IMPLEMENTASI *CLEAN ARCHITECTURE* PADA SISTEM MANAJEMEN TIKET PENANGANAN INSIDEN

Bahtiar Rifa'i[1], Dian Hanifudin Subhi[2], Mungki Astiningrum[3]

[1,2 ,3]Teknik Informatika, Teknologi Informasi, Politeknik Negeri Malang, Indonesia
[1]bahtiarderifai@gmail.com, [2] dhanifudin@polinema.ac.id, [3] mungki.astiningrum@polinema.ac.id

Abstrak

Keamanan siber telah menjadi isu global yang semakin krusial seiring pesatnya perkembangan teknologi informasi. Laporan Badan Siber dan Sandi Negara (BSSN) mencatat lebih dari 888 juta serangan siber di Indonesia pada tahun 2021, menegaskan tingginya urgensi akan sistem manajemen insiden yang efektif, terstruktur, dan responsif. Penelitian ini bertujuan membangun sistem manajemen tiket insiden yang mengintegrasikan platform komunikasi populer, WhatsApp, sebagai jalur pelaporan otomatis, sekaligus menerapkan prinsip *Clean Architecture* untuk memastikan modularitas, fleksibilitas, serta kemudahan pengujian. Sistem dikembangkan dengan teknologi *backend* berperforma tinggi, integrasi otomatis dengan WhatsApp, dan antarmuka web yang mendukung notifikasi *real-time*. Metodologi pengembangan mencakup studi literatur, analisis kebutuhan, perancangan sistem, implementasi, serta pengujian. Hasil evaluasi menunjukkan bahwa *unit test* pada lapisan *use case* mencapai *coverage* 91,6%, melampaui standar minimal 80% untuk kestabilan logika bisnis. Integrasi WhatsApp memungkinkan pembuatan tiket otomatis dan notifikasi real-time, sehingga mempercepat eskalasi insiden. Sementara itu, pengujian *User Acceptance Test (UAT)* oleh tujuh responden menghasilkan skor rata-rata 90,61%, dengan aspek percepatan eskalasi mencapai 97,14%. Temuan ini membuktikan bahwa kombinasi *Clean Architecture* dan integrasi WhatsApp mampu menghasilkan sistem manajemen insiden yang modular, responsif, serta efektif dalam mendukung kebutuhan operasional keamanan siber modern dan siap untuk diadopsi di lingkungan industri.

Kata kunci: *Clean Architecture*, Sistem Manajemen Tiket, Keamanan Siber, Golang

1. Pendahuluan

Keamanan siber telah menjadi fokus krusial seiring pesatnya perkembangan teknologi informasi. Laporan Badan Siber dan Sandi Negara (BSSN) menunjukkan lebih dari 888 juta serangan siber terpantau di Indonesia pada tahun 2021. Angka ini menegaskan bahwa infrastruktur digital, baik pemerintah maupun swasta, berada dalam risiko tinggi. Pemantauan insiden secara *real-time* menjadi strategi utama untuk meminimalkan dampak serangan (Irawan et al., 2024). Ancaman siber tidak hanya menyasar lembaga besar namun juga perusahaan lokal, sehingga diperlukan sistem manajemen insiden keamanan informasi yang andal dan terintegrasi agar penanganan insiden lebih efisien.

PT. XYZ, sebuah perusahaan sektor telekomunikasi, menghadapi tantangan dalam pengelolaan insiden keamanan siber. Saat ini, notifikasi insiden dari sistem pemantauan diterima melalui grup WhatsApp dan dicatat manual ke aplikasi tiket terpisah. Proses pencatatan ini memakan waktu dan rentan terhadap kesalahan, terutama saat terjadi lonjakan insiden yang membutuhkan respons cepat dan akurat.. Akibatnya, efisiensi operasional menurun drastis, terutama ketika terjadi banyak insiden kritis yang membutuhkan respons cepat dan akurat. Berdasarkan kendala tersebut, diperlukan pengembangan aplikasi *web* yang mampu mengelola insiden keamanan siber secara terpusat dan terstruktur dengan integrasi langsung ke platform WhatsApp.

Struktur kode yang kurang terorganisir menjadi tantangan dalam pengembangan perangkat lunak. Logika bisnis dan tampilan yang tercampur dalam satu modul menyulitkan kode untuk dibaca, diuji, dan dikembangkan (Martin, 2017). Pemilihan *Clean Architecture* dalam penelitian ini didasarkan pada beberapa studi komparatif terdahulu. Penerapan *Clean Architecture* terbukti dapat memisahkan tugas dan tanggung jawab setiap komponen dengan jelas, sehingga memudahkan pemeliharaan dan pengembangan sistem di masa mendatang (Santiago-Salazar & Rico-Bautista, 2023). Dibandingkan dengan pendekatan arsitektur tradisional seperti MVC dan *layered architecture*, *Clean Architecture* memiliki keunggulan dalam kemudahan pengujian

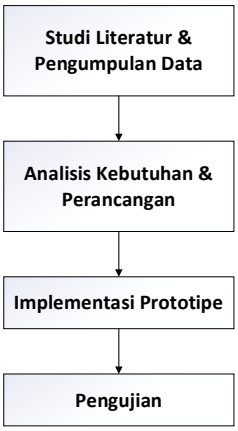
dan pemeliharaan. Pemisahan yang jelas antara logika bisnis inti dengan komponen teknis eksternal membuat sistem lebih fleksibel terhadap perubahan teknologi (Gluschah, 2023; Maryono, 2025). Dalam konteks sistem yang memerlukan integrasi dengan platform eksternal seperti layanan komunikasi, *Clean Architecture* memungkinkan komponen teknologi diganti tanpa mempengaruhi logika bisnis inti. Kemampuan *plug-and-play* ini menjadi keunggulan utamanya dibandingkan arsitektur berlapis konvensional (Martin, 2017).

Beberapa penelitian terdahulu telah memberikan kontribusi dalam domain manajemen insiden dan arsitektur perangkat lunak. *Ticketing system* dapat meningkatkan efisiensi SOC melalui penyediaan lokasi terpusat untuk *tracking* dan *managing incidents*, mengurangi waktu respons, dan meningkatkan efektivitas penanganan insiden secara keseluruhan (Lutz, 2023). Integrasi bot WhatsApp dapat mempercepat pelaporan dengan notifikasi dalam waktu kurang dari 5 detik (Fu'adi et al., 2025). Implementasi *Clean Architecture* pada Golang juga telah terbukti menghasilkan aplikasi dengan performa tinggi dan mudah dipelihara (Annisa et al., 2024; Pamungkas & Setiaji, 2024). Meskipun penelitian-penelitian tersebut memberikan kontribusi signifikan, belum ditemukan studi yang mengintegrasikan *Clean Architecture* dengan platform WhatsApp untuk sistem manajemen tiket insiden keamanan siber. Penelitian terdahulu umumnya fokus pada satu aspek saja, sehingga terdapat celah penelitian dalam mengombinasikan prinsip arsitektur yang baik dengan platform komunikasi untuk mempercepat penanganan insiden.

Berdasarkan hal tersebut, penelitian ini bertujuan mengisi celah *riset* dengan mengimplementasikan *Clean Architecture* pada sistem manajemen tiket insiden yang terintegrasi dengan WhatsApp untuk penanganan insiden keamanan siber. Berbeda dengan penelitian terdahulu yang umumnya fokus pada satu aspek (arsitektur atau integrasi *platform*), penelitian ini mengombinasikan kedua aspek tersebut dengan solusi insiden manual dan berkontribusi akademis dengan bukti efektivitas *Clean Architecture* pada sistem keamanan siber *real-time*. Hasil penelitian ini diharapkan memberikan rujukan bagi praktisi dan peneliti dalam membangun sistem manajemen insiden yang terstruktur, *testable*, dan fleksibel terhadap perubahan teknologi di masa mendatang.

2. Metode

Secara umum, metodologi yang digunakan meliputi Studi Literatur & Pengumpulan Data, Analisis Kebutuhan & Perancangan, Implementasi Prototipe, dan Pengujian. Seluruh tahapan ini dijalankan untuk memastikan bahwa sistem yang dibangun dapat mengatasi tantangan di PT XYZ, yaitu membuat penanganan insiden keamanan siber menjadi lebih terstruktur. Dalam penelitian ini, pengembangan sistem dilakukan sesuai dengan tahapan yang diilustrasikan pada Gambar 1. Dimulai dari pemahaman teori hingga implementasi teknis, setiap tahap penelitian saling berkaitan di mana hasil satu tahap menjadi dasar bagi tahap berikutnya, sehingga menciptakan alur kerja yang terarah dan sistematis.



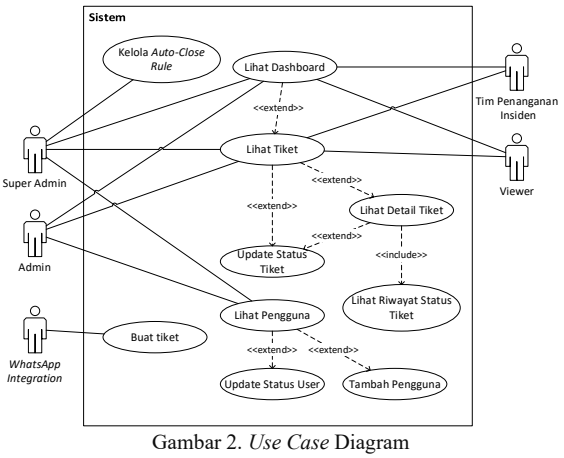
Gambar 1. Metode Pengembangan

2.1 Studi Literatur & Pengumpulan Data

Mengkaji penelitian terdahulu, jurnal, buku, dan dokumentasi teknis terkait integrasi WhatsApp, serta *Clean Architecture*. Selain itu, dilakukan wawancara dengan tim keamanan siber PT. XYZ untuk mengidentifikasi proses bisnis dan kebutuhan sistem saat ini.

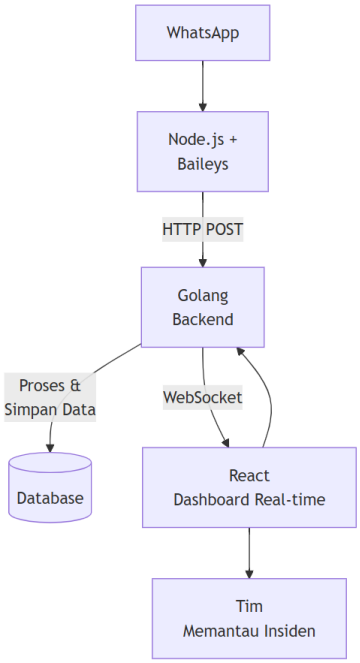
2.2 Analisis Kebutuhan & Perancangan

Berdasarkan observasi dan diskusi dengan tim operasional keamanan siber PT XYZ, sistem harus memenuhi kebutuhan: (1) pembuatan tiket otomatis dari pesan WhatsApp, (2) *dashboard real-time* statistik insiden, (3) pengelolaan tiket dengan pembaruan status dan catatan, (4) notifikasi *real-time*, serta (5) manajemen pengguna dan aturan *auto-close*. Diagram *use case* pada Gambar 2. menggambarkan interaksi aktor (Super Admin, Admin, Tim Penanganan Insiden) dengan sistem.



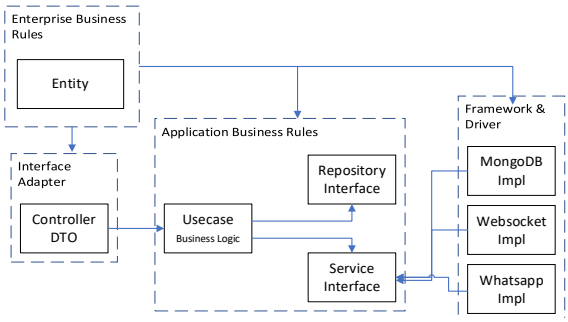
Gambar 2. *Use Case* Diagram

Integrasi WhatsApp berfungsi sebagai aktor utama dalam proses pembuatan tiket otomatis, yang membedakan sistem ini dari prosedur lama. Tim Penanganan Insiden dapat melihat dasbor dan tiket, memperbarui status tiket, dan melihat riwayat status. Sementara itu, peran Admin dan Super Admin memiliki akses yang lebih luas, termasuk mengelola aturan auto-close dan menambahkan pengguna baru. Diagram ini menegaskan pemisahan peran dan tanggung jawab, yang merupakan inti dari desain sistem yang terstruktur. Pada Gambar 3. dimana proses dimulai dari penerimaan data dari WhatsApp hingga tim bisa memantau insiden masuk.



Gambar 3. Alur Kerja Sistem

Alur kerja sistem dimulai dari penerimaan pesan WhatsApp yang diproses oleh *gateway* Baileys untuk ekstraksi data. Data tervalidasi kemudian disimpan ke *database* dan memicu notifikasi *real-time* ke antarmuka pengguna melalui WebSocket. Tim penanganan insiden dapat memperbarui status tiket, menambahkan catatan penanganan, dan melampirkan bukti penyelesaian. Pada Gambar 4, arsitektur sistem dirancang dalam empat lapisan utama yang terorganisir sesuai dengan prinsip *Clean Architecture*.



Gambar 4. Arsitektur Sistem

- *Enterprise Business Rules (Entities):* Lapisan ini menyimpan data fundamental sistem seperti informasi tiket dan pengguna beserta aturan validasi dasarnya. Lapisan ini dirancang tidak bergantung pada teknologi spesifik sehingga dapat bertahan meskipun infrastruktur teknis berubah.
- *Application Business Rules (Use Cases):* Lapisan ini mengatur alur proses utama sistem seperti pembuatan tiket, pencarian data, pembaruan status, dan autentikasi pengguna. Lapisan ini dirancang independen dari komponen eksternal, sehingga perubahan pada database atau layanan eksternal tidak mempengaruhi logika bisnis inti.
- *Interface Adapters:* Lapisan ini menjembatani logika bisnis dengan antarmuka eksternal. *Controller* pada lapisan ini menerima dan memvalidasi *input* pengguna, kemudian mengadaptasikan format data untuk diproses oleh lapisan *use case*. Hasil pemrosesan kemudian diubah kembali menjadi format yang sesuai untuk ditampilkan ke pengguna.
- *Frameworks & Drivers:* Lapisan terluar ini mengintegrasikan teknologi spesifik seperti *database* dan layanan notifikasi *real-time*. Karakteristik utama lapisan ini adalah kemudahan penggantian teknologi, misalnya migrasi *database* dari MongoDB ke PostgreSQL atau platform komunikasi WhatsApp ke platform komunikasi lain tanpa mengubah logika bisnis sistem.

2.3 Implementasi Prototipe

Mengembangkan prototipe sistem sesuai desain. *Backend* dibangun dengan bahasa pemrograman Go (Golang) yang dirancang Google untuk menangani banyak proses secara bersamaan dengan efisien, cocok untuk sistem yang membutuhkan respons cepat terhadap banyak insiden secara simultan (Cox et al., 2022). Implementasi *backend* mengikuti struktur *Clean Architecture*, yang memisahkan logika bisnis inti dari detail teknis seperti akses data dan layanan eksternal. *Frontend* dikembangkan dengan React, *framework* JavaScript yang mendukung pembuatan antarmuka responsif dan interaktif (Komperla et al., 2022). Penggunaan TypeScript meningkatkan keandalan kode melalui deteksi *error* lebih awal, (Rozentals, 2021), sementara integrasi REST API dan WebSocket memungkinkan sinkronisasi data *real-time* dengan *backend*. WebSocket dipilih untuk memungkinkan komunikasi dua arah yang terus terhubung antara sistem dan pengguna, sehingga notifikasi insiden dapat diterima secara instan tanpa perlu memuat ulang halaman (Pimentel & Nickerson, 2012).

Integrasi WhatsApp diwujudkan dengan memanfaatkan *library* Baileys pada Node.js sebagai *WhatsApp gateway*, sehingga sistem dapat menerima pesan insiden otomatis dari grup WhatsApp. MongoDB dipilih sebagai *database* karena setiap

jenis insiden memiliki informasi yang berbeda-beda. MongoDB memungkinkan penyimpanan data dengan struktur yang fleksibel, sehingga sistem dapat menangani berbagai jenis data tanpa perlu mengubah skema *database* (Abu Kausar et al., 2022). Pendekatan ini lebih efisien dibandingkan *database* tradisional yang mengharuskan struktur data seragam untuk semua jenis insiden.

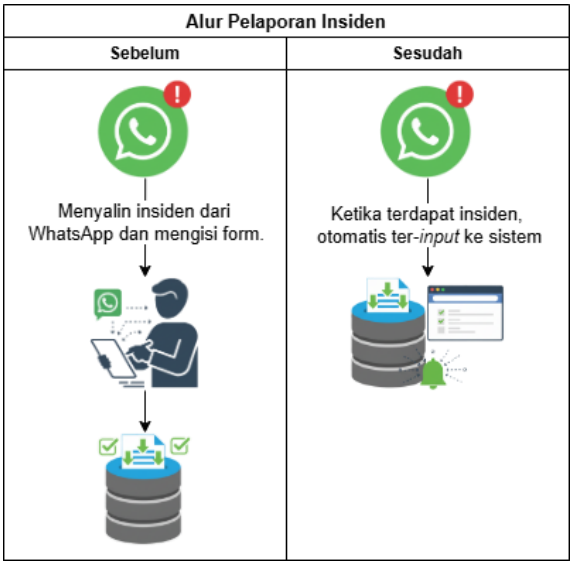
2.4 Pengujian

Pengujian dilakukan untuk menilai kualitas sistem baik dari sisi arsitektur maupun dari sisi kesesuaian kebutuhan pengguna. Pada tahap pertama dilakukan pengujian *Clean Architecture*, yaitu dengan melakukan *unit test*. *Unit testing* merupakan praktik pengujian perangkat lunak pada tingkat paling dasar yang menguji fungsi individual secara terisolasi untuk memverifikasi setiap unit kode berfungsi sesuai spesifikasi (Farooq et al., 2021). *Unit test* difokuskan pada lapisan *use case* untuk memverifikasi logika bisnis dapat berjalan dengan benar secara terisolasi. Untuk memastikan sistem menerapkan prinsip *Clean Architecture*, dilakukan percobaan pergantian integrasi dari *WhatsApp* ke *TXTPlatform* untuk menilai independensi logika bisnis terhadap platform eksternal.

Tahap berikutnya adalah pengujian kesesuaian kebutuhan yang dilakukan melalui *User Acceptance Test (UAT)*. UAT merupakan fase pengujian akhir di mana sistem divalidasi oleh pengguna akhir atau stakeholder untuk memastikan perangkat lunak memenuhi kebutuhan bisnis dan berfungsi sesuai harapan dalam skenario dunia nyata (Gordon et al., 2022). UAT efektif karena melibatkan pengguna aktual yang mengoperasikan sistem sehari-hari, mampu mengidentifikasi permasalahan usability dan kesesuaian alur kerja operasional yang terlewatkan pada pengujian teknis (Aliyah et al., 2025). UAT melibatkan tujuh anggota tim operasional keamanan siber PT XYZ dengan skenario pelaporan insiden nyata melalui WhatsApp dan pemantauan proses penanganan insiden. Setiap responden mengisi kuesioner penilaian aspek fungsionalitas, kemudahan penggunaan, kecepatan respons, dan manfaat sistem. Hasil UAT dihitung dalam bentuk persentase rata-rata kepuasan dengan skala Likert.

3. Hasil dan Pembahasan

Implementasi sistem menghasilkan tiga kontribusi utama. Pertama, otomatisasi pembuatan tiket dari WhatsApp mengeliminasi langkah pencatatan manual yang sebelumnya dilakukan petugas. Kedua, penerapan *Clean Architecture* menghasilkan struktur kode modular dengan pemisahan tanggung jawab yang jelas antar lapisan. Ketiga, notifikasi *real-time* melalui WebSocket memungkinkan tim menerima informasi insiden secara instan. Gambar 5. menunjukkan perbandingan alur kerja sebelum dan setelah implementasi sistem.



Gambar 5. Perbandingan Alur Pelaporan Insiden Sebelum & Setelah Implementasi Sistem

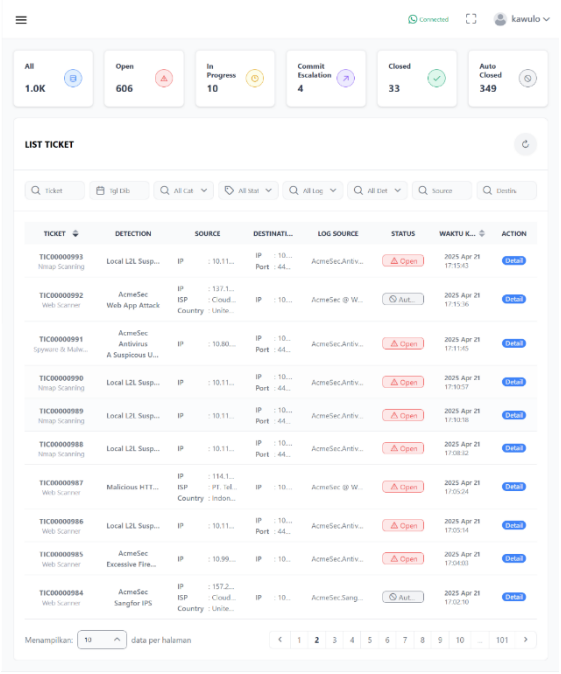
3.1 Hasil

Dashboard real-time menyajikan visualisasi statistik kategori insiden, tingkat ancaman, dan aktivitas sumber IP yang ditunjukkan pada Gambar 6. Agregasi data ini memungkinkan tim keamanan mengidentifikasi pola serangan dan memprioritaskan penanganan secara lebih efisien dibandingkan metode manual sebelumnya.

Gambar 6. Tampilan Dashboard

Pengelolaan tiket dilengkapi dengan mekanisme *audit trail* yang mencatat setiap perubahan status dan catatan penanganan seperti ditunjukkan pada Gambar 7. Pencatatan otomatis ini memastikan akuntabilitas dan transparansi dalam proses penanganan insiden.

Fitur pencarian dan filter berdasarkan prioritas, status, atau waktu mempercepat akses informasi yang diperlukan tim.



Gambar 7. Tampilan Daftar Tiket Insiden

3.1.1 Hasil Pengujian *Clean Architecture*

Pengujian dilakukan pada lapisan logika bisnis inti dengan memisahkan dari komponen eksternal. Hasil pengujian menunjukkan 91,6% kode berhasil diuji dan berfungsi dengan benar, melampaui standar minimal 80% yang direkomendasikan untuk sistem kritikal (Andersen, 2025). Angka ini mengindikasikan bahwa sebagian besar alur bisnis sistem telah terverifikasi kehandalannya..

Meskipun terdapat pandangan bahwa *coverage* tinggi tidak selalu identik dengan kualitas fungsional, ambang 80% pada lapisan *use case* tetap dianggap rasional untuk memastikan sebagian besar alur logika telah diuji (Jemuović, 2022). Dengan demikian, capaian penelitian ini menunjukkan bahwa sistem memenuhi prinsip *Clean Architecture*, di mana logika bisnis terpisah dari infrastruktur teknis dan dapat diuji secara menyeluruh, sebagaimana ditunjukkan pada Gambar 8.

```
$ go test -coverprofile=coverage.out ./internal/usecase/...
ok      github.com/derivai7/csirt-v2/internal/usecase   2.963
s  coverage: 91.6% of statements
```

Gambar 8. Hasil *Test Coverage* pada Lapisan *Use Case*

Untuk menguji fleksibilitas arsitektur, dilakukan simulasi pergantian platform komunikasi dari WhatsApp menjadi platform berbasis file teks. Hasil pengujian pada Gambar 9. menunjukkan sistem berhasil beradaptasi tanpa mengubah logika bisnis inti, membuktikan prinsip independensi yang menjadi kekuatan *Clean Architecture*.

```
[17-06-2025 13:01:32 +07:00] INFO: 🚀 Starting TXT platform...
[17-06-2025 13:01:32 +07:00] INFO: Loaded 5 processed incidents
[17-06-2025 13:01:32 +07:00] INFO: 👀 Watching: ./data/incidents.txt
[17-06-2025 13:01:32 +07:00] INFO: File processed successfully
[17-06-2025 13:01:32 +07:00] INFO: Server running on port 3100
[17-06-2025 13:01:57 +07:00] INFO: File modified, processing...
[17-06-2025 13:01:57 +07:00] INFO: Detected incident message, sending to API...
[17-06-2025 13:01:57 +07:00] INFO: File processed successfully
[17-06-2025 13:01:57 +07:00] INFO: Incident sent successfully
```

Gambar 9. Hasil pengujian pergantian dari WhatsApp ke TxtPlatform

3.1.2 Pengujian Kesesuaian Kebutuhan

Pengujian kesesuaian kebutuhan dilakukan setelah sistem digunakan secara nyata oleh tim penanganan insiden keamanan siber di PT. XYZ. Sebanyak **7** responden yang merupakan pengguna aktif diminta mengisi kuesioner evaluasi dengan tujuh pernyataan yang mencakup aspek kemudahan penggunaan, efisiensi proses, kejelasan informasi, serta stabilitas layanan. Perhitungan skor dilakukan dengan metode skala Likert 1-5, di mana 1 menunjukkan "sangat tidak setuju" dan 5 menunjukkan "sangat setuju". Hasil rekapitulasi ditampilkan pada Tabel 1.

Tabel 1. Hasil pengujian pergantian dari WhatsApp ke TxtPlatform

No	Pertanyaan	Persentase
1	Sistem mudah digunakan dan dipahami	91,43%
2	Proses pembuatan dan pelacakan tiket berjalan lancar	91,43%
3	Notifikasi insiden mudah dikenali dan tidak tertinggal	80,00%
4	Tampilan dan navigasi antarmuka cukup informatif dan mudah digunakan	91,43%
5	Fitur pencatatan/filter tiket membantu pekerjaan	91,43%
6	Sistem mempercepat eskalasi dan penanganan insiden	97,14%
7	Waktu respons sistem cukup cepat	91,43%
	Total rata-rata	90,61%

Berdasarkan perhitungan menggunakan rumus $P = \frac{I}{Y} x100\%$, hasil evaluasi menunjukkan bahwa rata-rata tingkat kepuasan pengguna mencapai 90,61%. Nilai tertinggi terdapat pada aspek *sistem mempercepat eskalasi dan penanganan insiden* dengan persentase 97,14%. Sementara itu, nilai terendah terdapat pada aspek *notifikasi insiden mudah dikenali dan tidak tertinggal* dengan persentase 80,00%.

3.2 Pembahasan

Pengujian *Clean Architecture* menunjukkan hasil yang sangat baik. *Unit test* pada lapisan *use case* menghasilkan *coverage* sebesar 91,6%, melampaui standar minimal 80% yang umum digunakan dalam industri. Capaian ini membuktikan bahwa sistem dapat diandalkan dan mudah dipelihara, karena perubahan pada satu bagian sistem dapat segera terdeteksi jika menyebabkan *error* pada bagian lain. Fleksibilitas arsitektur divalidasi melalui pengujian pergantian platform komunikasi dari WhatsApp ke TxtPlatform, dimana sistem tetap berfungsi tanpa modifikasi pada logika pengelolaan tiket. Pemisahan ini memiliki implikasi strategis untuk pengembangan masa depan. Sebagai ilustrasi, pergantian kanal

pelaporan seperti Telegram atau Slack hanya memerlukan perubahan pada lapisan infrastruktur tanpa memengaruhi sistem pengelolaan tiket yang sudah ada. Modularitas ini juga mempercepat *onboarding developer* baru melalui struktur tanggung jawab yang terdefinisi dengan jelas.

Pengujian kesesuaian kebutuhan dilakukan melalui *User Acceptance Test* (UAT) yang melibatkan tujuh anggota tim operasional keamanan siber PT XYZ. Hasil UAT memberikan skor rata-rata 90,61%, dengan nilai tertinggi 97,14% pada aspek percepatan eskalasi dan penanganan insiden. Skor ini mencerminkan bahwa sistem dinilai mampu memenuhi kebutuhan pengguna, terutama dalam hal fungsionalitas, kemudahan penggunaan, dan kecepatan respons. Walaupun terdapat satu aspek yang memperoleh nilai lebih rendah, yaitu notifikasi yang mudah dikenali (80%), secara keseluruhan pengguna menyatakan kepuasan yang tinggi terhadap sistem. Hal ini menegaskan bahwa sistem tidak hanya layak secara teknis, tetapi juga efektif dalam menjawab kebutuhan operasional sehari-hari tim keamanan siber.

4. Kesimpulan

Penelitian ini membuktikan efektivitas *Clean Architecture* dalam konteks sistem manajemen insiden keamanan siber. Capaian *unit test coverage* 91,6% menunjukkan modularitas dan *testability* yang tinggi, sementara integrasi WhatsApp menghasilkan peningkatan signifikan dalam efisiensi operasional dengan skor kepuasan pengguna 90,61% serta percepatan eskalasi insiden hingga 97,14%.

Penelitian ini memberikan bukti empiris efektivitas *Clean Architecture* pada sistem keamanan siber dengan kebutuhan *real-time*, sekaligus mengisi celah riset terkait arsitektur dan integrasi *platform* komunikasi. Studi ini juga membuktikan bahwa sistem dapat beradaptasi dengan *platform* komunikasi berbeda tanpa mengubah logika bisnis inti melalui pengujian pergantian dari WhatsApp ke *platform* alternatif. Selain itu, diusulkan kerangka evaluasi yang mengombinasikan pengujian teknis dengan pengujian penerimaan pengguna, sehingga kualitas teknis sistem dapat diselaraskan dengan kebutuhan operasional nyata.

Untuk pengembangan di masa depan, disarankan dari hasil UAT untuk memperkaya sistem dengan kemampuan analitik cerdas melalui integrasi model *machine learning* untuk deteksi pola ancaman secara proaktif. Selain itu, disarankan pula untuk mengembangkan antarmuka pengguna grafis (GUI) yang memungkinkan konfigurasi aturan parsing pesan secara dinamis, sehingga meningkatkan fleksibilitas sistem untuk beradaptasi dengan berbagai format sumber insiden tanpa memerlukan perubahan kode.

Daftar Pustaka:

Abu Kausar, M., Nasar, M., & Soosaimanickam, A. (2022). *A study of performance and comparison of nosql databases: Mongodb, cassandra, and redis using ycsb*. Indian Journal of Science and Technology, 15(31), 1532–1540.

Aliyah, A., Hartono, N., & Muin, A. A. (2025). *Penggunaan User Acceptance Testing (UAT) pada pengujian sistem informasi pengelolaan keuangan dan inventaris barang*. Switch: Jurnal Sains Dan Teknologi Informasi, 3(1), 84–100.

Andersen, G. (2025, June 12). *Master Clean Architecture Patterns for Android App Success*. MoldStud. https://moldstud.com/articles/p-master-clean-architecture-patterns-for-building-successful-android-apps

Annisa, R., Ananda, R. A., & Sulistiono, W. E. (2024). *Implementasi Golang Clean Architecture Pada Perancangan Backend Point Of Sales Website*. Jurnal Informatika Dan Teknik Elektro Terapan, 12(2).

Cox, R., Griesemer, R., Pike, R., Taylor, I. L., & Thompson, K. (2022). *The Go programming language and environment*. Communications of the ACM, 65(5), 70–78.

Farooq, M. S., Tehseen, R., Omer, U., Riaz, S., & Tahir, S. (2021). *Software Testing Education: A Systematic Literature Review*. VFAST Transactions on Software Engineering, 9(4), 109–125.

Fu'adi, M., Iswoyo, D., & Tuhehay, Y. A. S. (2025). *Sistem pelaporan kampus terintegrasi dengan WhatsApp untuk mendukung pembangunan infrastruktur smart campus*. Seminar Nasional Teknologi & Sains, 4(1), 743–749.

Gluschah, R. (2023, October 10). *Understanding Hexagonal, Clean, Onion, and Traditional Layered Architectures: A Deep Dive*. Medium. https://romanglushach.medium.com/understanding-hexagonal-clean-onion-and-traditional-layered-architectures-a-deep-dive-c0f93b8a1b96

Gordon, S., Crager, J., Howry, C., Barsdorf, A. I., Cohen, J., Crescioni, M., Dahya, B., Delong, P., Knaus, C., & Reasner, D. S. (2022). *Best practice recommendations: user acceptance testing for systems designed to collect clinical outcome assessment data electronically*. Therapeutic Innovation & Regulatory Science, 56(3), 442–453.

Irawan, A., Fadholi, W. H. N., Erikamaretha, Z., & Sinlae, F. (2024). *Tantangan dan Strategi Manajemen Keamanan Siber di Indonesia berbasis IoT*. Journal Zetroem, 6(1), 114–119.

Jemuović, V. (2022, December 8). *Code Coverage Targets - Recipe for Disaster*. Optivem

Journal. https://journal.optivem.com/p/code-coverage-targets-recipe-for-disaster

Komperla, V., Deenadhayalan, P., Ghuli, P., & Pattar, R. (2022). *React: A detailed survey*. Indonesian Journal of Electrical Engineering and Computer Science, 26(3), 1710–1717.

Lutz, M. (2023, May 10). *Maximizing SOC Efficiency: The Power of Ticketing Systems and SIEM Integration*. LinkedIn. https://www.linkedin.com/pulse/maximizing-soc-efficiency-power-ticketing-systems-siem-martin-lutz/

Martin, R. C. (2017). *Clean architecture: a craftsman's guide to software structure and design*. Prentice Hall Press.

Maryono, D. (2025). *Understanding MVC and Clean Architecture in Backend Development*. Didik Maryono.

Pamungkas, F., & Setiaji, H. (2024). *Implementasi Clean Architecture Pada Pembuatan Api Menggunakan Golang*. Jurnal INSTEK (Informatika Sains Dan Teknologi), 9(1), 80–86.

Pimentel, V., & Nickerson, B. (2012). *WebSocket for Communication and Display of Real-Time Data*. Internet Computing.

Rozentals, N. (2021). *Mastering TypeScript: Build enterprise-ready, modular web applications using TypeScript 4 and modern frameworks*. Packt Publishing Ltd.

Santiago-Salazar, J. A., & Rico-Bautista, D. (2023). *Clean Architecture: Impact on Performance and Maintainability of Native Android Projects*. Colombian Conference on Computing, 82–90.

Halaman ini sengaja dikosongkan