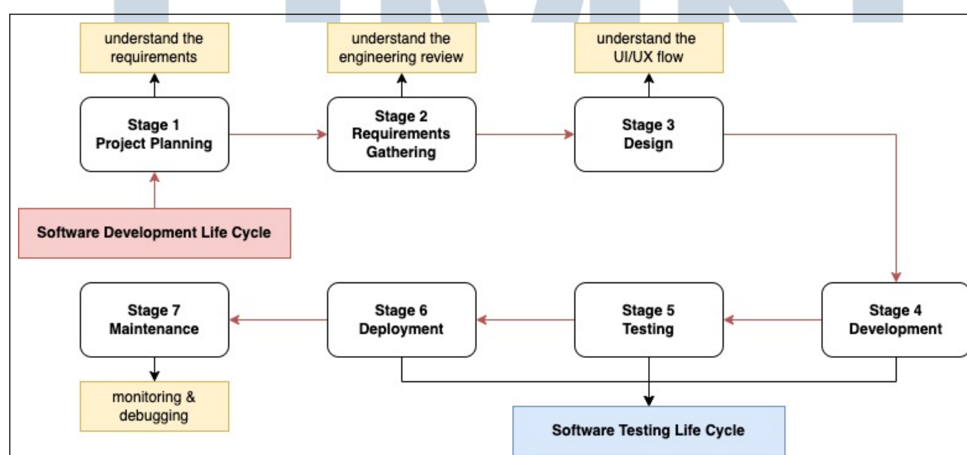
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Selama kegiatan *internship* di Sea Labs Indonesia, penulis berkesempatan untuk memahami struktur organisasi dan peran yang ada dalam tim, khususnya dalam divisi *Quality Assurance* (QA) sebagai *Quality Assurance Engineer Intern* di tim Studio Games 2 dari divisi *Games*. Divisi *Games* sendiri terbagi menjadi tiga studio, yaitu Studio Games 1, Studio Games 2, dan Gamification. Setiap studio memiliki komposisi tim yang terdiri dari beberapa peran utama, seperti GE (*Game Engineer*), BE (*Backend Engineer*), QA (*Quality Assurance*), PM (*Product Manager*), PD (*Product Designer*), dan TPM (*Technical Project Manager*).

Secara umum, peran *Quality Assurance* (QA) dalam organisasi ini terlibat di setiap fase *Software Development Life Cycle* (SDLC) seperti yang dapat dilihat di 3.1, dengan fokus utama pada fase *testing* yang dikenal dengan nama *Software Testing Life Cycle* (STLC). Setiap tahap dalam pengembangan memiliki peran spesifik bagi *Quality Assurance* (QA), yang mulai dari tahap perencanaan hingga pemeliharaan produk setelah diluncurkan.



Gambar 3.1. Diagram SDLC dan tugas QA didalamnya

Pada *Stage 1: Planning of Project*, tim *Quality Assurance* (QA) diundang untuk menghadiri *briefing* PRD (*Product Requirements Document*) yang dipimpin oleh

TPM (*Technical Project Manager*) dan PM (*Product Manager*). Pada tahap ini, tugas utama *Quality Assurance* (QA) adalah untuk memahami kebutuhan proyek yang akan dikembangkan dan memberikan umpan balik atau pertanyaan terkait hal-hal yang tidak jelas atau potensi alur yang terlewat dalam dokumen PRD (*Product Requirements Document*). Hal ini penting agar di awal proyek, tim *Quality Assurance* (QA) sudah bisa mengidentifikasi potensi masalah yang mungkin muncul di kemudian hari. Di *Stage 2: Analysis & Requirement Gathering*, *Quality Assurance* (QA) kembali diundang untuk menghadiri *briefing* ERD (*Engineering Review Document*) yang dipresentasikan oleh tim BE (*Backend Engineer*). Pada fase ini, *Quality Assurance* (QA) mempelajari arsitektur teknis produk, termasuk *stack* teknologi yang digunakan, seperti *Application Programming Interface* (API), database, dan logika aplikasi. Peran *Quality Assurance* (QA) adalah untuk memberikan umpan balik mengenai hal-hal yang belum jelas, atau kemungkinan alur logika yang terlewat dalam ERD, untuk memastikan bahwa pengembangan berjalan dengan lancar. Pada *Stage 3: Design*, *Quality Assurance* (QA) berperan dalam mengikuti review desain UI/UX yang diadakan bersama tim UI/UX. Di sini, *Quality Assurance* (QA) bertugas untuk memahami alur antarmuka pengguna, memastikan bahwa desain yang dibuat sesuai dengan kebutuhan pengguna dan dapat dijalankan dengan baik pada fase pengembangan nanti.

Memasuki *Stage 4: Development*, tim *Quality Assurance* (QA) mulai memasuki fase pengujian secara resmi setelah TPM membagikan estimasi waktu pengembangan. Pada tahap ini, meskipun fokus utama adalah pengembangan oleh tim *engineering*, *Quality Assurance* (QA) harus memastikan bahwa mereka siap untuk memulai proses pengujian yang telah direncanakan. Fase berikutnya adalah *Stage 5: Testing*, di mana *Quality Assurance* (QA) melakukan pengujian secara menyeluruh pada produk yang telah dikembangkan. Pada tahap ini, proses *Quality Assurance* (QA) terdiri dari beberapa langkah seperti pembuatan *test case*, eksekusi pengujian di *testing environment* untuk *functional test* dan *UAT environment* untuk *regression test*, dan manajemen *defect* untuk memastikan bahwa produk yang dikembangkan bebas dari bug dan sesuai dengan standar kualitas yang ditetapkan.

Setelah fase pengujian, produk siap untuk masuk ke *Stage 6: Deployment*, di mana *Quality Assurance* (QA) memberikan persetujuan untuk peluncuran produk ke lingkungan produksi setelah semua pengujian selesai dilakukan. Pada tahap ini, *Quality Assurance* (QA) memastikan bahwa produk yang dideploy ke pengguna

sudah bebas dari bug besar dan siap digunakan. Akhirnya, di *Stage 7: Maintenance*, *Quality Assurance* (QA) tetap berperan dalam memantau dan menangani masalah yang muncul di produksi. Tim *Quality Assurance* (QA) harus siap untuk membuat laporan bug, menganalisis insiden yang dilaporkan, serta memastikan bahwa masalah yang ditemukan diperbaiki dan dideploy kembali ke produksi.

Selain tugas-tugas *manual testing* yang terkait dengan *Software Development Life Cycle* (SDLC), *Quality Assurance Engineer* (QAE) juga memiliki tanggung jawab tambahan yang berfokus pada *automation testing* dan *performance testing*. *Performance testing* adalah salah satu aspek penting yang dilakukan untuk mengukur kinerja sistem di bawah beban tertentu, guna memastikan stabilitas produk. Seiring dengan berkembangnya fitur yang lebih kompleks dan beragam, *manual testing* terkadang menjadi waktu yang lebih lama dan berulang. Untuk mengatasi tantangan ini, QAE menggunakan berbagai tools otomatisasi, khususnya dalam pengujian *Application Programming Interface* (API), untuk mempercepat dan meningkatkan keandalan proses pengujian.

Secara keseluruhan, peran *Quality Assurance* (QA) di Studio Games 2 sangat penting dalam memastikan kualitas produk yang tinggi dan kelancaran proses pengembangan. Dengan terlibat dalam setiap fase *Software Development Life Cycle* (SDLC), mulai dari perencanaan hingga pemeliharaan, serta penggunaan *automation testing* untuk mengatasi tantangan *manual testing*, *Quality Assurance* (QA) berkontribusi besar terhadap kesuksesan tim dalam menghadirkan produk yang stabil dan berkualitas kepada pengguna.

3.2 Tugas yang Dilakukan

3.2.1 Application Programming Interface (API) Automation Testing

Seiring dengan perkembangan produk dan fitur yang semakin kompleks, proses *Quality Assurance* (QA) *Manual Testing* menjadi semakin memakan waktu, berulang, dan berpotensi meningkatkan risiko *defect* yang muncul pada tahap produksi. Terutama pada produk dengan fitur yang kompleks, di mana pengujian *end-to-end* yang dilakukan secara manual dapat berisiko tidak mencakup semua skenario yang mungkin terjadi. Oleh karena itu, untuk mengatasi permasalahan tersebut, tim *QA Engineering* (QAE) memanfaatkan berbagai *automation tools*

pengujian (*automation tools*), yang bertujuan untuk meningkatkan kecepatan dan keandalan pengujian tanpa menggantikan peran penting *QA tester*.

Salah satu pendekatan yang diterapkan adalah *Application Programming Interface (API) Automation* menggunakan alat *open-source* yang dikenal dengan Karate. Alat ini mengintegrasikan beberapa fitur seperti *API test-automation*, *mocking*, *performance testing*, dan *UI automation* dalam satu framework yang terstruktur. Karate menggunakan sintaksis *Behavior Driven Development* (BDD), yang memudahkan pemahaman oleh pemangku kepentingan non-teknis (misalnya, pengembang, analis bisnis, dan penguji), dengan menyediakan laporan dalam format yang mudah dibaca.

Pada proyek magang ini, penulis bertanggung jawab untuk menambahkan *API Automation Coverage* pada beberapa *Games* yang tergabung dalam Games 2, yang meliputi:

1. Shopee Capit
2. Shopee Fruity
3. Shopee Arcade Games: Slap

Automasi pengujian API dilakukan pada *environment* UAT (*User Acceptance Test*), yang merupakan lingkungan pengujian sebelum aplikasi diluncurkan ke produksi (*live*). Dalam proses pengujian, API diuji dengan mengirimkan *request body* dan *header* yang sesuai, kemudian memverifikasi respons yang diterima apakah sesuai dengan ekspektasi. Pengujian dilakukan dengan mencakup *positive* dan *negative cases*, untuk memastikan bahwa API berfungsi dengan baik dalam berbagai skenario.

Pengujian *Application Programming Interface* (API) dilakukan dengan menggunakan sintaksis BDD (*Behavior Driven Development*), yang mengadopsi struktur *Given-When-Then* untuk mendeskripsikan langkah-langkah pengujian. Potongan Kode 3.1 adalah contoh skenario pengujian *Application Programming Interface* (API) untuk salah satu permainan:

```
1 @regression-test
2 Feature: Verify Post Example
3
```

```
4   Background:
5     * def tcIDs = karate.tagValues.case
6     * callonce read('classpath:common.feature')
7     * callonce read('classpath:util/headers.feature@case=
    headerNewCookie')
8     * configure headers = headerAPI
9     * def exampleSchema = read ("classpath:api/public-api/post-
    example/schema/schema.json")
10    * def requestScenario = read('classpath:util/requests.
    feature@post')
11
12    Scenario Outline: <case>
13    Given def fullURL = BASE_URL + '/example/path/' + <
    example_id_1> + '?queryParam=' + <example_id_2>
14    And def reqBody =
15    """
16      {
17              "example_string": #(example_string),
18              "example_number": #(example_number)
19      }
20    """
21
22    When call requestScenario
23
24    Then match responseStatus == constant.status_code.success
25    And exampleSchema.data = schema
26    And match response == exampleSchema
27
28    @case=123456
29    Examples:
30      | case           | example_id_1 | example_id_2 |
    example_string    | example_number |
31      | "Example Case" | 1            | 2            | "This is
    string" | 123            |
32
```

Listing 3.1: Contoh potongan kode API Automation

M U L T I M E D I A
N U S A N T A R A

Pengujian *Application Programming Interface* (API) dapat dijalankan dengan menggunakan Maven, yaitu alat otomatisasi dan manajemen proyek yang dikembangkan oleh Apache Software Foundation dan digunakan untuk membangun dan mengelola dependensi proyek, serta menjalankan tes otomatis. Perintah dapat dijalankan dengan menuliskan perintah di bawah.

```
$ mvn clean test -Dtest=ExamplesTest

```

Setelah pengujian dijalankan, sistem otomatisasi Karate menghasilkan laporan dalam dua format utama, yaitu:

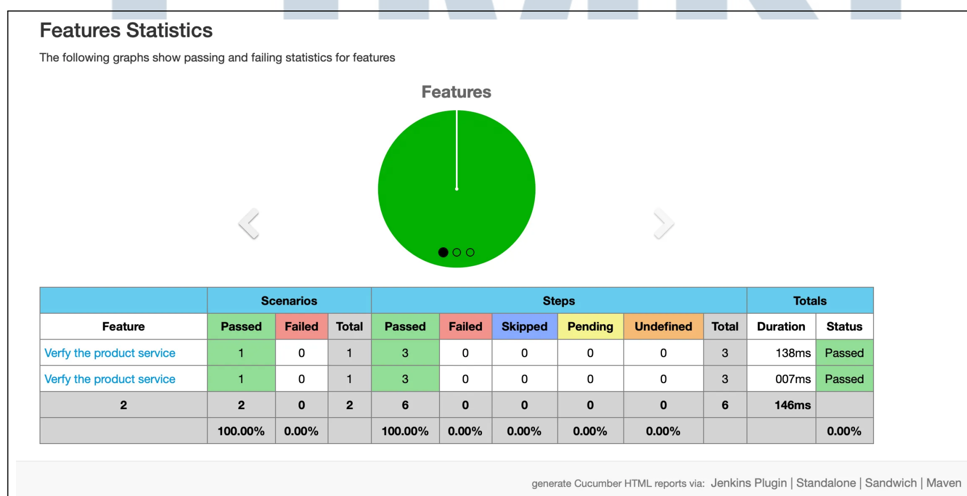
1. ***Karate Native Report*** yaitu laporan standar yang disediakan oleh Karate, yang dapat ditemukan pada direktori berikut:

```
$ ls -l /target/karate-reports/karate-summary.html

```

2. ***Cucumber HTML Report*** yaitu laporan yang disediakan oleh Cucumber dengan antarmuka pengguna yang lebih menarik dan informatif. Laporan ini dapat ditemukan pada direktori:

```
$ ls -l /target/cucumber-html-reports/overview-features.
html

```

Laporan-laporan ini memberikan informasi mendalam mengenai hasil pengujian, mencakup status pengujian (lulus/gagal), metrik pengujian, dan detail lainnya yang relevan. Jika ada *error*, seperti *assertion* yang tidak tepat, maka *Quality Assurance Engineer* (QAE) dapat melihat laporan hasil pengujian untuk kebutuhan *debugging*. Gambar 3.2 adalah contoh struktur laporan yang dihasilkan oleh Cucumber setelah pengujian selesai dijalankan.



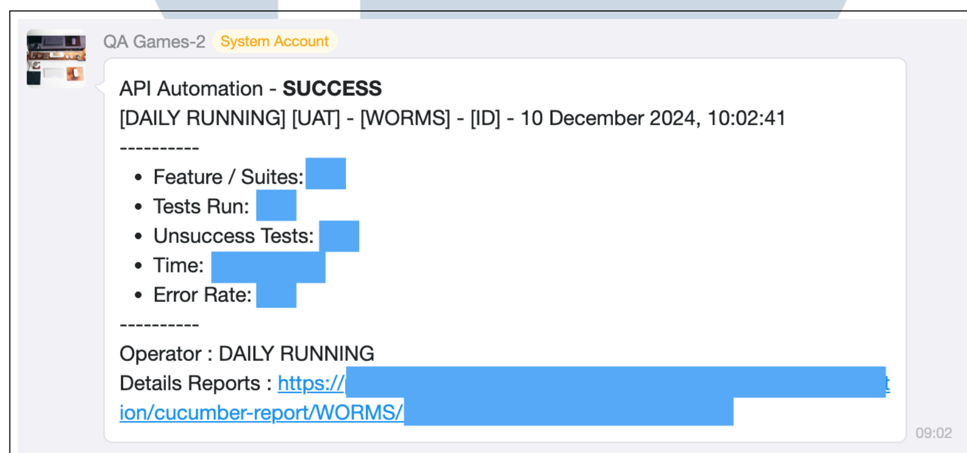
	Scenarios			Steps						Totals	
Feature	Passed	Failed	Total	Passed	Failed	Skipped	Pending	Undefined	Total	Duration	Status
Verfy the product service	1	0	1	3	0	0	0	0	3	138ms	Passed
Verfy the product service	1	0	1	3	0	0	0	0	3	007ms	Passed
2	2	0	2	6	0	0	0	0	6	146ms	
	100.00%	0.00%		100.00%	0.00%	0.00%	0.00%	0.00%			0.00%

Gambar 3.2. Contoh *Cucumber HTML Report*

Sumber: [6]

API Automation Test diintegrasikan dengan SeaTalk, sebuah platform komunikasi yang dikembangkan oleh perusahaan Sea. Platform ini dirancang khusus untuk mendukung kolaborasi perusahaan kecil hingga besar dan digunakan sebagai platform komunikasi utama di seluruh grup Sea. Setiap hari, *API Automation Test* dijalankan secara otomatis pada waktu-waktu tertentu menggunakan *CRON scheduler*. Setelah pengujian berhasil, sistem secara otomatis mengirimkan notifikasi ke grup SeaTalk yang telah dikonfigurasi sebelumnya.

Notifikasi tersebut mencakup informasi-informasi, seperti jumlah test case yang berhasil, jumlah test case yang gagal, dan URL menuju *Cucumber Report* yang dihasilkan oleh *framework* Karate. Gambar 3.3 menunjukkan contoh notifikasi SeaTalk yang dikirimkan secara automatis dari hasil *API Automation Test*.



Gambar 3.3. Contoh Notifikasi *API Automation* SeaTalk

Dengan integrasi ini, tim dapat memantau hasil pengujian secara *real-time* dan mengambil tindakan segera jika terdapat kegagalan.

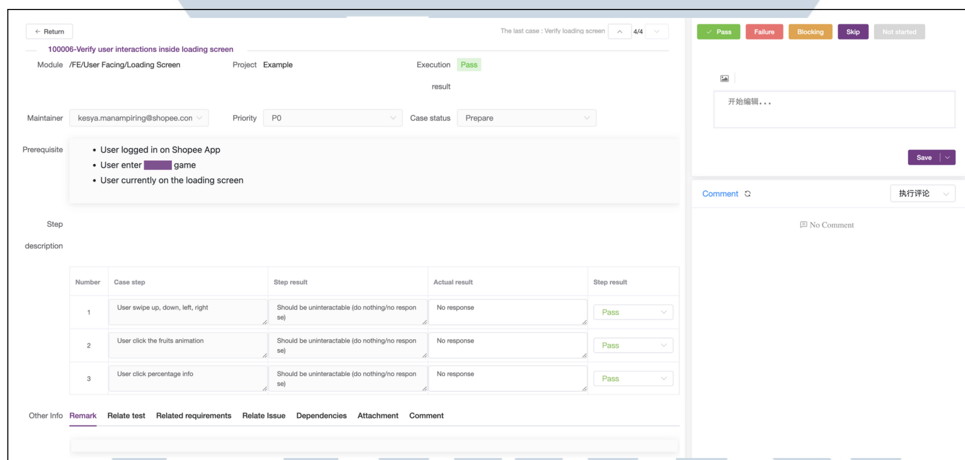
UNIVERSITAS MULTIMEDIA NUSANTARA

3.2.2 Manual Testing

Manual Testing merupakan proses pengujian perangkat lunak secara manual untuk memastikan kualitas aplikasi sesuai dengan kebutuhan dan spesifikasi. Di divisi *Games*, *manual testing* menjadi metode utama untuk pengujian karena banyak elemen visual yang tidak dapat diuji secara otomatis. Tahapan *manual testing* mencakup berbagai aktivitas yang dimulai dengan persiapan, di mana *Quality Assurance Lead* meninjau *timeline* dan perencanaan sumber daya. Biasanya,

Quality Assurance Lead mengikuti rapat *scope & testing timeline* bersama *anggota Quality Assurance*. Jika tidak memungkinkan, *Quality Assurance Lead* akan mengadakan rapat terpisah untuk meninjau *timeline testing* dan kebutuhan sumber daya sebelum memberikan konfirmasi kepada TPM bahwa *timeline* final sudah siap.

Setelah tahap persiapan selesai, pengembangan dimulai melalui *kick off development*. Tim *developer* memulai pengembangan dan melakukan *self test* untuk memastikan fungsi dasar berjalan dengan baik. *Quality Assurance* (QA) bertanggung jawab memastikan *test case* telah dibuat dan ditinjau sebelum *self test* dimulai. *Test case* dibuat menggunakan *Test Case Manager Tools* seperti yang ada pada Gambar 3.4 dan ditinjau melalui *peer review* sebelum melanjutkan ke tahap berikutnya. Selain itu, *Quality Assurance* (QA) dapat menulis skrip otomatisasi API untuk mengecek tingkat keberhasilan *Application Programming Interface* (API) (*API success rate*).



Gambar 3.4. Contoh tampilan *test case detail* pada *Test Case Manager Tools*

Pada tahap eksekusi *manual testing*, *Quality Assurance* (QA) melaksanakan *manual functional test* di lingkungan pengujian (*Test Environment*) berdasarkan *test case* yang telahdibuat. Jika hasilnya aman, pengujian dilanjutkan ke *manual regression test* di *UAT Environment* untuk memastikan tidak ada regresi pada fitur yang sudah berjalan. Selanjutnya, tim lokal seperti tim marketing melakukan *External UAT* Test untuk mengevaluasi aplikasi secara menyeluruh. Jika ditemukan *defect* di *Test Environment* atau *UAT Environment*, *defect* tersebut harus segera dilaporkan dan diperbaiki sebelum *deployment* ke *Live Environment*.

Tahap akhir adalah *deployment*, dimana *Quality Assurance* (QA) akan melakukan

sanity test di *Live Environment*. *Quality Assurance* (QA) memastikan tidak ada masalah signifikan dengan melakukan *sanity test* dan memantau sistem menggunakan Grafana, termasuk memantau *User Interface* (UI) dan *Application Programming Interface* (API) *error rate* agar tidak meningkat secara signifikan.

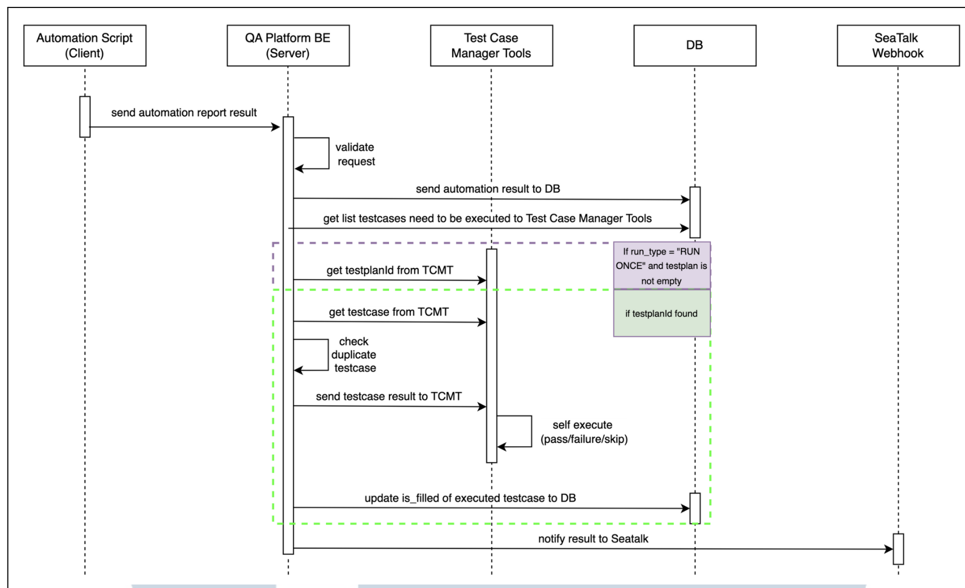
Sebagai bagian dari tim *Quality Assurance* (QA) di divisi *Games* 2, penulis telah berkontribusi dalam *manual testing* beberapa proyek bisnis di berbagai game, seperti Shopee Fruity, Shopee Capit, dan beberapa *Arcade Games*, termasuk Brickbreaker, Slap, dan Cooking. Pengujian dilakukan dengan perangkat nyata yang dipisahkan antara platform iOS dan Android untuk memastikan pengalaman pengguna yang optimal di berbagai perangkat.

3.2.3 Integrasi Application Programming Interface (API) Automation Test dengan Test Case Manager Tools

Sistem pengujian perangkat lunak harus terintegrasi secara menyeluruh untuk memberikan efisiensi dan efektivitas bagi tim *Quality Assurance* (QA) dalam melaksanakan tugasnya. Saat ini, hasil pengujian otomatisasi dan laporan di *Test Case Manager Tools* masih terpisah, sehingga *QA Engineer* perlu secara manual memasukkan hasil pengujian otomatis ke dalam *Test Case Manager Tools*.

Untuk menghemat waktu dan meningkatkan produktivitas, diperlukan integrasi antara hasil *API Automation Test* dan *Test Case Manager Tools*. Integrasi ini bertujuan agar hasil otomatisasi dapat langsung diperbarui di *Test Case Manager Tools*, sehingga proses menjadi lebih efisien dan minim kesalahan. Integrasi ini dirancang untuk dijalankan secara otomatis ketika *API Automation Test* dipicu secara manual (*triggered run once*) dan memiliki konfigurasi rencana pengujian (*test plan*) yang valid sesuai dengan yang ada di *Test Case Manager Tools*. Gambar 3.5 menjelaskan alur sistem dari integrasi ini.

UNIVERSITAS
MULTIMEDIA
NUSANTARA



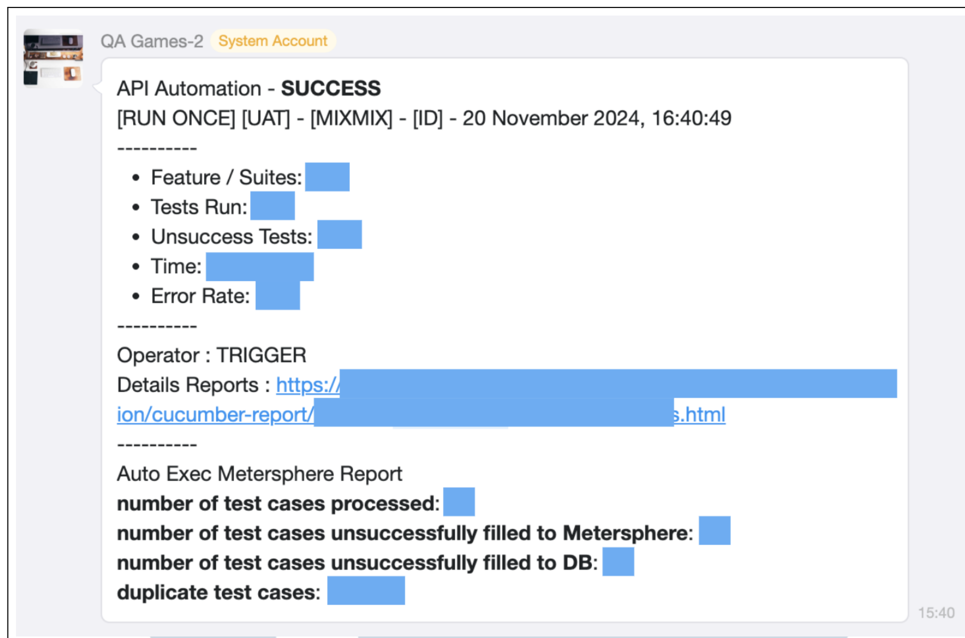
Gambar 3.5. Diagram Alur Sistem Integrasi API Automation Test dengan Test Case Manager Tools

Jika pengujian berhasil, status *test case* di *Test Case Manager Tools* akan berubah dari "*Not Started*" menjadi "*Pass*". Selain itu, eksekutor akan diberi label qa.automation untuk menunjukkan bahwa pengujian tersebut dijalankan oleh sistem otomatisasi seperti yang ada pada Gambar 3.6.

Gambar 3.6. Contoh tampilan berhasil *auto-execute* pada Test Case Manager Tools

Integrasi ini juga diintegrasikan dengan SeaTalk untuk mengirim notifikasi terkait informasi keberhasilan *auto-execute test plan* bersamaan dengan jalannya notifikasi *API Automation Test*. Namun notifikasi tambahan terkait integrasi dengan Test Case Manager Tools hanya muncul sekali dan tidak muncul di *daily running*. Gambar 3.7 adalah contoh dari notifikasi yang diterima di SeaTalk.

UNIVERSITAS MULTIMEDIA NUSANTARA



Gambar 3.7. Contoh Notifikasi Integrasi *Test Case Manager Tools* di SeaTalk

Dalam tugas integrasi ini, penulis mendapat kepercayaan untuk mengerjakan penuh projek ini. Namun, penulis juga dibantu terkait informasi sistem saat ini,*briefing* projek, dan revisi dari *Reporting Manager*, *Quality Assurance Lead* dari seluruh studio, dan *peer Quality Assurance*. Potongan Kode 3.2 adalah contoh potongan kode fungsi untuk melakukan perubahan status *test case* dengan mengubah status berdasarkan *test case number*.

```
async function updateTestCaseStatus(caseId, id, status) {
  try {
    const headers = await getHeader();
    const response = await axios.post(
      'https://example.test.case.manager.tools/case/edit',
      { caseId, id, status },
      { headers },
    );
    return response.data;
  } catch (error) {
    logger.error('Error updating test case status in Test Case
   Manager Tools:   , error);
    return {
      error: new Error('error updating test case status in test
   case manager tools ),
      message: 'error updating test case status in test   case
   manager tools ,
```

```
15        };
16    }
17 }
18
```

Listing 3.2: Contoh potongan kode Integrasi Test Case Manager Tools

3.2.4 Performance Testing

Performance Testing adalah pengujian yang dilakukan untuk mengevaluasi kinerja komponen-komponen sistem dalam kondisi beban tertentu. Selama pengujian, komponen sistem dipantau untuk memastikan stabilitas sistem yang diuji. Pengujian ini penting untuk memastikan aplikasi dapat menangani lalu lintas pengguna sesuai dengan proyeksi, terutama saat terjadi lonjakan trafik pada momen kampanye atau *event* tertentu. Di divisi *Games*, terdapat dua jenis pengujian kinerja utama, yaitu:

A. Load Testing

Pengujian ini dilakukan untuk mengevaluasi perilaku sistem di bawah beban tertentu, atau untuk menentukan titik batas di mana sistem mulai menunjukkan penurunan kinerja [3].

Contoh: Sistem harus mampu menangani 2500 RPS (*Request per Second*), berdasarkan proyeksi dari PM. Berikut langkah pengujian:

1. Tambahkan 500 VUS pada percobaan pertama selama durasi 30 detik hingga 1 menit dan amati hasilnya.
2. Tambahkan 1000 VUS pada percobaan kedua dengan durasi yang sama.
3. Tambahkan 1500 VUS pada percobaan ketiga.
4. Tambahkan 2500 VUS pada percobaan keempat.
5. Lakukan pengujian hingga ditemukan titik maksimum kemampuan server menangani permintaan.

B. Stress Testing

Pengujian ini dilakukan untuk mengevaluasi perilaku aplikasi di luar kondisi normal atau beban puncak. Tujuannya adalah untuk menguji fungsi aplikasi di bawah tekanan tinggi [3].

Contoh: *Landing page* suatu *game* akan diakses oleh sekitar 15.000 RPS selama kampanye berlangsung (±5 menit). Berikut langkah pengujian:

1. Tambahkan xxx VUS secara bertahap untuk durasi kampanye tertentu.
2. Amati apakah sistem dapat menangani permintaan sesuai dengan yang diharapkan.

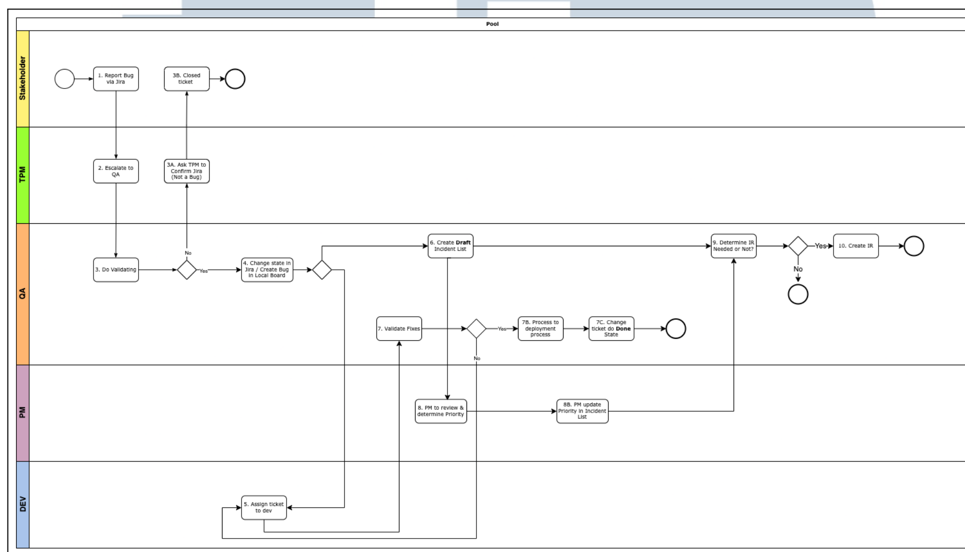
Sebelum melakukan pengujian kinerja, *Quality Assurance* (QA) perlu mempersiapkan beberapa hal, seperti diskusi dengan PM atau divisi terkait untuk mendapatkan data proyeksi trafik dan menentukan jenis pengujian (*Load Testing* dan/atau *Stress Testing*) yang sesuai dengan kebutuhan. Setelah itu, QA dapat melakukan pengujian menggunakan script yang telah dirancang. Setelah pengujian selesai dilakukan, QA bertanggung jawab untuk menganalisis hasil pengujian dan menyusun laporan yang mencakup:

1. **Spesifikasi *Resource***: Termasuk spesifikasi *database* (DB) dan Redis yang digunakan.
2. **Spesifikasi Pengujian Kinerja**: Detail mesin yang digunakan untuk pengujian.
3. **Detail Hasil Pengujian**: Meliputi tingkat keberhasilan API, *API error rate*, *total request*, analisis menggunakan Grafana, detail per VUS, dan data penggunaan aplikasi.

Pada penugasan *performance testing* ini, penulis berkolaborasi dengan tim *developer* dan *peer Quality Assurance* untuk membuat *script* untuk versi terbaru dari Shopee Fruity. Penulis juga menulis laporan hasil *testing* secara mandiri mengikuti *guideline* yang telah ditetapkan.

3.2.5 Defect & Bug Report

Dalam proses *Software Testing Life Cycle* (STLC), kemungkinan ditemukan *defect* di *lower environment* seperti *Test Environment* atau *UAT Environment* sangatlah tinggi. Bahkan setelah aplikasi dideploy ke *Live Environment*, masih ada kemungkinan munculnya *bug*. Proses pelaporan *defect* dan *bug* dapat dilakukan oleh *Quality Assurance* (QA), meskipun *bug* juga sering kali dilaporkan oleh tim lokal atau pengguna aplikasi.



Gambar 3.8. Alur *Defect & Bug Report*

Ketika *Quality Assurance* (QA) menemukan atau menerima laporan *defect* atau *bug*, langkah pertama adalah membuat *defect* atau *bug report* sesuai dengan alur yang telah ditetapkan seperti yang ada pada Gambar 3.8. Setiap *defect* atau *bug* yang dilaporkan harus dikategorikan berdasarkan *Severity* dan *Priority*:

1. **Severity**: Mengacu pada tingkat dampak *defect* terhadap fungsionalitas sistem. *Quality Assurance* (QA) bertanggung jawab untuk menentukan tingkat *severity* berdasarkan seberapa besar pengaruhnya terhadap aplikasi.

2. **Priority**: Menentukan urutan pengerjaan *defect*. *Defect* dengan prioritas tinggi akan diperbaiki lebih dahulu.

QA kemudian meneruskan *defect* tersebut ke developer untuk diperbaiki. Jika *defect* ditemukan di *lower environment*, *defect* dengan *severity* dan *priority* tinggi akan segera diperbaiki sebelum naik ke *environment* yang lebih tinggi, terutama

jika usaha perbaikannya tidak terlalu besar. Namun, untuk *defect* yang terjadi di *Live Environment* dengan *severity* atau *priority* rendah, perbaikannya dapat diintegrasikan ke versi rilis berikutnya atau dibuatkan *hotfix* jika diperlukan.

Selain itu, *Quality Assurance* (QA) harus memvalidasi keabsahan laporan *defect* dengan mencoba mereproduksi *bug* tersebut. Setelah *defect* diperbaiki, QA akan melakukan pengujian ulang untuk memastikan bahwa *defect* telah terselesaikan sepenuhnya.

Jika bug di *Live Environment* memengaruhi banyak pengguna atau menyebabkan kerugian, *Quality Assurance* (QA) akan membuat *Incident Report* yang berisi data lengkap tentang jumlah pengguna yang terdampak dan estimasi kerugian yang ditimbulkan.

Selama menjalani program magang, penulis menemukan beberapa *defect* dan *bug* baik di *lower environment* maupun di *Live Environment*. Penulis berperan aktif dalam proses pelaporan sesuai *guideline* dan alur kerja yang telah ditentukan oleh tim, serta memastikan *defect* dan *bug* terselesaikan dengan baik.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang diuraikan seperti pada Tabel 3.1.

Tabel 3.1. Tugas yang dilakukan setiap minggu selama magang

Minggu Ke-	Pekerjaan yang dilakukan
1	*Onboarding*, mempelajari tugas dan *knowledge* QA dengan membaca QA Handbook, *briefing* terkait kerja magang dengan reporting manager, *briefing* terkait *QA tools* bersama peer QA, *briefing* terkait JIRA dengan TPM, *set up environment*, *request access internal tools* dan melakukan eksplorasi terkait Karate Framework
Lanjut pada halaman berikutnya	

Tabel 3.1 Tugas yang dilakukan setiap minggu selama magang (lanjutan)

Minggu Ke-	**Pekerjaan yang dilakukan**
2	Melakukan eksplorasi terkait *QA tools*, mengikuti PRD briefing untuk versi terbaru Shopee Fruity, melakukan *smoke test* dan *sanity test*, melakukan eksplorasi terkait *logging* dan *monitoring tools*, dan mengerjakan *automation test* Shopee Capit
3	Melakukan *task breakdown* untuk Shopee Fruity, membuat *test case* untuk versi terbaru Shopee Fruity, mengikuti *firedrill* persiapan *campaign* 9.9 dan mengerjakan *automation test* Shopee Capit
4	Mengerjakan *automation test* Shopee Capit dan melakukan *refactor config* Shopee Capit
5	Melakukan *functional test* untuk versi terbaru Shopee Fruity, mengikuti PRD briefing untuk versi terbaru Shopee Fruity, dan mengerjakan *automation test* Shopee Capit
6	Melakukan *regression test* untuk versi terbaru Shopee Fruity dan mengerjakan *automation test* Shopee Capit
7	Mengikuti PRD dan ERD briefing untuk versi terbaru dari Shopee Fruity, mengerjakan *automation test* Shopee Capit, mengikuti *task breakdown* untuk versi terbaru Shopee Fruity, dan melakukan *smoke test* Arcade Game: Slap
8	Melakukan *sanity test* untuk Arcade Game: Slap, mengerjakan *automation test* Shopee Capit, dan memperbaiki *flaky error* di *automation test* Arcade Game: Slap
9	Memperbaiki flaky error di *automation test* Arcade Game: Slap dan membuat *test case* untuk versi terbaru dari Shopee Fruity
10	Melakukan *functional test* untuk versi terbaru Shopee Fruity dan memperbaiki *flaky error* di automation test Shopee Capit
11	Melakukan *functional test* untuk versi terbaru Shopee Fruity, melakukan *regression test* untuk versi terbaru Shopee Fruity, dan melakukan *mass testing* untuk versi terbaru Shopee Fruity

Lanjut pada halaman berikutnya

UNIVERSITAS MULTIMEDIA NUSANTARA

Tabel 3.1 Tugas yang dilakukan setiap minggu selama magang (lanjutan)

Minggu Ke-	Pekerjaan yang dilakukan
12	Melakukan *load test* dan *endurance test* untuk versi terbaru Shopee Fruity, menambah *test case* di *automation test* dengan API versi terbaru dari Shopee Fruity, dan melakukan *sanity test* versi terbaru dari Shopee Fruity
13	Memperbaiki *flaky error* di *automation test* Shopee Fruity, melakukan *smoke test* untuk versi terbaru Shopee Fruity di negara Colombia, Chille, Malaysia, dan Thailand, dan melakukan *update game admin* untuk *automation test* Shopee Capit ke versi terbaru
14	Mengerjakan integrasi *auto execution test case tools* dari *automation* yang berjalan
15	Melakukan *regression test* untuk versi *hotfix* Shopee Fruity dan melakukan *sanity test* untuk versi *hotfix* Shopee Fruity
16	Menambahkan *API Automation Test Coverage* untuk Shopee Fruity, melakukan *handle bug ticket*, dan melakukan eksplorasi terkait *improvement API Automation*
17	Melakukan eksplorasi terkait *improvement API Automation*, melakukan *handle bug ticket*, dan memperbaiki *flaky error* di *automation test* Shopee Fruity

3.4 Kendala dan Solusi yang Ditemukan

Selama praktik *internship* dilaksanakan, terdapat beberapa kendala yang dialami. Berikut adalah beberapa kendala yang dihadapi selama proses *internship* berlangsung.

1. Kesulitan dalam menyesuaikan versi *tech stack* atau *library* yang digunakan saat melakukan *set up environment*, serta kesulitan mencari informasi terkait tools yang dokumentasinya tersebar di beberapa halaman Confluence.

2. Adanya migrasi API yang mengharuskan pengecekan manual berulang, meskipun tidak terdapat perubahan yang signifikan.

3. QA harus memverifikasi ulang API lama yang telah di-*cover* oleh *automation*

UNIVERSITAS MULTIMEDIA NUSANTARA

test dan melakukan pembaruan status *pass/fail* secara manual pada *test case manager tools*.

Berdasarkan kendala serta masalah yang dialami selama proses kerja *internship*, beberapa solusi telah ditemukan untuk mengatasi kendala tersebut.

1. Menambahkan informasi lengkap mengenai *tech stack* dan *library* yang diperlukan di Confluence, serta memanfaatkan bot SIQAQ (bot LLM internal QA) untuk membantu menemukan halaman Confluence yang relevan secara cepat.

2. Meningkatkan cakupan *automation test* untuk mempermudah pengecekan perubahan API melalui pengujian otomatis.

3. Mengintegrasikan *automation test* dengan *test case manager tools*, sehingga QA tidak perlu melakukan pengujian ulang atau pembaruan status secara manual.

UNIVERSITAS MULTIMEDIA NUSANTARA