BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama pelaksanaan kegiatan magang di PT Adicipta Inovasi Teknologi (AdIns), posisi magang berada di departemen *Software Testing* (ST). Dalam struktur organisasi departemen tersebut, koordinasi langsung dilakukan dengan ST *Client Solution* 2 *Section Head* dan *Test Analyst*. Posisi sebagai *Software Tester Staff* berperan dalam mendukung proses pengujian perangkat lunak, baik untuk produk internal maupun proyek yang dikustomisasi. Tanggung jawab utama mencakup verifikasi bahwa sistem yang dikembangkan telah memenuhi standar kualitas, berjalan sesuai dengan *requirement* dan spesifikasi, serta terintegrasi secara optimal sebelum tahap implementasi ke klien.

Koordinasi pelaksanaan tugas berlangsung secara vertikal dan horizontal. Secara vertikal, hasil pekerjaan disampaikan kepada *Test Analyst* dan selanjutnya diteruskan kepada *Section Head*. Secara horizontal, koordinasi dilakukan dengan anggota tim dalam satu section, khususnya sesama ST *Staff*. Dalam beberapa tugas yang berkaitan dengan *automation testing*, kolaborasi juga dilakukan bersama *Senior Automation Test Engineer* dan QE *Staff* untuk memastikan bahwa implementasi dan pengujian berjalan sesuai standar yang berlaku.

Komunikasi harian memanfaatkan platform Microsoft seperti Teams untuk diskusi dan koordinasi, serta OneDrive untuk pengelolaan dokumen dan email. Sistem Jira digunakan sebagai alat utama dalam manajemen tugas dan pelacakan tiket. Selama pelaksanaan WFH, koordinasi dilakukan secara daring melalui Microsoft Teams dan Discord. Tim juga mengadakan pertemuan rutin mingguan (*weekly meeting*) untuk menyampaikan progres, membahas kendala teknis, serta menerima jadwal pengujian terbaru yang disampaikan oleh *Section Head*.

Seluruh aktivitas magang berada di bawah arahan dan pengawasan Ibu Yoanessa Anandri selaku *Section Head* dari ST *Client Solution* 2. Selain itu, bimbingan dan pemahaman terkait proses bisnis diperoleh langsung dari Ibu Julie Oktaviani yang menjabat sebagai *Test Analyst* dan bertindak sebagai supervisor selama periode magang.

Posisi magang ini memberikan pengalaman teknis langsung di bidang *software testing*, sekaligus membuka wawasan terhadap pentingnya kolaborasi dan

komunikasi yang efektif dalam lingkungan kerja profesional. Struktur organisasi dan mekanisme koordinasi antar tim terbukti memainkan peran penting dalam mendukung keberhasilan pengembangan sistem yang andal dan sesuai dengan kebutuhan klien.

3.2 Tugas yang Dilakukan

Selama kegiatan magang di PT Adicipta Inovasi Teknologi, posisi magang berada pada divisi ST *Client Solution* 2 dengan peran sebagai *Software Tester*. Divisi ini memiliki tanggung jawab utama dalam memastikan kualitas aplikasi melalui proses pengujian, khususnya pada tahap *System Integration Testing* (SIT).

Peran yang dijalankan melibatkan keterlibatan langsung dalam pengujian SIT terhadap produk inti perusahaan, yaitu sistem CONFINS, yang terdiri dari berbagai modul untuk mendukung bisnis *multifinance*. Produk ini terus dikembangkan agar dapat digunakan oleh berbagai klien. Sebelum mulai menangani SIT, diberikan waktu khusus untuk mempelajari dokumen *user* manual sebagai panduan awal untuk memahami fungsi dasar dari setiap modul dalam produk CONFINS. Proses ini menjadi langkah awal yang penting dalam memahami alur bisnis serta fitur utama dari modul yang akan diuji. Pemahaman praktis juga diperoleh melalui eksplorasi langsung terhadap modul-modul di *environment testing*.

Fokus utama SIT adalah menguji integrasi antar modul dalam sistem secara menyeluruh dan memastikan seluruh proses bisnis berjalan sesuai desain, tanpa terjadi eror, konflik antar modul, atau gangguan fungsi lainnya. Pengujian ini penting untuk menjamin bahwa pembaruan pada satu modul tidak mengganggu kestabilan modul lain yang saling terhubung. Adapun tugas dan kegiatan yang dijalankan selama proses SIT meliputi:

- Melakukan eksekusi *test scenario* yang telah disusun oleh ST *Staff* dan telah melalui proses *review* dan *approval*, untuk menguji fungsionalitas modul yang mengalami perubahan atau penggabungan ke dalam *core* CONFINS.
- Menjalankan aplikasi sebagai *end-user* dengan mengikuti langkah-langkah dalam *test scenario*, untuk mensimulasikan alur penggunaan riil oleh pengguna akhir.
- Mencatat temuan (*defect*) selama proses pengujian ke dalam *tools*

seperti JIRA, setelah melalui proses *review* oleh *test analyst*, untuk didokumentasikan dan ditindaklanjuti oleh tim developer.

- Melakukan koordinasi dengan tim developer dan ST *Staff* lain untuk klarifikasi serta verifikasi *defect*, termasuk permintaan pengecekan pada sisi *environment testing* dan pelaksanaan proses *End of Day* (EOD) bila diperlukan.
- Melakukan *regression testing* terhadap modul yang telah diperbaiki untuk memastikan kestabilan sistem, khususnya pada *defect* yang sebelumnya dilaporkan melalui JIRA.

Kegiatan pengujian juga didukung dengan persiapan yang meliputi pemahaman fungsi dasar modul dan partisipasi dalam diskusi internal tim, seperti pembagian tugas dan evaluasi hasil pengujian.

Meskipun selama kegiatan magang belum pernah terlibat langsung dalam penyusunan *test scenario*, kegiatan magang ini memberikan pemahaman menyeluruh terhadap proses pengujian perangkat lunak skala *enterprise*, termasuk identifikasi *defect*, uji integrasi, *automation testing*, serta pentingnya dokumentasi dan koordinasi tim dalam menjaga kualitas sistem yang kompleks seperti CONFINS.

Selain berkontribusi dalam pengujian SIT, terdapat pula pengalaman dalam melakukan API *automation testing* pada salah satu produk. Proses ini melibatkan pembelajaran dan penggunaan *tools* seperti Postman dan Katalon Studio untuk menguji *endpoint* API secara otomatis, guna memastikan respons dan komunikasi antar sistem berjalan sesuai harapan.

Di luar tugas teknis, kegiatan magang juga mencakup partisipasi dalam berbagai pelatihan internal perusahaan. Program pelatihan ini dirancang untuk seluruh jenjang karyawan, termasuk peserta magang. Materi pelatihan meliputi pengembangan *hardskill* seperti *Structured Query Language* (SQL), *Software Development Life Cycle* (SDLC), *Quality Control*, *Introduction to Multifinance*, serta pelatihan teknis lain yang mendukung pekerjaan dalam pengujian perangkat lunak dan pemahaman sistem CONFINS.

Pelatihan juga mencakup pengembangan *soft skill*, salah satunya melalui sesi *Basic Mentality* yang bertujuan membentuk sikap profesional, tangguh, dan adaptif di lingkungan kerja. Seluruh program pelatihan ini memberikan kontribusi signifikan dalam memperluas wawasan serta meningkatkan kompetensi teknis dan

non-teknis yang dibutuhkan dalam menjalankan peran sebagai *Software Tester* secara optimal.

3.3 Uraian Pelaksanaan Magang

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

Minggu Ke -	Pekerjaan yang dilakukan
1	Melakukan proses *onboarding* dan administrasi seperti registrasi *fingerprint*, pembuatan akun perusahaan, serta instalasi berbagai *tools* kerja; mengenal struktur perusahaan dan organisasi; mengikuti pelatihan serta ujian dasar seperti SQL, SDLC, *Quality Control*, *Intro to Multifinance*, dan CONFINS *Overview*; mengikuti *briefing* awal terkait sistem kerja magang dan tanggung jawab sebagai *Software Tester*, serta perkenalan dengan departemen *Software Testing*; mempelajari dokumen *user* manual terkait salah satu lini bisnis dari sebuah modul produk yang terdapat pada CONFINS.
2	Mempelajari dokumen *user* manual terkait salah satu lini bisnis dari sebuah modul produk dalam sistem CONFINS; menyusun rangkuman dan presentasi dari hasil pemahaman terhadap produk yang telah dipelajari; mengikuti *weekly team meeting* pertama serta *briefing* awal penggunaan *tools* seperti JIRA dan Clockify; melanjutkan pembelajaran terhadap dokumen *user* manual untuk modul-modul lain dalam CONFINS.
3	Menyusun rangkuman dan presentasi dari hasil pemahaman terhadap produk; mempelajari dan berlatih API *automation testing* menggunakan Postman; melakukan instalasi Katalon Studio; mempelajari dan mencoba API *automation testing* menggunakan Katalon Studio; membantu proses awal pengujian *environment* untuk *System Integration Testing* pada salah satu proyek; mengikuti *daily briefing* dengan tim *Quality Engineering*; mengerjakan tiket API *automation testing* untuk produk.

Minggu Ke -	Pekerjaan yang dilakukan
4	Mengikuti *daily briefing* bersama tim *Quality Engineering*; mengerjakan tiket API *automation testing* untuk produk; mempelajari dokumen *user* manual terkait modul-modul produk yang terdapat dalam sistem CONFINS.
5	Mempelajari dokumen *user* manual terkait modul-modul produk yang terdapat pada CONFINS; melakukan eksplorasi terhadap seluruh modul yang termasuk dalam cakupan pengujian pada *environment testing* untuk keperluan SIT; serta menyusun administrasi laporan KPIM.
6	Melakukan eksplorasi terhadap seluruh modul yang termasuk dalam cakupan pengujian pada *environment testing* untuk keperluan SIT.
7	Melakukan *System Integration Testing* (SIT) terhadap produk A.
8	Melakukan *System Integration Testing* (SIT) terhadap produk A.
9	Melakukan *System Integration Testing* (SIT) terhadap produk A; mempelajari *guideline* terbaru untuk penyusunan *test scenario*; mempelajari dokumen *user* manual terkait modul produk yang terdapat pada CONFINS sambil melakukan eksplorasi langsung terhadap modul tersebut pada *environment testing* untuk keperluan SIT.
10	Membuat catatan singkat terkait salah satu modul yang menjadi tanggung jawab saat pelaksanaan *System Integration Testing* (SIT) untuk produk A, guna keperluan dokumen Berita Acara Testing; mempelajari dokumen *user* manual terkait modul produk yang terdapat pada CONFINS sambil melakukan eksplorasi langsung terhadap modul tersebut di *environment testing* untuk kebutuhan SIT; mengikuti pelatihan *soft skill* seperti *Basic Mentality*; mengikuti pertemuan *buddy system*; serta menyusun administrasi laporan IPMS.

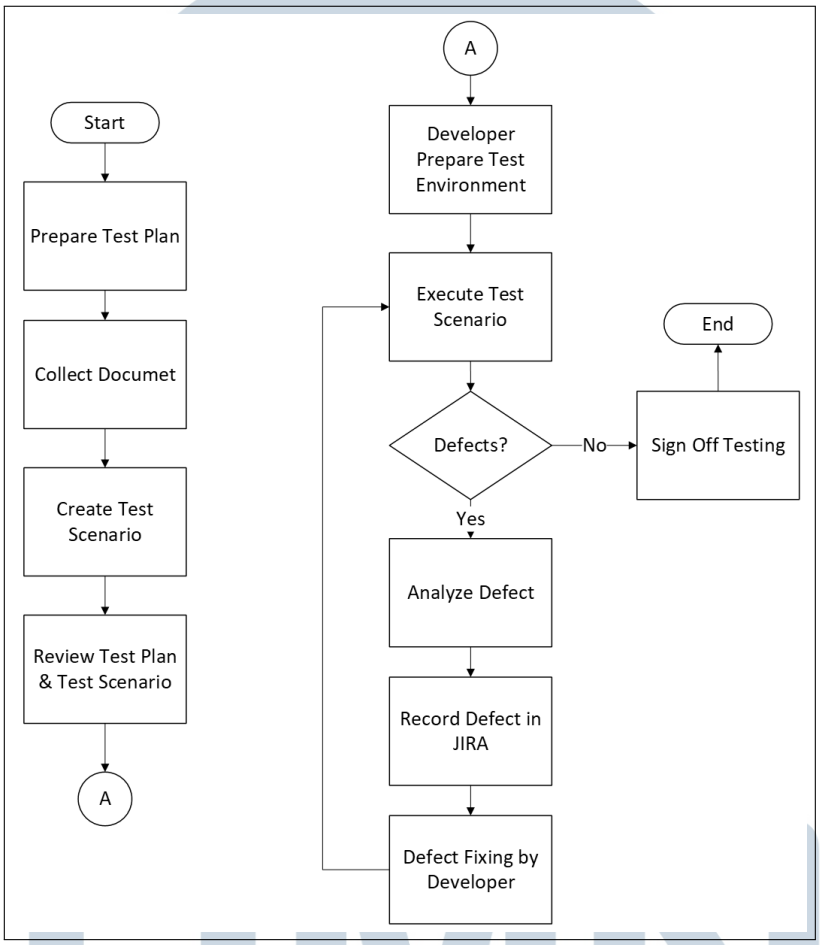
Minggu Ke -	Pekerjaan yang dilakukan
11	Melakukan *System Integration Testing* (SIT) terhadap produk B; serta mengikuti pelatihan *hard skill* seperti *Intro to Multifinance*, CONFINS *Overview*, *Basic Finance*, dan *Basic Accounting*.
12	Melakukan *System Integration Testing* (SIT) terhadap produk B; serta mengikuti pelatihan *hard skill* terkait salah satu modul produk di CONFINS.
13	Melakukan *System Integration Testing* (SIT) terhadap produk B; serta mengikuti pelatihan *hard skill* terkait salah satu modul produk di CONFINS.
14	Melakukan *System Integration Testing* (SIT) terhadap produk B; serta mengikuti pelatihan *hard skill* terkait salah satu modul produk di CONFINS.
15	Melakukan *System Integration Testing* (SIT) terhadap produk B; serta mengikuti pelatihan *hard skill* terkait salah satu modul produk di CONFINS.
16	Melakukan *System Integration Testing* (SIT) terhadap produk B; serta mengikuti pelatihan *hard skill* terkait salah satu modul produk di CONFINS.

3.3.1 System Integration Testing (SIT)

System Integration Testing (SIT) merupakan tahapan krusial dalam proses pengujian perangkat lunak yang bertujuan untuk memastikan bahwa seluruh modul atau komponen sistem dapat bekerja secara terpadu dan saling terintegrasi sesuai dengan alur bisnis yang telah dirancang [3]. Tahapan ini dilakukan setelah sistem selesai dikembangkan secara menyeluruh, untuk mengevaluasi sejauh mana masing-masing komponen mampu berinteraksi secara efektif, serta memastikan bahwa sistem secara keseluruhan telah memenuhi spesifikasi fungsional dan teknis yang telah ditetapkan dalam dokumen kebutuhan [4].

Pengujian pada tahap SIT yang menjadi tanggung jawab tim *Software Tester*. SIT dilakukan pada sistem yang telah terintegrasi sepenuhnya dan berfokus pada validasi proses bisnis secara menyeluruh. Tim penguji menjalankan skenario pengujian berdasarkan *business process flow* yang telah disusun, guna memastikan bahwa seluruh modul dalam sistem bekerja secara harmonis dan tidak terjadi

kegagalan proses pada titik-titik integrasi. Tahapan ini juga menjadi sarana untuk mendeteksi *defect* yang muncul dari interaksi lintas modul atau kesalahan integrasi data antar proses.



Gambar 3.1. Alur Umum Proses System Integration Testing (SIT)

Secara umum, alur pelaksanaan *System Integration Testing* mengikuti tahapan-tahapan seperti yang ditampilkan pada Gambar 3.1. Diagram tersebut memperlihatkan proses dari awal perencanaan pengujian, pembuatan skenario, pelaksanaan pengujian, pencatatan *defect*, hingga *sign off*. Setiap tahapan tersebut merupakan bagian yang saling terhubung dan menjadi indikator penting dalam keberhasilan proses integrasi sistem secara menyeluruh.

Selama menjalani program magang, pengalaman yang diperoleh berfokus pada kegiatan testing produk, khususnya pada tahap *System Integration Testing* (SIT) terhadap sistem CONFINS. Tahapan ini menjadi kesempatan pertama untuk terlibat langsung dalam pengujian perangkat lunak pada level sistem terintegrasi. Kegiatan pengujian dilakukan dengan mengikuti skenario uji yang mencerminkan

proses bisnis dalam industri *multifinance*, guna memastikan bahwa sistem berjalan sesuai dengan standar kebutuhan fungsional yang berlaku secara umum.

Selama menjalani proses SIT, beberapa aktivitas teknis yang dilakukan sebagai berikut:

1. Mempelajari Dokumen User Manual Guide

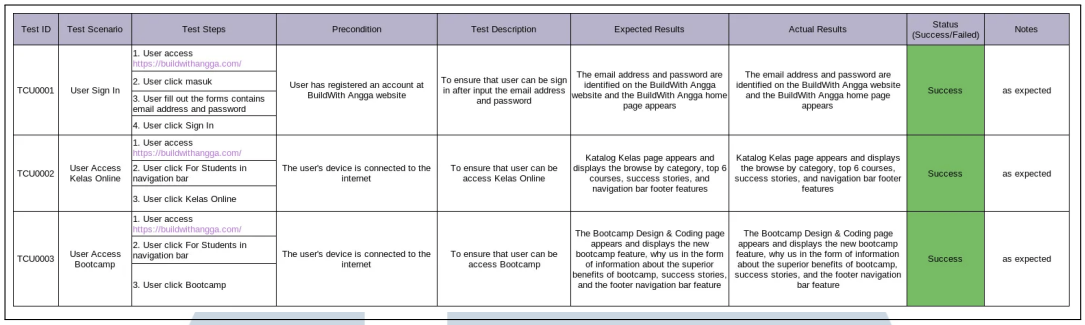
   Sebelum terlibat langsung dalam proses *System Integration Testing* (SIT), kegiatan diawali dengan mempelajari dokumen-dokumen *user* manual yang digunakan oleh tim *Software Tester* sebagai acuan utama dalam pengujian.

   Dokumen *user* manual tersebut memuat penjabaran lengkap mengenai seluruh menu, *page*, *action*, *field*, serta deskripsi fungsi dari setiap elemen yang terdapat dalam modul-modul di sistem CONFINS. Selain itu, dokumen ini juga menjelaskan alur penggunaan sistem dari sudut pandang pengguna akhir, sehingga memberikan pemahaman menyeluruh mengenai cara akses dan operasionalisasi fitur dalam konteks penggunaan nyata.

   Pemahaman terhadap dokumen ini memungkinkan proses pemetaan skenario pengujian dilakukan dengan lebih tepat, membantu dalam mengidentifikasi titik-titik krusial yang perlu diuji, serta memastikan bahwa setiap langkah pengujian selaras dengan ekspektasi pengguna dan standar fungsionalitas sistem.

2. Eksekusi Test Case

   Proses pengujian dilakukan dengan mengeksekusi *test case* yang merupakan turunan langsung dari *test scenario* yang telah disusun berdasarkan dokumen *requirement* serta alur proses bisnis (*business process flow*) dari sistem. Setiap *test case* dirancang untuk memverifikasi apakah fitur-fitur dalam sistem berfungsi sesuai harapan, baik dari sisi fungsionalitas maupun integrasinya dengan modul lain.



Test ID	Test Scenario	Test Steps	Precondition	Test Description	Expected Results	Actual Results	Status (Success/Failed)	Notes
TCU0001	User Sign In	1. User access https://buildwithangga.com/ 2. User click masuk 3. User fill out the forms contains email address and password 4. User click Sign In	User has registered an account at BuildWith Angga website	To ensure that user can be sign in after input the email address and password	The email address and password are identified on the BuildWith Angga website and the BuildWith Angga home page appears	The email address and password are identified on the BuildWith Angga website and the BuildWith Angga home page appears	Success	as expected
TCU0002	User Access Kelas Online	1. User access https://buildwithangga.com/ 2. User click For Students in navigation bar 3. User click Kelas Online	The user's device is connected to the internet	To ensure that user can be access Kelas Online	Katalog Kelas page appears and displays the browse by category, top 6 courses, success stories, and navigation bar footer features	Katalog Kelas page appears and displays the browse by category, top 6 courses, success stories, and navigation bar footer features	Success	as expected
TCU0003	User Access Bootcamp	1. User access https://buildwithangga.com/ 2. User click For Students in navigation bar 3. User click Bootcamp	The user's device is connected to the internet	To ensure that user can be access Bootcamp	The Bootcamp Design & Coding page appears and displays the new bootcamp feature, why us in the form of information about the superior benefits of bootcamp, success stories, and the footer navigation bar feature	The Bootcamp Design & Coding page appears and displays the new bootcamp feature, why us in the form of information about the superior benefits of bootcamp, success stories, and the footer navigation bar feature	Success	as expected

Gambar 3.2. Contoh Tampilan Umum Test Case

Sumber: [5]

Gambar 3.2 menampilkan contoh tampilan umum daftar *test case* yang menyerupai format yang digunakan dalam proses *System Integration Testing* (SIT) di AdIns. Dalam praktiknya, daftar *test case* telah dikelompokkan berdasarkan modul-modul yang berada di dalam sistem CONFINS. Setiap baris mencantumkan ID skenario induk, nama *test case*, dan status eksekusi terkini.

Pengelolaan dan dokumentasi *test case* dilakukan menggunakan Microsoft Excel, yang disesuaikan dengan template tes standar internal AdIns. Excel digunakan karena kemudahannya dalam pengorganisasian data, penandaan warna untuk status eksekusi, serta fitur filter dan sort yang memudahkan navigasi antar *test case*. Selain itu, Excel juga berfungsi sebagai media kolaboratif antara ST *Staff, Test Analyst*, dan Developer dalam melakukan pembaruan status atau menambahkan catatan penting selama proses pengujian berlangsung.

Setiap *test case* dilengkapi dengan data uji (*test data*) yang telah disesuaikan dengan kondisi bisnis dan berbagai variasi input yang mungkin muncul dalam penggunaan sistem nyata. Keberadaan *test data* ini sangat krusial dalam memastikan keabsahan hasil pengujian, karena input yang relevan akan menggambarkan skenario pengguna secara realistis.

Selain daftar *test case* dan data uji, proses pengujian juga ditunjang oleh fitur monitoring progres yang mencakup jumlah *test case* yang telah dieksekusi, status pelaksanaan seperti *unexecuted*, *success*, *failed*, dan *blocked*, serta visualisasi status dalam bentuk grafik atau persentase. Tampilan ini sangat berguna untuk mengevaluasi pencapaian target pengujian harian dan mengidentifikasi area yang masih memerlukan perhatian lebih lanjut.

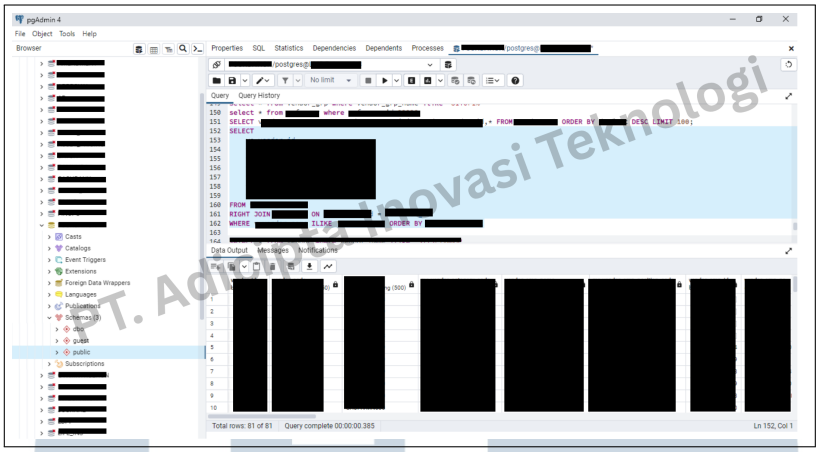
Selain itu, tersedia pula *notes* atau kolom catatan yang digunakan untuk mencatat hasil observasi dan melakukan pemetaan terhadap *defect* yang ditemukan selama proses eksekusi. Catatan ini menjadi rujukan penting bagi tim pengembang dalam proses debugging dan perbaikan, serta mendukung penelusuran ulang apabila terjadi regresi pada fitur terkait di masa mendatang.

Keberadaan elemen-elemen seperti *test case*, data uji, *monitoring* progres, dan lainnya menjadikan proses pengujian berjalan lebih sistematis, terarah, dan terdokumentasi dengan baik. Dokumentasi tersebut juga berperan penting dalam pelaporan hasil pengujian kepada pihak manajemen proyek dan tim developer, serta menjadi referensi penting apabila ditemukan *defect* pada sistem.

3. Verifikasi Database menggunakan pgAdmin 4

   Pengujian basis data dilakukan dengan menggunakan pgAdmin 4, sebuah alat grafis resmi untuk PostgreSQL yang memudahkan akses dan manipulasi data secara langsung melalui SQL *query*. Pengujian ini bertujuan untuk memastikan bahwa input dan output sistem yang tersimpan di dalam basis data PostgreSQL telah sesuai dengan ekspektasi, serta tidak terdapat kehilangan data (*data loss*) maupun inkonsistensi.

   Validasi melalui *database* ini menjadi sangat penting, terutama untuk pengujian pada fitur-fitur yang melibatkan proses transaksi, penyimpanan data, dan relasi antar entitas. Dengan menggunakan pgAdmin 4, dapat dilakukan penelusuran data backend secara *real-time* dan membandingkannya dengan hasil pada antarmuka pengguna (*frontend*). Tampilan antarmuka pgAdmin 4 yang digunakan untuk proses ini dapat dilihat pada Gambar 3.3.



Gambar 3.3. Antarmuka pgAdmin 4 saat digunakan untuk verifikasi Database
Sumber: Data Perusahaan

Pengujian dilakukan dengan mengeksekusi berbagai jenis query SQL, mulai dari query dasar seperti *'SELECT * FROM'* hingga query kompleks yang melibatkan penggunaan klausa *'JOIN'*, *'WHERE'*, *'ORDER BY'*, *'UNION'*, dan operator logika seperti *'AND'* maupun *'ILIKE'*. Eksekusi *query* ini bertujuan untuk:

- Mengecek keberadaan dan integritas data.
- Menemukan inkonsistensi antara antarmuka pengguna dan isi *database*.
- Memverifikasi hasil kalkulasi dan pemrosesan data oleh sistem.
- Memastikan keterhubungan antar tabel melalui relasi yang tepat.

Beberapa contoh *query* SQL yang umum digunakan selama proses pengujian antara lain:

```
SELECT * FROM users;
```

Kode 3.1: Contoh Query dasar untuk menampilkan seluruh isi tabel

```
SELECT * FROM transactions
WHERE status = 'FAILED';
```

Kode 3.2: Contoh Query untuk mencari transaksi gagal

```
SELECT name, email
FROM customers
WHERE email ILIKE '%gmail.com';
```

Kode 3.3: Contoh Query dengan pencarian email menggunakan ILIKE

```sql
SELECT o.order_id, o.order_date, c.name
FROM orders o
LEFT JOIN customers c ON o.customer_id = c.customer_id
WHERE o.order_date >= '2024-01-01';
```

Kode 3.4: Contoh Query LEFT JOIN antara orders dan customers

```sql
SELECT u.username, t.transaction_id, t.amount, t.status
FROM users u
RIGHT JOIN transactions t ON u.user_id = t.user_id
WHERE t.status = 'SUCCESS'
AND t.amount > 500000
ORDER BY t.transaction_date DESC;
```

Kode 3.5: Contoh Query RIGHT JOIN dengan filter dan sorting

```sql
SELECT user_id, 'Login' AS activity_type, login_time AS
    activity_time
FROM login_logs
WHERE login_time > '2025-01-01'

UNION

SELECT user_id, 'Transaction' AS activity_type,
    transaction_date AS activity_time
FROM transactions
WHERE transaction_date > '2025-01-01'
ORDER BY activity_time DESC;
```

Kode 3.6: Contoh Query UNION untuk menggabungkan aktivitas login dan transaksi

```sql
SELECT t.transaction_id, u.username, t.amount, 'Transaction'
    AS type
FROM transactions t
LEFT JOIN users u ON t.user_id = u.user_id
WHERE t.status = 'SUCCESS'

UNION

SELECT l.log_id, u.username, NULL AS amount, 'Login Failed'
    AS type
FROM login_logs l
RIGHT JOIN users u ON l.user_id = u.user_id
WHERE l.result = 'FAILED'

```

```
13 ORDER BY username;
```

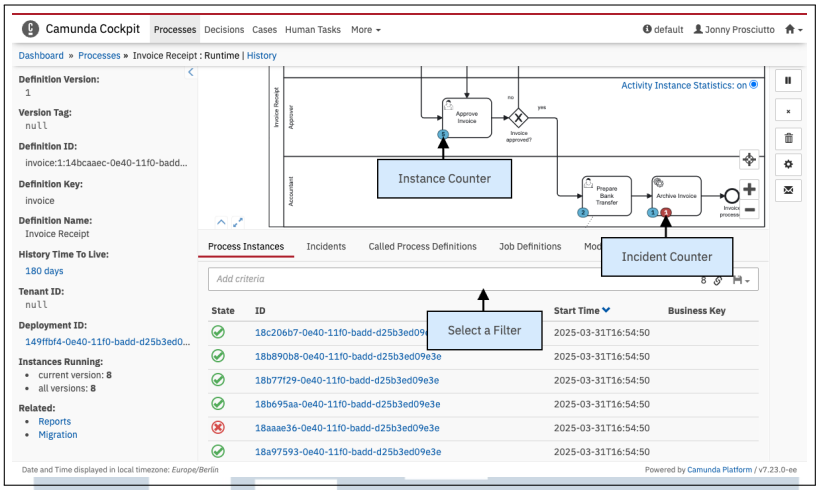
Kode 3.7: Contoh Query kompleks dengan LEFT JOIN, RIGHT JOIN, UNION

Penggunaan kombinasi *query* seperti di atas memungkinkan untuk melakukan validasi menyeluruh terhadap berbagai aspek sistem. Penggunaan *'JOIN'* memungkinkan relasi antar tabel untuk diverifikasi, sementara penggunaan *'UNION'* membantu menyatukan data dari dua sumber berbeda dalam satu tampilan analisis. Selain itu, penggunaan *'ILIKE'* sangat bermanfaat dalam pencarian data *non-case-sensitive*, misalnya untuk pengecekan domain email atau nama pengguna.

Dengan pendekatan ini, *Software Tester* dapat memastikan bahwa tidak hanya tampilan sistem yang berfungsi dengan benar, namun juga logika dan penyimpanan datanya telah berjalan sesuai dengan kebutuhan sistem dan spesifikasi yang telah ditentukan.

4. Verifikasi Alur Bisnis Menggunakan Camunda Cockpit
   Camunda Cockpit merupakan alat bantu berbasis *Business Process Model and Notation* (BPMN) yang digunakan untuk memvisualisasikan dan memverifikasi jalannya alur proses bisnis secara otomatis dalam sistem *backend*. Dengan pendekatan ini, proses seperti *approval*, *validasi*, maupun pengiriman notifikasi dapat dimonitor dalam bentuk diagram yang merepresentasikan setiap tahapan secara eksplisit.

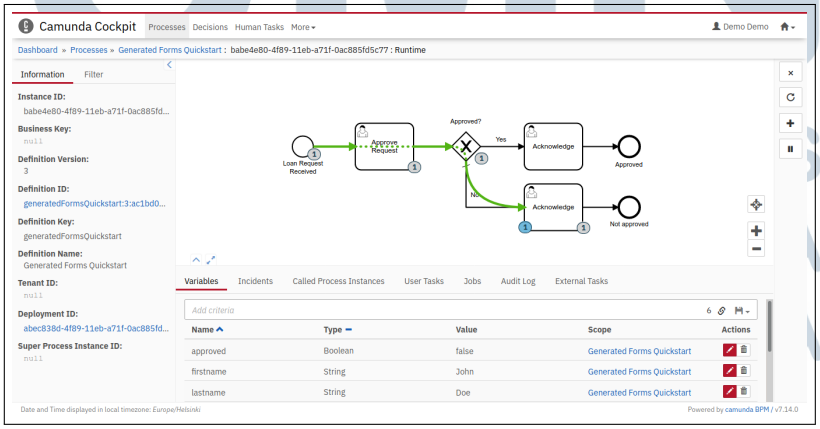
   Melalui fitur daftar proses, pengguna dapat melihat semua BPMN yang telah didefinisikan dan di *deploy* ke dalam sistem. Setiap proses memiliki ID, versi, dan informasi lainnya yang dapat ditelusuri untuk melihat status eksekusi dari instansinya. Gambar 3.4 berikut menampilkan daftar instansi proses yang telah atau sedang dijalankan.



Gambar 3.4. Contoh Daftar Instansi Proses yang Sudah atau Sedang Berjalan
Sumber: [6]

Informasi pada Gambar 3.4 memungkinkan pengguna untuk memantau status setiap instansi proses, apakah dalam kondisi aktif, selesai, atau mengalami kegagalan. Jika terdapat instansi yang terhenti atau gagal pada *node* tertentu, maka dapat dilakukan analisis untuk mendeteksi kesalahan logika atau kendala teknis yang terjadi pada *flow engine*.

Lebih lanjut, Camunda menyediakan tampilan visual dari *sequence flow* untuk setiap proses yang dijalankan. Hal ini digambarkan dalam Gambar 3.5, yang menampilkan jalur eksekusi proses dalam bentuk diagram BPMN secara *real-time*. *Node-node* yang telah berhasil dieksekusi akan diberi penanda visual (misalnya warna hijau), sementara *node* yang belum dilalui akan dibiarkan kosong.



Gambar 3.5. Contoh Tampilan Visual Sequence Flow Proses
Sumber: [6]

Tampilan visual pada Gambar 3.5 sangat membantu dalam proses verifikasi pengujian, karena memungkinkan pengamat untuk memastikan bahwa seluruh langkah dalam proses bisnis berjalan sesuai urutan yang telah dirancang. Jika terdapat bottleneck atau proses yang tertahan, maka sistem akan memberikan indikasi visual yang mempermudah proses debugging dan pelacakan kesalahan.

Dengan memanfaatkan Camunda, proses validasi terhadap alur bisnis menjadi lebih transparan, terstruktur, dan efisien, khususnya dalam konteks pengujian sistem terotomatisasi.

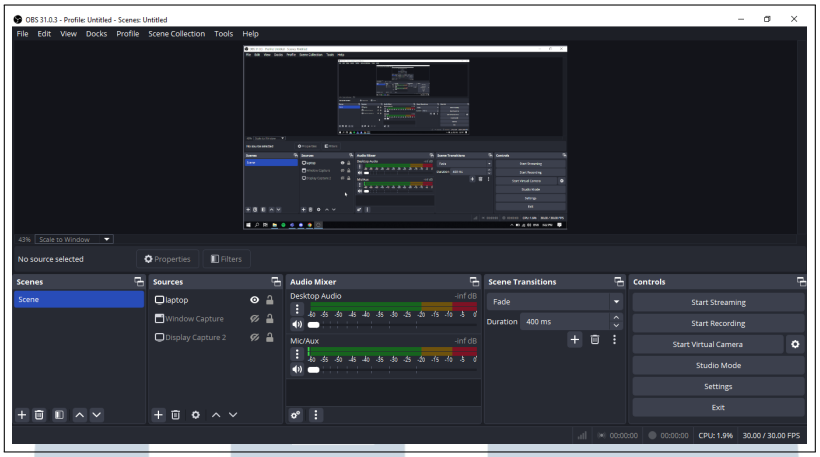
5. Dokumentasi Proses Testing

   Setiap aktivitas pengujian dicatat dalam bentuk tangkapan layar (*screenshot*) sebagai bukti pengujian. Dokumentasi ini digunakan untuk keperluan *review* dan validasi oleh tim *Software Tester* (ST) serta *stakeholders* terkait. Pendokumentasian juga memudahkan proses penelusuran (*traceability*) jika terjadi eror atau ketidaksesuaian hasil.

   Tangkapan layar biasanya diambil menggunakan aplikasi bawaan Windows seperti Snipping Tool atau Snip & Sketch, maupun menggunakan aplikasi pihak ketiga seperti Lightshot yang memungkinkan pengambilan *screenshot* dengan cepat dan fleksibel, serta dilengkapi fitur anotasi untuk memberi penanda pada bagian penting.

   Selain menggunakan tangkapan layar, tim ST juga memanfaatkan perangkat lunak seperti OBS Studio untuk merekam proses pengujian secara langsung. Penggunaan OBS memungkinkan pendokumentasian alur interaksi pengguna dengan sistem secara visual dan dinamis, yang sangat membantu dalam menjelaskan langkah-langkah pengujian atau mereplikasi skenario uji tertentu.

   Sebagai contoh, Gambar 3.6 berikut menunjukkan tampilan dari perangkat lunak OBS yang digunakan untuk merekam proses pengujian aplikasi.



Gambar 3.6. Tampilan Software OBS yang Digunakan untuk Merekam Proses Pengujian

6. Pelaporan Defect melalui JIRA

   Jika ditemukan kesalahan atau ketidaksesuaian (*defect*), *defect* tersebut dilaporkan melalui sistem manajemen isu JIRA. Laporan *defect* mencakup informasi penting seperti langkah-langkah replikasi, hasil aktual, hasil yang diharapkan, tingkat prioritas *defect*, serta bukti visual berupa tangkapan layar dari sistem yang diuji. Setiap *defect* yang telah dilaporkan akan terus dipantau status penyelesaiannya untuk memastikan bahwa permasalahan ditindaklanjuti secara tepat oleh tim developer.

   Selama masa magang, kegiatan *System Integration Testing* (SIT) dilakukan pada dua produk yang berbeda, yaitu produk A dan produk B. Menariknya, proses pelaporan defect pada masing-masing produk menggunakan dua versi antarmuka JIRA yang berbeda.

   Pada saat pengujian produk A, digunakan antarmuka lama JIRA (JIRA klasik) yang memiliki tampilan yang lebih simpel dan field pengisian seperti *Summary*, *Menu*, *Steps to Reproduce*, dan *Attachment* serta *field* lainnya ditampilkan dalam satu halaman terpisah. Gambar 3.7 berikut menampilkan logo JIRA klasik sebagai representasi dari sistem yang digunakan pada produk A.

Gambar 3.7. Logo JIRA Klasik yang Digunakan pada Produk A

Sementara itu, pada saat pengujian produk B, digunakan antarmuka JIRA versi terbaru yang hadir dengan desain lebih modern dan modular. *Field* seperti *Priority*, *Description*, dan *Attachments* ditampilkan secara dalam satu halaman namun melalui tampilan *pop-up card*. Logo JIRA terbaru yang digunakan dalam pengujian produk B dapat dilihat pada Gambar 3.8.

Gambar 3.8. Logo JIRA Versi Terbaru yang Digunakan pada Produk B

Meskipun gambar hanya menampilkan logo masing-masing versi, pengalaman menggunakan kedua varian JIRA ini memberikan wawasan mengenai variasi alur kerja pelaporan *defect* dalam lingkungan industri. Perbedaan tampilan antarmuka menunjukkan pentingnya fleksibilitas dan kemampuan beradaptasi terhadap *tools* yang digunakan oleh masing-masing tim proyek.

Dalam proses pelaporan *defect* di JIRA, terdapat klasifikasi tipe masalah (*issue type*) yang terdiri dari:

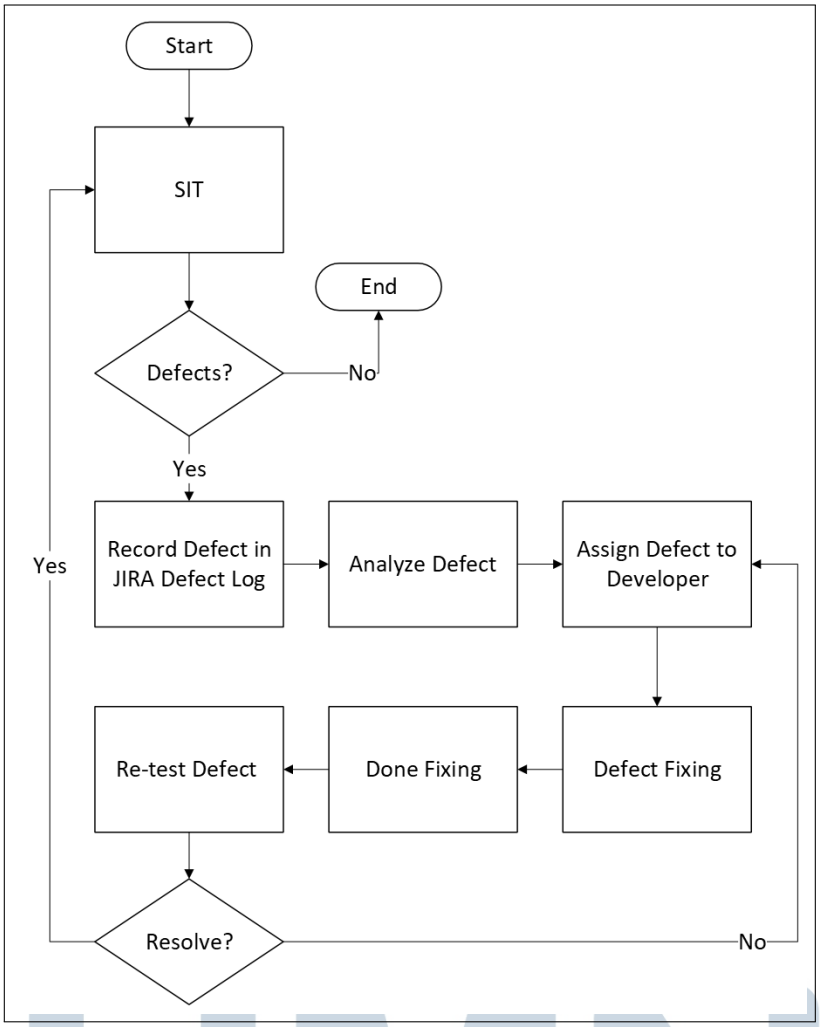
- Bug : untuk pelaporan kesalahan fungsi sistem atau ketidaksesuaian *output*.
- Clarification : untuk permintaan klarifikasi terkait *requirement* atau fitur sistem yang tidak jelas.
- Suggestion : untuk usulan peningkatan atau improvisasi fitur berdasarkan hasil pengujian.

Selain mendokumentasikan setiap *defect* yang ditemukan selama proses pengujian, tim penguji juga mengklasifikasikannya berdasarkan jenis dan tingkat prioritas. Klasifikasi jenis *defect* bertujuan untuk mengelompokkan bentuk kesalahan secara umum, seperti kesalahan tampilan antarmuka, fungsi tombol atau tautan yang tidak berjalan semestinya, kesalahan perhitungan sistem, validasi input yang tidak sesuai, serta kegagalan integrasi antar modul atau sistem eksternal. Pengelompokan ini memudahkan tim pengembang dalam mengidentifikasi akar permasalahan dan menetapkan tindakan korektif yang tepat.

Selain berdasarkan jenis, setiap *defect* juga diklasifikasikan berdasarkan tingkat prioritas, yang ditentukan dari seberapa besar dampaknya terhadap jalannya sistem. *Defect* dengan prioritas *Blocker* adalah kesalahan yang menyebabkan proses bisnis utama tidak dapat dijalankan sama sekali atau sistem berhenti total. Pada tingkat *High*, *defect* mengacu pada kesalahan yang signifikan dan berdampak besar terhadap fungsi sistem, meskipun sistem masih dapat digunakan secara terbatas. Sementara itu, *Medium* menunjukkan kesalahan yang tidak memengaruhi alur utama, tetapi tetap perlu ditangani karena berpotensi menimbulkan kebingungan atau menurunkan kenyamanan pengguna. Terakhir, prioritas *Low* mencakup kesalahan minor seperti tampilan visual, ikon, atau penempatan teks yang tidak rapi, yang tidak berdampak langsung pada fungsionalitas sistem.

Dengan melakukan klasifikasi ini, baik dari sisi jenis maupun prioritas, proses analisis dan penanganan *defect* menjadi lebih terstruktur dan efisien. Praktik ini juga membantu tim pengembang dan penguji dalam menyusun skala urgensi penyelesaian, serta memastikan bahwa kualitas sistem tetap terjaga selama siklus pengembangan perangkat lunak berlangsung.

Sebagai pelengkap dalam pelaporan *defect*, tim *Software Tester* diwajibkan menyertakan *attachment* berupa *screenshot* (tangkapan layar) atau *screen record* (rekaman video) yang menunjukkan secara langsung kondisi sistem saat *defect* terjadi. Bukti pendukung ini memberikan konteks yang lebih jelas bagi tim developer dalam memahami skenario dan dampak *defect*, serta mempercepat proses analisis dan perbaikan. Lampiran ini juga menjadi bagian penting dalam proses validasi kembali oleh tim ST saat melakukan *retesting* atau *regression testing*.



Gambar 3.9. Alur Pelaporan Defect oleh Software Tester

Selain memahami teknis pelaporan *defect*, pengalaman langsung dalam proses pengujian turut memberikan pemahaman mengenai alur kerja atau *workflow* status *defect*, mulai dari awal pelaporan hingga *defect* selesai diperbaiki. Gambar 3.9 memperlihatkan alur pelaporan *defect* secara umum, dimulai dari pengujian sistem, identifikasi bug, dokumentasi *defect*, hingga pelaporan ke tim developer melalui JIRA. Setelah perbaikan dilakukan, tim ST akan memverifikasi ulang *defect* untuk memastikan bahwa masalah telah terselesaikan dengan benar.

7. Retesting dan Regression Testing

   Setelah tim pengembang menyelesaikan perbaikan terhadap *defect* yang ditemukan, dilakukan pengujian ulang untuk memastikan sistem telah

berfungsi dengan baik. Pengujian ini terdiri atas dua tahapan utama, yaitu *Retesting* dan *Regression Testing*.

Retesting bertujuan untuk memverifikasi bahwa perbaikan *defect* telah berhasil dan tidak menimbulkan kesalahan yang sama. Pengujian dilakukan dengan menjalankan kembali skenario uji yang sebelumnya gagal menggunakan langkah dan data yang sama. Jika hasilnya sesuai harapan, pengujian dilanjutkan ke tahap *Regression Testing*.

Regression Testing dilakukan untuk memastikan bahwa perbaikan tidak menimbulkan efek samping terhadap fitur atau modul lain dalam sistem. Tahap ini menjadi penting guna menjaga stabilitas dan konsistensi sistem setelah adanya perubahan.

Apabila seluruh pengujian menunjukkan hasil yang baik, maka *defect* dapat dinyatakan selesai. Namun jika masih ditemukan kendala, *defect* akan dikembalikan untuk diperbaiki ulang. Dengan tahapan ini, kualitas sistem dapat tetap terjaga dan risiko kesalahan lanjutan dapat diminimalkan.

8. Sign Off Testing

   Setelah seluruh skenario pengujian berhasil dijalankan tanpa menyisakan *defect* kritikal, tim *Software Tester* (ST) melaksanakan proses *sign off testing* sebagai bentuk persetujuan akhir bahwa sistem telah dinyatakan lulus pada tahap *System Integration Testing* (SIT). Proses ini menjadi langkah penting yang menandai bahwa sistem telah memenuhi seluruh kriteria kelayakan yang telah ditetapkan, dan pengujian telah dilaksanakan secara menyeluruh sesuai dengan skenario yang telah disepakati bersama.

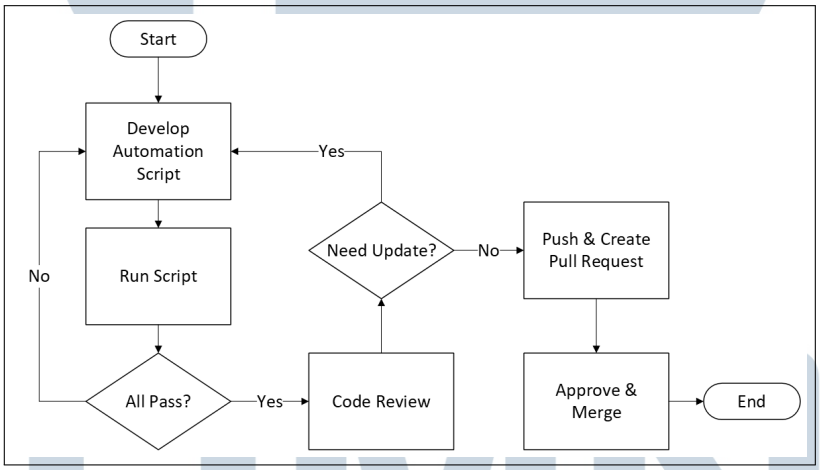
   Proses *sign off* dilakukan setelah seluruh rangkaian pengujian SIT selesai sesuai dengan jadwal yang telah ditentukan. Selain itu, proses ini juga menandai bahwa telah terjadi kesepakatan antara tim ST dan developer mengenai jadwal *last deploy* ke lingkungan pengujian, di mana tidak akan ada perubahan lebih lanjut yang dapat memengaruhi stabilitas sistem selama periode penilaian akhir dilakukan.

   Setelah proses *sign off*, tim ST akan menyusun dan mengajukan dokumen PLR (Persetujuan Layak Rilis) kepada pihak-pihak terkait seperti *Product Owner*, Developer, dan *Business Unit*. PLR berfungsi sebagai dasar pengambilan keputusan terhadap kelayakan sistem untuk melanjutkan ke tahap pengembangan berikutnya.

Sebagai penutup dari seluruh rangkaian pengujian, dibuat sebuah dokumen resmi berupa Berita Acara Testing. Dokumen ini berisi ringkasan hasil pengujian termasuk jumlah skenario, hasil pengujian, daftar *defect* yang ditemukan dan statusnya, catatan tentang apa yang telah terjadi selama proses SIT misalnya masalah server, atau lainnya serta pihak-pihak yang terlibat. Berita acara ini ditandatangani oleh perwakilan ST dan *Product Owner* sebagai bukti validasi akhir terhadap hasil pengujian sistem.

3.3.2 Application Programming Interface (API) Automation Testing

API *Automation Testing* merupakan proses pengujian terhadap antarmuka pemrograman aplikasi API secara otomatis untuk memastikan integrasi antar sistem berjalan sesuai spesifikasi yang telah ditentukan.



Gambar 3.10. Alur Pengerjaan API Automation Testing

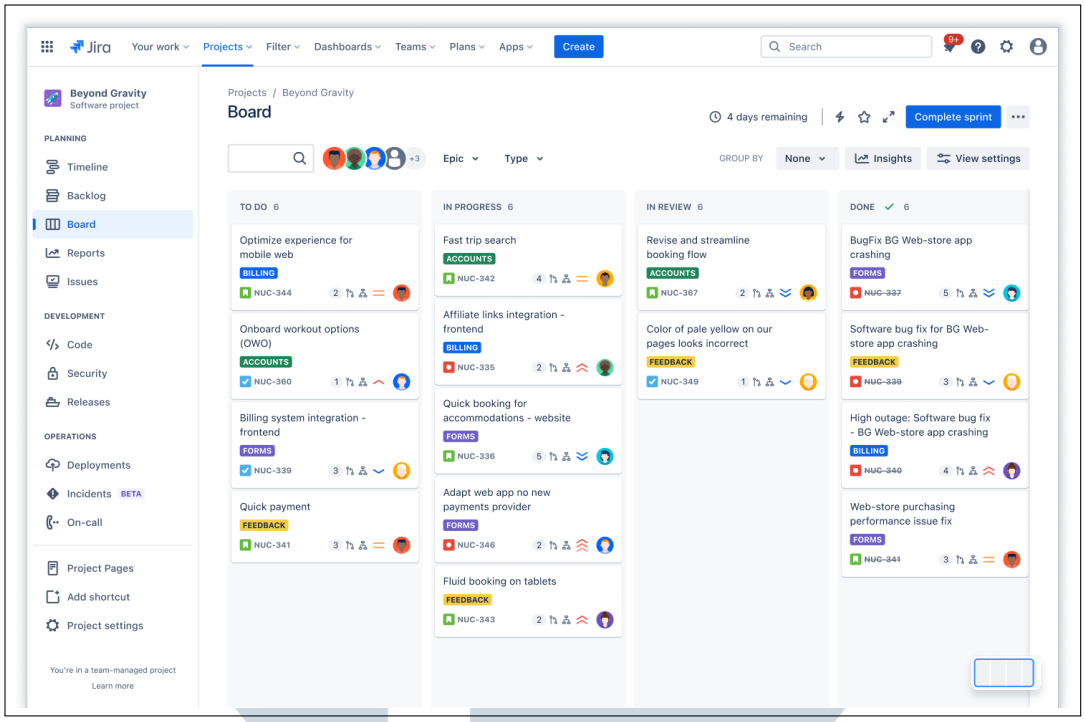
Gambar 3.10 menggambarkan alur umum proses API *Automation Testing* yang dijalankan, mulai dari penerimaan tiket pengujian, penulisan skrip otomatisasi, hingga proses *pull request* dan *merge* serta pelaporan hasil pengujian melalui sistem manajemen tiket. Ilustrasi ini membantu memberikan pemahaman menyeluruh terhadap tahapan yang dilalui selama proses pengujian berjalan, serta menunjukkan keterkaitan antar aktivitas secara sistematis.

Selama masa magang, kegiatan dimulai dengan mempelajari konsep dasar dan praktik pengujian API, dari pemahaman awal hingga penerapan otomatisasi pengujian. Eksplorasi dilakukan secara mandiri menggunakan Postman. Dokumentasi API dan *user* manual digunakan untuk memahami struktur endpoint, parameter yang digunakan, metode HTTP (`GET`, `POST`, `PUT`,

`DELETE`), serta format data pada request dan response. Referensi berupa file `postman_collection.json` juga dimanfaatkan sebagai acuan skenario uji awal. Meskipun proses ini dilakukan secara otodidak, diskusi aktif dengan Senior *Automation Engineer* sangat membantu ketika menemui kesulitan teknis.

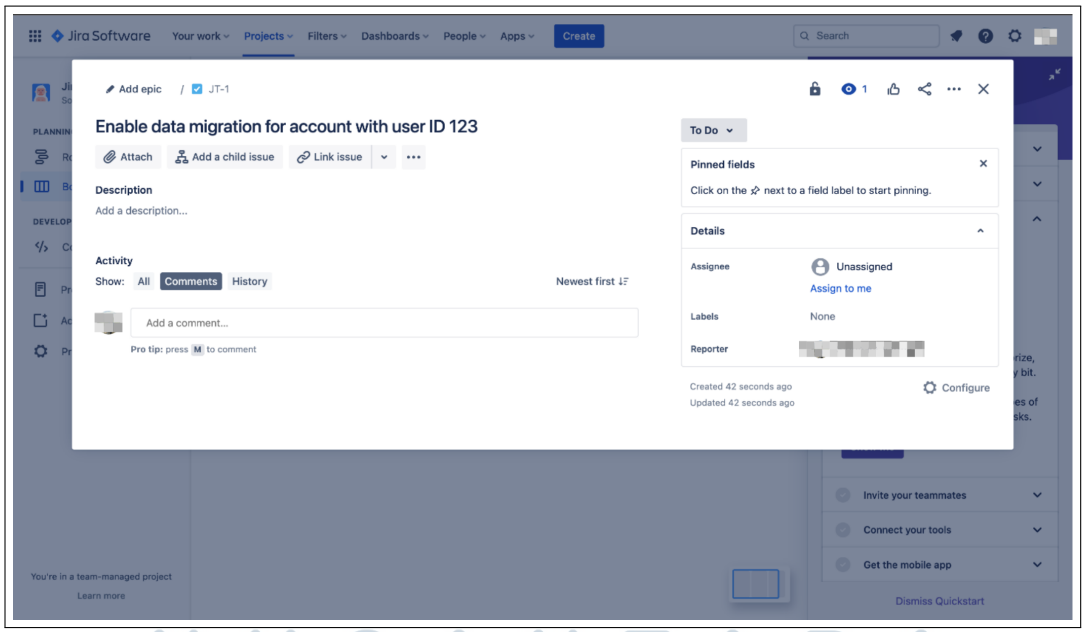
Setelah memahami alur pengujian dan integrasi API, proses dilanjutkan dengan penggunaan Katalon Studio alat otomasi yang digunakan oleh tim *Quality Engineering* dalam kegiatan *automation testing*. Pada tahapan ini, skrip pengujian otomatis mulai ditulis dan dijalankan berdasarkan dokumentasi yang tersedia. Skrip mencakup pengiriman request ke API, validasi terhadap status kode HTTP (seperti `200 OK`, `400 Bad Request`, `500 Internal Server Error`), serta verifikasi *response body* sesuai dengan *data contract* yang ditetapkan.

Setelah menyelesaikan fase eksplorasi dan latihan teknis, tanggung jawab baru diberikan dalam bentuk pengerjaan langsung tiket pengujian otomasi sebagai kontribusi terhadap proyek berjalan. Pengelolaan tiket dilakukan melalui platform JIRA. Setiap tiket mencantumkan deskripsi fitur, skenario pengujian, serta target hasil yang harus dicapai. Tugas utama dalam pengerjaan tiket mencakup menjalankan *test case* yang relevan sesuai instruksi dan mencatat hasil pengujian melalui komentar atau pembaruan status (*status update*) pada JIRA. Gambar 3.11 dan 3.12 menampilkan tampilan antarmuka JIRA secara umum, seperti yang digunakan dalam banyak proses pengelolaan tiket dan *sprint* di industri.



Gambar 3.11. Ilustrasi umum tampilan board Sprint di JIRA

Sumber: [7]

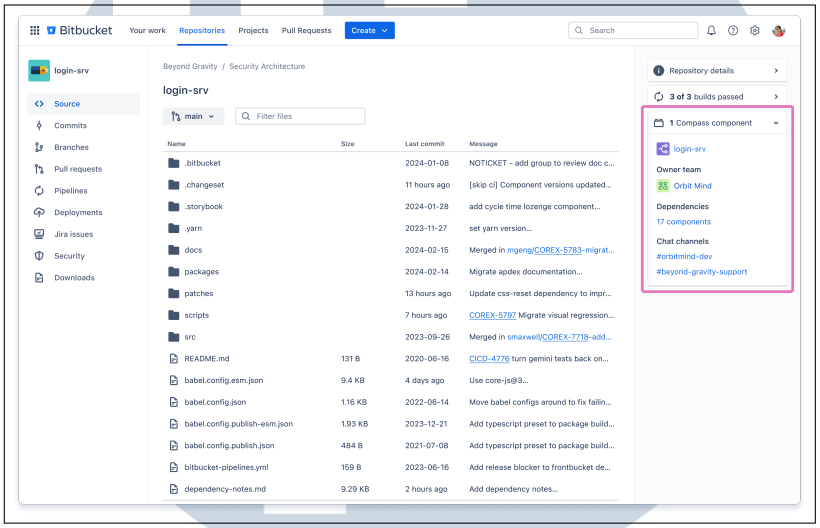


Gambar 3.12. Ilustrasi tampilan tiket pada platform JIRA

Sumber: [8]

Setelah mendapatkan tiket pengujian melalui JIRA, proses selanjutnya mengacu pada dokumentasi teknis yang disimpan di dalam *repository* Bitbucket.

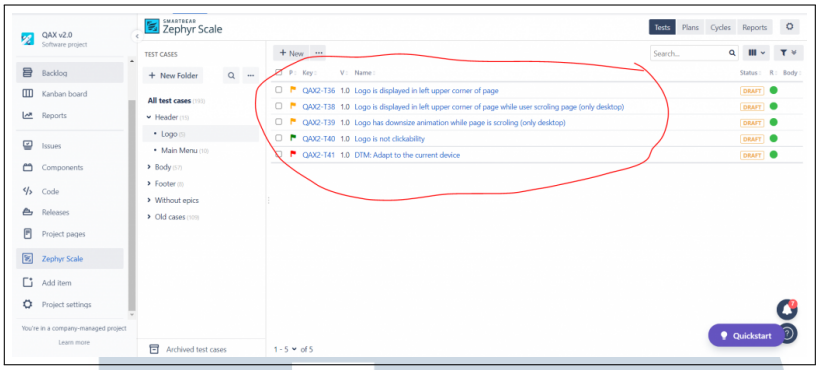
Gambar 3.13 memperlihatkan ilustrasi umum tampilan *repository* Bitbucket, bukan tampilan spesifik milik AdIns, mengingat informasi di dalamnya bersifat rahasia. *Repository* ini umumnya berisi *file-file* penting seperti skrip otomatisasi, konfigurasi proyek, *file* pendukung integrasi, serta dokumentasi teknis lain yang diperlukan dalam proses pengembangan dan pengujian.



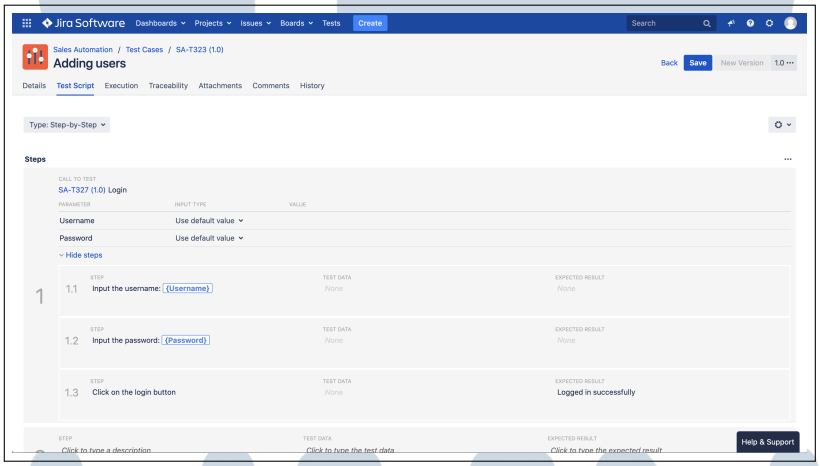
Gambar 3.13. Ilustrasi Umum Repositori Dokumentasi pada Bitbucket
Sumber: [9]

Bitbucket sendiri berperan sebagai platform penyimpanan dan kolaborasi kode berbasis Git, yang memungkinkan setiap anggota tim untuk mengakses dan memperbarui dokumentasi atau skrip pengujian secara terpusat dan *versioned*. Dokumen pengujian yang tersimpan di dalamnya menjadi referensi penting sebelum proses otomasi dijalankan.

Seluruh skenario dan hasil pengujian yang telah dilakukan kemudian dicatat dan dikelola melalui Zephyr Scale. Zephyr Scale merupakan alat manajemen pengujian yang terintegrasi langsung dengan JIRA untuk kebutuhan perencanaan dan pelacakan *test cycle*. Ilustrasi umum mengenai tampilan pengelolaan *test scenario*, *test case* dan *test steps* pada Zephyr Scale ditampilkan pada Gambar 3.14 dan Gambar 3.15.



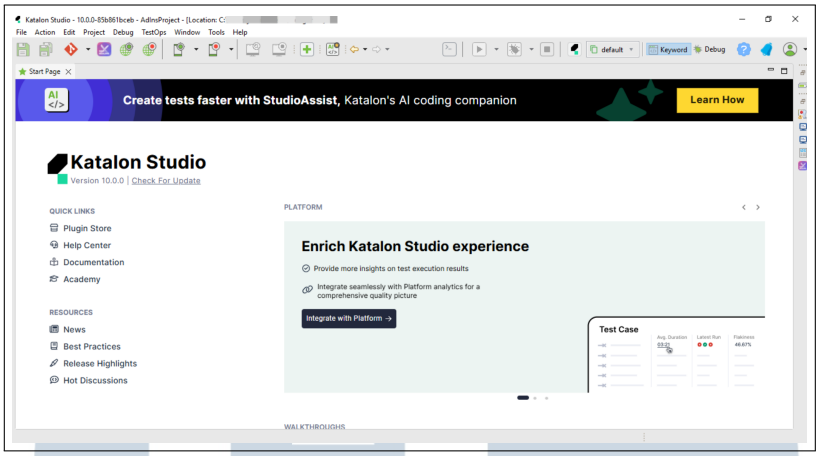
Gambar 3.14. Ilustrasi Umum Test Case pada Zephyr Scale
Sumber: [10]



Gambar 3.15. Ilustrasi Umum Test Steps yang Ada di Test Case pada Zephyr Scale
Sumber: [11]

Ketiga platform ini JIRA, Bitbucket, dan Zephyr Scale saling terintegrasi dalam satu ekosistem pengujian yang terpadu. JIRA digunakan sebagai sistem manajemen tugas dan pelaporan *defect*, Bitbucket berperan sebagai pusat dokumentasi teknis dan skrip otomasi, sedangkan Zephyr Scale mengelola seluruh siklus pengujian termasuk pelacakan hasil dan cakupan *test case*. Integrasi ini memungkinkan kolaborasi lintas tim menjadi lebih efisien dan transparan, serta memudahkan evaluasi progres pengujian dari satu platform tunggal.

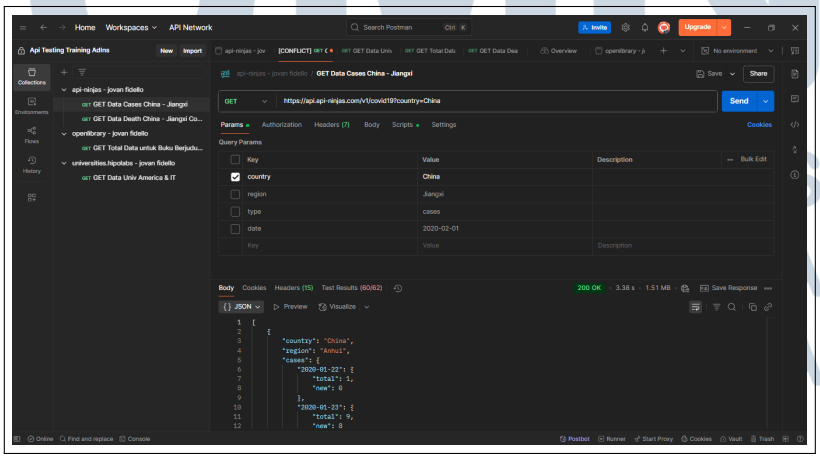
Pada tahap pelaksanaan, proses dimulai dengan membuka Katalon Studio sebagai platform utama untuk menjalankan otomasi pengujian (Gambar 3.16), kemudian menyusun skrip pengujian sesuai dengan skenario yang telah ditentukan.



Gambar 3.16. Tampilan awal Katalon Studio
Sumber: Dokumentasi Pribadi

Berikut beberapa uraian aktivitas yang dilakukan dalam proses API *Automation Testing* antara lain:

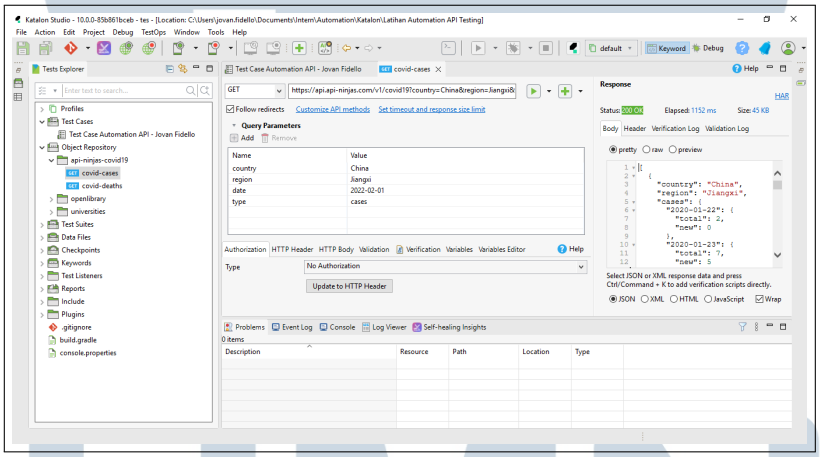
- Melakukan Validasi Response dan Status Code
  Sebelum menyusun skrip untuk *automation testing*, tahap awal yang dilakukan adalah proses validasi terhadap respons API yang akan diuji. Validasi ini dilakukan secara ganda, yaitu dengan menggunakan aplikasi Postman mandiri serta fitur Postman bawaan yang tersedia di dalam Katalon Studio. Tujuan dari pendekatan ganda ini adalah untuk memastikan bahwa respons dari *endpoint* API konsisten, stabil, dan sesuai dengan spesifikasi yang telah ditentukan baik pada lingkungan pengujian manual maupun saat diotomasi.



Gambar 3.17. Contoh validasi response API menggunakan aplikasi Postman
Sumber: Dokumentasi Pribadi

Validasi mencakup pemeriksaan terhadap status kode dari *response* yang diterima, seperti `200 OK`, `400 Bad Request`, `401 Unauthorized`, hingga `500 Internal Server Error`. Selain itu, validasi juga dilakukan terhadap struktur data dan isi (konten) yang dikembalikan oleh server, agar sesuai dengan format yang diharapkan dan memenuhi *data contract* dari sistem.

Setelah validasi dilakukan pada Postman, tahap selanjutnya adalah melakukan validasi ulang melalui Postman di dalam Katalon Studio. Fitur ini membantu mengintegrasikan skenario pengujian langsung ke dalam platform otomasi, sekaligus menjadi jembatan antara pengujian manual dan otomatis. Dengan validasi ganda ini, potensi kesalahan akibat perbedaan lingkungan atau konfigurasi dapat diminimalisir.



Gambar 3.18. Contoh validasi response API menggunakan Postman bawaan Katalon Studio
Sumber: Dokumentasi Pribadi

Adapun hasil yang diharapkan (*expected result*) dari proses validasi ini adalah status kode `200 OK`, yang menandakan bahwa permintaan telah berhasil diproses oleh server dan tidak terjadi kesalahan pada sisi logika maupun data.

- Mendevelop dan Mengeksekusi Script API Automation Testing
  Setelah proses validasi manual selesai dilakukan, tahap selanjutnya adalah menyusun skrip pengujian otomatis menggunakan Katalon Studio. Penyusunan skrip ini bertujuan untuk menguji berbagai kondisi dari *endpoint* tertentu secara konsisten, efisien, dan berulang tanpa intervensi manual. Seluruh pengujian disusun berdasarkan skenario dan konfigurasi yang telah ditentukan sebelumnya.

Skrip pengujian dalam Katalon Studio ditulis menggunakan bahasa pemrograman Groovy, yaitu bahasa dinamis berbasis Java yang mendukung pemrograman scripting sekaligus tetap kompatibel dengan pustaka Java yang ada. Bahasa ini dipilih karena sintaksnya yang ringkas, fleksibel, dan mudah dipahami, sangat cocok untuk kebutuhan otomasi dalam pengujian API.

Dalam proses pengembangan skrip, digunakan berbagai pustaka dan fungsi standar dari Katalon Studio yang berfungsi untuk mengirim permintaan HTTP, memverifikasi status, mengevaluasi elemen dalam respons, serta melakukan *parsing* dan validasi terhadap data JSON. Beberapa fungsi dan pustaka yang paling sering digunakan saat membuat skrip API *automation testing* selama mengerjakan tiket antara lain:

- `WS.sendRequest(request)`: Digunakan untuk mengirim *request* API sesuai dengan *Test Object* yang telah disiapkan.
- `WS.verifyResponseStatusCode(response, expectedCode)`: Memastikan status kode HTTP dari *response* sesuai dengan nilai yang diharapkan, seperti `200 OK` atau `401 Unauthorized`.
- `WS.verifyElementPropertyValue(response, 'path.to.property', expectedValue)`: Mengecek nilai properti tertentu dalam JSON *response* untuk memastikan nilainya sesuai.
- `new JsonSlurper().parseText(response.getResponseBodyContent())`: *Parsing* data *response* berbentuk JSON agar dapat dimanipulasi atau dianalisis lebih lanjut dalam bentuk struktur data.
- `assert`: Digunakan untuk membuat validasi tambahan secara eksplisit di dalam kode, yang akan gagal jika kondisi tidak terpenuhi.

Contoh berikut merupakan pseudocode dari skrip API *Automation Testing* untuk produk C yang telah dibuat untuk mengeksekusi *Test Step* yang ada pada *Test Case* di Zephyr Scale. Setiap pengujian bertujuan untuk memverifikasi respons dari sistem terhadap skenario tertentu, mulai dari validasi input yang salah hingga pengujian dokumen baru dan simulasi kondisi saldo habis.

Langkah Awal Setup Katalon:
Pseudocode ini memuat langkah awal yang diperlukan untuk mempersiapkan eksekusi pengujian, termasuk inisialisasi *library*, pemuatan konfigurasi JSON, serta deklarasi fungsi pembantu seperti `generateRandomName()`.

```
IMPORT libraries Katalon (WebService, TestObject, JSON parser, dll)
LOAD JSON config dari file 'xxxxxxxxx.json'
INISIALISASI APIFullService untuk DB operations
INISIALISASI SimpleDateFormat untuk tanggal

FUNCTION generateRandomName():
    BUAT array karakter A-Z dan a-z
    GENERATE random length 5-10 karakter
    RETURN string random hasil gabungan karakter
```

Kode 3.8: Contoh pseudocode setup awal

Menguji dengan Base URL yang Salah:
Pengujian ini bertujuan untuk memastikan sistem mengembalikan error 404 ketika permintaan dikirim ke alamat *endpoint* yang tidak dikenali.

```
BUAT request object dengan semua parameter valid dari JSON config
SET request URL = GlobalVariable.BaseURLAPI + 'Invalid'
KIRIM request menggunakan WS.sendRequest()

EXPECTED: status code = 404, message = "404 Not Found"
IF response status == 404 THEN
    PRINT "Expected response received: 404 Not Found"
    PRINT "Test base URL invalid PASSED"
ELSE
    PRINT "Unexpected status code: " + actual_status
```

Kode 3.9: Contoh pseudocode uji base URL tidak valid

Menguji dengan API Key yang Invalid:
Pseudocode ini menguji validasi keamanan terhadap API *Key* yang salah. Sistem seharusnya mengembalikan status 401 dan kode eror khusus.

```
BUAT request object dari Object Repository
SET HTTP header properties:
    x-api-key = jsonConfig.apiKeyInvalid + jsonConfig.tenantCode
KIRIM request

EXPECTED: status code = 401
IF response status == 401 THEN
    VERIFY response.status.code == 9008
    VERIFY response.status.message == 'API Key salah'
    PRINT "Expected response received: API Key salah"
    PRINT "Test case api key invalid PASSED"
```

```
ELSE
    PRINT "Unexpected status code: " + response.statusCode
    ASSERT response.statusCode == 401
```

Kode 3.10: Contoh pseudocode uji API Key tidak valid

Menguji dengan API Key yang Kosong:
Pengujian ini dilakukan untuk memastikan bahwa ketika API *Key* tidak diisi, sistem tetap menolak permintaan dan memberikan pesan eror yang sesuai.

```
BUAT request object dari Object Repository
SET HTTP header properties:
    x-api-key = jsonConfig.apiKeyBlank + jsonConfig.
    tenantCodeBlank (empty values)
KIRIM request

EXPECTED: status code = 401
IF response status == 401 THEN
    VERIFY response.status.code == 9008
    VERIFY response.status.message == 'API Key not found in
    request.'
    PRINT "Expected response received: API Key not found in
    request."
    PRINT "Test case api key kosong PASSED"
ELSE
    PRINT "Unexpected status code: " + response.statusCode
```

Kode 3.11: Contoh pseudocode uji API Key kosong

Menguji dengan Tenant Code yang Invalid:
Validasi terhadap *Tenant Code* sangat penting untuk mencegah akses ke tenant yang tidak terdaftar. Tes ini mengecek sistem terhadap kombinasi `x-api-key` yang tidak valid.

```
BUAT request object dari Object Repository
SET HTTP header properties:
    x-api-key = jsonConfig.apiKey + jsonConfig.
    tenantCodeInvalid
KIRIM request

EXPECTED: status code = 401
IF response status == 401 THEN
    VERIFY response.status.code == 9008
    VERIFY response.status.message == 'Tenant tidak ditemukan
    '
    PRINT "Expected response received: Tenant tidak ditemukan
    "
```

```
    PRINT "Test case tenant code invalid PASSED"
ELSE
    PRINT "Unexpected status code: " + response.
   statusCodeBUAT request object dari Object Repository
SET HTTP header properties:
    x-api-key = jsonConfig.apiKey + jsonConfig.
   tenantCodeInvalid
KIRIM request

EXPECTED: status code = 401
IF response status == 401 THEN
    VERIFY response.status.code == 9008
    VERIFY response.status.message == 'Tenant tidak ditemukan
   '
    PRINT "Expected response received: Tenant tidak ditemukan
   "
    PRINT "Test case tenant code invalid PASSED"
ELSE
    PRINT "Unexpected status code: " + response.statusCode
```

Kode 3.12: Contoh pseudocode uji Tenant Code tidak valid

Menguji dengan Tenant Code yang Kosong:
Pseudocode berikut menguji tanggapan sistem ketika bagian tenant dari API *Key* tidak diberikan. Respons yang benar harus berupa status 401 dan pesan spesifik terkait tenant.

```
BUAT request object dari Object Repository
SET HTTP header properties:
    x-api-key = jsonConfig.apiKey + jsonConfig.
   tenantCodeBlank (empty)
KIRIM request

EXPECTED: status code = 401
IF response status == 401 THEN
    VERIFY response.status.code == 9008
    VERIFY response.status.message == 'Api Key does not
   contains tenant code. example: apikey@tenantCode'
    PRINT "Expected response received: Api Key does not
   contains tenant code. example: apikey@tenantCode"
    PRINT "Test case tenant code kosong PASSED"
ELSE
    PRINT "Unexpected status code: " + response.statusCode
```

Kode 3.13: Contoh pseudocode uji Tenant Code kosong

Menguji dengan Data Dokumen Lama (Duplikat):
Pengujian ini bertujuan untuk memverifikasi bahwa sistem menolak dokumen dengan ID transaksi atau nomor dokumen yang sudah pernah digunakan sebelumnya.

```
SET current date menggunakan SimpleDateFormat("yyyy-MM-dd")
UPDATE JSON config:
    docDate = current date
    documentNumber = documentNumberOld
    documentTransactionId = documentTransactionIdOld
    docName = docNameOld

BUAT request dengan parameter old document data
KIRIM request

EXPECTED: error duplicate document
VERIFY response.status.code == 4029
VERIFY response.status.message == "Dokumen dengan ID
    Transaksi ${documentTransactionIdOld} sudah ada"
PRINT "Expected response received: Dokumen dengan ID
    Transaksi ${documentTransactionIdOld} sudah ada"
PRINT "Test case dokumen lama PASSED"
```

Kode 3.14: Contoh pseudocode uji dokumen lama/duplikat

Menguji dengan Data Dokumen Baru (Sukses dan Verifikasi DB):
Tes ini merupakan skenario utama untuk memastikan dokumen baru dapat diproses dengan sukses, dan hasilnya tercermin dalam *database* dengan struktur serta nilai kolom yang benar.

```
// Step 1: Login untuk cek saldo
BUAT login request dengan credentials dari JSON config
KIRIM login request
IF login berhasil (status 200) THEN
    EXTRACT access_token dari response
    INITIALIZE Balance = ""

    // Step 2: Check saldo postpaid
    BUAT balance check request
    SET Authorization header = 'Bearer ' + access_token
    SET request body
    KIRIM balance check request

    IF balance check berhasil (status 200) THEN
        PARSE response untuk cari saldo
        SET Balance = saldo yang ditemukan
```

```
17
18        // Step 3: Evaluasi perlu top-up atau tidak
19        SET needTopUp = false
20        IF Balance ditemukan THEN
21            PRINT "Saldo ditemukan: " + Balance.
    currentBalance
22            SET needTopUp = (currentBalance == 0)
23        ELSE
24            PRINT "Saldo tidak ditemukan!"
25            SET needTopUp = true
26
27        // Step 4: Lakukan top-up jika diperlukan
28        IF needTopUp == true THEN
29            PRINT "Saldo kosong atau tidak ditemukan,
    melakukan Top Up..."
30
31            // Login dengan kredensial admin top-up
32            BUAT login request dengan username& password
33            KIRIM login request
34            IF login admin berhasil (status 200) THEN
35                EXTRACT admin access_token
36
37                // Buat top-up request
38                BUAT top-up request object
39                SET headers:
40                    Authorization = 'Bearer ' +
    admin_access_token
41                    x-api-key
42                    Content-Type = 'application/json'
43
44                GENERATE unique reference number = "xxxx" +
    timestamp
45                SET balance expired date = current date + 1
    tahun
46                SET request body dengan semua parameter top-
    up
47                KIRIM top-up request
48
49                PARSE top-up response
50                IF statusCode == 0 AND message == 'Success'
    THEN
51                    PRINT "Top Up berhasil!"
52                ELSE
```

```
                    PRINT "Top Up gagal! Status: [code], Message: [message]"
                    ASSERT failure "Top Up Saldo Gagal!"
        ELSE
            PRINT "Saldo cukup, tidak perlu Top Up."

// Step 5: Test dengan dokumen baru
GENERATE unique identifiers:
    uniqueId = current date format
    uniqueName = generateRandomName()
    fullDocId = "xx-" + uniqueName + "-xxxx-xxxx-" + uniqueId

BUAT request dengan semua parameter dari JSON config
UPDATE request parameters:
    documentNumber = fullDocId
    documentTransactionId = fullDocId
    docName = fullDocId
    docDate = current date

KIRIM request
PARSE response menggunakan JsonSlurper

// Evaluasi response berdasarkan HTTP status
IF HTTP status == 200 THEN
    EXTRACT status.code dan status.message dari response

    IF status.code == 0 THEN
        VERIFY status.message == 'null' (success case)

        // Verifikasi konsistensi database
        CALL VerifyDB()
        IF DB verification == true THEN
            PRINT "    DB Check PASSED: Kolom dan nilainya cocok."
        ELSE
            ASSERT failure "    DB Check GAGAL: Struktur atau nilai kolom antara tidak sesuai!"

    ELSE IF status.code == 5062 THEN
        VERIFY status.message == 'Gagal. Silahkan coba kembali nanti'

    ELSE
```

```
92          PRINT "    Kode status tidak dikenali: " + statusCode
    + ", Pesan: " + statusMessage
93
94 ELSE
95      PRINT "    HTTP Response Error: " + HTTP_status_code
```

Kode 3.15: Contoh pseudocode uji dokumen baru lengkap

Menguji dengan Saldo yang Sudah Habis:
Pseudocode ini mensimulasikan kondisi saldo habis dengan mengulang pengiriman dokumen sampai limit saldo terpakai, lalu mengecek bahwa sistem menolak permintaan berikutnya dengan kode eror gagal.

```
1  // Langkah 1: Evaluasi saldo saat ini dan tentukan strategi
2  IF Balance.currentBalance > 0 AND Balance.currentBalance < 4
       THEN
3      PRINT "Saldo tersedia: " + currentBalance + ". Melakukan
       loop request sampai habis..."
4
5      // Loop untuk menghabiskan saldo
6      WHILE Balance.currentBalance > 0 DO
7          // Generate unique document untuk setiap iterasi
8          GENERATE fullDocIdx = "xx-" + generateRandomName() +
       "-xxxx-xxxx-" + current_date_format
9
10         BUAT request dengan parameter unik
11         UPDATE semua document identifiers dengan fullDocIdx
12         KIRIM request
13
14         PARSE response
15         EXTRACT status.code dan status.message
16
17         IF status.code == 0 THEN
18             PRINT "Request sukses, mengurangi saldo..."
19             // Continue loop
20         ELSE IF status.code == 5062 THEN
21             PRINT "Saldo telah habis."
22             BREAK dari loop
23         ELSE
24             PRINT "Gagal, " + status.message
25             BREAK dari loop
26     END WHILE
27
28 ELSE IF Balance.currentBalance >= 4 THEN
29     PRINT "Saldo saat ini adalah " + currentBalance + ",
```

```
    masih cukup tinggi."
30      PRINT "Skip looping karena akan menghasilkan terlalu
    banyak dokumen baru."
31
32  ELSE
33      PRINT "Saldo kosong atau tidak valid. Tidak bisa
    melakukan proses."
34
35  // Langkah 2: Test final - kondisi saldo habis
36  PRINT "Menjalankan test case untuk kondisi saldo habis..."
37  GENERATE saldoHabisDocId = generateRandomName() + "-xxxx-xxxx
    -" + current_date_format
38
39  BUAT request dengan document ID yang baru
40  UPDATE semua identifiers dengan saldoHabisDocId
41  KIRIM request
42
43  PARSE response dan extract responseBody
44  IF HTTP status == 200 THEN
45      IF response.status.code == 5062 THEN
46          VERIFY response.status.message == 'Gagal. Silahkan
    coba kembali nanti'
47          PRINT "    Test case kondisi saldo habis PASSED"
48      ELSE
49          PRINT "    Diharapkan gagal karena saldo habis, namun
     response: " + status.message
50  ELSE
51      PRINT "    HTTP Response Error: " + HTTP_status_code
```

Kode 3.16: Contoh pseudocode uji kondisi saldo habis

Pseudocode-pseudocode tersebut merupakan representasi terstruktur dari skenario pengujian yang dijalankan untuk mengevaluasi keandalan dan keamanan API secara menyeluruh. Setiap *test case* disusun untuk meniru kondisi nyata yang mungkin terjadi saat API digunakan oleh *end-user* atau sistem lain. Dengan pendekatan berbasis pengujian otomatis seperti ini, seluruh proses menjadi lebih efisien, mudah dipelihara, dan dapat dijalankan berulang untuk regresi testing saat terjadi perubahan sistem atau fitur baru ditambahkan.

- Meninjau dan Mengevaluasi Hasil Script Automation Testing
  Setelah seluruh skenario pengujian otomatis berhasil dijalankan dan memberikan hasil yang sesuai ekspektasi (berstatus *Pass*), tahap selanjutnya

adalah proses peninjauan dan evaluasi hasil oleh tim. Evaluasi ini dilakukan untuk memastikan bahwa tidak terdapat potensi kesalahan logika, kegagalan integrasi antarmodul, ataupun ketidaksesuaian terhadap standar coding yang berlaku di lingkungan tim ST.

Proses ini dilaksanakan bersama Senior *Automation Engineer*, yang akan melakukan *code review* menyeluruh terhadap skrip yang telah dikembangkan. Jika ditemukan bagian yang perlu diperbaiki seperti pemakaian variabel yang masih *hardcoded*, ketidaksesuaian dalam penggunaan variabel global, ataupun variabel yang seharusnya diambil dari konfigurasi `.json`, maka pengembang skrip wajib melakukan revisi sebelum melanjutkan ke tahap berikutnya.

Best practice dalam penulisan kode automation testing menjadi fokus utama dalam proses *review* ini. Tujuannya adalah untuk memastikan bahwa skrip yang dikembangkan bersifat *maintainable*, *reusable*, dan sesuai standar otomatisasi perusahaan. Jika seluruh masukan telah diakomodasi dan kode dinyatakan valid oleh *reviewer*, maka skrip tersebut siap untuk dilanjutkan ke tahap *pull request*.

- Melakukan Pull Request dan Merging
  Setelah proses *code review* disetujui, langkah berikutnya adalah melakukan *pull request* (PR) terhadap *repository* utama di Bitbucket. Tahapan ini bertujuan untuk menggabungkan (*merge*) perubahan yang telah dilakukan ke dalam *branch* pengembangan utama atau *branch* fitur terkait.

  Proses *pull request* biasanya dilakukan melalui *command line interface* (CLI) menggunakan beberapa contoh perintah Git berikut:

  - `git checkout -b nama-branch` untuk membuat dan berpindah ke *branch* baru,
  - `git add .` untuk menambahkan seluruh perubahan,
  - `git commit -m "pesan commit"` untuk membuat *commit* dengan deskripsi yang jelas,
  - `git push origin nama-branch` untuk mengunggah perubahan ke *remote repository*.

  Setelah *push* dilakukan, proses *pull request* langsung dieksekusi secara otomatis melalui integrasi antara sistem lokal dengan Bitbucket, tanpa

perlu diajukan secara manual melalui antarmuka web. Mekanisme ini dimungkinkan karena Bitbucket telah dikonfigurasi untuk menerima *pull request* secara langsung dari perintah *git push* apabila *branch* baru telah terbentuk dan memenuhi kriteria tertentu.

Setelah *pull request* tercatat di Bitbucket, maka akan masuk ke tahap *review* oleh Senior *Automation Engineer*. Jika telah disetujui, maka kode akan digabungkan (*merge*) ke dalam *branch* utama.

3.3.3 Training (Hardskill & Softskill)

Selama mengikuti program magang di PT Adicipta Inovasi Teknologi, kegiatan pelatihan terbagi ke dalam dua kategori utama, yaitu pelatihan *hardskill* dan *softskill*. Seluruh pelatihan ini bertujuan memperkuat kemampuan teknis serta membentuk sikap profesional yang dibutuhkan dalam dunia kerja.

Pelatihan *hardskill* difokuskan pada peningkatan pemahaman teknis terkait proses bisnis industri *multifinance* serta keterampilan dalam pengembangan dan pengujian sistem, khususnya yang berkaitan dengan produk CONFINS. Materi pelatihan yang diterima antara lain:

- SQL (Structured Query Language)
  Materi ini mencakup dasar-dasar penggunaan SQL untuk melakukan query terhadap *database*, seperti perintah `SELECT`, `JOIN`, `WHERE`, `ALTER`, serta manipulasi data lainnya dalam proses validasi sistem. Pembelajaran juga mencakup penggunaan *query functions*, pembuatan *view*, serta implementasi *stored procedure* dan *function* guna mengotomatisasi pengolahan data dalam skenario pengujian yang kompleks.
- Introduction to Multifinance
  Menjelaskan konsep industri *multifinance*, termasuk jenis layanan pembiayaan, produk keuangan yang tersedia, dan peran sistem teknologi informasi dalam menunjang aktivitas bisnis.
- CONFINS Overview
  Menyajikan gambaran umum mengenai sistem CONFINS, meliputi arsitektur sistem serta modul-modul utama yang saling terintegrasi untuk mendukung kelancaran proses bisnis dalam perusahaan *multifinance*.

- Software Development Life Cycle (SDLC)
  Membahas tahapan dalam siklus pengembangan perangkat lunak mulai dari analisis kebutuhan, desain, implementasi, pengujian, hingga pemeliharaan. Materi juga menyoroti keterlibatan tim *Software Testing* di tiap tahap.

- Quality Control in Software Testing
  Menjelaskan metode pengendalian mutu dalam proses pengujian perangkat lunak, termasuk pentingnya dokumentasi pengujian, proses *review*, dan pelaporan hasil testing.

- Basic Finance dan Basic Accounting
  Memberikan pemahaman dasar terkait konsep keuangan dan akuntansi yang diterapkan dalam pengembangan sistem *multifinance*, seperti jurnal transaksi, neraca, serta laporan laba rugi.

- CONFINS Module A Part 1 – Part 6
  Menjelaskan proses-proses bisnis utama dalam modul A secara bertahap, serta menjabarkan peran fitur-fitur dalam modul tersebut untuk mendukung kegiatan operasional *multifinance*.

- CONFINS Module B
  Fokus pelatihan ini adalah pemahaman terhadap proses bisnis yang terdapat dalam modul B dan keterkaitannya dengan modul lain dalam sistem CONFINS dari perspektif pengguna bisnis.

- CONFINS Module C Part 1
  Menyampaikan alur kerja bisnis pada modul C, serta logika proses bisnis secara fungsional yang diakomodasi oleh modul ini dalam sistem CONFINS.

Pelatihan *softskill* bertujuan membentuk karakter profesional dan kesiapan mental dalam menghadapi lingkungan kerja yang dinamis. Materi pelatihan yang diberikan meliputi:

- Basic Mentality
  Sesi ini berfokus pada pembentukan pola pikir positif, ketangguhan, kedisiplinan, dan rasa tanggung jawab dalam menyelesaikan pekerjaan. Selain itu, peserta juga memperoleh wawasan tentang pentingnya inisiatif, komunikasi efektif, serta etika kerja di lingkungan profesional. Materi pelatihan turut membahas visi dan misi perusahaan, nilai-nilai inti

(*core values*) yang dijunjung tinggi yaitu *INTEGRITY*, *COMMITMENT*, *RESPECT*, *SYNERGY*, dan *INNOVATIVE*, serta pengenalan terhadap budaya kerja perusahaan yang berlandaskan pada 3 *Agile* DNA yang menjadi fondasi dalam menjalankan kegiatan bisnis dan kolaborasi tim.

Setiap pelatihan yang diselenggarakan dilengkapi dengan ujian atau *exam* sebagai bentuk evaluasi pemahaman terhadap materi yang telah disampaikan. Nilai dari ujian ini turut menjadi salah satu indikator dalam penilaian performa selama program magang berlangsung. Setelah setiap sesi, peserta juga diminta memberikan umpan balik terhadap pelaksanaan pelatihan, baik dari segi isi materi, durasi waktu, hingga kualitas penyampaian oleh instruktur. Evaluasi ini digunakan sebagai masukan untuk peningkatan kualitas pelatihan di masa mendatang.

Pelatihan-pelatihan ini memberikan bekal yang bermanfaat, baik dari segi teknis maupun non-teknis, dalam mempersiapkan diri menghadapi dunia kerja profesional.

3.4 Kendala dan Solusi yang Ditemukan

Selama kegiatan magang berlangsung, terdapat beberapa kendala yang muncul dalam menjalankan tugas-tugas pengujian maupun pengembangan otomatisasi. Kendala-kendala ini mencakup aspek teknis, pemahaman sistem, serta adaptasi terhadap lingkungan kerja yang profesional. Berikut ini adalah beberapa kendala yang dihadapi selama proses magang:

- Dunia *multifinance* menjadi hal yang baru karena latar belakang sebelumnya berasal dari bidang Informatika. Banyak istilah bisnis, alur proses, dan logika sistem yang belum dikenal, sehingga dibutuhkan waktu lebih untuk memahami keseluruhan konteks kerja.
- Beberapa *test case* memiliki penjelasan yang kurang rinci atau ambigu, sehingga menyulitkan dalam eksekusi pengujian dan interpretasi hasil yang diharapkan.
- Beberapa laporan *defect* memerlukan waktu lama untuk ditindaklanjuti, bahkan ada yang harus di-*reopen* lebih dari satu kali karena hasil perbaikannya belum sesuai ekspektasi tim ST.

- *Tools* dan *framework* pengujian otomatis seperti Katalon Studio, Git, dan Bitbucket masih tergolong baru, sehingga membutuhkan proses adaptasi teknis dan pembelajaran secara mandiri.
- Aplikasi Katalon Studio terkadang mengalami kendala teknis seperti gagal *login* atau tertutup sendiri secara tiba-tiba. Dalam beberapa kasus, aplikasi tampak seperti terhapus dari sistem tanpa alasan yang jelas.

Berdasarkan berbagai kendala tersebut, beberapa solusi telah dilakukan untuk mengatasi hambatan-hambatan yang muncul selama proses kerja magang:

- Untuk mempercepat pemahaman terhadap sistem dan dunia *multifinance*, dilakukan pembelajaran melalui dokumen *user* manual, mengikuti sesi pelatihan internal, serta diskusi langsung bersama supervisor dan tim senior.
- Klarifikasi langsung kepada *Test Analyst* menjadi langkah yang diambil ketika menemui *test case* yang ambigu, agar pelaksanaan pengujian dapat sesuai dengan skenario yang diharapkan.
- Proses *follow-up* secara aktif dilakukan kepada *Product Owner* maupun tim developer untuk mempercepat penanganan *defect*, serta koordinasi lanjutan dengan *Test Analyst* bila dibutuhkan.
- Untuk mempelajari *tools* baru, eksplorasi mandiri dilakukan secara rutin disertai dengan latihan menggunakan skrip-skrip sederhana, serta berkonsultasi dengan Senior *Automation Engineer* saat menghadapi kendala yang kompleks.
- Saat terjadi masalah teknis pada Katalon Studio, langkah yang diambil meliputi instalasi ulang, membersihkan *cache* sistem, dan login ulang ke akun resmi. Dokumentasi *troubleshooting* juga dibuat sebagai referensi jika masalah serupa terjadi kembali.