AITI: Jurnal Teknologi Informasi, Volume 23 No. 2 Juni 2026, 290-304
ISSN 1693-8348 E-ISSN 2615-7128
DOI: 10.24246/aiti.v23.i2.290-304

JURNAL TEKNOLOGI INFORMASI

# Rancang bangun sistem *FlowTrackr*: Aplikasi manajemen *test case* dan *bug report* berbasis *web*

**Desril Fatra Aditama*, Imelda**
Magister Ilmu Komputer, Universitas Budi Luhur
Jl. Ciledug Raya No. 99, RT.10/RW.2, Petukangan Utara, Kec. Pesanggrahan,
Kota Jakarta Selatan, DKI Jakarta 12260
Email: *2411601046@student.budiluhur.ac.id

| Riwayat Artikel | | |
|---|---|---|
| Diterima: | Direvisi: | Disetujui: |
| 16-07-2025 | 29-09-2025 | 15-10-2025 |

***Abstract***

*Software testing is an essential stage to ensure application quality before deployment. Manual testing practices by Quality Assurance (QA) teams often face challenges, such as scattered test results and bug reports across multiple media, making them difficult to trace and prone to miscommunication. This study designs FlowTrackr, a web-based integrated application for centralized test case management and bug reporting. The development followed the Waterfall methodology, covering requirements analysis, system design, implementation, and testing. Functional testing across 10 scenarios confirmed that all features operated according to specifications, while User Acceptance Testing (UAT) involving 5 QA testers demonstrated that the system was easy to use and more efficient than manual methods. Compared with tools such as Jira, TestRail, and Bugzilla, which tend to be more complex, FlowTrackr offers a lightweight, practical solution for small teams. Therefore, FlowTrackr can serve as an alternative to support manual QA processes, making them more structured and well-documented.*

***Keywords:*** *software testing, quality assurance, bug report, test case management, web application*

**Abstrak**

Pengujian perangkat lunak merupakan tahap penting dalam siklus pengembangan sistem untuk memastikan kualitas, keandalan, dan kesesuaian aplikasi dengan kebutuhan pengguna sebelum digunakan. Pada praktiknya, proses pengujian manual oleh tim Quality Assurance (QA) sering menghadapi berbagai kendala, seperti pencatatan hasil uji dan pelaporan bug yang tersebar di berbagai media, sehingga menyulitkan proses pelacakan dan dokumentasi, serta meningkatkan risiko miskomunikasi antaranggota tim. Penelitian ini merancang FlowTrackr, sebuah aplikasi web terintegrasi untuk manajemen test case dan pelaporan bug secara terpusat. Pengembangan dilakukan dengan metode Waterfall melalui analisis kebutuhan, perancangan, implementasi, dan pengujian. Hasil pengujian fungsional pada sepuluh skenario menunjukkan bahwa seluruh fitur berjalan sesuai spesifikasi, sementara User Acceptance Testing (UAT) dengan lima QA memperlihatkan sistem yang mudah digunakan dan lebih efisien dibandingkan dengan metode manual. Dibandingkan dengan Jira, TestRail, atau Bugzilla yang lebih kompleks, FlowTrackr menawarkan solusi yang ringan dan praktis untuk tim kecil. Dengan demikian, FlowTrackr

dapat menjadi alternatif untuk mendukung proses QA manual agar lebih terstruktur dan terdokumentasi.

**Kata kunci:** pengujian perangkat lunak, *quality assurance*, *bug report*, manajemen *test case*, aplikasi *web*

## Pendahuluan

Pengujian perangkat lunak merupakan fase fundamental dalam siklus pengembangan sistem untuk memastikan bahwa aplikasi yang dikembangkan memenuhi standar kualitas, kestabilan, dan keandalan sebelum diterapkan pada lingkungan produksi [1]. Praktik pengujian manual masih menjadi pendekatan dominan dalam proses *Quality Assurance* (QA), khususnya di organisasi dengan sumber daya terbatas atau proses pengembangan yang bersifat dinamis dan *iteratif*. Meskipun metode ini relatif fleksibel dan mudah diterapkan, pengujian manual kerap menghadapi berbagai tantangan operasional. Hasil pengujian yang tersebar di berbagai dokumen, pelaporan *bug* yang tidak terstruktur, serta terbatasnya kolaborasi antara tim QA dan pengembang merupakan beberapa hambatan umum yang berkontribusi terhadap rendahnya efisiensi dan keterlacakan proses pengujian [2]. Dalam konteks pengembangan perangkat lunak yang semakin kompleks dan menuntut kecepatan iterasi tinggi, kendala tersebut dapat berdampak langsung pada keterlambatan rilis, peningkatan risiko kesalahan, dan penurunan kualitas produk akhir [3]. Oleh karena itu, diperlukan pendekatan sistematis yang mampu mengintegrasikan seluruh proses pengujian manual ke dalam sebuah sistem yang efisien, terdokumentasi, dan mudah digunakan oleh seluruh pemangku kepentingan dalam tim pengembang [4].

Dalam praktiknya, proses pengujian perangkat lunak secara manual masih sering dilakukan menggunakan alat bantu konvensional seperti *spreadsheet, email,* atau dokumen *text*, yang tidak dirancang khusus untuk mendukung kompleksitas manajemen pengujian [5]. Ketergantungan pada media yang tersebar ini menyulitkan pelacakan *test case*, identifikasi regresi *bug*, serta verifikasi hasil pengujian secara historis. Selain itu, pelaporan *bug* yang tidak memiliki struktur baku seringkali menyebabkan miskomunikasi antara tim QA dan pengembang, memperlambat proses *debugging* dan penanganan kesalahan [6]. Tidak adanya sistem terintegrasi untuk mencatat hasil uji, mengelola laporan *bug*, serta mendokumentasikan *evidence* secara sistematis menjadi salah satu penyebab utama inefisiensi dalam proses QA. Ketiadaan fitur-fitur *critical* seperti pengelompokan *test case*, pelacakan status uji, dan pelaporan otomatis memperburuk kondisi ini. Dengan demikian, terdapat kebutuhan mendesak untuk menghadirkan solusi teknologi yang mampu menjawab permasalahan tersebut secara menyeluruh dalam satu platform yang ringkas, efisien, dan mudah diimplementasikan oleh tim penguji perangkat lunak.

Penelitian ini bertujuan untuk merancang dan membangun *FlowTrackr*, sebuah aplikasi berbasis *web* yang terintegrasi untuk mendukung manajemen

pengujian perangkat lunak secara manual. Sistem ini dikembangkan guna mengatasi tantangan umum yang dihadapi oleh tim *Quality Assurance,* khususnya dalam hal pencatatan hasil uji yang tidak terpusat, pelaporan *bug* yang tidak terstruktur, serta kurangnya dokumentasi *evidence* yang terdigitalisasi [7],[8]. *FlowTrackr* dirancang untuk menyediakan fitur-fitur inti seperti pencatatan *test case* secara sistematis, pelaporan *bug* berbasis *form* terstandarisasi, unggahan bukti pengujian (*evidence*)*,* pengelolaan pengguna, serta kemampuan ekspor laporan uji secara otomatis. Melalui sistem ini, diharapkan proses pengujian manual menjadi lebih efisien, terdokumentasi dengan baik, dan mendukung kolaborasi yang lebih produktif antara tim QA dan pengembang. Selain itu, penelitian ini juga bertujuan untuk membuktikan bahwa solusi berbasis teknologi ringan namun terintegrasi dapat menjadi alternatif yang efektif bagi organisasi yang belum mengadopsi sistem QA berskala *enterprise*.

Berbagai penelitian dan pengembangan sistem dalam bidang *Quality Assurance* perangkat lunak selama ini umumnya terfokus pada otomasi pengujian, seperti pengembangan *automated testing frameworks* dan *continuous integration pipelines*. Sementara itu, studi yang secara khusus mengkaji dan menyediakan solusi untuk pengujian manual terutama dalam hal pencatatan *test case* dan pelaporan *bug*—masih relatif terbatas [9]. Baik dari sisi jumlah maupun kedalaman implementasinya. Di sisi lain, platform yang tersedia secara komersial seperti *Jira, TestRail,* atau *Bugzilla* memang menawarkan fitur manajemen QA, namun umumnya bersifat kompleks, memerlukan konfigurasi tingkat lanjut, serta tidak selalu cocok untuk kebutuhan tim kecil atau proyek dengan anggaran terbatas [10]. Selain itu, belum banyak sistem yang mengintegrasikan fitur pelaporan *bug,* dokumentasi *evidence,* manajemen *test case,* dan ekspor laporan dalam satu antarmuka berbasis *web* yang ringan dan mudah digunakan. Kondisi ini menunjukkan adanya celah dalam literatur dan praktik yang dapat diisi oleh penelitian ini, yakni dengan menghadirkan sistem alternatif yang mengakomodasi kebutuhan pengujian manual secara menyeluruh namun tetap praktis dan terjangkau dalam implementasi.

Penelitian ini menghadirkan *FlowTrackr* sebagai solusi baru yang mengintegrasikan seluruh proses pengujian manual mulai dari pencatatan *test case*, pelaporan *bug*, unggahan *evidence*, hingga ekspor laporan pengujian—ke dalam satu platform *web* yang ringan dan mudah diakses. Berbeda dari pendekatan sebelumnya yang cenderung fokus pada otomasi pengujian atau pengelolaan *bug* secara terpisah, *FlowTrackr* menekankan pada kebutuhan tim QA yang masih bergantung pada proses manual namun membutuhkan sistem yang terstruktur dan efisien. Kebaruan penelitian ini terletak pada kombinasi fungsionalitas yang ditawarkan secara terpadu dan dirancang khusus untuk menjawab tantangan nyata di lapangan, terutama bagi organisasi yang belum memiliki sistem QA berskala *enterprise*. Selain kontribusi praktis, penelitian ini juga memberikan nilai tambah secara akademik dengan menunjukkan bagaimana sistem sederhana namun tepat

guna dapat meningkatkan efisiensi, akurasi, dan kolaborasi dalam proses QA. Dengan demikian, *FlowTrackr* tidak hanya menawarkan inovasi pada tataran teknis, tetapi juga memperluas ruang diskusi tentang kebutuhan kontekstual dalam manajemen pengujian perangkat lunak.

Penelitian terkini juga menunjukkan tren menuju otomatisasi dalam proses *Quality Assurance*. Salah satu studi mengusulkan proses *automatic bug triaging* berbasis *Large Language Models* (LLM) untuk meningkatkan akurasi klasifikasi laporan kesalahan [11]. Pendekatan ini efektif untuk organisasi besar dengan kebutuhan otomasi tinggi, namun membutuhkan sumber daya komputasi dan integrasi yang kompleks sehingga kurang sesuai bagi tim kecil. Kondisi ini menegaskan adanya *gap* penelitian, yaitu perlunya sistem terintegrasi yang tetap mendukung pengujian manual secara sederhana, praktis, dan terjangkau. *FlowTrackr* dikembangkan untuk mengisi *gap* tersebut melalui platform *web* ringan yang berfokus pada manajemen *test case*, pelaporan *bug*, dan dokumentasi *evidence* dalam satu alur terpusat.

## Metode Penelitian

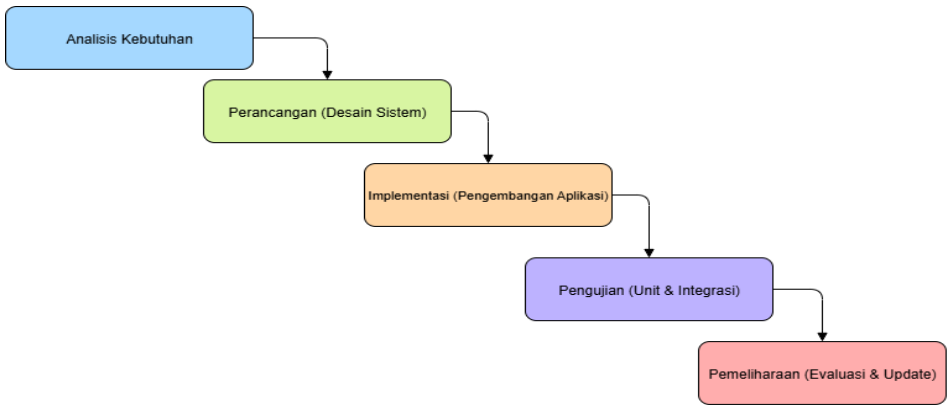
Metode penelitian yang digunakan dalam pengembangan sistem *FlowTrackr* adalah metode *Waterfall*, yang merupakan salah satu pendekatan rekayasa perangkat lunak bersifat sekuensial dan sistematis [12]. Metode ini dipilih karena cocok untuk proyek dengan kebutuhan yang jelas dan terdefinisi sejak awal [13],[14]. Tahapan dalam metode *Waterfall* meliputi:

1. Analisis Kebutuhan. Tahap ini bertujuan untuk mengumpulkan dan menganalisis kebutuhan pengguna terkait sistem yang akan dikembangkan. Kebutuhan diperoleh melalui observasi proses pengujian manual, wawancara dengan QA, dan studi pustaka. Hasil analisis kebutuhan dikelompokkan berdasarkan peran pengguna, yaitu *Admin* dan *User* sebagai berikut:
   - Kebutuhan *Admin*: membutuhkan fitur untuk mengelola *user* (menambah, mengubah, dan menghapus data pengguna), mengelola proyek (membuat, memperbarui, dan menghapus proyek pengujian), serta mengekspor laporan hasil pengujian.
   - Kebutuhan *User*: membutuhkan fitur untuk meng-*input test case* agar skenario pengujian terdokumentasi dengan baik, melaporkan *bug* dengan detail, kondisi saat *bug* terjadi, serta tingkat keparahan (*severity*). Selain itu, membutuhkan kemampuan untuk mengunggah *evidence* (misalnya *screenshot* atau *file* pendukung) sebagai bukti pengujian, serta melihat laporan hasil pengujian untuk memantau *progres* secara transparan.
2. Perancangan Sistem (Desain Sistem). Pada tahap ini dilakukan perancangan sistem meliputi desain antarmuka pengguna (*mockup*), *Use Case Diagram, Activity Diagram, Class Diagram,* dan *Sequence Diagram.* Tujuannya adalah untuk menggambarkan alur kerja, struktur data, dan interaksi antar komponen sistem.

3. Implementasi (Pengembangan Aplikasi). Sistem dikembangkan menggunakan teknologi berbasis *web* dengan *stack* berikut:
   - *Frontend: React.js* berbasis *HTML5, CSS3,* dan *JavaScript* digunakan untuk membangun antarmuka pengguna yang responsif, interaktif, serta mudah dipahami.
   - *Backend: Node.js* dengan *framework Express* dipilih untuk menangani logika bisnis, autentikasi, serta komunikasi antara *frontend* dan basis data secara efisien.
   - *Database: MySQL* digunakan sebagai sistem manajemen basis data relasional (*RDBMS*) untuk menyimpan data pengguna, proyek, *test case, bug report,* serta *evidence* secara terstruktur dan konsisten.

   Implementasi dilakukan berdasarkan rancangan yang telah disusun sebelumnya [15].
4. Pengujian (Unit dan Integrasi) dilakukan untuk memastikan bahwa sistem berjalan sesuai dengan kebutuhan. Pengujian dilakukan secara fungsional untuk setiap modul (*unit testing*) dan menyeluruh terhadap integrasi antar modul (*integration testing*).
5. Pemeliharaan (Evaluasi dan *Update*). Setelah sistem diuji dan diimplementasikan, dilakukan pemeliharaan untuk memperbaiki *bug*, menyesuaikan dengan masukan pengguna, dan memastikan performa tetap optimal.

Diagram alur metode *Waterfall* yang digunakan dalam penelitian ini ditunjukkan pada Gambar 1. Diagram tersebut menggambarkan tahapan proses pengembangan sistem *FlowTrackr* secara berurutan, sistematis, dan terstruktur.




**Gambar 1** Tahapan metode pengembangan model *waterfall*

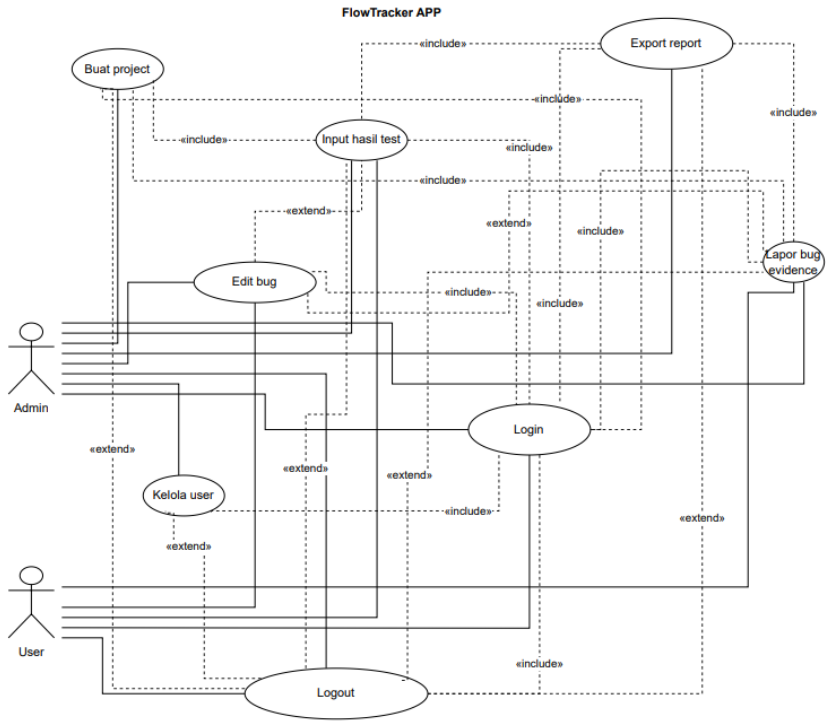
## Hasil dan Pembahasan

Sistem *FlowTrackr* yang dikembangkan dalam penelitian ini merupakan sebuah aplikasi berbasis *web* yang dirancang untuk memfasilitasi proses pengujian perangkat lunak secara manual, khususnya dalam konteks pencatatan hasil uji dan pelaporan *bug* oleh tim *Quality Assurance* (QA). Sistem ini dibangun untuk menjawab tantangan umum yang sering dihadapi dalam aktivitas QA konvensional,

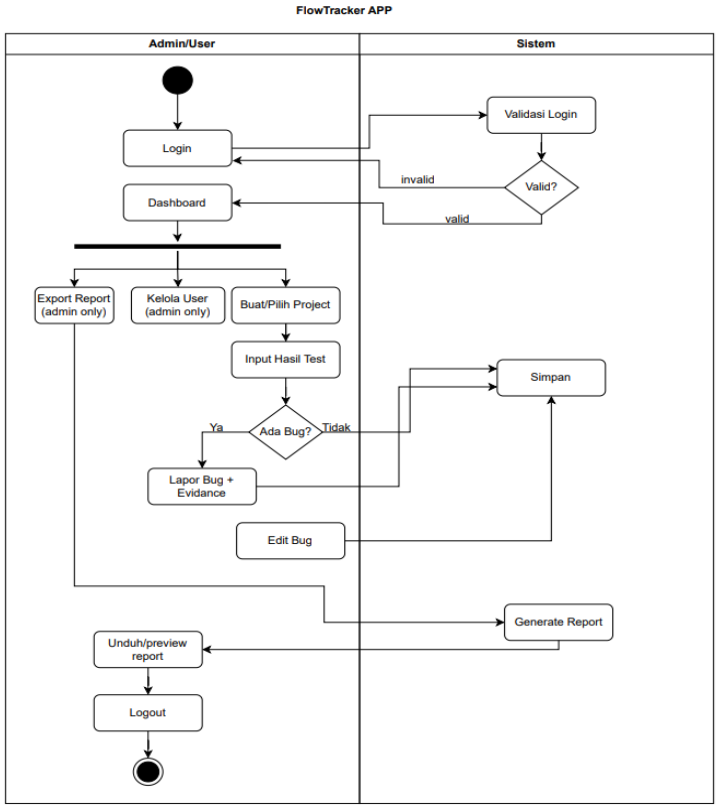
seperti pencatatan *test case* yang tersebar, pelaporan *bug* yang tidak terstruktur, serta minimnya integrasi antar aktivitas uji. *FlowTrackr* mengintegrasikan seluruh alur pengujian ke dalam satu platform terpadu, yang mencakup fitur pencatatan hasil uji, pelaporan *bug*, unggahan bukti pengujian (*evidence*), pengelolaan pengguna, serta ekspor laporan uji secara otomatis. Dengan pendekatan tersebut, *FlowTrackr* tidak hanya menyederhanakan alur kerja QA, tetapi juga meningkatkan keterlacakan (*traceability*) dan transparansi antar tim yang terlibat dalam pengembangan perangkat lunak. Sistem ini dikembangkan dengan mempertimbangkan kebutuhan praktis tim QA, sehingga fokus pada kemudahan penggunaan, efisiensi alur, dan kesesuaian dengan praktik pengujian manual yang umum digunakan di lapangan.

Untuk memastikan bahwa sistem memenuhi kebutuhan fungsional yang telah didefinisikan pada tahap analisis, dilakukan pemodelan sistem menggunakan pendekatan *Unified Modeling Language (UML)*, dimulai dengan penyusunan *Use Case Diagram*. Diagram ini digunakan untuk memvisualisasikan interaksi antara sistem dan pihak yang terlibat secara langsung dalam pengoperasiannya, yaitu para aktor. Dalam pengembangan *FlowTrackr*, terdapat dua aktor utama: *Admin* dan *User* (QA). Aktor *Admin* memiliki peran sebagai pengelola sistem secara keseluruhan, termasuk manajemen akun pengguna, pembuatan proyek, serta kontrol atas data pengujian yang tersedia. Sementara itu, aktor *User* (QA) memiliki tanggung jawab terhadap proses operasional pengujian, seperti mencatat hasil *test case*, melaporkan *bug*, mengunggah *evidence*, dan menghasilkan laporan uji. Beberapa *use case* penting yang dimodelkan meliputi *login*, pengelolaan proyek, pengisian data pengujian, pelaporan *bug*, pengelolaan bukti uji, dan ekspor hasil uji. Penyusunan *Use Case Diagram* ini berfungsi sebagai landasan konseptual untuk merancang sistem secara terarah, sekaligus memastikan bahwa seluruh kebutuhan fungsional dari masing-masing peran telah terakomodasi dengan baik dalam desain sistem. Diagram ini juga mendukung keterlacakan antara kebutuhan pengguna dan implementasi sistem, sehingga meminimalkan potensi kesenjangan spesifikasi dalam proses pengembangan.

Gambar 2 menampilkan *Use Case Diagram* dari sistem *FlowTrackr*, yang merepresentasikan hubungan antara dua aktor utama, yaitu *Admin* dan *User* (QA), dengan sejumlah fungsionalitas utama yang mereka akses. Diagram ini memberikan gambaran awal mengenai batasan sistem (*system boundary*), peran masing-masing aktor, serta interaksi mereka dengan fitur-fitur utama dalam platform, yang menjadi dasar dalam proses perancangan dan implementasi sistem secara keseluruhan.




**Gambar 2** *Use case diagram* sistem *FlowTrackr*




**Gambar 3** *Activity diagram* sistem *FlowTrackr*

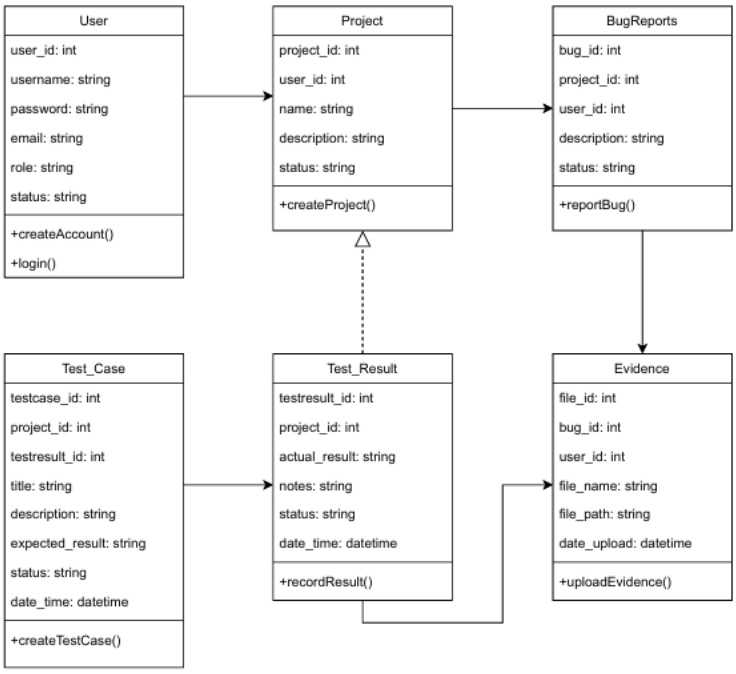
Setelah pemodelan kebutuhan fungsional melalui *Use Case Diagram*, dilakukan penyusunan *Activity Diagram* untuk menggambarkan alur aktivitas utama dalam sistem *FlowTrackr*. *Activity Diagram* ini menyajikan representasi

visual mengenai urutan proses yang dilakukan oleh aktor *Admin* maupun *User* (QA) dalam berinteraksi dengan sistem, mulai dari *login* hingga pengelolaan hasil uji dan pelaporan *bug*. Dengan adanya diagram ini, alur kerja menjadi lebih jelas, sehingga memudahkan dalam memahami bagaimana setiap *use case* dijalankan secara terstruktur. Diagram ini juga membantu dalam mengidentifikasi titik-titik penting interaksi antar aktor dan sistem, yang nantinya berperan sebagai acuan dalam tahap implementasi maupun pengujian sistem. Gambar 3 menampilkan *activity diagram* sistem *FlowTrackr*.

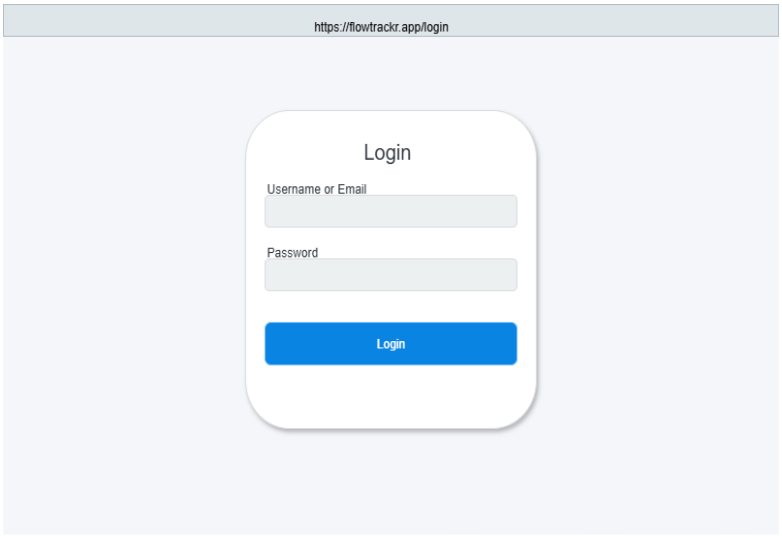
Sebagai bagian dari tahap perancangan sistem, dilakukan pemodelan struktur data menggunakan *Class Diagram* untuk merepresentasikan entitas inti dalam sistem *FlowTrackr* beserta relasi antarentitas tersebut. Diagram ini tidak hanya berfungsi sebagai dokumentasi arsitektur data, tetapi juga sebagai referensi penting dalam proses implementasi basis data dan logika program. Dalam konteks sistem ini, beberapa *class* utama yang diidentifikasi antara lain: *User*, *Project*, *TestCase*, *BugReport*, dan *Evidence*. *Class User* menyimpan informasi terkait akun pengguna, dengan atribut seperti *user_id*, *username*, *role*, dan *email*. *Class Project* berelasi dengan *User* (khususnya *Admin*) dan mencakup data seperti *project_id*, *project_name*, dan *created_date*. Sementara itu, *TestCase* dan *BugReport* merepresentasikan elemen utama dalam proses QA, masing-masing terhubung dengan *Project* serta memiliki atribut detail untuk mendeskripsikan skenario pengujian dan laporan kesalahan. *Class Evidence* digunakan untuk mendokumentasikan *file* bukti pengujian yang diunggah oleh pengguna, dan berelasi dengan *BugReport* untuk memperkuat konteks laporan *bug*. Relasi antar *class* dimodelkan dengan notasi asosiasi, agregasi, dan kardinalitas yang menggambarkan *dependensi* dan jumlah keterlibatan entitas. Dengan demikian, *Class Diagram* menjadi tulang punggung dalam pengembangan struktur data sistem, sekaligus menjamin konsistensi antara desain konseptual dan implementasi aktual pada lapisan basis data maupun *backend* aplikasi. Gambar 4 merupakan *class diagram* sistem *FlowTrackr*.

Desain antarmuka pengguna (*User Interface* atau UI) merupakan aspek penting dalam pengembangan sistem *FlowTrackr* karena berperan langsung dalam menunjang efisiensi dan kenyamanan pengguna dalam mengakses fitur-fitur yang disediakan. *Mockup* dirancang untuk menampilkan alur penggunaan yang sederhana, intuitif, dan konsisten sesuai dengan tugas harian tim QA. Antarmuka *login* menjadi titik masuk utama untuk otentikasi pengguna, memastikan keamanan dan pemisahan hak akses antara *Admin* dan QA. Setelah berhasil masuk, pengguna diarahkan ke halaman *dashboard* yang menyajikan ringkasan aktivitas dan navigasi menuju modul utama seperti pengelolaan proyek, input hasil *test case*, serta pelaporan *bug*. *Form* input *test case* didesain dengan struktur yang jelas untuk memudahkan pencatatan hasil pengujian berdasarkan skenario tertentu. Sementara itu, tampilan pelaporan *bug* menyediakan kolom deskriptif dan opsi unggahan

*evidence*, sehingga QA dapat menyampaikan laporan dengan lebih terstruktur dan informatif. Pemilihan elemen UI, seperti tombol aksi dan *form input* memperhatikan prinsip *user-centered design* dan *minimal cognitive load*, sehingga sistem dapat digunakan secara efektif oleh pengguna non-teknis sekalipun. Penyusunan *mockup* ini tidak hanya bertujuan sebagai alat bantu visual bagi pengembang, tetapi juga sebagai validasi awal terhadap kebutuhan pengguna yang diperoleh pada tahap analisis sistem. Gambar 5 menampilkan *mockup login* sistem *FlowTrackr*.



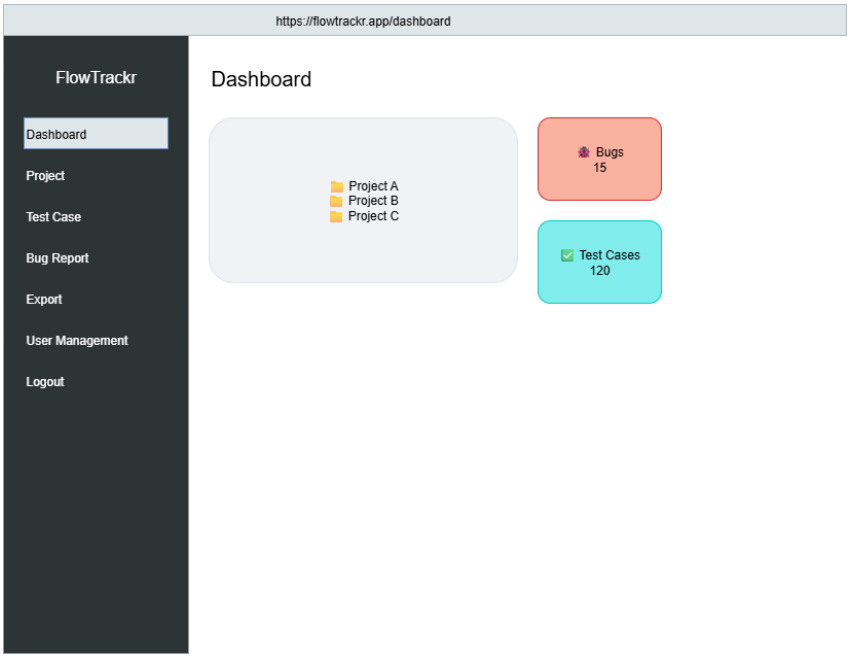

**Gambar 4** *Class diagram* sistem *FlowTrackr*




**Gambar 5** *Login* sistem *FlowTrackr*

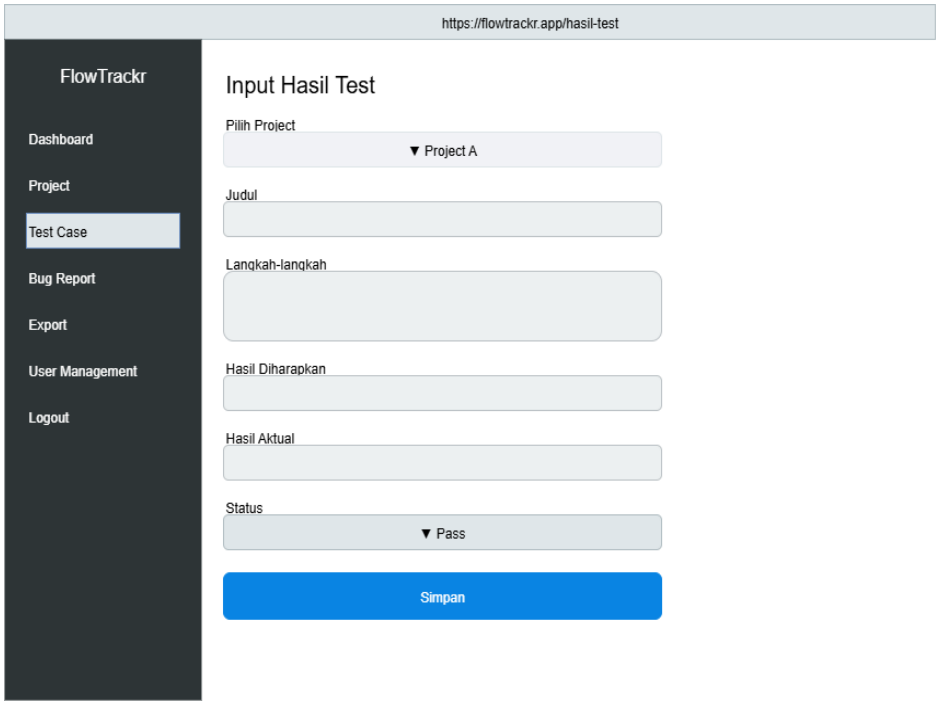
Halaman *dashboard* pada sistem *FlowTrackr* berfungsi sebagai pusat informasi dan navigasi utama bagi pengguna. Tampilan ini menyajikan ringkasan

proyek, jumlah *test case*, *bug* yang dilaporkan, serta status pengujian secara ringkas. Navigasi dirancang agar pengguna dapat dengan cepat mengakses fitur-fitur penting seperti input uji, pelaporan *bug*, dan ekspor laporan. Desain yang bersih dan terstruktur pada *dashboard* membantu meningkatkan efisiensi kerja serta mempercepat proses pengambilan keputusan dalam aktivitas QA. Gambar 6 menampilkan *mockup dashboard* sistem *FlowTrackr*.




**Gambar 6** *Dashboard* sistem *FlowTrackr*

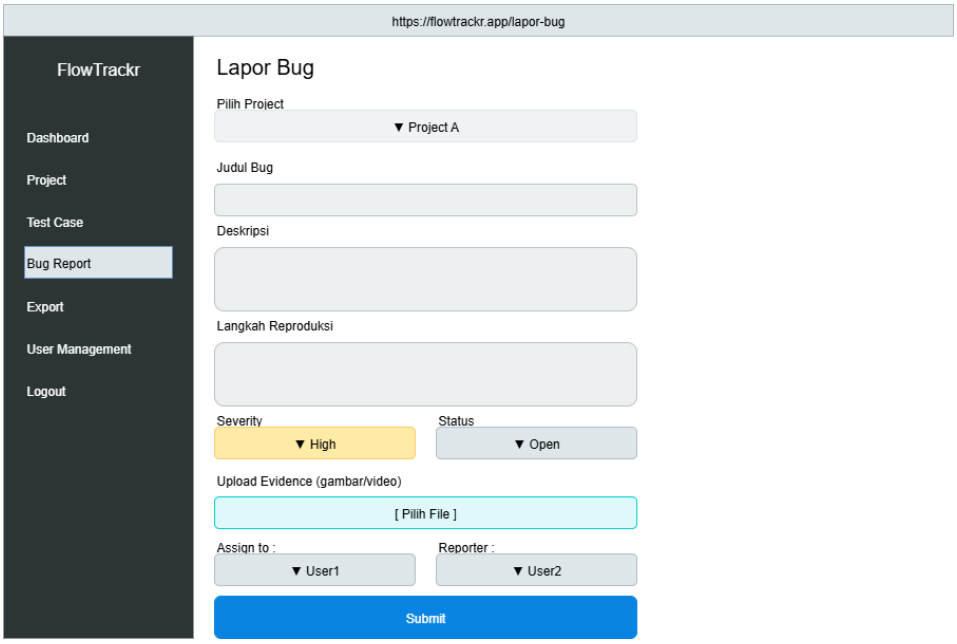
*Form input* hasil *test case* disediakan untuk mendokumentasikan proses pengujian secara sistematis dan terstruktur. Melalui *form* ini, pengguna dapat mencatat informasi penting seperti nama *test case*, skenario pengujian, langkah pelaksanaan, hasil aktual, serta status kelulusan (*pass/fail*). Penyusunan elemen *form* dibuat dalam susunan kolom dan *field* yang konsisten untuk memudahkan QA dalam mencatat setiap pengujian tanpa kehilangan detail krusial. Selain itu, adanya opsi pilihan atau *drop-down* pada beberapa *field* membantu mempercepat proses input dan mengurangi potensi kesalahan entri data. Tampilan yang bersih dan intuitif juga memungkinkan pengguna untuk fokus pada konten pengujian, sehingga seluruh tahapan uji terdokumentasi dengan rapi dan siap untuk ditelusuri kembali saat dibutuhkan. Dengan fitur ini, *FlowTrackr* mendukung praktik QA yang lebih disiplin dan terdokumentasi dengan baik. Gambar 7 menampilkan *mockup* hasil *test case* sistem *FlowTrackr*.




**Gambar 7** *Test case* sistem *FlowTrackr*

*Form* pelaporan *bug* merupakan salah satu komponen sentral dalam sistem *FlowTrackr* karena berfungsi sebagai sarana utama QA dalam mendokumentasikan temuan kesalahan selama proses pengujian. Tampilan antarmuka pada modul ini dirancang untuk memfasilitasi pelaporan yang jelas, terstruktur, dan mudah ditelusuri. Pengguna dapat mengisi detail seperti deskripsi *bug*, langkah reproduksi, hasil yang diharapkan, serta hasil aktual yang menunjukkan kegagalan. Selain itu, tersedia opsi untuk mengunggah *evidence* berupa tangkapan layar atau *file* pendukung guna memperkuat laporan. Desain *form* mengedepankan kejelasan dan kelengkapan informasi, sehingga laporan yang dikirimkan dapat dengan mudah dipahami oleh tim pengembang. Dengan alur pelaporan yang terstandarisasi dan terdokumentasi dengan baik, sistem ini mendukung proses *debugging* yang lebih cepat, serta meningkatkan efektivitas komunikasi antara tim QA dan *developer* dalam siklus pengembangan perangkat lunak. Gambar 8 menunjukkan *bug report* sistem.

Pengujian fungsional dilakukan untuk memastikan bahwa setiap fitur utama dalam sistem *FlowTrackr* berjalan sesuai dengan kebutuhan dan spesifikasi yang telah ditetapkan. Metode pengujian yang digunakan adalah *black box testing*, di mana fokus pengujian terletak pada kesesuaian *output* yang dihasilkan terhadap *input* yang diberikan, tanpa memperhatikan struktur internal program. Sepuluh skenario pengujian dirancang berdasarkan fitur inti yang digunakan oleh pengguna, seperti *login, input test case,* pelaporan *bug,* dan ekspor laporan. Hasil pengujian menunjukkan bahwa seluruh fungsi sistem dapat dijalankan dengan baik tanpa ditemukan *error* fungsional. Rangkuman hasil pengujian disajikan pada Tabel 1.




**Gambar 8** *Bug report* sistem *FlowTrackr*

**Tabel 1** Hasil pengujian fungsional sistem *FlowTrackr*

| | Pengujian | Keterangan | Hasil |
|---|---|---|---|
| 1. | *Login* dengan kredensial valid. | Pengguna berhasil masuk ke sistem. | Berhasil |
| 2. | *Login* dengan kredensial tidak valid. | Sistem menampilkan pesan *error*. | Berhasil |
| 3. | *Input test case* baru. | Data *test case* tersimpan. | Berhasil |
| 4. | Pelaporan *bug* dengan *evidence*. | Laporan dan *file evidence* tersimpan. | Berhasil |
| 5. | Edit *bug* yang sudah dilaporkan. | Perubahan data *bug* tersimpan dengan benar. | Berhasil |
| 6. | Unggah *evidence* pendukung pada *bug* report. | *File evidence* berhasil diunggah. | Berhasil |
| 7. | *Export* laporan pengujian. | *File* Laporan berhasil diunduh. | Berhasil |
| 8. | Buat proyek baru oleh *admin*. | Proyek muncul di *dashboard* pengguna. | Berhasil |
| 9. | Lihat detail *bug* oleh QA. | Informasi *bug* ditampilkan sesuai laporan. | Berhasil |
| 10. | *Logout* dari sistem. | Pengguna keluar dan dialihkan ke halaman *login*. | Berhasil |

Selain pengujian fungsional, dilakukan juga validasi pengguna melalui *User Acceptance Testing* (UAT) untuk memastikan sistem *FlowTrackr* dapat digunakan secara praktis oleh tim QA. UAT melibatkan lima orang QA yang diminta mencoba fitur utama seperti *login*, pencatatan *test case*, pelaporan *bug* dengan unggahan *evidence*, serta ekspor laporan. Hasilnya menunjukkan bahwa mayoritas responden menilai sistem mudah dipahami dan tidak membutuhkan pelatihan teknis yang rumit. Sebanyak empat dari lima QA (80 persen) menyatakan bahwa pencatatan *test case* dan pelaporan *bug* lebih cepat dibandingkan metode sebelumnya (*spreadsheet/email*), sementara seluruh responden (100 persen) sepakat bahwa

laporan hasil uji lebih rapi dan mudah ditelusuri. Temuan ini memperkuat klaim bahwa *FlowTrackr* berfokus pada kebutuhan praktis tim QA, terutama bagi organisasi dengan sumber daya terbatas.

Jika dibandingkan dengan alat manajemen QA yang lebih mapan seperti *Jira*, *TestRail*, atau *Bugzilla*, *FlowTrackr* menawarkan pendekatan yang lebih sederhana namun tetap terintegrasi. *Jira* dan *TestRail* menyediakan fitur yang lengkap, tetapi kompleksitas konfigurasi serta biaya lisensi sering menjadi kendala bagi tim kecil. *Bugzilla* relatif lebih ringan, namun antarmuka dan alur kerjanya masih terkesan teknis sehingga membutuhkan pemahaman tambahan. *FlowTrackr* hadir sebagai alternatif yang menggabungkan kebutuhan dasar QA *test case management*, *bug reporting*, *evidence upload*, dan *report export* dalam satu platform *web* ringan tanpa konfigurasi lanjutan. Dengan hasil pengujian fungsional, UAT, serta analisis perbandingan tersebut, *FlowTrackr* dapat diposisikan sebagai solusi praktis bagi organisasi yang memerlukan manajemen QA manual yang efisien, terdokumentasi, dan kolaboratif tanpa terbebani kompleksitas sistem QA berskala besar.

## Simpulan

Berdasarkan hasil perancangan, implementasi, dan pengujian, *FlowTrackr* berhasil dikembangkan sebagai platform berbasis *web* yang mampu mengintegrasikan proses pengujian perangkat lunak manual ke dalam sistem yang lebih terstruktur, efisien, dan mudah digunakan. Fitur inti seperti pencatatan *test case*, pelaporan *bug* dengan unggahan *evidence*, serta ekspor laporan telah diuji melalui pengujian fungsional dan validasi pengguna (UAT), yang menunjukkan bahwa sistem berjalan sesuai spesifikasi dan dinilai praktis oleh mayoritas tim QA. Dibandingkan dengan alat QA lain seperti *Jira* atau *TestRail* yang lebih kompleks, *FlowTrackr* menawarkan solusi ringan yang sesuai bagi tim kecil dengan kebutuhan dasar QA. Namun, penelitian ini masih memiliki keterbatasan, seperti belum adanya integrasi dengan sistem *Continuous Integration/Continuous Deployment* (CI/CD) dan belum mendukung otomasi pengujian. Ke depan, penelitian selanjutnya dapat diarahkan pada pengembangan fitur otomasi *testing*, integrasi dengan sistem manajemen kode (seperti *Git*), serta evaluasi dengan skala pengguna yang lebih luas untuk memvalidasi kinerja sistem dalam konteks industri yang lebih beragam.

## Daftar Pustaka

[1] M. G. Alkhairi, S. P. A. Alkadri, and P. Y. Utami, “Implementasi unit testing dan end-to-end testing pada sistem informasi akademik teknik informatika,” *JIPI (Jurnal Ilm. Penelit. dan Pembelajaran Inform.*, vol. 9, no. 4, pp. 2208–2219, 2024, doi: 10.29100/jipi.v9i4.5626.

[2] F. C. Ferdiana, J. U. D. Hatmoko, and B. H. Setiadji, “Pengaplikasian tingkatan sistem manajemen mutu pada proyek konstruksi: quality inspection, quality control,” *Syntax Lit.*, vol. 8, no. 7, pp. 5050–5065, 2023,

doi: 10.36418/syntax-literate.v6i6.

[3] M. W. R. Sastyawan, A. R. S. P. A. Yani, Z. Z. Alkaf, T. Wisudawati, R. D. Putera, and Hurun'in, "Digitalization of TQM: From manual to smart quality system," *J. Tek. Mesin Indones.*, vol. 20, no. 1, pp. 203–209, 2025, doi: 10.36289/jtmi.v20i1.890.

[4] E. Setyabudi and R. Renaldy, "IN-FEST 2024 membangun masa depan: peran AI dalam transformasi kota pintar yang berkelanjutan," in *Proceedings of IN-FEST 2024*, 2024, pp. 637–643.

[5] A. Ernawati, D. Kurnia, and A. Hindasyah, "Sistem informasi quality assurance proses produksi menggunakan metode agile berbasis web," *Inform. Univ. Pamulang*, vol. 6, no. 3, pp. 491–497, 2021, [Online]. Available: http://openjournal.unpam.ac.id/index.php/informatika/article/view/10272/pdf

[6] S. I. Mandala, L. Fitriani, and A. Sutedi, "Rancang bangun aplikasi pengelolaan bug tracker berbasis web pada pengembangan software," *J. Algoritm.*, vol. 19, no. 1, pp. 180–191, 2022, doi: 10.33364/algoritma/v.19-1.1046.

[7] J. L. Min, A. Istiqomah, and A. Rahmani, "Evaluasi penggunaan manual dan automated software testing pada pelaksanaan end-to-end testing," *J. Teknol. Terap.*, vol. 6, no. 1, p. 18, 2020, doi: 10.31884/jtt.v6i1.256.

[8] A. Maspupah, A. Rahmani, J. L. Min, and T. A. Roshinta, "Software testing in the Indonesian industry: survey of methods, tools, and documentation," *Innov. Res. Informatics*, vol. 6, no. 2, pp. 106–117, 2024, doi: 10.37058/innovatics.v6i2.12636.

[9] D. A. N. Hasibuan and T. Dirgahayu, "Pengujian dengan unit testing dan test case pada proyek pengembangan modul manajemen pengguna," *J. Inform. Univ. Islam Indones.*, vol. 2, no. 1, pp. 103–109, 2020.

[10] S. L. Ramadhani and H. P. Putro, "Manajemen pengujian perangkat lunak menggunakan aplikasi Qase," *Automata*, vol. 4, no. 1, 2023.

[11] D. Chhabra and R. Chadha, "Automatic bug triaging process: an enhanced machine learning approach through large language models," *Eng. Technol. Appl. Sci. Res.*, vol. 14, no. 6, pp. 18557–18562, 2024, doi: 10.48084/etasr.8829.

[12] A. A. Wahid, "Analisis metode waterfall untuk pengembangan sistem informasi," *J. Ilmu-ilmu Inform. dan Manaj. STMIK*, vol. 1, no. 1, pp. 1–5, 2020, [Online]. Available: https://www.researchgate.net/publication/346397070

[13] R. Rifai and M. Mailasari, "Metode waterfall pada sistem informasi

pengolahan data penjualan dan pembelian barang," *J. Inform. Univ. Pamulang*, vol. 5, no. 3, p. 394, 2020, doi: 10.32493/informatika.v5i3.6721.

[14] K. Fahrezi, A. R. Mulana, S. Melinda, N. Nurhaliza, and S. Mulyati, "Penerapan model waterfall dalam pengembangan sistem informasi akademik berbasis web sebagai sistem pengolahan nilai siswa," *J. Teknol. Sist. Inf. dan Apl.*, vol. 4, no. 2, p. 98, 2021, doi: 10.32493/jtsi.v4i2.10196.

[15] A. R. Dzulmi and A. B. Tjandrarini, "Pengembangan sistem informasi pengadaan barang menggunakan metode waterfall," *J. Ilm. Scroll Inform.*, vol. 11, no. 1, pp. 22–28, 2023, [Online]. Available: http://univ45sby.ac.id/ejournal/index.php/informatika/article/download/371/321