PENGUJIAN SECARA FUNGSIONAL, STRUKTURAL, DAN PERFORMA PADA KMS DESA DIGITAL INDONESIA

AHMAD QAULAN SADIDA

PROGRAM SARJANA ILMU KOMPUTER
SEKOLAH SAINS DATA, MATEMATIKA, DAN INFORMATIKA
INSTITUT PERTANIAN BOGOR
BOGOR
2026

PERNYATAAN MENGENAI SKRIPSI DAN SUMBER INFORMASI SERTA PELIMPAHAN HAK CIPTA

Dengan ini saya menyatakan bahwa skripsi dengan judul "Pengujian Secara Fungsional, Struktural, dan Performa pada KMS Desa Digital Indonesia" adalah karya saya dengan arahan dari dosen pembimbing dan belum diajukan dalam bentuk apa pun kepada perguruan tinggi mana pun. Sumber informasi yang berasal atau dikutip dari karya yang diterbitkan maupun tidak diterbitkan dari penulis lain telah disebutkan dalam teks dan dicantumkan dalam Daftar Pustaka di bagian akhir skripsi ini.

Dengan ini saya melimpahkan hak cipta dari karya tulis saya kepada Institut Pertanian Bogor.

Bogor, Juli 2026

Ahmad Qaulan Sadida
G6401221032

ABSTRAK

AHMAD QAULAN SADIDA. Pengujian Secara Fungsional, Struktural, dan Performa pada KMS Desa Digital Indonesia. Dibimbing oleh YANI NURHADRYANI dan HENDRA RAHMAWAN.

Pengujian merupakan tahapan penting dalam memastikan kualitas dan kesiapan sistem sebelum digunakan secara luas. Pada penelitian ini, pengujian diterapkan pada *Knowledge Management System* Desa Digital Indonesia (KMS DDI) dengan tiga pendekatan, yaitu *black-box*, *white-box*, dan *performance*, mengikuti *Software Testing Life Cycle* (STLC). *Black-box testing* dilakukan menggunakan Katalon Studio melalui kombinasi manual dan *automation testing* dengan teknik *error guessing*. *White-box testing* menggunakan teknik *basis path testing* dengan *framework* Jest. *Performance testing* dilakukan menggunakan Apache JMeter dengan skenario *load testing* 1–500 *virtual users*. Hasil *black-box* dari 312 skenario pengujian menghasilkan 51 *error* dengan *success rate* 84%, didominasi *functional error*. Pada *white-box testing*, ditemukan lima *error* dengan *success rate* 91%. Pada *performance testing*, sebagian besar *endpoint* menunjukkan level Apdex *Good* dan *Excellent*, namun masih terdapat temuan seperti *error* dengan pesan *"connection reset"*, dan *request* lambat pada beberapa *endpoint* dan *request* awal. Seluruh *error* yang ditemukan kemudian dianalisis untuk diberikan rekomendasi perbaikan agar dapat meningkatkan kualitas sistem.

Kata kunci: KMS desa digital, perbaikan sistem, pengujian fungsional, pengujian performa, pengujian struktural.

ABSTRACT

AHMAD QAULAN SADIDA. Functional, Stuctural, and Performance Testing of Digital Village Knowledge Management System (KMS DDI). Supervised by YANI NURHADRYANI and HENDRA RAHMAWAN.

Testing is a critical stage in ensuring the quality and readiness of a system prior to its widespread deployment. In this study, testing was applied to the Knowledge Management System of Desa Digital Indonesia (KMS DDI) using three approaches—black-box, white-box, and performance testing, following the Software Testing Life Cycle (STLC). Black-box testing was conducted using Katalon Studio through a combination of manual and automated testing with the error guessing technique. White-box testing employed the basis path testing technique using the Jest framework. Performance testing was carried out using Apache JMeter with a load testing scenario ranging from 1 to 500 virtual users. The black-box testing, comprising 312 test scenarios, identified 51 errors with an 84% success rate, most of which were functional errors. The white-box testing identified five errors with a 91% success rate. In the performance testing, most endpoints achieved Good to Excellent Apdex levels, however, several issues were still observed, including errors with the message "connection reset" and slow response times on certain endpoints as well as during initial requests. All identified errors were subsequently analyzed to provide improvement recommendations aimed at enhancing the overall quality of the system.

Keywords: Digital village KMS, functional testing, performance testing, system improvement, structural testing.

PENGUJIAN SECARA FUNGSIONAL, STRUKTURAL, DAN PERFORMA PADA KMS DESA DIGITAL INDONESIA

AHMAD QAULAN SADIDA

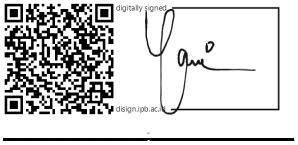
Skripsi
sebagai salah satu syarat untuk memperoleh gelar
Sarjana pada
Program Studi Ilmu Komputer

PROGRAM SARJANA ILMU KOMPUTER
SEKOLAH SAINS DATA, MATEMATIKA, DAN INFORMATIKA
INSTITUT PERTANIAN BOGOR
BOGOR
2026

Penguji pada Ujian Skripsi:
Firman Ardiansyah, S.Kom., M.Si.

Judul Skripsi : Pengujian Secara Fungsional, Struktural, dan Performa pada KMS Desa Digital Indonesia
Nama : Ahmad Qaulan Sadida
NIM : G6401221032

Disetujui oleh



Pembimbing 1:
Dr. Yani Nurhadryani, S.Si., M.T.

Pembimbing 2:
Dr. Hendra Rahmawan, S.Kom., M.T.

Diketahui oleh

Ketua Program Sarjana Ilmu Komputer:
Dr. Sony Hartono Wijaya, S.Kom., M.Kom.
NIP 198108092008121002

Tanggal Ujian:
1 Juli 2026

Tanggal Lulus:

PRAKATA

Puji dan syukur penulis panjatkan kepada Allah *subhanaahu wa ta'ala* atas segala karunia-Nya sehingga karya ilmiah ini berhasil diselesaikan. Penelitian ini telah dilaksanakan sejak bulan Desember 2025 sampai bulan Juli 2026 dengan judul "Pengujian Secara Fungsional, Struktural, dan Performa pada KMS Desa Digital Indonesia".

Penelitian ini dapat diselesaikan dengan baik tentunya dengan bantuan dari banyak pihak. Oleh karena itu, penulis ingin menyampaikan banyak terima kasih kepada:

a Kedua orang tua dan keluarga lainnya yang senantiasa mendukung, memberikan kasih sayang, doa, dan motivasi kepada penulis selama menempuh masa perkuliahan.
b Ibu Dr. Yani Nurhadryani, S.Si., M.T. dan Bapak Dr. Hendra Rahmawan, S.Kom., M.T. selaku pembimbing yang telah memberikan bimbingan, arahan, dan mendorong penulis agar segera menyelesaikan penelitian ini.
c Bapak Firman Ardiansyah, S.Kom., M.Si. selaku dosen penguji yang telah memberikan saran sehingga penelitian ini menjadi lebih baik.
d Pemilik NIM F2401221027 yang telah memberikan dukungan, semangat, serta selalu siap menjadi tempat berkeluh-kesah selama masa perkuliahan dan penyelesaian penelitian ini.
e Tim Peneliti Desa Digital, yaitu Wisnu dan Yuda, yang bersama-sama menjadi teman diskusi dan mendukung penulis selama masa penelitian.
f Ahmad, Fari, Aji, Pandu, Ridho, Vergi, dan Mocha yang sudah menjadi rekan seperjuangan penulis selama masa studi di Ilmu Komputer.
g Seluruh teman Ilkomerz 59 yang telah memberikan semangat dan berjuang bersama-sama selama masa perkuliahan.

Semoga karya ilmiah ini bermanfaat bagi pihak yang membutuhkan dan bagi kemajuan ilmu pengetahuan.

Bogor, Juli 2026

Ahmad Qaulan Sadida

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

DAFTAR LAMPIRAN

I PENDAHULUAN

1.1 Latar Belakang

Data BPS (2023) menunjukkan bahwa jumlah desa di Indonesia mencapai 83.971. Angka ini menunjukkan besarnya potensi yang dapat dikembangkan, salah satunya melalui penerapan program desa digital. Desa digital merupakan program yang memanfaatkan teknologi informasi dalam mengembangkan potensi desa, mempermudah akses layanan, serta meningkatkan kualitas pelayanan publik melalui sistem berbasis digital (Alvaro dan Octavia 2019) dalam (Nuryantika 2023). Pemanfaatan teknologi ini juga penting untuk memastikan masyarakat desa tetap terhubung dengan berbagai informasi dan komunitas melalui internet, sehingga mereka dapat menggunakan teknologi tersebut untuk menyelesaikan permasalahan terkait inovasi digital dan meningkatkan kualitas hidup mereka (Nurhadryani *et al*. 2022). Pengelolaan desa digital menghadapi tantangan dalam menyediakan dan mendistribusikan informasi secara efektif agar dapat diakses oleh seluruh pihak, mulai dari pemerintah desa hingga masyarakat. Tantangan ini dapat diatasi dengan memanfaatkan *knowledge management system* (KMS) sebagai *platform* yang mampu mengelola pengetahuan secara terstruktur, mudah diakses, dan berkelanjutan (Dalkir 2023). Sebagai bentuk implementasi KMS dalam konteks desa digital tersebut, dikembangkanlah KMS Desa Digital Indonesia (KMS DDI) sejak tahun 2023 (Nurhadryani *et al*. 2022).

KMS DDI sebagai sistem *web-mobile* yang menyajikan informasi penerapan inovasi digital di desa-desa Indonesia berperan penting dalam mendukung konsep, keberadaan, dan pengembangan ekosistem desa digital di Indonesia (Awalia 2024). Perkembangan KMS DDI sendiri diawali dengan penelitian Tasnim (2023) yang merancang *User Experience* (UX) secara menyeluruh berbasis *Lean UX*, kemudian dilanjutkan oleh Nuryantika (2023) yang membangun modul *front-end* berbasis *mobile web*. Kedua penelitian ini menghasilkan sebuah *platform* awal yang berfungsi sebagai sistem manajemen dokumen (KMS Level 1). Penelitian lainnya yang dilakukan Hutauruk (2025) telah melakukan pengujian *black-box* terhadap 134 skenario yang menunjukkan 127 (94,8%) berhasil dan tujuh (5,2%) gagal. Seiring dengan pengembangan lanjutan, KMS DDI mengalami perubahan arsitektur melalui proses *re-engineering*, khususnya pada migrasi *database* dari Firestore ke MongoDB. Perubahan ini berimplikasi pada penyesuaian struktur data, alur logika program, serta mekanisme akses dan pengelolaan data, sehingga diperlukan pengujian ulang yang lebih mendalam untuk memastikan sistem tetap berjalan secara benar, efisien, dan stabil setelah proses migrasi dilakukan.

Dalam pengembangan perangkat lunak, pengujian memiliki peran penting dalam menghasilkan sistem yang berkualitas. Tahapan ini diperlukan karena *developer* tidak luput dari kesalahan, sebagian kesalahan tidak memiliki dampak besar, namun ada juga yang bisa menyebabkan kerugian atau membahayakan pengguna (Uddin dan Anand 2019). Setiap sistem yang dikembangkan harus diuji terlebih dahulu agar kualitasnya terjamin sebelum digunakan oleh masyarakat luas. Namun dalam praktik industri, tahap pengujian sering kali dilewatkan karena dianggap menambah waktu dan biaya pengembangan, sehingga berbagai kesalahan sistem yang sebenarnya bisa dicegah tetap terjadi (Uddin dan Anand 2019). Teknik pengujian perangkat lunak terdiri dari beberapa pendekatan, antara lain pengujian

fungsional yang terdiri dari *black-box* yang menguji input dan output sistem tanpa melihat struktur internal program, dan *white-box* yang memeriksa logika dan struktur internal program (Jampani *et al.* 2016). Selain pengujian tersebut, terdapat jenis pengujian non-fungsional seperti pengujian performa yang menilai bagaimana sistem merespon ketika diberikan beban kerja tertentu (Ali *et al*. 2022).

Melihat perkembangan KMS DDI serta keterbatasan pengujian yang dilakukan pada penelitian sebelumnya, diperlukan evaluasi yang lebih menyeluruh terhadap seluruh aspek sistem. Pengujian fungsional melalui *black-box testing* penting untuk memastikan setiap fitur berjalan sesuai kebutuhan pengguna. Di sisi lain, *white-box testing* dibutuhkan untuk menilai kualitas logika internal dan struktur kode guna memastikan sistem bekerja dengan benar dari dalam. Selain itu, pengujian performa perlu diterapkan untuk mengetahui kemampuan sistem dalam menangani beban pengguna yang semakin besar seiring berkembangnya ekosistem desa digital. Integrasi ketiga jenis pengujian ini akan memberikan gambaran yang komprehensif mengenai kesiapan KMS DDI sebelum digunakan secara luas oleh pengguna.

1.2 Rumusan Masalah

Sebelum sistem ini digunakan secara luas oleh berbagai *stakeholder* dan masyarakat, diperlukan evaluasi yang sistematis untuk memastikan seluruh aspek siap digunakan oleh masyarakat luas. Berdasarkan permasalahan tersebut, rumusan masalah dari penelitian ini adalah:

a Bagaimana melakukan pengujian fungsional pada fitur-fitur KMS DDI untuk memastikan seluruh fungsi berjalan sesuai kebutuhan pengguna?
b Bagaimana menerapkan pengujian struktural yang mampu mengevaluasi alur logika, struktur internal, dan kualitas kode agar sistem berfungsi sesuai implementasi yang diharapkan?
c Bagaimana mengukur performa KMS DDI dalam menangani variasi jumlah pengguna, beban akses tinggi, dan potensi lonjakan trafik?
d Bagaimana menyusun dokumentasi hasil pengujian dan merumuskan rekomendasi perbaikan *bug* secara sistematis agar sistem dapat beroperasi dengan stabil dan siap digunakan oleh publik serta *stakeholder*?

1.3 Tujuan

Tujuan dari penelitian ini adalah:

a Melakukan pengujian secara fungsional terhadap fitur-fitur KMS DDI untuk memastikan seluruh fungsi berjalan sesuai kebutuhan pengguna.
b Menerapkan pengujian secara struktural guna mengevaluasi struktur internal, alur logika, serta kualitas kode program sehingga dapat dipastikan sistem berfungsi sesuai dengan alur yang ditentukan.
c Melakukan pengujian performa untuk menilai kemampuan sistem dalam menangani variasi jumlah pengguna, beban akses yang tinggi, serta potensi lonjakan trafik.
d Menyusun dokumentasi hasil pengujian dan merumuskan rekomendasi perbaikan *bug* yang ditemukan, sehingga sistem dapat beroperasi secara stabil dan siap digunakan oleh publik serta para *stakeholder*.

1.4 Manfaat

Penelitian ini bermanfaat untuk mengidentifikasi serta memastikan bahwa KMS DDI bebas dari kesalahan sistem dan memiliki performa yang optimal sebelum diterapkan dalam lingkungan produksi. Selain itu, penelitian ini diharapkan dapat memberikan keyakinan kepada para *stakeholder* bahwa KMS DDI telah melalui proses pengujian yang komprehensif dan siap berfungsi secara optimal dalam mendukung kebutuhan operasional mereka.

1.5 Ruang Lingkup

Ruang lingkup dari penelitian yang dilakukan meliputi:

a Jenis pengujian yang dilakukan meliputi *Black Box Testing, White Box Testing,* dan *Performance Testing*

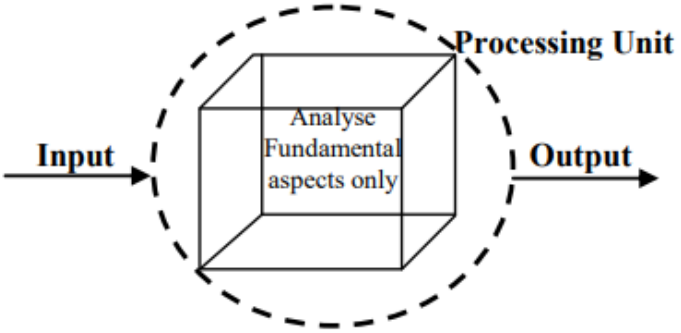
b Pengujian dilakukan pada *environment staging* dan berfokus pada perbaikan *bug* sebelum sistem siap dipublikasikan.

c Pengujian hanya mencakup fitur yang sudah ada, sementara fitur lainnya yang sedang dikembangkan seperti modul *chatbot* tidak termasuk dalam lingkup pengujian.

II TINJAUAN PUSTAKA

2.1 Pengujian *Black-Box*

Pengujian *black-box* juga dikenal sebagai pengujian fungsional, yaitu metode pengujian di mana penguji tidak perlu mengetahui terkait struktur internal dan logika sistem. Pengujian ini berfokus pada spesifikasi kebutuhan dan tidak mengharuskan penguji memahami atau menganalisis kode program. Pada tahap ini, penguji perlu mengumpulkan kebutuhan pengguna dan menyusun skenario pengujian berdasarkan kebutuhan tersebut (Jampani *et al.* 2016).



Gambar 1 Ilustrasi pengujian *black-box* (Ehmer dan Khan 2012)

Dalam pengujian *black-box*, terdapat beberapa teknik pengembangan *test case* yang dikemukakan oleh Bhagat (2016), antara lain:

a *Equivalence Partitioning*, teknik ini membagi domain input ke dalam beberapa kelas data berdasarkan *equivalence class*, yaitu sekelompok kondisi input yang dianggap mewakili keadaan valid atau tidak valid secara setara.

b *Boundary Value Analysis* (BVA), teknik ini didasarkan pada asumsi bahwa nilai input di sekitar batas rentang (*boundary*) lebih berpotensi menimbulkan kesalahan dibandingkan nilai-nilai di tengah domain input. Oleh karena itu, BVA digunakan untuk menemukan kesalahan di sekitar nilai minimum, maksimum, serta nilai yang berdekatan dengan kedua batas tersebut.

c *Cause Effect Graphing Technique*, teknik ini memetakan hubungan logis antara kondisi input (*cause*) dan hasil atau respons sistem yang diharapkan (*effect*) dalam bentuk grafik terarah. Hubungan antar-*cause* dan *effect* digambarkan menggunakan operator logika seperti AND, OR, dan NOT, sehingga kombinasi kondisi yang memengaruhi suatu hasil dapat teridentifikasi secara sistematis.

d *Decision Table*, teknik ini digunakan untuk menguji sistem yang memiliki beberapa kondisi logis yang saling berkaitan dan perlu diuji dalam berbagai kombinasi. Penerapannya dilakukan dengan mengidentifikasi kondisi (*condition*) dan aksi (*action*) berdasarkan spesifikasi sistem, kemudian menyusunnya ke dalam tabel keputusan agar seluruh kemungkinan kombinasi dapat diuji secara sistematis.

e *State Transition*, teknik ini berfokus pada perubahan kondisi (*state*) suatu sistem sepanjang alur proses tertentu. Melalui teknik ini, dapat diperiksa apakah sistem berpindah antarkondisi hanya melalui urutan aksi (*event*) yang

telah ditentukan, sehingga tidak terjadi lompatan proses atau kondisi yang seharusnya tidak mungkin terjadi.

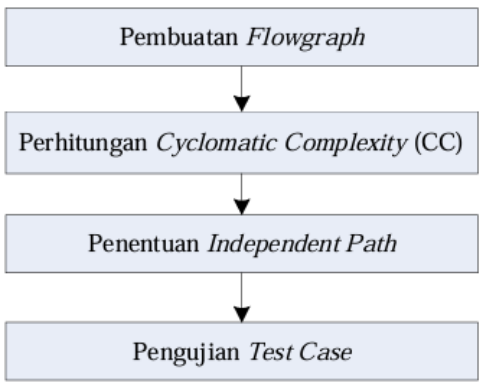
f *Error Guessing*, teknik ini menyusun *test case* berdasarkan perkiraan kesalahan yang mungkin muncul pada sistem, dengan mengandalkan pengalaman dan intuisi penguji terhadap sistem yang diuji yang sering disebut pendekatan *what-if*. Teknik ini efektif untuk menemukan *bug* tersembunyi, terutama melalui pengujian menggunakan input tidak valid untuk memastikan sistem menanganinya dengan baik.

2.2 Pengujian *White-Box*

Pengujian *white-box* dikenal juga sebagai pengujian struktural, yang berfokus melakukan pengujian dengan menganalisis struktur internal dan alur logika program (Solissa *et al*. 2023). Metode ini memungkinkan cakupan pengujian lebih maksimal karena jalur-jalur logika dapat dipetakan secara detail. Namun, tidak semua jalur dapat diuji karena kompleksitas kode, sehingga masih ada kemungkinan *bug* yang belum ditemukan (Ehmer dan Khan 2012).

Pengujian *white-box* memiliki beberapa teknik antara lain, *control flow testing* yang menggunakan alur kontrol program untuk memastikan semua bagian logika diuji, seperti percabangan (*if-else*), pengulangan (*looping*), dan kondisi bersyarat lainnya. Selanjutnya *data flow testing* yang memeriksa program secara diagramatis untuk melihat variabel dideklarasikan, digunakan, dan berubah dalam sebuah sistem, kemudian *loop testing* yang digunakan untuk memastikan struktur perulangan seperti *for, while,* dan *do-while* berfungsi dengan benar (Jampani *et al*. 2016).

Teknik lainnya adalah *basis path testing*, yang berfokus pada pengujian setiap eksekusi *independent path* dalam suatu modul minimal satu kali, untuk memastikan keseluruhan logika program teruji secara menyeluruh (Solissa *et al*. 2023). Pendekatan ini dilakukan dengan menyiapkan serangkaian langkah independen pada *control flow graph* untuk menganalisis kode program dalam menghitung nilai kompleksitasnya melalui *cyclomatic complexity* yang menentukan jumlah *independent path* pada struktur kontrol program (Solissa *et al*. 2023). *Basis path testing* memiliki beberapa tahapan yang dapat dilihat pada Gambar 2.



Gambar 2 Tahapan *basis path testing*

Tahapan diawali dari pembuatan *flowgraph* yang merupakan representasi visual dari alur logika program yang dapat dibangun berdasarkan kode program

atau *flowchart* sistem. Notasi yang digunakan pada *flowgraph* adalah lingkaran sebagai *node* yang merepresentasikan pernyataan prosedural dan anak panah sebagai *edge* yang menunjukkan arah alur logika. Tahap selanjutnya adalah perhitungan *cyclomatic complexity* (CC) yang digunakan untuk mengukur tingkat kompleksitas program secara kuantitatif. Nilai CC digunakan untuk menentukan jumlah *independent path* yang harus diuji minimal satu kali (Solissa *et al*. 2023). Nilai *cyclomatic complexity* dapat dihitung dengan persamaan (1) dan (2).

$$V(G) = E - N + 2 \quad (1)$$
$$V(G) = P + 1 \quad (2)$$

dengan:
V(G) = Cyclomatic Complexity
E = jumlah *edge* pada *flowgraph*
N = jumlah *node* pada *flowgraph*
P = jumlah *predicate node* pada *flowgraph*

Tahapan berikutnya adalah menentukan *independent path* yang merupakan jalur penghubung *node* awal dan *node* akhir pada program. Tahapan terakhir adalah menyusun *test case* berdasarkan jalur-jalur yang sudah diidentifikasi. Jumlah *independent path* yang sudah dihitung menentukan jumlah *test case* minimal yang harus dibuat. Setiap *test case* dirancang agar bisa menelusuri jalur tertentu dalam logika program, sehingga setiap alur bisa diuji dan divalidasi dengan baik (Abdillah 2025).

2.3 Pengujian Performa

Pengujian performa merupakan salah satu bentuk pengujian non-fungsional yang bertujuan mengevaluasi bagaimana sistem berperilaku ketika diberikan beban kerja tertentu (Ali *et al.* 2022). Pengujian ini menilai keandalan (*reliability*), skalabilitas (*scalability*), dan responsivitas (*responsiveness*) dari *Application Under Test* (AUT) di bawah berbagai kondisi beban. Pengujian performa memiliki tujuan bukan untuk menemukan *bug* atas kesalahan pada sistem, melainkan menilai bagaimana sistem merespons dan mempertahankan kestabilannya saat digunakan (Costa *et al.* 2020).

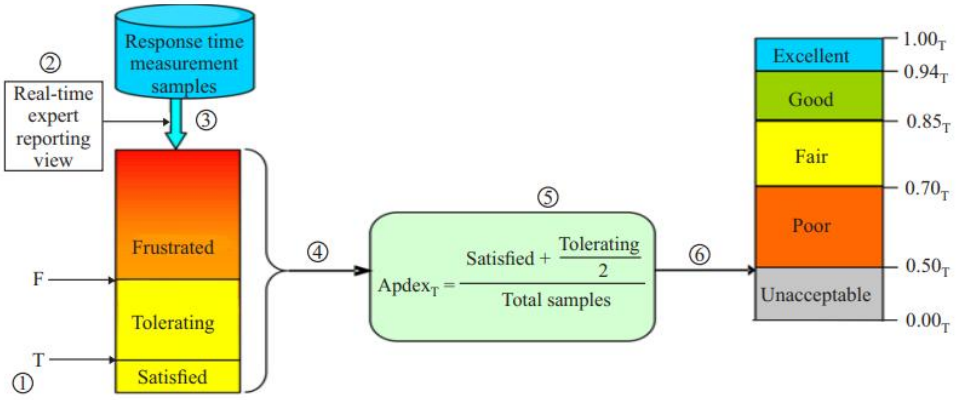
Menurut Microsoft Performance Guide dalam Ali *et al* (2022), terdapat enam kategori pengujian performa yang umum diterapkan untuk menilai kemampuan *Application Under Test* (AUT) dalam menghadapi berbagai kondisi beban, yaitu sebagai berikut.

a *Load testing*, menguji respons sistem terhadap beban kerja dalam kondisi penggunaan normal.
b *Stress testing*, menguji ketahanan sistem saat beban melebihi kapasitas normal untuk menemukan titik kegagalannya.
c *Spike testing*, menguji respons sistem terhadap lonjakan beban yang terjadi secara mendadak dan singkat.
d *Endurance testing,* menguji stabilitas sistem dalam penggunaan jangka panjang untuk mendeteksi masalah seperti *memory leak*.
e *Volume testing,* menguji kemampuan sistem dalam mengolah data berskala besar.
f *Scalability testing,* menguji kemampuan sistem beradaptasi terhadap peningkatan beban maupun perubahan sumber daya.

Salah satu metrik yang digunakan dalam penilaian performa berdasarkan variasi *response time* adalah *Application Performance Index* (Apdex). Apdex merupakan ukuran numerik yang digunakan dalam menilai tingkat kepuasan pengguna (*user satisfaction*) terhadap performa suatu sistem. Metode ini mengubah berbagai pengukuran performa menjadi nilai tunggal pada skala nol hingga satu, di mana nol menunjukkan tidak ada pengguna yang merasa puas, sedangkan satu berarti seluruh pengguna merasa puas (Sevcik 2005). Penilaian Apdex didasarkan pada tiga kategori waktu respons, yaitu:

a *Satisfied*, jika waktu respons pengguna ≤ T detik.
b *Tolerating,* jika waktu respons berada di T < waktu respons ≤ F detik
c *Frustated,* jika waktu respons pengguna > F detik.

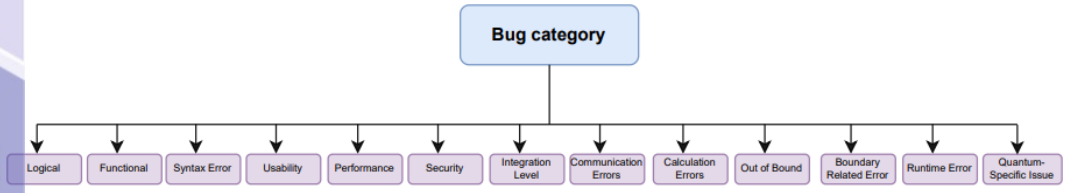
Skor Apdex dihitung dengan menjumlahkan jumlah *satisfied request* dan setengah dari *tolerating request*, kemudian dibagi total permintaan, sebagaimana ditunjukkan pada Gambar 3 (Samudrala 2022). Klasifikasi request didasarkan pada waktu respons terhadap nilai ambang T (*threshold*) yang masih dapat diterima pengguna, serta nilai F = 4T sebagai batas atas sebelum pengguna dianggap frustrasi (Uhl dan Rompf 2015). Skor yang dihasilkan kemudian dikategorikan ke dalam lima tingkatan kualitas performa, mulai dari *Excellent* hingga *Unacceptable*, seperti ditunjukkan pada Gambar 3.



Gambar 3 Apdex *Process Overview* (Uhl dan Rompf 2015)

2.4 Dokumentasi *Bug*

Dalam proses pengembangan perangkat lunak, *bug* adalah kesalahan pada perangkat lunak yang berpotensi menyebabkan kegagalan fungsi sistem (Dewi *et al*. 2024). Pada umumnya *bug* disebabkan oleh *wrong code* yaitu logika program yang salah atau fungsi yang tidak berjalan sesuai kebutuhan, *missing code* yaitu fitur yang seharusnya ada namun belum dikembangkan oleh pengembang, dan *extra code* yaitu fitur yang dikembangkan meskipun tidak tercantum dalam kebutuhan sistem (Rana *et al.* 2017). Pada penelitian ini, *error type* atau klasifikasi *bug* yang digunakan terdapat pada penelitian Yousuf dan Sofi (2025) yang mengelompokkan *bug* ke dalam kategori seperti pada Gambar 4.



Gambar 4 *Bug category* sebagai *error type* (Yousuf dan Sofi 2025)

Setelah diklasifikasikan berdasarkan jenis kesalahan (*error type*), pada penelitian ini *bug* juga dikategorikan berdasarkan tingkat keparahannya (*severity level*). Klasifikasi ini digunakan untuk menentukan prioritas perbaikan *bug* yang ditemukan selama proses pengujian. IEEE (2009) mengelompokkan tingkat keparahan *bug* ke dalam lima kategori, yaitu:

a *Critical* merupakan *bug* yang dapat menyebabkan kegagalan sistem pada fungsi utama sehingga sistem tidak dapat beroperasi dengan normal.
b *Major* merupakan *bug* yang menyebabkan kegagalan fungsi penting, tetapi masih terdapat alternatif untuk menyelesaikan alur yang diinginkan.
c *Average* merupakan *bug* yang tidak menyebabkan kegagalan sistem, tetapi menghasilkan *output* yang tidak akurat, kurang lengkap, atau mengurangi *usability* sistem.
d *Minor* merupakan *bug* yang memiliki dampak kecil dan tidak mengganggu fungsi utama sistem.
e *Exception* merupakan *bug* yang berkaitan dengan aspek estetika, ketidaksesuaian standar, atau permintaan peningkatan fitur (*enhancement*).

Pada penelitian ini, tingkat severity disederhanakan menjadi empat kategori, yaitu *critical*, *high*, *medium*, dan *low*. Kategori *high* mengacu pada *major*, *medium* mengacu pada *average*, sedangkan *low* mencakup *minor* dan *exception*.

2.5 Kasus Uji (*Test Case*)

Test case merupakan dokumen pada pengujian perangkat lunak yang berisi sekumpulan kondisi untuk menguji suatu fitur atau fungsi tertentu. Inti dari sebuah pengujian adalah menentukan kumpulan *test case* yang tepat untuk fitur yang diuji. Semakin tepat dan lengkap *test case* dibuat, semakin baik kualitas proses pengujiannya (Uddin dan Anand 2019). *Test case* yang lengkap biasanya memuat ID *test case,* tujuan pengujian, prasyarat yang harus dipenuhi, input yang akan diuji, output yang diharapkan, kondisi setelah pengujian (*postcondition*), dan riwayat eksekusi yang digunakan untuk menentukan keberhasilan dan kegagalan pengujian (Jorgensen 2013).

Pembuatan *test case* berarti memastikan prasyarat sudah terpenuhi, memberikan *input*, mengamati *output*, dan membandingkan hasil aktual dengan hasil yang diharapkan. Setelah itu, diperiksa juga apakah kondisi akhir yang diharapkan sudah tercapai. Peran penting *test case* dalam menjamin kualitas sistem, membuatnya dianggap sama berharga dengan kode program. Oleh karena itu, *test case* perlu dibuat dengan benar, ditinjau secara berkala, digunakan dalam pengujian, serta dikelola dan disimpan dengan baik (Jorgensen 2013). Bahasa yang digunakan dalam penulisan *test case* adalah *gherkin language*, yaitu format sederhana yang mudah dibaca oleh semua *stakeholder*. *Gherkin language* menggunakan kata kunci

seperti "*Given*", "*When*", dan "*Then*" untuk menjelaskan kondisi awal, tindakan yang dilakukan, dan hasil yang diharapkan. Selain itu, *gherkin language* juga memiliki struktur dasar seperti fitur (*feature*), *scenario*, dan langkah-langkah (*step definitions*) yang membantu menuliskan alur pengujian secara jelas dan teratur (Pahlevi 2024).

```
Feature: Search for products containing the keyword "Kitchen"

  Scenario: User searches for products with the keyword "Kitchen"
    Given the user is on the portal homepage
    When the user clicks on the search field
    And the user types "Kitchen" into the search field
    Then products containing the name "Kitchen" should appear
```

Gambar 5 Contoh penulisan *gherkin language* (Pahlevi 2024)

2.6 Pengembangan KMS Desa Digital Indonesia

Knowledge Management System (KMS) merupakan sistem yang dikembangkan untuk mengelola, menyimpan, dan mendistribusikan pengetahuan dalam suatu organisasi, baik berupa dokumen, basis data, maupun pengalaman individu (Semarandhanu dan Yulhendri 2024). Menurut Laudon dan Laudon (2022), KMS dapat dikategorikan menjadi tiga jenis, yaitu *Enterprise Knowledge Management System*, *Knowledge Work Management System*, dan *Intelligent Knowledge Management System*. Inovasi desa digital yang diterapkan melalui KMS termasuk ke dalam kategori *Enterprise Knowledge Management System*, karena dirancang untuk mendukung pengelolaan pengetahuan secara menyeluruh di tingkat desa (Ariyadi 2025).

Pengembangan KMS DDI merupakan rangkaian penelitian berkesinambungan yang dimulai dari perancangan konseptual hingga implementasi teknis, guna mendukung percepatan pembangunan desa digital dan desa cerdas di Indonesia (Nurhadryani *et al.* 2022). Penelitian diawali oleh Tasnim (2023) yang merancang pengalaman pengguna dengan pendekatan *Lean UX*, menghasilkan prototipe *medium-fidelity* sebagai acuan pengembangan selanjutnya. Prototipe tersebut kemudian dikembangkan lebih lanjut oleh Nuryantika (2023) menjadi modul *front-end mobile web* dengan fitur dasar seperti profil desa dan inovator, meski masih menggunakan data *dummy*, sebelum akhirnya diintegrasikan dengan *backend* Firebase oleh Ariyadi (2025) untuk mendukung penyimpanan dan autentikasi data yang lebih stabil.

Pengembangan selanjutnya mencakup penambahan model rekomendasi inovasi berbasis *content-based filtering* oleh Ferdiansah (2025), penerapan *digital nudge* untuk meningkatkan partisipasi desa oleh Hutauruk (2025), serta pengembangan *dashboard monitoring* bagi admin dan perangkat desa oleh Yusrina (2025) dan *dashboard* interaktif bagi inovator dan kementerian oleh Nurmaulydia (2025). Rangkaian penelitian tersebut secara konsisten memperkuat fungsionalitas dan kualitas informasi pada KMS DDI, sehingga sistem ini semakin mampu mendukung penyebaran inovasi digital secara efektif dan berkelanjutan di desa-desa Indonesia.

III METODE

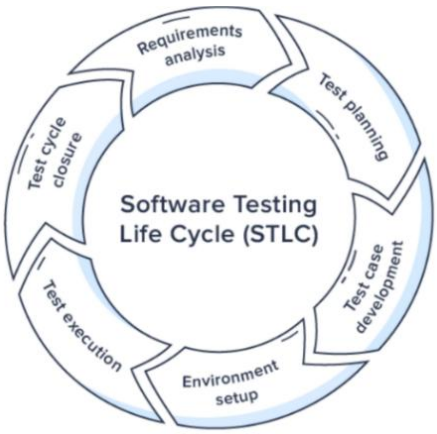
3.1 Objek Penelitian

Pada penelitian ini objek yang akan diuji adalah KMS DDI yang sudah dikembangkan sejak tahun 2023. KMS DDI berfungsi sebagai pusat informasi untuk memantau inovasi desa dan perkembangan inovasi yang dikelola oleh inovator. Saat ini sistem KMS sudah memiliki 228 inovator, 203 desa, dan 210 inovasi. KMS DDI memiliki empat jenis pengguna, yaitu admin, inovator, perangkat desa, dan kementerian. Perkembangan KMS DDI sudah masuk tahun ketiga dengan berbagai fitur yang sudah dikembangkan, fitur-fitur tersebut bersamaan dengan pembagian hak akses dapat dilihat pada Tabel 1.

Tabel 1 Pembagian hak akses pada KMS DDI

No	Fitur	*Role*
1	*Register* dan *Login*	Inovator, Desa, Admin, Kementerian
2	Pencarian dan *Filter*	Inovator, Desa, Admin
3	*Form* tambah inovasi	Inovator
4	*Form edit profile*	Inovator, Desa
5	*Form edit* inovasi	Inovator
6	Kategori, *card* inovasi, dan *inovator*	Inovator, Desa, Admin
7	*Card* desa	Inovator, Desa, Admin
8	Klaim inovasi dan manual	Desa
9	Daftar pengajuan klaim	Desa
10	Daftar pengajuan inovasi	Inovator
11	Verifikasi desa dan inovator	Admin
12	Verifikasi klaim dan tambah inovasi	Admin
13	*Dashboard*	Inovator, Desa, Admin, Kementerian
14	Notifikasi	Inovator, Desa, Admin, Kementerian
15	Rekomendasi inovasi	Desa

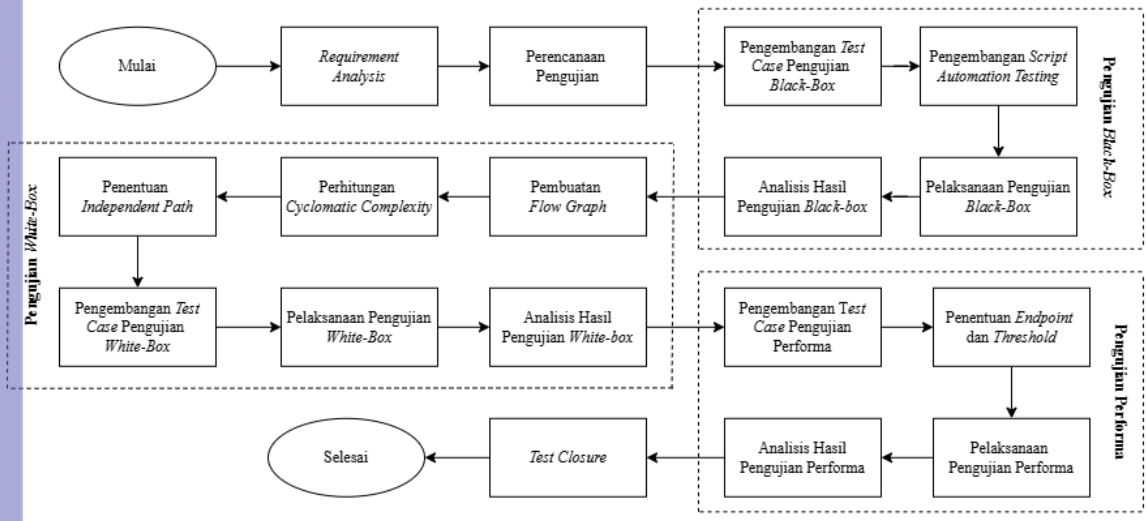
3.2 Tahapan Penelitian



Gambar 6 *Software Testing Life Cycle* (STLC)

Tahapan dari penelitian ini menggunakan metode *Software Testing Life Cycle* (STLC) yang merupakan pengembangan dari siklus perangkat lunak *Software*

Development Life Cycle (SDLC), dengan tahapan yang dimulai dari fase *planning*, *testing*, dan *deploying* (Ragunath *et al*. 2010). Pada SDLC fase *testing* merupakan fase penting yang bertujuan untuk memverifikasi kebutuhan pengguna dan memvalidasi kualitas perangkat lunak (Arfan dan Hendrik 2022). Fase *testing* tersebut memiliki fasenya sendiri dengan enam tahap utama meliputi *requirement analysis*, *test planning*, *test case development*, *environment setup*, *test execution*, dan *test closure* (Kusum *et al.* 2024).



Gambar 7 Tahapan penelitian

Penelitian terdiri dari 17 tahapan, diawali dengan *requirement analysis* dalam memetakan fitur yang akan diuji, dilanjutkan dengan perencanaan pengujian dalam menentukan metode dan cakupan pengujian, selanjutnya pembuatan *test case, environment setup,* dan pelaksanaan disesuaikan dengan masing-masing pengujian. Setelah pengujian selesai tahapan diakhiri dengan *test closure* untuk mendokumentasikan *bug* yang ditemukan, merumuskan rekomendasi perbaikan, dan merekap hasil akhir yang didapat selama siklus pengujian. Tahapan penelitian lengkap dapat dilihat pada Gambar 7.

3.3 *Requirement Gathering*

Tahapan pertama yang dilakukan adalah *requirement gathering*, tahap ini bertujuan untuk melakukan pemahaman menyeluruh terhadap fitur-fitur yang akan diuji. Tahap ini diawali dengan pemahaman *use case* dan *activity diagram* yang sebelumnya sudah dibuat oleh Nuryantika (2023). Pada pengujian, *use case* digunakan sebagai dasar dalam penentuan skenario pengujian, lalu *activity diagram* digunakan dalam menentukan alur navigasi yang harus diuji (Santi 2026). Setelah *use case* dan *activity diagram* tersebut selesai dipahami, tahap selanjutnya adalah menyusun *Requirements Traceability Matrix* (RTM) untuk memastikan tidak ada fitur yang terlewat dalam pengujian, fitur dibangun atas dasar kebutuhan yang jelas, dan memastikan alur pengujian yang konsisten (Santi 2026).

3.4 Perencanaan Pengujian

Pada tahap ini dilakukan perencanaan pengujian yang akan dilakukan meliputi penentuan cakupan dan arah pengujian, perkiraan waktu pengujian, metode yang

digunakan, serta resiko yang mungkin terjadi dalam pengujian (Arfan dan Hendrik 2022). Pada pengujian KMS DDI akan dilakukan tiga pengujian yaitu pengujian *black-box*, *white-box*, dan performa. Pengujian *black-box* diterapkan dengan pelaksanaan pengujian secara automasi menggunakan Katalon Studio dan manual berdasarkan observasi penguji. Pada pengujian *white-box* digunakan teknik *basis path* dengan beberapa tahapan seperti pembuatan *flow graph*, *cyclomatic complexity*, dan penentuan *independent path*, *software* yang digunakan pada pengujian ini adalah Jest yang digunakan untuk mengeksekusi *path* pada setiap *independent path*. Terakhir pada pengujian performa dengan teknik *load testing, software* yang digunakan adalah Apache JMeter dengan menerapkan beberapa variasi beban kerja tertentu dalam mensimulasikan beban pengguna harian.

3.5 Pengujian *Black-Box*

3.5.1 Pengembangan *Test Case*

Pada tahap ini dilakukan pengembangan *test case* untuk pengujian *black-box*. *Test case* disesuaikan berdasarkan hasil pemahaman sistem dengan menerapkan teknik *decision table, equivalence partitioning, boundary value analysis,* dan *error guessing* sebagai teknik komplementer. *Gherkin language* diterapkan menggunakan struktur *Given-When-Then* sebagai format penulisan. Struktur *test case* terdiri atas 10 atribut, dengan penjelasan fungsi dari tiap atribut dapat dilihat pada Tabel 2.

Tabel 2 Struktur *test case* pengujian *black-box* (Abdillah 2025)

No	Kolom	Penjelasan
1	*Test ID*	Identitas unik yang mempermudah pelacakan dan referensi *test case* tertentu
2	*Feature*	Fitur atau modul sistem yang sedang diuji
3	*Scenario Testing*	Deskripsi singkat mengenai skenario pengujian yang dilakukan
4	*Scenario Type*	Mengklasifikasi skenario sebagai pengujian positif atau negatif
5	*Pre-Condition*	Menjelaskan kondisi atau syarat yang harus dipenuhi sebelum pengujian dilakukan
6	*Step Testing*	Langkah-langkah yang harus dilakukan selama pengujian
7	*Test Data*	Data yang digunakan untuk menjalankan skenario pengujian
8	*Expected Result*	Hasil yang seharusnya muncul jika sistem berjalan sesuai yang diharapkan
9	*Actual Result*	Hasil nyata setelah skenario pengujian dijalankan
10	*Test Status*	Menyatakan status pengujian, apakah berhasil, gagal, atau belum dilakukan

3.5.2 Pengembangan *Script Automation Testing*

Pada tahap ini, proses pengujian berlanjut dengan mengembangkan *script* automasi berdasarkan *test case* yang telah disusun sebelumnya. Pengembangan dilakukan menggunakan Katalon Studio, setiap langkah pada *test case* diterjemahkan menjadi *script* automasi yang akan dijalankan. Selanjutnya, *script* automasi dikelompokkan ke dalam *test suite* sehingga dapat dijalankan secara terstruktur sesuai fiturnya. *Test suite* yang sudah disusun kemudian ditambahkan *script setUp* dan *tearDown* untuk menyisakan langkah-langkah penting dalam *test case* dan mencegah langkah yang berulang pada awal dan akhir pengujian.

3.5.3 Pelaksanaan Pengujian

Setelah semua *test case* selesai disusun dan *script* automasi siap digunakan, pengujian akan dijalankan untuk memeriksa fungsionalitas dalam setiap fitur pada sistem. Pelaksanaan pengujian *black-box* hanya berfokus memeriksa kesesuaian antara input dan output yang dihasilkan sistem. Pengujian dilakukan dengan pendekatan automasi dan manual sesuai pembagian fitur yang sudah direncanakan. Selama pengujian berlangung, hasil dari setiap eksekusi yang dijalankan akan dicatat pada kolom *actual result* dan setiap *bug* atau *error* yang ditemukan akan dicatat pada *bug documentation*.

3.5.4 Analisis Hasil Pengujian

Hasil pengujian *black-box* yang sudah didapat akan dianalisis yang berfokus dalam untuk identifikasi *bug* yang ditemukan, termasuk klasifikasi berdasarkan *error type* dan *severity* berdasarkan dampaknya terhadap fungsi sistem. Klasifikasi *error type* pada *bug* yang ditemukan didasarkan pada penelitian Yousuf dan Sofi (2025) dan *severity* yang berdasar pada IEEE (2009), sedangkan urutan perbaikan *bug* yang ditemukan didasarkan pada *severity* dan urgensi dari setiap fiturnya.

3.6 Pengujian *White-Box*

Pada pengujian *white-box* digunakan pendekatan *basis path* yang berfokus pada logika internal serta jalur eksekusi yang mungkin terjadi. Tujuan dari penerapan pengujian ini adalah memastikan bahwa seluruh jalur logika program teruji dan tidak ada kode yang tidak dieksekusi. Berdasarkan seluruh fitur yang terdapat pada KMS DDI, dipilih fitur utama yang didasarkan pada kompleksitas logika sekaligus menguji alur paling optimal pada fitur-fitur tersebut.

3.6.1 Pembuatan *Flow Graph*

Tahap ini menghasilkan representasi visual dari alur logika kode menggunakan diagram alir (*flow graph*). Setiap *node* pada *flowgraph* menggambarkan blok program yang akan diuji, sedangkan *edge* menunjukkan hubungan aliran eksekusi antar blok tersebut. *Flowgraph* membantu penguji memahami struktur kontrol sebuah fungsi atau modul dengan lebih jelas, termasuk cabang *if*, *loop*, dan titik keputusan (*decision node*). Diagram ini menjadi dasar untuk menghitung kompleksitas dan menentukan jalur pengujian. Pembuatan *flow graph* dilakukan dengan menggunakan *platform* Draw.io.

3.6.2 Perhitungan *Cyclomatic Complexity* (CC)

Nilai CC dihitung berdasarkan *flowgraph* untuk mengetahui tingkat kompleksitas logika program serta jumlah *independent path* yang harus diuji. Nilai kompleksitas dihitung menggunakan rumus $V(G) = E - N + 2$, di mana E adalah jumlah *edge* dan N adalah jumlah *node*. Nilai kompleksitas ini menunjukkan banyaknya *independent path* yang wajib diuji untuk mencapai *basis path coverage* secara menyeluruh. Semakin tinggi nilainya, semakin kompleks kode tersebut dan semakin banyak *test case* yang diperlukan.

3.6.3 Penentuan *Independent Path*

Setelah mengetahui nilai CC, dilakukan penentuan *independent path* pada *flowgraph*. *Independent path* adalah jalur eksekusi unik yang mewakili satu set kondisi logika yang berbeda dari jalur lainnya. Jalur ini dipilih untuk memastikan seluruh keputusan dan cabang yang mungkin terjadi telah tercakup. Setiap jalur kemudian dijadikan dasar penyusunan *test case* sehingga semua kemungkinan alur logika dalam kode dapat diuji sepenuhnya.

3.6.4 Pengembangan *Test Case*

Test case yang disusun disesuaikan dengan *independent path* yang sudah ditentukan. Struktur yang digunakan pada pengujian ini terdiri atas 11 atribut, dengan penjelasan dari tiap atribut dapat dilihat pada Tabel 3.

Tabel 3 Struktur *test case* pengujian *white-box* (Abdillah 2025)

No	Kolom	Penjelasan
1	*Path ID*	Mengidentifikasi jalur logika tertentu yang diuji dalam kode program.
2	*Graph*	Menampilkan diagram atau alur kontrol yang menunjukkan *node* pada program.
3	*Feature*	Menunjukkan fitur pada program yang akan diuji secara internal.
4	*Independent Path*	Menjelaskan jalur eksekusi unik dalam kode seperti percabangan (*if-else*) atau *loop*.
5	*Scenario Testing*	Menguraikan skenario tertentu yang digunakan untuk menguji suatu jalur (*path*).
6	*Scenario Type*	Mengklasifikasi skenario sebagai pengujian positif atau negatif.
7	*Step Testing*	Menjabarkan langkah-langkah untuk mengeksekusi jalur logika tersebut.
8	*Test Data*	Input yang digunakan untuk menjalankan skenario tertentu.
9	*Expected Result*	Hasil yang diharapkan setelah langkah pengujian dijalankan.
10	*Actual Result*	Hasil sesungguhnya setelah langkah pengujian dijalankan.
11	*Test Status*	Penanda hasil pengujian berupa berhasil, gagal, dan belum diuji.

3.6.5 Pelaksanaan Pengujian

Pada tahap ini dilakukan eksekusi *test case* berdasarkan *independent path* pada *flowgraph* yang sudah disusun. Setiap skenario dijalankan menggunakan Jest dan React Testing Library untuk melakukan pengujian berdasarkan *path* yang terdapat dalam setiap jalur *independent.* Untuk melakukan visualisasi terhadap *error* yang didapat, Postman digunakan dalam memverifikasi output dari *endpoint* yang terdapat *error*. Hasil dari setiap skenario yang diuji akan dicatat pada kolom *actual result* dan setiap *bug* yang ditemukan akan didokumentasikan dalam bentuk *bug documentation*.

3.6.6 Analisis Hasil Pengujian

Analisis hasil pada pengujian *white-box* berfokus untuk meninjau penyebab terjadinya *bug* dan dampaknya pada jalur logika sistem. Temuan yang didapat diklasifikasikan berdasarkan *error type, severity,* dan prioritas dengan metode yang sama seperti pengujian *black-box* untuk memudahkan penentuan urutan perbaikan logika sistem.

3.7 Pengujian Performa

3.7.1 Pengembangan *Test Case*

Pada tahap ini *test case* disusun dengan variasi jumlah *virtual users* sebanyak 1, 10, 20, 50, 100, 250, dan 500. Jumlah tersebut disesuaikan dalam mensimulasikan beban pengguna harian. Struktur *test case* pada pengujian performa terdiri dari 13 atribut yang disesuaikan untuk pengujian dengan teknik *load testing*, bersumber dari penelitian Abdillah (2025). Tabel 4 merupakan penjelasan dari struktur *test case* yang digunakan pada pengujian performa.

Tabel 4 Struktur *test case* pengujian performa (Abdillah 2025)

No	Kolom	Penjelasan
1	*API Endpoint*	Menjelaskan *endpoint* API yang diuji pada beban kerja tertentu
2	*Virtual Users* (VU)	Menyatakan jumlah pengguna virtual yang disimulasikan selama pengujian
3	*Ramp-Up Period*	Mencatat durasi peningkatan jumlah pengguna hingga mencapai beban penuh
4	*Satisfied Threshold*	Menentukan batas waktu respon yang dianggap memuaskan
5	*Tolerating Threshold*	Menunjukkan batas toleransi maksimum dari waktu respon
6	*AVG Response Time*	Rata-rata waktu respon pada seluruh durasi pengujian
7	*Min / Max / Median*	Menyajikan data statistik berupa waktu respon tercepat, terlama, dan nilai tengahnya
8	*90 Percentile*	Menggambarkan waktu respon yang dialami oleh 90% pengguna
9	*Throughput*	Mengukur jumlah *request* yang dapat diproses per detik
10	*Network (Received/Sent)*	Menunjukkan data jaringan yang masuk dan keluar selama pengujian berlangsung
11	*Error Rate*	Persentase kegagalan *request* selama pengujian berlangsung
12	*APDEX Score*	Mengukur tingkat kepuasan pengguna berdasarkan *response time* pada *threshold* yang ditentukan
13	*APDEX Level*	Mengklasifikasikan level tingkat kepuasan berdasarkan skor Apdex, yaitu *Excellent, Good, Fair, Poor,* atau *Unacceptable*

3.7.2 Penentuan *Endpoint* dan *Threshold*

Tahap ini dimulai dengan mengidentifikasi *endpoint* yang akan diuji, pemilihan *endpoint* dilakukan berdasarkan tingkat kritikalitas fungsi serta potensi beban tinggi yang dapat terjadi pada penggunaan nyata. Setelah *endpoint* ditentukan, langkah berikutnya adalah menetapkan nilai *threshold* sebagai dasar evaluasi performa sistem. Proses penetapan *threshold* dilakukan dengan merujuk pada prinsip *response time limit* yang dijelaskan oleh Nielsen (1993), di mana respons 0,1 detik dinilai instan, 1 detik masih dapat diterima, dan 10 detik merupakan batas toleransi maksimal interaksi pengguna, sementara praktik modern cenderung menggunakan acuan Apdex sesuai kebutuhan bisnis (Samudrala 2022).

3.7.3 Pelaksanaan Pengujian

Pengujian performa dilakukan menggunakan Apache JMeter sesuai dengan test case yang telah dirancang. Seluruh *error* dan ketidaksesuaian yang muncul selama proses pengujian dicatat sebagai bagian dari proses evaluasi. Setiap

skenario menerapkan pengaturan *ramp-up period* untuk menaikkan jumlah virtual user secara bertahap hingga mencapai beban maksimum. Nilai *ramp-up* diselaraskan dengan jumlah pengguna virtual, sehingga pengiriman *request* tetap berada pada kapasitas normal sistem dan tidak langsung memicu lonjakan beban.

3.7.4 Analisis Hasil Pengujian

Hasil pengujian akan dianalisis untuk mengidentifikasi temuan *error* atau ketidaksesuaian *response time* terhadap *threshold* yang sudah ditentukan. Analisis berfokus pada *endpoint* yang mengalami *bottleneck* dan keterlambatan *response time*, dengan level Apdex minimal untuk tiap *endpoint* berada pada level *good*. Analisis ini bertujuan untuk mengetahui bagian mana dari sistem yang memerlukan optimalisasi dari segi performa.

3.8 *Test Closure*

Tahap ini merupakan fase penutup dalam *Software Testing Life Cycle* (STLC). Pada tahap ini akan disusun laporan akhir yang memuat hasil tes, metrik pengujian serta tingkat keberhasilan pengujian yang dilakukan (Arfan dan Hendrik 2022). Aktivitas yang dilakukan meliputi evaluasi cakupan pengujian, perbandingan *exit criteria* dengan hasil aktual, serta analisis *bug* yang ditemukan dan menyusun rekomendasi perbaikan sistem. Hasil yang didapatkan akan membantu proses identifikasi bagian sistem yang memerlukan optimasi agar performa sistem memenuhi standar yang ditetapkan. Selain itu, tahap ini juga mencakup pengumpulan seluruh data hasil pengujian dan evaluasi menyeluruh terhadap proses yang telah dilakukan untuk meningkatkan efektivitas pengujian pada siklus berikutnya (Ruliansyah *et al.* 2023).

3.9 Peralatan Penelitian

Penelitian ini dilakukan dengan perangkat keras dan perangkat lunak sebagai berikut:

a Perangkat keras pribadi berupa laptop dengan spesifikasi:
   1) Processor Intel Core i3-1115G4
   2) RAM 8 GB dan Penyimpanan SSD 512 GB

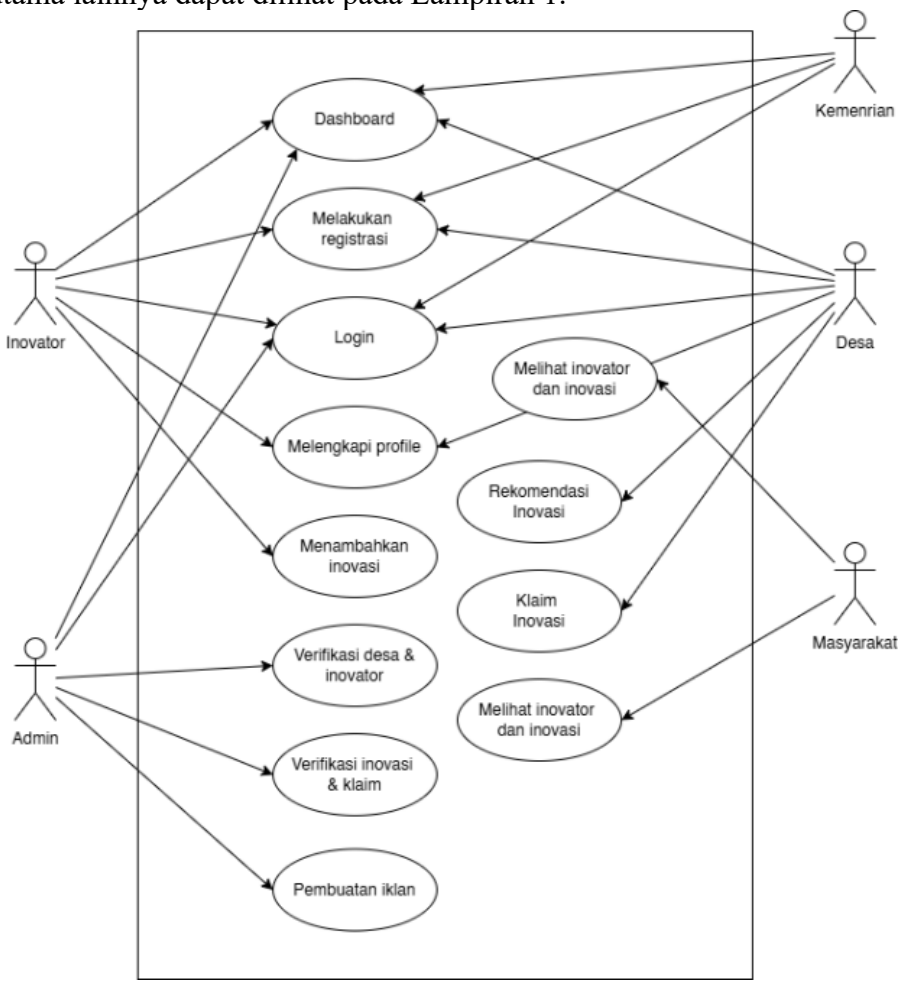
b Spesifikasi server KMS DDI
   1) *Cloud service*: Hostinger
   2) CPU: 4 vCPU
   3) Memori: 16 GB
   4) Penyimpanan: 200 GB
   5) Sistem Operasi: Ubuntu 24.04 LTS

c Perangkat lunak yang digunakan antara lain:
   1) Sistem Operasi: Windows 11
   2) *Test Case*: Google Spreadsheet
   3) Dokumentasi: Google Drive
   4) Pengujian *Black-Box*: Katalon Studio
   5) Pengujian *White-Box*: Jest dan Postman
   6) Pengujian Performa: Apache JMeter

IV HASIL DAN PEMBAHASAN

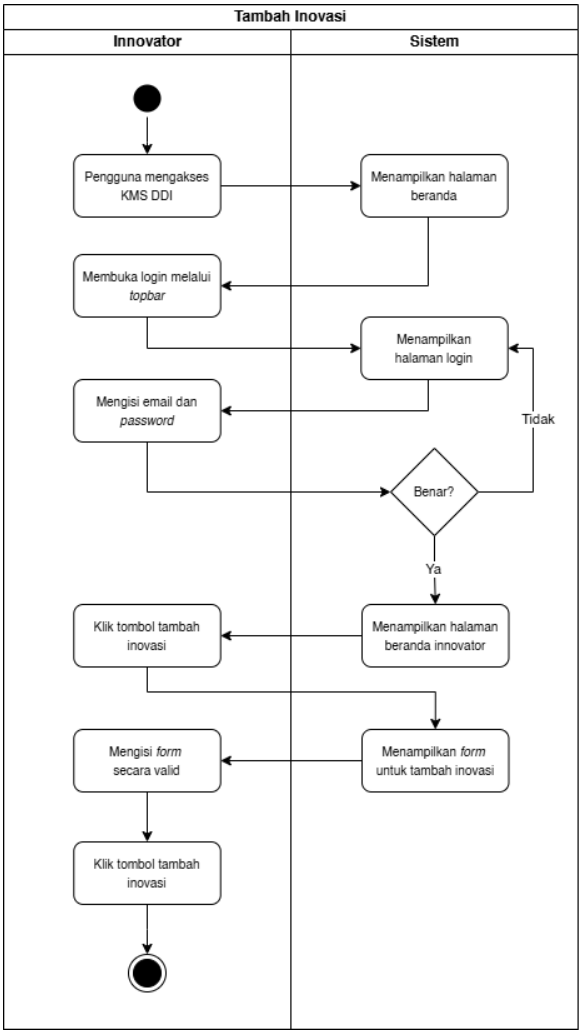
4.1 *Requirement Gathering*

Tahap *requirement gathering* bertujuan memahami fitur yang akan diuji secara menyeluruh berdasarkan *use case* dan *activity diagram* yang telah disusun dari penelitian sebelumnya. Gambar 8 merupakan *use case* terbaru hasil modifikasi dari penelitian Tasnim (2023) dan Gambar 9 merupakan *activity diagram* untuk fitur tambah inovasi berdasarkan penelitian Nuryantika (2023). *Activity diagram* untuk fitur utama lainnya dapat dilihat pada Lampiran 1.



Gambar 8 *Use case* diagram KMS DDI (dimodifikasi dari Tasnim 2023)

Setelah dilakukan pemahaman terkait *use case* dan *activity diagram*, dibuat RTM sebagai dokumen yang memetakan fitur-fitur yang diuji ke dalam *test case*. *Use case* tersebut terdiri dari 12 fitur yang dipetakan menjadi Req ID, kemudian alur dari tiap fiturnya dipetakan dari *activity diagram* menjadi *requirement description*. Hasil lengkap analisis RTM dapat dilihat pada Lampiran 2, kemudian contoh untuk fitur login, pencarian dan filter, dan tambah inovasi dapat dilihat pada Tabel 5.



Gambar 9 *Activity diagram* menambahkan inovasi (Nuryantika 2023)

Tabel 5 Analisis RTM fitur KMS DDI

Req ID	Fitur	*Requirement Description*	*Test Case ID*
REQ-001	Login	Email dan password wajib diisi dengan kombinasi yang benar	TC-A-001, TC-A-002, TC-A-003, TC-A-004, TC-A-005
		Email harus sudah terdaftar dan sesuai format	TC-A-006, TC-A-007, TC-A-008
		Terdapat *error message* jika input email tidak valid / password salah	TC-A-009, TC-A-010,TC-A-011
		Login dengan google bisa dilakukan dengan email yang sudah terdaftar	TC-A-012
		Setelah berhasil login, terdapat opsi logout pada popup profil	TC-B-001
		Terdapat mekanisme lupa password, dengan mengirim tautan reset ke email terdaftar	TC-D-001, TC-D-002, TC-D-003, TC-D-004
REQ-003	Pencarian dan Filter	Pada *homepage,* terdapat pencarian yang bisa mencari inovasi secara spesifik	TC-E-001, TC-E-002

Tabel 5 Analisis RTM fitur KMS DDI *(lanjutan)*

Req ID	Fitur	*Requirement Description*	*Test Case ID*
		Pada halaman desa, terdapat pencarian dan filter yang bisa mencari desa secara spesifik	TC-F-001, TC-F-002, TC-G-001, TC-G-002, TC-G-003, TC-G-004, TC-G-005, TC-G-006
		Pada halaman inovator, terdapat pencarian dan filter yang bisa mencari inovator secara spesifik	TC-H-001, TC-H-002, TC-I-001, TC-I-002, TC-I-003
REQ-004	Tambah Inovasi	Semua *field* wajib dalam form inovasi harus terisi	TC-J-001, TC-J-006, TC-J-007, TC-J-008
		Foto inovasi dibatasi maksimal lima foto dan harus JPG atau PNG	TC-J-002, TC-J-003, TC-J-004, TC-J-005
		Field manfaat bisa ditambah setelah mengisi *field* pertama	TC-J-009, TC-J-010
		Field infrastruktur bisa ditambah setelah mengisi *field* pertama	TC-J-011, TC-J-012
		Inovasi yang sudah ditambah bisa diedit atau dihapus	TC-O-001, TC-O-002

4.2 Perencanaan Pengujian

Perencanaan dimulai dengan penentuan fitur yang akan diuji pada tiap pengujian. Pengujian *black-box* akan menguji seluruh fitur yang sudah dikembangkan, pengujian *white-box* berfokus pada fitur di alur kerja utama sistem, sementara pengujian performa diprioritaskan pada *endpoint* yang bersifat kritis dan berpotensi memiliki beban akses yang tinggi. Daftar fitur dan *endpoint* yang akan diuji pada tiap pengujian dapat dilihat pada Tabel 6.

Tabel 6 Pembagian fitur yang akan diuji

Tipe Pengujian	Fitur yang diuji	
Black-box	REQ-001 - Login	REQ-009 - Verifikasi admin
	REQ-002 - Register	REQ-010 - Pembuatan Iklan
	REQ-003 - Pencarian & Filter	REQ-011 - Notifikasi
	REQ-004 - Tambah & edit inovasi	REQ-012 - *Digital Nudge*
	REQ-005 - Profil desa	REQ-013 - Rekomendasi inovasi
	REQ-006 - Profil inovator	REQ-014 - *Dashboard*
	REQ-007 - Klaim inovasi & manual	REQ-015 - *Card* dan Kontak
	REQ-008 - Pengajuan klaim & inovasi	
White-box	1 - *Login*	5 - Profil desa
	2 - *Register*	6 - Profil inovator
	3 - Pengenalan autentikasi	7 - Klaim inovasi
	4 - Tambah inovasi	8 - Verifikasi admin
Performance	1 - Lihat inovasi	7 - Hapus inovasi
	2 - Lihat detail desa	8 - Pencarian inovasi
	3 - Tambah inovasi	9 - Pencarian desa
	4 - Klaim inovasi	10 - Autentikasi
	5 - Verifikasi admin	11 - *Dashboard* desa
	6 - Edit klaim	12 - *Dashboard* inovator

Selanjutnya pada Tabel 7 ditentukan *entry criteria* berupa syarat minimum yang harus dipenuhi sebelum pengujian dimulai dan *exit criteria* berupa standar minimum yang harus dicapai agar pengujian dinyatakan selesai.

Tabel 7 *Entry* & *Exit Criteria*

Metode Pengujian	*Entry Criteria*	*Exit Criteria*
Black-box	Dokumen perencaan pengujian selesai disusun	Seluruh *test case* (312 skenario) sudah dieksekusi
	Sistem KMS DDI sudah di-*deploy* pada lingkungan *staging*	Tingkat keberhasilan pengujian (*success rate*) minimal 80%
	Test case pengujian selesai disusun	*Bug* yang ditemukan selesai diklasifikasikan
White-box	*Source code* sistem stabil dan siap diuji	Seluruh *independent path* telah dilewati minimal satu kali
	Struktur logika sistem sudah dipetakan dalam bentuk node	Nilai *cyclomatic complexity* selesai dihitung
	Flowgraph sistem selesai dibuat	Struktur logika dengan *error* dibuat saran perbaikan
Performance	Pengujian fungsional selesai dilakukan dengan 80% *success rate*	Melakukan simulasi beban pada 13 *endpoint* yang sudah dipilih
	Tools yang digunakan selesai dikonfigurasi	Hasil yang didapatkan seperti *response time*, *error rate, throughput* selesai direkap
	Sistem KMS DDI sudah di-*deploy* pada lingkungan *staging*	Apdex *score* sudah ditentukan berdasarkan hasil pengujian

4.3 Pengujian *Black-Box*

4.3.1 Pengembangan *Test Case*

Pengujian *black-box* dilaksanakan pada seluruh fitur KMS DDI yang didasarkan pada Req ID. Terdapat total 312 skenario yang dijalankan, terdiri dari 251 skenario positif dan 61 skenario negatif. Pengujian ini dilaksanakan dengan dua pendekatan yaitu *manual testing* berdasarkan observasi sebanyak 190 skenario dan *automation testing* menggunakan Katalon studio sebanyak 122 skenario. Rincian jumlah skenario untuk setiap fitur dapat dilihat pada Tabel 8, sedangkan pembagian fitur berdasarkan pengujian yang digunakan ditunjukkan pada Tabel 9.

Tabel 8 Rincian jumlah skenario pengujian *black-box*

Req ID	Fitur	Positif	Negatif	Total Skenario
REQ-001	Login	10	7	17
REQ-002	Register	5	7	12
REQ-003	Pencarian & Filter	15	0	15
REQ-004	*Form* Tambah Inovasi	10	4	14
REQ-005	Profil desa	9	5	14
REQ-006	Profil inovator	7	4	11
REQ-007	Klaim inovasi	30	27	57

Tabel 8 Rincian jumlah skenario pengujian *black-box* (*lanjutan*)

Req ID	Fitur	Positif	Negatif	Total Skenario
REQ-008	Daftar pengajuan	11	0	11
REQ-009	Verifikasi admin	24	4	28
REQ-010	Pembuatan Iklan	6	3	9
REQ-011	Notifikasi	17	0	17
REQ-012	*Digital Nudge*	6	0	6
REQ-013	Sistem rekomendasi	2	0	2
REQ-014	*Dashboard* admin, desa, inovator, menteri	72	0	72
REQ-015	Kategori, *card*, dan kontak	27	0	27
	Total	**251**	**61**	**312**

Tabel 9 Pembagian fitur berdasarkan pendekatan pengujian *black-box*

Automation Testing	*Manual Testing*
Autentikasi (REQ-001, REQ-002)	Pengajuan klaim dan inovasi (REQ-009)
Pencarian dan Filter (REQ-003)	Notifikasi (REQ-011)
Form tambah inovasi (REQ-004)	*Digital nudge* (REQ-012)
Form registrasi profil (REQ-005, REQ-006)	*Dashboard* (REQ-014)
Klaim inovasi (REQ-007)	Kategori, *card*, dan kontak (REQ-015)
Verifikasi admin (REQ-009)	
Pembuatan iklan (REQ-010)	

Pengembangan *test case* dilakukan dengan menerapkan beberapa teknik pengujian *black-box*, yaitu *equivalence partitioning, boundary value analysis* (BVA), *decision table* yang disusun berdasarkan analisis *cause-effect*, serta *state transition*. Selain keempat teknik utama tersebut, digunakan teknik *error guessing* sebagai pelengkap untuk menjangkau skenario pengujian yang belum tercakup oleh teknik-teknik sebelumnya. *Test case* lengkap disertai dengan teknik yang digunakan dapat dilihat pada tautan di Lampiran 3, adapun contoh analisis pengembangan test case dari setiap teknik yang diterapkan sebagai berikut:

a) *Equivalence Partitioning*

Pada pengujian KMS, teknik ini diimplementasikan pada *form* autentikasi, misalnya pada fitur registrasi. Pada fitur registrasi, kondisi dibagi menjadi input valid yaitu mengisi semua *field* dengan lengkap, dan input tidak valid dengan mengosongkan atau mengisi *field* dengan format salah. Tabel 10 menunjukkan contoh *equivalence class* pada fitur registrasi.

Tabel 10 *Equivalence Partitioning* fitur registrasi

Equivalence Class	Contoh Input
Valid	Form lengkap, Input sesuai format, Daftar dengan Google
Invalid	Form tidak lengkap, Email tidak menggunakan (@), Role tidak diisi, Email sudah terdaftar, Password kurang enam karakter

b) Boundary Value Analysis

Pada pengujian KMS, teknik ini diimplementasikan pada *form* tambah inovasi, khususnya pada *field* deskripsi inovasi yang memiliki batas minimal 1 kata dan batas maksimal 80 kata. Pengujian dilakukan dengan mengambil titik data tepat pada batas, satu nilai di bawah batas, dan satu nilai di atas batas, baik pada batas bawah maupun batas atas. Tabel 11 menunjukkan contoh *boundary value* pada *field* deskripsi inovasi.

Tabel 11 *Boundary Value Analysis* deskripsi inovasi

Titik Uji	Jumlah Kata	Kategori
Min - 1	0 kata	Invalid
Min	1 kata	Valid
Min + 1	2 kata	Valid
Max – 1	79 kata	Valid
Max	80 kata	Valid
Max + 1	81 kata	Invalid

c) *Decision Table*

Pada pengujian KMS teknik ini diterapkan pada fitur klaim inovasi, sebagai berikut:

Condition (*Cause*):

- C1 = Data desa lengkap
- C2 = Desa terverifikasi
- C3 = Mengisi data klaim

Action (*Effect*)

- E1 = Klaim inovasi berhasil
- E2 = Klaim inovasi gagal

Tabel 12 *Decision table* fitur klaim

Kondisi	Rule 1	Rule 2	Rule 3	Rule 4
C1 = Data desa lengkap	Y	Y	T	T
C2 = Desa terverifikasi	Y	T	-	-
C3 = Mengisi data klaim	Y	-	Y	-
E1 = Klaim berhasil	✓	-	-	-
E2 = Klaim gagal	-	✓	✓	✓

Keterangan:

- Y = Ya (kondisi terpenuhi)
- T = Tidak (kondisi tidak terpenuhi)
- – = *don't care* (nilai kondisi tidak memengaruhi hasil karena sudah ditentukan oleh kondisi lain yang bernilai Tidak)

Berdasarkan tabel keputusan (*decision table*) tersebut, *rule* yang terbentuk adalah:

- Rule 1: Jika data desa lengkap AND desa terverifikasi AND mengisi data klaim, maka klaim inovasi berhasil.
- Rule 2: Jika data desa lengkap AND desa belum terverifikasi, maka klaim inovasi gagal.
- Rule 3: Jika data desa belum lengkap AND mengisi data klaim, maka klaim inovasi gagal.

- Rule 4: Jika data desa belum lengkap AND tidak mengisi data klaim, maka klaim inovasi gagal.

d) *State Transition*

Pada pengujian KMS, teknik ini diimplementasikan pada fitur verifikasi admin, khususnya untuk menguji perubahan status (*state*) akibat aksi (*event*) yang dilakukan oleh admin. Status terdiri atas tiga kondisi, yaitu Menunggu, Terverifikasi, dan Ditolak, yang dapat berubah melalui event *verify*, *reject*, maupun *cancel* pada proses penolakan. Tabel 13 menunjukkan contoh *state transition* pada fitur verifikasi klaim inovasi.

Tabel 13 *State Transition* fitur verifikasi klaim inovasi

State Awal	Event	State Akhir	Keterangan
Pending	Verify	Terverifikasi	Klaim berhasil diverifikasi dan tampil pada profil desa
Pending	Reject & Isi alasan	Ditolak	Klaim berhasil ditolak dengan alasan tersimpan
Pending	Reject, lalu cancel	Menunggu	Proses penolakan dibatalkan, status tidak berubah

e) *Error Guessing*

Pada implementasi dalam pengujian KMS, teknik pengujian ini diterapkan pada fitur profil desa. Fitur ini memiliki beberapa fungsi unggah dan hapus gambar, seperti logo, header, serta foto inovasi desa, yang berpotensi menimbulkan perilaku tidak terduga apabila tidak ditangani dengan baik oleh sistem. Tabel 14 menampilkan contoh skenario pengujian menggunakan pendekatan *error guessing* pada fitur profil desa.

Tabel 14 Contoh penerapan teknik *error guessing* pada fitur profil desa

Skenario	Tujuan
As a Village User, I am able to delete uploaded logo and header images	Menguji apakah sistem dapat mengembalikan tampilan awal, yaitu memunculkan kembali tombol tambah foto, setelah gambar dihapus.
As a Village User, I am able to delete village innovation photos	Menguji apakah sistem dapat memperbarui jumlah foto yang tersisa dan memunculkan kembali tombol tambah foto setelah sebagian foto dihapus.
As a Village User, I am able to cancel a full village profile submission	Menguji apakah sistem dapat mempertahankan data yang telah diisi pada form ketika proses submit dibatalkan oleh pengguna.

Setelah seluruh *test case* dirancang berdasarkan teknik-teknik di atas, selanjutnya penulisan *test case* akan diseragamkan ke dalam format yang terstruktur dan mudah dipahami menggunakan *gherkin language*. *Gherkin language* merupakan sebuah format penulisan skenario yang menggunakan struktur *Given-When-Then*, di mana *Given* menjelaskan kondisi awal sistem, *When* menjelaskan aksi yang dilakukan, dan *Then* menjelaskan hasil yang diharapkan setelah aksi tersebut dilakukan (Pahlevi 2024). Penggunaan format ini bertujuan agar setiap *test case* memiliki struktur yang konsisten, mudah dibaca baik oleh pengembang maupun non-teknis, serta memudahkan proses dokumentasi dan identifikasi skenario pengujian pada tahap selanjutnya. Keseluruhan *test case* pengujian *black-*

box dapat dilihat pada Lampiran 3, sedangkan contoh *test case* dengan format *gherkin* pada fitur tambah inovasi dapat dilihat pada Tabel 15.

Tabel 15 Contoh *test case* untuk fitur tambah inovasi

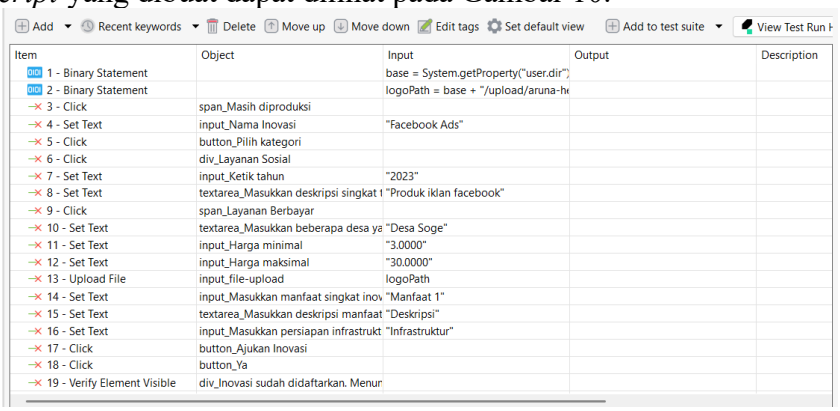
ID	*Scenario Testing*	*Scenario Type*	*Step Testing*	*Test Data*
TC-J-001	*As an Innovator User, I am unable to submit an empty innovation form*	*Negative*	***Given*** *I am on the add innovation form* ***When*** *I leave all required input fields empty* ***And*** *I click the submit innovation button* ***And*** *I confirm the popup message* ***Then*** *the innovation submission should fail*	*Status: (empty)* *Name: (empty)* *Category: (empty)* *...all fields empty*
TC-J-002	*As an Innovator User, I am able to upload multiple photos for an innovation*	*Positive*	***Given*** *I am on the add innovation form* ***When*** *I fill all required fields* ***And*** *I upload an additional photo in the photo field* ***And*** *I click submit and confirm the popup* ***Then*** *the photos should be uploaded*	*Photo 1: inovasi_1.png* *Photo 2: inovasi_2.jpg*
TC-J-006	*As an Innovator User, I am able to successfully submit a fully valid innovation form*	*Positive*	***Given*** *I have filled all form fields with valid inputs* ***When*** *I click submit* ***And*** *I click "Yes" on the confirmation popup* ***Then*** *the form should submit properly*	*All required fields valid*
TC-J-008	*As an Innovator User, I am unable to submit partially filled fields*	*Negative*	***Given*** *I am on the add innovation form* ***When*** *I partially fill the required fields* ***And*** *I click submit and confirm the popup* ***Then*** *the submission should fail*	*All required fields valid*

4.3.2 Pengembangan *Script Automation Testing*

1) Pembuatan *test script*

Tahapan dimulai dengan pembuatan *test script* yang didasarkan pada *test case* yang sudah disusun. *Test script* dikembangkan berdasarkan *record* manual yang dilakukan menggunakan fitur *web recorder* pada Katalon Studio. Setiap skenario dijalankan secara manual dengan seluruh aktivitas, input teks, dan navigasi halaman akan

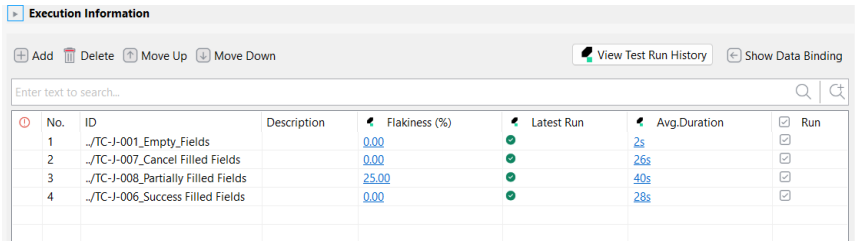
dikonversi menjadi *script* sesuai dengan aktivitas penguji. Contoh *test script* yang dibuat dapat dilihat pada Gambar 10.



Gambar 10 Contoh *test script* fitur tambah inovasi

2) Pengelompokkan *test suite*

Setelah semua *script* selesai dikonfigurasi, selanjutnya *script* tersebut akan dikelompokkan sesuai dengan fiturnya masing-masing pada *test suite*. *Test suite* berfungsi sebagai wadah untuk mengelompokkan *test case* berdasarkan tujuan tertentu, misalkan *test case* pada fitur *register, login*, lupa *password*, dan *logout* dikelompokkan menjadi *test suite* autentikasi untuk dijalankan secara bersamaan agar pengujian berjalan lebih efisien.



Gambar 11 Pengelompokkan *test suite* fitur yang sama

3) Fungsi *setUp* dan *tearDown*

Fungsi *setUp* dan *tearDown* digunakan pada *test case* yang sudah dikelompokkan pada *test suite*. Kedua fungsi ini merupakan konfigurasi tambahan untuk menyiapkan lingkungan agar bersih dan hanya berfokus pada *step* utama tiap *test case*. Pada *test suite* untuk fitur tambah inovasi, fungsi *setUp* berfungsi untuk menjalankan *step* membuka *browser*, melakukan login, dan navigasi ke *form* tambah inovasi, fungsi ini akan dijalankan satu kali pada awal *test suite*. Fungsi *tearDown* berfungsi untuk menjalankan *step* di akhir, seperti *logout* dan menutup *browser*. Gambar 12 merupakan contoh fungsi *setUp* dan *tearDown* yang sudah dikonfigurasi.

```
@SetUp(skipped = false)
def setUp() {

    WebUI.openBrowser('')

    WebUI.navigateToUrl('https://desa-digital-v3-git-dev-ahmad-qaulan-sadidas-projects.vercel.app')

    WebUI.click(findTestObject('Z-Record/Inovasi/Page_Desa Digital Indonesia/a_Masuk'))

    WebUI.setText(findTestObject(
        'Z-Record/Inovasi/Page_Desa Digital Indonesia/input_Email'),
        'meta@gmail.com')

    WebUI.setEncryptedText(findTestObject(
        'Z-Record/Inovasi/Page_Desa Digital Indonesia/input_Kata sandi'),
        'xAqBSq2UvX4iVgyZ+55qUQ==')

    WebUI.click(findTestObject('Z-Record/Inovasi/Page_Desa Digital Indonesia/button_Masuk'))


    WebUI.click(findTestObject('Z-Record/Inovasi/Page_Desa Digital Indonesia/button_Tambah Inovasi'))

    WebUI.waitForPageLoad(10)
}
```

(a)

```
import static com.kms.katalon.core.testobject.ObjectRepository.findTestObject

@SetUp(skipped = false)
def setUp() {
}

@SetupTestCase(skipped = false)
def setupTestCase() {

    WebUI.navigateToUrl(
        'https://desa-digital-v3-git-dev-ahmad-qaulan-sadidas-projects.vercel.app'
    )

    WebUI.waitForPageLoad(10)
}

@TearDown(skipped = false)
def tearDown() {

    WebUI.closeBrowser()
}
```

(b)

Gambar 12 a) *setUp script* dan b) *teardown script* pada fitur tambah inovasi

4.3.3 Pelaksanaan Pengujian

Setelah seluruh *script automation* selesai dikembangkan, pengujian dilakukan melalui *automation* dan observasi manual. *Automation testing* menggunakan Katalon studio membantu mempercepat eksekusi *test case* secara konsisten, sedangkan pengujian manual digunakan untuk melakukan observasi lebih detail terhadap perilaku sistem dan mampu mendeteksi lebih banyak *bug* sebelum fase produksi (Rakly dan Andriyani 2025). Tabel 16 menunjukkan hasil pelaksanaan pengujian pada masing-masing fitur.

Tabel 16 Hasil pengujian *black-box*

Req ID	Fitur	Total Skenario	Berhasil	Gagal
REQ-001	Login	17	13	4
REQ-002	Register	12	9	3
REQ-003	Pencarian & Filter	15	15	0
REQ-004	*Form* Tambah Inovasi	14	12	2
REQ-005	Profil desa	14	14	0
REQ-006	Profil inovator	11	10	1
REQ-007	Klaim inovasi	57	56	1
REQ-008	Daftar pengajuan	11	10	1

Tabel 16 Hasil pengujian *black-box* (*lanjutan*)

Req ID	Fitur	Total Skenario	Berhasil	Gagal
REQ-009	Verifikasi admin	28	28	0
REQ-010	Pembuatan Iklan	9	8	1
REQ-011	Notifikasi	17	15	2
REQ-012	*Digital Nudge*	6	2	4
REQ-013	Sistem rekomendasi	2	0	2
REQ-014	*Dashboard* admin, desa, inovator, menteri	72	47	25
REQ-015	Kategori, *card*, dan kontak	27	22	5
Total		**312**	**261**	**51**

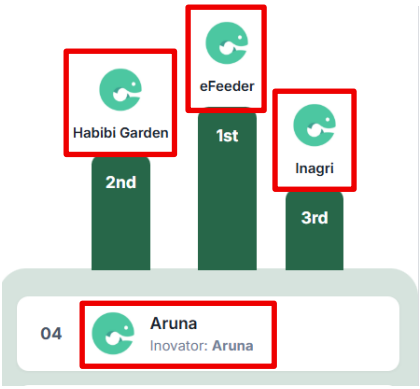
Dari total 312 skenario yang diuji, sebanyak 261 skenario berhasil dijalankan, sedangkan 51 skenario mengalami kegagalan dengan tingkat keberhasilan sebesar 84%. Berdasarkan pendekatan pengujiannya, ditemukan 11 *bug* pada *automation testing* dan 40 *bug* pada *manual testing*. Hasil tersebut menunjukkan bahwa kombinasi kedua pendekatan membantu menemukan *bug* secara lebih menyeluruh, terutama *bug* yang sulit dideteksi secara otomatis, seperti ketidaksesuaian data *leaderboard* yang ditampilkan pada skenario TC-AA-020 dan fitur rekomendasi yang masih menampilkan *dummy data* pada skenario TC-T-032. Seluruh skenario yang gagal kemudian dianalisis lebih lanjut untuk mengidentifikasi penyebab kesalahan serta menentukan prioritas perbaikan sistem. Ringkasan temuan *bug* yang mewakili masing-masing fitur utama sistem dapat dilihat pada Tabel 17, sementara dokumentasi *bug* lengkap dapat dilihat pada Lampiran 9.

Tabel 17 Temuan *bug* pengujian *black-box*

TC ID	*Scenario Testing*	*Error Type*	*Error Description*	*Severity*
TC-A-008	*As a user, I am unable to login with empty email and password fields*	*Functional*	*Error message* pada *input validation* yang ditampilkan belum sesuai	*Medium*
TC-T-032	*As a Village User, I am able to analyze ranking metrics for recommended innovations*	*Functional*	Rekomendasi yang ditampilkan berasal dari *dummy data*	*High*
TC-S-004	*As a user, I am able to view a supported village's details directly from an innovation's page*	*Boundary Related*	Jumlah desa yang ditampilkan pada detail inovasi belum dibatasi	*Medium*
TC-Z-006	*As an Admin User, I am prevented from adding an advertisement with empty fields*	*Functional*	*Form* masih bisa submit ketika input media kosong	*High*
TC-AA-007	*As an admin, I can view edited resubmission notifications from users*	*Communication Error*	Notifikasi edit *submission* tidak masuk ke admin	*Medium*
TC-AA-020	*As a user, I can see the correct leaderboard rankings*	*Logical*	*Leaderboard* inovator belum menampilkan data yang benar	*High*

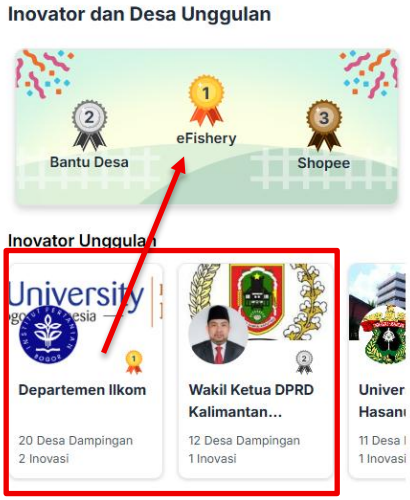
4.3.4 Analisis Hasil Pengujian

Berdasarkan hasil pengujian yang telah dilakukan, beberapa skenario menunjukkan kegagalan dengan *severity high*, Temuan ini harus diprioritaskan dalam proses perbaikan sistem agar dapat mengurangi dampak signifikan yang bisa disebabkan terhadap pengalaman pengguna. Salah satu contoh temuan dengan severity *high* terdapat pada fitur rekomendasi inovasi, ketika perangkat desa mencoba mendapatkan rekomendasi berdasarkan hasil klaim, data yang ditampilkan masih *dummy* dan tidak menggambarkan sistem rekomendasi yang seharusnya. Seharusnya sistem terhubung dengan *endpoint* eksternal untuk melakukan rekomendasi, sehingga hasilnya bisa akurat menampilkan rekomendasi berdasarkan profil desa dan inovasi yang sudah diklaim. Gambar 13 merupakan tampilan sistem rekomendasi dengan data *dummy*.



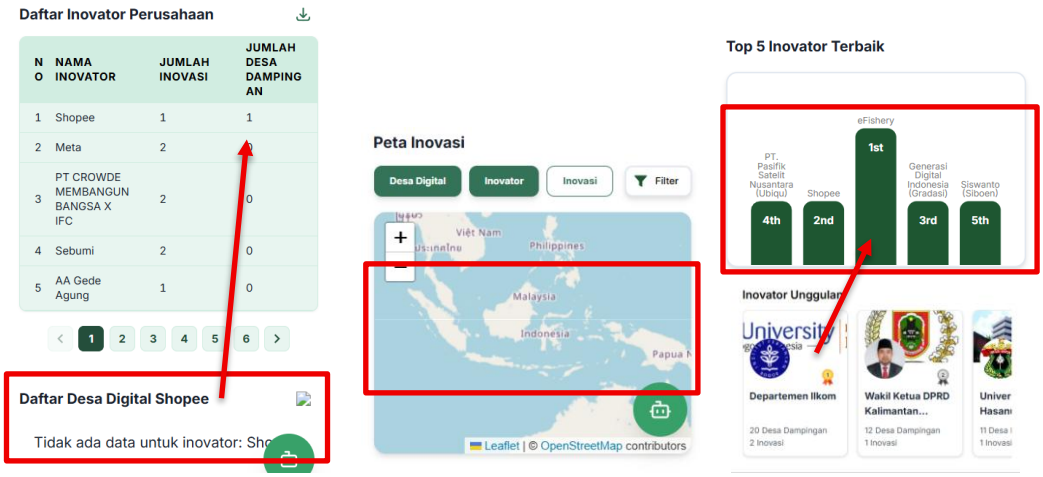
Gambar 13 Tampilan data *dummy* dan belum sinkron pada sistem rekomendasi

Temuan kedua terdapat pada *leaderboard* inovator unggulan di halaman utama, bagian tersebut seharusnya sinkron dengan data inovator unggulan yang ditampilkan di bagian bawahnya, namun yang terjadi justru data tidak sinkron dan menampilkan data yang tidak akurat. Gambar 14 memperlihatkan tidak sinkronnya inovator unggulan pada halaman utama.



Gambar 14 Ketidaksinkronan data *leaderboard* unggulan pada halaman utama

Temuan lainnya didominasi pada *dashboard* admin, inovator, dan kementerian, yaitu 25 dari total 51 *bug* yang teridentifikasi. Secara umum, permasalahan berpusat pada tiga pola utama, pertama berupa kegagalan menampilkan data relasional antar desa, inovasi, atau inovator yang berimplikasi pada tabel maupun *file* unduhan yang kosong, kedua berupa visualisasi peta yang tidak menampilkan lokasi akibat data koordinat yang tidak tersedia pada *record* yang ditampilkan, lalu yang ketiga berupa ketidaksesuaian hasil perhitungan pada grafik maupun tabel ringkasan, seperti *ranking* yang tidak menampilkan data sebenarnya serta *filter* yang tidak berfungsi. Pola kegagalan yang berulang tersebut mengindikasikan bahwa akar masalah berasal dari *query* atau logika pengambilan data yang berasal dari *endpoint* sama, bukan kesalahan tampilan yang berdiri sendiri pada masing-masing komponen. Gambar 15 menampilkan contoh temuan pada fitur *dashboard*.



(a) (b) (c)

Gambar 15 Temuan (a) data relasional desa tidak ditemukan, (b) titik peta tidak tampil, dan (c) *leaderboard* tidak sinkron dengan halaman utama

Selain temuan tersebut, temuan lainnya akan dibuatkan rekomendasi perbaikannya pada bagian *test closure*. Hal ini bertujuan untuk mempermudah proses perbaikan sistem dengan panduan rekomendasi berdasarkan hasil pengujian yang didapat.

4.4 Pengujian *White-Box*

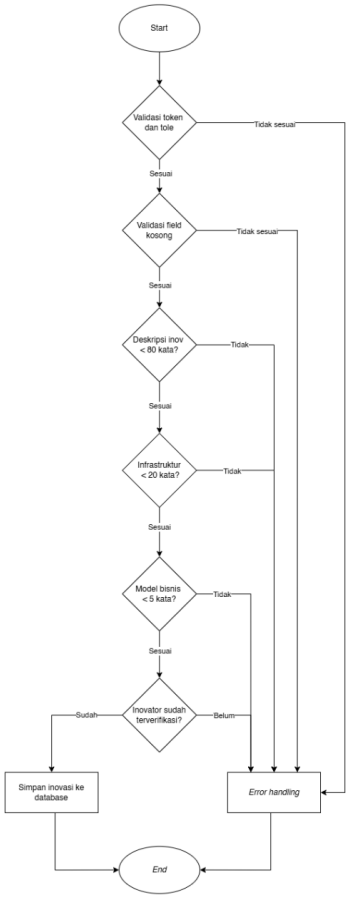
4.4.1 Pembuatan *Flow Graph*

Pada pengujian *white-box*, pengembangan *test case* didasarkan pada *independent path* yang ditentukan berdasarkan *flow graph* dan *cyclomatic complexity* dari logika program pada delapan fitur yang merupakan alur bisnis utama. Fitur tersebut terdiri dari *Login*, *Register*, Pengenalan autentikasi, Registrasi profil desa, Registrasi profil inovator, Tambah inovasi, Klaim inovasi, dan Verifikasi admin. Sebagai contoh, Tabel 18 merupakan kode program pada fitur tambah inovasi beserta penentuan nodenya.

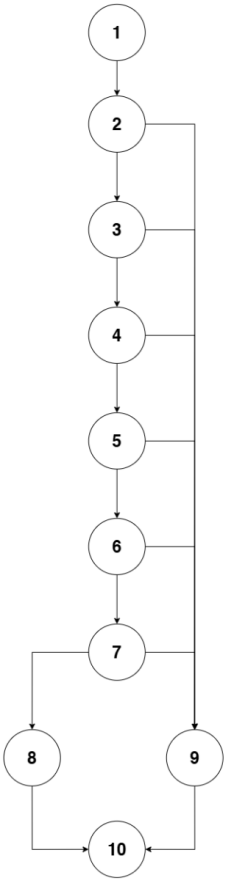
Tabel 18 Kode program fitur tambah inovasi dan penentuan node

Node	Kode Program	Fungsi
1	`export async function POST(req) { ... }`	*Start*
2	`if (auth instanceof NextResponse)`	Pengecekan token dan *role*
3	`if (!Array.isArray(...)`	Pengecekan *field* kosong
4	`if (validateWordLimit(deskripsi) > 80)`	Pengecekan batas deskripsi
5	`If(validateWordLimit(inputDesaMenerapkan) > 20)`	Pengecekan batas kata infrasturktur
6	`if (modelBisnis.some(word > 5))`	Pengecekan model bisnis
7	`if (!innovatorProfile`	Pengencekan inovator terverifikasi
8	`await collection.insertOne(...)`	Simpan inovasi
9	`catch (error)`	*Error handling* (API)
10	`}`	*End*

Berdasarkan analisis tersebut, dibuat *flowchart* dan *flowgraph* yang merepresentasikan alur logika program. Gambar 16 merupakan *flow chart* dan *flow graph* pada fitur tambah inovasi, sedangkan penentuan node, *flow chart,* dan *flow graph* untuk fitur lainnya dapat dilihat pada Lampiran 4.



(a)



(b)

Gambar 16 a) *Flow chart* dan b) *Flow graph* pada fitur tambah inovasi

4.4.2 Perhitungan *Cyclomatic Complexity*

Setelah *flow chart* dan *flow graph* sudah didefinisikan akan dihitung *cylomatic complexity* untuk menentukan *independent path* yang dihasilkan, dengan perhitungan sebagai berikut:

V(G) = 15 – 10 + 2

V(G) = 7

Berdasarkan hasil perhitungan, diperoleh nilai 7, sehingga terdapat 7 *independent path* yang harus diuji. Perhitungan *cyclomatic complexity* untuk fitur lainnya dapat dilihat pada Lampiran 5.

4.4.3 Penentuan *Independent Path*

Berdasarkan hasil perhitungan *cyclomatic complexity*, diperoleh 5 *independent path* pada fitur tambah inovasi, yaitu:

Path 1 = 1—2—9—10

Path 2 = 1—2—3—9—10

Path 3 = 1—2—3—4—9—10

Path 4 = 1—2—3—4—5—9—10

Path 5 = 1—2—3—4—5—6—9—10

Path 6 = 1—2—3—4—5—6—7—9—10

Path 7 = 1—2—3—4—5—6—7—8—10

Independent path ini dijadikan sebagai dasar dalam pembuatan *test case*, perhitungan lengkap untuk fitur lainnya dapat dilihat pada Lampiran 5. Skenario pengujian yang dihasilkan pada fitur tambah inovasi dapat dilihat pada Tabel 19.

Tabel 19 Skenario pengujian *white-box* pada fitur tambah inovasi

Independent Path	Skenario Pengujian
1-2-9-10	*As an unauthorized user, I cannot create an innovation.*
1-2-3-9-10	*As an innovator, I cannot create an innovation with missing text fields.*
1-2-3-4-9-10	*As an innovator, I cannot create an innovation with a description exceeding 80 words.*
1-2-3-4-5-9-10	*As an innovator, I cannot create an innovation with village names exceeding 20 words.*
1-2-3-4-5-6-9-10	*As an innovator, I cannot create an innovation if a business model item exceeds five words.*
1-2-3-4-5-6-7-9-10	*As an innovator, I cannot create an innovation if my profile is not verified.*
1-2-3-4-5-6-7-8-10	*As a verified innovator, I want to successfully create a new innovation.*

4.4.4 Pengembangan *Test Case*

Berdasarkan *independent path* yang dihasilkan, diperoleh 53 skenario pengujian yang terdiri dari 41 skenario negatif dan 12 skenario positif. Tabel 20 merupakan rincian jumlah skenario setiap fitur, sedangkan *test case* dapat dilihat pada Lampiran 6.

Tabel 20 Rincian jumlah skenario pengujian *white-box*

ID	Fitur	Positif	Negatif	Total Skenario
1	*Login*	3	3	6
2	*Register*	2	5	7
3	Pengenalan autentikasi	1	4	5
4	Profil Inovator	1	6	7
5	Profil Desa	1	6	7
6	Tambah Inovasi	1	6	7
7	Klaim Inovasi	1	8	9
8	Verifikasi Admin	2	3	5
	Total	**12**	**41**	**53**

4.4.5 Pelaksanaan Pengujian

Pengujian dilakukan menggunakan *script* berbasis *framework* Jest dengan metode *mocking* pada depedensi eksternal seperti koneksi MongoDB, autentkasi Firebase, dan mekanisme *cache*, sehingga setiap *endpoint* dapat diuji secara terisolasi berdasarkan setiap *independent path*. Jika ditemukan *error* pada pengujian, Postman akan digunakan dalam membantu visualisasi *bug* atau *error* yang ditemukan. Tabel 21 menunjukkan hasil pengujian *white-box* pada masing-masing fitur, dan Gambar 17 menampilkan contoh *script* pengujian *white-box* untuk *path* kedua pada fitur tambah inovasi (1-2-3-9-10), sedangkan *script* lengkap berbasis Jest dapat dilihat pada tautan *repository* di Lampiran 7.

Tabel 21 Hasil pengujian *white-box*

ID	Fitur	Total Skenario	Berhasil	Gagal
1	*Login*	6	6	0
2	*Register*	7	7	0
3	Pengenalan autentikasi	5	5	0
4	Profil Inovator	7	6	1
5	Profil Desa	7	6	1
6	Tambah Inovasi	7	6	1
7	Klaim Inovasi	9	8	1
8	Verifikasi Admin	5	4	1
	Total	**53**	**48**	**5**

```
test('Path 2 (Error): Salah satu field wajib kosong', async () =>
{
    (requireRole as jest.Mock).mockResolvedValueOnce({ uid:
    'user123', role: 'innovator' });

    // Missing 'kategori'
    const req = createRequest({
      namaInovasi: 'Inovasi A',
      tahunDibuat: '2023',
      deskripsi: 'Deskripsi Inovasi A',
      innovatorId: 'innovator123',
```

```
      statusInovasi: 'Aktif',
      modelBisnis: ['B2B'],
      inputDesaMenerapkan: 'Desa A',
      manfaat: ['Efisiensi'],
      infrastruktur: ['Server'],
    });

    const res = await POST(req);
    const json = await res.json();

    expect(res.status).toBe(400);
    expect(json.message).toMatch(/Field wajib tidak lengkap/i);
  });
```

Gambar 17 *Script* pengujian berbasis Jest

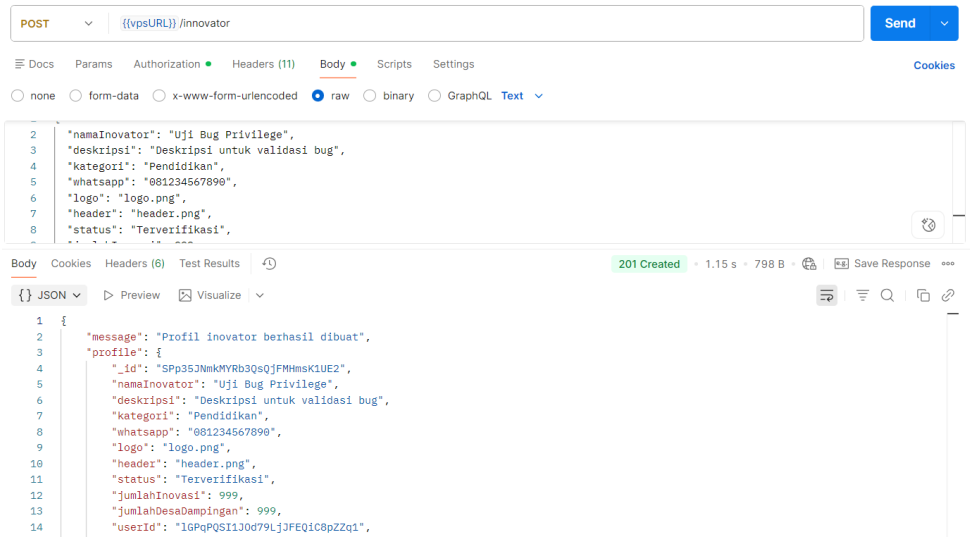
Dari total 53 skenario yang diuji, sebanyak 48 skenario berhasil dijalankan, sedangkan lima skenario mengalami kegagalan dengan tingkat keberhasilan sebesar 91%. Seluruh skenario yang gagal kemudian dianalisis lebih lanjut untuk mengidentifikasi penyebab kesalahan serta menentukan prioritas perbaikan sistem, *bug* yang ditemukan pada pengujian dapat dilihat pada Tabel 22.

Tabel 22 Temuan *bug* pengujian *white-box*

TC ID	*Scenario Testing*	*Error Type*	*Error Description*	*Severity*
KLM-001	*As a Village user, I am able to submit an innovation claim with incomplete video evidence and the system still accepts it.*	*Logical*	Validasi bukti video menggunakan operator .every(), sehingga bukti valid dan kosong masih tetap lolos.	*Medium*
INV-001	*As an Innovator, I am able to submit a new innovation using another innovator's ID and have their name & logo attached to my submission.*	*Security*	Tidak ada validasi innovatorId di *body*, sehingga innovator lain bisa menambah inovasi dan terjadi IDOR.	*High*
IPR-001	*As a newly registering Innovator, I am able to set my own profile status to 'Terverifikasi' and statistic counters to any value, bypassing admin review.*	*Security*	Beberapa *field* seperti status, jumlahInovasi, desaDampingan bisa diubah langsung oleh *request body* tanpa cek *role*.	*Critical*
VPR-001	*As a Village user, I am able to force the _id of my newly created village profile document to an arbitrary value.*	*Security*	*Mass assignment* bisa terjadi dengan identitas ID tidak divalidasi, sehingga nilai bisa dipalsukan.	*High*
VER-001	*As an Admin, submitting a claim verification request with an empty or unrecognized status value results in the claim being auto-approved.*	*Security*	Status yang dimasukkan selain ditolak akan menjadi Terverifikasi, termasuk body kosong	*Critical*

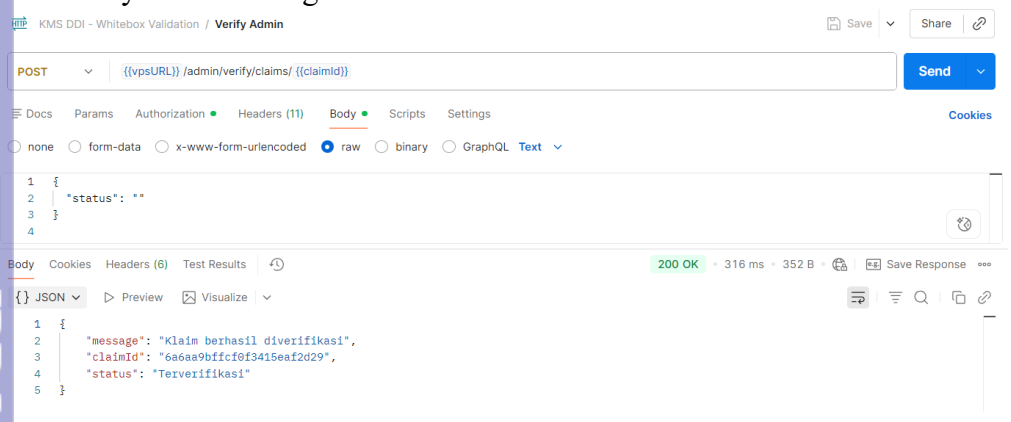
4.4.6 Analisis Hasil Pengujian

Berdasarkan hasil pengujian *white-box*, ditemukan beberapa celah dengan tingkat *severity critical* yang harus menjadi prioritas utama dalam penentuan perbaikan sistem. Temuan pertama dengan *severity critical* terdapat pada *endpoint* pembuatan profil, di mana *field* status, jumlahInovasi, dan jumlahDesaDampingan diambil langsung dari *request body* tanpa ada pemeriksaan *role* pengguna yang mengirimkannya. Hal ini berdampak jika seorang inovator yang baru pertama kali mendaftar menyertakan status "Terverifikasi" pada pembuatan profilnya, nilai tersebut akan langsung tersimpan tanpa peninjauan dari admin. Gambar 18 merupakan temuan yang memperlihatkan profil langsung berstatus "Terverifikasi".



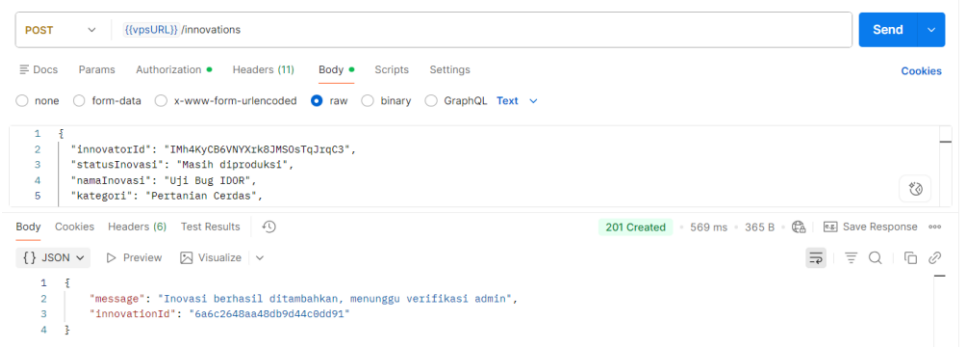
Gambar 18 Manipulasi status profil pada inovator baru

Temuan lainnya dengan *severity critical* terdapat pada *endpoint* verifikasi admin, baik untuk klaim, inovasi, profil inovator, maupun profil desa. Sistem seharusnya hanya menyetujui pengajuan jika admin mengirimkan nilai "Terverifikasi" secara valid, namun yang terjadi justru sistem mnganggap nilai apapun selain "Ditolak" termasuk *request* tanpa *body* sebagai status "Terverifikasi". Gambar 19 merupakan contoh pengujian tanpa *field* status yang seharusnya ditolak dengan kode 400.



Gambar 19 Validasi input lemah pada verifikasi Admin

Temuan ketiga dengan *severity high* terdapat pada *endpoint* penambahan inovasi, ketika seorang innovator yang sudah terverifikasi mengirimkan *innovatorId* milik pengguna lain, sistem tidak melakukan pengecekan terhadap kepemilikan identitas tersebut dan tetap menyimpan dengan nama serta logo innovator lain pada inovasi yang diajukan. Sistem seharusnya melakukan validasi bahwa *innovatorId* benar-benar merupakan identitas milik pengguna yang sedang terautentikasi. Gambar 20 merupakan celah yang ditemukan pada pengujian tambah inovasi.



Gambar 20 Celah pemalsuan *innovatorId* pada tambah inovasi

4.5 Pengujian Performa

4.5.1 Pengembangan *Test Case*

Tahapan ini berfokus pada perancangan format *test case* untuk pengujian performa. Struktur *test case* memuat informasi seperti *endpoint*, *method*, *ramp-up*, *threshold*, dan kolom-kolom untuk hasil yang akan dicatat. *Test case* disusun berdasarkan variasi jumlah *virtual users* sebesar 1, 10, 20, 50, 100, 250, dan 500 pengguna untuk mensimulasikan peningkatan beban secara bertahap dari kondisi normal hingga ekstrem. Seluruh *test case* yang akan diuji dapat dilihat pada Lampiran 8.

4.5.2 Penentuan *Endpoint* dan *Threshold*

Berdasarkan hasil analisis, terdapat 13 *endpoint* kritis yang dipilih untuk pengujian, yaitu:

a) GET /api/innovator untuk mengakses data innovator
b) GET /api/villages/[id] untuk mengakses detail desa
c) POST /api/innovations untuk menambahkan inovasi baru
d) POST /api/villages/claim untuk melakukan klaim inovasi oleh desa
e) POST /api/admin/verify/claims/[id] untuk verifikasi klaim oleh admin
f) POST /api/admin/verify/innovation/[id] untuk verifikasi inovasi oleh admin
g) PUT /api/villages/claim/[id] untuk mengubah data klaim
h) DELETE /api/innovations/[id] untuk menghapus data inovasi.
i) GET /api/innovations?search={keyword}untuk pencarian inovasi
j) GET /api/villages?search={keyword} untuk pencarian desa
k) GET /api/auth/me untuk mengenali pengguna setelah *login*
l) GET /api/ministry/dashboard untuk mengakses *dashboard* kementerian
m) GET /api/admin/dashboard untuk mengakses *dashboard* admin

Nilai *threshold* ditetapkan berdasarkan kompleksitas proses bisnis pada masing-masing *endpoint* dengan mengacu pada waktu *respons* ideal (*threshold* T) dan batas toleransi (*threshold* 4T) yang digunakan dalam evaluasi Apdex, selain itu disesuaikan juga dengan mempertimbangkan prinsip *usability* atau *response time limit* oleh Nielsen (1993). Daftar *endpoint* beserta nilai *threshold* yang digunakan dapat dilihat pada Tabel 23.

Tabel 23 *Endpoint* dan *threshold* pengujian performa

ID	*Endpoint*	*Method*	*Satisfied*	*Tolerating*
1	/api/innovator	GET	≤ 0,5	≤ 2
2	/api/villages/[id]	GET	≤ 0,5	≤ 2
3	/api/innovations	POST	≤ 0,8	≤ 3,2
4	/api/villages/claim	POST	≤ 1	≤ 4
5	/api/admin/verify/claims/[id]	POST	≤ 0,8	≤ 3,2
6	/api/admin/verify/innovation/[id]	POST	≤ 0,8	≤ 3,2
7	/api/villages/claim/[id]	PUT	≤ 0,8	≤ 3,2
8	/api/innovations/[id]	DELETE	≤ 0,5	≤ 2
9	/api/innovations?search={keyword}	GET	≤ 0,5	≤ 2
10	/api/villages?search={keyword}	GET	≤ 0,5	≤ 2
11	/api/auth/me	GET	≤ 0,7	≤ 2,8
12	/api/ministry/dashboard	GET	≤ 0,8	≤ 3,2
13	/api/admin/dashboard	GET	≤ 0,8	≤ 3,2

4.5.3 Pelaksanaan Pengujian Performa

Pengujian performa dilakukan dengan *tools* Apache JMeter untuk menjalankan setiap *endpoint* dengan variasi *virtual users* dan *ramp-up* dari 1-500 pengguna yang meningkat setiap detik. Berikut merupakan tahapan pelaksanaan pengujian performa:

a) Menyiapkan CSV *data set config*

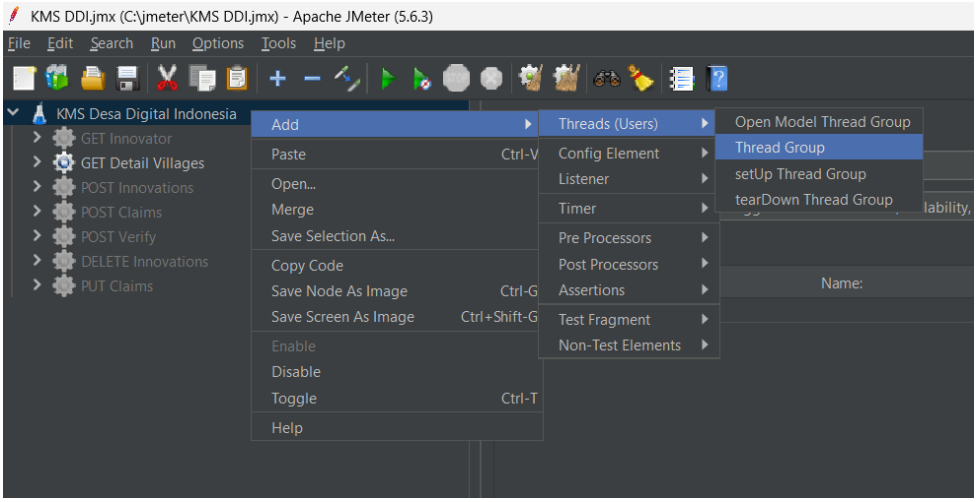
Tahap pertama yang dilakukan adalah menyiapkan CSV *data set config* sebagai sumber input dinamis untuk setiap *endpoint* yang diuji. Berkas ini memuat data yang dibutuhkan selama pengujian, seperti *request body* dan token autentikasi untuk *endpoint* POST, PUT, dan DELETE. Contoh *data set config* untuk POST *innovations* dapat dilihat pada Tabel 24.

Tabel 24 Contoh CSV *data set config* POST *Innovations*

Desa	Status	Nama	Kategori	Deskripsi
Citorek Barat	Masih diproduksi	SmartGov Desa	E-Government	Sistem digital untuk layanan administrasi desa
Citorek Sabrang	Tidak diproduksi	Desa Wisata App	E-Tourism	Platform promosi wisata desa berbasis digital
Sirnaresmi	Masih diproduksi	Infrastruktur IoT Desa	Infrastruktur Lokal	Sistem monitoring infrastruktur desa berbasis IoT
Puntang	Tidak diproduksi	Fintech Desa	Layanan Keuangan	Layanan keuangan digital untuk masyarakat desa

b) Membuat *thread group*

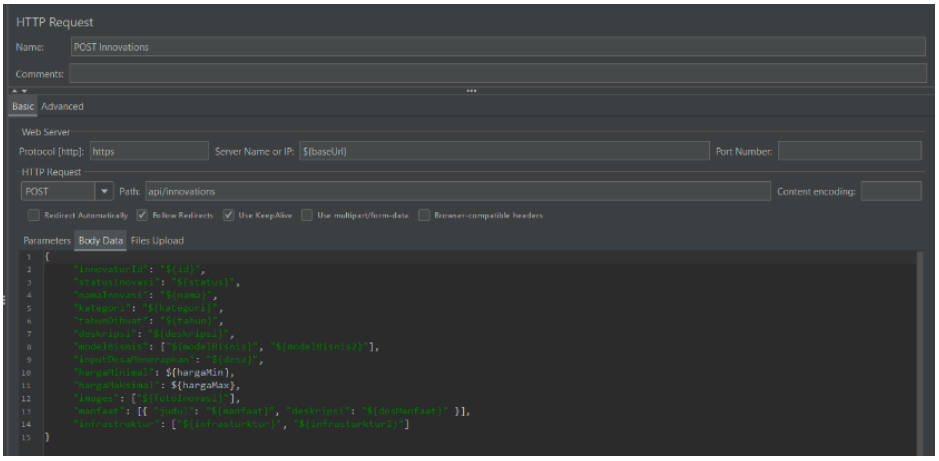
Tahap selanjutnya adalah membuat *thread group* untuk mensimulasikan beban pengguna sesuai skenario pengujian. Gambar 21 menunjukkan proses konfigurasi *thread group* pada Apache JMeter. Pada tahap ini, beberapa parameter utama ditentukan, yaitu jumlah *virtual users* dan *ramp-up period* dengan rasio 1 : 1 terhadap jumlah pengguna, serta *duration* selama 300 detik untuk mempertahankan koneksi tetap aktif sesaat setelah beban maksimum tercapai.



Gambar 21 Membuat *thread group*

c) Menambahkan HTTP *request*

Pada *thread group*, ditambahkan HTTP *request* sebagaimana ditunjukkan pada Gambar 22. Konfigurasi yang digunakan meliputi *protocol* HTTPS, *server name* ${baseUrl} yang mengarah ke domain digidesa.tech, *method*, serta *path* berupa *endpoint* yang digunakan. Bagian *body data* diisi dengan parameter input yang dibutuhkan oleh *endpoint*, setiap nilai variabel diambil secara dinamis dari CSV *data set config* yang telah disiapkan sebelumnya.

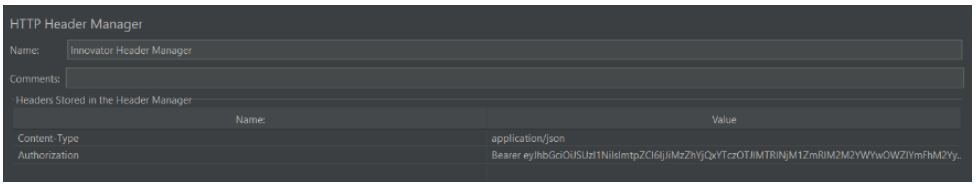


Gambar 22 Menambahkan HTTP *request*

d) Menambahkan HTTP *header manager*

Untuk *request* yang memerlukan autentikasi, seperti pada *method* POST, PUT, dan DELETE, ditambahkan *HTTP header manager* untuk menyertakan

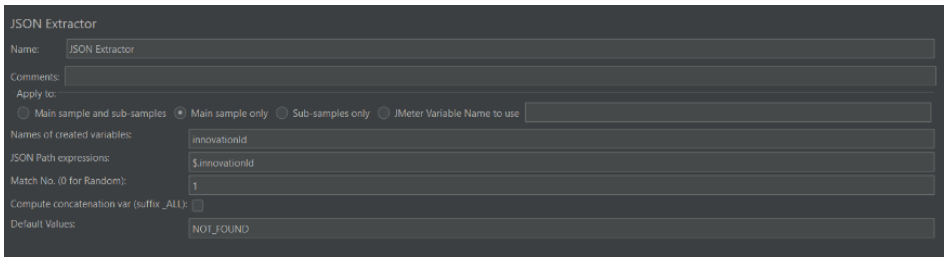
token maupun *header* lain yang diperlukan. Pada sistem KMS DDI, mekanisme autentikasi menggunakan *bearer token* yang dikirim melalui header *authorization*. Token yang digunakan pada pengujian merupakan token dengan *role* admin, sehingga dapat digunakan untuk mengakses seluruh *endpoint* yang diuji dalam sistem.



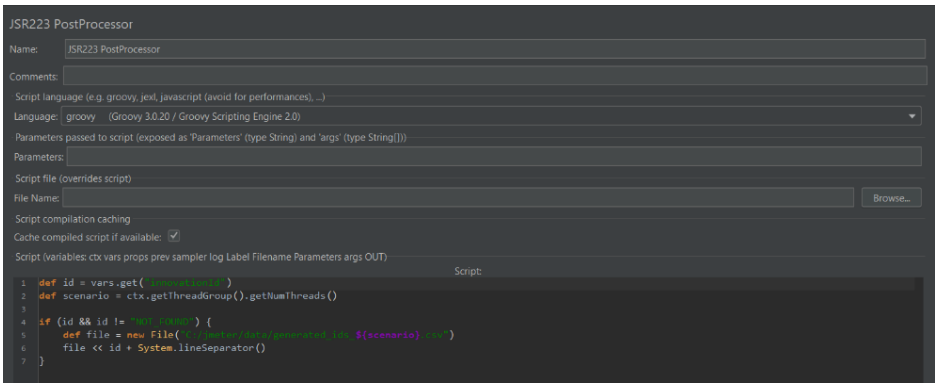
Gambar 23 Menambahkan HTTP *header manager*

e) Menambahkan JSON *Extractor* dan JSR223 *PostProcessor*

Pada *endpoint* yang menghasilkan ID baru, seperti POST *innovations* dan POST *claims*, ditambahkan *JSON extractor* untuk menangkap nilai ID dari respon yang dihasilkan. ID tersebut kemudian digunakan kembali pada skenario lanjutan, seperti verifikasi admin, proses edit, maupun *delete*, sehingga alur pengujian dapat berjalan secara berkesinambungan. Setelah ID berhasil diekstraksi, nilai tersebut dituliskan ke dalam berkas CSV menggunakan *JSR223 postprocessor*, sehingga data hasil *request* dapat digunakan kembali pada pengujian berikutnya.



Gambar 24 Menambahkan JSON *Extractor*



Gambar 25 Menambahkan JR223 *Post Processor*

f) Menjalankan pengujian

Untuk menjalankan pengujian secara otomatis dan berurutan, disiapkan skrip otomatisasi yang dijalankan melalui mode *non-GUI* Apache JMeter pada *PowerShell*. Setiap skenario pengujian, dengan variasi *virtual users* dan *ramp-up* dieksekusi secara berurutan sesuai daftar yang disimpan pada

variabel $tests. Nilai *duration* pada setiap skenario disesuaikan dengan beban yang dijalankan dan ditambahkan 300 detik untuk mempertahankan kondisi sistem setelah mencapai jumlah *request* maksimum. Setelah proses selesai, sistem secara otomatis menghasilkan *dashboard report* dalam bentuk index.html, nilai Apdex, berkas hasil (results.jtl), serta log pengujian ke dalam direktori yang telah disesuaikan dengan *endpoint* dan skenario yang dijalankan. *Script* otomatisasi dan alur eksekusi pengujian ditunjukkan pada Gambar 26.

Gambar 26 *Script* dan Proses menjalankan pengujian

Secara keseluruhan, hasil pengujian menunjukkan bahwa sistem mampu menangani berbagai tingkat beban pengguna dengan cukup stabil, meskipun terdapat variasi karakteristik respons pada beberapa *endpoint*. Mayoritas *endpoint* menunjukkan tingkat keberhasilan yang tinggi dengan *error rate* 0% dan nilai Apdex yang secara dominan berada pada kategori *Good* hingga *Excellent* saat sistem mulai menerima beban yang stabil (10 VU hingga 500 VU). Namun, pada skenario pengujian awal dengan satu VU, banyak *endpoint* memiliki nilai Apdex *Poor* karena waktu respons yang melebihi batas *threshold*. Tabel 25 merupakan hasil pengujian yang sudah dipetakan berdasarkan standar Apdex. Hasil lengkap pengujian performa pada setiap *endpoint* dapat dilihat pada Lampiran 8.

Tabel 25 Hasil pengujian berdasarkan standar Apdex

ID	*Endpoint*	1	10	20	50	100	250	500
1	GET /api/innovator							
2	GET /api/villages/[id]							
3	POST /api/innovations							
4	POST /api/villages/claim							
5	POST /api/admin/verify/claims/[id]							
6	POST /api/admin/verify/innovation/[id]							
7	PUT /api/villages/claim/[id]							
8	DELETE /api/innovations/[id]							
9	GET /api/innovations?search={keyword}							
10	GET /api/villages?search={keyword}							
11	GET /api/auth/me							
12	GET /api/ministry/dashboard							
13	GET /api/admin/dashboard							

Klasifikasi warna pada tabel mengacu pada standar Apdex, yaitu biru (*Excellent*), hijau (*Good*), kuning (*Fair*), merah (*Poor*), dan abu-abu (*Unacceptable*).

4.5.4 Analisis Hasil Pengujian Performa

Sebagai temuan positif pada pengujian performa, hasil yang didapat sudah menunjukkan kualitas sistem yang optimal dan memuaskan, khususnya pada *endpoint* pembacaaan data *(read)*. Pada pengujian ini *endpoint* seperti GET /api/innovator, GET /api/villages[id], dan pencarian (?search={keyword}) mampu mempertahankan waktu respons yang sangat rendah pada rentang 100-200 ms dengan nilai Apdex *Excellent* hingga beban 500 VU. Hal ini didasari dari pemanfaatan Nginx sebagai *web server* dan *load balancer* yang mengatur lalu lintas *request* yang tinggi. Selain itu sistem ini juga menerapkan *caching* pada sebagian *endpoint* GET sehingga dalam melayani *request* langsung dari VPS tanpa membebani *query* ke *database* eksternal, sehingga dapat memberikan respons instan kepada pengguna.

Pada sisi lain, terdapat juga fenomena tingginya *response time* pada *request* pertama pada *endpoint write* data seperti POST, PUT, dan DELETE. Sebagai contoh pada POST /api/innovations mencapai 1287 ms yang tidak ideal karena sistem berjalan di VPS yang seharusnya *alwoys-on*, fenomena ini diindikasikan bukan merupakan *cold start* dari *environment server*. Lonjakan waktu ini diduga dipengaruhi oleh arsitektur sistem yang menggunakan layanan *database* eksternal, sehingga pada *request* pertama server harus membentuk jalur koneksi aman yang melintasi jaringan publik ke layanan eksternal tersebut. Namun setelah koneksi awal terbentuk, *response time* pada *request* berikutnya menurun drastis dan stabil di kategori *Excellent* karena dapat menggunakan ulang jalur koneksi yang sudah dibuat. Selain itu, pada *endpoint* verifikasi admin yaitu POST /api/admin/verify/claims dijalankan bersamaan dengan POST /api/admin/verify/innovation dengan proses klaim dijalankan lebih awal. Hal ini berimplikasi pada nilai latensi awal yang tinggi (1003 ms) akibat inisialisasi koneksi jaringan sebelum kinerjanya juga membaik pada *request* berikutnya.

Temuan terakhir terdapat pada *endpoint* GET /api/ministry/dashboard yang mencatatkan *response time* lebih lambat dibanding *endpoint* GET lainnya. Pada awal pengujian ini, *endpoint* ini mencatatkan waktu respons sangat lambat mencapai 2570 ms dengan kategori *Poor*, bahkan ketika beban dinaikkan menjadi 10 VU dan 20 VU, skor Apdex masih berada pada kategori *Fair* dan *Good,* dengan *response time* berkisar antara 780 ms – 1000 ms. Tingginya angka ini memperlihatkan proses *fetching* dan agregasi data yang kompleks, dengan pertukaran data yang cukup besar. Hal ini juga disebabkan karena pada *endpoint* ini belum menerapkan *cache* seperti *endpoint* GET lainnya, sehingga setiap koneksi melintasi jaringan publik ke layanan eksternal, walaupun pada skenario 50 VU koneksi akhirnya mampu stabil dengan kategori Apdex *Excellent*.

4.6 *Test Closure*

Seluruh aktivitas pengujian yang direncanakan pada penelitian ini telah dilaksanakan sesuai dengan tahapan STLC. Pengujian mencakup aspek fungsional melalui *black-box*, aspek struktural sistem melalui *white-box*, dan aspek non-fungsional melalui pengujian performa. Berdasarkan hasil pengujian, sebagian besar skenario pengujian berhasil dijalankan dengan tingkat keberhasilan yang tinggi sehingga sistem dinilai telah memenuhi kebutuhan fungsional yang

ditetapkan. Tabel 26 menampilkan ringkasan hasil pengujian sebelum dan sesudah dilakukan perbaikan.

Tabel 26 Ringkasan hasil pengujian KMS DDI

Pengujian	Hasil Pengujian	Success Rate	Status Pengujian
Black-box	261 *Passed* 61 *Failed*	84%	*Passed*
White-box	48 *Passed* 5 *Failed*	91%	*Passed*
Performa	1 *error* dari 12.103 *request*	99.99%	*Passed*

Pada pengujian *black-box, bug* yang ditemukan didominasi oleh *functional errors,* sedangkan berdasarkan tingkat keparahan mayoritas *bug* berada pada kategori *high* dan *medium severity*. Pada pengujian *white-box*, *bug* yang ditemukan berasal dari kategori *security* dan *logical errors*, dengan empat temuan berkategori *security* dan satu temuan berkategori *logical*, berdasarkan tingkat keparahannya, dua diantaranya tergolong *critical*, dua tergolong *high*, dan satu tergolong *medium*. Penggabungan hasil kedua jenis pengujian ini menghasilkan total 56 *bug*, dengan kedua *severity critical* yang ditemukan tidak menyebabkan kegagalan sistem secara menyeluruh seperti *crash* total, melainkan berpotensi disalahgunakan untuk memanipulasi data maupun melewati proses verifikasi apabila tidak segera ditangani. Ringkasan sebaran *bug* berdasarkan *error type* dan *severity* dapat dilihat pada Tabel 27 dan Tabel 28.

Tabel 27 Sebaran *bug* berdasarkan *error type*

Error Type	Jumlah	Persentase (%)
Functional	23	41,07%
Usability	7	12,50%
Runtime Error	6	10,71%
Logical	5	8,93%
Security	4	7,14%
Integration Level	3	5,36%
Boundary Related	3	5,36%
Calculation Error	3	5,36%
Out Of Bound	1	1,79%
Communication Error	1	1,79%
Total	**56**	**100%**

Tabel 28 Sebaran *bug* berdasarkan *severity*

Severity	Jumlah	Persentase (%)
Critical	2	3,57%
High	27	48,21%
Medium	25	44,64%
Low	2	3,52%
Total	**56**	**100%**

Berdasarkan *bug* yang ditemukan, dilakukan analisis berdasar hasil pengujian. Pada pengujian *black-box*, analisis difokuskan pada identifikasi *bug* yang memengaruhi fungsi sistem, termasuk penyebab dan dampaknya terhadap proses

bisnis. Pada pengujian *white-box*, analisis dilakukan terhadap kesalahan logika program, percabangan, atau jalur eksekusi yang tidak berjalan sesuai yang diharapkan. Sementara itu, pada pengujian performa analisis difokuskan pada *endpoint* yang mengalami keterlambatan atau *error*, termasuk *bottleneck* dan respons yang melampaui *threshold*. Tabel 29 merupakan analisis hasil perbaikan sebagai landasan untuk proses perbaikan *bug* dan *error*.

Tabel 29 Analisis hasil pengujian sistem

Jenis Pengujian	Temuan	Solusi
Black-box	Kurang informatif *error message* pada validasi input *form* autentikasi.	Menambahkan validasi input dari sisi *frontend* yang informatif.
	Data *leaderboard,* inovasi, dan rekomendasi belum sinkron *database.*	Memastikan proses *fetching* dan sinkronisasi data dilakukan secara *real time.*
	Beberapa fitur mengalami kesalahan navigasi dan *redirect*	Memperbaiki *routing* agar pengguna diarahkan ke halaman yang sesuai setelah suatu aksi dilakukan.
	Sistem *badge* untuk gamifikasi belum diimplementasikan dengan baik.	Melakukan pengembangan fitur *badge* untuk kebutuhan *digital nudge*.
	Komponen *preview* dan *toast* belum disesuaikan dengan kebutuhan sistem.	Melakukan pengembangan komponen *preview* dan memperbaiki ukuran *toast*.
	Mekanisme notifikasi ke admin belum berjalan secara konsisten.	Menambahkan *trigger* notifikasi setelah proses *resubmission*.
	Visualisasi *dashboard* tidak merepresentasikan data yang akurat.	Memperbaiki sinkronisasi data lokasi, perhitungan, dan mekanisme filter.
	Data relasional antar entitas pada *dashboard* gagal ditampilkan.	Memperbaiki logika *query* dan menambahkan tampilan saat data tidak ditemukan.
White-box	Validasi bukti video pengajuan klaim tidak konsistem dengan bukti lainnya, sehingga data kosong dapat lolos.	Mengganti operator .every() menjadi .some() pada validasi bukti video agar konsisten dengan jenis bukti lainnya.
	Endpoint tambah inovasi tidak melakukan validasi kepemilikan innovatorId.	Membatasi pengambilan data inovator hanya dari profil pengguna yang sudah diverifikasi.
	Pembuatan profil innovator memperbolehkan *field* status, jumlahInovasi, dan jumlahDesa disii melalui *request body*.	Menerapkan nilai *default* (status: "Menunggu", jumlah: 0) untuk *request* selain Admin.
	Field _id pada profil desa dapat ditimpa oleh nilai lain pada *request body*.	Membiarkan mekanisme *default* untuk *field* _id dan menghapus *field* tersebut sebelum disimpan ke *database*.
	Pada verifikasi admin, nilai apapun selain "Ditolak", termasuk *body* kosong otomatis dianggap sebagai "Terverifikasi".	Menambahkan validasi terhadap nilai status dan nilainya harus "Terverifikasi" atau "Ditolak", selainnya mengembalikkan status 400.

Tabel 29 Analisis hasil pengujian sistem (*lanjutan*)

Jenis Pengujian	Temuan	Solusi
Performance	Latensi pada inisialisasi awal yang tinggi pada *endpoint* POST, PUT, dan DELETE.	Menerapkan *warm instance pool* yang mempertahankan koneksi secara berkelanjutan. (Lin dan Gilkson 2019)
	Terdapat satu *error* dengan pesan "*Connection reset*" (0,40%) pada *endpoint dashboard* kementerian	Menyesuaikan maxIdleTimeMS dan socketTimeoutMS, disertai mekanisme *retry* untuk kegagalan koneksi sesaat (Mongo 2024).
	Waktu respons yang lambat akibat besarnya beban kalkulasi kueri pada proses agregasi *dashboard* kementerian.	Menerapkan *materialized view* terjadwal terhadap hasil agregasi dan dikombinasikan dengan *caching* (Privalov dan Stupina 2023).

Selanjutnya dilakukan evaluasi dengan membandingkan *exit criteria* yang telah ditentukan pada masing-masing pengujian dengan hasil aktual yang diperoleh selama proses pengujian. Apabila seluruh *exit criteria* telah terpenuhi, maka pengujian dapat dinyatakan selesai dan dianggap telah memenuhi standar kualitas yang sudah ditetapkan. Hasil evaluasi perbandingan *exit criteria* dan hasil aktual ditampilkan pada Tabel 30.

Tabel 30 Evaluasi pencapaian *exit criteria*

Pengujian	Exit Criteria	Hasil Aktual	Status
Black-box	Seluruh *test case* (312 skenario) sudah dieksekusi	312 *test case* berhasil dieksekusi.	Tercapai
	Tingkat keberhasilan pengujian (*success rate*) minimal 80%	84% *success rate*.	Tercapai
	Bug yang ditemukan selesai diklasifikasikan	Semua *bug* yang ditemukan selesai diklasifikasikan.	Tercapai
White-box	Seluruh *independent path* telah dilewati minimal satu kali	Seluruh *independent path* yang dijadikan dasar *test case* berhasil dieksekusi.	Tercapai
	Nilai *cyclomatic complexity* selesai dihitung	Semua nilai *cyclomatic complexity* berhasil dihitung.	Tercapai
	Bug yang ditemukan dibuat saran perbaikan	Rekomendasi perbaikan selesai disusun	Tercapai
Performa	Melakukan simulasi beban pada 13 *endpoint* yang sudah dipilih	13 *endpoint* yang ditetapkan berhasil diuji	Tercapai
	Hasil yang didapatkan seperti *response time*, *error rate, throughput* selesai direkap	Seluruh metrik pengujian yang ditetapkan berhasil direkap.	Tercapai
	Apdex *score* sudah ditentukan berdasarkan hasil pengujian	Apdex *score* untuk tiap *endpoint* berhasil dihitung.	Tercapai

Berdasarkan Tabel 30, *seluruh exit criteria* pada ketiga jenis pengujian berhasil tercapai, yang menandakan sistem sudah melalui verifikasi menyeluruh baik dari

sisi fungsional, strktural, maupun performa. Pada pengujian *black-box,* seluruh 312 *test case* berhasil dieksekusi dengan *success rate* sebesar 84%, seluruh *bug* yang ditemukan telah diklasifikasikan berdasarkan tingkat keparahannya. Pada pengujian *white-box*, seluruh *independent path* yang dihasilkan dari perhitungan *cyclomatic complexity* pada tiap fitur berhasil menemukan sejumlah celah pada *source code* yang diujikan, seluruhnya telah diklasifikasikan dan dibuatkan rekomendasi perbaikan. Pada pengujian performa, simulasi beban berhasil dilakukan pada 13 *endpoint* terpilih, dengan seluruh metrik yang dibutuhkan berhasil didapat dan dihitung nilai Apdex-nya dalam melihat tingkat kepuasan performa pada masing-masing *endpoint*. Dengan terpenuhinya seluruh *exit criteria* yang telah ditetapkan, proses pengujian dapat dinyatakan selesai dan sistem dinilai siap untuk digunakan maupun dikembangkan lebih lanjut pada tahap berikutnya.

V SIMPULAN DAN SARAN

5.1 Simpulan

Pengujian pada sistem KMS DDI berhasil dilakukan menggunakan metode *Software Testing Life Cycle* (STLC) melalui pengujian *black-box*, *white-box*, dan performa untuk mengevaluasi kualitas sistem dari aspek fungsional dan non-fungsional. Berikut merupakan simpulan yang didapat dari hasil penelitian:

a. Pengujian *black-box* melalui kombinasi *manual* dan *automation testing* dengan Katalon Studio menghasilkan *success rate* sebesar 84% dari 312 skenario pengujian dengan temuan 51 *bug* yang didominasi oleh *functional error.*
b. Pengujian *white-box* menggunakan teknik *basis path testing* dengan Jest menghasilkan *success rate* 91% dengan temuan lima kegagalan dari 53 skenario, temuan didominasi oleh *security error*.
c. Pengujian performa, secara umum berhasil menangani variasi jumlah pengguna dari pengujian yang dilakukan, yaitu 1 hingga 500 pengguna, dengan rata-rata Apdex level *Excellent* dan hanya satu *error* dari 12.103 *request* dengan pesan "*Connection reset*". Temuan yang didapat adalah masih terdapat inisialisasi awal pada *endpoint write* dan waktu respons lambat pada *dashboard* kementerian.
d. Hasil pengujian yang mencakup *bug* pada pengujian *black-box*, *white-box,* dan temuan *error* pengujian performa sudah dirumuskan rekomendasi perbaikannya, sehingga memudahkan dalam proses perbaikan yang akan dilakukan.

Pengujian dan rumusan perbaikan yang dilakukan menjadi dasar yang kuat untuk memastikan sistem siap digunakan dan sudah terjamin baik kualitas maupun stabilitasnya untuk digunakan oleh semua jenis pengguna.

5.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, pengujian *white-box* masih dapat dikembangkan dengan memperluas cakupan pengujian pada lebih banyak fitur sehingga validasi jalur logika program dapat dilakukan secara lebih menyeluruh. Pada pengujian *black-box* juga dapat dikembangkan menjadi *fully automated testing* agar proses pengujian dapat dilakukan secara lebih cepat dan efisien ketika sistem mengalami perubahan atau penambahan fitur baru dengan *tools* lain seperti Cypress atau Playwright yang memiliki fitur lebih lengkap. Pengujian performa dapat dilakukan dengan teknik lainnya seperti *stress testing, spike testing,* atau *scalability testing.* Selain itu pengujian non-fungsional lainnya seperti *security testing, usability testing,* atau *user acceptance testing* (UAT) dapat dilakukan untuk memverifikasi kesiapan sistem dari segi keamanan, kemudahan, dan kesesuaian dengan kebutuhan serta ekspektasi pengguna.

DAFTAR PUSTAKA

Abdillah MI. 2025. Pengujian *black-box, white-box*, dan *performance* pada *road asset management system* (studi kasus: Jalan Tol Bakauheni-Terbanggi Besar) [skripsi]. Bogor: IPB University.

Ali A, Maghawry H, Badr N. 2022. Automation of performance testing: a review. *Int J Intell Comput Inf Sci*. 22:35–50. doi:10.21608/ijicis.2022.161846.1219.

Arfan A, Hendrik. 2022. Penerapan stlc dalam pengujian automation aplikasi mobile (studi kasus: lms amikom center pt.git solution). *Automata*. 3(2).

Ariyadi IT. 2025. Integrasi firebase pada modul front-end (studi kasus kms inovasi desa digital) [skripsi]. Bogor: IPB University.

Awalia R. 2024. Pengembangan modul back-end menggunakan low/no-code platform jojonomic officeless operating system [skripsi]. Bogor: IPB University.

Bhagat D. 2016. Effective testing techniques. *International Journal of Multidisciplinary Research and Development.* 3(2):21–23.

Costa V, Girardon G, Bernardino M, Machado R, Legramante G, Neto A, Basso FP, de Macedo Rodrigues E. 2020. Taxonomy of performance testing tools. Di dalam: *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. New York, NY, USA: ACM. hlm 1997–2004.

Dalkir K. 2023. Knowledge management in theory and practice (4th edition). Ed ke-4. Cambridge, Massachusetts: The MIT Press.

Dewi AK, Azmina HY, Sugiarti Y. 2024. Penggunaan metrik software defect dalam pengembangan perangkat lunak: literature review. *Acad J Comput Sci Res*. 6(2):89–97. doi:10.38101/ajcsr.v6i2.15649.

Ehmer M, Khan F. 2012. A comparative study of white box, black box and grey box testing techniques. *Int J Adv Comput Sci Appl*. 3(6). doi:10.14569/IJACSA.2012.030603.

Ferdiansah MAS. 2025. Pengembangan sistem rekomendasi menggunakan content-based filtering pada kms desa digital [skripsi]. Bogor: IPB University.

Hutauruk DH. 2025. Upaya peningkatan partisipasi desa melalui implementasi digital nudge interaktif pada kms desa digital indonesia [tesis]. Bogor: IPB University.

IEEE. (2009). IEEE standard classification for software anomalies (ieee std 1044-2009). *New York: Institute of Electrical and Electronics Engineers (IEEE).*

Jampani R, Talasu N, Manjula R. 2016. Survey of software testing techniques. *International Journal for Research in Applied Science & Engineering Technology (IJRASET)*. 4(4):24-29.

Jorgensen PC. 2013. Software testing: a craftsman's approach 4th edition. Boca Raton, US.

Kusum, Talwar P, Puri A, Kumar G. 2024. Overview of software testing. *Glob J Eng Technol Adv*. 19(1):104–112. doi:10.30574/gjeta.2024.19.1.0060.

Laudon KC, Laudon JP. 2022. Managing information system: managing the digital firm. Edisi ke-17. New Jersey (USA): Pearson Education, Inc.

Lin PM, Glikson A. 2019. Mitigating cold starts in serverless platforms: a pool-based approach. arXiv preprint arXiv:1903.12221. Tersedia pada: https://arxiv.org/abs/1903.12221.

MongoDB Inc. 2024. Manage connections with connection pools Node.js driver. [diakses 2026 Jul 30]. Tersedia pada: https://www.mongodb.com/docs/drivers/node/current/connect/connection-options/connection-pools/.

Nielsen J. 1993. Response times: the 3 important limits. [diakses 2025 Des 2]. Tersedia pada: https://www.nngroup.com/articles/response-times-3-important-limits/

Nurhadryani Y, Mardiana R. Asfarian A, Ardiansyah F, Hermadi I, Herlina M. Nindyasari Y. 2022. Mengenal ekosistem desa digital. Bogor: IPB Press.

Nurmaulydia AZ. 2025. Pengembangan *dashboard* interaktif pada *knowledge management system* inovasi desa digital [skripsi]. Bogor: IPB University.

Nuryantika F. 2023. Pengembangan modul front-end mobile web inovasi desa digital dengan metode prototyping [skripsi]. Bogor: IPB University.

Pahlevi SA. 2024. Otomatisasi proses pengujian perangkat lunak dengan metode behavior driven development [skripsi]. Malang: Universitas Islam Negeri Maulana Malik Ibrahim.

Privalov MV, Stupina MV. 2023. Improving web-oriented information systems efficiency using Redis caching mechanisms. *Indonesian Journal of Electrical Engineering and Computer Science.* 33(3):1667-1675. doi:10.11591/ijeecs.v33.i3.pp1667-1675.

Ragunath PK, Velmourougan S. Davachelvan P, Kayal S, Ravimohan R. 2010. Evolving a new model (sdlc model-2010) for software development life cycle (sdlc). *IJCSNS*. doi: 10(1):112-119.

Rakly BD, Andriyani W. 2025. Software testing in e-commerce: a comparison between manual and automated testing using katalon studio and python. *Journal of Artificial Intelligence and Software Engineering*. 5(1):156-163. doi:10.30811/jaise.v5i1.6448.

Rana A, Rawat AS, Bijalwan A. 2017. Process of finding defects in software testing. 10(2)297–300. Di dalam: *Proceedings of the Second International Conference on Research in Intelligent and Computing in Engineering (RICE 2017)*. ACSIS. 10:297–300. doi:10.15439/2017R18.

Ruliansyah, Tukino, Baenil Huda, April Lia Hananto. 2023. Penerapan software testing life cycle pada pengujian otomatisasi platform dzikra. *CSRID (Computer Sci Res Its Dev Journal)*. 15(1):01–11. doi:10.22303/csrid.15.1.2023.01-11.

Samudrala LNR. 2022. Application performance index (apdex): calculation and usefulness in real user monitoring. *International Journal of Innovative Research and Creative Technology.* 8(3):1–6.

Semarandhanu LGM, Yulhendri. 2024. Penerapan knowledge management system: tacit knowledge, explicit knowledge, dan sharing knowledge (studi kasus: lembaga kursus dan pelatihan pendidikan non formal “giri putri” di kota singaraja - bali). *Jurnal Multidisiplin Saintek*. 2(3):80–89.

Santi IH. 2026. Software Testing Essentials: Panduan Validasi dan Verifikasi Sistem untuk Skripsi Rekayasa Perangkat Lunak. Pekalongan (ID): Penerbit NEM.

Sevcik P. 2005. Defining the application performance index. *Business Communications Review*. 20(8).

Solissa YJ, Putra F, Putri AN, Nursari SRC. 2023. Pengujian white box berbasis path pada form daftar jobstreet.co.id. *Konvergensi Teknologi dan Sistem Informasi*. 3(2):353-362. doi:10.24002/konstelasi.v3i2.8287.

Uddin A, Anand A. 2019. Importance of software testing in the process of software development. *IJSRD*. 6(12):141–145.

Uhl T, Rompf M. 2015. New tool for investigating qos in the www service. *Journal of Telecommunications and Information Technology*. 1:3–9.

Tasnim H. 2023. Perancangan pengalaman pengguna sistem manajemen pengetahuan inovasi desa digital [skripsi]. Bogor: IPB University.

Yousuf MM, Sofi SA. 2025. Bug classification in quantum software: a rule-based framework and its evaluation. arXiv preprint arXiv:2506.10397. Tersedia pada: https://doi.org/10.48550/arXiv.2506.10397.

Yusrina S. 2025. Pengembangan dashboard monitoring untuk admin dan perangkat desa pada knowledge management system desa digital indonesia [skripsi]. Bogor: IPB University.