian.

A Test Case Individual API

Individual testing mencakup satu API yang akan dibangun test case-nya secara khusus. Pembuatan test case ini akan mempermudah pengujian pada alur main flow, karena jika suatu prosedur API memerlukan pemanggilan beberapa API sebelumnya, maka prosedur tersebut cukup memanggil test case yang sudah tersedia. Berikut adalah contoh alur pembuatan test case API.



Gambar 3.10. Alur pembangunan test case API

Berbeda dari proyek sebelumnya, pada implementasi kali ini seluruh value yang terdapat di dalam body API telah disesuaikan menjadi nilai dinamis (*dynamic value*) seperti pada kode 3.11. Pendekatan ini diterapkan untuk meningkatkan fleksibilitas dan kemudahan dalam melakukan pengujian ulang (*reusability testing*) di masa mendatang. Dengan menjadikan seluruh data bersifat dinamis, pengujian dapat dilakukan untuk berbagai skenario tanpa perlu melakukan penyesuaian manual terhadap isi request setiap kali dilakukan perubahan data. Selain itu, penggunaan *dynamic value* juga membantu mempercepat proses pembuatan test

case, mempermudah pemeliharaan skrip otomatisasi, serta mendukung pengujian yang lebih stabil dan efisien secara keseluruhan.

```
BEGIN ConstructDynamicRequestBody
  SET Id = "${Id}"
  SET Descr = "${Descr}"
  SET Notes = "${Notes}"

  INITIALIZE ListDocument AS ARRAY
    CREATE Document
      SET No = "${No}"
      SET Name = ARRAY OF "${Name}"
    ADD Document TO ListDocument

  SET RequestTime = "${RequestTime}"

  COMBINE Id, Descr, Notes, ListDocument, RequestTime INTO
   RequestBody
RETURN RequestBody
END
```

Kode 3.11: Pseudocode *dynamic value body object repository*

Setelah menerapkan object repository beserta variabel-variabel yang telah disesuaikan, object tersebut dapat langsung digunakan di dalam test case. Penggunaannya mencakup pengecekan response maupun penginputan data, baik melalui format JSON, database, maupun default variabel yang telah disiapkan sebelumnya. Pendekatan ini mempermudah integrasi serta meningkatkan efisiensi dalam proses automasi pengujian. Berikut adalah contoh test case individual API yang digunakan untuk menambah data customer.

```
BEGIN SimulateCustomerRegistration

  INITIALIZE actionHelper = new UtilityAction()
  INITIALIZE dateHelper = new UtilityDate()
  INITIALIZE dbVerifier = new DatabaseVerifier()
  INITIALIZE dbQuery = new DataSource()

  SET currentDate = dbQuery.getCurrentDate()

  SET idNumber = actionHelper.generateRandomId()
  SET emailAddress = actionHelper.generateRandomEmail()
  SET shortName = actionHelper.generateShortName()
  SET fullName = actionHelper.generateFullName()
  SET phone1 = actionHelper.generatePhoneNumber()
  SET phone2 = actionHelper.generatePhoneNumber()
  SET phone3 = actionHelper.generatePhoneNumber()
  SET birthDate = dateHelper.generateRandomBirthDate()

  CREATE requestPayload WITH dynamicFields INCLUDING:
    identification, names, contactNumbers, addressData,
    customerFlags, relationshipData, and currentDate

  SEND request TO targetAPIEndpoint WITH requestPayload

  VERIFY response.statusCode == 200

  IF response is valid THEN
    PARSE responseBody
    ASSERT response contains expected message, ID, and no errors

```

```
    FETCH customerRecord FROM dbVerifier USING response ID

    IF customerRecord exists THEN
      VERIFY customerRecord fields MATCH requestPayload values
      DISPLAY "Verification successful"
    ELSE
      DISPLAY "Database verification failed"
    ENDIF

    EXTRACT responseMetadata: rowVersion, responseTime, status,
   message, errorList

    COMBINE all request + response data INTO structuredResult

    IF params is not defined THEN
      INITIALIZE params as empty map
    ENDIF

    IF params.collectMode is TRUE THEN
      RETURN structuredResult
    ELSE
      READ existing JSON data from storageFile
      OVERWRITE with new structuredResult (latest only)
      WRITE updated data to storageFile
      DISPLAY confirmation of save
    ENDIF

    RETURN structuredResult

  ENDIF

END
```

Kode 3.12: Pseudocode *customer registration process*

Melalui Kode 3.12, proses pengujian ini dimulai dengan inisialisasi beberapa *class* yang digunakan untuk menghasilkan data dinamis, seperti nomor identitas, nama, email, nomor telepon, dan tanggal lahir. Data ini kemudian dikombinasikan bersama sejumlah parameter lainnya ke dalam struktur *payload* yang akan digunakan untuk melakukan permintaan (*request*) ke *endpoint* API. Seluruh *field* dalam *payload* disusun secara dinamis agar dapat digunakan kembali dalam berbagai skenario pengujian.

Setelah *request* dikirim, sistem akan memverifikasi status respon untuk memastikan bahwa permintaan berhasil (*status code* 200). Jika berhasil, *response* akan diurai (*parse*) untuk memastikan pesan sesuai. Kemudian, dilakukan verifikasi antara data yang dikirim dengan data yang disimpan di *database* menggunakan ID dari respon. Verifikasi *database* dilakukan melalui pendekatan yang serupa, yaitu melalui *Keywords*.

```

DEFINE keyword: CheckCustomer(name)

    INITIALIZE database connection to DB
    INITIALIZE statement as null

    TRY:
        PREPARE insert statement into customer table
```

```
            (with fields such as name, contact, birth date, etc.)

        LOG: "Executing customer insert operation..."

        BIND input parameters to prepared statement
        EXECUTE the insert statement

        IF data successfully inserted THEN
            LOG: "Customer data inserted."
        ELSE
            LOG: "Insertion failed."

    CATCH any error:
        LOG: "Error occurred during database operation."
        ASSERT failure with error message

    FINALLY:
        IF statement exists THEN close statement
        IF connection exists THEN close connection

    RETURN operation status

END DEFINE
```

Kode 3.13: Pseudocode *insert and check customer data database*

Data hasil response dikombinasikan dengan data request untuk disimpan dalam struktur yang telah ditentukan. Jika mode tertentu diaktifkan, maka data akan dikembalikan untuk keperluan penggunaan lanjutan. Namun jika tidak, maka data akan disimpan ke dalam file JSON sebagai arsip. Penyimpanan ini ditujukan untuk penggunaan di masa mendatang.

B Main Flow API

API individual yang telah dibuat dalam masing-masing test case kemudian akan digabungkan dan digunakan dalam satu alur (flow). Alur ini mencakup rangkaian lengkap dari seluruh API yang diperlukan untuk satu skenario pengujian secara menyeluruh. Pendekatan ini memungkinkan pengujian dilakukan secara end-to-end, sehingga setiap tahapan dalam skenario dapat diuji sesuai urutan dan keterkaitannya.

```

TRY:

    INITIALIZE connection to  database

    CALL test case: Generate Customer Data
        PARAMETERS: true

    LOAD customer data from JSON file
    SELECT the first customer record

    IF customer data is missing or unreadable:
        MARK error and STOP process

    EXTRACT customer ID from data
```

```
16
17      LOG the customer ID being processed
18
19      QUERY customer number using the customer ID
20
21      LOG the retrieved customer number
22
23      CALL test case: Generate Agreement Number
24      CALL test case: Generate Application Number
25      CALL test case: Generate Asset Number
26
27      IF all generated numbers are available:
28          LOG the transaction numbers
29
30      ELSE:
31          MARK error and STOP process
32
33      CALL test case: Create Agreement
34          PARAMETERS: combination of customer number, agreement
        number, application number, asset number, and additional
        configuration data
35
36      BUILD final agreement number using the generated ID
37
38      CALL test case: Aktivasi dengan Agreement
39          PARAMETERS: agreement number
40
41  CATCH any exception:
42      LOG the error message that occurred during processing
43
44  FINALLY:
45      LOG that all processes have been completed
```

Kode 3.14: Pseudocode proses main flow

Pseudocode pada kode 3.14 menggambarkan alur utama proses pembuatan dan aktivasi pembuatan dasar agreement secara otomatis dalam sistem. Proses diawali dengan inisialisasi koneksi ke database yang digunakan untuk pengambilan dan verifikasi data pelanggan. Setelah itu, sistem menjalankan test case untuk menambahkan data pelanggan baru dengan parameter tertentu. Data kemudian dibaca dari file JSON, dan sistem memilih entri pertama dari data tersebut untuk diproses. Jika data pelanggan tidak tersedia atau gagal dibaca, maka proses langsung dihentikan dengan penandaan error.

Selanjutnya, sistem mengekstrak ID pelanggan dari data yang berhasil dimuat dan mencatatnya dalam log. Berdasarkan ID tersebut, sistem melakukan query ke database untuk mendapatkan nomor pelanggan (customer number), yang kemudian juga dicatat. Setelah nomor pelanggan didapatkan, sistem memanggil beberapa test case untuk menghasilkan nomor-nomor transaksi yang dibutuhkan, yaitu agreement number, application number, dan nomor aset asset number. Jika ketiga nomor tersebut berhasil digenerate, maka sistem melanjutkan proses dan mencatat informasi tersebut. Namun, jika salah satu gagal, maka proses akan dihentikan dan ditandai sebagai error.

Proses kemudian dilanjutkan dengan menjalankan test case selanjutnya menggunakan kombinasi data yang sudah dikumpulkan sebelumnya, termasuk informasi pelanggan dan seluruh nomor transaksi. Setelah perjanjian berhasil dibuat, sistem membentuk nomor perjanjian final (misalnya dengan menambahkan prefix tertentu), dan menggunakan nomor tersebut untuk menjalankan test case aktivasi, yang menandai bahwa perjanjian telah resmi aktif dalam sistem. Jika terjadi kesalahan pada proses manapun, maka sistem akan menangkap error-nya dan mencatat pesan kesalahan ke dalam log. Setelah seluruh proses selesai, baik sukses maupun gagal, sistem mencatat log akhir sebagai tanda bahwa semua tahapan telah dijalankan.

Untuk Setiap test case yang dipanggil untuk menjalankan main flow, akan memiliki alur tersendiri seperti berikut.

1. Test Case Generate Data Customer

```
TEST CASE: Add Customer Data
PARAMETERS: isAddSpouse (boolean)

BEGIN
    IF isAddSpouse == true THEN
        FILL form: customer data
    ELSE
        FILL form: customer data only
    ENDIF

    SUBMIT customer creation request
    RECEIVE response

    VERIFY response.statusCode == 200

    IF response is valid THEN
        PARSE response body
        ASSERT response contains expected customer ID and
    success message

        QUERY customer record from database using returned ID

        IF record exists THEN
            VERIFY fields match input values
            LOG "Verification successful"
        ELSE
            LOG "Customer data not found in database"
        ENDIF

        EXTRACT metadata: status, responseTime, message,
    errorList

        PACKAGE request and response into structured data
    format

        IF params is undefined THEN
            INIT empty params map
        ENDIF

        IF params.collectMode is true THEN
            RETURN structured data
        ELSE
```

```
            LOAD existing JSON
            REPLACE with new data
            SAVE to storage
            LOG "Data saved successfully"
        ENDIF
    ENDIF
END
```

Kode 3.15: Pseudocode *add customer data*

Pseudocode ini merepresentasikan proses pengujian untuk menambahkan data customer ke dalam sistem. Test case akan menerima parameter yang dibutuhkan untuk input data, lalu mengirim request ke sistem. Setelah respons diterima, dilakukan verifikasi bahwa status kode adalah 200 OK. Jika respons valid, respon akan diverifikasi dan berdasarkan ID hasil respons, dilakukan pencocokan data dengan informasi yang ada di database. Jika data ditemukan, diverifikasi bahwa seluruh field yang dimasukkan sesuai dengan data yang tersimpan. Data kemudian akan disimpan ke json untuk proses berikutnya.

2. Test Case Agreement Number

```
TEST CASE: Generate Agreement Number
PARAMETERS: officeCode, masterSeqCode

BEGIN
    REQUEST agreement number generation service
        INPUT: officeCode, masterSeqCode

    RECEIVE response

    VERIFY response.statusCode == 200

    IF response is valid THEN
        PARSE response
        ASSERT agreement number is not empty

        STORE metadata from response

        LOG and RETURN agreement number
    ELSE
        LOG "Failed to generate agreement number"
    ENDIF
END
```

Kode 3.16: Pseudocode *generate agreement number*

Pseudocode ini berfungsi untuk menghasilkan nomor transaksi baru berdasarkan parameter officeCode dan masterSeqCode. Sistem mengirim permintaan ke API lalu setelah respons diterima, diverifikasi bahwa status kode adalah 200, dan dilakukan parsing terhadap konten respons. Nomor transaksi yang dihasilkan akan diekstrak dan dicocokkan dengan format atau pola yang diharapkan. Jika valid, sistem akan menyimpan hasilnya

dan mengembalikannya sebagai output. Informasi lengkap dari request dan response juga akan disimpan ke dalam file json.

3. Test Case Application Number

```
TEST CASE: Generate Application Number
PARAMETERS: officeCode, masterSeqCode

BEGIN
    CALL application number generation API
    RECEIVE response

    VERIFY response.statusCode == 200

    IF response is valid THEN
        PARSE body
        EXTRACT application number
        ASSERT application number is generated
        RETURN it
    ELSE
        LOG "Application number generation failed"
    ENDIF
END
```

Kode 3.17: Pseudocode *generate application number*

Pseudocode ini menjelaskan alur pengujian otomatis untuk proses pembuatan Application Number. Pengujian dilakukan dengan cara memanggil API untuk menghasilkan nomor aplikasi berdasarkan parameter seperti officeCode dan masterSeqCode. Setelah permintaan dikirim, sistem akan menerima respons dan memverifikasi bahwa statusCode dari respons tersebut adalah 200, yang menandakan bahwa proses permintaan berhasil. Jika respons valid, maka isi dari body respons akan diparsing untuk mengekstrak nomor aplikasi yang dihasilkan. Selanjutnya adalah proses verifikasi lebih lanjut untuk memastikan bahwa Application Number benar-benar dihasilkan, sebelum kemudian dikembalikan sebagai output test case.

4. Test Case Asset Number

```
TEST CASE: Generate Asset Number
PARAMETERS: officeCode, masterSeqCode

BEGIN
    CALL asset number service with parameters
    RECEIVE response

    VERIFY response.statusCode == 200

    IF response is valid THEN
        PARSE and EXTRACT asset number
        ASSERT number is correctly formatted
        RETURN asset number
    ELSE
        LOG "Asset number creation failed"
    ENDIF
```

```
END
```

Kode 3.18: Pseudocode *generate asset number*

Pseudocode ini menggambarkan proses pengujian otomatis untuk pembuatan Asset Number, seperti test case sebelumnya, proses dimulai dengan memanggil pembuatan Asset Number menggunakan parameter officeCode dan masterSeqCode. Setelah itu, sistem akan menunggu respons dari API dan memverifikasi bahwa status kodenya adalah 200. Jika respons dinyatakan valid, maka isi dari respons akan diparsing, dan nomor aset akan diambil. Nomor yang diperoleh kemudian diuji apakah sudah diformat dengan benar dan akan digunakan untuk proses selanjutnya.

5. Test Case Create Agreement

```
TEST CASE: Create Agreement
PARAMETERS: customerNo, agreementNo, appNo, assetNo,
    configurationData

BEGIN
    COMPOSE request using all parameters
    SUBMIT to agreement creation service
    RECEIVE response

    VERIFY response.statusCode == 200

    IF response is valid THEN
        PARSE response
        EXTRACT agreement ID

        QUERY agreement data from database using ID
        IF match found THEN
            ASSERT database fields match request
            LOG "Agreement created and verified"
        ELSE
            LOG "Agreement not found in DB"
        ENDIF

        RECORD metadata: rowVersion, responseTime, status

        IF params.collectMode is true THEN
            RETURN structured result
        ELSE
            UPDATE storage with new result
            LOG "Stored agreement data"
        ENDIF
    ENDIF
END
```

Kode 3.19: Pseudocode *create agreement*

Pseudocode ini menggambarkan skenario pembuatan *agreement* atau perjanjian kontrak berdasarkan data input seperti nomor *customer*, nomor *agreement*, nomor aplikasi, dan nomor aset. Setelah data dipanggil, *request* dikirim ke sistem. *Response* dari sistem harus memiliki *status code* 200,

dan jika valid, akan diverifikasi bahwa *agreement* berhasil dibuat. Validasi dilakukan dengan mencocokkan data yang dikirim dengan informasi di *database*, untuk memastikan tidak ada perbedaan nilai dari dua sisi. Hasil *response* juga dikumpulkan dan hasil akhir disimpan dalam berkas JSON. Mode penyimpanan ditentukan berdasarkan parameter konfigurasi, apakah data dikumpulkan sebagai kumpulan atau hanya disimpan sebagai versi terakhir.

6. Test Case Activation

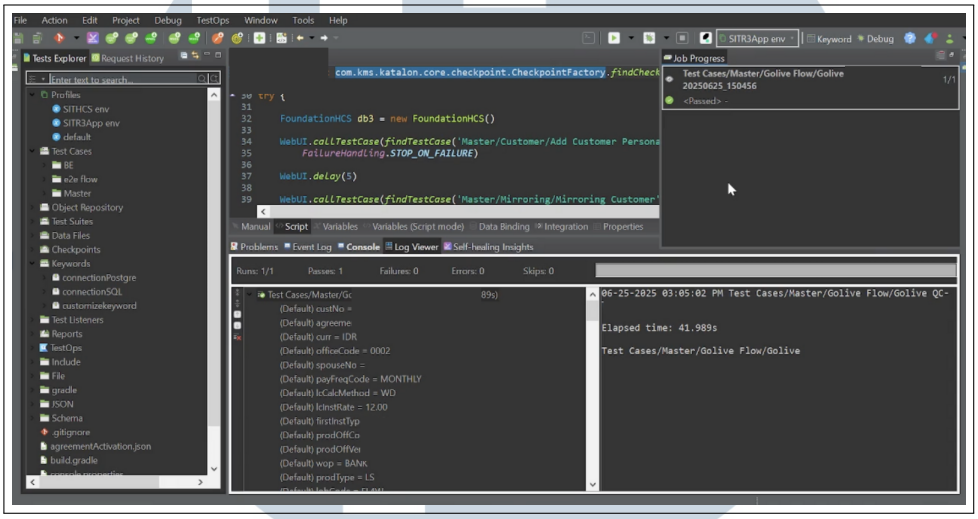
```
TEST CASE: Activation Agreement
PARAMETERS: agreementNo

BEGIN
    REQUEST activation service using agreement number
    RECEIVE response

    VERIFY response.statusCode == 200

    IF response is valid THEN
        PARSE response and CONFIRM status change
        ASSERT agreement status is updated to "Active"

        QUERY agreement by ID to confirm change

        IF DB status is "Active" THEN
            LOG "Test case successful and verified"
        ELSE
            LOG "DB status mismatch"
        ENDIF

        CAPTURE response metadata and store
    ENDIF
END
```

Kode 3.20: Pseudocode *activation agreement*

Terakhir, *test case activation* akan mengaktifkan sebuah *agreement*. Setelah parameter berupa nomor *agreement* diterima, sistem akan mengirimkan permintaan aktivasi. *Response* diverifikasi terlebih dahulu, lalu dipastikan bahwa status dan pesan sukses muncul. Jika valid, sistem akan memeriksa apakah perubahan status *agreement* sudah tercermin pada *database*, dan nilai status sesuai dengan yang diharapkan. Proses ini mencakup pengambilan data, validasi final, dan penyimpanan hasil. Tujuan akhir dari proses ini adalah memastikan bahwa *agreement* yang telah dibuat sebelumnya kini aktif dan dapat digunakan secara resmi dalam sistem.

Berikut adalah tampilan *test case* yang berstatus *pass* untuk skenario *main flow* pada proyek CONFINS, di mana seluruh proses pengecekan aktivasi *agreement* berhasil dijalankan hingga data terverifikasi aktif di dalam *database*. Hal ini

menjadi indikator valid bahwa proses *main flow* telah berjalan dengan sukses dan sesuai dengan ekspektasi sistem. Proses pengujian otomatis ini diselesaikan dalam waktu sekitar 1 menit maksimal, lebih cepat dibandingkan dengan pengujian manual yang pada umumnya memerlukan waktu 10–15 menit untuk menyelesaikan satu alur pengujian serupa yang bersifat *end-to-end*.



Gambar 3.11. Hasil pengujian *main Flow* API Confins

Secara keseluruhan, pemanggilan semua test case dalam *main flow* ini dirancang agar dapat digunakan pada berbagai skenario yang berbeda. Setiap skenario ditentukan berdasarkan variasi data input yang digunakan, sehingga memungkinkan pengujian dilakukan terhadap kondisi yang beragam tanpa perlu membangun ulang alur secara keseluruhan. Pemanggilan test case disusun secara modular dan dinamis dengan tujuan utama meningkatkan fleksibilitas dalam menyesuaikan kebutuhan skenario pengujian yang terus berkembang.

Struktur ini tidak hanya memudahkan dalam pengelolaan skrip pengujian otomatis, tetapi juga memungkinkan satu rangkaian main flow untuk digunakan kembali secara efisien di berbagai sesi pengujian, termasuk regresi dan validasi fitur baru. Hasil dari pembangunan skrip automation ini kemudian didistribusikan dan dimanfaatkan pada tahap produksi, khususnya oleh tim QA manual, untuk melakukan validasi lebih lanjut dengan data yang lebih realistis dan kompleks. Dengan pendekatan ini, proses pengujian menjadi lebih terintegrasi, terukur, dan mampu mendukung kesinambungan antara tim *automation* dan QA manual dalam menjaga kualitas produk secara berkelanjutan.

3.4 Kendala dan Solusi yang Ditemukan

Selama periode kerja magang di PT Adicipta Inovasi Teknologi sebagai *quality engineer* atau *automation engineer*, terdapat beberapa kendala yang dialami sebagai berikut:

1. Terbatasnya pemahaman awal terhadap sistem dan alur bisnis aplikasi yang diuji, sehingga membutuhkan waktu adaptasi dalam memahami alur pengujian serta tujuan dari setiap skenario.
2. Perubahan kebutuhan (*requirement*) yang terjadi secara dinamis selama proses pengujian, sehingga *test case* perlu diperbarui agar tetap selaras dengan sistem yang sedang dikembangkan.

Melalui kendala tersebut yang tergolong minor, berikut adalah solusi yang dilakukan:

1. Untuk mengatasi keterbatasan pemahaman terhadap sistem dan alur bisnis aplikasi, dilakukan pendekatan dengan aktif berdiskusi bersama tim AT, mengikuti pelatihan (*training*), serta memanfaatkan dokumentasi yang tersedia untuk mempercepat pemahaman. Selain itu, dilakukan eksplorasi langsung terhadap aplikasi untuk memahami *behavior* dan *dependency* antar modul.
2. Dalam menghadapi perubahan kebutuhan yang dinamis, dilakukan manajemen perubahan sesuai prosedur pada setiap *requirement*, serta menggunakan pendekatan *reusable test case* agar penyesuaian cukup dilakukan pada bagian yang terdampak, tanpa perlu mengubah keseluruhan skenario.