PENGEMBANGAN PLATFORM MANAJEMEN TIM *E-SPORTS*: STUDI KASUS PADA KOMUNITAS *CLASH OF CLANS* DI INDONESIA

Disusun Oleh:

N a m a : Dicky Galuh Kurniawan

NIM : 20523234

PROGRAM STUDI INFORMATIKA – PROGRAM SARJANA

FAKULTAS TEKNOLOGI INDUSTRI

UNIVERSITAS ISLAM INDONESIA

2026

HALAMAN PENGESAHAN DOSEN PEMBIMBING

PENGEMBANGAN PLATFORM MANAJEMEN TIM *E-SPORTS*: STUDI KASUS PADA KOMUNITAS *CLASH OF CLANS* DI INDONESIA

TUGAS AKHIR

Disusun Oleh:

N a m a : Dicky Galuh Kurniawan

NIM : 20523234

Yogyakarta, 21 Januari 2026

Pembimbing,

(Kholid Haryono, ST., M.Kom.)

HALAMAN PENGESAHAN DOSEN PENGUJI

PENGEMBANGAN PLATFORM MANAJEMEN TIM *E-SPORTS*: STUDI KASUS PADA KOMUNITAS *CLASH OF CLANS* DI INDONESIA

TUGAS AKHIR

Telah dipertahankan di depan sidang penguji sebagai salah satu syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Informatika – Program Sarjana di Fakultas Teknologi Industri Universitas Islam Indonesia

Yogyakarta, 21 Januari 2026

Tim Penguji

Kholid Haryono, ST., M.Kom. _____

Anggota 1

Aridhanyati Arifin, S.T., M.Cs. _____

Anggota 2

Elyza Gustri Wahyuni, S.T., M.Cs. _____

Mengetahui,
Ketua Program Studi Informatika – Program Sarjana
Fakultas Teknologi Industri
Universitas Islam Indonesia

(Dhomas Hatta Fudholi, S.T., M.Eng., Ph.D.)

HALAMAN PERNYATAAN KEASLIAN TUGAS AKHIR

Yang bertanda tangan di bawah ini:

Nama : Dicky Galuh Kurniawan

NIM : 20523234

Tugas akhir dengan judul:

PENGEMBANGAN PLATFORM MANAJEMEN TIM *E-SPORTS*: STUDI KASUS PADA KOMUNITAS *CLASH OF CLANS* DI INDONESIA

Menyatakan bahwa seluruh komponen dan isi dalam tugas akhir ini adalah hasil karya saya sendiri. Apabila di kemudian hari terbukti ada beberapa bagian dari karya ini adalah bukan hasil karya sendiri, tugas akhir yang diajukan sebagai hasil karya sendiri ini siap ditarik kembali dan siap menanggung risiko dan konsekuensi apapun.

Demikian surat pernyataan ini dibuat, semoga dapat dipergunakan sebagaimana mestinya.

Yogyakarta, 21 Januari 2026

(Dicky Galuh Kurniawan)

HALAMAN PERSEMBAHAN

Sujud syukur kupanjatkan kepada Allah SWT, pemilik segala ilmu dan takdir. Atas segala rahmat, karunia, dan hidayah-Nya yang tak ternilai, sehingga penulis diberi kekuatan, kesabaran, dan kelapangan hati untuk menyelesaikan perjalanan panjang ini. Karya sederhana ini penulis persembahkan kepada:

Bapak Samsudin dan Ibu Mase Tercinta Dua pilar terkuat dalam hidupku. Terima kasih atas setiap tetes keringat, doa yang dilangitkan di setiap sujud, serta kasih sayang tanpa syarat yang menjadi bahan bakar semangat penulis. Pencapaian ini adalah bukti kecil bakti penulis, meski takkan pernah cukup membalas pengorbanan Bapak dan Ibu.

Lendra Ningrum Puspita Sari Wanita hebat yang kelak menjadi pendamping hidupku. Terima kasih telah menjadi rumah tempat pulang saat lelah, pendengar paling setia di kala keluh kesah, dan penyemangat utama agar penulis tidak berhenti di tengah jalan. Kehadiranmu membuat perjalanan ini terasa lebih ringan.

Diva Azia Husna Adikku tersayang, terima kasih atas keceriaan, candaan, dan dukungan moral yang selalu berhasil mencairkan suasana. Semoga keberhasilan ini bisa memotivasimu untuk melangkah lebih jauh.

Bapak Kholid Haryono, S.T., M.Kom. Dosen pembimbing sekaligus orang tua kedua di kampus. Terima kasih yang tak terhingga bukan hanya atas bimbingan akademisnya, tetapi juga atas kesabaran, nasihat, dan kepedulian Bapak yang luar biasa. Terima kasih telah menjadi sosok yang membantu penulis bangkit dari masa-masa terberat dan pulih dari keterpurukan mental. Tanpa dukungan Bapak, penulis mungkin tidak akan mampu berdiri tegak menyelesaikan karya ini.

Segenap Keluarga Besar Terima kasih atas doa, dukungan, dan kehangatan kekeluargaan yang selalu menyertai setiap langkah penulis.

Almamater Universitas Islam Indonesia Tempat penulis menimba ilmu, menempa diri, dan menemukan keluarga baru.

Semoga skripsi ini menjadi wujud rasa cinta, hormat, dan terima kasih penulis atas semua ketulusan yang telah kalian berikan.

HALAMAN MOTO

"*If* I have seen further, it is by standing on the shoulders of giants."
(Jika aku melihat lebih jauh, itu karena aku berdiri di atas bahu para raksasa.)
– Sir Isaac Newton

"Bagi ilmuwan, tidak ada yang lebih berharga daripada mewariskan ilmu yang bermanfaat bagi orang lain, yang akan tetap hidup dan berkembang setelah penulisnya tiada."
– Muhammad ibn Musa Al-Khawarizmi (Mukadimah Kitab Al-Jabr wa'l-Muqabala)

Kode adalah puisi logika yang ditulis untuk memecahkan masalah manusia. Skripsi ini adalah bukti kecil bahwa algoritma yang dirancang dengan hati dapat menciptakan harmoni dalam sebuah komunitas.

Teruslah belajar, teruslah berkarya, dan jadilah bermanfaat.

KATA PENGANTAR

Alhamdulillahi rabbil ‘alamin, segala puji dan syukur penulis panjatkan ke hadirat Allah SWT, Tuhan semesta alam yang Maha Pengasih lagi Maha Penyayang. Atas limpahan rahmat, hidayah, dan karunia-Nya, penulis dapat menyelesaikan tugas akhir dengan judul “PENGEMBANGAN PLATFORM MANAJEMEN TIM *E-SPORTS*: STUDI KASUS PADA KOMUNITAS *CLASH OF CLANS* DI INDONESIA” dengan baik sebagai salah satu syarat untuk memperoleh gelar Sarjana Informatika di Universitas Islam Indonesia.

Shalawat serta salam semoga senantiasa tercurah kepada junjungan Nabi Muhammad SAW, yang telah membawa umat manusia dari zaman kegelapan menuju zaman yang penuh dengan cahaya ilmu dan iman.

Penyusunan tugas akhir ini tentunya tidak terlepas dari bantuan, bimbingan, dukungan, dan doa dari berbagai pihak. Untuk itu, penulis ingin menyampaikan terima kasih dan penghargaan yang setinggi-tingginya kepada:

1. Allah SWT, yang selalu memberikan kekuatan, kesabaran, dan kelapangan hati dalam setiap langkah kehidupan penulis hingga detik ini.
2. Bapak Samsudin dan Ibu Mase tercinta, sebagai orang tua hebat yang senantiasa mencurahkan kasih sayang, doa tulus di setiap sujud, serta dukungan moril dan materil yang tak ternilai harganya. Keberhasilan ini penulis persembahkan untuk Bapak dan Ibu.
3. Lendra Ningrum Puspita Sari, tunangan penulis yang selalu menjadi pendengar setia, memberikan semangat tanpa henti, serta mendampingi penulis melewati masa-masa sulit selama pengerjaan skripsi ini.
4. Diva Azia Husna, adik penulis yang telah memberikan warna, keceriaan, dan motivasi bagi penulis untuk segera menyelesaikan studi.
5. Prof. Fathul Wahid, S.T., M.Sc., Ph.D., selaku Rektor Universitas Islam Indonesia.
6. Prof. Dr. Ir. Hari Purnomo, M.T., IPU, ASEAN.Eng., selaku Dekan Fakultas Teknologi Industri Universitas Islam Indonesia.
7. Dr. R. Teduh Dirgahayu, S.T., M.Sc., selaku Ketua Jurusan Informatika Fakultas Teknologi Industri Universitas Islam Indonesia.
8. Bapak Kholid Haryono, S.T., M.Kom., selaku dosen pembimbing tugas akhir, yang dengan sabar telah meluangkan waktu, tenaga, dan pikiran untuk memberikan bimbingan, arahan, serta ilmu yang sangat berharga sejak awal perancangan hingga selesainya laporan ini.

9. Bapak Ari Sujarwo, S.Kom., MIT. (Hons)., selaku dosen pembimbing akademik, atas segala nasihat dan motivasi yang diberikan selama masa studi penulis di UII.
10. Seluruh Dosen dan Staf Program Studi Informatika UII, atas segala ilmu, dedikasi, dan pelayanan yang telah diberikan selama penulis menuntut ilmu.
11. Riza Elfany, teman satu kos yang telah menjadi saudara seperjuangan, tempat berbagi keluh kesah, dan saling menyemangati dalam kehidupan sehari-hari di perantauan.
12. Keluarga besar penulis yang selalu mendoakan kesuksesan penulis.
13. Komunitas *Clash of Clans* Indonesia yang telah bersedia menjadi responden dan memberikan wawasan berharga dalam pengembangan sistem ini.

Penulis menyadari bahwa tugas akhir ini masih jauh dari kesempurnaan. Oleh karena itu, penulis sangat mengharapkan saran dan kritik yang membangun guna perbaikan di masa mendatang.

Akhir kata, penulis berharap karya ini dapat memberikan manfaat dan menjadi kontribusi positif bagi perkembangan teknologi di bidang *e-sports* dan manajemen komunitas, serta menjadi amal jariyah bagi kita semua.

Wassalamualaikum Warahmatullahi Wabarakatuh.

Yogyakarta, 21 Januari 2026

(Dicky Galuh Kurniawan)

SARI

Perkembangan industri *e-sports* di Indonesia, khususnya pada komunitas gim strategi *Clash of Clans*, menghadapi tantangan manajerial yang signifikan di tingkat akar rumput (*grassroots*). Pengelolaan organisasi klan (tim) saat ini masih didominasi oleh metode manual yang terfragmentasi, menyebabkan inefisiensi administrasi bagi pemimpin komunitas. Selain itu, proses rekrutmen anggota sering kali tidak tepat sasaran akibat ketiadaan alat penyaringan (*filtering*) yang presisi serta maraknya krisis kepercayaan (*trust issue*) terkait validitas akun pemain. Penelitian ini bertujuan untuk mengembangkan "ClashHub", sebuah platform manajemen tim *e-sports* terintegrasi yang mampu mengatasi permasalahan tersebut melalui pendekatan teknologi.

Metodologi pengembangan sistem yang digunakan adalah Metode *Prototype* dengan pendekatan iteratif, yang memungkinkan penyesuaian fitur berdasarkan umpan balik pengguna secara berkelanjutan. Sistem dibangun berbasis *web* dengan desain antarmuka *Mobile-First* menggunakan kerangka kerja Next.js dan basis data Firebase. Arsitektur sistem mengintegrasikan *Clash of Clans API* untuk agregasi data statistik waktu nyata (*real-time*) dan teknologi Google Gemini AI (*Generative AI*) untuk memberikan analisis strategi tekstual. Validasi sistem dilakukan melalui pengujian fungsional (Black Box), pengujian usabilitas (*System Usability Scale*), dan validasi akseptansi pengguna (*User Acceptance Testing*).

Temuan penelitian menunjukkan bahwa implementasi fitur *Smart Sync* berhasil memangkas waktu administrasi klan secara signifikan dibandingkan metode manual. Penerapan algoritma pengurutan (*sorting*) berdasarkan parameter Skor Reputasi dan Level *Town Hall* terbukti meningkatkan akurasi dalam pencarian tim. Fitur AI Strategy Assistant dinilai efektif memberikan saran taktis yang kontekstual melalui teknik penyuntikan data fakta klan. Hasil pengujian usabilitas terhadap 15 responden menghasilkan skor rata-rata 86.2 (*Excellent*), mengindikasikan bahwa sistem memiliki tingkat penerimaan dan kemudahan penggunaan yang tinggi bagi komunitas target.

Kata kunci: Manajemen *E-Sports*, Metode *Prototype*, *Generative AI*, Agregasi Data, *Clash of Clans*.

GLOSARIUM

Access Control	Mekanisme pembatasan hak akses pengguna dalam suatu sistem.
Actor	Entitas berupa pengguna manusia atau sistem lain yang berinteraksi langsung dengan aplikasi.
Adjective *Rating*	Sistem penilaian kualitatif (berupa kata sifat, seperti *Excellent* atau *Good*) yang didasarkan pada konversi skor angka kuantitatif untuk memudahkan interpretasi kelayakan sistem.
Alternating Items	Teknik penyusunan pernyataan dalam kuesioner (seperti *SUS*) secara selang-seling antara pernyataan positif dan negatif untuk mencegah bias respons dari pengguna.
Auto-scaling	Kemampuan infrastruktur komputasi awan untuk menyesuaikan alokasi sumber daya secara otomatis sesuai dengan beban trafik yang masuk.
BaaS (*Backend-as-a-Service*)	Model layanan komputasi awan di mana pengembang menyewa infrastruktur *backend* siap pakai sehingga tidak perlu membangun peladen dari nol.
Badge	Lencana visual yang berfungsi sebagai identitas unik sebuah klan.
Behavioral Testing	Pendekatan pengujian perangkat lunak yang berfokus pada evaluasi perilaku operasional dan fungsionalitas sistem (sinonim dari *Black Box Testing*).
Black Box Testing	Pengujian perangkat lunak yang berfokus pada fungsionalitas antarmuka sistem tanpa melihat struktur kode internalnya.
Bracket	Bagan atau skema pertandingan dalam sebuah turnamen yang menunjukkan alur kompetisi dari awal hingga final.
Breakpoint	Titik batas ukuran resolusi layar di mana tata letak antarmuka *web* akan berubah secara adaptif (misal: batas transisi antara tampilan ponsel dan tablet).
Bundle Size	Total ukuran berkas kode (HTML/*CSS*/JS) yang harus diunduh oleh peramban pengguna saat pertama kali membuka aplikasi *web*.
Caching	Mekanisme penyimpanan salinan data sementara untuk mempercepat proses pengambilan data dan mengurangi beban akses langsung ke peladen utama.

Casual gaming	Gaya bermain gim yang santai untuk sekadar rekreasi, tanpa orientasi atau target kompetitif yang ketat.
Clan War	Fitur perang antar klan dalam gim *Clash of Clans* di mana dua klan saling menyerang untuk mengumpulkan bintang terbanyak.
Clan War Leagues (*CWL*)	Sistem liga kompetitif bulanan dalam *Clash of Clans* di mana sebuah klan bertanding selama satu minggu berturut-turut melawan tujuh klan lainnya.
Compile-time	Fase di mana kode program sedang ditulis atau dikompilasi oleh sistem, sebelum aplikasi tersebut dijalankan oleh pengguna.
Dashboard	Tampilan antarmuka visual yang merangkum informasi penting dan metrik kinerja dalam satu layar agar mudah dipantau.
Decision Maker	Pihak atau individu yang memiliki wewenang untuk mengambil keputusan strategis dalam suatu organisasi/klan.
Decision Node	Titik percabangan dalam diagram alur (*Activity Diagram*) yang merepresentasikan pengambilan keputusan berdasarkan kondisi logika tertentu (*If*/*Else*).
Digital Footprint	Jejak digital atau rekam jejak aktivitas yang ditinggalkan oleh pengguna di dunia maya.
Endpoint API	Titik akses spesifik berupa *URL* yang digunakan sebagai jalur komunikasi dan pertukaran data antar sistem perangkat lunak.
Evolusioner	Sifat pengembangan perangkat lunak yang tumbuh dan disempurnakan secara bertahap dari bentuk yang paling sederhana menjadi lebih kompleks.
Federated Identity	Sistem manajemen identitas yang memungkinkan pengguna melakukan *login* ke sebuah aplikasi menggunakan kredensial akun dari pihak ketiga yang terpercaya (seperti Google Sign-In).
Filtering	Proses penyaringan kumpulan data untuk hanya menampilkan informasi yang memenuhi kriteria atau parameter spesifik tertentu.
Firebase	Platform pengembangan aplikasi dari Google yang menyediakan layanan basis data (*Cloud* Firestore), autentikasi, dan layanan peladen tanpa peladen (*serverless*).
Forecasting	Peramalan atau prediksi kejadian di masa depan yang didasarkan pada analisis pola data historis.

Grassroots Akar rumput; merujuk pada komunitas pemain biasa atau amatir di tingkat dasar, bukan bagian dari organisasi *e-sports* profesional yang memiliki sponsor.

Hopper Istilah komunitas untuk pemain yang sering berpindah-pindah klan dalam waktu singkat (kutu loncat) hanya untuk mencari keuntungan pribadi.

Hybrid Rendering Teknik dalam pengembangan *web* modern yang menggabungkan proses *rendering* di sisi peladen (*Server-Side*) dengan interaktivitas cepat di sisi klien (*Client-Side*) untuk mengoptimalkan performa.

Information Overload Kondisi di mana seseorang menerima terlalu banyak informasi mentah sehingga justru menyulitkan mereka dalam mengambil keputusan yang tepat.

Interface Kontrak atau struktur definisi data yang ketat (biasanya dalam bahasa TypeScript) untuk menjamin konsistensi tipe data dalam kode program.

Iteratif Pendekatan siklus pengembangan perangkat lunak yang dilakukan secara berulang-ulang untuk menghasilkan penyempurnaan fitur secara bertahap berdasarkan umpan balik.

JSON (*JavaScript Object Notation*) Format standar pertukaran data yang ringan dan terstruktur, mudah dibaca oleh manusia maupun mesin.

Leader/*Co-Leader* Jabatan manajerial tertinggi dalam sebuah klan yang memiliki wewenang penuh untuk mengatur rekrutmen anggota dan strategi perang (Ketua/Wakil Ketua).

Likert Scale (Skala Likert) Skala psikometrik yang umum digunakan dalam kuesioner untuk mengukur tingkat persetujuan responden terhadap suatu pernyataan (biasanya dalam rentang nilai 1 hingga 5).

Listener Mekanisme kode program yang bertugas "mendengarkan" atau memantau perubahan data pada basis data dan secara otomatis memicu pembaruan di antarmuka (*real-time*).

LLM (Large Language Model) Model kecerdasan buatan (*Generative AI*) berbasis jaringan saraf tiruan yang dilatih menggunakan himpunan teks berskala besar untuk memahami, meringkas, dan menghasilkan bahasa alami (contoh: Google Gemini).

Actor Modern Web Application Aplikasi berbasis *web* yang dibangun menggunakan kerangka kerja mutakhir, dirancang untuk memiliki kecepatan, interaktivitas, dan pengalaman layaknya aplikasi bawaan (*native*) di PC atau ponsel.

Multimedia Aggregation Proses pengumpulan dan pengelompokan konten media (seperti video YouTube) dari berbagai sumber eksternal untuk disajikan ke dalam satu platform terpusat.

Native app Aplikasi seluler yang dibangun secara spesifik menggunakan bahasa asli sistem operasi tertentu (misal: Java/Kotlin untuk Android, Swift untuk iOS).

Next.js Kerangka kerja (*framework*) berbasis pustaka React yang digunakan untuk membangun antarmuka *web* yang sangat cepat, optimal, dan ramah mesin pencari.

NoSQL Sistem manajemen basis data yang fleksibel dan menyimpan data dalam format dokumen (seperti *JSON*), tidak menggunakan relasi tabel kaku (non-relasional) layaknya basis data SQL konvensional.

OAuth 2.0 Protokol standar industri yang digunakan secara luas untuk proses otorisasi akses dan pendelegasian identitas yang aman (misal: akses *login* via Google).

Primary Key Kunci utama atau identitas unik dalam sebuah basis data (*database*) yang berfungsi membedakan satu baris data dengan data lainnya.

Production Build Versi kompilasi akhir dari sebuah kode aplikasi perangkat lunak yang telah dioptimasi secara penuh (dikompresi) dan siap untuk didistribusikan kepada publik.

Profil Validator Deskripsi latar belakang, keahlian, dan kualifikasi dari pakar atau individu yang bertugas melakukan pengujian dan validasi kelayakan sistem.

Rate Limiting (*Rate Limit*) Pembatasan jumlah akses atau permintaan maksimal yang diizinkan ke sebuah peladen *API* dalam kurun waktu tertentu untuk mencegah beban berlebih atau penyalahgunaan.

Real-time Pemrosesan, sinkronisasi, atau penyajian data yang terjadi secara instan dengan jeda waktu yang sangat minimal, hampir bersamaan dengan kejadian aslinya.

Refactoring Proses merestrukturisasi atau merapikan arsitektur penulisan kode program dari dalam, tanpa mengubah perilaku atau fungsi antarmuka asli dari kode tersebut.

Reliabel (Reliabilitas) Tingkat keandalan atau konsistensi suatu alat ukur; sejauh mana hasil pengukuran tetap sama dan stabil jika pengujian diulang pada kondisi yang serupa.

Requirements gathering Proses identifikasi, pengumpulan, dan pendefinisian kebutuhan teknis sebuah sistem dari pengguna akhir atau pemangku kepentingan.

RESTful API Standar arsitektur antarmuka pemrograman aplikasi yang menggunakan protokol komunikasi HTTP untuk melakukan pertukaran data antar sistem di jaringan *web*.

Roster management Proses pengaturan dan pengelolaan daftar susunan pemain aktif di dalam sebuah tim.

Runtime Error Jenis kesalahan (*bug*) atau kegagalan sistem yang baru terdeteksi pada saat aplikasi sedang berjalan atau digunakan oleh pengguna akhir.

SDLC (*Software* Development Life Cycle) Kerangka kerja metodologis yang mendefinisikan seluruh fase siklus hidup pengembangan perangkat lunak, mulai dari tahap perencanaan, perancangan, pembuatan, hingga pemeliharaan.

SEO (Search Engine Optimization) Serangkaian teknik optimasi teknis dan konten agar sebuah situs *web* lebih mudah ditemukan dan diindeks secara prioritas oleh mesin pencari seperti Google.

Serverless Model arsitektur komputasi awan di mana pengembang tidak perlu lagi melakukan instalasi, konfigurasi, atau manajemen peladen fisik, karena semua infrastruktur dikelola sepenuhnya oleh penyedia *cloud*.

Snapshot Tangkapan status atau salinan wujud data yang diambil pada satu titik waktu spesifik.

Software Quality Assurance (SQA) Serangkaian aktivitas sistematis dan terencana yang dilakukan untuk memastikan bahwa produk perangkat lunak yang dibangun memenuhi standar kualitas yang ditetapkan.

Sorting Proses pengurutan daftar data berdasarkan parameter terukur tertentu, baik dari nilai tertinggi ke terendah (descending) maupun sebaliknya (ascending).

Stack teknologi	Kumpulan kombinasi perangkat lunak, kerangka kerja (*framework*), dan bahasa pemrograman yang digunakan secara bersamaan untuk membangun sebuah aplikasi.
Stakeholder	Pemangku kepentingan atau seluruh pihak yang memiliki keterlibatan, kepentingan, dan pengaruh dalam kesuksesan sebuah proyek (misal: pengguna, klien, pengembang).
Static Typing	Fitur dalam bahasa pemrograman (seperti TypeScript) di mana jenis/tipe data dari sebuah variabel harus didefinisikan secara tegas di awal dan akan divalidasi dengan ketat sebelum kode dapat dijalankan.
Strategi	Pendekatan atau rencana komprehensif yang menjelaskan bagaimana suatu proses (seperti eksekusi pengujian sistem) akan dilakukan dan diukur keberhasilannya.
Strategic insights	Wawasan strategis atau pemahaman mendalam yang didapatkan dari hasil analisis data untuk mendukung pengambilan keputusan manajerial yang lebih baik.
Superuser	Tingkatan pengguna yang memiliki hak akses tak terbatas terhadap seluruh fitur dan data di dalam suatu sistem.
Superset	Bahasa pemrograman yang dirancang dengan mencakup seluruh sintaksis dan fitur dari bahasa induknya (misal: JavaScript), ditambah dengan fungsionalitas baru (misal: TypeScript).
Swift Trust	Bentuk kepercayaan yang terbangun dengan sangat cepat di dalam tim virtual tanpa interaksi fisik, didasarkan pada atribut visual yang langsung terlihat (seperti label data atau reputasi).
Team finding	Fitur, modul, atau proses pencarian tim/klan *e-sports* yang selaras dengan kriteria dan kemampuan seorang pemain.
Town Hall (*TH*)	Bangunan sentral di markas pemain dalam gim *Clash of Clans* yang berfungsi sebagai indikator mutlak penentu level dan batas maksimal kekuatan akun pemain tersebut.
Toxicity	Perilaku beracun, negatif, atau merusak keharmonisan yang dilakukan oleh pemain di dalam lingkungan gim daring.
Tree-Shaking	Proses optimasi otomatis pada tahap produksi (*Production Build*) yang bertujuan mendeteksi dan menghapus modul/kode (seperti kelas *CSS*) yang tidak dipakai guna memperkecil ukuran akhir berkas aplikasi.

Trust	Tingkat keyakinan dan kepercayaan antar anggota yang diperlukan untuk mendukung kolaborasi dalam sebuah tim virtual.
UML (*Unified Modeling Language*)	Bahasa standar visual grafis yang digunakan secara global dalam industri rekayasa perangkat lunak untuk merancang, memvisualisasikan, dan mendokumentasikan spesifikasi sistem.
Usability Testing	Pengujian tingkat kegunaan yang dilakukan untuk mengevaluasi seberapa mudah, efisien, dan nyaman sebuah aplikasi saat dioperasikan oleh pengguna akhir.
Utility-First	Paradigma desain antarmuka (seperti pada Tailwind *CSS*) yang mengandalkan komposisi kelas-kelas utilitas berukuran kecil untuk membangun gaya desain langsung di dalam markah HTML, bukan melalui berkas *CSS* terpisah.
Validasi	Proses evaluasi perangkat lunak di tahap akhir untuk memastikan bahwa sistem yang dibangun sudah benar-benar sesuai dengan kebutuhan dan keinginan pengguna (fokus: "Are we building the right product?").
Verifikasi	Proses evaluasi komponen sistem untuk memastikan bahwa perangkat lunak dibangun dengan teknik dan spesifikasi yang benar secara internal (fokus: "Are we building the product right?").
War Weight	Angka atau nilai pembobotan tersembunyi yang mengukur kekuatan total pertahanan/serangan sebuah akun, digunakan sebagai dasar dalam algoritma matchmaking atau parameter pengurutan.
Wireframe	Sketsa atau kerangka visual dasar dengan tingkat presisi rendah (low-fidelity) yang menunjukkan rancangan tata letak elemen antarmuka halaman *web* sebelum desain akhir dibuat.
Workflow	Alur kerja sistematis atau urutan logis langkah-langkah prosedural yang harus dilalui dalam menjalankan suatu tugas atau proses bisnis tertentu.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan industri *Electronic Sports* (*e-sports*) di Indonesia menunjukkan pertumbuhan yang signifikan dalam satu dekade terakhir. *E-sports* telah bertransformasi dari sekadar aktivitas hobi rekreasional menjadi ekosistem digital yang bernilai ekonomi. Pergeseran pandangan masyarakat yang mulai melihat gim kompetitif sebagai aktivitas yang positif didukung oleh infrastruktur turnamen yang semakin memadai. Industri gim di Indonesia mencatatkan potensi pendapatan yang besar dengan proyeksi pertumbuhan yang terus meningkat, menempatkan sektor *e-sports* sebagai salah satu bagian menarik dalam ekonomi kreatif digital nasional (Newzoo, 2024; Kementerian Kominfo, 2024).

Indonesia memiliki karakteristik unik dalam demografi pemain gim yang didominasi oleh penggunaan perangkat seluler (*mobile*). Didorong oleh penetrasi *smartphone* yang luas dan akses internet yang terjangkau, Indonesia menjadi salah satu pasar *mobile gaming* yang besar dengan lebih dari 150 juta pemain aktif (We Are Social, 2024). Berbeda dengan pasar negara maju yang mungkin lebih berbasis pada PC atau konsol, ekosistem *e-sports* di Indonesia sangat berpusat pada platform *mobile*. Kondisi ini menciptakan kebutuhan akan teknologi pendukung yang responsif dan mudah diakses melalui perangkat seluler (*mobile-first*) untuk menopang aktivitas komunitas.

Di tengah banyaknya gim baru yang bermunculan, *Clash of Clans* (CoC) menunjukkan ketahanan sebagai salah satu gim strategi yang masih mempertahankan basis pemainnya. Meskipun telah beroperasi lebih dari satu dekade, data statistik mencatat bahwa hingga tahun 2024, gim ini masih memiliki lebih dari 40 juta pengguna aktif bulanan secara global (ActivePlayer.io, 2024). Fakta ini menunjukkan bahwa komunitas CoC di Indonesia masih eksis dan justru semakin matang dengan struktur klan yang lebih terorganisir, menciptakan kebutuhan akan pengelolaan komunitas yang lebih baik.

Secara karakteristik permainan, *Clash of Clans* memiliki perbedaan dibandingkan genre populer lainnya seperti *MOBA* (*Multiplayer Online Battle Arena*) yang mengandalkan refleks cepat. CoC khususnya pada tingkat *Town Hall* (*TH*) tinggi menitikberatkan pada manajemen makro, perencanaan, dan kerja sama tim (Klan). Keberhasilan sebuah klan sangat bergantung pada koordinasi strategi perang (*Clan War*), manajemen donasi, dan kekompakan anggota.

Kompleksitas manajerial inilah yang menuntut adanya media pendukung yang lebih terstruktur dibandingkan sekadar grup obrolan media sosial.

Berdasarkan observasi awal dan wawancara dengan pemain aktif komunitas, ditemukan permasalahan mendasar terkait inefisiensi dalam manajemen tim dan rekrutmen. Banyak pemain kesulitan menemukan klan yang memiliki visi selaras, misalnya pemain kompetitif yang terjebak di klan santai, atau sebaliknya. Hal ini sering memicu keluarnya anggota dari klan (*turnover*) yang mengganggu stabilitas persiapan klan dalam menghadapi kompetisi liga bulanan (*Clan War Leagues*). Selain itu, fitur pencarian klan di dalam gim (*in-game*) dinilai masih memiliki keterbatasan filter yang spesifik. Pemimpin klan sering kali kesulitan melakukan penyaringan (*filtering*) dan pengurutan (*sorting*) anggota potensial berdasarkan parameter spesifik, seperti skor *Town Hall* (*TH*) maupun reputasi pemain, yang sebenarnya krusial untuk membangun komposisi tim yang solid.

Permasalahan selanjutnya berkaitan dengan pengelolaan data klan. Fitur bawaan gim seringkali hanya menampilkan data sesaat (*snapshot*) tanpa menyimpan rekam jejak historis jangka panjang secara mendetail, seperti riwayat performa perang atau keaktifan donasi dari waktu ke waktu. Ketiadaan agregasi data historis yang terorganisir ini menyulitkan pemimpin klan dalam mengambil keputusan objektif, misalnya dalam menentukan promosi jabatan atau seleksi anggota untuk perang. Akibatnya, pengambilan keputusan sering kali didasarkan pada intuisi atau perkiraan subjektif karena tidak adanya sistem pendukung keputusan yang berbasis data.

Untuk mengatasi hal tersebut, sebagian pengelola komunitas menggunakan alat bantu manual seperti *spreadsheet*. Namun, penggunaan alat berbasis *desktop* (seperti Excel) seringkali kurang praktis bagi pemain *mobile game* karena kendala antarmuka (*User Interface*) di layar ponsel yang kecil. Proses pencatatan manual juga rentan terhadap kesalahan manusia (*human error*) dan memakan waktu yang cukup lama. Selain itu, dalam penyelenggaraan turnamen komunitas tingkat amatir (*grassroots*), panitia sering menghadapi kendala dalam validasi data peserta dan pengelolaan bagan pertandingan yang masih dilakukan secara manual.

Penelitian terdahulu mengenai sistem informasi *e-sports* menunjukkan bahwa mayoritas penelitian masih berfokus pada analisis performa *gameplay* atau tim profesional, sementara aspek manajemen organisasi untuk komunitas amatir atau *grassroots* masih belum banyak dieksplorasi secara mendalam (Pérez-Rubio et al., 2024). Selain itu, komunitas *grassroots* sering kali diidentifikasi kekurangan infrastruktur digital yang memadai dibandingkan tim profesional (Toth et al., 2024).

Berdasarkan permasalahan tersebut, penelitian ini mengusulkan pengembangan sebuah platform berbasis *website* bernama "ClashHub" sebagai solusi bantu manajemen tim *e-sports* komunitas *Clash of Clans*. Mengingat kebutuhan komunitas yang dinamis serta perlunya iterasi fitur yang cepat untuk menyesuaikan dengan masukan pengguna, pengembangan sistem ini akan menggunakan metode *Prototype*. Pendekatan *Prototype* dipilih karena karakteristiknya yang iteratif memungkinkan pengembang dan pengguna (komunitas) untuk mengevaluasi fungsionalitas sistem sejak tahap awal, memastikan bahwa fitur-fitur yang dibangun benar-benar menjawab kebutuhan operasional klan sebelum sistem final dirilis (Pressman, 2014).

Platform ini dirancang untuk menyediakan fitur manajemen data klan yang terintegrasi langsung dengan *Application Programming Interface* (*API*) resmi dari pengembang gim (Supercell). Integrasi ini memungkinkan agregasi data statistik secara otomatis dan *real-time*, menjamin keakuratan informasi tanpa perlu *input* manual. Sistem akan menyediakan fitur pencarian tim dengan filter yang lebih mendalam, termasuk pengurutan berdasarkan skor *TH* dan reputasi. Selain itu, untuk mendukung proses pengambilan keputusan, platform ini akan dilengkapi dengan fitur asisten cerdas yang terintegrasi dengan *Generative AI* (Gemini) untuk memberikan rekomendasi strategi dan analisis berbasis data yang telah dikumpulkan. Diharapkan, purwarupa (*prototype*) yang dihasilkan dari penelitian ini dapat menjadi solusi alternatif yang efektif dalam meningkatkan efisiensi pengelolaan komunitas *e-sports Clash of Clans* di tingkat akar rumput.

1.2 Rumusan Masalah

Berdasarkan latar belakang masalah yang telah diuraikan, maka rumusan masalah dalam penelitian ini adalah:

a. Bagaimana merancang dan membangun platform pencarian tim (*team finding*) yang memiliki kemampuan penyaringan (*filtering*) dan pengurutan (*sorting*) berdasarkan parameter spesifik (seperti skor *Town Hall* dan reputasi) untuk mengatasi kesulitan pemain dalam menemukan klan yang sesuai?
b. Bagaimana mengembangkan sistem manajemen klan yang mengoptimalkan integrasi *Application Programming Interface* (*API*) resmi *Clash of Clans* untuk menyajikan agregasi data statistik *real-time* serta memanfaatkan *Generative AI* sebagai asisten pendukung keputusan bagi pemimpin klan?
c. Bagaimana menyediakan fitur pengelolaan turnamen yang dapat membantu panitia dalam memvalidasi data peserta secara otomatis guna meminimalisir kecurangan administrasi pada kompetisi tingkat komunitas?

1.3 Batasan Masalah

Agar pembahasan dalam penelitian ini lebih terarah dan tidak menyimpang dari tujuan utama, penulis menetapkan batasan masalah sebagai berikut:

a. Objek Penelitian: Sistem dikembangkan sebagai studi kasus untuk komunitas pemain gim *Clash of Clans* di Indonesia. Fungsionalitas utama berfokus pada manajemen data klan berbasis agregasi data, pencarian tim dengan kriteria yang dapat disesuaikan (*custom criteria*), dan administrasi turnamen sederhana. Sistem ini tidak memodifikasi *gameplay* asli ataupun membuat *bot* yang dapat memainkan gim secara otomatis, sehingga tidak melanggar ketentuan layanan (*Terms of Service*) pengembang gim.
b. Platform & Teknologi: Aplikasi dibangun berbasis *website* dengan desain antarmuka yang dioptimalkan untuk perangkat seluler (*mobile-first*). Pengembangan menggunakan kerangka kerja Next.js dan basis data Firebase. Aplikasi ini tidak tersedia dalam format aplikasi bawaan (*native app*) Android maupun iOS.
c. Integrasi Data: Data statistik klan dan pemain yang diolah meliputi level *Town Hall*, level *Hero*, skor reputasi, dan riwayat perang, yang diambil menggunakan integrasi resmi *Clash of Clans API*. Kelengkapan dan ketersediaan data bergantung pada batasan akses (*rate limit*) yang diberikan oleh penyedia *API*.
d. Fitur Kecerdasan Buatan (AI): Implementasi AI (menggunakan teknologi *Generative AI*) dibatasi fungsinya sebagai "Asisten Strategi" yang memberikan saran tekstual berdasarkan data yang tersedia. AI tidak berfungsi sebagai mesin prediksi hasil pertandingan (*forecasting*) ataupun sistem pengambil keputusan otomatis.
e. Manajemen Turnamen: Fitur turnamen berfokus pada aspek administrasi, yaitu validasi level akun peserta dan pembuatan bagan pertandingan (*bracket*) sederhana. Sistem tidak menyediakan layanan *live streaming* video, melainkan hanya menyediakan kolom untuk menyematkan tautan video dari platform eksternal (seperti YouTube).
f. Pengguna: Sistem ditujukan untuk penggunaan oleh pemimpin klan (*Leader*/*Co-Leader*), penyelenggara turnamen komunitas, dan pemain umum (publik), bukan untuk manajemen tim *e-sports* profesional tingkat dunia.

1.4 Tujuan Penelitian

Berdasarkan rumusan masalah yang telah ditetapkan, tujuan yang ingin dicapai dari penelitian ini adalah:

a. Merancang dan membangun platform pencarian tim (*team finding*) yang menyediakan fitur penyaringan (*filtering*) dan pengurutan (*sorting*) berdasarkan parameter spesifik (seperti skor *Town Hall* dan reputasi), agar pemain dapat menemukan klan yang sesuai dengan kriteria kompetitif maupun sosial mereka secara lebih akurat.
b. Mengembangkan sistem manajemen klan yang terintegrasi dengan *API* resmi *Clash of Clans* untuk menyajikan agregasi data statistik anggota secara *real-time*, serta mengimplementasikan fitur asisten berbasis *Generative AI* guna memberikan rekomendasi strategi yang mendukung pengambilan keputusan pemimpin klan.
c. Menyediakan fitur administrasi turnamen otomatis yang mampu memvalidasi data level peserta dan membuat bagan pertandingan (*bracket*), sehingga penyelenggaraan kompetisi komunitas menjadi lebih efisien, transparan, dan minim kesalahan administrasi.

1.5 Manfaat Penelitian

Hasil dari penelitian ini diharapkan dapat memberikan kontribusi dan manfaat bagi berbagai pihak, antara lain:

a. Bagi Penulis
    1. Sebagai sarana untuk mengimplementasikan ilmu Rekayasa Perangkat Lunak, khususnya dalam penerapan metode *Prototype* yang berfokus pada pengembangan iteratif sesuai kebutuhan pengguna.
    2. Menambah wawasan dan pengalaman teknis dalam mengintegrasikan *Application Programming Interface* (*API*) pihak ketiga dan teknologi *Generative AI* ke dalam sebuah sistem informasi manajemen berbasis *web*.
b. Bagi Pengguna (Komunitas *Clash of Clans*)
    1. Pemimpin Klan (*Leader*): Mendapatkan alat bantu yang menyajikan data agregat nan objektif untuk memantau perkembangan anggota, serta memperoleh wawasan strategis (*strategic insights*) untuk pengelolaan klan yang lebih profesional.
    2. Pemain (*Player*): Memudahkan proses pencarian klan yang presisi sesuai dengan preferensi (gaya bermain dan reputasi), serta memiliki profil statistik yang terverifikasi secara otomatis untuk kebutuhan rekrutmen.
    3. Penyelenggara Turnamen: Meningkatkan efisiensi waktu dalam verifikasi data peserta (validasi akun) dan penyusunan jadwal pertandingan, serta meminimalisir kesalahan pencatatan (*human error*).
c. Bagi Akademisi dan Ilmu Pengetahuan

1. Memberikan referensi mengenai pengembangan sistem informasi manajemen untuk komunitas *e-sports* tingkat amatir (*grassroots*) yang masih jarang dibahas dibandingkan *e-sports* profesional.
2. Menambah literatur terkait pemanfaatan data *game API* dan teknologi *web* modern (seperti Next.js dan Firebase) dalam studi kasus pengembangan perangkat lunak interaktif.

1.6 Metodologi Penelitian

Dalam pengembangan sistem "ClashHub", metode penelitian yang digunakan adalah metode *Prototype*. Menurut Pressman (2014), paradigma *prototyping* sangat sesuai digunakan ketika pelanggan (pengguna) mendefinisikan serangkaian tujuan umum perangkat lunak, tetapi tidak mengidentifikasi kebutuhan *input*, pemrosesan, atau *output* secara terperinci di awal. Metode ini memungkinkan pengembang dan pengguna untuk saling berinteraksi dalam proses pembuatan model sistem yang berjalan. Adapun tahapan pengembangan sistem mengacu pada model *Prototype* yang terdiri dari lima tahapan utama, yaitu:

a. Komunikasi (*Communication*)

   Tahap ini merupakan langkah awal di mana penulis melakukan identifikasi masalah dan kebutuhan melalui observasi serta wawancara dengan perwakilan komunitas *Clash of Clans* (pemimpin klan dan pemain). Tujuannya adalah untuk memahami tujuan umum perangkat lunak, khususnya kebutuhan akan sistem pencarian tim dengan filter yang spesifik dan manajemen data klan yang lebih terorganisir.

b. Perencanaan Cepat (*Quick Plan*)

   Setelah kebutuhan awal teridentifikasi, penulis melakukan perencanaan cepat yang berfokus pada representasi aspek-aspek perangkat lunak yang akan terlihat oleh pengguna akhir. Tahap ini mencakup penentuan teknologi yang akan digunakan (Next.js dan Firebase) serta penjadwalan pengembangan fitur-fitur utama, termasuk fitur agregasi data dan asisten strategi.

c. Pemodelan Desain Cepat (*Modeling Quick Design*)

   Pada tahap ini, penulis membuat rancangan antarmuka (*User Interface*) dan alur sistem secara garis besar. Fokus utama adalah pada tata letak menu, fitur penyaringan (*filtering*) pencarian, format tampilan data statistik, dan alur pendaftaran turnamen. Desain ini dibuat untuk memberikan gambaran visual kepada pengguna mengenai bentuk aplikasi yang akan dibangun.

d. Pembentukan Prototipe (*Construction of Prototype*)

Berdasarkan desain cepat yang telah dibuat, penulis mulai melakukan pengkodean (*coding*) untuk membangun sistem yang sebenarnya. Pada tahap ini, fitur-fitur inti seperti integrasi *API Clash of Clans* untuk agregasi data, manajemen basis data klan, dan fitur asisten berbasis *Generative AI* mulai diimplementasikan menjadi perangkat lunak yang dapat dijalankan.

e. Penyerahan dan Umpan Balik (*Deployment Delivery & Feedback*)

Prototipe yang telah dibangun kemudian diserahkan kepada pengguna (perwakilan komunitas) untuk diuji coba. Pengujian dilakukan menggunakan metode *Black Box Testing* untuk memastikan fungsionalitas berjalan lancar, dan *Usability Testing* untuk mengukur kemudahan penggunaan. Umpan balik yang diberikan oleh pengguna pada tahap ini akan menjadi bahan evaluasi untuk penyempurnaan sistem sebelum dianggap layak sebagai solusi akhir.

1.7 Sistematika Penulisan

Untuk memberikan gambaran menyeluruh dan terstruktur mengenai penyusunan laporan tugas akhir ini, materi pembahasan disusun dalam lima bab dengan sistematika sebagai berikut:

BAB 1: PENDAHULUAN

Bab ini menguraikan latar belakang masalah mengenai pentingnya sistem manajemen komunitas *e-sports*, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, serta metodologi pengembangan sistem yang digunakan.

BAB 2: LANDASAN TEORI

Bab ini memuat kajian pustaka dari penelitian terdahulu yang relevan serta landasan teori yang menjadi fondasi pengembangan sistem. Pembahasan mencakup teori manajemen komunitas *e-sports*, konsep *Application Programming Interface* (*API*), metode *Prototype*, serta teknologi pengembangan *web* modern seperti Next.js, Firebase, dan *Generative AI*.

BAB 3: METODOLOGI PENELITIAN

Bab ini menjelaskan secara rinci mengenai alur penelitian yang dilakukan berdasarkan tahapan metode *Prototype*. Bab ini mencakup analisis kebutuhan sistem (spesifikasi fungsional), perancangan sistem menggunakan diagram *UML* (*Unified Modeling Language*), serta perancangan antarmuka (*User Interface*) aplikasi.

BAB 4: IMPLEMENTASI DAN PEMBAHASAN

Bab ini memaparkan hasil pembangunan sistem "ClashHub", mulai dari konfigurasi lingkungan pengembangan hingga implementasi kode program untuk fitur-fitur utama. Bab ini juga menyajikan hasil pengujian sistem yang telah dilakukan terhadap pengguna, beserta analisis pembahasannya untuk memvalidasi apakah sistem telah berjalan sesuai tujuan.

BAB 5: PENUTUP

Bab ini berisi kesimpulan akhir dari seluruh rangkaian penelitian yang telah dilakukan, serta saran-saran konstruktif untuk pengembangan fitur sistem di masa mendatang.

DAFTAR PUSTAKA

Berisi daftar referensi buku, jurnal ilmiah, dan sumber valid lainnya yang digunakan sebagai acuan dalam penyusunan laporan ini.

LAMPIRAN

Berisi dokumen pendukung seperti pedoman wawancara, kuesioner pengujian, dan dokumentasi pendukung lainnya.

BAB II
LANDASAN TEORI

2.1 Tinjauan Pustaka

Kajian literatur merupakan bagian penting dalam penelitian untuk memetakan perkembangan ilmu pengetahuan terkini terkait topik yang sedang diteliti. Dalam konteks pengembangan sistem informasi untuk ekosistem *e-sports*, penulis melakukan analisis terhadap beberapa penelitian terdahulu yang relevan, khususnya yang berfokus pada manajemen turnamen, pengelolaan organisasi tim, dan integrasi teknologi dalam komunitas gim.

Penelitian mengenai sistem informasi manajemen turnamen telah dilakukan sebelumnya, salah satunya membahas rancang bangun sistem untuk turnamen *Mobile Legends*. Penelitian tersebut dilatarbelakangi oleh inefisiensi proses administrasi yang masih manual, dan solusinya adalah pengembangan sistem berbasis *web* dengan fitur pendaftaran daring serta pembuatan bagan pertandingan otomatis. Hasil pengujian menunjukkan peningkatan efisiensi waktu administrasi, namun sistem ini hanya berfokus pada manajemen acara sesaat tanpa fitur pengelolaan tim jangka panjang (Pratama & Wibowo, 2021).

Studi lain dilakukan terkait digitalisasi manajemen *event e-sports* di wilayah Kabupaten Sumbawa. Sistem yang dibangun bertujuan untuk mendata kegiatan *e-sports* daerah yang sebelumnya tidak terorganisir. Meskipun berhasil mendigitalkan proses pendaftaran, sistem ini memiliki kelemahan karena tidak terintegrasi secara *real-time* dengan data gim, sehingga validasi data statistik pemain (seperti *Rank* atau Level) masih harus dilakukan secara manual oleh panitia dan rentan terhadap kesalahan pencatatan (Yunanri & Measer, 2022).

Selain fokus pada turnamen, terdapat juga penelitian yang menggabungkan informasi kompetisi dengan lokasi fisik, seperti studi mengenai sistem informasi turnamen dan warung kopi *gaming* di Pontianak. Fitur utama sistem ini meliputi peta lokasi, profil tempat, dan jadwal turnamen lokal. Kendati demikian, sistem ini lebih berfungsi sebagai portal informasi atau direktori daripada sistem manajerial fungsional yang dapat membantu ketua tim dalam mengelola anggota atau strategi (Arham et al., 2024).

Analisis Kesenjangan (*Gap Analysis*)

Berdasarkan tinjauan terhadap penelitian-penelitian di atas, ditemukan celah penelitian (*research gap*) yang dapat diisi. Mayoritas sistem yang ada saat ini cenderung berfokus pada

"Manajemen *Event*", yaitu hanya melayani kebutuhan penyelenggaraan turnamen sesaat. Belum banyak penelitian yang secara spesifik mengembangkan sistem untuk kebutuhan manajemen internal klan atau komunitas di tingkat akar rumput (*grassroots*) yang terintegrasi langsung dengan data permainan. Masalah seperti kesulitan validasi data pemain secara otomatis, serta ketiadaan fitur penyaringan anggota yang mendetail, belum terakomodasi sepenuhnya oleh solusi yang ada.

Kebaruan Penelitian (*Novelty*)

Penelitian ini bertujuan mengisi celah tersebut dengan mengembangkan "ClashHub", sebuah platform manajemen tim berbasis *web*. Kebaruan (*novelty*) yang ditawarkan dalam penelitian ini meliputi:

a. Integrasi *API* Resmi: Penggunaan *Application Programming Interface* (*API*) *Clash of Clans* untuk agregasi data statistik pemain secara otomatis dan *real-time*, meminimalisir *input* manual dan manipulasi data.
b. Mekanisme Pencarian Presisi: Penerapan fitur penyaringan (*filtering*) dan pengurutan (*sorting*) berdasarkan parameter spesifik (seperti skor *Town Hall* dan reputasi) untuk mempertemukan pemain dan klan yang sesuai.
c. Fitur Asisten Cerdas: Pemanfaatan teknologi *Generative AI* sebagai alat bantu analisis strategi tekstual bagi pemimpin klan, berbeda dengan sistem konvensional yang hanya menampilkan angka.
d. Metode Pengembangan: Penggunaan metode *Prototype* yang melibatkan partisipasi aktif komunitas dalam iterasi fitur agar sesuai dengan kebutuhan lapangan.

Perbandingan fitur antara penelitian terdahulu dengan sistem yang dikembangkan dalam penelitian ini dapat dilihat pada Tabel 2.1 berikut:

Tabel 2.1 Perbandingan Penelitian Terdahulu dan Penelitian Sekarang

Peneliti / Tahun	**Topik Penelitian**	**Metode & Teknologi**	**Fitur Utama**	**Perbedaan / Kebaruan (*Novelty*)**
Pratama & Wibowo (2021)	Manajemen Turnamen *Mobile Legends*	*Waterfall* (PHP)	Pendaftaran Daring, Verifikasi Pembayaran, dan *Bracket Generator*	Fokus hanya pada *event* sesaat, Tidak ada fitur manajemen klan, dan Tidak ada integrasi *API* gim.
Yunanri & Measer (2022)	Manajemen *Event E-Sport* (ESI Sumbawa)	*Waterfall* (*PHP Native*)	Pendaftaran *Event*, Jadwal Pertandingan, dan Laporan Kegiatan	Validasi data pemain manual (Tanpa *API*), Tidak ada fitur analisis performa, dan Rentan *human error*.
Arham et al. (2024)	Turnamen *E-Sports* & Warkop *Gaming*	*Prototype* (*Web*)	Info Turnamen, Peta Lokasi, dan Profil *Venue*	Bersifat portal informasi, bukan manajerial, Tidak ada fitur manajemen tim, dan Tidak ada integrasi data statistik.
(Penelitian Ini)	Platform Manajemen Tim *E-Sports* (ClashHub)	*Prototype* (Next.js + Firebase)	Asisten Strategi (*Generative AI*), Integrasi *API* CoC, Pencarian Tim (*Sorting* & *Filtering*), dan Manajemen Turnamen	Pemanfaatan *API* untuk data *real-time*, Fokus pada manajemen komunitas jangka panjang, dan Fitur pencarian tim yang presisi.

Sumber: Hasil Olahan Peneliti (2025)

Dari tabel perbandingan di atas, dapat disimpulkan bahwa penelitian ini memiliki posisi strategis dalam melengkapi kekurangan sistem terdahulu. ClashHub menawarkan solusi yang mengintegrasikan manajemen data otomatis melalui *API* dengan fitur pengelolaan komunitas yang lebih terstruktur dan cerdas.

2.2 Landasan Teori *E-sports* dan Komunitas Gim

2.2.1 Manajemen Tim *E-sports*

Electronic Sports (*E-Sports*) didefinisikan sebagai area aktivitas olahraga di mana aspek mental dan fisik manusia dilatih serta diuji melalui penggunaan teknologi informasi dan komunikasi. Berbeda dengan sekadar bermain gim untuk rekreasi (*casual gaming*), *e-sports* melibatkan struktur organisasi yang formal, aturan standar yang ketat, serta ekosistem

kompetitif yang menuntut profesionalisme dari para pelakunya (Wagner, 2006; Hamari & Sjöblom, 2017).

Dalam konteks manajerial, sebuah tim atau klan *e-sports* beroperasi layaknya sebuah tim virtual di mana interaksi antar anggota lebih banyak terjadi secara daring. Tantangan terbesar dalam tim dengan model seperti ini adalah membangun kepercayaan (*trust*), mengingat anggota jarang atau bahkan tidak pernah bertatap muka secara fisik. Menurut Jarvenpaa & Leidner (1999), ketiadaan interaksi fisik ini membuat penilaian terhadap kompetensi dan integritas anggota menjadi sulit dilakukan. Oleh karena itu, diperlukan mekanisme transparansi rekam jejak yang valid untuk mengurangi risiko salah rekrut dan membangun kepercayaan antar anggota tim.

Manajemen tim *e-sports* modern juga telah bergeser dari pendekatan subjektif yang mengandalkan intuisi menjadi pendekatan objektif berbasis data. Tim yang sukses tidak hanya bergantung pada kemampuan mekanik pemain, tetapi juga pada kemampuan pemimpin dalam melakukan agregasi data statistik performa untuk pengambilan keputusan strategis. Aspek manajerial ini mencakup pengelolaan daftar pemain (*roster management*), analisis riwayat pertandingan, serta administrasi jadwal kompetisi (Pérez-Rubio et al., 2024).

Namun, pada tingkat komunitas amatir atau akar rumput (*grassroots*), infrastruktur pendukung manajemen berbasis data ini sering kali belum tersedia. Pengelola komunitas sering menghadapi kesulitan karena harus memproses data aktivitas anggota yang tersebar secara manual tanpa bantuan sistem terintegrasi. Kesenjangan ini menunjukkan pentingnya pengembangan sistem informasi manajemen yang mampu mengotomatisasi proses pengolahan data untuk meningkatkan efisiensi operasional tim non-profesional (Toth et al., 2024).

2.2.2 Ekosistem *Clash of Clans* (CoC)

Clash of Clans adalah gim strategi berbasis daring yang dikembangkan oleh Supercell. Gim ini memiliki mekanisme sosial yang kuat melalui fitur "Klan", di mana pemain berkumpul untuk saling membantu dan berkompetisi. Dalam tinjauan sistem informasi, mekanisme permainan ini memiliki struktur data yang kompleks yang mencakup hierarki organisasi klan dan aktivitas kompetitif terukur.

Definisi dan Fungsi Klan

Dalam ekosistem *Clash of Clans*, Klan adalah unit sosial utama yang berfungsi sebagai wadah komunitas bagi para pemain. Sebuah klan dapat menampung hingga 50 anggota dan memiliki identitas unik berupa nama, lencana (*badge*), dan tingkat klan (*clan level*) yang

memberikan keuntungan pasif tertentu. Secara fungsional, klan memfasilitasi donasi pasukan untuk pertahanan atau penyerangan, serta menjadi syarat mutlak untuk partisipasi dalam mode kompetitif tim (Supercell, 2024).

Struktur Hierarki Klan dan Pemain

Sistem manajemen klan dalam *Clash of Clans* mengadopsi hierarki peran (*Role-Based Hierarchy*) yang ketat. Struktur ini menjadi acuan standar dalam manajemen hak akses (*access control*) pada sistem pengelolaan komunitas gim. Struktur tersebut terdiri dari:

a. *Leader* (Pemimpin): Memiliki otoritas penuh (*superuser*) untuk memulai perang, menaikkan atau menurunkan jabatan anggota, dan mengeluarkan anggota dari klan.
b. *Co-Leader* (Wakil Pemimpin): Tangan kanan pemimpin yang memiliki hampir seluruh hak akses pemimpin, kecuali hak untuk membubarkan klan.
c. *Elder* (Sesepuh): Anggota senior yang memiliki hak terbatas untuk menerima anggota baru atau mengeluarkan anggota biasa (*kick*).
d. *Member* (Anggota): Anggota biasa tanpa hak manajerial khusus.
e. *Free Agent* (Agen Bebas): Istilah umum dalam *e-sports* untuk pemain yang saat ini tidak terikat dengan klan manapun, yang dalam konteks rekrutmen merupakan target utama pencarian bakat.

Aktivitas Kompetitif Klan

Terdapat beberapa mode kompetisi yang menuntut manajemen data tim yang presisi, antara lain:

a. *Clan War Leagues* (*CWL*): Fitur kompetitif bulanan di mana 8 klan bertarung dalam satu grup selama 7 hari berturut-turut. Mode ini menuntut manajemen rotasi pemain yang ketat karena batasan jumlah peserta (15 vs 15 atau 30 vs 30).
b. *War Classic* (Perang Klasik): Pertempuran antar klan yang dapat dilakukan kapan saja dengan pengaturan jumlah peserta fleksibel. Data performa dari mode ini sering digunakan sebagai indikator keaktifan dan keahlian menyerang pemain.
c. *Raid Weekend* (Serbuan Akhir Pekan): Mode kooperatif mingguan di mana kontribusi anggota diukur berdasarkan jumlah "*Capital Gold*" yang dikumpulkan untuk pembangunan ibu kota klan.

Turnamen Resmi

Sebagai acuan standar kompetisi, Supercell menyelenggarakan turnamen yang menjadi tolak ukur (*benchmark*) bagi komunitas kompetitif:

a. *Clash of Clans World Championship*: Turnamen tingkat dunia yang mempertemukan tim-tim profesional.
b. *Town Hall Cups*: Turnamen resmi yang dikategorikan berdasarkan tingkat *Town Hall* spesifik (misalnya TH9 Cup atau TH12 Cup). Skema ini menunjukkan pentingnya pengelompokan (*clustering*) pemain berdasarkan level akun untuk menciptakan iklim kompetisi yang adil.

Atribut Identitas dan Data

Dalam perancangan sistem informasi yang terintegrasi dengan gim ini, terdapat dua atribut teknis vital yang menjadi acuan pengolahan data:

a. *Player*/*Clan Tag* (Tagar Identitas): Kode unik alfanumerik (contoh: #ABC123) yang melekat pada setiap entitas pemain dan klan secara permanen. Dalam konsep basis data relasional, atribut ini berfungsi sebagai kunci utama (*primary key*) yang memungkinkan sinkronisasi data lintas platform melalui *API*.
b. *Town Hall* (Balai Kota): Indikator utama tingkat kemajuan akun. Level *Town Hall* tidak hanya menunjukkan kekuatan pertahanan, tetapi juga menjadi parameter utama dalam algoritma penyaringan (*filtering*) dan pengurutan (*sorting*) anggota, guna memastikan keseimbangan bobot kekuatan (*war weight*) dalam sebuah tim.

2.2.3 Sistem Reputasi dan Kepercayaan dalam Komunitas Maya

Dalam ekosistem komunitas daring yang anonim, membangun kepercayaan (*trust*) merupakan tantangan utama karena minimnya interaksi tatap muka. Studi terbaru menyoroti bahwa mekanisme moderasi konvensional sering kali gagal mengatasi perilaku negatif (*toxicity*) secara efektif karena kurangnya transparansi dan akuntabilitas. Oleh karena itu, diperlukan sistem reputasi yang tidak hanya berfungsi sebagai alat sanksi sosial, tetapi juga sebagai sarana transparansi informasi untuk memprediksi perilaku pengguna lain di masa depan (Holland, 2025).

Dalam konteks tim *e-sports* atau tim virtual (*virtual ad hoc teams*), kepercayaan memegang peranan krusial dalam keberhasilan kolaborasi. Lee et al. (2025) menemukan bahwa tim virtual sering mengalami hambatan komunikasi yang dapat menggerus rasa percaya antar-anggota. Ketiadaan isyarat non-verbal dan pertemuan fisik memaksa pemain untuk bergantung

pada konsep "kepercayaan cepat" (*swift trust*). Konsep ini menjelaskan bahwa dalam situasi waktu terbatas, kepercayaan dibangun bukan melalui kedekatan emosional jangka panjang, melainkan berdasarkan indikator data atau reputasi yang terlihat (*visual cues*). Tanpa adanya rekam jejak yang terkuantifikasi, pemain cenderung memiliki prasangka negatif atau kewaspadaan berlebih terhadap anggota baru.

Masalah kepercayaan ini semakin diperparah oleh perilaku toksik yang marak terjadi di lingkungan gim daring. Xu (2025) menjelaskan bahwa perilaku seperti pelecehan verbal, *trolling*, atau tindakan merugikan lainnya tidak hanya merusak pengalaman bermain, tetapi juga berdampak negatif pada kesejahteraan mental pemain. Dalam studi kasus pada gim strategi seperti *Clash of Clans*, hal ini tercermin dalam fenomena "Kutu Loncat" (*Hopper*) istilah untuk pemain yang bergabung hanya untuk keuntungan sesaat lalu pergi serta perilaku tidak sopan yang merusak harmoni klan.

Untuk mengatasi krisis kepercayaan tersebut, penerapan mekanisme reputasi berbasis data (*data-driven reputation mechanism*) menjadi solusi teknis yang relevan dalam pengembangan sistem komunitas. Berbeda dengan moderasi manual yang subjektif, sistem reputasi bertujuan menciptakan jejak digital (*digital footprint*) yang objektif bagi setiap pengguna. Dengan menyediakan riwayat perilaku yang terverifikasi, sistem ini memungkinkan pemimpin komunitas melakukan penyaringan (*filtering*) preventif terhadap anggota yang berpotensi merusak tim, sekaligus memberikan insentif gamifikasi bagi pemain untuk menjaga perilaku positif demi membangun portofolio karir *e-sports* yang baik.

2.3 Landasan Teori Sistem Informasi dan Platform

2.3.1 Sistem Informasi Manajemen (SIM) dan Agregasi Data

Sistem Informasi Manajemen (SIM) didefinisikan sebagai sekumpulan komponen yang saling berhubungan yang mengumpulkan, memproses, menyimpan, dan mendistribusikan informasi untuk mendukung pengambilan keputusan dan pengendalian dalam suatu organisasi. Dalam era digital saat ini, SIM tidak hanya berfungsi sebagai alat pencatatan administratif semata, tetapi juga menjadi aset strategis yang membantu organisasi dalam menghadapi tantangan operasional dan meningkatkan efisiensi kerja (Laudon & Laudon, 2021).

Dalam konteks manajemen organisasi yang dinamis, seperti tim *e-sports*, kebutuhan pengelola tidak hanya sebatas melihat data mentah, tetapi memerlukan informasi yang telah diolah untuk mengatasi masalah kelebihan informasi (*information overload*). Oleh karena itu, penerapan SIM modern sering kali melibatkan mekanisme agregasi data (*data aggregation*).

Agregasi data adalah proses pengumpulan data dari berbagai sumber untuk diringkas ke dalam bentuk yang lebih sederhana dan bermakna, sehingga memudahkan manusia dalam melihat pola atau tren tanpa harus menganalisis setiap entri data satu per satu.

Berbeda dengan Sistem Pendukung Keputusan (SPK) konvensional yang menggunakan model matematis kompleks untuk memberikan solusi tunggal, pendekatan SIM berbasis agregasi data berfokus pada penyajian fakta yang komprehensif untuk mendukung otonomi pengambil keputusan (*decision maker*). Menurut Sharda et al. (2020), sistem informasi yang efektif harus mampu menjalankan tiga fungsi manajerial utama:

a. Integrasi Data (*Data Integration*): Kemampuan mengonsolidasikan data yang terfragmentasi dari sumber eksternal (seperti *API* atau *database* pihak ketiga) ke dalam satu wadah terpusat untuk menjamin konsistensi informasi.
b. Visualisasi Informasi (*Information Visualization*): Kemampuan mengubah data statistik angka menjadi representasi visual (seperti grafik atau *dashboard* interaktif) yang memungkinkan pemantauan kinerja (*performance monitoring*) secara sekilas namun akurat.
c. Dukungan Keputusan Berbasis Fakta (*Evidence-Based Decision Support*): Penyediaan rekomendasi atau wawasan yang didasarkan pada data historis yang valid, sehingga mengurangi ketergantungan pada intuisi atau penilaian subjektif dalam proses manajerial.

2.3.2 Platform Berbasis *Web* Modern dan Arsitektur *Serverless*

Paradigma Aplikasi *Web* Modern (*Modern Web Application*) telah menjadi standar baru dalam pengembangan perangkat lunak lintas platform. Berbeda dengan situs *web* tradisional yang bersifat statis dan memerlukan pemuatan ulang halaman (*page reload*) secara penuh untuk setiap interaksi, aplikasi *web* modern menawarkan interaktivitas tinggi dan performa yang responsif layaknya aplikasi bawaan (*native app*). Menurut Pressman & Maxim (2015), keunggulan utama platform berbasis *web* adalah aksesibilitas universal (*ubiquity*), di mana pengguna dapat mengakses layanan melalui berbagai perangkat (ponsel cerdas, tablet, maupun PC) tanpa hambatan instalasi atau kompatibilitas sistem operasi.

Dalam konteks pengembangan sistem untuk audiens yang didominasi pengguna seluler, terdapat tiga konsep teknis utama yang sering dijadikan landasan arsitektur:

a. *Hybrid Rendering* (*Server & Client*)

   Kerangka kerja *web* modern, seperti Next.js, menerapkan metode *Hybrid Rendering* untuk menyeimbangkan performa dan interaktivitas. Metode ini menggabungkan kecepatan

pemuatan awal halaman yang diproses di server (*Server-Side Rendering*) dengan perpindahan antar-menu yang mulus di sisi pengguna (*Client-Side Navigation*). Mekanisme ini krusial untuk memastikan konten yang berat data (*data-heavy*) dapat dimuat dengan cepat, bahkan pada kondisi jaringan internet yang tidak stabil (Vercel, 2024).

b. Pendekatan Desain *Mobile-First*

Filosofi Desain *Mobile-First* (*Mobile-First Design*) adalah pendekatan perancangan antarmuka yang memprioritaskan batasan layar perangkat seluler terlebih dahulu sebelum disesuaikan (*scaling up*) ke layar yang lebih besar. Mengingat tren penggunaan gim daring yang mayoritas berbasis seluler, pendekatan ini memastikan elemen interaktif seperti tombol dan navigasi memiliki ukuran dan tata letak yang ergonomis untuk layar sentuh, sehingga meningkatkan kenyamanan pengguna (*User Experience*).

c. Arsitektur *Serverless*

Arsitektur *Serverless* adalah model eksekusi komputasi awan di mana pengelolaan infrastruktur server fisik ditangani sepenuhnya oleh penyedia layanan awan (*cloud provider*). Dalam model ini, pengembang fokus pada logika aplikasi tanpa perlu memusingkan pemeliharaan server (*server maintenance*). Keunggulan utama arsitektur ini adalah skalabilitas otomatis (*auto-scaling*), di mana sistem dapat menyesuaikan kapasitas sumber daya secara dinamis berdasarkan lonjakan trafik pengguna, yang sering terjadi pada saat penyelenggaraan *event* atau turnamen daring.

2.4 Landasan Teori Teknis dan Alat Pengembangan

Pemilihan teknologi dalam pengembangan sistem informasi modern didasarkan pada kebutuhan akan arsitektur yang responsif, skalabel, dan mampu menangani pertukaran data secara *real-time*. Sub-bab ini menguraikan landasan teori dari perangkat lunak dan kerangka kerja (*framework*) yang menjadi standar industri saat ini.

2.4.1 Next.js 14 (*App Router Architecture*)

Next.js adalah kerangka kerja pengembangan *web* berbasis pustaka React yang dirancang untuk memfasilitasi pembuatan aplikasi *web* performa tinggi dan ramah mesin pencari (*SEO-friendly*). Versi terbaru, Next.js 14, memperkenalkan arsitektur App Router yang mengubah paradigma pengelolaan rute dan data dalam aplikasi *web*. Struktur ini menawarkan efisiensi yang lebih baik dalam manajemen status (*state management*) dan *rendering* dibandingkan

arsitektur tradisional (Vercel, 2024). Terdapat tiga fitur fundamental dalam arsitektur ini yang mendukung performa aplikasi modern:

a. *React Server Components* (*RSC*)

   React Server Components (*RSC*) adalah paradigma yang membagi beban komputasi antara server dan klien. Dalam model ini, komponen yang memerlukan pemrosesan data berat (seperti pengambilan data dari *API* pihak ketiga) dieksekusi sepenuhnya di sisi server. Hasilnya dikirim ke peramban pengguna dalam bentuk HTML ringan, bukan kode JavaScript yang berat. Mekanisme ini secara signifikan mengurangi ukuran unduhan (*bundle size*), sehingga mempercepat waktu muat halaman terutama pada perangkat seluler dengan koneksi terbatas (Vercel, 2024).

b. *Server Actions*

   Server Actions adalah fitur yang memungkinkan fungsi mutasi data (*create*, *update*, *delete*) dijalankan secara langsung dari komponen antarmuka tanpa perlu membuat titik akhir (*endpoint*) *API* terpisah secara manual. Teknologi ini menyederhanakan alur komunikasi antara antarmuka pengguna (*frontend*) dan basis data (*backend*), serta meningkatkan keamanan karena logika bisnis dieksekusi di lingkungan server yang terisolasi. Hal ini berdampak langsung pada efisiensi proses iterasi kode, yang sangat krusial dalam metode *Prototype* di mana perubahan fitur sering terjadi berdasarkan umpan balik pengguna.

c. Metadata *API* untuk SEO Dinamis

   Keterlihatan (visibility) sebuah platform di mesin pencari sangat bergantung pada pengelolaan metadata. Fitur Metadata *API* pada Next.js memungkinkan pembuatan judul, deskripsi, dan *tag Open* Graph secara dinamis berdasarkan konten yang sedang diakses. Dalam konteks sistem berbasis data, fitur ini memastikan bahwa setiap halaman profil atau entitas data memiliki identitas unik yang dapat diindeks dengan baik oleh algoritma mesin pencari (SEO), meningkatkan aksesibilitas informasi bagi pengguna publik.

2.4.2 TypeScript

TypeScript adalah bahasa pemrograman sumber terbuka (*open-source*) yang dikembangkan oleh Microsoft. Bahasa ini merupakan pengembangan (*superset*) dari JavaScript yang menambahkan fitur pengetikan statis (*static typing*). Fitur ini mewajibkan pengembang untuk mendefinisikan jenis data (seperti angka, teks, atau objek) secara eksplisit saat menulis kode, berbeda dengan JavaScript konvensional yang bersifat dinamis namun rentan terhadap kesalahan tipe data (Microsoft, 2024). Dalam konteks pengembangan sistem

yang mengandalkan integrasi data eksternal, penggunaan TypeScript memberikan tiga manfaat teknis utama:

a. Jaminan Keamanan Tipe Data (*Type Safety*)

   Pengolahan data yang bersumber dari *API* eksternal sering kali melibatkan struktur data bertingkat (*nested objects*) yang kompleks. TypeScript memungkinkan pembuatan kontrak data atau "*Interface*" yang ketat. Mekanisme ini memastikan bahwa sistem hanya memproses data yang valid sesuai skema yang ditentukan, meminimalisir risiko kegagalan sistem (*crash*) akibat ketidaksesuaian format data yang diterima dari server.

b. Deteksi Kesalahan Dini (*Compile-Time Error Detection*)

   Pada bahasa pemrograman skrip biasa, kesalahan logika sering kali baru terdeteksi saat aplikasi sedang dijalankan oleh pengguna (*runtime error*). TypeScript memitigasi risiko ini dengan mendeteksi kesalahan sintaksis dan logika tipe data pada saat proses penulisan kode (*compile-time*). Hal ini secara signifikan mengurangi jumlah kutu (*bug*) yang lolos ke tahap produksi.

c. Dukungan Pemeliharaan dalam Pengembangan Iteratif

   Fitur IntelliSense (saran kode otomatis) dan dokumentasi mandiri (*self-documenting code*) pada TypeScript sangat krusial dalam metode pengembangan iteratif seperti *Prototype*. Ketika terjadi perubahan fitur atau *refactoring* kode, TypeScript membantu pengembang menelusuri dampak perubahan tersebut pada seluruh sistem dengan cepat, sehingga proses perbaikan (*debugging*) dan penambahan fitur baru dapat dilakukan dengan lebih efisien dan aman.

2.4.3 Google Firebase (*Backend-as-a-Service*)

Google Firebase adalah platform *Backend-as-a-Service* (*BaaS*) yang menyediakan berbagai layanan infrastruktur siap pakai untuk pengembangan aplikasi *web* dan seluler. Dalam rekayasa perangkat lunak modern, penggunaan *BaaS* menjadi solusi strategis untuk mengeliminasi beban pengelolaan server fisik (server management), sehingga pengembang dapat berfokus sepenuhnya pada logika bisnis dan pengalaman pengguna. Pendekatan ini sangat relevan dengan metode pengembangan *Prototype* yang menuntut kecepatan iterasi dan penyampaian fitur (*delivery*) dalam waktu singkat. Dalam arsitektur sistem berbasis data *real-time*, terdapat dua layanan fundamental Firebase yang menjadi standar industri:

Cloud Firestore (Basis Data NoSQL)

Cloud Firestore adalah basis data NoSQL berbasis dokumen (*document-oriented database*) yang dirancang untuk skalabilitas global. Berbeda dengan basis data relasional (SQL) yang menggunakan tabel kaku, Firestore menyimpan data dalam koleksi dokumen yang fleksibel (Google Developers, 2024). Keunggulan teknisnya meliputi:

a. Kompatibilitas Struktur Data Hierarkis

   Struktur penyimpanan dokumen pada Firestore memiliki kesamaan format dengan *JavaScript Object Notation* (*JSON*). Hal ini memungkinkan data kompleks yang bersumber dari *RESTful API* eksternal dapat disimpan dan diolah secara langsung tanpa memerlukan proses transformasi skema yang rumit (*schema migration*).

b. Sinkronisasi *Real-time*

   Firestore menggunakan mekanisme pendengar (*listener*) yang memungkinkan pembaruan data disinkronkan secara instan ke semua klien yang terhubung. Fitur ini krusial untuk aplikasi kolaboratif yang membutuhkan tampilan data langsung (seperti skor pertandingan atau status aktivitas) tanpa mengharuskan pengguna melakukan pemuatan ulang halaman (refresh).

Firebase Authentication

Firebase Authentication adalah layanan manajemen identitas yang mendukung autentikasi federasi (*federated identity*). Layanan ini memungkinkan integrasi sistem *login* menggunakan penyedia identitas terpercaya (seperti Google Sign-In) melalui protokol standar industri OAuth 2.0. Penggunaan layanan ini meningkatkan aspek keamanan sistem karena penanganan data sensitif, seperti kata sandi dan token sesi, dikelola sepenuhnya oleh infrastruktur keamanan Google, sehingga meminimalisir risiko kebocoran data akibat celah keamanan pada kode aplikasi buatan sendiri.

2.4.4 Tailwind *CSS* (*Utility-First Framework*)

Dalam rekayasa antarmuka pengguna (*User Interface Engineering*), pemilihan kerangka kerja *CSS* (*Cascading Style Sheets*) sangat menentukan fleksibilitas dan kecepatan pengembangan desain. Tailwind *CSS* merupakan kerangka kerja yang mengadopsi paradigma *Utility-First*. Berbeda dengan kerangka kerja tradisional yang menyediakan komponen siap pakai (*pre-designed components*) dengan gaya yang kaku, paradigma *Utility-First* menyediakan himpunan kelas utilitas tingkat rendah (*low-level utility classes*) yang dapat disusun langsung pada struktur HTML. Pendekatan ini memungkinkan pembangunan desain

yang unik (*custom design*) tanpa perlu menulis aturan *CSS* konvensional dari nol (Tailwind Labs, 2024). Penerapan Tailwind *CSS* dalam pengembangan aplikasi *web* modern didasarkan pada tiga keunggulan arsitektural utama:

a. Penerapan Prinsip *Mobile-First* secara Bawaan

   Tailwind *CSS* dirancang dengan sistem *breakpoint* yang memprioritaskan perangkat seluler (*Mobile-First*). Secara teknis, gaya dasar yang ditulis akan dirender untuk layar kecil terlebih dahulu, sementara penyesuaian untuk layar yang lebih besar (seperti tablet atau *desktop*) dilakukan menggunakan pengubah responsif (*responsive modifiers*). Mekanisme ini memastikan bahwa aplikasi *web* secara otomatis memiliki antarmuka yang adaptif dan ergonomis di berbagai ukuran layar, sejalan dengan tren penggunaan internet global yang didominasi oleh perangkat seluler.

b. Standar Aksesibilitas dan Mode Gelap

   Dukungan terhadap Mode Gelap (*Dark Mode*) kini telah menjadi standar aksesibilitas dalam desain pengalaman pengguna (*User Experience*/*UX*). Tailwind menyediakan mekanisme kontrol tema yang terintegrasi langsung dengan preferensi sistem operasi pengguna (*media query* '*prefers-color-scheme*'). Fitur ini krusial untuk menjaga kenyamanan visual (*visual comfort*) dan mengurangi kelelahan mata pengguna saat mengakses aplikasi dalam kondisi pencahayaan rendah.

c. Optimasi Performa melalui *Tree-Shaking*

   Salah satu tantangan utama dalam pengembangan *web* adalah ukuran berkas gaya yang membengkak (*bloated code*). Tailwind *CSS* mengatasi hal ini dengan fitur optimasi otomatis saat tahap produksi (*production build*). Sistem akan memindai seluruh kode HTML dan JavaScript, lalu menghapus (*purge*) semua kelas *CSS* yang tidak digunakan. Proses ini, yang dikenal sebagai *Tree-Shaking*, menghasilkan berkas *CSS* akhir yang sangat kecil (sering kali di bawah 10KB), yang berdampak signifikan pada peningkatan kecepatan pemuatan halaman (*load time*) dan performa aplikasi secara keseluruhan.

2.4.5 Integrasi *Application Programming Interface* (*API*)

Application Programming Interface (*API*) adalah seperangkat aturan dan protokol yang memungkinkan berbagai aplikasi perangkat lunak untuk saling berkomunikasi. Dalam arsitektur perangkat lunak modern, integrasi *API* pihak ketiga (*Third-Party API Integration*) menjadi strategi vital untuk memperkaya fungsionalitas sistem tanpa membebani sumber daya server lokal. Pendekatan ini memungkinkan pengembang untuk memanfaatkan layanan

eksternal yang sudah matang seperti basis data permainan, kecerdasan buatan, atau layanan video secara efisien melalui pertukaran data standar (biasanya berformat *JSON*). Terdapat tiga kategori integrasi *API* yang menjadi pilar teknis dalam pengembangan sistem manajemen komunitas berbasis data:

a. *RESTful API* dan Manajemen *Rate Limiting*

Integrasi data permainan (*Game Data Integration*) umumnya dilakukan melalui protokol *RESTful API* yang disediakan oleh pengembang gim. Mekanisme ini memungkinkan sistem eksternal untuk menarik data statistik (seperti riwayat klan atau profil pemain) secara *real-time* guna mengatasi masalah fragmentasi data manual. Namun, tantangan utama dalam integrasi ini adalah batasan laju permintaan atau *Rate Limiting*. Secara teoritis, untuk menjaga stabilitas performa server penyedia data, jumlah permintaan *API* dibatasi dalam periode waktu tertentu. Oleh karena itu, implementasi sistem harus disertai dengan mekanisme penyimpanan sementara (*Caching Strategy*). *Caching* berfungsi menyimpan salinan data yang sering diakses ke dalam basis data lokal untuk mengurangi frekuensi pemanggilan *API* langsung, sehingga mencegah pemblokiran akses akibat permintaan berlebih (Supercell, 2024).

b. *Generative AI* dan *Large Language Models* (*LLM*)

Generative AI adalah cabang kecerdasan buatan yang mampu menciptakan konten baru, termasuk teks naratif, berdasarkan pola data yang dipelajari. Teknologi ini, khususnya *Large Language Models* (*LLM*) seperti Google Gemini, dapat difungsikan sebagai mesin analisis pendukung keputusan. Dalam konteks sistem informasi manajemen, *LLM* tidak hanya menyajikan angka, tetapi mampu memproses ringkasan data statistik mentah menjadi wawasan (*insights*) atau rekomendasi strategi yang mudah dipahami oleh manusia (*natural language explanation*). Hal ini merevolusi cara penyajian data, dari sekadar visualisasi grafik menjadi narasi analitis yang mendalam (Google *Cloud*, 2024).

c. *Multimedia Content Aggregation* (YouTube Data *API*)

Untuk mendukung aspek edukasi dalam komunitas, sistem informasi modern sering kali menerapkan konsep Agregasi Konten Multimedia. Melalui integrasi *API* layanan video (seperti YouTube Data *API*), sistem dapat secara otomatis menyaring dan menampilkan konten video terbaru yang relevan (misalnya strategi permainan atau tutorial) dari sumber terpercaya. Mekanisme ini memastikan pengguna mendapatkan referensi visual terkini (*up-to-date meta*) langsung di dalam platform, meningkatkan nilai guna aplikasi sebagai pusat pengetahuan terpadu (Google Developers, 2024).

2.5 Metodologi Pengembangan Perangkat Lunak

Dalam disiplin rekayasa perangkat lunak, metodologi pengembangan berfungsi sebagai kerangka kerja yang memandu proses pembuatan sistem dari tahap konseptual hingga penyebaran. Pemilihan model yang tepat sangat bergantung pada karakteristik proyek, kejelasan kebutuhan pengguna, dan tenggat waktu penyelesaian. Salah satu model yang berfokus pada pendekatan evolusioner adalah model *Prototype*.

2.5.1 Model *Prototype*

Model *Prototype* (*Prototyping*) adalah paradigma pengembangan perangkat lunak yang didasarkan pada pembuatan model kerja awal sistem untuk diuji dan dievaluasi oleh pengguna. Menurut Roger S. Pressman (2014), pendekatan ini sangat ideal diterapkan dalam kondisi di mana pelanggan (pengguna) memiliki tujuan umum mengenai perangkat lunak yang akan dibangun, namun belum dapat mendefinisikan kebutuhan *input*, pemrosesan, atau *output* secara rinci.

Berbeda dengan model linier (seperti *Waterfall*) yang kaku, model *Prototype* bersifat iteratif (berulang). Siklus pengembangan difokuskan pada penyampaian fitur inti dengan cepat agar pengguna dapat segera berinteraksi dengan sistem. Interaksi ini memungkinkan pengembang untuk memvalidasi kebutuhan pengguna lebih awal dan meminimalisir risiko kegagalan sistem akibat kesalahpahaman spesifikasi. Tahapan dalam model *Prototype* mengacu pada kerangka kerja Pressman yang terdiri dari lima fase siklus:

a. Komunikasi (*Communication*)

   Fase ini merupakan titik awal kolaborasi antara pengembang dan pemangku kepentingan (*stakeholder*). Fokus utamanya adalah mengidentifikasi kebutuhan dasar dan tujuan umum perangkat lunak, serta memetakan area-area di mana definisi kebutuhan masih belum jelas atau ambigu.

b. Perencanaan Cepat (*Quick Plan*)

   Setelah kebutuhan awal teridentifikasi, dilakukan perencanaan kilat yang berfokus pada representasi fitur perangkat lunak yang akan terlihat oleh pengguna akhir (seperti antarmuka pengguna). Tahap ini mengabaikan detail teknis mendalam demi kecepatan iterasi.

c. Pemodelan Desain Cepat (*Modeling Quick Design*)

Fase ini melibatkan pembuatan rancangan konseptual yang berfokus pada tata letak, alur navigasi, dan interaksi sistem. Desain ini bukan desain final, melainkan sketsa fungsional yang bertujuan memberikan gambaran visual konkret kepada pengguna mengenai bentuk aplikasi yang akan dibangun.

d. Pembentukan Prototipe (*Construction of Prototype*)

Berdasarkan desain cepat, pengembang mulai menyusun kode program untuk membangun purwarupa (*prototype*) yang dapat dijalankan. Pada tahap ini, sistem mungkin belum memiliki fitur lengkap atau keamanan sempurna, namun sudah cukup fungsional untuk diuji coba.

e. Penyerahan dan Umpan Balik (*Deployment Delivery & Feedback*)

Prototipe diserahkan kepada pengguna untuk dievaluasi. Pengguna memberikan umpan balik (*feedback*) mengenai kesesuaian fitur dan kenyamanan penggunaan. Masukan ini kemudian menjadi dasar untuk revisi dan penyempurnaan pada siklus iterasi berikutnya hingga sistem dianggap layak rilis.

2.5.2 Pemodelan Sistem *Unified Modeling Language* (*UML*)

Dalam tahap "Pemodelan Desain Cepat" (*Modeling Quick Design*) pada metode *Prototype*, diperlukan alat bantu visual yang standar untuk menggambarkan rancangan sistem sebelum diterjemahkan ke dalam kode program. Alat bantu yang digunakan dalam penelitian ini adalah *Unified Modeling Language* (*UML*). *UML* didefinisikan sebagai bahasa visual standar untuk pemodelan, spesifikasi, konstruksi, dan dokumentasi artefak sistem perangkat lunak. Penggunaan *UML* memungkinkan pengembang dan pemangku kepentingan (*stakeholder*) untuk memiliki pemahaman yang sama mengenai struktur dan perilaku sistem yang akan dibangun (Rosa & Shalahuddin, 2018). Berdasarkan kebutuhan pengembangan sistem manajemen komunitas berbasis *web*, terdapat tiga jenis diagram *UML* yang relevan untuk diterapkan:

a. *Use Case Diagram*

Use Case Diagram berfungsi untuk menggambarkan interaksi fungsional antara pengguna (aktor) dan sistem. Diagram ini memetakan "apa" yang dapat dilakukan sistem, namun tidak menjelaskan secara rinci "bagaimana" sistem melakukannya.

1. Aktor (*Actor*): Merepresentasikan entitas luar (seperti Pemimpin Klan atau Anggota) yang berinteraksi dengan sistem.

2. *Use Case*: Merepresentasikan fungsionalitas atau layanan spesifik yang disediakan sistem untuk aktor.
3. Relasi: Garis penghubung yang menunjukkan hubungan interaksi antar elemen.

b. *Activity Diagram*

Activity Diagram (Diagram Aktivitas) digunakan untuk memodelkan alur kerja (*workflow*) atau proses bisnis dari sebuah fitur. Diagram ini sangat krusial dalam metode *Prototype* untuk memvalidasi logika prosedur, seperti alur pendaftaran turnamen atau mekanisme verifikasi data klan. Simbol utamanya meliputi:

1. *Initial State*: Titik awal dimulainya aktivitas.
2. *Action*/*Activity*: Tahapan proses yang dilakukan.
3. *Decision Node*: Titik percabangan keputusan (logika *if*/*else*).
4. *Final State*: Titik akhir selesainya aktivitas.

c. *Sequence Diagram*

Sequence Diagram (Diagram Urutan) menggambarkan interaksi antar objek di dalam sistem berdasarkan urutan waktu. Diagram ini lebih teknis dibandingkan *Use Case*, karena memperlihatkan bagaimana pesan (*message*) dikirimkan antar komponen (seperti dari Antarmuka ke *API*, lalu ke *Database*) untuk menjalankan suatu fungsi. Diagram ini menjadi acuan utama bagi pemrogram (*programmer*) dalam menulis kode logika pertukaran data, khususnya untuk fitur yang melibatkan integrasi *API* eksternal.

1. Desain Antarmuka (*UI*/*UX*): Membuat rancangan tampilan aplikasi (*wireframe*) dengan pendekatan *Mobile-First*. Hal ini dilakukan untuk memastikan tata letak tombol dan menu nyaman digunakan di layar ponsel, sesuai dengan kebiasaan pemain gim *mobile*.
2. Perancangan Basis Data: Merancang skema penyimpanan data pada *Cloud* Firestore untuk menampung data *JSON* dari *API Clash of Clans*, seperti koleksi data pemain (Players) dan riwayat perang (*War Logs*).

2.6 Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dalam jaminan kualitas perangkat lunak (*Software Quality Assurance*). Proses ini merepresentasikan tinjauan akhir terhadap spesifikasi, desain, dan pembuatan kode. Menurut Pressman & Maxim (2015), tujuan utama pengujian bukan sekadar membuktikan bahwa sistem berjalan benar, melainkan untuk menemukan kesalahan (*errors*) atau cacat (*bugs*) yang tersembunyi sebelum sistem diserahkan

kepada pengguna akhir. Secara umum, strategi pengujian dapat dikategorikan menjadi dua pendekatan utama: Verifikasi (apakah produk dibangun dengan benar?) dan Validasi (apakah produk yang dibangun sudah benar sesuai kebutuhan pengguna?).

2.6.1 *Black Box Testing* (Uji Fungsional)

Black Box Testing (Pengujian Kotak Hitam), atau sering disebut pengujian perilaku (*behavioral testing*), adalah metode pengujian yang berfokus pada persyaratan fungsional perangkat lunak. Sesuai namanya, metode ini memperlakukan sistem sebagai "kotak hitam" di mana penguji tidak perlu mengetahui struktur logika internal, kode program, atau variabel internal yang bekerja di dalamnya (Nidhra & Dondeti, 2012).

Pengujian ini dilakukan dengan cara memberikan serangkaian masukan (*input*) tertentu ke dalam sistem, kemudian mengamati apakah keluaran (*output*) yang dihasilkan sudah sesuai dengan spesifikasi yang diharapkan. Metode ini sangat relevan untuk menguji antarmuka sistem yang berinteraksi langsung dengan pengguna atau sistem eksternal (seperti *API*). Menurut Pressman (2015), *Black Box Testing* didesain untuk mengungkap kategori kesalahan sebagai berikut:

a. Fungsi yang Tidak Benar atau Hilang: Fitur tidak berjalan sesuai spesifikasi atau ada fitur wajib yang belum tersedia.
b. Kesalahan Antarmuka (*Interface Errors*): Ketidaksesuaian tampilan atau masalah interaksi pada elemen formulir, tombol, dan navigasi.
c. Kesalahan Struktur Data atau Akses Basis Data: Kegagalan sistem dalam menyimpan, mengambil, atau memanipulasi data dari sumber eksternal.
d. Kesalahan Kinerja (*Performance Errors*): Respon sistem yang lambat atau tidak stabil saat memproses beban kerja tertentu.
e. Kesalahan Inisialisasi dan Terminasi: Masalah yang terjadi saat sistem pertama kali dijalankan atau saat pengguna keluar dari sistem.

2.6.2 *System Usability Scale* (*SUS*)

Dalam pengembangan aplikasi yang menyasar pengguna umum (*end-user*), aspek kegunaan (*usability*) menjadi parameter penentu keberhasilan adopsi sistem. *Usability* mengacu pada sejauh mana suatu produk dapat digunakan oleh pengguna tertentu untuk mencapai tujuan tertentu dengan efektivitas, efisiensi, dan kepuasan (ISO 9241-11).

Untuk mengukur parameter tersebut secara kuantitatif, metode *System Usability Scale* (*SUS*) digunakan sebagai standar industri. Diciptakan oleh John Brooke (1996), *SUS* dikenal sebagai alat ukur yang "cepat dan kotor" (*quick and dirty*), artinya metode ini mampu memberikan hasil yang valid dan reliabel secara statistik meskipun dengan jumlah sampel responden yang kecil.

a. Instrumen Pengukuran

SUS terdiri dari 10 item pernyataan standar yang menggunakan skala Likert 5 poin, mulai dari 1 (Sangat Tidak Setuju) hingga 5 (Sangat Setuju). Instrumen ini dirancang dengan teknik item selang-seling (*alternating items*) antara pernyataan positif dan negatif untuk mengurangi bias respon pengguna.

1. Item Ganjil (Positif): Fokus pada kemudahan dan keinginan menggunakan sistem (contoh: "Saya rasa sistem ini mudah digunakan").
2. Item Genap (Negatif): Fokus pada hambatan dan kerumitan sistem (contoh: "Saya menemukan sistem ini sangat membingungkan").

b. Mekanisme Perhitungan Skor

Skor *SUS* tidak dihitung dari rata-rata langsung, melainkan melalui rumus konversi untuk menghasilkan nilai dalam rentang 0 hingga 100. Menurut Sauro (2011), algoritma perhitungannya adalah:

1. Untuk item ganjil (1, 3, 5, 7, 9): Skor Item = (Posisi Skala Pengguna – 1).
2. Untuk item genap (2, 4, 6, 8, 10): Skor Item = (5 – Posisi Skala Pengguna).
3. Skor Akhir *SUS* = Total jumlah skor dari kesepuluh item dikalikan dengan konstanta 2,5.

c. Interpretasi Kelayakan Sistem

Meskipun skor *SUS* memiliki rentang 0-100, nilainya tidak dapat diinterpretasikan langsung sebagai persentase nilai ujian sekolah. Bangor et al. (2009) mengembangkan pedoman interpretasi yang memetakan skor *SUS* ke dalam kategori kelayakan (*Adjective Ratings*):

1. Skor < 51: *Grade* F (*Not Acceptable* / Tidak Layak).
2. Skor 51 - 68: *Grade* D (*Marginal* / Perlu Perbaikan).
3. Skor 68: Nilai Rata-rata Industri (*Average Benchmark*).
4. Skor > 68: *Grade* C hingga A (*Acceptable* / Layak).
5. Skor > 80.3: *Grade* A (*Excellent* / Sangat Baik).

Landasan teori ini menegaskan bahwa sebuah sistem perangkat lunak dianggap memenuhi standar kegunaan yang baik jika mampu mencapai skor di atas ambang batas rata-rata 68.

BAB III
METODOLOGI PENELITIAN

3.1 Objek dan Subjek Penelitian

3.1.1 Objek Penelitian

Objek utama dalam penelitian ini adalah pengembangan sistem informasi manajemen komunitas *e-sports* yang diberi nama "ClashHub". Sistem ini dibangun berbasis situs *web* (*website*) dengan studi kasus spesifik pada komunitas gim *Clash of Clans* di Indonesia. Pemilihan objek ini didasarkan pada urgensi kebutuhan infrastruktur teknologi yang mampu mendukung ekosistem *e-sports* di tingkat akar rumput (*grassroots*) agar lebih terorganisir, transparan, dan efisien (Kementerian Kominfo, 2024). Secara spesifik, lingkup pengembangan pada objek penelitian ini difokuskan pada empat modul fungsional utama untuk menjawab permasalahan mitra komunitas:

a. Modul Pencarian Tim Presisi (*Precision Team Finding*)

   Pengembangan fitur pencarian klan dan pemain yang dilengkapi dengan algoritma penyaringan (*filtering*) dan pengurutan (*sorting*) dinamis. Parameter pengurutan mencakup Skor *Town Hall* (*TH*), Level Reputasi, dan orientasi bermain (kompetitif atau santai). Fitur ini dirancang untuk mengatasi inefisiensi rekrutmen manual yang sering kali tidak tepat sasaran.

b. Modul Manajemen Klan Berbasis Data

   Implementasi sistem manajemen yang terintegrasi langsung dengan *Application Programming Interface* (*API*) resmi *Clash of Clans* (Supercell, 2024). Modul ini berfungsi melakukan agregasi data statistik anggota (seperti riwayat perang dan donasi) secara otomatis dan *real-time*, meminimalisir kesalahan pencatatan manusia (*human error*).

c. Modul Administrasi Turnamen Otomatis

   Penyediaan fitur pengelolaan kompetisi yang meliputi validasi otomatis level akun peserta (untuk mencegah kecurangan/*smurfing*) dan pembuatan bagan pertandingan (*bracket generation*). Modul ini bertujuan mengefisiensikan kinerja panitia turnamen komunitas yang selama ini terkendala proses verifikasi manual.

d. Modul Asisten Cerdas (*Smart Strategy Assistant*)

   Penerapan fitur pendukung keputusan bagi pemimpin klan dengan memanfaatkan teknologi *Generative AI*. Modul ini bertugas menganalisis data statistik perang yang telah

dikumpulkan, kemudian menyajikannya dalam bentuk narasi saran strategi tekstual untuk membantu pemimpin klan mengambil keputusan yang objektif.

3.1.2 Subjek Penelitian

Subjek penelitian ditentukan menggunakan teknik *purposive sampling* (pengambilan sampel bertujuan). Teknik ini dipilih karena fitur spesifik "ClashHub" hanya relevan bagi segmen pemain tertentu yang memiliki pemahaman mendalam tentang manajemen klan dan strategi permainan. Subjek penelitian diklasifikasikan menjadi dua kelompok utama, yaitu Pakar Komunitas sebagai validator dan Pemain Aktif sebagai responden uji coba.

Pakar Komunitas (*Expert Validators*)

Berbeda dengan pengujian akademis konvensional, validasi sistem ini melibatkan "*Key Opinion Leaders*" (*KOL*) atau tokoh berpengaruh dalam komunitas *Clash of Clans* Indonesia. Para pakar ini dipilih berdasarkan rekam jejak mereka dalam mengelola komunitas besar, menyelenggarakan turnamen, atau pengalaman kompetitif tingkat nasional. Peran mereka adalah memberikan penilaian validitas terkait fungsionalitas fitur manajemen dan akurasi saran strategi AI. Tabel 3.1 berikut adalah profil 5 pakar komunitas yang ditetapkan sebagai validator:

Tabel 3.1 Pengembangan Profil Pakar Validasi (Expert)

ID	Peran Komunitas	Kualifikasi	Fokus Validasi
KOL-01	*Leader* Klan "Indo Pride"	Level Klan 24 (*Top* Lokal)	Manajemen Klan & Akurasi AI
KOL-02	Ketua Panitia Turnamen	Penyelenggara 5+ Turnamen	Sistem & Bagan Turnamen
KOL-03	Konten Kreator CoC	50k+ *Subscribers*	Akurasi Data Profil & *UI*
KOL-04	Kapten Tim *E-Sports*	Finalis Turnamen Nasional	Fitur Pencarian Tim (*Team Finding*)
KOL-05	*Admin Discord*	Moderator 2.000+ Anggota	Sistem Reputasi & Kepercayaan

Sumber: Hasil Olahan Peneliti (2025)

Partisipan Pengujian (*User Respondents*)

Kelompok ini merupakan target pengguna akhir yang dilibatkan dalam tahap pengujian fungsional dan usabilitas (*System Usability Scale*). Partisipan dipilih melalui seleksi terbuka di grup komunitas dengan kriteria inklusi:

a. Pemain aktif dengan level *Town Hall* (*TH*) minimal 12.
b. Memiliki pengalaman mengelola klan atau berpartisipasi dalam turnamen.
c. Menggunakan perangkat seluler sebagai alat bermain utama.

Sebanyak 15 partisipan terpilih untuk mewakili berbagai kategori pengguna, mulai dari Manajer Klan, Peserta Turnamen, hingga Penyelenggara, dengan rincian seperti dalam Tabel 3.2 berikut:

Tabel 3.2 Data Partisipan Pengujian

ID	**Inisial & Peran**	**Level Akun**	**Perangkat Uji**	**Kategori Pengguna**
P-01	Kevin Aditya (*Leader*)	*TH* 16 (*Max*)	iPad *Pro* M2	Manajer Klan
P-02	Lukman Budiarto (*Leader*)	*TH* 15	Samsung S23 Ultra	Manajer Klan
P-03	Qori Lukman (*Co-Leader*)	*TH* 16	iPhone 14	Manajer Klan
P-04	Irwan Wibowo (*Co-Leader*)	*TH* 14	Poco F5	Manajer Klan
P-05	Satria Baskoro (*Leader*)	*TH* 15	Xiaomi Pad 6	Manajer Klan
P-06	Putra Sanjaya (*Elder*)	*TH* 16	ROG Phone 7	Peserta Turnamen
P-07	Zacky Maulana (*Member*)	*TH* 13	Infinix GT 10 *Pro*	Peserta Turnamen
P-08	Rizky Akbar (*Elder*)	*TH* 14	iPhone 11	Peserta Turnamen
P-09	Candra Setiawan (*Member*)	*TH* 15	Samsung A54	Peserta Turnamen
P-10	Xavier Lubis (*Member*)	*TH* 13	Realme 10 *Pro*	Peserta Turnamen
P-11	Teguh Gunawan (*Leader*)	*TH* 16	PC (Emulator)	Penyelenggara
P-12	Andi Irawan (*Co-Leader*)	*TH* 15	Laptop (Chrome)	Penyelenggara
P-13	Eko Hermawan (*Member*)	*TH* 12	Redmi Note 12	Peserta Turnamen
P-14	Surya Tama (*Co-Leader*)	*TH* 14	Oppo Reno 8	Manajer Klan
P-15	Bima Mahardika (*Elder*)	*TH* 13	Vivo V27	Penyelenggara

Sumber: Hasil Olahan Peneliti (2025)

3.1.3 Waktu dan Lokasi Penelitian

Pelaksanaan penelitian ini mencakup dua aspek operasional utama, yaitu alokasi waktu pengembangan sistem dan lingkungan tempat penelitian berlangsung.

Waktu Penelitian

Penelitian ini dilaksanakan selama kurun waktu 5 bulan, terhitung mulai dari penetapan judul penelitian pada 1 September 2025 hingga persiapan sidang akhir pada 21 Januari 2026. Jadwal kegiatan disusun mengacu pada fase iteratif metode *Prototype* dengan rincian sebagai berikut:

a. Tahap Komunikasi (September 2025)
   1. Studi literatur terkait manajemen *e-sports* dan teknologi *Generative AI*.
   2. Observasi lapangan di grup komunitas untuk memvalidasi masalah "fragmentasi data".
   3. Wawancara kebutuhan fitur dengan Pakar Komunitas (*Leader*/*Co-Leader*).

b. Tahap Perencanaan & Pemodelan (Oktober 2025)
   1. Analisis kebutuhan fungsional (Modul Pencarian Tim & Turnamen).
   2. Perancangan arsitektur basis data NoSQL di Firebase.
   3. Desain antarmuka (*Wireframe*) dengan pendekatan *Mobile-First*.

c. Tahap Pembentukan Prototipe (Oktober – November 2025)
   1. Iterasi 1: Pengembangan modul autentikasi dan integrasi *API Clash of Clans*.
   2. Iterasi 2: Implementasi algoritma penyaringan (*filtering*) dan pengurutan (*sorting*) data pemain.
   3. Iterasi 3: Pengembangan modul administrasi turnamen otomatis dan integrasi fitur Asisten Strategi AI.

d. Tahap Penyerahan & Umpan Balik / Pengujian (November – Desember 2025)

   Fase ini merupakan inti dari evaluasi sistem. Waktu pelaksanaan pengujian dijadwalkan secara sistematis dengan rincian durasi sebagai berikut:
   1. Rekrutmen Partisipan: 20 November – 27 November 2025.
   2. Pengujian Fungsional (Black Box) & Usabilitas (*SUS*): Dilaksanakan pada 30 November – 6 Desember 2025. Pengujian dilakukan secara mandiri oleh 15 partisipan dengan durasi estimasi 15-20 menit per sesi.
   3. Validasi Akseptansi Pengguna (*UAT*): Dilaksanakan pada 7 Desember – 10 Desember 2025 melalui wawancara mendalam bersama 5 Pakar Komunitas (*KOL*).

e. Pelaporan (Desember 2025 – Januari 2026)

   Analisis hasil pengujian, penyusunan dokumentasi naskah skripsi, dan finalisasi laporan untuk kesiapan sidang pada 21 Januari 2026.

Lokasi Penelitian

Mengingat objek penelitian berbasis perangkat lunak dengan komunitas yang terdistribusi secara digital, penelitian dilaksanakan di dua lingkungan kerja:

a. Lokasi Pengembangan (*Development Environment*):Proses rekayasa perangkat lunak, mulai dari *coding*, *debugging*, hingga *deployment*, dilakukan di Laboratorium Informatika UII serta stasiun kerja pribadi penulis. Lingkungan ini dikondisikan untuk simulasi akses *mobile* guna memastikan responsivitas aplikasi.
b. Lokasi Pengujian dan Pengambilan Data (*Test Environment*):Interaksi dengan subjek penelitian dan pelaksanaan pengujian (*Remote Testing*) dilakukan secara daring melalui kanal komunikasi resmi komunitas *Clash of Clans* Indonesia (server *Discord* dan grup *WhatsApp*). Aplikasi yang diuji ditempatkan pada infrastruktur *cloud* Vercel dengan lokasi server Singapura (sin1) untuk memastikan kecepatan akses bagi pengguna di seluruh wilayah Indonesia.

3.2 Metode Penelitian

Mengacu pada landasan teori metode *Prototype* yang telah dipaparkan pada Bab 2, alur pengembangan sistem "ClashHub" dilaksanakan melalui lima tahapan iteratif. Pendekatan ini memungkinkan penulis untuk menyesuaikan spesifikasi fitur berdasarkan umpan balik langsung dari Pakar Komunitas dan responden pengguna.

3.2.1 Penerapan Model *Prototype*

Berikut adalah rincian aktivitas teknis yang dilakukan pada setiap tahapan:

a. Tahap Komunikasi (*Communication*) Tahap ini merupakan fase penggalian kebutuhan (*requirement gathering*) untuk memvalidasi permasalahan.
    1. Aktivitas: Penulis melakukan wawancara mendalam dengan 5 orang Pakar Komunitas (Validator Ahli) untuk memetakan alur kerja manajemen klan yang ideal.
    2. Hasil: Teridentifikasi kebutuhan prioritas akan fitur Pencarian Tim Presisi (menggunakan filter skor *Town Hall* dan Reputasi) serta kebutuhan otomatisasi bagan turnamen yang selama ini dikelola manual via Excel.
b. Tahap Perencanaan Cepat (*Quick Plan*) Berdasarkan hasil komunikasi, penulis menyusun rencana teknis pengembangan sistem.

1. Arsitektur Teknologi: Menetapkan penggunaan Next.js 14 sebagai kerangka kerja *frontend* untuk performa tinggi, dan Google Firebase (Firestore) sebagai basis data NoSQL yang fleksibel untuk menampung struktur data *JSON* dari gim.
2. Skema Integrasi: Merancang logika *middleware* untuk menjembatani komunikasi antara aplikasi dengan *API Clash of Clans* (untuk data statistik) dan Google Gemini (untuk analisis strategi).

c. Tahap Pemodelan Desain Cepat (*Modeling Quick Design*) Penulis menerjemahkan kebutuhan fungsional ke dalam rancangan visual dan logika sistem.
   1. Pemodelan *UML*: Membuat diagram *Use Case* untuk mendefinisikan aktor (*Leader*, *Member*, *Admin*) dan diagram *Activity* untuk memvalidasi alur pendaftaran turnamen.
   2. Desain Antarmuka: Merancang *Wireframe* dan *Mockup* aplikasi menggunakan pendekatan *Mobile-First Design*. Fokus desain diarahkan pada kemudahan navigasi satu tangan (*thumb-friendly zone*) mengingat mayoritas responden menggunakan ponsel.

d. Tahap Pembentukan Prototipe (*Construction of Prototype*) Tahap ini adalah fase eksekusi penulisan kode program (*coding*) yang dilakukan dalam beberapa iterasi modul:
   1. Modul 1 (*Core*): Pengembangan sistem autentikasi dan integrasi *API* untuk sinkronisasi data profil pemain secara *real-time*.
   2. Modul 2 (*Features*): Implementasi algoritma pengurutan (*sorting*) dinamis berdasarkan level *Town Hall* dan skor Reputasi.
   3. Modul 3 (*Intelligence*): Integrasi layanan *Generative AI* untuk fitur "Asisten Strategi" dan algoritma *bracket generator* otomatis untuk turnamen.

e. Tahap Penyerahan dan Umpan Balik (*Deployment & Feedback*) Prototipe fungsional diserahkan kepada subjek penelitian untuk diuji coba.
   1. Pengujian Teknis: Penulis melakukan *Black Box Testing* untuk memverifikasi fungsi *input-output* pada fitur pencarian dan manajemen data.
   2. Pengujian Pengguna: Melibatkan 15 responden (P-01 s.d P-15) untuk mencoba aplikasi di lingkungan produksi (*Production Environment*) dan mengisi kuesioner *SUS*. Masukan dari tahap ini (seperti permintaan fitur *Export* Excel) dicatat sebagai bahan evaluasi pengembangan selanjutnya.

3.2.2 Alat dan Lingkungan Pengembangan

Untuk mendukung kelancaran proses penelitian dan pengembangan sistem, penulis menetapkan spesifikasi lingkungan pengembangan (*development environment*) yang terdiri dari perangkat keras dan perangkat lunak. Pemilihan alat ini didasarkan pada kebutuhan sistem akan performa komputasi yang memadai untuk menjalankan simulasi server lokal dan kompatibilitas dengan tumpukan teknologi (*tech stack*) yang digunakan.

Perangkat Keras (*Hardware*)

Proses pengembangan, pengkodean, dan simulasi sistem dilakukan menggunakan perangkat komputasi dengan spesifikasi sebagai berikut:

a. Laptop Pengembangan:
   1. Prosesor: Intel *Core* i7 (Generasi ke-11) atau setara, untuk menangani proses kompilasi kode Next.js yang intensif.
   2. Memori (RAM): 16 GB DDR4, untuk menjalankan *local server*, emulator, dan *browser* secara simultan tanpa hambatan (*bottleneck*).
   3. Penyimpanan: SSD 512 GB NVMe, untuk mempercepat proses baca-tulis (*I/O operations*) aset proyek.

b. Perangkat Pengujian (*Test Devices*):
   1. *Smartphone* Android & iOS berbagai ukuran layar.
   2. Digunakan untuk validasi desain *Mobile-First* secara langsung pada perangkat fisik (*real-device testing*), memastikan elemen *UI*/*UX* responsif di tangan pengguna.

Cucu Subbab Perangkat Lunak (*Software*)

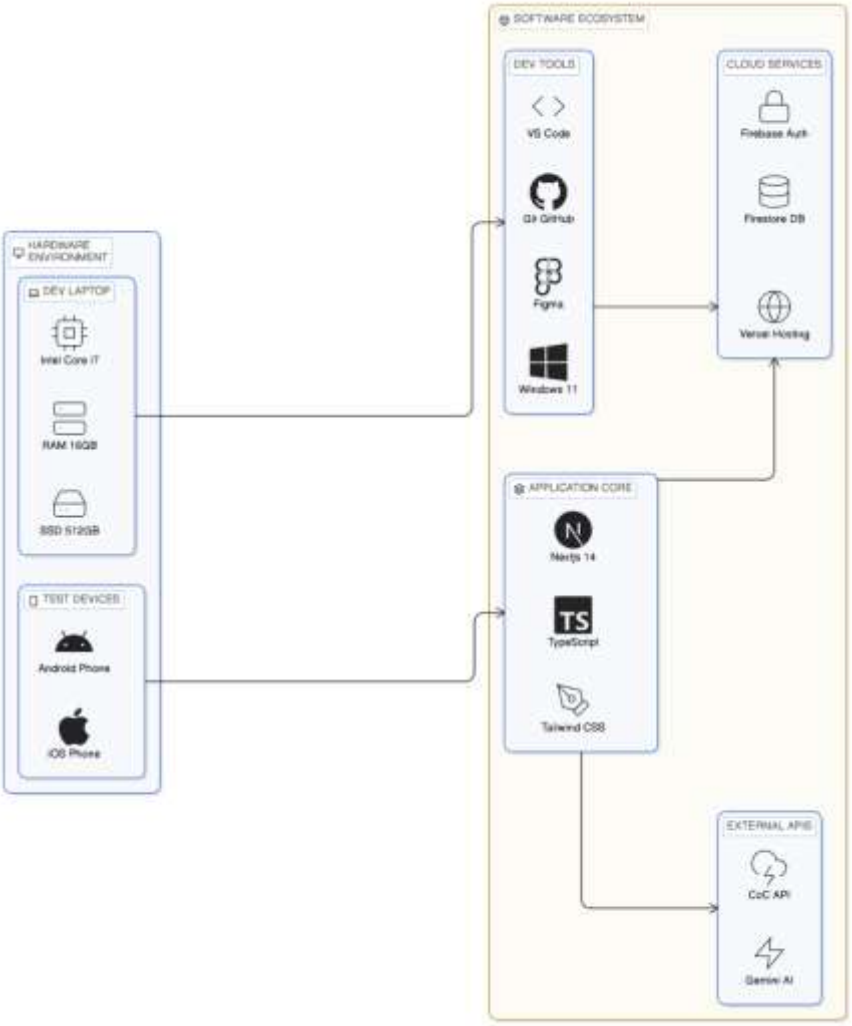
Perangkat lunak yang digunakan dipilih berdasarkan dukungan komunitas yang luas (*Community Support*) dan kesesuaiannya dengan metode *Prototype* yang menuntut kecepatan iterasi. Daftar perangkat lunak beserta fungsinya dalam penelitian ini disajikan pada Tabel 3.3 berikut:

Tabel 3.3 Daftar Perangkat Lunak Pengembangan

Kategori	Perangkat Lunak	Deskripsi & Fungsi dalam Penelitian
Sistem Operasi	Windows 11	Lingkungan kerja utama untuk manajemen berkas sistem.
Bahasa Pemrograman	TypeScript	Bahasa utama pengembangan yang menyediakan keamanan tipe data (*type safety*) untuk meminimalisir *runtime error* saat pengolahan data *API*.
Kerangka Kerja	Next.js 14	*Framework* berbasis React dengan arsitektur App Router untuk membangun antarmuka *web* yang performan dan ramah SEO.
Basis Data	Google *Cloud* Firestore	Basis data NoSQL untuk menyimpan dokumen *JSON* (Profil Pemain, Data Klan, Turnamen) secara fleksibel dan *real-time*.
Autentikasi	Firebase Auth	Layanan manajemen identitas untuk menangani sesi *login* pengguna secara aman menggunakan akun Google (*Federated Identity*).
Desain *UI*	Tailwind *CSS*	Pustaka *utility-first* untuk mempercepat *styling* antarmuka yang adaptif (*responsive*) terhadap berbagai ukuran layar ponsel.
Sumber Data	*Clash of Clans API*	*RESTful API* resmi yang berfungsi sebagai sumber utama agregasi data statistik pemain dan klan.
Kecerdasan Buatan	Google Gemini *API*	Layanan *Generative AI* yang diintegrasikan untuk memproses analisis data menjadi narasi "Asisten Strategi".
Editor Kode	Visual Studio Code	Lingkungan pengembangan terpadu (*IDE*) untuk penulisan dan *debugging* kode program.
Versi Kontrol	Git & GitHub	Sistem kontrol versi untuk manajemen riwayat perubahan kode (*version control*) dan kolaborasi.
Desain Prototipe	Figma	Alat desain visual untuk merancang *Wireframe* dan *Mockup* antarmuka sebelum tahap pengkodean.
Deployment	Vercel	Platform *cloud* untuk mempublikasikan hasil akhir aplikasi agar dapat diakses publik melalui internet.

Sumber: Hasil Olahan Peneliti (2025)

Ilustrasi hubungan antar komponen perangkat lunak tersebut dapat dilihat pada Gambar 3.1 berikut:



Gambar 3.1 Arsitektur Lingkungan Pengembangan

Sumber: Hasil Olahan Peneliti (2025)

3.2.3 Identifikasi Masalah

Sebagai implementasi nyata dari tahap Komunikasi pada metode *Prototype*, penulis melakukan penggalian masalah secara mendalam untuk memvalidasi urgensi pengembangan sistem. Pengumpulan data dilakukan melalui dua pendekatan: survei kuantitatif terhadap 100 responden aktif (didominasi usia 16-30 tahun) dan wawancara kualitatif dengan 5 Pakar Komunitas (*Leader* Senior). Berdasarkan analisis data lapangan tersebut, ditemukan lima akar permasalahan utama yang menjadi landasan spesifikasi fitur sistem:

a. Inefisiensi Pencarian Tim akibat Ketiadaan Filter Presisi

   Meskipun 86% responden menyatakan keinginan untuk bergabung dalam tim, proses pencarian saat ini dinilai tidak efisien karena informasi tersebar acak di media sosial. Masalah Spesifiknya adalah terjadi ketidakcocokan visi (*Vision Mismatch*). Data

menunjukkan 43% pemain berorientasi santai (*casual*), sedangkan 34% sangat kompetitif. Ketiadaan fitur penyaringan (*filtering*) dan pengurutan (*sorting*) berdasarkan orientasi bermain dan Level *Town Hall* menyebabkan tingginya angka keluar-masuk anggota (*turnover*).

b. Krisis Kepercayaan dan Fenomena "Kutu Loncat" (*Hopper*)

Sebanyak 71% responden menyatakan keresahan terhadap perilaku pemain yang tidak loyal. Masalah utama yang dihadapi pemimpin klan adalah fenomena "Kutu Loncat" (*Hopper*) pemain yang bergabung hanya untuk meminta donasi lalu pergi. Dampaknya adalah pemimpin klan tidak memiliki instrumen objektif untuk memvalidasi rekam jejak (*track record*) calon anggota. Hal ini mendasari kebutuhan mendesak akan fitur Sistem Reputasi berbasis data untuk memberikan label kredibilitas pada profil pemain.

c. Keterbatasan Manajemen Data pada Perangkat Seluler

Mayoritas responden (84%) adalah pemain tingkat lanjut yang membutuhkan manajemen data klan yang rapi. Namun, 60% pemimpin klan mengaku terpaksa menggunakan alat bantu manual (Excel/*Spreadsheet*) yang sulit dioperasikan di layar ponsel. Masalah Teknisnya adalah antarmuka alat bantu manual tidak responsif (*Bad User Experience*) di perangkat seluler, mengakibatkan proses pemantauan aktivitas anggota menjadi lambat dan rentan kesalahan.

d. Fragmentasi Referensi Strategi (*Information Overload*)

Pemain tingkat lanjut membutuhkan referensi strategi penyerangan (*Meta Game*) yang spesifik. Saat ini, informasi strategi tersebar secara acak di YouTube tanpa pengelompokan yang jelas berdasarkan level *Town Hall*. Dampaknya adalah pemimpin klan kesulitan memberikan saran strategi yang cepat dan akurat kepada anggota, mendasari perlunya fitur Asisten Strategi berbasis *Generative AI* yang mampu merangkum data menjadi saran taktis.

e. Inefisiensi Administrasi Turnamen Komunitas

Penyelenggara turnamen komunitas sering menghadapi kendala teknis dalam validasi peserta. Dampaknya adalah panitia kesulitan memverifikasi apakah level *Town Hall* pendaftar sesuai dengan kategori turnamen (sering terjadi manipulasi data/akun joki). Proses validasi manual ini memakan waktu lama dan berpotensi menimbulkan kecurangan, sehingga diperlukan sistem validasi otomatis melalui integrasi *API*.

Berdasarkan analisis data lapangan tersebut, ditemukan lima akar permasalahan utama yang menjadi landasan spesifikasi fitur sistem seperti yang disajikan pada Gambar 3.2 berikut:

Gambar 3.2 Grafik Persentase Masalah Utama Responden

Sumber: Hasil Pengisian Kuisioner (2025)

3.2.4 Pengumpulan Data

Dalam tahap Komunikasi pada metode *Prototype*, penulis menerapkan teknik pengumpulan data triangulasi untuk memastikan spesifikasi sistem yang dibangun valid dan sesuai kebutuhan lapangan. Data dikelompokkan menjadi dua kategori utama, yaitu data primer dan data sekunder.

Data Primer (Sumber Langsung)

Data primer diperoleh secara langsung melalui interaksi penulis dengan subjek penelitian di komunitas *Clash of Clans* Indonesia.

a. Kuesioner Daring (*Quantitative Survey*):

Penulis menyebarkan kuesioner kepada 100 responden terpilih secara *purposive*. Hasil analisis data menunjukkan karakteristik demografi yang menjadi landasan desain sistem:

1. Mobilitas Pengguna: Dominasi usia 16-30 tahun (84%) dengan status Pelajar/Mahasiswa. Temuan ini memvalidasi keputusan penggunaan desain antarmuka "*Mobile-First*" karena segmen ini memiliki mobilitas tinggi dan jarang menggunakan PC untuk bermain.
2. Tingkat Keahlian: Sebanyak 84% responden adalah pemain tingkat lanjut (Veteran/TH10+). Hal ini menuntut sistem untuk menyajikan data statistik mendalam (*Advanced Analytics*), bukan sekadar panduan dasar.

b. Wawancara Mendalam (*In-Depth Interview*):

Penulis melakukan diskusi teknis dengan 5 Pakar Komunitas (Validator Ahli). Temuan kunci dari wawancara ini adalah:

1. Inefisiensi Alat: Penggunaan *spreadsheet* di ponsel dinilai sangat tidak ergonomis (*bad UX*) untuk *input* data perang.
2. Kebutuhan Filter: Pemimpin klan kesulitan menyaring anggota baru karena sering terjadi ketidakcocokan visi (43% santai vs 34% kompetitif). Hal ini menjadi dasar pengembangan fitur Penyaringan (*Filtering*) dan Pengurutan (*Sorting*) berbasis preferensi bermain.

c. Observasi Partisipatif (*Field Observation*):Penulis terjun langsung ke dalam grup *WhatsApp* dan server *Discord* penyelenggara turnamen. Ditemukan fakta bahwa panitia memvalidasi data peserta secara manual satu per satu. Proses ini memakan waktu lama dan rentan kecurangan (*smurfing*), yang mendasari kebutuhan fitur Validasi Turnamen Otomatis.

Data Sekunder (Studi Dokumentasi)

Data sekunder diperoleh dari dokumen teknis dan literatur pendukung untuk memperkuat landasan pengembangan fitur.

a. Dokumentasi Teknis *API* (Supercell): Penulis membedah dokumentasi resmi *Clash of Clans API* untuk memetakan *endpoint JSON* yang tersedia, seperti players/{*playerTag*} dan clans/{clanTag}/*warlog*. Studi ini penting untuk memahami batasan teknis (*rate limit*s) agar mekanisme sinkronisasi data berjalan stabil.

b. Dokumentasi Google Gemini AI: Mempelajari parameter model *Generative AI* untuk merancang teknik *Prompt Engineering* yang efektif, sehingga AI mampu menerjemahkan data statistik angka menjadi narasi strategi yang natural.

c. Studi Literatur: Mengkaji penelitian terdahulu mengenai konsep *Swift Trust* dalam tim virtual sebagai landasan teoritis penerapan Sistem Reputasi pemain.

3.2.5 Kerangka Pikir Penelitian

Kerangka pikir penelitian disusun untuk memberikan gambaran sistematis mengenai alur penyelesaian masalah, mulai dari identifikasi kendala hingga dampak yang diharapkan dari solusi yang dibangun. Penelitian ini menggunakan pendekatan logis *Input*-Proses-*Output*-Dampak (*Input-Process-Output-Outcome*) yang diuraikan sebagai berikut:

a. Masukan (*Input*)

Tahap ini merangkum akar permasalahan fundamental yang ditemukan pada tahap identifikasi masalah:

1. Inefisiensi Rekrutmen: Tingginya tingkat keluar-masuk anggota (*turnover*) akibat ketidakcocokan visi bermain (santai vs kompetitif) dan ketiadaan alat filter pencarian yang presisi.
2. Krisis Kepercayaan: Kesulitan pemimpin klan dalam memvalidasi rekam jejak (*track record*) pemain baru, memicu risiko masuknya pemain "kutu loncat" (*hopper*) yang merugikan klan.
3. Kendala Administrasi: Proses validasi peserta turnamen dan pencatatan statistik perang yang masih dilakukan secara manual, rentan terhadap manipulasi dan kesalahan manusia (*human error*).

b. Proses (*Process*)

Tahap ini menjelaskan metode dan teknologi yang diterapkan sebagai solusi penyelesaian masalah:

1. Metode Pengembangan: Penerapan metode *Prototype* dengan siklus iteratif untuk memastikan fitur sesuai kebutuhan komunitas.
2. Implementasi Teknologi: Pengembangan aplikasi berbasis *web* (Next.js) dengan arsitektur *Mobile-First*, didukung basis data NoSQL Firebase untuk fleksibilitas penyimpanan.
3. Integrasi Sistem: Pemanfaatan *API Clash of Clans* untuk agregasi data otomatis, Algoritma *Sorting* & *Filtering* untuk pencarian tim, serta *Generative AI* sebagai asisten strategi.

c. Luaran (*Output*)

Hasil fisik dari pengembangan sistem berupa platform "ClashHub" yang memiliki modul fungsional:

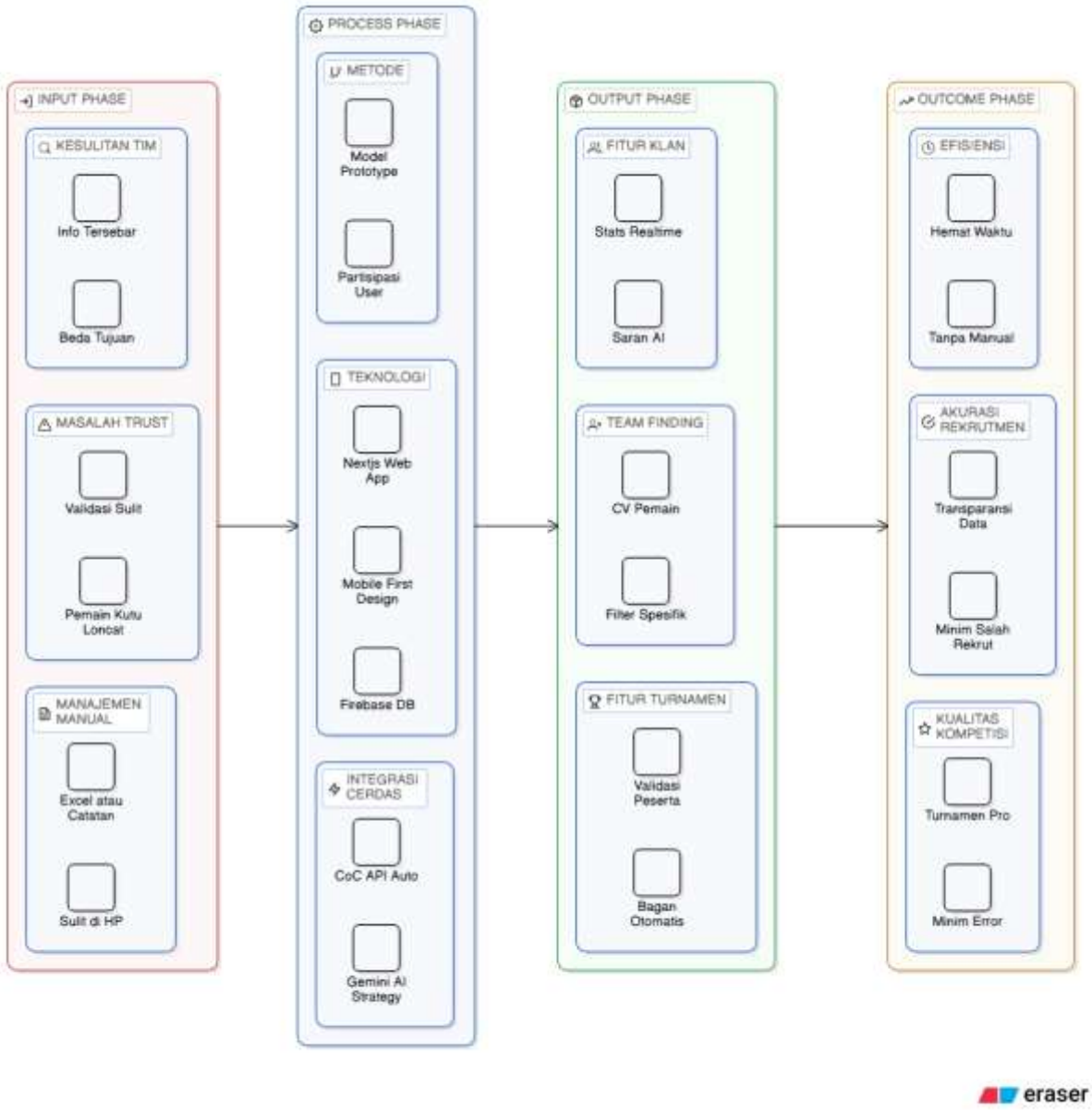
1. Modul Pencarian Tim Presisi: Fitur pencarian klan/pemain dengan filter berbasis Skor *Town Hall*, Level Reputasi, dan Orientasi Bermain.
2. Modul Manajemen Berbasis Data: Dasbor klan yang menampilkan statistik *real-time* dan riwayat aktivitas anggota yang terverifikasi.
3. Modul Asisten Cerdas: Fitur analisis strategi tekstual berbasis AI.
4. Modul Turnamen Otomatis: Sistem validasi peserta dan *bracket generator* otomatis.

d. Dampak (*Outcome*)

Manfaat jangka panjang yang diharapkan bagi ekosistem komunitas:

1. Efisiensi Operasional: Eliminasi pencatatan manual bagi pemimpin klan dan panitia turnamen.
2. Transparansi & Kepercayaan: Terciptanya ekosistem rekrutmen yang objektif berbasis data dan reputasi yang valid.
3. Peningkatan Kualitas Kompetisi: Penyelenggaraan turnamen yang lebih profesional, jujur, dan terorganisir.

Visualisasi alur kerangka pikir penelitian dapat dilihat pada Gambar 3.3 berikut:



Gambar 3.3 Bagan Kerangka Pikir Penelitian

Sumber: Hasil Olahan Peneliti (2025)

3.3 Analisis Kebutuhan Sistem

Analisis kebutuhan sistem merupakan langkah krusial dalam tahap Perencanaan Cepat untuk menerjemahkan permasalahan pengguna menjadi spesifikasi teknis yang terukur. Spesifikasi ini menjadi acuan utama (*blueprint*) bagi penulis dalam proses pengkodean fitur.

3.3.1 Kebutuhan Fungsional (*Functional Requirements*)

Kebutuhan fungsional mendefinisikan layanan spesifik yang harus disediakan oleh sistem serta respons sistem terhadap *input* tertentu. Berdasarkan modul utama yang telah ditetapkan, spesifikasi kebutuhan fungsional "ClashHub" dirinci pada Tabel 3.4 berikut:

Tabel 3.4 Spesifikasi Kebutuhan Fungsional Sistem

Kode	Modul Sistem	Deskripsi Kebutuhan Fungsional	Aktor Terkait
F-01	Manajemen Identitas	Sistem wajib memvalidasi kepemilikan akun *Clash of Clans* menggunakan token *API* resmi (*Player API Token*) untuk mencegah klaim akun palsu (*Identity Theft*).	Pengguna
F-02	Manajemen Identitas	Sistem secara otomatis menyinkronkan status klan pemain. Status "Agen Bebas" (*Free Agent*) hanya dapat diaktifkan jika *API* mengonfirmasi bahwa pemain tidak terikat dengan klan manapun.	Pemain
F-03	Pencarian Tim	Sistem menyediakan fitur pencarian dengan filter dinamis dan mekanisme pengurutan (*sorting*) berdasarkan: Level *Town Hall*, Skor Reputasi, dan Orientasi Visi (Kompetitif/Santai).	Pengguna
F-04	Manajemen Klan	Sistem melakukan agregasi data statistik klan (Riwayat Perang, Rasio Donasi, Aktivitas) dari *API* Supercell secara *real-time* dan menyimpannya ke basis data.	Sistem
F-05	Manajemen Klan	Sistem memvisualisasikan data kontribusi anggota dalam bentuk grafik interaktif pada dasbor, memudahkan pemantauan tren keaktifan.	Pemimpin Klan
F-06	Asisten Strategi (AI)	Sistem mengintegrasikan layanan *Generative AI* untuk menganalisis statistik perang dan menghasilkan rekomendasi strategi tekstual yang relevan dengan komposisi lawan.	Pemimpin Klan
F-07	Manajemen Turnamen	Sistem memvalidasi pendaftaran peserta turnamen secara otomatis. Pendaftaran ditolak jika level *Town Hall* akun tidak memenuhi syarat kategori turnamen (Validasi *Anti-Smurfing*).	Peserta
F-08	Manajemen Turnamen	Sistem memiliki algoritma *Bracket Generator* yang secara otomatis menyusun bagan pertandingan sistem gugur (*Single Elimination*) segera setelah kuota peserta terpenuhi.	Penyelenggara
F-09	Sistem Reputasi	Sistem memfasilitasi mekanisme penilaian dua arah (*Two-way Review*) yang menghasilkan Skor Reputasi kuantitatif untuk mengukur tingkat kepercayaan (*Trust Score*) pemain.	Pengguna
F-10	Pusat Pengetahuan	Sistem melakukan kurasi konten video strategi terbaru dari YouTube *API* berdasarkan *meta-tag Town Hall* yang relevan dengan profil pengguna.	Komunitas

Sumber: Hasil Olahan Peneliti (2025)

Pemetaan Kebutuhan terhadap Solusi Masalah:

a. F-03 & F-09 secara spesifik menjawab masalah Inefisiensi Rekrutmen dan Krisis Kepercayaan.

b. F-01, F-04, & F-07 menjawab masalah Validasi Data Manual dan *Human Error*.

c. F-06 menjawab kebutuhan Analisis Strategi Cerdas bagi pengguna tingkat lanjut.

3.3.2 Kebutuhan Non-Fungsional (*Non-Functional Requirements*)

Kebutuhan non-fungsional mendefinisikan batasan operasional dan standar kualitas yang harus dipenuhi sistem untuk menjamin kepuasan pengguna (*User Experience*). Mengingat target pengguna adalah komunitas *mobile gamer* yang kritis terhadap performa, aspek kecepatan dan responsivitas menjadi prioritas utama. Rincian spesifikasi kebutuhan non-fungsional sistem disajikan pada Tabel 3.5 berikut:

Tabel 3.5 Spesifikasi Kebutuhan Non-Fungsional Sistem

Kode	Kategori	Deskripsi Kebutuhan Non-Fungsional	Metrik / Teknologi
NF-01	Performa	Waktu pemuatan konten utama (*Largest Contentful Paint*) harus di bawah 2,5 detik pada jaringan 4G untuk memastikan retensi pengguna seluler.	*Server-Side Rendering* (*SSR*)
NF-02	Responsivitas	Sistem wajib memberikan umpan balik visual (*Skeleton Loading* atau *Spinner*) saat memproses data berat, khususnya saat menunggu respons analisis dari *Generative AI* (est. 3-5 detik).	*UI Feedback State*
NF-03	Keamanan	Mekanisme autentikasi pengguna wajib menggunakan protokol standar OAuth 2.0 (via Google Sign-In) untuk menjamin keamanan token sesi tanpa menyimpan password di basis data lokal.	*Federated Authentication*
NF-04	Keamanan	Sistem harus memvalidasi seluruh *input* pengguna di sisi server (*Server-Side Validation*) untuk mencegah injeksi skrip berbahaya (*XSS*) pada formulir pendaftaran turnamen.	*Input Sanitization*
NF-05	Skalabilitas	Infrastruktur sistem harus mampu menangani lonjakan trafik secara otomatis (*Auto-scaling*) saat periode pendaftaran turnamen dibuka serentak.	*Serverless Architecture*
NF-06	Keandalan	Sistem menerapkan mekanisme *Caching Data* (penyimpanan sementara) untuk data klan guna mematuhi batasan *Rate Limit API Clash of Clans* dan mencegah pemblokiran IP.	*Middleware Caching*
NF-07	Usabilitas	Antarmuka wajib mendukung Mode Gelap (*Dark Mode*) yang sinkron dengan pengaturan sistem operasi pengguna, mengakomodasi pola bermain di kondisi minim cahaya.	*CSS Media Query*
NF-08	Usabilitas	Desain antarmuka menerapkan prinsip *Mobile-First*, di mana area sentuh (*touch target*) tombol minimal berukuran 44x44 piksel sesuai standar aksesibilitas layar sentuh.	*Responsive Breakpoints*

Sumber: Hasil Olahan Peneliti (2025)

Justifikasi Teknis:

a. NF-01 & NF-08 dirancang khusus untuk demografi responden pelajar/mahasiswa yang 100% menggunakan *smartphone*.
b. NF-06 (*Caching*) adalah solusi teknis mutlak untuk menjaga stabilitas integrasi *API* Supercell yang membatasi jumlah *request* per detik.

c. NF-02 diterapkan untuk menjaga kenyamanan pengguna saat menunggu proses AI yang membutuhkan waktu komputasi lebih lama dibanding proses biasa.

3.3.3 Arsitektur Integrasi *API* (*System Integration*)

Untuk mendukung kapabilitas sistem dalam melakukan agregasi data dan analisis cerdas, penulis merancang arsitektur integrasi yang terpusat melalui *Middleware Server* (*API Proxy*) di lingkungan Next.js. Pendekatan ini dipilih untuk menjamin keamanan kunci akses (*API Key*) agar tidak terekspos di sisi klien (*browser*), serta untuk memusatkan logika *caching*. Arsitektur integrasi sistem terdiri dari tiga jalur komunikasi data utama:

a. Integrasi *API Clash of Clans* (Jalur Agregasi Data)

Integrasi ini berfungsi sebagai tulang punggung fitur Manajemen Klan dan Pencarian Tim. Mengingat adanya batasan *Rate Limit* dari server Supercell, penulis menerapkan algoritma "Sinkronisasi Cerdas" (*Smart Sync*) dengan logika sebagai berikut:

1. Intersepsi Permintaan: Saat pengguna mengakses profil klan/pemain, sistem terlebih dahulu memeriksa ketersediaan data di basis data lokal (Firestore).
2. Validasi *TTL* (*Time-To-Live*): Jika data lokal berusia kurang dari 5 menit (masih segar), sistem akan menyajikan data tersebut (*Cache Hit*).
3. Pembaruan Data: Jika data kadaluwarsa atau belum ada, *Middleware* akan meminta data terbaru ke *API* Supercell, menyimpannya ke Firestore, lalu menyajikannya ke pengguna.
4. Dukungan *Sorting*: Data *JSON* yang diambil mencakup atribut vital seperti Level *Town Hall*, Tropi, dan Peran, yang kemudian diindeks untuk keperluan algoritma Pengurutan (*Sorting*) dan Penyaringan (*Filtering*).

b. Integrasi Google Gemini AI (Jalur Analisis Cerdas)

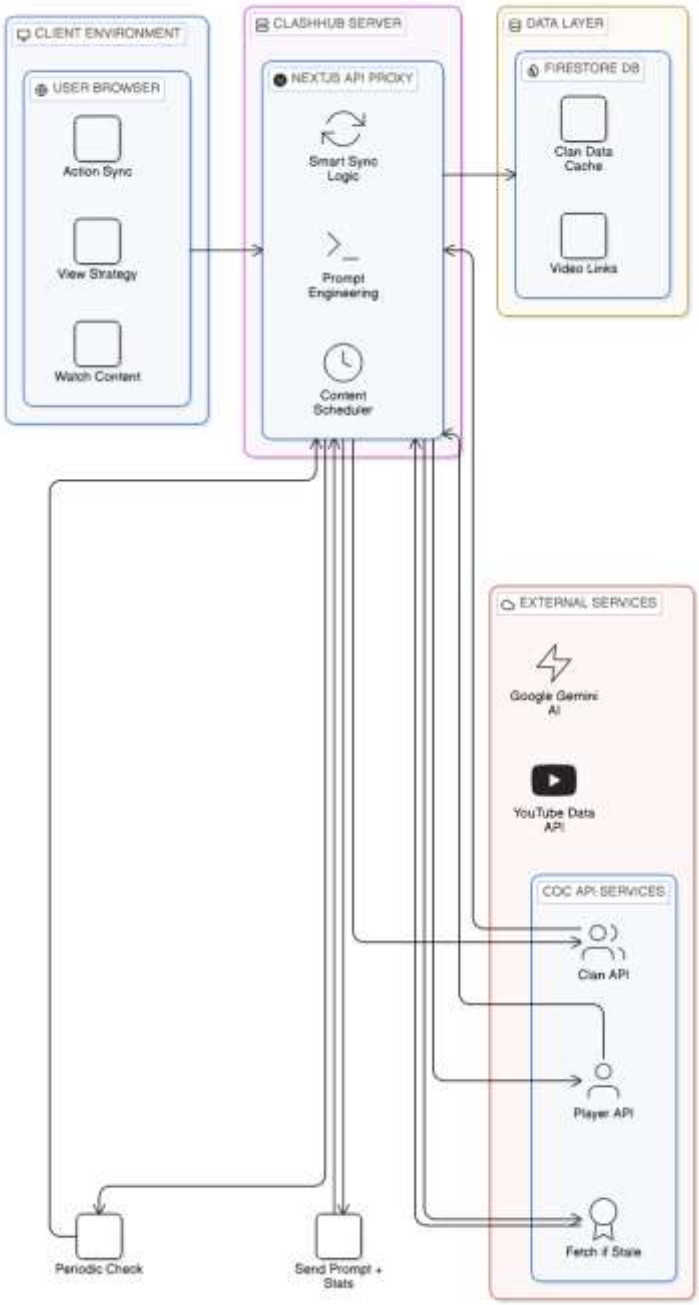
Jalur ini mendukung fitur Asisten Strategi. Mekanisme kerjanya melibatkan teknik *Prompt Engineering* dinamis:

1. Konstruksi Konteks: Sistem mengumpulkan ringkasan statistik perang klan (misal: "Rata-rata kerusakan serangan udara: 45%", "Tingkat kehancuran pertahanan: 80%").
2. Injeksi Instruksi: Data tersebut digabungkan dengan *System Prompt* khusus: "Bertindaklah sebagai pelatih *e-sports* profesional. Analisis data statistik berikut dan berikan 3 poin saran perbaikan taktis dalam Bahasa Indonesia."
3. Generasi Respon: *Generative AI* memproses instruksi tersebut dan mengembalikan narasi teks yang langsung ditampilkan di dasbor pemimpin klan.

c. Integrasi YouTube Data *API* (Jalur Konten Edukasi)

Integrasi ini digunakan pada modul Pusat Pengetahuan. Sistem secara berkala (dijadwalkan via *Cron Job*) memindai saluran kreator konten terpercaya untuk mengambil metadata video strategi terbaru. Hal ini memastikan referensi *Meta Game* yang tersedia di aplikasi selalu relevan dengan pembaruan gim terkini.

Visualisasi alur pertukaran data antar komponen dapat dilihat pada Gambar 3.4 berikut:



Gambar 3.4 Arsitektur Integrasi *API* dan Alur Data

Sumber: Hasil Olahan Peneliti (2025)

3.3.4 Analisis Aktor Sistem (*User Roles*)

Analisis aktor bertujuan untuk mengidentifikasi entitas pengguna yang berinteraksi dengan sistem serta batasan hak akses masing-masing. Berdasarkan modul fungsional yang dirancang, pengguna sistem diklasifikasikan ke dalam empat kategori aktor utama dengan

tingkat otorisasi bertingkat (*Role-Based Access Control*). Pembagian peran dan spesifikasi hak akses dijelaskan pada Tabel 3.6 berikut:

Tabel 3.6 Definisi Aktor dan Hak Akses Sistem

Aktor	Deskripsi Peran	Hak Akses Utama
Pengunjung (*Guest*)	Pengguna umum yang belum *login* atau mendaftar. Merepresentasikan publik yang mencari informasi awal.	Melihat halaman beranda (*Landing Page*) dan fitur unggulan, Mencari data klan/pemain di fitur Pencarian Tim (Mode: *Read-Only*), Memantau bagan turnamen yang sedang berlangsung, dan Mengakses konten video di Pusat Pengetahuan.
Pemain (*Verified Player*)	Pengguna terdaftar yang telah memverifikasi kepemilikan akun *Clash of Clans* melalui token *API*.	Mengelola profil statistik diri ("CV Pemain") dan preferensi visi, Mengaktifkan/Menonaktifkan status "Agen Bebas", Mendaftar turnamen komunitas, dan Memberikan ulasan reputasi terhadap klan yang pernah ditinggali.
Pemimpin Klan (*Clan Leader*)	Aktor dengan jabatan *Leader* atau *Co-Leader* di dalam gim. Memiliki otoritas manajerial penuh.	Memiliki seluruh akses Pemain, Mengakses dasbor manajemen klan, Melakukan sinkronisasi data statistik *API*, Menggunakan fitur Asisten Strategi AI, dan Menerima/Menolak lamaran anggota baru.
Penyelenggara (*Organizer*)	Aktor khusus (biasanya *Leader* senior) yang memiliki hak membuat kompetisi.	Membuat *event* turnamen baru & mengatur syarat *TH*, Memvalidasi daftar peserta (manual/otomatis), Menjalankan *Bracket Generator*, dan Memperbarui skor pertandingan.

Sumber: Hasil Olahan Peneliti (2025)

Sistem menerapkan prinsip pewarisan hak akses, di mana seorang Pemimpin Klan secara otomatis mewarisi seluruh kemampuan Pemain. Hal ini dirancang untuk memastikan fleksibilitas (*usability*), sehingga pemimpin klan tetap dapat berpartisipasi dalam aktivitas sosial atau turnamen individu tanpa perlu melakukan *logout* atau membuat akun ganda.

3.4 Perancangan Sistem

Sesuai dengan tahap Pemodelan Desain Cepat (*Modeling Quick Design*) pada metode *Prototype*, perancangan sistem dilakukan untuk memvisualisasikan interaksi fungsional dan logika alur kerja aplikasi sebelum implementasi kode. Penulis menggunakan standar notasi *Unified Modeling Language* (*UML*) untuk memastikan konsistensi pemahaman antara pengembang dan pemangku kepentingan.

3.4.1 Diagram *Use Case* (*Use Case Diagram*)

Diagram *Use Case* menggambarkan fungsionalitas sistem dari sudut pandang interaksi antara aktor (pengguna) dengan sistem. Diagram ini memetakan "apa yang dilakukan sistem" tanpa menjelaskan detail teknis "bagaimana caranya".

Definisi Aktor

Berdasarkan analisis aktor pada sub-bab 3.3.4, terdapat empat aktor utama yang terlibat:

a. Pengunjung (*Guest*): Aktor eksternal yang belum terautentikasi. Memiliki akses terbatas pada informasi publik.
b. Pemain (*Verified Player*): Pengguna terautentikasi yang telah memverifikasi akun gim. Mewarisi (*Generalization*) seluruh hak akses Pengunjung.
c. Pemimpin Klan (*Clan Leader*): Pemain dengan jabatan manajerial dalam gim. Mewarisi seluruh hak akses Pemain.
d. Penyelenggara (*Organizer*): Peran khusus yang diberikan kepada Pemimpin Klan tertentu untuk mengelola kompetisi.

Skenario *Use Case* Utama

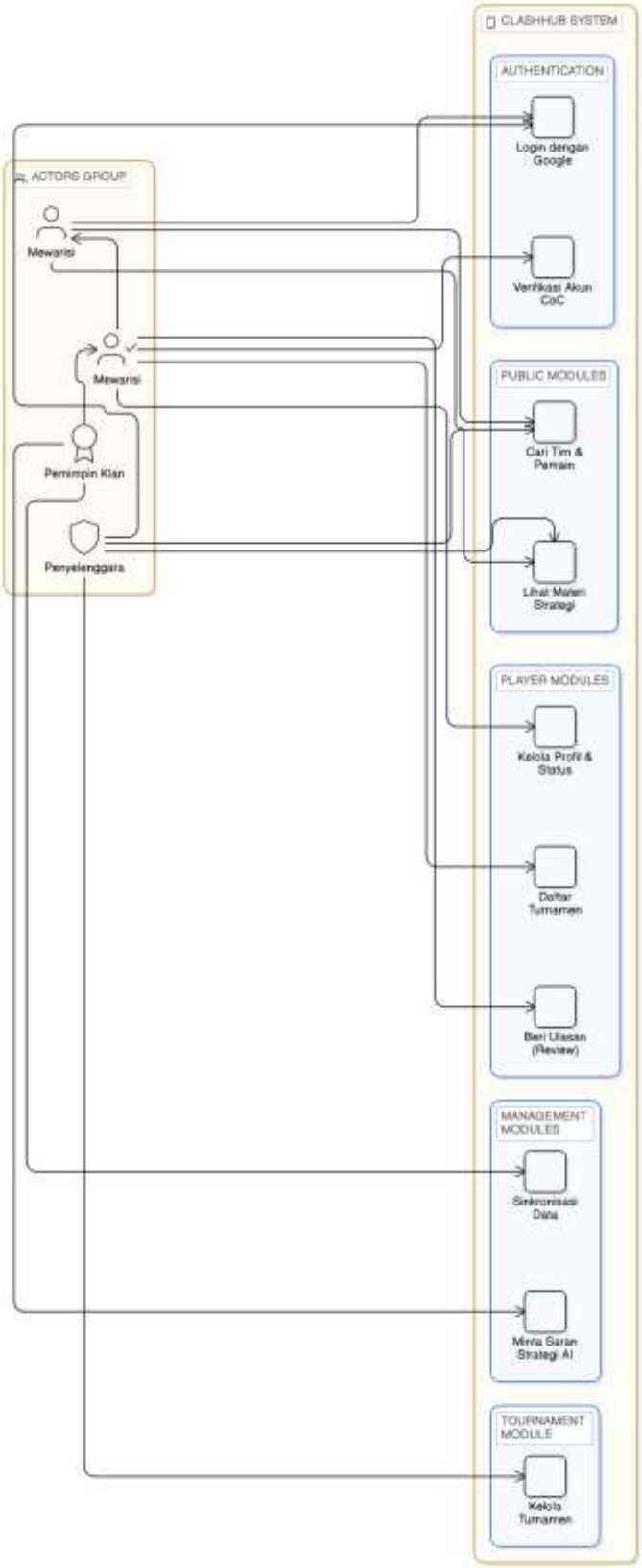
Sistem dirancang memiliki 10 *Use Case* utama yang merepresentasikan fitur inti aplikasi. Rincian skenario dijelaskan pada Tabel 3.7 berikut:

Tabel 3.7 Daftar *Use Case* Utama Sistem

Modul	***ID***	**Nama *Use Case***	**Deskripsi Singkat**	**Aktor Utama**
Autentikasi	UC-01	*Login* dengan Google	Pengguna masuk menggunakan akun Google (OAuth 2.0).	Semua Aktor
Autentikasi	UC-02	Verifikasi Akun CoC	Pengguna menautkan akun gim menggunakan Token *API* resmi.	Pemain
Profil	UC-03	Kelola Profil & Status	Pemain memperbarui data diri dan mengubah status rekrutmen ("Agen Bebas").	Pemain
Pencarian	UC-04	Cari & Filter Tim (*Sorting*)	Pengguna mencari klan/pemain dengan parameter filter (*TH*, Visi) dan pengurutan (Reputasi).	Semua Aktor
Manajemen	UC-05	Sinkronisasi Data *API*	Pemimpin klan memperbarui statistik anggota secara *real-time* dari server Supercell.	Pemimpin Klan
Manajemen	UC-06	Minta Saran Strategi (AI)	Pemimpin klan meminta analisis taktis dari *Generative AI*.	Pemimpin Klan
Turnamen	UC-07	Kelola Turnamen	Penyelenggara membuat *event*, memvalidasi peserta, dan menyusun bagan (*bracket*).	Penyelenggara
Turnamen	UC-08	Daftar Turnamen	Pemain mendaftarkan diri/tim pada kompetisi yang aktif.	Pemain
Sosial	UC-09	Beri Ulasan Reputasi	Pengguna memberikan penilaian bintang dan ulasan kinerja terhadap klan/pemain.	Pemain, *Leader*
Edukasi	UC-10	Lihat Materi Strategi	Pengguna mengakses video kurasi strategi dari *Knowledge Hub*.	Semua Aktor

Sumber: Hasil Olahan Peneliti (2025)

Visualisasi hubungan antara aktor dan *use case* dapat dilihat pada Gambar 3.5 berikut:



Gambar 3.5 Diagram *Use Case* Sistem ClashHub

Sumber: Hasil Olahan Peneliti (2025)

3.4.2 Diagram Aktivitas (*Activity Diagram*)

Diagram aktivitas digunakan untuk menggambarkan alur kerja (*workflow*) atau urutan langkah-langkah yang terjadi dalam sebuah proses bisnis sistem. Diagram ini membantu memahami bagaimana pengguna berinteraksi dengan fitur-fitur utama sistem "ClashHub". Perancangan diagram aktivitas difokuskan pada lima proses utama yang menjadi inti dari aplikasi ini.

Aktivitas Sinkronisasi Data (Agregasi *API*)

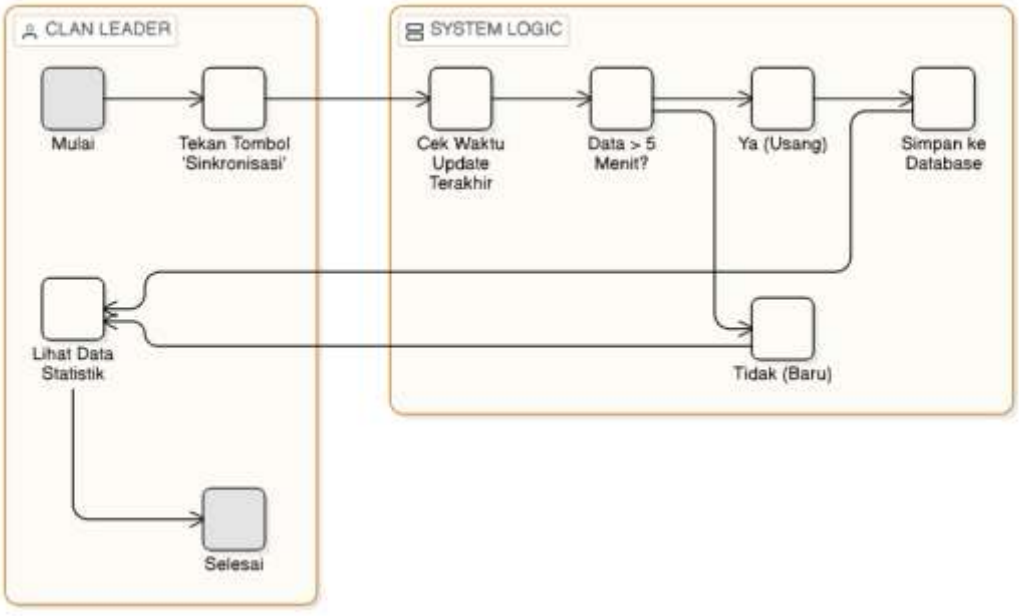
Alur ini menggambarkan mekanisme *Smart Sync* untuk mengatasi batasan *rate limit* saat memperbarui data klan.

a. Aktor: Pemimpin Klan.

b. Alur Logika:

1. Pemimpin menekan tombol "Sinkronisasi" pada dasbor.
2. Sistem memeriksa *timestamp* data terakhir di *database*.
3. Percabangan (*Decision*): jika data < 5 menit maka sistem memuat data dari *Cache* lokal (Hemat *API*) dan jika data > 5 menit maka sistem menarik data terbaru dari *API* Supercell dan memperbarui *database*.
4. Sistem menampilkan statistik visual kepada pengguna.

Visualisasi aktivitas sinkronisasi data klan dapat dilihat pada Gambar 3.6 berikut:



Gambar 3.6 Diagram Aktivitas Sinkronisasi Data Klan

Sumber: Hasil Olahan Peneliti (2025)

Aktivitas Analisis Strategi (*Generative AI*)

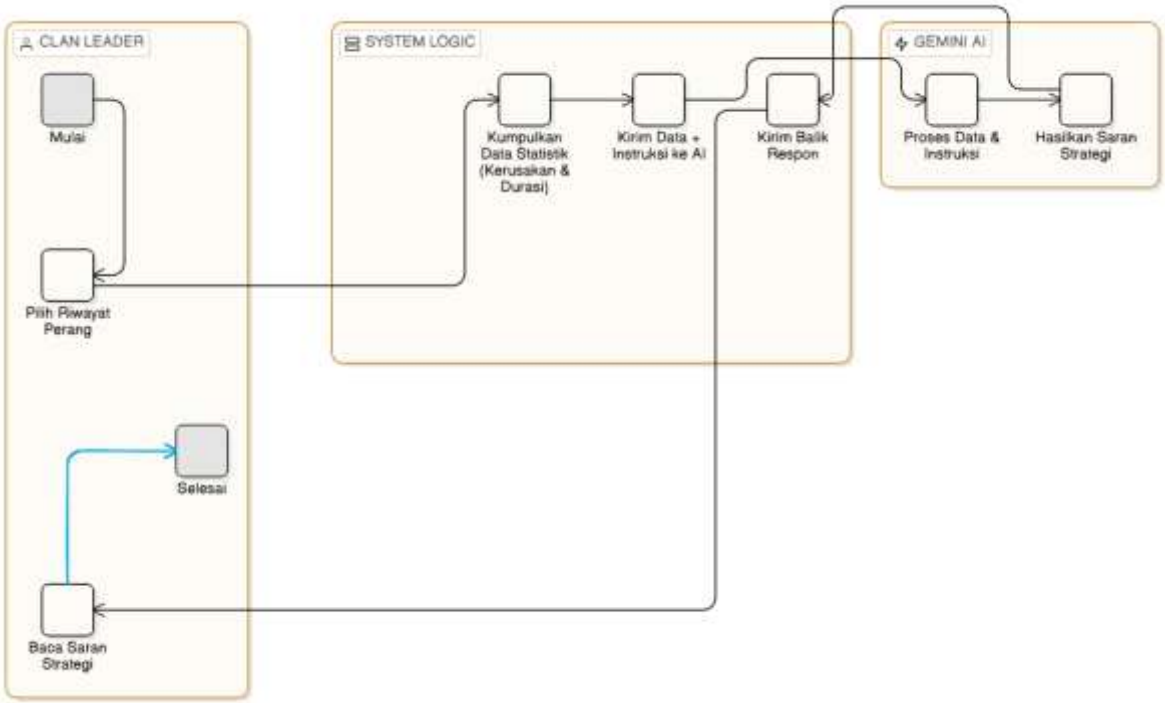
Alur ini menjelaskan proses transformasi data angka menjadi narasi strategi.

a. Aktor: Pemimpin Klan.

b. Alur Logika:

1. Pemimpin memilih riwayat perang spesifik.
2. Sistem mengemas statistik perang menjadi format teks *JSON*.
3. Sistem mengirimkan *Prompt Context* ke layanan Google Gemini.
4. AI memproses analisis dan mengembalikan teks saran.
5. Sistem merender respon AI ke antarmuka pengguna.

Visualisasi aktivitas analisis strategi dapat dilihat pada Gambar 3.7 berikut:



Gambar 3.7 Diagram Aktivitas Asisten Strategi

Sumber: Hasil Olahan Peneliti (2025)

Aktivitas Pendaftaran Turnamen (Validasi Otomatis)

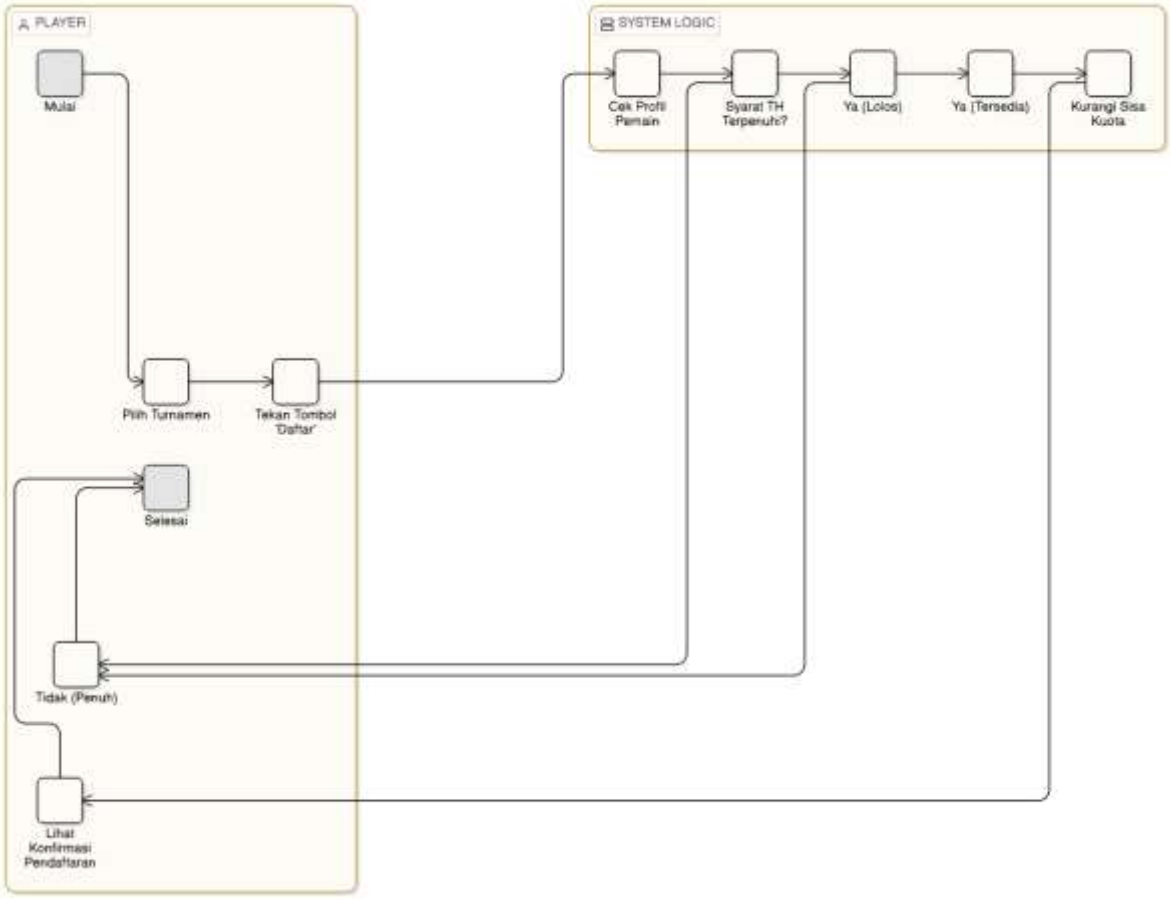
Alur ini menggambarkan mekanisme *Anti-Smurfing* untuk mencegah kecurangan level akun.

a. Aktor: Pemain.

b. Alur Logika:

1. Pemain memilih turnamen dan menekan "Daftar".
2. Sistem mengambil data profil terbaru pemain via *API*.
3. Percabangan (*Decision*): jika *TH Player* < Syarat Turnamen maka sistem menolak pendaftaran (Validasi Gagal) dan jika *TH Player* >= Syarat maka sistem mengecek kuota slot.
4. Jika valid, sistem menyimpan data peserta dan memperbarui sisa slot.

Visualisasi aktivitas pendaftaran turnamen dapat dilihat pada Gambar 3.8 berikut:



Gambar 3.8 Diagram Aktivitas Pendaftaran Turnamen

Sumber: Hasil Olahan Peneliti (2025)

Aktivitas Pencarian Tim (*Sorting* & *Filtering*)

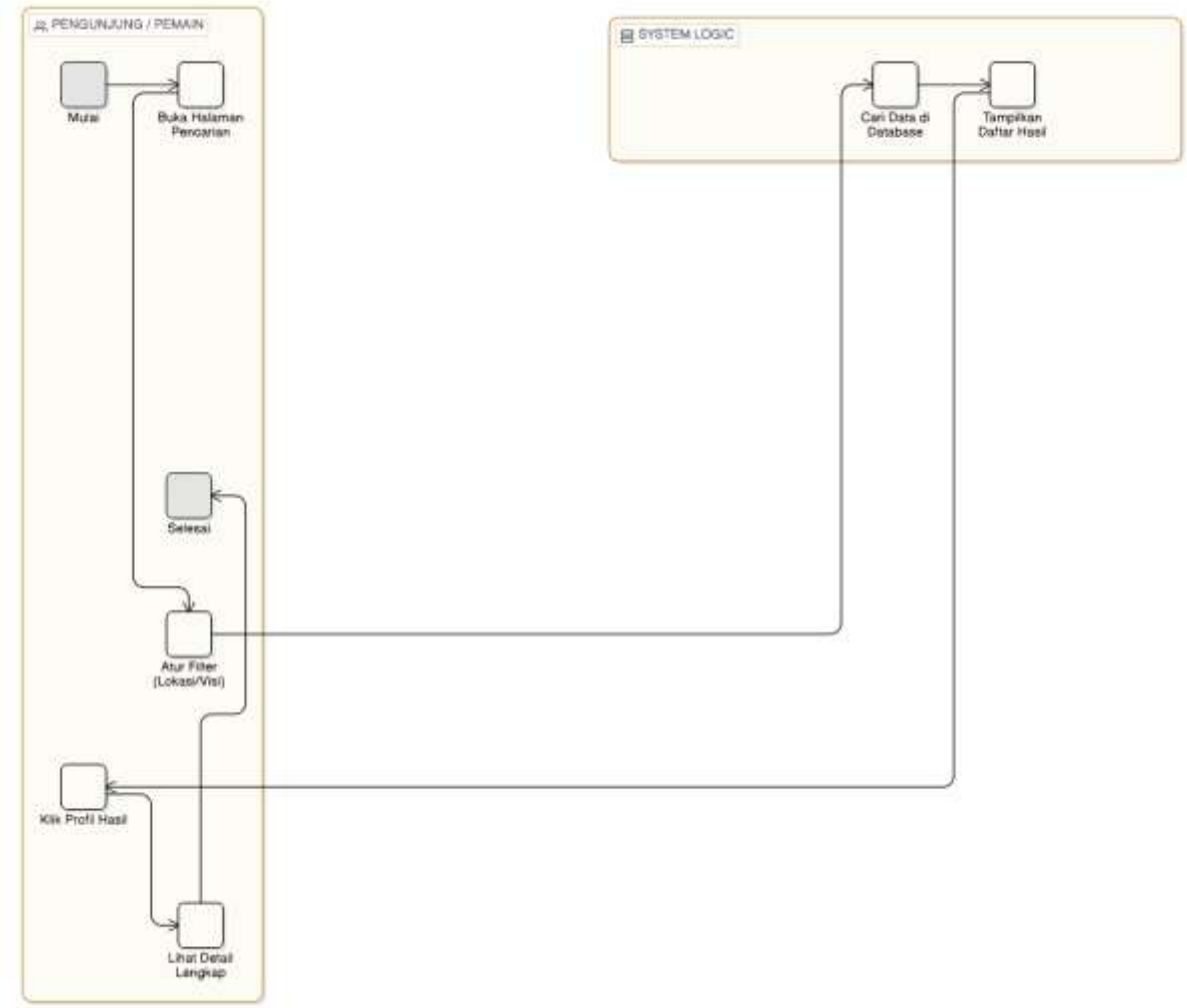
Proses ini menjelaskan bagaimana pengguna menemukan klan atau pemain yang cocok.

a. Aktor: Pengguna.

b. Alur Logika:

1. Pengguna membuka halaman pencarian.
2. Pengguna mengatur Filter (Lokasi, Visi) dan Pengurutan/*Sorting* (Skor Reputasi Tertinggi / *TH* Tertinggi).
3. Sistem melakukan *query* ke Firestore berdasarkan parameter tersebut.
4. Sistem menampilkan daftar hasil yang telah terurut.
5. Pengguna melihat detail profil yang relevan.

Visualisasi aktivitas pencarian tim dapat dilihat pada Gambar 3.9 berikut:



Gambar 3.9 Diagram Aktivitas Pencarian Tim dengan *Sorting*

Sumber: Hasil Olahan Peneliti (2025)

Aktivitas Pemberian Reputasi (*Trust System*)

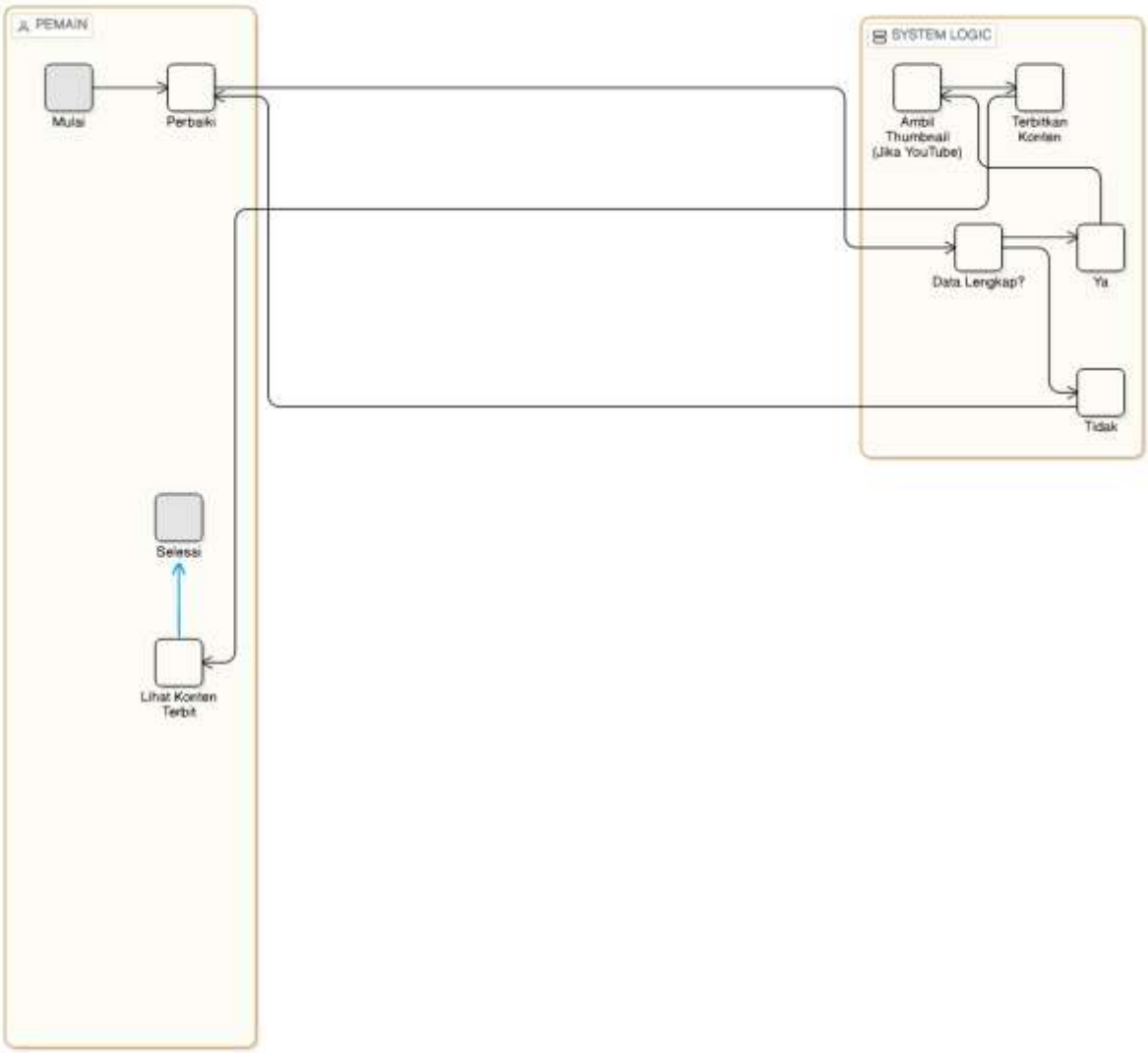
Alur ini menggantikan fungsi manual dalam membangun kepercayaan antar pemain.

a. Aktor: Pemain / Pemimpin.

b. Alur Logika:

1. Pengguna membuka profil pemain atau klan sasaran.
2. Pengguna memberikan *Rating* (Bintang 1-5) dan ulasan teks.
3. Sistem memvalidasi apakah pengguna berhak memberi ulasan (misal: akun terverifikasi).
4. Sistem menghitung ulang rata-rata Skor Reputasi (*Trust Score*) target.
5. Skor reputasi baru ditampilkan secara publik.

Visualisasi aktivitas pemberian reputasi dapat dilihat pada Gambar 3.10 berikut:



Gambar 3.10 Diagram Aktivitas Sistem Reputasi

Sumber: Hasil Olahan Peneliti (2025)

3.5 Perancangan Basis Data (*Database Design*)

Perancangan basis data dalam sistem "ClashHub" mengadopsi pendekatan NoSQL menggunakan layanan Google *Cloud* Firestore. Keputusan ini didasarkan pada karakteristik

data gim *Clash of Clans* yang bersumber dari *API* dengan format *JSON* (*JavaScript Object Notation*) yang dinamis dan bersarang (*nested*). Model dokumen (*Document-Oriented*) dipilih karena memungkinkan fleksibilitas skema dan performa baca yang lebih cepat untuk aplikasi *mobile* dibandingkan basis data relasional (SQL). Perancangan data dikelompokkan ke dalam koleksi (*Collections*) utama berdasarkan entitas sistem.

3.5.1 Skema Koleksi Pengguna (*Users Collection*)

Koleksi *users* berfungsi sebagai repositori utama profil pengguna. Dokumen ini menyimpan data gabungan antara identitas autentikasi (Google) dan statistik permainan (Supercell). Skema ini dirancang khusus untuk mendukung algoritma Pengurutan (*Sorting*) dan Penyaringan (*Filtering*) pada fitur Pencarian Tim. Rincian struktur dokumen pengguna dijelaskan pada Tabel 3.8 berikut:

Tabel 3.8 Struktur Data Dokumen *Users* (*Path*: *users*/{*uid*})

Nama *Field*	Tipe Data	Deskripsi & Fungsi	Sumber Data
Uid	*String* (PK)	Kunci primer unik yang diterbitkan oleh Firebase Auth saat *login*.	Google Auth
displayName	*String*	Nama tampilan pengguna di aplikasi.	*Input* Pengguna
playerTag	*String*	Tagar unik akun gim (contoh: #ABC123). *Field* ini dikunci (*Immutable*) setelah verifikasi untuk mencegah manipulasi data turnamen.	*Input* & Validasi *API*
thLevel	*Number*	Tingkat *Town Hall* pemain. Parameter utama untuk Filter Pencarian dan syarat validasi Turnamen.	CoC *API* (*Sync*)
trophies	*Number*	Jumlah piala pemain saat ini. Digunakan sebagai parameter sekunder untuk mengukur keaktifan pemain.	CoC *API* (*Sync*)
warStars	*Number*	Total bintang perang yang pernah diraih. Indikator pengalaman tempur (*Veterancy*).	CoC *API* (*Sync*)
reputation	*Number*	Skor Reputasi (*Trust Score*) akumulatif dari ulasan komunitas. Digunakan untuk *Sorting* kredibilitas pemain.	Sistem Reputasi
role	*String*	Peran akses di aplikasi: '*guest*', '*member*', '*leader*', atau '*organizer*'.	Sistem
isFreeAgent	*Boolean*	Status penanda apakah pemain sedang mencari klan (*True*) atau tidak (*False*).	*Input* Pengguna
playStyle	*Array*	Label gaya bermain (misal: ['Kompetitif', 'Donatur']) untuk pencocokan visi.	*Input* Pengguna
cachedHeroes	*Map*	Salinan lokal level *Hero* (Raja, Ratu, Wali) untuk mempercepat tampilan profil tanpa *request API* berulang.	CoC *API* (*Cache*)
lastUpdated	*Timestamp*	Penanda waktu sinkronisasi terakhir untuk manajemen *Cache Expiry*.	Sistem

Sumber: Hasil Olahan Sendiri (2025)

Strategi Implementasi:

a. Integritas Data (*Anti-Smurfing*): *Field* krusial seperti *thLevel*, *trophies*, dan *warStars* bersifat *Read-Only* bagi pengguna. Nilainya hanya dapat diperbarui oleh fungsi sistem (*Server Action*) yang memanggil *API* Supercell, sehingga menjamin validitas data saat pendaftaran turnamen.
b. Optimasi Performa: Teknik *Data Embedding* diterapkan pada *field cachedHeroes*. Alih-alih menyimpannya di koleksi terpisah, data *hero* disimpan di dalam dokumen *user* agar profil pemain dapat dimuat dalam satu kali pengambilan (*single fetch*), mendukung prinsip *User Experience* yang responsif.

3.5.2 Skema Koleksi Klan (*Managed Clans Collection*)

Koleksi *managedClans* merupakan repositori data untuk klan yang telah terdaftar dan dikelola menggunakan sistem "ClashHub". Berbeda dengan data mentah *API* yang hanya bersifat statistik, skema ini diperkaya dengan atribut sosial dan manajerial untuk mendukung algoritma Pencarian Tim Presisi. Struktur dokumen klan dirancang untuk mendukung operasi *filtering* dan *sorting* yang efisien, sebagaimana dirinci pada Tabel 3.9 berikut:

Tabel 3.9 Struktur Data Dokumen Klan (*Path*: *managedClans*/{clanTag})

Nama *Field*	Tipe Data	Deskripsi & Fungsi	Sumber Data
Tak	*String* (PK)	Tagar unik klan (misal: #2QU...) yang berfungsi sebagai *Primary Key* untuk sinkronisasi *API*.	*Input* Pengguna
name	*String*	Nama resmi klan.	CoC *API*
ownerUid	*String*	*UID* dari Pemimpin Klan yang memegang hak akses manajerial sistem.	Firebase Auth
clanLevel	*Number*	Level klan saat ini. Parameter utama untuk *Sorting* kekuatan klan.	CoC *API* (*Sync*)
warWinStreak	*Number*	Jumlah kemenangan perang beruntun. Indikator untuk filter "Klan Kompetitif".	CoC *API* (*Sync*)
averageTownHall	*Number*	Rata-rata tingkat *TH* anggota. Digunakan untuk filter "Kesesuaian Level" bagi pelamar.	Sistem (Kalkulasi)
reputationScore	*Number*	Skor Reputasi klan berdasarkan ulasan mantan anggota. Indikator "Kesehatan Lingkungan Klan" (*Non-Toxic Environment*).	Sistem Reputasi
vision	*Enum*	Label visi utama: 'Kompetitif', 'Santai', '*Farming*', atau '*Push Trophy*'.	*Input* Pengguna
recruitingStatus	*Enum*	Status penerimaan anggota: '*Open*', '*Invite Only*', atau '*Closed*'.	*Input* Pengguna
socialLinks	*Map*	Tautan media sosial eksternal (*Discord*, *WhatsApp*) untuk komunikasi cepat.	*Input* Pengguna
description	*Text*	Deskripsi profil klan yang lebih mendalam (mendukung *Rich Text*) dibandingkan deskripsi *in-game*.	*Input* Pengguna
memberTags	*Array*	Daftar tagar anggota aktif. Digunakan untuk *indexing* relasi anggota-klan.	CoC *API*
lastSynced	*Timestamp*	Penanda waktu pembaruan data terakhir dari server Supercell.	Sistem

Sumber:Hasil Olahan Peneliti (2025)

Strategi Optimasi dan Arsitektur Data:

a. Pemisahan Data (*Data Partitioning*)

Penulis menerapkan strategi pemisahan antara "Data Profil" (Ringan) dan "Data Log" (Berat).

1. Dokumen Utama: Hanya menyimpan data yang diperlukan untuk tampilan kartu klan di halaman pencarian.

2. *Sub-Collection*: Data berat seperti riwayat perang lengkap (*warLog*) dan donasi harian (*donations*) disimpan dalam *Sub-Collection* terpisah (*managedClans*/{*tag*}/logs). Strategi ini memastikan halaman pencarian tim dapat memuat ratusan klan dalam hitungan milidetik tanpa membebani *bandwidth* pengguna.

b. Kalkulasi Sisi Server (*Server-Side Calculation*)

Field seperti *averageTownHall* tidak tersedia langsung dari *API* Supercell. Oleh karena itu, sistem melakukan kalkulasi otomatis (*Cloud Function*) setiap kali sinkronisasi terjadi, sehingga pengguna bisa langsung melakukan *sorting* berdasarkan rata-rata *TH* tanpa perlu menunggu perhitungan di sisi klien.

3.5.3 Skema Koleksi Turnamen (*Tournaments Collection*)

Koleksi *tournaments* dirancang untuk memfasilitasi manajemen kompetisi secara *end-to-end*, mulai dari pendaftaran peserta, validasi syarat otomatis, hingga visualisasi bagan pertandingan (*bracket*). Struktur data ini menjadi landasan fitur Administrasi Turnamen Otomatis yang bertujuan mengeliminasi *human error* panitia. Struktur dokumen induk untuk turnamen dijelaskan pada Tabel 3.10 berikut:

Tabel 3.10 Struktur Data Dokumen Turnamen (*Path*: *tournaments*/{*tournamentId*})

Nama *Field*	Tipe Data	Deskripsi & Fungsi
Ide	*String* (PK)	Kode identitas unik turnamen (UUID).
Title	*String*	Judul resmi turnamen.
status	*Enum*	Siklus hidup turnamen: '*Draft*', 'RegistrationOpen', '*OnGoing*', '*Completed*', atau '*Cancelled*'.
format	*Enum*	Model kompetisi: '1v1' (Solo) atau '5v5' (Tim).
thMinRequirement	*Number*	Syarat Minimum *Town Hall*. Digunakan sistem untuk menolak pendaftaran otomatis jika akun peserta tidak memenuhi syarat (Validasi *Anti-Smurfing*).
thMaxRequirement	*Number*	Batas Maksimum *Town Hall*. Mencegah pemain *overpowered* masuk ke kategori pemula.
organizerUid	*String*	*UID* penyelenggara yang memiliki hak akses *admin* turnamen ini.
hostClanTags	*Array*	Daftar tagar klan fasilitas milik panitia yang disiapkan sebagai wadah pelaksanaan *Friendly War*.
maxParticipants	*Number*	Kuota maksimal peserta (misal: 16, 32, 64) untuk keperluan pembentukan bagan biner sempurna.
registrationDeadline	*Timestamp*	Batas waktu pendaftaran. Sistem otomatis mengubah status menjadi '*OnGoing*' atau '*Closed*' saat waktu terlewati.
bracketData	*Map*	Struktur data pohon (*tree structure*) yang menyimpan relasi antar pertandingan untuk visualisasi bagan.

Sumber: Hasil Oalahan Peneliti (2025)

Mengingat kompleksitas relasi data dalam sebuah kompetisi, penulis menerapkan strategi pemisahan data ke dalam *Sub-Collections* untuk menjaga performa dokumen induk:

a. Sub-koleksi Peserta (*participants*): Menyimpan daftar tim atau individu yang mendaftar. Setiap dokumen dilengkapi *field verificationStatus* ('*Pending*', '*Approved*', 'Rejected'). Ini memungkinkan panitia memantau status validasi peserta secara *real-time*.
b. Sub-koleksi Pertandingan (*matches*): Menyimpan data *node* bagan pertandingan. Struktur dokumen dirancang untuk mendukung algoritma *Bracket Generator*:
    1. *round*: Nomor babak (misal: Babak 16 Besar, Perempat Final).
    2. *player1Id* & *player2Id*: Identitas peserta yang bertanding.
    3. *winnerId*: Pemenang yang akan melaju ke *node* pertandingan berikutnya (*Next Match Logic*).
    4. *videoLink*: Tautan bukti rekaman pertandingan (YouTube) untuk transparansi hasil.

3.5.4 Skema Koleksi Ulasan & Reputasi (*Reviews Collection*)

Koleksi *reviews* adalah implementasi teknis untuk mengatasi masalah krisis kepercayaan (*trust issue*) dan fenomena "Kutu Loncat" (*hopper*) dalam komunitas. Koleksi ini memfasilitasi sistem penilaian dua arah (*Two-way Review*): klan menilai pemain, dan pemain menilai klan. Data ini menjadi basis perhitungan Skor Reputasi yang digunakan dalam algoritma pengurutan (*sorting*) pencarian tim. Struktur dokumen ulasan dirancang sebagai berikut pada Tabel 3.11:

Tabel 3.11 Struktur Data Dokumen Ulasan (*Path*: *reviews*/{*reviewId*})

Nama *Field*	Tipe Data	Deskripsi & Fungsi
Ide	*String* (PK)	Kode identitas unik ulasan (UUID).
targetType	*Enum*	Penanda objek yang dinilai: '*USER*' (untuk pemain) atau '*CLAN*' (untuk klan).
targetId	*String*	*UID* Pemain atau *Tag* Klan yang sedang dinilai.
authorUid	*String*	*UID* pengguna yang menulis ulasan.
authorName	*String*	Nama penulis ulasan (disimpan denormalisasi agar tidak perlu *join query* ke tabel *user* saat menampilkan *list* ulasan).
rating	*Number*	Nilai kuantitatif skala 1-5. Komponen utama pembentuk skor reputasi.
comment	*Text*	Narasi pengalaman penulis terhadap target.
isVerified	*Boolean*	Status penanda apakah ulasan ini berasal dari rekan satu tim yang valid (pernah satu klan) untuk mencegah *spam review*.
createdAt	*Timestamp*	Waktu ulasan dibuat.

Sumber: Hasil Olahan Sendiri (2025)

Agar fitur *sorting* berdasarkan reputasi dapat berjalan cepat (O(1)) tanpa perlu menghitung ulang rata-rata setiap kali halaman dibuka, sistem menerapkan logika *Event Trigger* menggunakan *Cloud Functions*:

a. *Trigger*: Setiap kali dokumen baru ditambahkan ke koleksi *reviews*.
b. Kalkulasi: Server mengambil semua ulasan terkait *targetId* tersebut dan menghitung nilai rata-rata (*averageRating*).
c. *Update*: Server memperbarui *field reputation* pada dokumen induk di koleksi *users* atau *managedClans*.
d. Hasil: Saat pengguna melakukan pencarian dengan filter "Reputasi Tertinggi", sistem cukup membaca nilai akhir yang sudah tersedia di profil target.

3.5.5 Skema Koleksi Pusat Pengetahuan (*Posts Collection*)

Anak Subbab dengan penomoran menggunakan 3 angka. Setelah Anak Subbab penomoran cukup dengan menggunakan huruf abjad besar dan dicetak tebal.

Koleksi *posts* dirancang sebagai repositori konten untuk fitur Pusat Pengetahuan (*Knowledge Hub*). Fitur ini mewadahi pertukaran informasi taktis antar anggota, memungkinkan pemain veteran membagikan komposisi pasukan (*Army Composition*) dan strategi penyerangan. Struktur dokumen postingan dirancang untuk mendukung konten multimedia (Video/*Deep Link*) sebagaimana dirinci pada Tabel 3.12 berikut:

Tabel 3.12 Struktur Data Dokumen Postingan (*Path*: *posts*/{*postId*})

Nama *Field*	Tipe Data	Deskripsi & Fungsi
Ide	*String* (PK)	Kode identitas unik postingan (UUID).
title	*String*	Judul strategi atau topik diskusi.
content	*Text*	Isi konten utama (mendukung format Markdown untuk penulisan panduan yang rapi).
authorId	*String*	*UID* penulis postingan (Relasi ke koleksi *users*).
authorName	*String*	Nama penulis (Denormalisasi) untuk mempercepat *rendering* daftar artikel tanpa *join query*.
category	*Enum*	Kategori konten: '*AttackStrategy*', '*BaseDesign*', '*TeamManagement*', atau '*General*'.
tags	*Array*	Label taktis spesifik untuk *Filtering*, misal: ['TH13', 'Udara', 'Darat', '*QueenWalk*'].
videoUrl	*String*	Tautan video YouTube. Sistem secara otomatis mengambil metadata (Judul & *Thumbnail*) via YouTube Data *API*.
troopLink	*String*	*Deep Link* resmi gim (link.clashofclans.com/...). Memungkinkan pengguna menyalin pasukan langsung ke dalam gim dengan satu klik.
likes	*Array*	Daftar *UID* pengguna yang menyukai postingan (untuk fitur "Strategi Terpopuler").
replyCount	*Number*	Penghitung jumlah komentar (Agregasi) untuk tampilan *preview* di beranda.
createdAt	*Timestamp*	Waktu postingan diterbitkan.

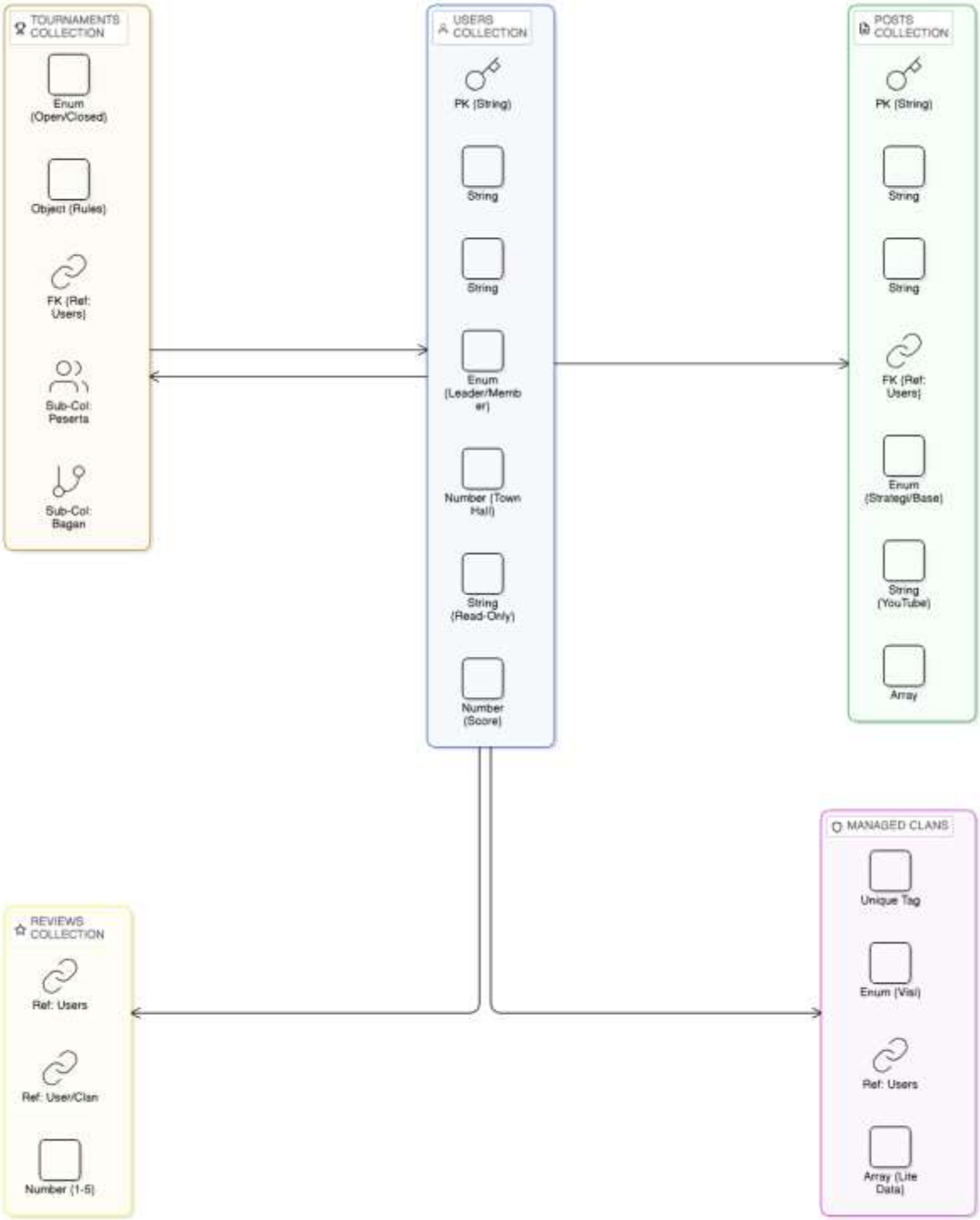
Sumber: Hasil Olahan Peneliti (2025)

Untuk menjaga performa pemuatan halaman utama (*List View*), data komentar tidak disimpan di dalam dokumen postingan (*Embedding*), melainkan menggunakan *Sub-Collection*:

a. *Path*: *posts*/{*postId*}/*replies*/{*replyId*}.

b. Alasan: Dokumen postingan tetap ringan (ukuran kecil) meskipun memiliki ribuan komentar. Data komentar hanya diunduh (*fetch*) saat pengguna membuka detail postingan (*Lazy Loading*).

Meskipun menggunakan basis data berorientasi dokumen yang fleksibel, sistem tetap mempertahankan integritas referensi antar entitas. Hubungan logis antara Pengguna, Klan, Turnamen, dan Postingan digambarkan dalam Gambar 3.11 berikut:



Gambar 3.11 Diagram Relasi Entitas NoSQL (*Logical Schema*)

Sumber: Hasil Olahan Peneliti (2025)

3.6 Perancangan Antarmuka (*User Interface Design*)

Tahap Pemodelan Desain Cepat difokuskan pada perancangan antarmuka pengguna (*User Interface*) yang ergonomis dan intuitif. Mengacu pada data demografi responden (84% pengguna ponsel), penulis menerapkan prinsip *Mobile-First Design*. Implementasi desain dilakukan menggunakan kerangka kerja Tailwind *CSS* untuk menjamin konsistensi visual dan responsivitas pada berbagai ukuran layar.

3.6.1 Halaman Profil Pemain (*Player Profile Dashboard*)

Halaman profil dirancang sebagai solusi atas masalah "kurangnya transparansi data". Berbeda dengan profil *in-game* yang statis, profil ClashHub berfungsi sebagai "CV Profesional Pemain" yang menggabungkan statistik gim (*API*) dengan rekam jejak sosial (Reputasi).

Komponen Antarmuka Utama

Susunan antarmuka menggunakan pola *Stacked Cards* (Kartu Bertumpuk) yang optimal untuk *scrolling* vertikal di ponsel:

a. Kartu Identitas & Status Verifikasi

   Menampilkan foto profil, *IGN* (*In-Game Name*), dan Lencana Verifikasi (*Verified Badge*). Lencana ini menandakan bahwa kepemilikan akun telah divalidasi via Token *API*, sehingga data *Town Hall* dijamin akurat untuk keperluan validasi turnamen. Terdapat juga *toggle* status "Agen Bebas" bagi pemain yang sedang mencari klan.

b. Kartu Statistik Tempur (*Game Stats*)

   Visualisasi data vital yang diambil *real-time* dari server Supercell:

   1. Level *Town Hall* & Level *Hero* (Raja, Ratu, Wali).
   2. Total Bintang Perang (*War Stars*) sebagai indikator pengalaman.
   3. Grafik Aktivitas Donasi untuk menilai kemurahan hati pemain.

c. Kartu Skor Reputasi (*Trust Score*)

   Elemen visual yang menampilkan rata-rata penilaian bintang (1-5) dari komunitas. Kartu ini dilengkapi label perilaku (misal: "Disiplin", "Pasif") untuk membantu pemimpin klan menilai karakter calon anggota sebelum merekrut. Skor ini menjadi parameter utama dalam algoritma *sorting* pencarian tim.

d. Kartu Riwayat Tim (*Hopper Detector*)

   Fitur *Timeline* yang menampilkan riwayat perpindahan klan pemain dalam kurun waktu tertentu. Visualisasi ini dirancang khusus untuk membantu pemimpin klan mendeteksi

pola perilaku "Kutu Loncat" (*Hopper*) yang sering berpindah-pindah klan dalam waktu singkat.

Desain Interaksi

a. Navigasi Tab: Informasi kompleks dipecah ke dalam tab ("Statistik", "Riwayat", "Ulasan") untuk mengurangi panjang halaman (*scroll depth*).
b. *Feedback* Visual: Menggunakan animasi *Skeleton Loading* saat sistem sedang melakukan sinkronisasi data *API*, memberikan persepsi performa yang cepat kepada pengguna.

Visualisasi desain interaksi dapat dilihat pada Gambar 3.12 berikut:

Gambar 3.12 Rancangan Antarmuka Halaman Profil Pemain
Sumber: Hasil Olahan Peneliti (2025)

3.6.2 Pusat Komunitas (*Team Hub* / *Clan Hub*)

Halaman ini dirancang sebagai solusi komprehensif untuk masalah "Inefisiensi Rekrutmen". Dalam tahap Pemodelan Desain Cepat, fokus utama perancangan adalah "*Discoverability*" (Kemudahan Penemuan). Antarmuka dirancang agar pengguna dapat melakukan penyaringan (*filtering*) dan pengurutan (*sorting*) data ribuan klan dalam hitungan detik.

Komponen Antarmuka Utama

Halaman ini menggunakan navigasi berbasis Tab ("Cari Klan" vs "Cari Pemain") dengan komponen interaktif sebagai berikut:

a. Bilah Pencarian & Filter Cerdas (*Smart Search Bar*)

Fitur ini membedakan ClashHub dengan pencarian *in-game* yang terbatas. Bilah ini menyediakan parameter kontrol ganda:

1. Filter (Penyaringan): Pengguna dapat mempersempit hasil berdasarkan Lokasi (Provinsi), Visi (Kompetitif/Santai), dan Level Klan Minimum.
2. Sortir (Pengurutan): Sesuai masukan validasi, sistem menyediakan opsi pengurutan dinamis: reputasi tertinggi untuk menampilkan klan/pemain dengan bintang kepercayaan paling banyak (Solusi *Trust Issue*), *town hall* tertinggi untuk menampilkan pemain dengan akun terkuat (Solusi Kebutuhan Turnamen), dan terbaru untuk menampilkan klan yang baru mendaftar.

b. Kartu Direktori Klan (*Clan Cards*)

Menampilkan daftar klan dalam *grid* responsif. Setiap kartu memuat informasi vital:

1. Nama & Level Klan.
2. Skor Reputasi (Bintang 1-5): Ditampilkan mencolok di pojok kartu.
3. Label Visi (misal: "*Badge* Kompetitif" berwarna emas).
4. Indikator "Slot Tersedia" (misal: 45/50 Anggota).

c. Bursa Pemain (*Free Agent Market*)

Tab khusus yang menampilkan daftar pemain berstatus "Agen Bebas". Tampilan ini dirancang bagi Pemimpin Klan untuk "belanja" pemain potensial.

1. Setiap kartu pemain menampilkan Level *Hero* (Raja/Ratu) dan Bintang Perang secara visual.
2. Tombol aksi cepat "Undang" (*Invite*) yang terintegrasi dengan notifikasi sistem.

Desain Interaksi

Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal.

a. *Empty States* (Kondisi Kosong): Jika kombinasi filter pengguna terlalu spesifik (misal: "Klan Level 20 di Papua"), sistem menampilkan ilustrasi ramah dengan tombol "*Reset* Filter" untuk mencegah kebingungan pengguna (*User Frustration*).
b. *Infinite Scroll*: Daftar klan dimuat secara bertahap saat pengguna menggulir ke bawah, menjaga performa aplikasi tetap ringan di ponsel.

Visualisasi desain interaksi *Clan Hub* dapat dilihat pada Gambar 3.13 berikut:

Gambar 3.13 Rancangan Antarmuka Pusat Komunitas (*Team Hub*)
Sumber: Hasil Olahan Peneliti (2025)

3.6.3 *Dashboard* Manajemen Klan (*Clan Management Suite*)

Halaman ini dirancang sebagai pusat komando operasional bagi Pemimpin Klan (*Leader & Co-Leader*). Tujuan desain utama adalah menggantikan alat bantu manual (*spreadsheet*) dengan dasbor interaktif yang menyajikan wawasan data (*data insights*) secara visual dan *real-time*.

Arsitektur Navigasi Tab

Untuk mengakomodasi kompleksitas data tanpa mengorbankan keterbacaan di layar tablet/ponsel, antarmuka dibagi menjadi tiga modul tab utama:

a. Ringkasan Eksekutif (*Executive Summary*)

Menyajikan metrik kesehatan klan dalam sekilas pandang:

1. Kartu Performa: Menampilkan rata-rata kehancuran perang, total donasi mingguan, dan tren aktivitas anggota (Grafik Garis).
2. *Top* Kontributor: Sorotan otomatis untuk anggota dengan donasi tertinggi atau bintang perang terbanyak ("*MVP* Minggu Ini").

b. Manajemen Personel (*Roster Management*)

Tabel interaktif yang menampilkan daftar seluruh anggota dengan fitur sortir canggih:

1. Kolom data mencakup: Nama, Peran, *TH*, Donasi, dan Skor Reputasi.
2. Fitur Audit: Pemimpin dapat melihat riwayat promosi/demosi jabatan untuk mendeteksi pola kepemimpinan yang tidak stabil.
3. Aksi Cepat: Tombol *Kick*, *Promote*, atau *Review* dapat diakses langsung dari baris tabel.

c. Pusat Komando Perang (War Room & AI)

Fitur unggulan yang mengintegrasikan data perang dengan kecerdasan buatan:

1. Status *Live War*: Menampilkan skor sementara dan sisa serangan musuh secara *real-time* dari *API*.
2. Asisten Strategi (*Generative AI*): Panel obrolan khusus di mana pemimpin dapat menekan tombol "Analisis Cepat". Sistem akan mengirimkan statistik perang saat ini ke Google Gemini, lalu menampilkan saran taktis tekstual (misal: "Klan musuh lemah di serangan udara, sarankan anggota menggunakan strategi *Electro Dragon*").

Desain Interaksi

a. *Floating Action Button* (*FAB*): Tombol "Sinkronisasi Data" diletakkan melayang di pojok kanan bawah agar mudah dijangkau jempol (*Thumb Zone*), memungkinkan pembaruan data instan sebelum mengambil keputusan.

b. *Micro-Interactions*: Notifikasi *Toast* muncul saat pemimpin berhasil mengubah pengaturan klan atau menyalin tagar anggota, memberikan umpan balik sistem yang responsif.

Rancangan desain interaksi Manajemen Klan dapat dilihat pada Gambar 3.14 berikut:

Gambar 3.14 Rancangan Antarmuka *Dashboard* Manajemen Klan

Sumber: Hasil Olahan Peneliti (2025)

3.6.4 Detail Turnamen dan Bagan Pertandingan (*Tournament Bracket*)

Halaman ini merupakan antarmuka publik yang menampilkan jalannya kompetisi. Dalam tahap Pemodelan Desain Cepat, tantangan utama yang dihadapi adalah memvisualisasikan struktur bagan pertandingan (*bracket*) yang lebar dan kompleks agar tetap terbaca (*readable*) pada layar perangkat seluler yang sempit.

Komponen Antarmuka Utama

Halaman ini dibagi menjadi dua segmen visual untuk memisahkan informasi administratif dan operasional pertandingan:

a. *Header* Informasi Turnamen

Menampilkan metadata kompetisi secara ringkas:

1. Status Turnamen: Label dinamis (misal: "Pendaftaran", "Berjalan", "Selesai") yang sinkron dengan basis data.
2. Syarat Kualifikasi: Menampilkan ikon *Town Hall* minimum (misal: TH12+) sebagai pengingat aturan *Anti-Smurfing*.
3. Penghitung Mundur (*Countdown*): Fitur *real-time* yang menghitung sisa waktu menuju dimulainya ronde berikutnya, bertujuan meminimalisir kejadian *Walk-Over* (*WO*) akibat keterlambatan peserta.

b. Area Bagan Interaktif (*Interactive Bracket*)

Komponen visual utama yang menampilkan pohon pertandingan sistem gugur. Fitur interaktif yang disematkan meliputi:

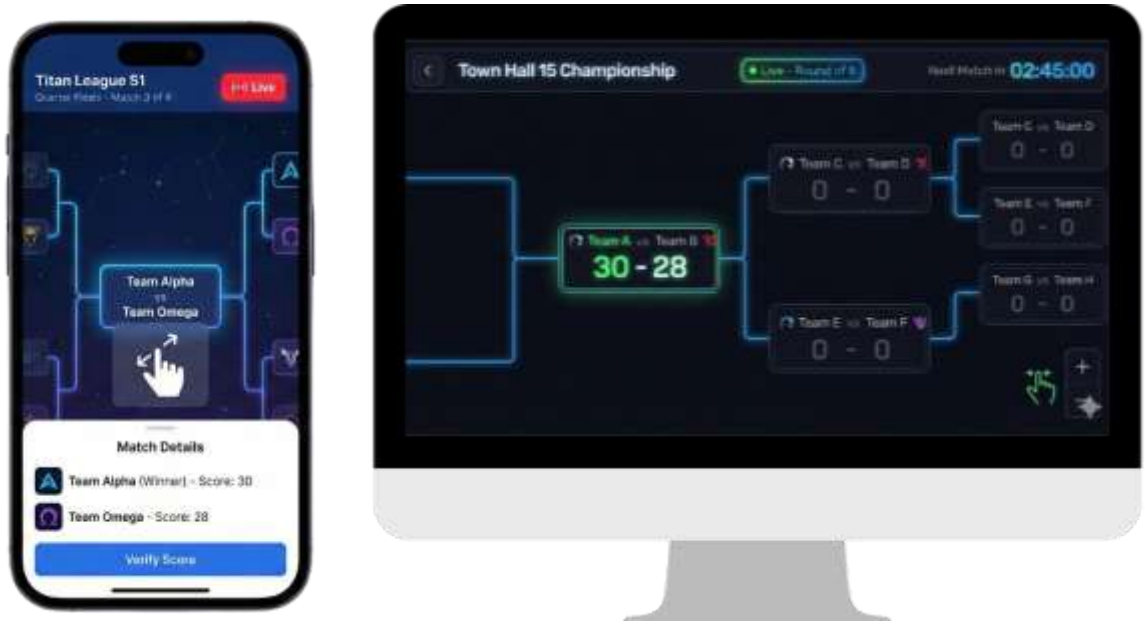
1. Kontrol Gestur (*Pan & Zoom*): Pengguna dapat mencubit layar (*pinch*) untuk memperbesar tampilan bagan atau menggesernya ke segala arah. Ini solusi teknis krusial untuk mengatasi keterbatasan *viewport* ponsel.
2. Indikator Status Visual: Garis penghubung antar tim menggunakan kode warna otomatis: Abu-abu (Belum main), Hijau (Pemenang melaju), dan Merah (Tereliminasi).
3. Kartu Pertandingan (*Match Node*): Kotak interaktif yang berisi Nama Tim, Skor Sementara, dan tombol aksi ("Tonton" atau "Lapor Skor").

Desain Interaksi Pelaporan Skor (*Self-Service*)

Untuk menjawab masalah inefisiensi panitia dalam mencatat skor manual, sistem menyediakan antarmuka pelaporan mandiri:

a. Formulir Lapor Skor: Kapten tim yang menang dapat menginput skor akhir secara langsung.
b. Unggah Bukti Valid: Wajib mengunggah tangkapan layar (*screenshot*) hasil perang sebagai bukti digital (*tamper-proof evidence*).
c. Mekanisme Konfirmasi: Jika kedua tim melaporkan skor yang sama, sistem otomatis memperbarui bagan ke babak selanjutnya. Jika terjadi konflik data, tombol "Panggil *Admin*" akan aktif untuk mediasi manual.

Rancangan desain interaksi Bagan Pertandingan dapat dilihat pada Gambar 3.15 berikut:



Gambar 3.15 Rancangan Antarmuka Bagan Turnamen Interaktif

Sumber: Hasil Olahan Peneliti (2025)

3.6.5 Pusat Pengetahuan (*Knowledge Hub*)

Halaman ini (app/knowledge-hub) berfungsi sebagai perpustakaan digital berbasis komunitas yang dirancang untuk mengatasi kesenjangan informasi strategi tingkat lanjut (*Meta Game*). Antarmuka mengadopsi pola desain *Content Feed* yang memprioritaskan visualisasi media video dan kemudahan kategorisasi.

Komponen Antarmuka Utama

Desain antarmuka dibagi menjadi dua mode interaksi utama, yaitu mode eksplorasi (Membaca) dan mode kontribusi (Menulis):

a. Umpan Strategi (*StrategyFeed*) berfungdi menampilkan daftar konten dalam tata letak *Grid* Responsif (1 kolom di seluler, 3 kolom di *desktop*). Setiap kartu konten (*StrategyCard*) menampilkan:
   1. *Thumbnail* Media: Gambar sampul video YouTube yang diambil otomatis melalui *API*.
   2. Metadata Taktis: Label kategori (misal: "Serangan Udara") dan *Tag* spesifik (misal: "TH15", "*Queen Walk*") untuk memudahkan pemindaian cepat oleh mata pengguna.
   3. Metrik Sosial: Indikator jumlah *Likes* dan *Replies* sebagai validasi kualitas konten.

b. Formulir Kontribusi Cerdas (*PostForm*) merupakan antarmuka *input* data yang dinamis. Jika pengguna memilih kategori "*Base Building*", formulir akan menyesuaikan diri menampilkan *input field* khusus untuk tautan penyalinan tata letak (*Layout Link*). Komponen ini dilengkapi dengan:
    1. Pratinjau Otomatis: Saat pengguna menempelkan tautan YouTube, sistem secara instan menampilkan *preview* video di dalam formulir sebelum diterbitkan (Vercel, 2024).
    2. Validasi *Input*: Indikator visual *real-time* (warna merah/hijau) jika data yang dimasukkan tidak memenuhi syarat (misal: Judul terlalu pendek).
c. Tampilan Detail & Diskusi (*FullPostDisplay*) merupakan halaman detail yang memuat konten lengkap. Fitur utamanya adalah:
    1. Pemutar Video Tersemat (*Embedded Player*): Memungkinkan pengguna menonton tutorial strategi tanpa meninggalkan aplikasi.
    2. Salin Pasukan (*Copy Army Link*): Tombol aksi cepat untuk menyalin komposisi pasukan langsung ke dalam gim *Clash of Clans*.
    3. Utas Komentar (*ReplySection*): Area diskusi berjenjang untuk tanya-jawab antar anggota komunitas.

Desain Interaksi

a. *Floating Action Button* (*FAB*): Pada tampilan *mobile*, tombol "Buat Postingan" didesain melayang di pojok kanan bawah untuk aksesibilitas ibu jari (*thumb-friendly zone*).
b. *Infinite Scroll*: Daftar konten memuat data tambahan secara otomatis saat pengguna menggulir ke bawah, menciptakan pengalaman eksplorasi yang tanpa henti (*seamless*).

Rancangan antarmuka Pusat Pengetahuan dapat dilihat pada Gambar 3.16 berikut:

Gambar 3.16 Rancangan Antarmuka *Knowledge Hub*

Sumber: Hasil Olahan Peneliti (2025)

3.7 Rencana Pengujian Sistem

Tahap pengujian merupakan fase validasi kritis dalam siklus *Deployment Delivery & Feedback* pada metode *Prototype*. Tujuan utamanya adalah memastikan integritas fungsional sistem sebelum diserahkan kepada komunitas pengguna.

3.7.1 Pengujian Fungsional (*Black Box Testing*)

Metode *Black Box Testing* dipilih untuk memvalidasi kesesuaian antara *input* dan *output* sistem tanpa membedah logika kode internal (*internal path*). Pengujian ini berfokus pada persyaratan fungsional. Rencana skenario pengujian dirancang untuk mencakup seluruh modul utama, sebagaimana dijabarkan pada Tabel 3.13 berikut:

Tabel 3.13 Rencana Pengujian Fungsional (*Black Box*)

Kode Uji	Modul / Fitur	Skenario Pengujian (*Input*)	Hasil yang Diharapkan (*Expected Output*)
TC-01	Verifikasi Akun	Pengguna memasukkan Tagar Pemain (*#TAG*) dan Token *API* yang valid pada menu profil.	Sistem berhasil melakukan *handshake* dengan *API* Supercell, status akun berubah menjadi *Verified*, dan data *TH* terkunci (*Immutable*).
TC-02	Sinkronisasi Data	Pemimpin klan menekan tombol "Sinkronisasi" pada dasbor klan.	Sistem memperbarui statistik (donasi, trofi) secara *real-time* dan memperbarui penanda waktu (*Timestamp*) pembaruan terakhir.
TC-03	Asisten AI	Pemimpin klan menekan tombol "Analisis Strategi" pada data perang.	Sistem mengirimkan *prompt* ke *Generative AI* dan menampilkan balasan berupa narasi taktis yang relevan (bukan sekadar angka).
TC-04	Validasi Turnamen	Pemain dengan akun TH11 mencoba mendaftar turnamen bersyarat "Minimal TH13".	Sistem secara otomatis menolak pendaftaran (Validasi *Anti-Smurfing*) dan menampilkan pesan: "Level akun tidak memenuhi syarat".
TC-05	Otomasi Bagan	Penyelenggara menjalankan fitur *Bracket Generator* setelah slot peserta penuh.	Sistem secara otomatis menyusun struktur bagan pertandingan sistem gugur (*Single Elimination*) yang seimbang.
TC-06	Pencarian Tim	Pengguna mengaktifkan Filter "Visi Kompetitif" dan *Sorting* "Reputasi Tertinggi".	Sistem menampilkan daftar klan yang terurut berdasarkan jumlah bintang reputasi tertinggi, mengabaikan klan yang tidak relevan.
TC-07	Pusat Pengetahuan	Pengguna menyematkan tautan video YouTube pada postingan strategi.	Sistem otomatis mengambil metadata (Judul & *Thumbnail*) via *API* dan merender pemutar video (*Embed Player*) pada konten.
TC-08	Sistem Reputasi	Pengguna memberikan *rating* bintang 5 dan komentar positif pada profil rekan tim.	Skor Reputasi pada profil target bertambah (dikalkulasi ulang), dan ulasan muncul di riwayat publik.

Sumber: Hasil Olahan Peneliti (2025)

Setiap skenario uji akan diberi status "Valid" jika hasil aktual sistem sama persis dengan hasil yang diharapkan, atau "*Invalid*" jika terjadi penyimpangan (*bug*). Jika ditemukan status *Invalid*, siklus pengembangan akan kembali ke tahap Konstruksi untuk perbaikan kode.

3.7.2 Pengujian Usabilitas (*System Usability Scale*)

Selain aspek fungsional, keberhasilan sistem juga ditentukan oleh tingkat penerimaan pengguna (*User Acceptance*). Untuk mengukur aspek kegunaan (*usability*) secara kuantitatif, penulis menggunakan metode standar industri yaitu *System Usability Scale* (*SUS*). Metode ini

dipilih karena validitasnya yang tinggi dalam mengukur persepsi kemudahan penggunaan pada aplikasi berbasis *web* dengan jumlah sampel kecil.

Partisipan dan Prosedur Pengujian

Pengujian usabilitas dilakukan dengan melibatkan 15 responden yang telah memenuhi kriteria inklusi (Pakar Komunitas & Pemain Veteran). Prosedur pengujian dilaksanakan dengan langkah-langkah berikut:

a. Sesi *Briefing*: Penulis menjelaskan tujuan aplikasi tanpa memberikan panduan langkah demi langkah, untuk melihat intuisi alami pengguna.
b. Simulasi Tugas (*Task Completion*): Responden diminta menyelesaikan 5 skenario utama secara mandiri di perangkat masing-masing:
    1. Mendaftar dan memverifikasi akun CoC.
    2. Melakukan pencarian klan dengan filter "Visi Kompetitif" dan pengurutan "Reputasi Tertinggi".
    3. Menggunakan fitur "Analisis Cepat" AI pada dasbor manajemen.
    4. Mendaftar ke turnamen yang aktif.
    5. Memberikan ulasan bintang kepada pemain lain.
c. Pengisian Kuesioner: Segera setelah simulasi selesai, responden mengisi 10 item pernyataan *SUS*.

Instrumen Kuesioner *SUS*

Instrumen terdiri dari 10 pernyataan dengan skala Likert 1-5 (Sangat Tidak Setuju s.d. Sangat Setuju). Pernyataan dirancang selang-seling (positif-negatif) untuk mengurangi bias seperti pada Tabel 3.14 berikut:

Tabel 3.14 Instrumen Pernyataan *SUS*

No	Kode	Pernyataan Instrumen
1	Q1 (+)	Saya pikir saya akan sering menggunakan sistem ini.
2	Q2 (-)	Saya merasa sistem ini terlalu rumit padahal bisa dibuat lebih sederhana.
3	Q3 (+)	Saya rasa sistem ini mudah digunakan.
4	Q4 (-)	Saya pikir saya membutuhkan bantuan teknis untuk menggunakan sistem ini.
5	Q5 (+)	Saya menemukan berbagai fungsi dalam sistem ini terintegrasi dengan baik.
6	Q6 (-)	Saya rasa banyak hal yang tidak konsisten dalam sistem ini.
7	Q7 (+)	Saya rasa kebanyakan orang akan belajar menggunakan sistem ini dengan sangat cepat.
8	Q8 (-)	Saya menemukan sistem ini sangat membingungkan untuk digunakan.
9	Q9 (+)	Saya merasa sangat percaya diri menggunakan sistem ini.
10	Q10 (-)	Saya harus belajar banyak hal sebelum saya bisa menggunakan sistem ini.

Sumber: Hasil Olahan Peneliti (2025)

Mekanisme Penilaian dan Target Keberhasilan

Skor akhir *SUS* dihitung dengan rumus khusus yang menghasilkan nilai antara 0 hingga 100. Nilai ini kemudian dikategorikan untuk menentukan kelayakan sistem:

a. Skor di atas 80.3: Sangat Baik (*Excellent*).
b. Skor 68 – 80.3: Baik (*Good*) – Dapat diterima.
c. Skor di bawah 68: Buruk (*Below Average*) – Perlu perbaikan desain.

Penelitian ini menargetkan skor minimal 68 (Kategori Baik). Jika skor ini tercapai, artinya desain *Mobile-First* yang diterapkan sudah berhasil membuat aplikasi nyaman digunakan di ponsel (Bangor et al., 2009).

3.7.3 Validasi Akseptansi Pengguna (*User Acceptance Testing*)

Tahap akhir dari siklus pengujian adalah *User Acceptance Testing* (*UAT*). Berbeda dengan pengujian fungsional yang mencari *bug*, *UAT* bertujuan untuk memvalidasi apakah solusi yang ditawarkan sistem benar-benar menjawab permasalahan bisnis pengguna di dunia nyata.

Validasi kualitatif dilakukan melalui wawancara mendalam (*In-Depth Interview*) dengan 5 Pakar Komunitas (Validator Ahli). Partisipan ini dipilih karena memiliki kapasitas manajerial (*Leader* Senior) untuk menilai dampak sistem terhadap efisiensi pengelolaan klan.

Indikator Keberhasilan (*Acceptance Criteria*)

Keberhasilan implementasi sistem dinilai berdasarkan komparasi kondisi *Before-After* pada tiga indikator kunci:

a. Efisiensi Waktu Administrasi
    1. Masalah Lama: Pemimpin klan menghabiskan rata-rata 1-2 jam per minggu untuk rekapitulasi data perang manual di *spreadsheet*.
    2. Target *UAT*: Fitur "Sinkronisasi Data" dan Dasbor Otomatis mampu memangkas waktu administrasi hingga di bawah 10 menit.
b. Akurasi Keputusan Strategis
    1. Masalah Lama: Keputusan strategi perang sering kali bersifat subjektif (*feeling*) atau spekulatif tanpa dasar data yang kuat.
    2. Target *UAT*: Saran taktis dari "Asisten Strategi AI" dinilai logis, relevan, dan membantu pemimpin mengambil keputusan objektif dalam situasi kritis (misal: penentuan target serangan).
c. Kepercayaan dalam Rekrutmen (*Trust*)
    1. Masalah Lama: Tingginya risiko merekrut "Kutu Loncat" karena ketiadaan rekam jejak yang transparan.
    2. Target *UAT*: Fitur "Skor Reputasi" dan "Riwayat Tim" memberikan rasa aman (*Psychological Safety*) bagi pemimpin klan sebelum menerima anggota baru.

Mekanisme Keputusan

Hasil wawancara akan dianalisis untuk menentukan status akhir sistem:

a. DITERIMA (*Accepted*): Jika mayoritas pakar menyatakan sistem meningkatkan efisiensi secara signifikan.
b. DITOLAK (Rejected): Jika sistem dinilai tidak memberikan nilai tambah atau justru menambah beban kerja baru.

BAB IV
IMPLEMENTASI DAN PEMBAHASAN

4.1 Lingkungan Implementasi dan Arsitektur Sistem

Bab ini menjabarkan realisasi teknis dari rancangan sistem yang telah didefinisikan pada Bab 3 (Tahap Konstruksi Prototipe). Pembahasan mencakup spesifikasi lingkungan pengembangan, implementasi kode program untuk modul-modul krusial, serta hasil pengujian sistem. Implementasi ini difokuskan untuk memenuhi kebutuhan operasional komunitas *Clash of Clans* yang menuntut kecepatan akses dan akurasi data *real-time*.

4.1.1 Lingkungan Pengembangan dan Operasional

Untuk menjamin kelancaran proses transformasi desain menjadi kode program (*coding*) hingga siap digunakan (*deployment*), spesifikasi lingkungan kerja dibagi menjadi dua kategori utama.

Spesifikasi Perangkat Keras (*Hardware*)

a. Lingkungan Lokal (*Development Environment*)

Perangkat ini digunakan oleh peneliti selama tahap konstruksi untuk penulisan kode, *debugging*, dan simulasi server. Spesifikasi tinggi dipilih untuk menangani proses kompilasi Next.js yang intensif sumber daya:

1. Prosesor: AMD Ryzen 7 (8 Inti) – Mendukung *multitasking* saat menjalankan *local server*, basis data emulator, dan *browser* secara simultan.
2. Memori (RAM): 16 GB – Mencegah *bottleneck* saat menjalankan simulasi antarmuka pada berbagai ukuran layar emulator.
3. Penyimpanan: SSD 512 GB (NVMe) – Menjamin kecepatan baca-tulis aset proyek (gambar/kode) secara instan.
4. Grafis (GPU): NVIDIA GeForce RTX 3050 – Digunakan untuk akselerasi perangkat keras saat me-*render* animasi antarmuka (Framer Motion).

b. Lingkungan Operasional (*Production Environment*)

Sistem "ClashHub" dipublikasikan (*deployed*) menggunakan infrastruktur awan (*Cloud Infrastructure*) untuk menjamin ketersediaan akses 24/7 tanpa mengelola server fisik.

1. Penyedia Layanan: Vercel (*Platform-as-a-Service*).

2. Lokasi Server: Singapura (sin1) – Dipilih strategis untuk meminimalisir latensi (< 50ms) bagi pengguna di Indonesia.
3. Skalabilitas: Kapasitas memori fungsi server (*Serverless Functions*) dikonfigurasi untuk menyesuaikan beban otomatis (hingga 1024 MB) saat terjadi lonjakan trafik pendaftaran turnamen.

Spesifikasi Perangkat Lunak (*Software*)

Perangkat lunak dan pustaka (*library*) yang digunakan dipilih berdasarkan kompatibilitasnya dengan arsitektur *Mobile-First*. Rincian perangkat lunak disajikan pada Tabel 4.1 berikut:

Tabel 4.1 Daftar Perangkat Lunak dan Pustaka Pendukung

No.	Nama Perangkat Lunak	Versi	Fungsi Implementasi dalam Sistem
1	*Node*.js	v18.17	Lingkungan eksekusi (*Runtime*) JavaScript di sisi server dan lokal.
2	Next.js	v14.2	Kerangka kerja utama dengan arsitektur App Router untuk performa *web* dan SEO.
3	TypeScript	v5.0	Bahasa pemrograman dengan pengetikan statis untuk validasi tipe data *API* yang ketat.
4	Firebase *SDK*	v10.12	Pustaka klien untuk koneksi ke layanan Autentikasi dan Firestore (NoSQL).
5	Firebase *Admin*	v12.1	*SDK* sisi server untuk manajemen hak akses (*Role Admin*) yang aman.
6	Tailwind *CSS*	v3.4	Kerangka kerja utilitas untuk membangun antarmuka responsif (*Mobile-First*).
7	Google *Generative AI*	v0.12	*SDK* resmi untuk mengintegrasikan model Gemini pada fitur "Asisten Strategi".
8	Framer Motion	v11.2	Pustaka animasi untuk transisi halaman yang halus (*Smooth Navigation*).
9	Visual Studio Code	1.89	*Integrated Development Environment* (*IDE*) utama dengan ekstensi pendukung.
10	Google Chrome	Terbaru	Peramban utama untuk pengujian *DevTools* dan simulasi responsivitas seluler.

Sumber: Hasil Olahan Peneliti (2025)

Sistem mengintegrasikan tiga layanan eksternal utama untuk mendukung fungsionalitas cerdas:

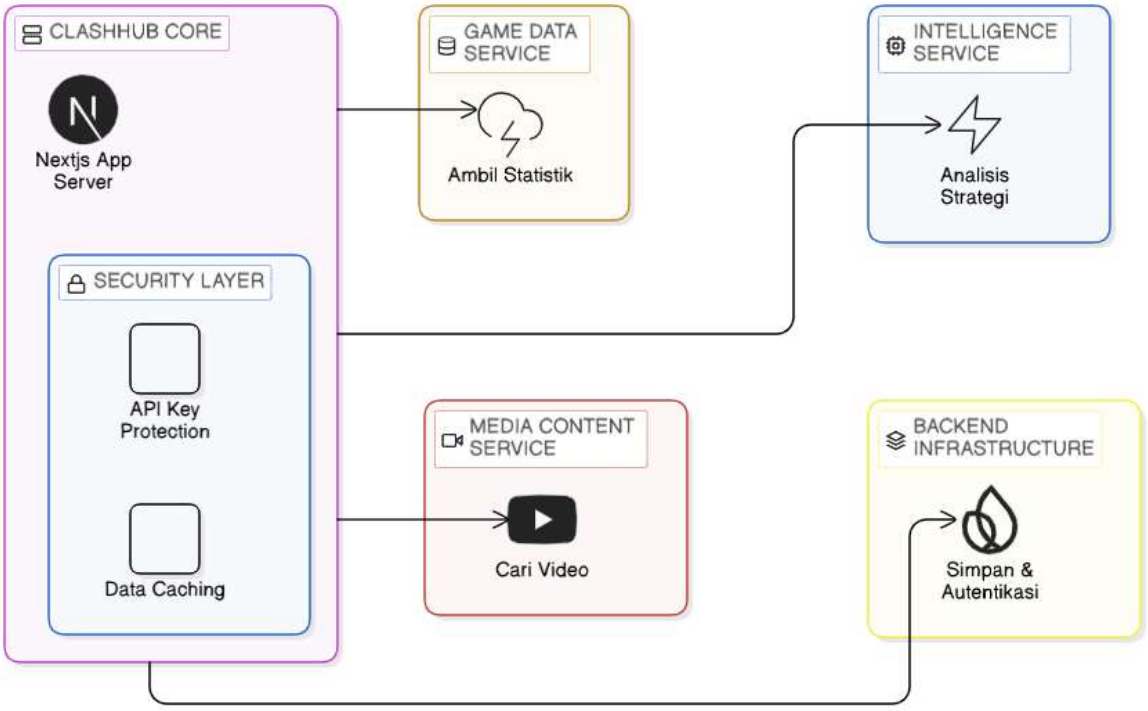
a. *Clash of Clans API*: Sumber tunggal kebenaran (*Single Source of Truth*) untuk agregasi data statistik pemain, klan, dan log perang secara *real-time*.
b. Google Gemini AI: Mesin kecerdasan buatan generatif yang bertugas memproses data *JSON* statistik perang menjadi narasi saran taktis yang mudah dipahami manusia.
c. YouTube Data *API*: Layanan agregasi konten untuk mengambil metadata video strategi terbaru secara otomatis pada modul "Pusat Pengetahuan".

4.1.2 Arsitektur Implementasi Sistem

Implementasi sistem "ClashHub" menerapkan pola arsitektur terpusat (*Centralized Architecture*) di mana server aplikasi Next.js bertindak sebagai orkestrator utama. Berbeda dengan aplikasi *web* tradisional yang mengakses layanan pihak ketiga langsung dari peramban (*client-side*), sistem ini menempatkan seluruh logika integrasi di sisi server (*server-side*). Pendekatan ini dipilih untuk menjawab tiga kebutuhan teknis krusial:

a. Keamanan Data (*Security*): Kunci akses sensitif (*API Secret Keys*) untuk layanan Supercell dan Google Gemini disimpan secara terenkripsi di lingkungan server (*Environment Variables*), sehingga mustahil dicuri oleh pengguna melalui inspeksi elemen peramban.
b. Efisiensi Kinerja (*Performance*): Server berfungsi sebagai *Aggregator* yang mengumpulkan dan mengolah data mentah dari berbagai sumber (*API* Gim, *Database* Reputasi) sebelum dikirim ke pengguna. Mekanisme ini memungkinkan fitur *Sorting* & *Filtering* data klan berjalan instan karena beban komputasi ditangani oleh server, bukan oleh ponsel pengguna.
c. Manajemen *Rate Limit*: Server menerapkan mekanisme antrean dan *caching* terpusat untuk mencegah pemblokiran akses akibat permintaan data yang berlebihan ke *API Clash of Clans*.

Visualisasi topologi arsitektur sistem yang telah dibangun disajikan pada Gambar 4.1 berikut:



Gambar 4.1 Arsitektur Implementasi Sistem ClashHub

Sumber: Hasil Olahan Peneliti (2025)

Keterangan Alur:

a. Pengguna (*Client*): Mengirim permintaan via antarmuka Next.js (misal: "Cari Klan").
b. Server (*Middleware*): Menerima permintaan, mengecek *cache*, dan memutuskan apakah perlu memanggil *API* eksternal.
c. Layanan Eksternal: Menyediakan data mentah (Statistik Gim, Analisis AI, Video) yang kemudian diolah kembali oleh server menjadi format *JSON* yang siap tampil.

Secara rinci, implementasi teknis sistem ini dikelompokkan menjadi enam modul fungsional berikut:

Implementasi Logika Verifikasi Akun

Bagian ini menjelaskan bagaimana sistem memastikan bahwa pengguna yang mendaftar benar-benar pemilik asli dari akun *Clash of Clans* tersebut. Fitur ini sangat penting untuk mencegah kecurangan (misalnya: orang lain mendaftarkan akun milik *top player* sembarangan). Proses verifikasi dilakukan di sisi server (*Backend*) agar aman dan tidak bisa dimanipulasi. Sistem menggunakan Token *API* (kode rahasia yang ada di dalam gim) sebagai

kunci pencocokan. Gambar 4.2 berikut adalah implementasi kode program yang menangani proses verifikasi:

```
// Lokasi: app/api/coc/verify-player/route.ts
// Fungsi: Memverifikasi kepemilikan akun menggunakan Token API

export async function POST(request: NextRequest) {
  try {
    // 1. Cek apakah pengguna sudah login
    const authUser = await getSessionUser();
    if (!authUser) {
      return NextResponse.json({ message: 'Harap login terlebih dahulu' },
{ status: 401 });
    }

    // 2. Ambil data tagar pemain dan token yang dikirim dari formulir
    const payload = await request.json();

    // 3. Validasi ke Server Resmi Supercell
    // Fungsi ini mengirim data ke API CoC untuk diperiksa kecocokannya
    const isTokenValid = await cocApi.verifyPlayerToken(payload.playerTag,
payload.apiToken);

    if (!isTokenValid) {
       throw new Error('Token API salah. Verifikasi gagal.');
    }

    // 4. Jika valid, simpan status "Terverifikasi" ke database
    await adminFirestore.collection('users').doc(authUser.uid).update({
        isVerified: true,
        playerTag: payload.playerTag,
        lastVerified: new Date()
    });

    return NextResponse.json({ success: true, message: 'Akun berhasil
diverifikasi!' });

  } catch (error) {
    return NextResponse.json({ message: 'Terjadi kesalahan sistem' }, {
status: 500 });
  }
}
```

Gambar 4.2 Kode Program Verifikasi Akun Sisi Server

Kode pada Gambar 4.2 bekerja dengan menerima Tagar Pemain dan Token dari pengguna, lalu menanyakannya langsung ke server Supercell. Jika Supercell menjawab "Valid", barulah sistem ClashHub menyimpan status tersebut ke *database*.

Implementasi Integrasi Data *Clash of Clans*

Bagian ini menjelaskan bagaimana sistem mengambil data statistik (seperti riwayat perang klan) agar bisa ditampilkan di aplikasi. Tantangan utamanya adalah adanya batasan jumlah permintaan (*rate limit*) dari *API* resmi. Jika aplikasi mengambil data terlalu sering, akses bisa diblokir sementara. Untuk mengatasinya, penulis menerapkan logika "Sinkronisasi Hemat". Artinya, sistem akan mengecek dulu apakah data yang tersimpan di *database* sudah usang atau masih baru. Jika data masih baru (kurang dari 5 menit), sistem tidak akan meminta data lagi ke server gim. Gambar 4.3 berikut adalah implementasi logika tersebut dalam kode program:

```
// Lokasi: app/api/clan/sync/war/route.ts
// Fungsi: Mengambil data perang dengan mekanisme pengecekan waktu
(Caching)

export async function POST(req: Request) {
    // 1. Ambil data terakhir yang tersimpan di database lokal
    const cacheDoc = await clanApiCacheRef.get();
    const lastUpdate = cacheDoc.data()?.lastUpdatedWar;

    // Hitung selisih waktu (dalam menit) antara sekarang dan update
terakhir
    const minutesSinceUpdate = (Date.now() - lastUpdate.toMillis()) /
60000;

    // 2. Logika Keputusan:
    // Jika data baru diperbarui kurang dari 5 menit yang lalu, pakai data
lama saja.
    if (minutesSinceUpdate < 5) {
        return NextResponse.json({
            message: 'Data masih baru, menggunakan simpanan lokal.',
            data: cacheDoc.data().currentWar
        });
    }

    // 3. Jika data sudah usang (> 5 menit), ambil data baru dari API
Supercell
    const warData = await cocApi.getClanCurrentWar(clanTag);

    // 4. Simpan data baru tersebut ke database untuk penggunaan berikutnya
    await clanApiCacheRef.set({
        currentWar: warData,
        lastUpdatedWar: FieldValue.serverTimestamp(),
    }, { merge: true });

    return NextResponse.json({ message: 'Data berhasil diperbarui dari
server gim.', data: warData });
}
```

Gambar 4.3 Kode Program Logika Sinkronisasi Data (*Caching*)

Logika pada Gambar 4.3, khususnya baris *if* (minutesSinceUpdate < 5), adalah kunci efisiensi sistem ini. Dengan cara ini, aplikasi tetap responsif dan mematuhi aturan penggunaan *API* dari pengembang gim.

Implementasi Modul Kecerdasan Buatan (Asisten Strategi)

Fitur ini merupakan inovasi utama sistem ClashHub yang membedakannya dengan aplikasi sejenis. Implementasinya berfokus pada penggunaan teknologi Google Gemini AI sebagai "Asisten Klan". Tantangan terbesar dalam penggunaan AI adalah mencegahnya memberikan jawaban yang tidak akurat atau mengarang (halusinasi). Untuk mengatasi hal tersebut, sistem menerapkan teknik Pemberian Konteks. Artinya, sebelum pengguna bertanya, sistem sudah membekali AI dengan data fakta tentang kondisi klan saat itu. Gambar 4.4 berikut adalah implementasi kode program untuk menyusun konteks tersebut:

```
// 1. Mengambil semua data klan dari database secara bersamaan agar cepat
const [clanData, membersData, warData] = await Promise.all([
  getManagedClanDataAdmin(clanId),      // Data Profil Klan
  getTeamMembersAdmin(clanId),          // Data Anggota
  getWarArchivesByClanId(clanId),       // Riwayat 50 Perang Terakhir
]);

// 2. Menyusun Instruksi Khusus (System Prompt)
// Data yang sudah diambil diringkas dan dimasukkan ke dalam pesan
instruksi.
// Instruksi ini "memaksa" AI untuk hanya menjawab berdasarkan data yang
diberikan.
const clanContext = `
  PERAN ANDA:
  Anda adalah "ASISTEN KLAN" untuk klan bernama ${clanData?.name}.

  ATURAN UTAMA:
  1. Jawab pertanyaan pengguna HANYA berdasarkan data di bawah ini.
  2. JANGAN MENGARANG DATA. Jika data tidak ada di bawah, katakan "Data
tidak tersedia".
  3. Berikan saran strategi yang sopan dan profesional.

  --- DATA FAKTA KLAN ---
  1. PROFIL: ${JSON.stringify(clanData)}
  2. DAFTAR ANGGOTA: ${JSON.stringify(membersData)}
  3. RIWAYAT PERANG: ${JSON.stringify(warData)}
  --- AKHIR DATA ---
`;

// 3. Memanggil Model Gemini dengan instruksi di atas
const model = genAI.getGenerativeModel({
  model: 'gemini-pro',
  systemInstruction: clanContext,
});
```

Gambar 4.4 Kode Program Integrasi AI dengan Konteks Data

Kode pada Gambar 4.4 menunjukkan bagaimana sistem "mengajari" AI tentang kondisi klan sebelum menjawab pertanyaan. Perintah tegas seperti "JANGAN MENGARANG DATA" sangat penting untuk menjaga kualitas jawaban agar tetap relevan dan dapat dipercaya oleh pemimpin klan.

Implementasi Modul Agregasi Konten (YouTube)

Modul ini berfungsi untuk mengisi fitur "Pusat Pengetahuan" dengan video strategi terbaru secara otomatis. Tujuannya adalah agar pengguna selalu mendapatkan informasi strategi terkini (*Meta Strategies*) tanpa perlu mencarinya satu per satu di YouTube. Sistem menggunakan mekanisme Penjadwalan Otomatis (*Cron Job*) yang berjalan setiap hari untuk mengecek apakah ada video baru dari saluran resmi kreator konten. Gambar 4.5 berikut adalah logika kode program untuk sinkronisasi video tersebut:

```
// Lokasi: app/api/youtube/sync/route.ts
// Fungsi: Mengambil video YouTube terbaru secara berkala

export async function GET(request: NextRequest) {
  // 1. Ambil video terbaru dari daftar saluran yang dipantau
  for (const channel of CHANNELS_TO_SYNC) {
    const latestVideos = await getLatestVideosFromPlaylist(channel.id);

    for (const video of latestVideos) {
      // 2. Siapkan data video dalam format yang rapi
      const videoData = {
        id: video.id,
        title: video.title,
        category: 'Berita Komunitas',
        url: `https://youtube.com/watch?v=${video.id}`
      };

      // 3. Simpan ke Database (Cegah Duplikasi)
      // Sistem mengecek dulu: "Apakah video ID ini sudah ada di database?"
      // Jika belum ada, baru disimpan.
      await saveVideoToFirestore(videoData);
    }
  }

  return NextResponse.json({ message: 'Sinkronisasi video selesai.' });
}
```

Gambar 4.5 Kode Program Sinkronisasi Video YouTube Otomatis

Logika pada Gambar 4.5 memastikan bahwa konten di aplikasi selalu segar (*up-to-date*). Pengecekan duplikasi di langkah ke-3 sangat penting untuk menghemat ruang penyimpanan *database*, sehingga video yang sama tidak akan disimpan dua kali.

Implementasi Keamanan *Backend*

Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal.

Bagian ini adalah tulang punggung yang menjaga keamanan data sistem. Implementasi di sisi server menggunakan Firebase *Admin SDK*, sebuah alat khusus yang memberikan hak akses penuh kepada sistem untuk mengelola data lintas pengguna tanpa dibatasi aturan keamanan biasa.

a. Inisialisasi Aman (Mencegah Koneksi Ganda)

   Sistem menerapkan teknik pencegahan koneksi ganda. Kode ini memastikan bahwa sistem tidak membuka koneksi baru ke *database* jika koneksi lama masih aktif, sehingga mencegah server menjadi lambat atau *error*. Gambar 4.6 berikut adalah kode implementasinya:

```
// Lokasi: lib/firebase-admin.ts
// Fungsi: Menghubungkan aplikasi ke database dengan aman

import * as admin from 'firebase-admin';

// Cek apakah aplikasi admin sudah ada?
// Jika belum ada (!apps.length), baru kita buat koneksi baru.
if (!admin.apps.length) {
  try {
    const serviceAccountKey = process.env.FIREBASE_SERVICE_ACCOUNT_KEY;

    // Gunakan kunci rahasia dari variabel lingkungan server (Environment
Variable)
    // agar tidak bisa dibaca oleh orang lain.
    admin.initializeApp({
        credential: admin.credential.cert(JSON.parse(serviceAccountKey)),
    });

    // Pengaturan agar sistem tidak error jika ada data kosong
    admin.firestore().settings({
      ignoreUndefinedProperties: true,
    });

  } catch (error) {
    console.error('Gagal menghubungkan Admin SDK:', error);
  }
}
```

```
// Ekspor fungsi database dan autentikasi untuk dipakai di file lain
const adminFirestore = admin.firestore();
const adminAuth = admin.auth();

export { adminFirestore, adminAuth };
```

Gambar 4.6 Kode Program Inisialisasi Firebase *Admin* yang Aman

b. Verifikasi Sesi Pengguna

Alih-alih mempercayai peramban (*browser*) begitu saja, sistem melakukan pengecekan ganda setiap kali pengguna mengakses halaman penting (seperti Dasbor *Leader*). Sistem membaca "kue kering" (*cookie*) sesi dan memvalidasinya langsung ke *database*. Gambar 4.7 berikut adalah kode implementasinya:

```
// Lokasi: lib/server-auth.ts
// Fungsi: Memastikan pengguna yang login adalah asli

export async function getSessionUser() {
  const cookieHeader = headers().get('cookie');

  // 1. Cari cookie bernama 'session-token' yang dikirim browser
  const sessionCookie = cookieHeader?.split(';').find(c =>
c.trim().startsWith('session-token='));

  if (sessionCookie) {
      // 2. Ambil ID Pengguna (UID) dari dalam token tersebut
      const uid = extractUidFromToken(sessionCookie);

      if (uid) {
          // 3. Cek ke Database: Apakah pengguna ini benar-benar ada?
          const userDoc = await
adminFirestore.collection('users').doc(uid).get();

          if (userDoc.exists) {
              return userDoc.data(); // Pengguna valid, izinkan akses
          }
      }
  }

  return null; // Sesi tidak valid, tolak akses
}
```

Gambar 4.7 Kode Program Verifikasi Sesi di Sisi Server

Manajemen Aset Gambar (Efisiensi Performa)

Berdasarkan analisis kebutuhan pengguna yang menginginkan aplikasi cepat dan hemat kuota, sistem menerapkan strategi Aset Statis Terkurasi. Artinya, pengguna tidak mengunggah foto profil sendiri (yang bisa berat dan memakan kuota server), melainkan memilih dari koleksi

avatar karakter *Clash of Clans* yang sudah disediakan. Strategi ini dipilih untuk menjaga estetika aplikasi agar tetap rapi (sesuai tema gim) dan memastikan pemuatan gambar instan karena aset sudah tersimpan di dalam aplikasi. Gambar 4.8 berikut adalah implementasi pemilihan avatar tersebut:

```
// Lokasi: app/profile/edit/EditProfileClient.tsx
// Fungsi: Menampilkan pilihan avatar siap pakai

const STATIC_AVATARS = [
  '/images/barbarian.png',
  '/images/archer.png',
  '/images/giant.png',
  '/images/pekka.png',
  '/images/hogrider.png',
];

// Tampilan Tombol Pilihan di Layar
<div className="flex flex-wrap gap-3 justify-center">
  {STATIC_AVATARS.map((url) => (
    <button
      key={url}
      type="button"
      onClick={() => setFormData({ avatarUrl: url })} // Simpan URL gambar
yang dipilih
      className="rounded-full hover:scale-110 transition"
    >
      <Image
         src={url}
         width={64}
         height={64}
         alt="Pilihan Avatar"
         className="border-2 border-yellow-500 rounded-full"
      />
    </button>
  ))}
</div>
```

Gambar 4.8 Kode Program Antarmuka Pemilihan Avatar Statis

Analisis Implementasi:

a. Efisiensi Biaya: Tidak perlu membayar biaya penyimpanan *cloud* (*Cloud Storage*) karena tidak ada fitur *upload* gambar.
b. Performa: Aplikasi menjadi jauh lebih cepat karena tidak perlu memproses dan memuat gambar berukuran besar yang diunggah pengguna.

4.1.3 Struktur Direktori Proyek (Realisasi Modular)

Implementasi kode program "ClashHub" disusun mengikuti standar struktur direktori dari kerangka kerja Next.js 14. Penulis menerapkan pendekatan Arsitektur Berbasis Fitur (*Feature-First Architecture*) dalam pengorganisasian berkas. Artinya, penulis tidak memisahkan berkas berdasarkan jenis teknisnya semata (seperti memisahkan folder khusus *Controller* dan *View* secara kaku), melainkan mengelompokkannya berdasarkan Modul Fitur. Contohnya, seluruh logika server dan tampilan antarmuka yang berkaitan dengan fitur "Manajemen Klan" disimpan dalam satu kelompok direktori yang berdekatan. Pendekatan ini dipilih untuk memudahkan proses perawatan kode (*maintenance*) dan perbaikan *bug* yang spesifik pada fitur tertentu tanpa mengganggu fitur lain. Visualisasi struktur direktori utama proyek disajikan pada Gambar 4.9 berikut:

```
/ (Folder Utama Proyek)
├── app/                          # Lapisan Aplikasi (Frontend & Backend)
│   ├── api/                      # Folder khusus logika Server (API Routes)
│   │   ├── clan/manage/          # Logika Manajemen Klan & Sinkronisasi
│   │   ├── coc/                  # Penghubung aman ke API Supercell
│   │   ├── tournaments/          # Logika Turnamen & Bagan
│   │   └── youtube/sync/         # Logika pengambilan video otomatis
│   ├── clan-hub/                 # Tampilan Halaman Pencarian Tim
│   ├── profile/                  # Tampilan Halaman Profil Pemain
│   └── tournament/               # Tampilan Halaman Turnamen
├── lib/                          # Folder Logika Bantuan (Shared Logic)
│   ├── firestore-admin/          # Fungsi koneksi database sisi server
(Admin)
│   ├── hooks/                    # Fungsi logika tampilan interaktif
│   └── types/                    # Definisi struktur data (TypeScript
Interfaces)
├── public/                       # Penyimpanan aset statis (Avatar, Ikon,
Logo)
└── scripts/                      # Skrip utilitas tambahan (Setup awal)
```

Gambar 4.9 Struktur Direktori Proyek ClashHub

Struktur di atas menunjukkan pembagian tugas yang jelas dalam sistem:

a. Folder app/*api* berfungsi sebagai *Backend* yang menangani urusan data dan logika server.
b. Folder lain di dalam app/ (seperti *clan*-hub, profile) berfungsi sebagai *Frontend* yang menangani tampilan antarmuka pengguna.
c. Folder lib/ berisi kode-kode bantuan yang bisa dipakai ulang di berbagai tempat (*reusable code*).

Struktur Rute *API* (*API Routes Structure*)

Direktori app/*api*/ berfungsi sebagai pusat kendali yang menghubungkan tampilan aplikasi dengan data di basis data atau layanan luar. Dalam perancangan ini, sistem menerapkan prinsip Fungsi Spesifik. Artinya, proses yang rumit dipecah menjadi beberapa alamat *API* (*endpoints*) yang lebih kecil dan khusus. Tujuannya adalah untuk mencegah server menjadi lambat (*bottleneck*) jika harus memproses terlalu banyak data sekaligus. Berdasarkan fungsinya, rute *API* dikelompokkan menjadi empat bagian utama:

a. Kelompok Sinkronisasi Data (Sinkronisasi Terpisah)

   Alih-alih membuat satu tombol "Perbarui Semua" yang bisa membuat aplikasi macet (*timeout*) karena data yang ditarik terlalu besar, sistem memecah proses sinkronisasi klan menjadi bagian-bagian kecil:

   1. .../*sync*/basic: Hanya memperbarui data ringan seperti Level Klan dan Poin Piala. Proses ini sangat cepat (< 1 detik).
   2. .../*sync*/war: Khusus memeriksa status perang klan saat ini (Sedang Perang / Persiapan).
   3. .../*sync*/members: Khusus memperbarui data statistik 50 anggota klan.
   4. .../*sync*/*warlog*: Menangani pengambilan riwayat perang jangka panjang (data paling berat), sehingga dipisahkan agar tidak mengganggu fungsi lain.

b. Kelompok Manajemen Turnamen

   Struktur *API* ini mencerminkan urutan jalannya sebuah kompetisi:

   1. .../register: Menangani pendaftaran tim dan memvalidasi apakah level akun memenuhi syarat (*Anti-Smurfing*).
   2. .../manage/generate-*bracket*: Jalur khusus yang memicu algoritma untuk menggambar bagan pertandingan secara otomatis.
   3. .../manage/match/[*matchId*]/report: Menangani pelaporan skor pertandingan dan mewajibkan unggahan bukti gambar.
   4. .../cron: Jalur otomatis yang dijalankan oleh server untuk menutup pendaftaran atau memulai turnamen sesuai jadwal.

c. Kelompok Integrasi Kecerdasan Buatan (AI)

   Direktori ini (.../ai-assistant/) berfungsi sebagai "ruang isolasi" untuk berinteraksi dengan Google Gemini. Fungsinya meliputi:

   1. Mengumpulkan data mentah dari basis data (Profil klan, Riwayat Perang).
   2. Mengubah data tersebut menjadi instruksi teks yang dimengerti oleh AI.

3. Menyimpan riwayat percakapan agar saran strategi tetap relevan dengan konteks sebelumnya.

d. Kelompok Layanan Perantara (Keamanan)

Berfungsi sebagai gerbang keamanan untuk melindungi data sensitif:

1. *api*/coc/verify-*player*: Menangani verifikasi kepemilikan akun pemain. Proses ini dilakukan di sisi server agar token *API* pemain aman dan tidak bisa dicuri orang lain.
2. *api*/youtube/*sync*: Menangani proses pengambilan video strategi terbaru dari YouTube secara berkala di latar belakang (*background process*).

Dengan memecah *API* menjadi bagian-bagian kecil seperti di atas, sistem menjadi lebih stabil. Contohnya, jika proses pengambilan data "Riwayat Perang" mengalami gangguan koneksi, fitur "Pendaftaran Turnamen" tidak akan ikut terganggu karena menggunakan jalur komunikasi yang berbeda.

Pengelompokan Fitur (*Feature-Based Structure*)

Pada bagian tampilan aplikasi (folder app/), struktur folder disusun berdasarkan fitur yang ada di aplikasi. Setiap folder fitur bersifat mandiri, artinya halaman (*Page*) dan komponen tampilan (*Components*) yang saling berkaitan disimpan dalam satu lokasi yang sama. Pendekatan ini sangat memudahkan pengembang: jika ingin mengubah fitur "Profil", cukup buka folder profile, tidak perlu mencari kode di tempat lain. Berikut adalah penjabaran lima modul fitur utama yang diimplementasikan:

a. Modul Manajemen Klan (app/*clan*/manage/)

Folder ini adalah pusat kendali bagi pemimpin klan. Semua komponen yang dibutuhkan untuk mengelola klan ada di sini:

1. .../*components*/*GeminiAssistantTab*.tsx: Komponen khusus untuk tampilan obrolan dengan AI.
2. .../*components*/*ActiveWarTabContent*.tsx: Komponen untuk memantau perang yang sedang berlangsung secara *real-time*.
3. .../*components*/MemberTable.tsx: Tabel daftar anggota yang memiliki tombol untuk mengubah jabatan.

b. Modul Pencarian Tim (app/*clan*-hub/)

Folder ini menangani fitur pertemuan antara pemain dan klan.

1. .../*components*/TeamHubFilterBar.tsx: Mengatur logika filter pencarian yang kompleks (misal: mencari klan level 10 di Indonesia).

2. .../*components*/*PlayersTab*.tsx: Bagian khusus untuk menampilkan daftar pemain "Agen Bebas" yang sedang mencari klan.

c. Modul Turnamen (app/tournament/)

Folder ini menangani seluruh proses kompetisi, mulai dari pembuatan hingga pertandingan selesai. Struktur foldernya mengikuti alur turnamen:

1. .../*create*/: Halaman formulir untuk membuat turnamen baru.
2. .../[*tournamentId*]/manage/: Halaman khusus panitia untuk memvalidasi skor.
3. .../[*tournamentId*]/*bracket*/: Halaman untuk menampilkan gambar bagan pertandingan (*Bracket*) kepada publik.

d. Modul Pusat Pengetahuan (app/knowledge-hub/)Folder ini menangani fitur berbagi strategi.

1. .../*create*/: Formulir untuk mengirim strategi baru, lengkap dengan fitur pengecekan link YouTube otomatis.
2. .../[*postId*]/: Halaman untuk membaca detail strategi dan menonton video.

e. Modul Profil Pemain (app/profile/)

Folder ini mengelola tampilan "CV Pemain". Komponennya dipisah agar kode tidak terlalu panjang:

1. .../*components*/PlayerHeroesCard.tsx: Komponen khusus untuk menampilkan grafik level *Hero*.
2. .../*components*/*TeamHistoryCard*.tsx: Komponen untuk menampilkan riwayat perpindahan klan (untuk mendeteksi "kutu loncat").

Dengan memisahkan kode berdasarkan fitur, pengembangan aplikasi menjadi lebih aman. Jika pengembang sedang mengutak-atik kode di folder tournament, fitur di profile tidak akan terganggu atau rusak (*error*), karena kodenya terisolasi masing-masing.

Implementasi Logika dan Keamanan Terpusat (*Shared Logic*)

Direktori lib/ berfungsi sebagai fondasi utilitas utama dalam arsitektur aplikasi. Di sinilah aturan-aturan bisnis, validasi keamanan, dan definisi struktur data disimpan secara terpusat. Tujuannya adalah menerapkan prinsip *Don't Repeat Yourself* (*DRY*), di mana satu fungsi logika dapat digunakan kembali oleh berbagai modul fitur tanpa perlu penulisan ulang, sehingga kode menjadi lebih efisien dan mudah dipelihara. Implementasi pada direktori ini dibagi menjadi tiga lapisan fungsional:

a. Logika Sisi Server & *Admin* (lib/firestore-*admin*/)

Folder ini merupakan Lapisan Logika Terproteksi. Kode di dalamnya hanya dapat dieksekusi di sisi server (*Backend*) karena menggunakan Firebase *Admin SDK* yang memiliki hak akses penuh (*Superuser*) ke seluruh basis data.

1. management.ts: Berisi fungsi validasi hak akses kritis, misalnya memverifikasi apakah pengguna yang menekan tombol "*Kick* Anggota" benar-benar memiliki jabatan Pemimpin Klan yang sah.
2. *users*.ts: Menangani mutasi data profil pengguna yang sensitif, seperti mengubah status verifikasi akun atau memperbarui Token *API*.
3. archives.ts: Mengelola proses pengarsipan data perang lama secara otomatis (*background task*) untuk menjaga performa basis data utama tetap ringan.

b. Logika Interaksi Antarmuka (lib/hooks/)

Folder ini berisi pustaka *Custom Hooks* (fungsi bantuan React) yang bertugas menyederhanakan pengambilan data di sisi tampilan (*Frontend*).

1. useManagedClan.ts: Fungsi ini melakukan abstraksi terhadap kompleksitas pengambilan data klan. Komponen tampilan cukup memanggil satu baris kode untuk mendapatkan data klan terbaru beserta status sinkronisasinya.
2. useNotifications.ts: Mengelola sistem notifikasi *real-time* menggunakan *Firestore Listener*, seperti memperbarui indikator angka pada ikon lonceng saat ada ulasan reputasi baru.

c. Definisi Struktur Data (lib/types/)

Folder ini menyimpan kontrak struktur data (*TypeScript Interfaces*) yang disepakati bersama antara *Frontend* dan *Backend*. Definisi ini menjamin bahwa data yang dikirim dan diterima selalu konsisten. Gambar 4.10 berikut adalah contoh implementasi kode definisi tipe data untuk objek "Pemain" yang mendukung fitur *Sorting* dan Reputasi:

```
// Lokasi: lib/types/user.types.ts
// Fungsi: Menentukan struktur data pemain yang valid

export interface PlayerProfile {
  uid: string;              // Identitas unik pengguna
  displayName: string;      // Nama tampilan
  playerTag: string;        // Tagar gim (#TAG)

  // Data Statistik untuk Fitur Sorting & Turnamen
  thLevel: number;          // Level Town Hall (misal: 12, 13, 16)
  trophies: number;         // Jumlah Piala saat ini
  warStars: number;         // Total Bintang Perang (Pengalaman)

  // Data Sosial untuk Fitur Kepercayaan
```

```
  reputationScore: number;  // Nilai rata-rata bintang (1.0 - 5.0)
  reputationCount: number;  // Jumlah orang yang memberikan ulasan
  isFreeAgent: boolean;     // Status apakah sedang mencari klan?

  // Metadata Sistem
  isVerified: boolean;      // Status validasi API (Centang Biru)
  lastUpdated: Date;        // Waktu sinkronisasi terakhir
}
```

Gambar 4.10 Kode Program Definisi Tipe Data Pemain (*TypeScript Interface*)

Pemisahan folder yang tegas antara firestore-*admin* (Server) dan hooks (Klien) merupakan penerapan prinsip Keamanan Berlapis. Hal ini memastikan bahwa logika bisnis yang kritikal (seperti menghapus data klan) terisolasi sepenuhnya dan tidak akan pernah bocor atau dapat diakses secara sembarangan melalui peramban pengguna.

4.2 Implementasi Logika Sistem dan Basis Data

Sub-bab ini memaparkan hasil dari tahap Konstruksi Prototipe, di mana rancangan kebutuhan dan diagram yang dibuat pada Bab 3 diterjemahkan menjadi kode program yang berfungsi. Fokus utamanya adalah membangun logika bisnis dan manajemen data agar sistem dapat berjalan dengan aman dan lancar sesuai skenario penggunaan.

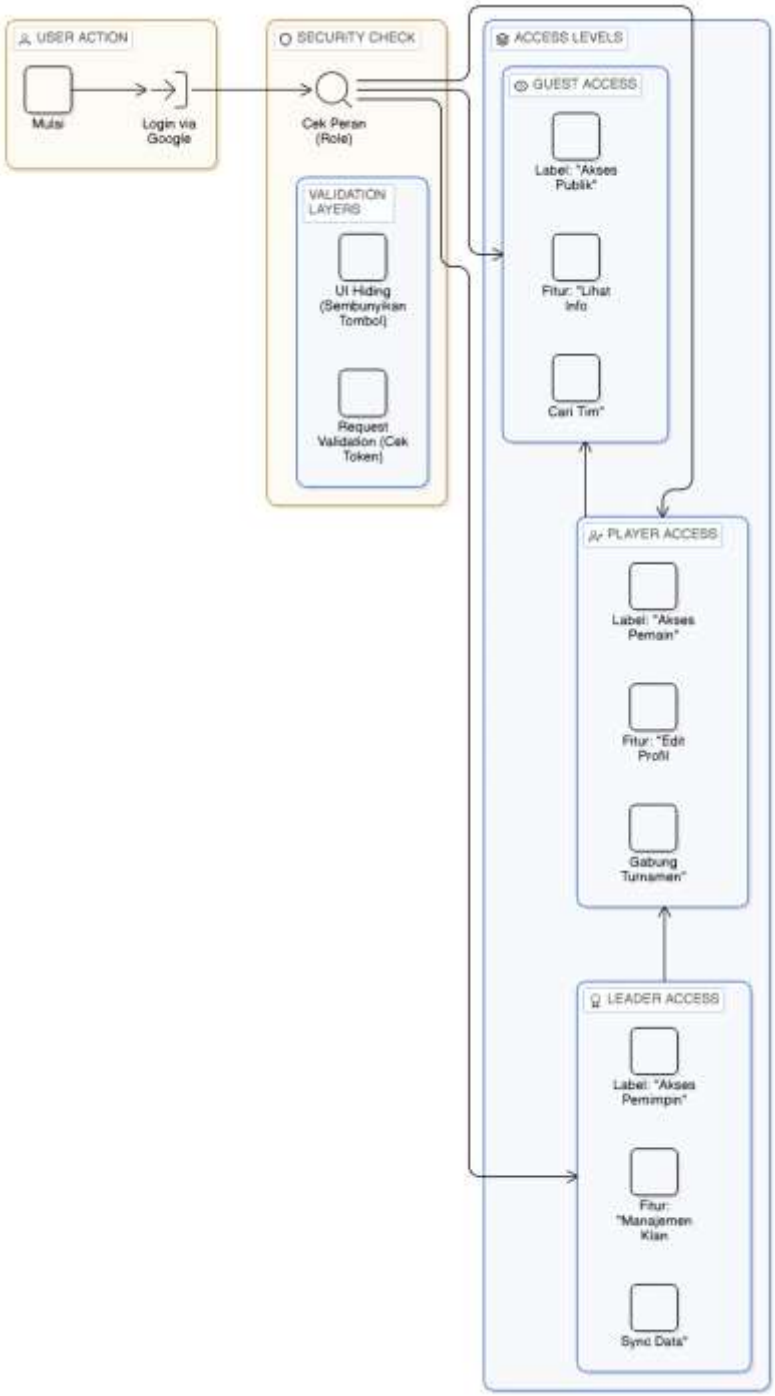
4.2.1 Implementasi Manajemen Aktor dan Hak Akses

Sesuai dengan rancangan pembagian peran pada Bab 3, sistem "ClashHub" menerapkan mekanisme pembagian hak akses (*Role-Based Access Control*). Tujuan utamanya adalah membedakan apa yang bisa dilakukan oleh Pengunjung (publik), Pemain biasa, dan Pemimpin Klan. Mekanisme keamanan ini dibangun menggunakan dua lapisan perlindungan agar sistem tidak mudah dibobol:

a. Pengecekan Tampilan (*Client-Side Protection*):Sistem mengatur tampilan antarmuka secara otomatis berdasarkan peran pengguna. Misalnya, tombol "Edit Profil Klan" hanya akan dirender (dimunculkan) di layar jika pengguna yang *login* terdeteksi memiliki peran *Leader* atau *Co-Leader*. Jika pengguna biasa, tombol tersebut disembunyikan untuk mencegah akses yang tidak sengaja.
b. Pengecekan Server (*Server-Side Validation*):Ini adalah lapisan keamanan yang paling krusial. Meskipun tombol disembunyikan, peretas yang mengerti teknis mungkin bisa memanipulasi kode tampilan. Oleh karena itu, setiap kali ada permintaan perubahan data (misal: menghapus anggota), server akan melakukan verifikasi ulang: "Apakah pengguna

yang mengirim permintaan ini benar-benar Pemimpin Klan?". Jika tidak, permintaan ditolak mentah-mentah oleh server.

Visualisasi alur pengecekan hak akses ini dapat dilihat pada Gambar 4.11 berikut:



Gambar 4.11 Alur Logika Pengecekan Hak Akses (RBAC)

Sumber: Hasil Olahan Peneliti (2025)

Alur proses dimulai dari pengguna melakukan *login*, sistem mengidentifikasi peran (*Role*) dari Token Sesi, kemudian memberikan akses ke menu yang sesuai dengan haknya. Implementasi teknis dari manajemen aktor ini dibagi menjadi tiga bagian spesifik yang akan dibahas pada poin-poin berikut:

Mekanisme Autentikasi Klien (*Client-Side Authentication*)

Sistem autentikasi pada ClashHub dirancang dengan menggabungkan kemudahan Firebase *SDK* di sisi tampilan antarmuka dan keamanan *Cookies* di sisi server. Mekanisme ini terdiri dari dua bagian utama: Manajemen Status *Login* dan Pembuatan Sesi Aman.

a. Manajemen Status Autentikasi (app/context/AuthContext.tsx)

Agar aplikasi tahu kapan harus menampilkan tombol "*Login*" atau foto profil pengguna, sistem menggunakan fitur *React Context*. Fitur ini memantau secara otomatis apakah pengguna sedang *login* atau tidak di peramban mereka.

Gambar 4.12 berikut adalah kode yang mendeteksi perubahan status *login* dan meminta server untuk membuat sesi aman:

```
// Lokasi: app/context/AuthContext.tsx
// Fungsi: Memantau status login pengguna di seluruh aplikasi

import { onAuthStateChanged, User } from "firebase/auth";
import { auth } from "@/lib/firebase";

export const AuthProvider = ({ children }: { children: React.ReactNode })
=> {
  const [user, setUser] = useState<User | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    // Pendengar (Listener) aktif: Mendeteksi jika ada user login/logout
    const unsubscribe = onAuthStateChanged(auth, async (currentUser) => {
      setUser(currentUser);

      // Jika pengguna berhasil login di browser,
      // Beritahu server untuk membuat 'Cookie Sesi' agar server juga tahu.
      if (currentUser) {
         await fetch('/api/login', {
             method: 'POST',
             body: JSON.stringify({ uid: currentUser.uid })
          });
      }

      setLoading(false);
    });

    return () => unsubscribe();
  }, []);
```

```
  return (
    <AuthContext.Provider value={{ user, loading }}>
      {!loading && children}
    </AuthContext.Provider>
  );
};
```

Gambar 4.12 Kode Program Pemantau Status *Login* (Auth Context)

b. Penetapan Sesi Aman (app/*api*/*login*/route.ts)

Setelah pengguna berhasil *login* di tampilan depan, server perlu mencatat identitas tersebut agar bisa melindungi data sensitif. Hal ini dilakukan dengan membuat *HTTP-Only Cookie*, yaitu sejenis "kartu identitas digital" yang disimpan di peramban pengguna namun dikunci agar tidak bisa dibaca oleh kode jahat (*script*). Gambar 4.13 berikut adalah kode implementasi penetapan sesi amannya:

```
// Lokasi: app/api/login/route.ts
// Fungsi: Membuat 'Cookie' aman yang tidak bisa dibaca oleh script jahat

import { NextResponse } from 'next/server';

export async function POST(request: Request) {
  const { uid } = await request.json();

  const oneDay = 60 * 60 * 24; // Sesi berlaku selama 24 jam

  // Membuat cookie dengan format khusus
  // Fitur 'HttpOnly' mencegah pencurian sesi melalui serangan XSS
  const cookieString = `session-token=logged_in_${uid}; Max-Age=${oneDay};
Path=/; HttpOnly; SameSite=Lax;`;

  return new NextResponse(JSON.stringify({ success: true }), {
    status: 200,
    headers: {
      'Set-Cookie': cookieString, // Perintah browser untuk simpan cookie
      'Content-Type': 'application/json',
    },
  });
}
```

Gambar 4.13 Kode Program Pembuatan Sesi Aman di Server

Kombinasi ini memberikan lapisan keamanan ganda. AuthContext membuat antarmuka terasa cepat (responsif) bagi pengguna, sementara *HTTP-Only Cookie* memastikan bahwa kunci akses utama tersimpan aman di server dan sulit dicuri oleh peretas.

Validasi Peran Sisi Server (*Server-Side Role Validation*)

Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal.

Berbeda dengan validasi di tampilan aplikasi yang hanya menyembunyikan tombol, validasi di sisi server berfungsi sebagai benteng terakhir keamanan. Sistem tidak boleh percaya begitu saja pada data yang dikirim dari aplikasi pengguna, karena data tersebut bisa saja dimanipulasi oleh peretas. Oleh karena itu, setiap kali ada permintaan perubahan data sensitif, server akan melakukan pemeriksaan ulang secara ketat. Implementasi utama terdapat pada fungsi verifyUserClanRole. Fungsi ini bertugas memastikan apakah pengguna benar-benar memiliki jabatan Pemimpin Klan sebelum diizinkan mengelola data klan. Gambar 4.14 berikut adalah implementasi kode keamanannya:

```
// Lokasi: lib/firestore-admin/management.ts
// Fungsi: Memastikan pengguna adalah Leader/Co-Leader yang sah

export const verifyUserClanRole = async (
  uid: string,
  clanId: string,
  allowedRoles: string[] = ['leader', 'coLeader']
) => {

  // 1. Ambil data profil terbaru langsung dari Database Server
  // Kita tidak boleh percaya data sesi dari browser karena bisa diedit
user.
  const userRef = adminFirestore.collection('users').doc(uid);
  const userSnap = await userRef.get();

  if (!userSnap.exists) return { isAuthorized: false };

  const userData = userSnap.data();

  // 2. Verifikasi Keanggotaan:
  // "Apakah pengguna ini benar-benar anggota dari klan yang ingin dia
edit?"
  const isMemberOfClan = userData?.clanId === clanId;

  // 3. Verifikasi Jabatan:
  // "Apakah jabatan dia termasuk yang diizinkan (Leader/Co-Leader)?"
  const hasValidRole = allowedRoles.includes(userData?.clanRole);

  // Keputusan Akhir
  if (isMemberOfClan && hasValidRole) {
    return { isAuthorized: true, userProfile: userData };
  } else {
    // Catat percobaan akses ilegal ke dalam log server (Audit Trail)
    console.warn(`[KEAMANAN] Akses Ditolak: User ${uid} mencoba mengakses
Klan ${clanId}`);
    return { isAuthorized: false, userProfile: null };
  }
```

```
};
```

Gambar 4.14 Kode Program Validasi Peran Pengguna di Server

Analisis Keamanan:

a. Pengecekan Langsung: Fungsi ini mengambil data langsung dari basis data server, bukan mempercayai data yang dikirim dari peramban pengguna.
b. Validasi Kontekstual: Sistem tidak hanya mengecek "Apakah dia *Leader*?", tapi juga "Apakah dia *Leader* dari klan ini?". Logika ini mencegah seorang *Leader* dari Klan A mengacak-acak data milik Klan B.
c. Pencatatan Log: Setiap upaya akses ilegal dicatat, sehingga pengelola sistem bisa mengetahui jika ada aktivitas mencurigakan.

Penerapan Aturan Hak Akses (*Access Control Matrix*)

Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal. Cucu Subbab tidak perlu penomoran tetapi dicetak tebal.

Berdasarkan sistem keamanan yang telah dibangun (*Login* dan Validasi Server), sistem menerapkan aturan ketat mengenai batasan wewenang pengguna. Aturan ini disusun dalam bentuk matriks untuk memastikan integritas data dan mencegah penyalahgunaan fitur. Tabel 4.2 berikut merincikan pembagian hak akses untuk setiap peran pengguna dalam sistem "ClashHub":

Tabel 4.2 Matriks Hak Akses Sistem ClashHub

Modul Fitur	**Fungsi / Aksi**	**Pengunjung**	**Pemain**	**Pemimpin Klan**	**Penyelenggara**
Autentikasi	*Login* / Daftar Akun Google	☑	-	-	-
	Verifikasi Token *API* CoC	-	☑	☑	☑
Profil	Ubah Status "Agen Bebas"	-	☑	☑	☑
Pencarian	Cari Tim (Filter & *Sorting* Reputasi)	☑	☑	☑	☑
Manajemen Klan	Lihat Dasbor Publik	☑	☑	☑	☑
	Sinkronisasi Data (*API*)	-	-	☑	-
	Minta Saran Strategi (AI)	-	-	☑	-
	Terima/Tolak Anggota	-	-	☑	-
Turnamen	Lihat Bagan Pertandingan	☑	☑	☑	☑
	Daftar Jadi Peserta	-	☑	☑	-
	Buat Turnamen Baru	-	-	-	☑
	Validasi Skor & *Anti-Smurfing*	-	-	-	☑
Sosial	Memberikan Skor Reputasi	-	☑	☑	☑
	Tulis Postingan Strategi	-	☑	☑	☑

Sumber: Hasil Olahan Sendiri (2025)

Analisis Penerapan Aturan:

a. Pemisahan Tugas (*Segregation of Duties*): Fitur sensitif yang memakan biaya server tinggi, seperti "Analisis AI" dan "Sinkronisasi *API*", dibatasi hak aksesnya hanya untuk Pemimpin Klan. Hal ini mencegah pemborosan kuota *API* oleh anggota biasa yang tidak berkepentingan.

b. Keterbukaan Informasi: Fitur Pencarian Tim dengan algoritma *Sorting* dibuka untuk Pengunjung umum. Tujuannya adalah sebagai strategi akuisisi pengguna (*User Acquisition*), agar orang luar bisa merasakan manfaat fitur pencarian canggih sebelum memutuskan untuk mendaftar.
c. Integritas Kompetisi: Hak untuk memvalidasi skor akhir pertandingan secara manual hanya diberikan kepada Penyelenggara. Hal ini untuk mencegah konflik kepentingan jika peserta turnamen mencoba memanipulasi hasil pertandingannya sendiri.

4.2.2 Implementasi Fitur Pengguna

Setelah hak akses diatur, tahap selanjutnya adalah implementasi fitur-fitur yang berinteraksi langsung dengan pengguna. Modul ini berfokus pada manajemen identitas dan profil pemain yang menjadi fondasi bagi fitur lanjutan seperti Turnamen dan Pencarian Tim.

Autentikasi dan Verifikasi Token (*Token Verification Logic*)

Fitur ini merupakan gerbang utama integritas data sistem. Berbeda dengan aplikasi komunitas biasa yang hanya membutuhkan *email*, "ClashHub" mewajibkan pengguna untuk membuktikan kepemilikan akun *Clash of Clans* menggunakan *API Token*. Mekanisme ini diimplementasikan untuk menjawab masalah "Kepercayaan" (*Trust Issue*) dan "Validasi Turnamen" (*Anti-Smurfing*). Pengguna tidak dapat memalsukan level *Town Hall* mereka karena data diambil langsung dari server Supercell. Gambar 4.15 berikut adalah implementasi logika verifikasi di sisi server:

```
// Lokasi: app/api/coc/verify-player/route.ts
// Fungsi: Memvalidasi kepemilikan akun CoC dan mengunci data statistik

export async function POST(request: NextRequest) {
  try {
    // 1. Cek Login: Pastikan pengguna sudah masuk akun Google
    const authUser = await getSessionUser();
    if (!authUser) {
      return NextResponse.json({ message: 'Akses Ditolak: Harap login
dahulu.' }, { status: 401 });
    }
    const uid = authUser.uid;

    // 2. Terima Data dari Pengguna (Tag Pemain & Token API)
    const payload = await request.json();
    const { playerTag, apiToken } = payload;

    // 3. Verifikasi ke Server Resmi Supercell
    // Server bertanya: "Apakah Token ini benar milik Tagar ini?"
    const isTokenValid = await cocApi.verifyPlayerToken(playerTag,
```

```
apiToken);

    if (!isTokenValid) {
       throw new Error('Token API salah. Pastikan Anda menyalin kode yang
benar dari gim.');
    }

    // 4. Ambil Data Pemain Terbaru dari API
    // Jika token benar, ambil data statistik valid langsung dari sumbernya
    const playerData = await cocApi.getPlayerData(playerTag);

    // 5. Logika Pemetaan Peran Otomatis (Role Mapping)
    // Jika di gim dia adalah Leader, otomatis jadi 'Manager' di aplikasi
    let appRole = 'member';
    if (playerData.role === 'leader' || playerData.role === 'coLeader') {
        appRole = 'manager';
        // Otomatis daftarkan klannya juga ke sistem manajemen
        await createOrLinkManagedClan(playerData.clan.tag, uid);
    }

    // 6. Simpan Data Terverifikasi ke Database (PENTING)
    // Menggunakan Admin SDK agar data terkunci dan aman
    await adminFirestore.collection('users').doc(uid).set({
      isVerified: true,
      playerTag: playerData.tag,

      // Data ini disimpan untuk keperluan fitur SORTING & FILTERING
      thLevel: playerData.townHallLevel,
      trophies: playerData.trophies,
      warStars: playerData.warStars,

      clanRole: playerData.role,    // Jabatan asli di gim
      role: appRole,                // Jabatan di aplikasi
      lastVerified: new Date(),
    }, { merge: true });

    return NextResponse.json({
      message: 'Verifikasi sukses! Akun Anda telah terhubung.',
      profile: playerData
    });

  } catch (error) {
    return NextResponse.json({ message: 'Terjadi kesalahan saat
verifikasi.' }, { status: 500 });
  }
}
```

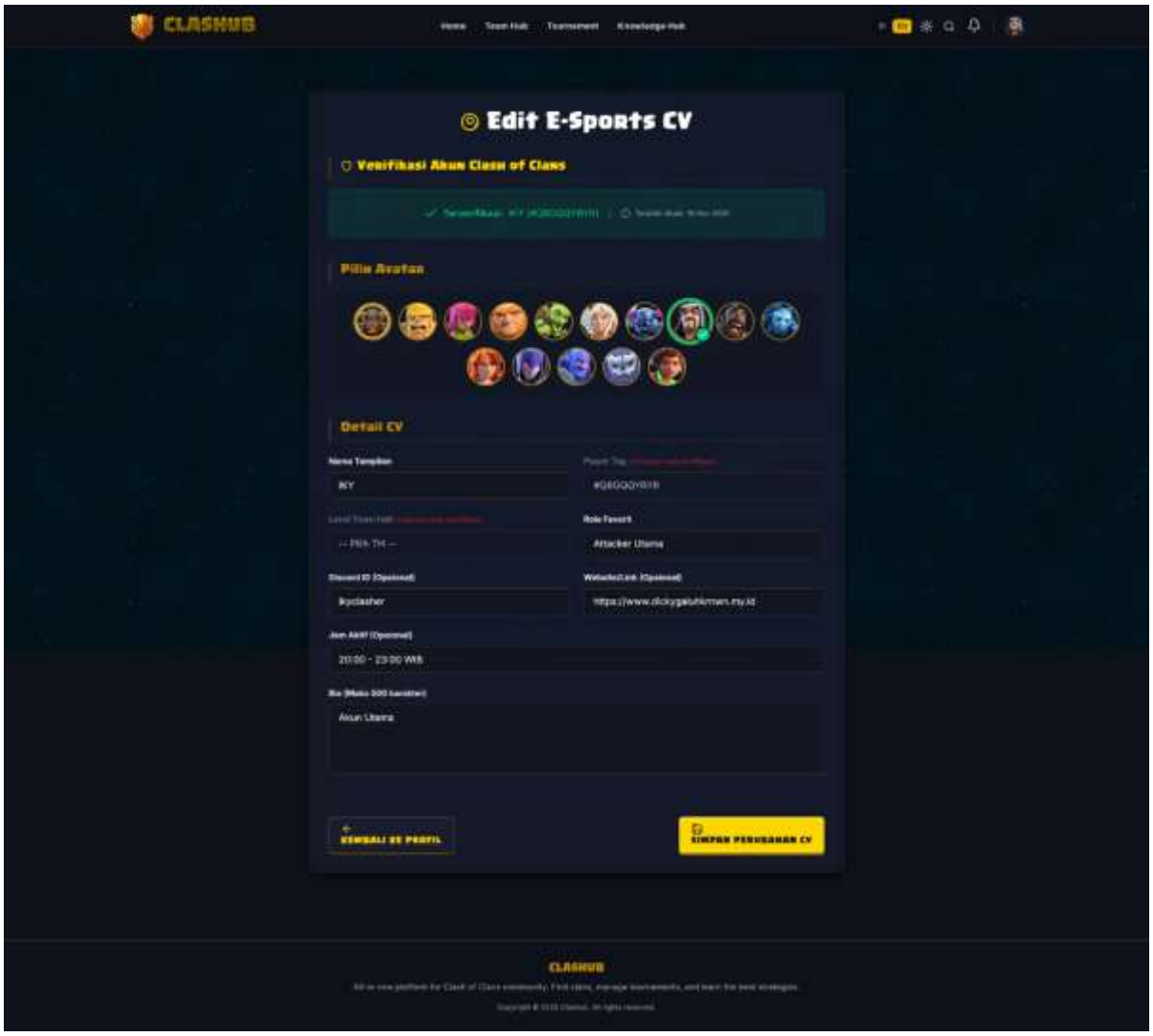
Gambar 4.15 Kode Program Logika Verifikasi Token *API*

Analisis Implementasi:

a. Keamanan Sisi Server: Proses verifikasi dilakukan di *backend* (server) agar Token *API* pengguna tidak pernah terekspos ke jaringan publik.
b. Dukungan Fitur *Sorting*: Kode di atas (poin 6) secara otomatis menyimpan *thLevel* dan *warStars*. Data inilah yang nantinya memungkinkan fitur "Urutkan berdasarkan *TH* Tertinggi" bekerja dengan akurat.

c. Integritas Data: Status *isVerified*: *true* hanya bisa ditulis oleh sistem. Ini mencegah peserta turnamen memanipulasi syarat pendaftaran.

Implementasi antarmuka untuk fitur ini dapat dilihat pada Gambar 4.16 berikut:



Gambar 4.16 Tampilan formulir verifikasi di aplikasi

Sumber: Tangkapan Layar Clashub (2025)

Profil Pemain dan Strategi Data Hibrida (*Hybrid Data Strategy*)

Setelah identitas pengguna terverifikasi, sistem akan menampilkan halaman profil yang berfungsi sebagai "CV Pemain". Tantangan utama di sini adalah bagaimana cara menampilkan data statistik yang akurat (*real-time*) tetapi tetap cepat dibuka di HP. Jika sistem selalu menunggu data dari server Supercell setiap kali halaman dibuka, aplikasi akan terasa lambat.

Oleh karena itu, sistem menerapkan Strategi Data Hibrida: menggabungkan data langsung (*Live*) dengan data yang tersimpan di memori (*Cache*).

a. Logika Tampilan Adaptif (*Frontend*)

Di sisi tampilan, kode program dirancang untuk memprioritaskan data terbaru. Namun, jika pengambilan data sedang berlangsung atau gagal, sistem otomatis menampilkan data cadangan dari *database* lokal agar layar tidak kosong. Gambar 4.17 berikut adalah logika kode program yang menangani prioritas data tersebut:

```
// Lokasi: app/profile/components/PlayerHeroesCard.tsx
// Fungsi: Menentukan data mana yang harus ditampilkan ke pengguna

export const PlayerHeroesCard = ({
  userProfile,    // Data Cadangan (Cache dari Database)
  fullPlayerData, // Data Terbaru (Live dari API CoC)
  isLoading,
}: PlayerHeroesCardProps) => {

  // LOGIKA PRIORITAS:
  // 1. Coba pakai data terbaru (fullPlayerData).
  // 2. Jika belum ada, pakai data cadangan (userProfile).
  // 3. Jika keduanya kosong, pakai daftar kosong.
  const heroesData = fullPlayerData?.heroes ?? userProfile?.cachedHeroes ??
[];

  // LOGIKA LOADING PINTAR:
  // Hanya tampilkan putaran loading jika data BENAR-BENAR kosong.
  // Jika ada data cadangan, tampilkan itu saja (User merasa aplikasi
instan).
  const showLoading = isLoading && !heroesData.length;

  if (showLoading) return <LoadingSpinner />;

  return (
    <div className="bg-gray-800 p-4 rounded-lg">
       <h3>Pahlawan Saya</h3>
       <div className="grid grid-cols-4 gap-2">
         {heroesData.map((hero) => (
           <div key={hero.name}>
             <span className={hero.level === hero.maxLevel ? "text-yellow-
400" : "text-white"}>
               {hero.name}: Lv.{hero.level}
             </span>
           </div>
         ))}
       </div>
    </div>
  );
};
```

Gambar 4.17 Kode Program Logika Tampilan Adaptif (*Hybrid Rendering*)

b. Mekanisme Pembaruan *Cache* (*Backend*)

Agar data cadangan di *database* tidak terlalu usang, sistem akan memperbaruinya secara diam-diam di latar belakang (*background update*) setiap kali pengguna berhasil mengambil data baru. Gambar 4.18 berikut adalah kode yang menangani pembaruan data cadangan tersebut:

```
// Lokasi: app/api/player/update-cache/route.ts
// Fungsi: Menyimpan data terbaru ke database untuk kunjungan berikutnya

export async function POST(request: Request) {
  try {
    // 1. Cek Login
    const sessionUser = await getSessionUser();
    if (!sessionUser) return NextResponse.json({ error: 'Dilarang' }, {
status: 401 });

    const playerData = await request.json();

    // 2. Validasi Keamanan:
    // Pastikan data ini benar-benar milik pengguna yang sedang login.
    // Jangan sampai User A mengupdate data profil User B.
    const userProfile = await
adminFirestore.collection('users').doc(sessionUser.uid).get();

    if (userProfile.data()?.playerTag === playerData.tag) {
       // 3. Simpan ke Database (Admin SDK)
       await
adminFirestore.collection('users').doc(sessionUser.uid).update({
          cachedHeroes: playerData.heroes,
          lastUpdated: new Date()
       });

       return NextResponse.json({ success: true });
    }

    return NextResponse.json({ success: false, message: 'Tag tidak cocok.'
});

  } catch (error) {
    return NextResponse.json({ error: 'Gagal update cache' }, { status: 500
});
  }
}
```

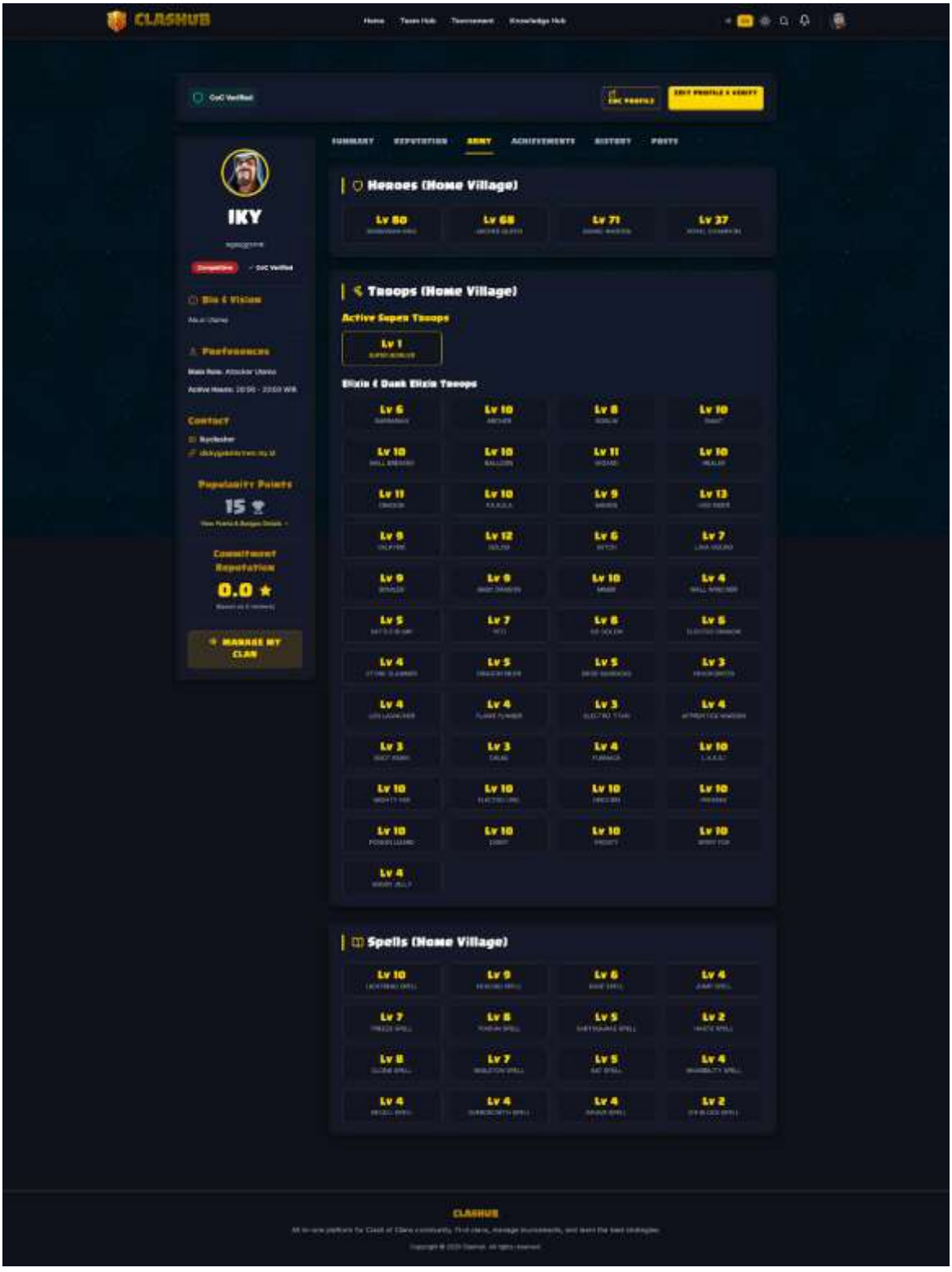
Gambar 4.18 Kode Program Pembaruan Data *Cache* di Server

Analisis Implementasi:

a. Pengalaman Pengguna (*UX*): Dengan logika di atas, pengguna yang sering membuka profilnya akan merasa aplikasi sangat cepat karena data langsung muncul dari *cache*, sementara data baru dimuat diam-diam.

b. Integritas Data: Sistem memastikan bahwa data statistik (seperti level *Hero*) hanya bisa diperbarui jika Tagar Pemain cocok dengan akun yang sedang *login*, mencegah orang lain mengacak-acak profil pemain.

Visualisasi hasil implementasi fitur profil pemain ini dapat dilihat pada Gambar 4.19 berikut:



Gambar 4.19 Antarmuka Halaman Profil Pemain

Sumber: Tangkapan Layar Clashub (2025)

Sistem Reputasi Pemain (*Trust System*)

Untuk menjawab masalah "Krisis Kepercayaan" dan fenomena "Kutu Loncat" yang sering meresahkan komunitas, sistem menerapkan aturan ketat dalam fitur ulasan. Berbeda dengan media sosial di mana siapa saja bisa berkomentar, ClashHub menerapkan aturan Bukti Interaksi. Artinya, seorang pengguna hanya diizinkan memberikan ulasan kepada pemain lain jika mereka terbukti pernah berada dalam satu klan yang sama. Sistem mengecek riwayat perpindahan klan dari kedua belah pihak (Pemberi Ulasan dan Penerima Ulasan) di basis data. Jika tidak ditemukan klan yang sama, sistem akan menolak ulasan tersebut. Gambar 4.20 berikut adalah kode program yang menangani logika validasi tersebut:

```
// Lokasi: app/api/reviews/player/route.ts
// Fungsi: Mencegah ulasan palsu dengan mengecek riwayat main bareng

export async function POST(request: NextRequest) {
  // ... (Pengecekan Login) ...
  const { targetPlayerId, reviewContext } = await request.json();

  // 1. Validasi Konteks Ulasan
  // Ulasan harus jelas: apakah sebagai teman se-klan atau rekan tim e-
sports?
  if (!['clan', 'esports'].includes(reviewContext)) {
     return NextResponse.json({ error: "Konteks tidak jelas" }, { status:
400 });
  }

  // 2. Cek Riwayat Bersama (Shared History Check)
  // Mencegah orang asing memberikan ulasan buruk sembarangan

  // Ambil riwayat klan si penulis dan si target dari database
  const [authorHistory, targetHistory] = await Promise.all([
      getClanHistoryAdmin(sessionUser.uid),
      getClanHistoryAdmin(targetPlayerId),
  ]);

  // Algoritma Pencocokan: Cari apakah ada nama klan yang sama?
  const authorClanTags = new Set(authorHistory.map(doc => doc.clanTag));
  const hasMetBefore = targetHistory.some(doc =>
authorClanTags.has(doc.clanTag));

  if (!hasMetBefore) {
      return NextResponse.json({
         error: 'Dilarang: Anda belum pernah satu klan dengan pemain ini.',
      }, { status: 403 });
  }

  // 3. Simpan Ulasan & Beri Hadiah Poin
  const reviewRef = await adminFirestore.collection('reviews').add({
      // ... data ulasan ...
      createdAt: new Date()
  });

  // Memberikan 10 Poin Popularitas kepada penulis sebagai ucapan terima
kasih
```

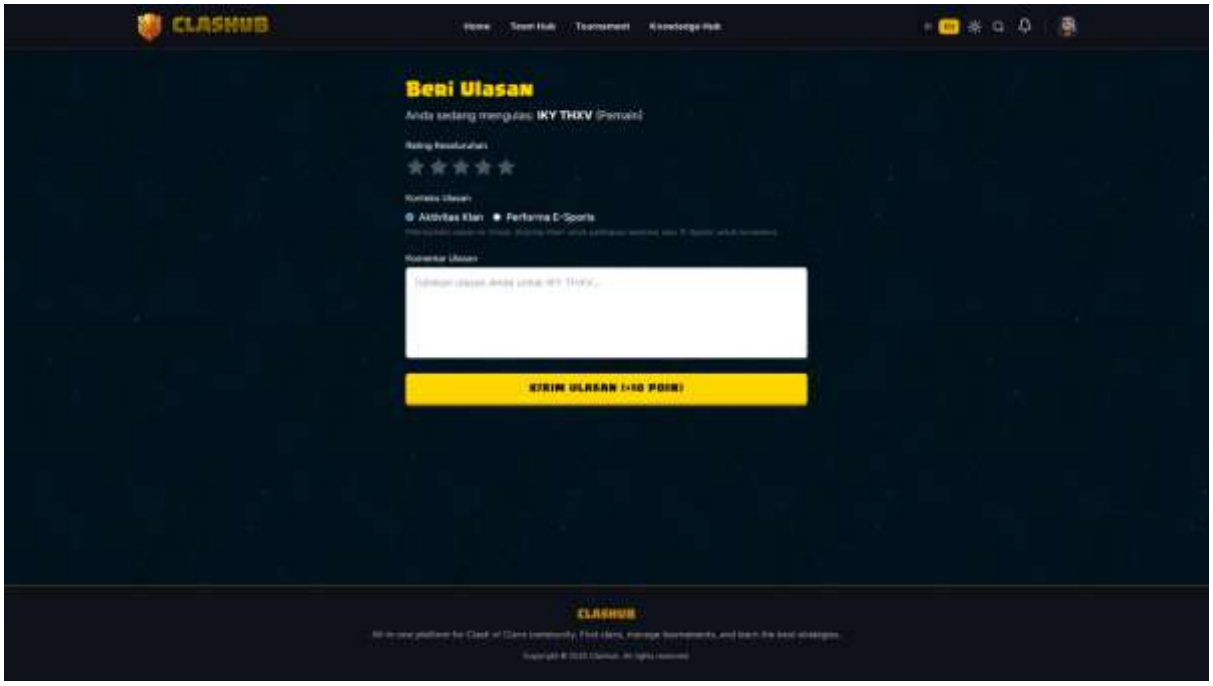
```
  await incrementPopularity(sessionUser.uid, 10);

  return NextResponse.json({ success: true, message: 'Ulasan berhasil
diterbitkan.' });
}
```

Gambar 4.20 Kode Program Validasi Ulasan Berbasis Riwayat Interaksi

Analisis Implementasi:

a. Integritas Sosial: Kode hasMetBefore menjamin bahwa setiap angka bintang yang muncul di profil pemain adalah hasil dari interaksi nyata, bukan serangan dari akun palsu (*bot*).
b. Gamifikasi Positif: Dengan memberikan poin tambahan kepada penulis ulasan, sistem mendorong anggota komunitas untuk aktif saling menilai, sehingga ekosistem kepercayaan terbentuk lebih cepat.
c. Konteks Spesifik: Sistem membedakan penilaian "Klan" (sikap/perilaku) dengan "*E-sports*" (kemampuan bermain), sehingga profil pemain menjadi lebih informatif bagi perekrut.

Visualisasi antarmuka fitur reputasi ini dapat dilihat pada Gambar 4.21 berikut:



Gambar 4.21 Antarmuka Formulir Ulasan dan Profil Reputasi

Sumber: Tangkapan Layar Clashub (2025)

4.2.3 Implementasi Fitur Manajemen Klan

Modul ini adalah jantung dari sistem ClashHub yang dirancang untuk menggantikan tugas pemimpin klan yang selama ini mencatat data secara manual di Excel atau kertas. Fokus utamanya adalah menerapkan konsep Agregasi Data untuk mengubah data statistik mentah menjadi informasi yang mendukung pengambilan keputusan manajerial.

Sinkronisasi Data dan Penilaian Partisipasi

Sistem menerapkan mekanisme cerdas untuk mengolah data agar pemimpin klan tidak perlu menghitung manual. Implementasinya dibagi menjadi dua bagian: Strategi Pengambilan Data dan Logika Penilaian Kinerja.

a. Sinkronisasi Data Bertahap (*Granular Sync*)

   Pada tahap awal pengembangan, sistem mencoba mengambil semua data klan sekaligus, namun sering gagal (*timeout*) karena ukuran data yang terlalu besar. Solusinya, sistem memecah proses pengambilan data menjadi bagian-bagian kecil agar lebih ringan:

   1. Sinkronisasi Perang: Khusus mengecek status perang aktif (sedang berlangsung atau persiapan).
   2. Sinkronisasi Anggota: Khusus memperbarui data donasi dan trofi anggota.
   3. Sinkronisasi Riwayat: Khusus mengambil data perang masa lalu (arsip) yang ukurannya besar.

b. Logika Pendukung Keputusan (Agregasi Keaktifan)

   Sistem memiliki logika khusus untuk menilai kedisiplinan anggota secara otomatis. Penilaian ini didasarkan pada agregasi data jumlah serangan yang dilakukan dalam perang klan. Gambar 4.22 berikut adalah implementasi kode program untuk logika penilaian tersebut:

```
// Lokasi: app/api/coc/sync-managed-clan/logic/participationAggregator.ts
// Fungsi: Menghitung kinerja anggota untuk saran Promosi/Demosi otomatis

// ATURAN PENILAIAN:
const BATAS_PROMOSI = 3;  // Jika sukses 3x berturut-turut -> Saran Naik
Jabatan
const BATAS_DEMOSI = 3;   // Jika gagal 3x berturut-turut -> Saran Turun
Jabatan
const WAJIB_SERANGAN = 2; // Setiap perang harus menyerang 2 kali

export function hitungKeaktifanAnggota(dataInput) {
  const { anggota, riwayatPerang, logJabatan } = dataInput;

  // Cek setiap perang yang pernah terjadi
```

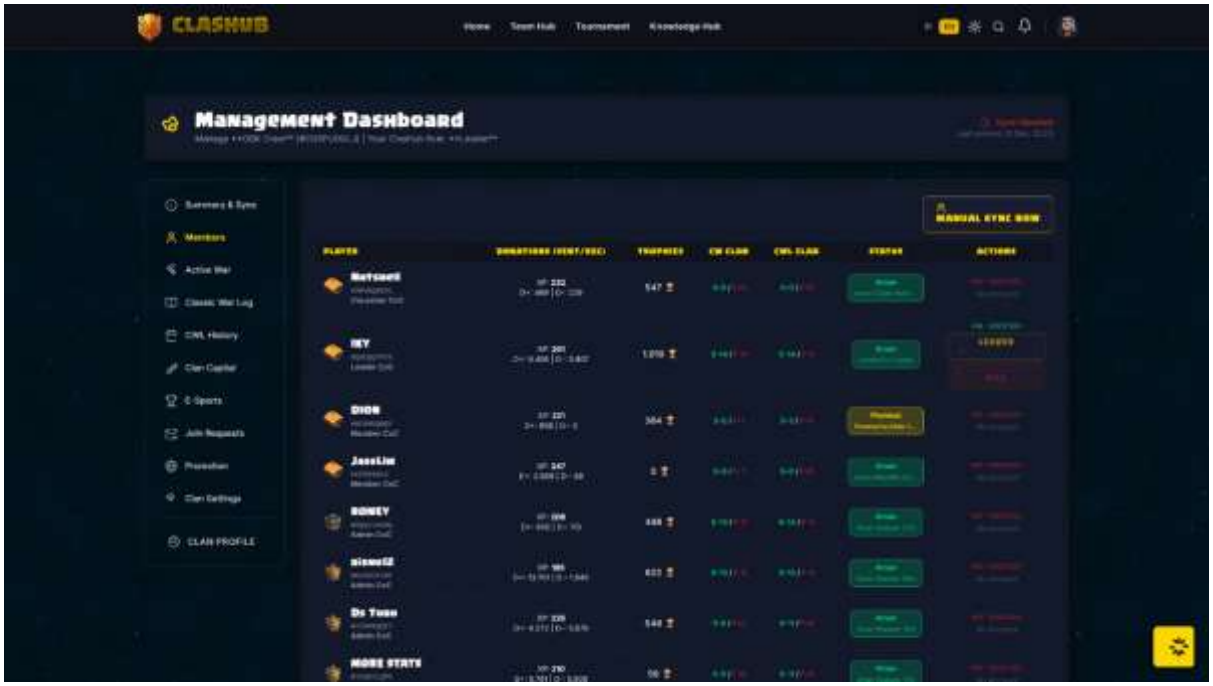
```
  riwayatPerang.forEach((perang) => {

    // Pastikan kita melihat data klan kita sendiri (bukan musuh)
    const dataKlanKita = perang.clan.tag === KLAN_SAYA ? perang.clan :
perang.opponent;

    dataKlanKita.members.forEach((anggotaPerang) => {
       const jumlahSerangan = anggotaPerang.attacks?.length || 0;

       // Skenario SUKSES: Rajin (Menyerang 2 kali)
       if (jumlahSerangan === WAJIB_SERANGAN) {
          tambahPoinSukses(anggotaPerang);
       }
       // Skenario GAGAL: Malas (Tidak menyerang sama sekali)
       else if (jumlahSerangan === 0) {

          // FITUR KEADILAN: "Reset Dosa Masa Lalu"
          // Hanya hitung kegagalan jika terjadi SETELAH kenaikan jabatan
terakhir.
          // Jika dia baru naik jabatan kemarin, kegagalan bulan lalu tidak
dihitung.
          if (waktuPerang > logJabatan.terakhirBerubah) {
             tambahPoinGagal(anggotaPerang);
          }
       }
    });
  });

  // KEPUTUSAN AKHIR SISTEM (Decision Support)
  if (totalGagal >= BATAS_DEMOSI && jabatan === 'elder') {
     return 'Saran: Turunkan Jabatan';
  } else if (totalSukses >= BATAS_PROMOSI && jabatan === 'member') {
     return 'Saran: Naikkan Jabatan';
  }

  return 'Stabil';
}
```

Gambar 4.22 Kode Program Logika Agregasi Penilaian Partisipasi

Analisis Implementasi:

a. Objektivitas: Penilaian murni berdasarkan angka agregat (Apakah menyerang 2 kali?), sehingga menghilangkan penilaian subjektif atau "pilih kasih" dari pemimpin klan.
b. Keadilan Waktu: Kode waktuPerang > logJabatan sangat penting. Ini memastikan bahwa jika seorang anggota baru saja dipromosikan, kesalahan mereka di masa lalu dianggap sudah "dimaafkan" atau di-*reset*, sehingga mereka bisa memulai lembaran baru dengan jabatan barunya.

Visualisasi hasil penilaian otomatis ini pada antarmuka tabel anggota dapat dilihat pada Gambar 4.23 berikut:



Gambar 4.23 Antarmuka Tabel Anggota dengan Label Saran Promosi/Demosi

Sumber: Tangkapan Layar Clashub (2025)

Implementasi Asisten Strategi AI (*Generative AI Implementation*)

Fitur "Asisten Strategi" adalah inovasi utama yang membedakan ClashHub dengan alat manajemen klan biasa. Fitur ini memanfaatkan teknologi Google Gemini AI untuk membaca data angka dan mengubahnya menjadi saran kalimat yang mudah dimengerti. Tantangan terbesar dalam menggunakan AI adalah risiko "halusinasi", di mana AI bisa memberikan jawaban yang salah atau mengarang data yang tidak ada. Untuk mencegah hal ini, sistem tidak membiarkan AI menjawab sembarangan. Sistem menggunakan teknik Penyuntikan Konteks (*Context Injection*), di mana data fakta klan dikirimkan bersamaan dengan pertanyaan pengguna. Implementasi teknisnya dibagi menjadi dua langkah utama:

a. Pengumpulan dan Peringkasan Data

Sebelum mengirim pertanyaan ke AI, sistem harus "belajar" dulu tentang kondisi klan saat ini. Sistem mengambil semua data penting dari basis data secara bersamaan agar cepat. Gambar 4.24 berikut adalah kode untuk mengumpulkan data tersebut:

```
// Lokasi: app/api/clan/ai-assistant/route.ts
// Fungsi: Mengumpulkan bahan data untuk dikirim ke AI

// 1. Ambil semua data secara bersamaan (Paralel) agar hemat waktu
const [profilKlan, dataAnggota, riwayatPerang] = await Promise.all([
  getClanData(clanId),
```

```
  getMembers(clanId),
  getWarLogs(clanId), // Ambil 50 perang terakhir
]);

// 2. Peringkasan Data (Agar hemat biaya token AI)
// Kita hanya ambil info penting saja, buang data yang tidak perlu.
// Contoh: AI tidak perlu tahu ID unik member, cukup Nama, TH, dan Jabatan.
const ringkasanAnggota = dataAnggota.map(m => ({
  nama: m.name,
  level: m.thLevel,
  jabatan: m.role
})).slice(0, 50); // Batasi hanya 50 anggota

const ringkasanPerang = riwayatPerang.map(w => ({
  lawan: w.opponent.name,
  hasil: w.result, // Menang/Kalah
  skor: `${w.clan.stars} vs ${w.opponent.stars}`
})).slice(0, 20); // Ambil 20 perang terakhir saja
```

Gambar 4.24 Kode Program Pengumpulan dan Peringkasan Data AI

b. Penyusunan Instruksi Sistem (*Prompt Engineering*)

Setelah data terkumpul, sistem menyusun "Instruksi Rahasia" kepada AI. Instruksi ini berisi aturan main yang ketat agar AI berperilaku seperti asisten profesional dan tidak melantur. Gambar 4.25 berikut adalah kode untuk menyusun intruksi sistem tersebut:

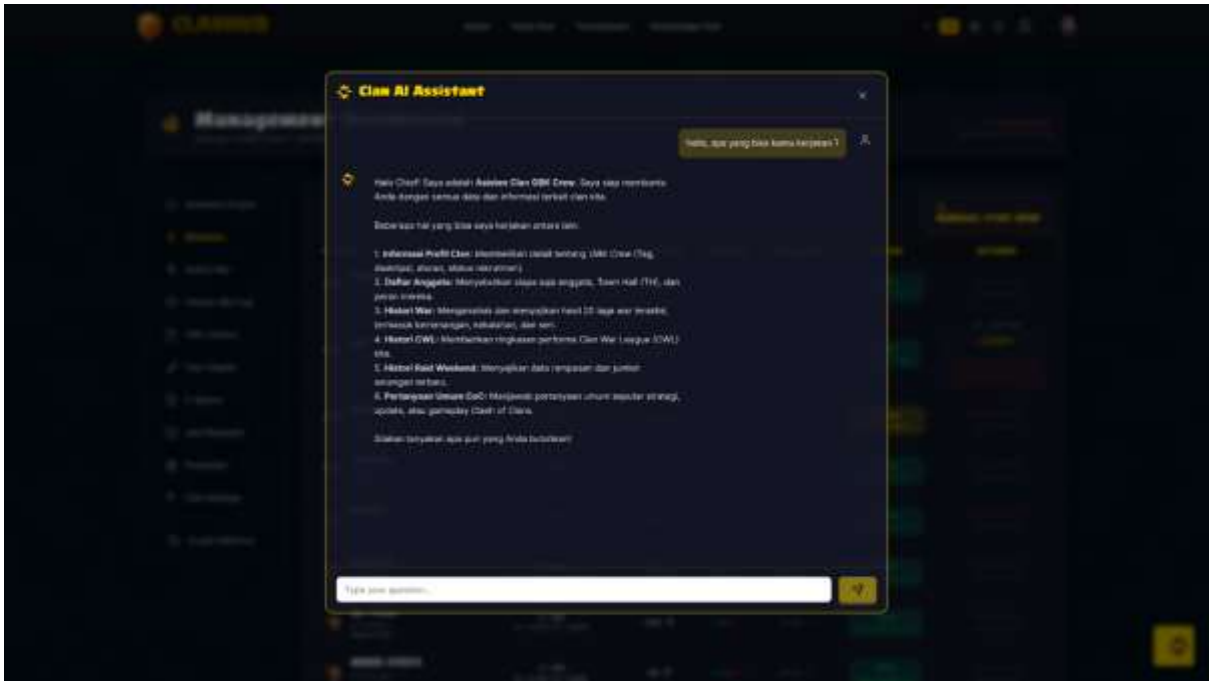
```
// Lokasi: app/api/clan/ai-assistant/route.ts
// Fungsi: Memberikan "Kepribadian" dan "Data" kepada AI

const instruksiSistem = `
  PERAN ANDA:
  Anda adalah "ASISTEN STRATEGI" untuk klan bernama ${profilKlan?.name}.
  Tugas Anda adalah membantu Pemimpin Klan menyusun strategi perang.

  ATURAN WAJIB:
  1. Jawablah menggunakan Bahasa Indonesia yang sopan dan taktis.
  2. JANGAN MENGARANG DATA. Jawaban Anda HARUS berdasarkan data JSON di
bawah ini.
  3. Jika data yang ditanya tidak ada, katakan "Maaf, data statistik belum
cukup."
  4. Berikan saran konkret, misalnya "Anggota X harus menyerang nomor 1".

  --- DATA FAKTA KLAN (JANGAN DIUBAH) ---
  A. PROFIL: ${JSON.stringify(profilKlan)}
  B. DAFTAR ANGGOTA: ${JSON.stringify(ringkasanAnggota)}
  C. RIWAYAT PERANG: ${JSON.stringify(ringkasanPerang)}
  --- AKHIR DATA ---
`;

// Kirim ke Google Gemini
const model = genAI.getGenerativeModel({
  model: 'gemini-pro',
  systemInstruction: instruksiSistem,
});
```

Gambar 4.25 Kode Program Instruksi Sistem (*System Prompt*)

Analisis Implementasi:

a. Pencegahan Kebohongan AI: Dengan menyertakan aturan "JANGAN MENGARANG DATA" dan memberikan data *JSON* asli di dalam instruksi, jawaban AI dijamin akurat sesuai fakta statistik klan.
b. Efisiensi Biaya: Kode peringkasan data (fungsi *.map* dan .slice) memastikan bahwa data yang dikirim tidak terlalu besar. Ini membuat respon AI menjadi lebih cepat dan biaya penggunaan *API* lebih murah (hemat token).
c. Keamanan: Fitur ini dilindungi oleh validasi peran server, sehingga hanya Pemimpin Klan yang sah yang bisa mengakses analisis strategi ini (sesuai Tabel Hak Akses).

Visualisasi antarmuka fitur ini dapat dilihat pada Gambar 4.26 berikut:



Gambar 4.26 Antarmuka Obrolan Asisten Strategi dengan Respon AI

Sumber: Tangkapan Layar Clashub (2025)

Implementasi *Dashboard* Manajemen Terpadu (*Unified Dashboard*)

Tantangan utama dalam mengelola klan secara manual adalah harus berpindah-pindah aplikasi (gim, *spreadsheet*, *WhatsApp*). Untuk mengatasi hal ini, ClashHub menyediakan *Dashboard* Satu Pintu. Di halaman ini, pemimpin klan bisa melihat semua informasi penting

tanpa perlu keluar aplikasi. Agar pengalaman pengguna terasa mulus dan cepat, halaman ini menggunakan teknik Navigasi Sisi Klien (*Client-Side Navigation*). Artinya, saat pengguna berpindah menu (misal: dari "Ringkasan" ke "Daftar Anggota"), halaman tidak perlu dimuat ulang (refresh) dari nol.

a. Struktur Komponen Modular

Halaman manajemen dipecah menjadi bagian-bagian kecil (komponen) yang independen agar mudah dikelola. Gambar 4.27 berikut adalah kode program yang mengatur tampilan menu tersebut:

```
// Lokasi: app/clan/manage/ManageClanClient.tsx
// Fungsi: Mengatur tampilan menu dashboard agar perpindahan tab instan

'use client';

export default function ManageClanClient({ clanData }) {
  // 1. Status Tab Aktif:
  // Menyimpan informasi "Menu apa yang sedang dibuka sekarang?"
  // Default-nya membuka menu 'Ringkasan' (summary).
  const [activeTab, setActiveTab] = useState("summary");

  return (
    <div className="flex flex-col gap-6">

      {/* 2. Bagian Header & Tombol Menu */}
      <ClanManagementHeader
         activeTab={activeTab}
         onTabChange={setActiveTab} // Fungsi untuk ganti menu
         lastSynced={clanData.lastSynced}
      />

      {/* 3. Area Konten Utama */}
      <div className="bg-gray-900 rounded-xl p-6 border border-gray-700">

        {/* Tampilkan konten sesuai menu yang dipilih (Conditional
Rendering) */}

        {activeTab === "summary" && (
           <SummaryTabContent clanData={clanData} />
        )}

        {activeTab === "members" && (
           <MemberTabContent members={clanData.cache?.members} />
        )}

        {activeTab === "active_war" && (
           <ActiveWarTabContent warData={clanData.cache?.currentWar} />
        )}

        {activeTab === "ai_assistant" && (
           <GeminiAssistantTab clanId={clanData.id} />
        )}
```

```
      </div>
    </div>
  );
}
```

Gambar 4.27 Kode Program Navigasi Tabular *Dashboard*

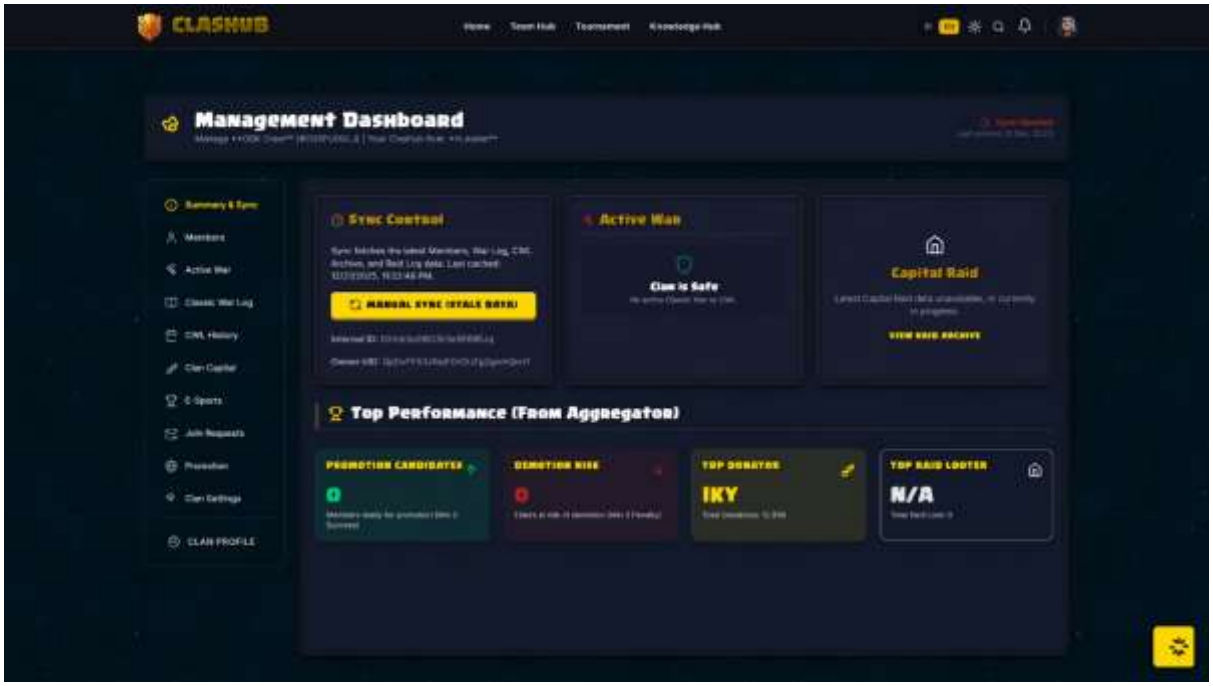
b. Integrasi Data Lintas-Modul

Keunggulan dari arsitektur ini adalah efisiensi data. Data klan (clanData) hanya diambil satu kali saat halaman pertama dibuka. Data tersebut kemudian dibagikan (*distributed*) ke semua tab anak (seperti *SummaryTabContent* atau *MemberTabContent*). Fitur ini bermanfaat saat pengguna berpindah tab, tidak ada lagi proses "*loading*" atau pengambilan data ulang ke server. Semuanya tampil seketika.

Analisis Implementasi:

a. Efisiensi Memori: Dengan teknik *Conditional Rendering* (kode activeTab === ...), peramban tidak perlu memproses elemen yang sedang tidak dilihat pengguna. Contohnya, tabel anggota yang berisi 50 nama tidak akan diproses jika pengguna sedang melihat tab "Ringkasan". Ini sangat membantu menghemat baterai dan memori HP pengguna.
b. Kemudahan Pengembangan: Jika di masa depan ingin menambah fitur baru (misalnya "Manajemen Keuangan"), pengembang cukup membuat komponen baru dan menambahkannya ke dalam logika tab di atas, tanpa merusak kode fitur lainnya.

Visualisasi antarmuka *dashboard* ini dapat dilihat pada Gambar 4.28 berikut:



Gambar 4.28 Tampilan *Dashboard* Manajemen Klan Terpadu

Sumber: Tangkapan Layar Clashub (2025)

4.2.4 Implementasi Modul Turnamen

Modul ini dibangun sebagai solusi teknis untuk masalah penyelenggaraan turnamen komunitas yang sering kacau karena dikelola secara manual menggunakan Excel. Implementasi ini berfokus pada otomatisasi tugas panitia, terutama dalam pembuatan jadwal pertandingan yang adil dan cepat.

Otomatisasi Bagan Pertandingan (*Automatic Bracket Generation*)

Fitur ini mengatasi kendala waktu panitia yang harus menyusun jadwal satu per satu. Sistem menggunakan algoritma pengacakan standar industri, yaitu *Fisher-Yates Shuffle*. Tujuannya adalah memastikan posisi tim diacak secara sempurna seperti mengocok kartu, sehingga tidak ada tim yang merasa dicurangi karena mendapatkan posisi bagan yang tidak menguntungkan. Salah satu keunikan (*Novelty*) sistem ini adalah pemahaman konteks gim. Karena perang di *Clash of Clans* harus dilakukan antar-klan (5vs5 *Friendly War*), sistem secara cerdas akan langsung membagi tugas:

a. Tim 1 diperintahkan masuk ke Klan Panitia A.
b. Tim 2 diperintahkan masuk ke Klan Panitia B.

Hal ini menghilangkan kebingungan peserta mengenai "di mana kami harus bertanding?", yang sering menjadi pertanyaan berulang di grup *WhatsApp* panitia. Gambar 4.29 berikut adalah implementasi kode program untuk logika pembuatan bagan tersebut:

```
// Lokasi: app/api/tournaments/manage/generate-bracket/route.ts
// Fungsi: Mengacak tim dan membuat jadwal pertandingan secara otomatis

export async function POST(request: NextRequest) {
  // ... (Validasi: Pastikan yang akses adalah Panitia) ...

  // Menggunakan 'runTransaction' agar proses aman.
  // Jika internet putus di tengah jalan, semua perubahan dibatalkan
(Rollback).
  // Ini mencegah bagan terbentuk setengah-setengah.
  await adminFirestore.runTransaction(async (transaction) => {

    // 1. Ambil semua tim yang sudah mendaftar
    const pesertaRef = tournamentRef.collection('participants');
    const snapshot = await transaction.get(pesertaRef);
    let timTerdaftar = snapshot.docs.map(doc => doc.data());

    // 2. Acak Posisi Tim (Algoritma Fisher-Yates)
    // Seperti mengocok kartu agar urutannya tidak bisa ditebak
    for (let i = timTerdaftar.length - 1; i > 0; i--) {
      const j = Math.floor(Math.random() * (i + 1));
      [timTerdaftar[i], timTerdaftar[j]] = [timTerdaftar[j],
timTerdaftar[i]];
    }

    // 3. Pasangkan Tim (Pairing)
    for (let i = 0; i < timTerdaftar.length / 2; i++) {
      const matchId = generateUniqueId();
      const timSatu = timTerdaftar[i * 2];     // Ambil tim urutan genap
      const timDua = timTerdaftar[i * 2 + 1];  // Ambil tim urutan ganjil

      // 4. Simpan Jadwal Pertandingan
      const matchBaru = {
        matchId,
        round: 1,
        team1: timSatu.name,
        team2: timDua.name,
        status: 'pending',

        // FITUR PINTAR: Tugaskan tempat bertanding otomatis
        team1Location: data.panitiaClanA, // Tim 1 masuk ke Klan A
        team2Location: data.panitiaClanB, // Tim 2 masuk ke Klan B
      };

      // Simpan ke database
      transaction.set(matchesRef.doc(matchId), matchBaru);
    }

    // 5. Ubah Status Turnamen menjadi "Berlangsung"
    transaction.update(tournamentRef, { status: 'ongoing' });
  });

  return NextResponse.json({ message: 'Bagan pertandingan berhasil dibuat!'
```

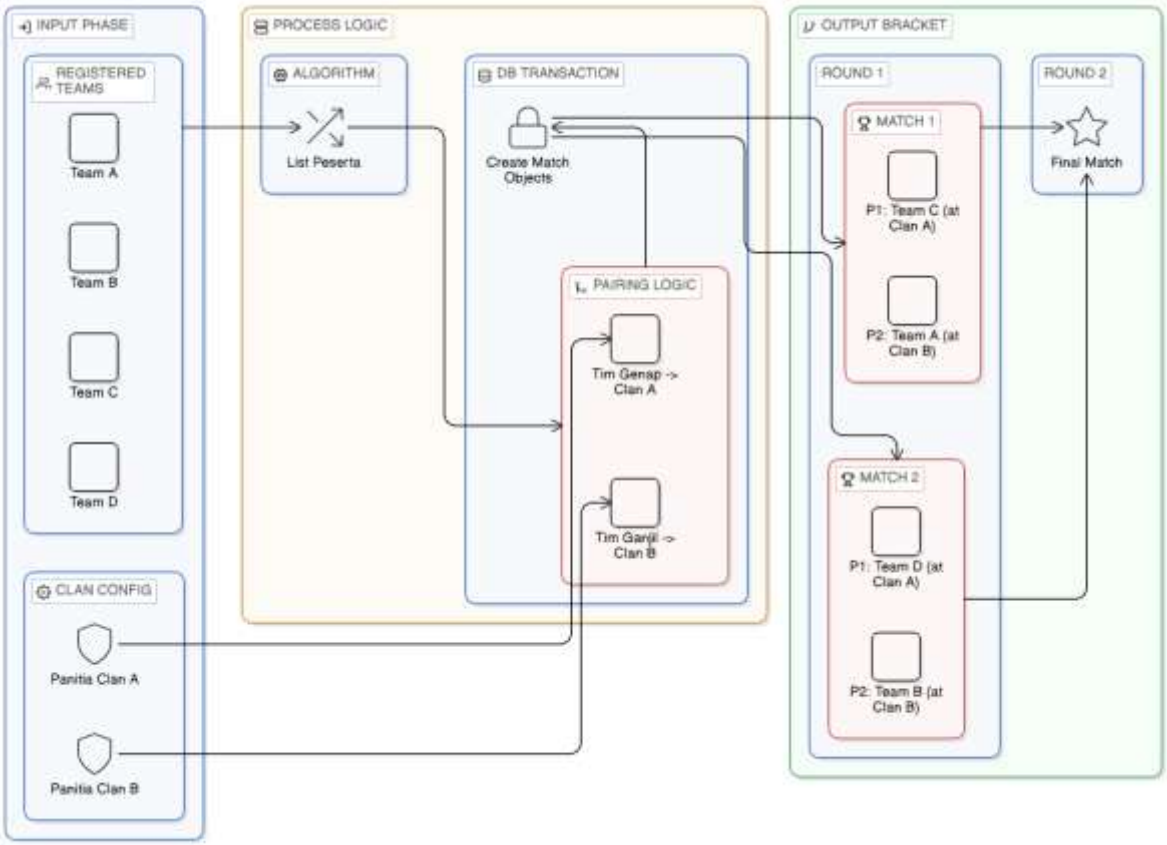
```
});
}
```

Gambar 4.29 Kode Program Algoritma Pembuatan Bagan Otomatis

Analisis Implementasi:

a. Keamanan Data: Penggunaan fungsi *runTransaction* menjamin integritas data. Tidak akan terjadi kasus *error* di mana sebagian tim sudah masuk bagan tapi sebagian lagi tertinggal.
b. Keadilan Sistem: Algoritma pengacakan memastikan setiap tim memiliki peluang yang sama untuk menempati posisi mana pun, menghilangkan kecurigaan bahwa panitia mengatur posisi tim unggulan (*match fixing*).
c. Efisiensi Operasional: Dengan pembagian klan otomatis (*team1Location*), panitia tidak perlu lagi memandu satu per satu peserta secara manual.

Visualisasi alur logika pembuatan bagan ini dapat dilihat pada Gambar 4.30 berikut:



Gambar 4.30 Alur Logika Pembuatan Bagan Turnamen

Sumber: Hasil Olahan Peneliti (2025)

Manajemen Status Turnamen (*Tournament Lifecycle*)

Setiap turnamen memiliki siklus hidup yang pasti, mulai dari "Draf" "Pendaftaran Dibuka" "Pendaftaran Ditutup" "Berlangsung" "Selesai". Sistem ClashHub mengelola perpindahan status ini dengan dua cara: Otomatis (berdasarkan waktu) dan Manual (intervensi panitia).

a. Otomatisasi Jadwal (*Cron Automation*)

Agar panitia tidak perlu siaga 24 jam hanya untuk menekan tombol "Buka Pendaftaran" atau "Tutup Pendaftaran", sistem menggunakan fitur penjadwalan otomatis (*Cron Job*). Sistem akan mengecek jam saat ini dan membandingkannya dengan jadwal turnamen yang telah diatur. Gambar 4.31 berikut adalah implementasi kode program untuk penjadwalan tersebut:

```
// Lokasi: app/api/tournaments/cron/route.ts
// Fungsi: Mengubah status turnamen secara otomatis sesuai jadwal

export async function GET(request: NextRequest) {
  // 1. Keamanan: Pastikan yang memanggil fungsi ini adalah Server Vercel
(bukan hacker)
  const secret = request.nextUrl.searchParams.get('secret');
  if (secret !== process.env.CRON_SECRET) {
     return NextResponse.json({ error: 'Akses Ditolak' }, { status: 401 });
  }

  const sekarang = Timestamp.now();
  const batch = adminFirestore.batch(); // Teknik tulis massal agar cepat

  // 2. Cek: Apakah ada turnamen yang harus BUKA pendaftaran sekarang?
  const jadwalBuka = await adminFirestore.collection('tournaments')
    .where('status', '==', 'scheduled')
    .where('registrationStartsAt', '<=', sekarang)
    .get();

  jadwalBuka.docs.forEach((doc) => {
     batch.update(doc.ref, { status: 'registration_open' });
  });

  // 3. Cek: Apakah ada turnamen yang harus TUTUP pendaftaran?
  const jadwalTutup = await adminFirestore.collection('tournaments')
    .where('status', '==', 'registration_open')
    .where('registrationEndsAt', '<=', sekarang)
    .get();

  jadwalTutup.docs.forEach((doc) => {
     batch.update(doc.ref, { status: 'registration_closed' });
  });

  // Jalankan semua perubahan sekaligus
  await batch.commit();

  return NextResponse.json({ message: 'Status turnamen diperbarui.' });
```

```
}
```

Gambar 4.31 Kode Program Otomatisasi Jadwal Turnamen (*Cron Job*)

b. Penanganan Kuota Ganjil (*Start Under Quota*)

Dalam kondisi nyata, seringkali jumlah peserta yang mendaftar tidak pas dengan slot yang tersedia (misal: hanya 6 tim mendaftar untuk bagan 8 slot). Jika menggunakan sistem manual, panitia akan bingung membuat bagan. Sistem ClashHub mengatasi ini dengan algoritma Penghitungan *BYE* Otomatis. Sistem akan mencari angka pangkat dua terdekat (*Next Power of 2*) dan mengisi kekosongan dengan slot "*BYE*" (Menang Tanpa Tanding). Gambar 4.32 berikut adalah implementasi kode program tersebut:

```
// Lokasi: lib/tournament-logic.ts
// Fungsi: Menangani kondisi jika peserta kurang dari kuota

const handleStartUnderQuota = async (turnamenId, timTerdaftar) => {
  await adminFirestore.runTransaction(async (t) => {

      const jumlahPeserta = timTerdaftar.length;

      // 1. Hitung Pangkat Dua Terdekat (Power of 2)
      // Contoh: Jika peserta 6, ukuran bagan terdekat adalah 8 slot.
      // Berarti ada 2 slot kosong (8 - 6 = 2 BYE).
      const ukuranBagan = Math.pow(2, Math.ceil(Math.log2(jumlahPeserta)));
      const jumlahBye = ukuranBagan - jumlahPeserta;

      // 2. Siapkan Slot Kosong (BYE)
      const daftarTim = [...timTerdaftar];
      for (let i = 0; i < jumlahBye; i++) {
         daftarTim.push(null); // 'null' artinya tidak ada lawan (BYE)
      }

      // 3. Acak Posisi Tim
      const timAcak = shuffleArray(daftarTim);

      // 4. Buat Pertandingan
      for (let i = 0; i < ukuranBagan; i += 2) {
         const tim1 = timAcak[i];
         const tim2 = timAcak[i+1];

         let pemenang = null;
         let status = 'pending';

         // Logika Otomatis Menang (BYE)
         if (tim1 && !tim2) {
              pemenang = tim1; // Tim 1 menang otomatis
              status = 'completed';
         }
         else if (!tim1 && tim2) {
              pemenang = tim2; // Tim 2 menang otomatis
              status = 'completed';
```

```
          }

          // Simpan pertandingan ke database...
      }

      // Paksa status turnamen jadi 'Berlangsung'
      t.update(tournamentRef, { status: 'ongoing' });
  });
}
```

Gambar 4.32 Kode Program Logika Penanganan Kuota Ganjil (*BYE System*)

Analisis Implementasi:

a. Efisiensi Waktu: Dengan otomatisasi (*Cron Job*), panitia tidak perlu khawatir lupa membuka atau menutup pendaftaran.
b. Fleksibilitas: Algoritma *Start Under Quota* memastikan turnamen tetap bisa berjalan meskipun peserta tidak memenuhi kuota maksimal, sehingga kompetisi tidak perlu dibatalkan.

Panel Kontrol Penyelenggara (*Organizer Panel*)

Selain otomatisasi, sistem juga memberikan kendali penuh kepada panitia untuk mengatur jalannya turnamen. Fitur ini sangat penting untuk menangani situasi di lapangan, seperti mendiskualifikasi peserta atau memvalidasi skor sengketa. Implementasi ini difokuskan pada dua fitur krusial: Manajemen Peserta yang Aman dan Logika Pemindahan Babak.

a. Manajemen Peserta Aman (Mencegah *Overbooking*)

   Tantangan teknis terbesar saat pendaftaran turnamen dibuka adalah risiko "Balapan Klik" (*Race Condition*). Contoh kasus: Kuota tinggal 1 slot, tapi ada 2 *admin* yang menekan tombol "Terima" untuk dua tim berbeda secara bersamaan. Tanpa pengaman, sistem bisa salah mencatat jumlah peserta melebihi kapasitas. Untuk mencegah hal ini, sistem menggunakan teknik Transaksi *Database* (*Atomic Transaction*). Teknik ini akan "mengunci" data turnamen sementara waktu saat proses penambahan peserta sedang berjalan. Gambar 4.33 berikut adalah kode programnya:

```
// Lokasi: app/api/tournaments/manage/participant/route.ts
// Fungsi: Menerima peserta tanpa takut kuota jebol

export async function POST(request: NextRequest) {
  // ... (Validasi: Pastikan yang akses adalah Panitia) ...

  // Gunakan Transaksi Database:
  // "Kunci" data turnamen sementara agar tidak ada yang bisa mengubahnya
```

```
bersamaan
  const hasil = await adminFirestore.runTransaction(async (transaction) =>
{

    // 1. Ambil Data Turnamen Terbaru (Real-time)
    const docTurnamen = await transaction.get(tournamentRef);
    const data = docTurnamen.data();

    // 2. Cek Kuota di detik terakhir
    if (statusBaru === 'approved') {
       // Pertanyaan Kritis: "Apakah kursi masih ada?"
       if (data.jumlahPesertaSaatIni >= data.kapasitasMaksimal) {
          throw new Error('Kuota Penuh! Tidak bisa menerima peserta
lagi.');
       }

       // Jika kursi ada, tambah penghitung peserta (+1)
       transaction.update(tournamentRef, {
           jumlahPesertaSaatIni: data.jumlahPesertaSaatIni + 1
       });
    }

    // 3. Update Status Peserta
    transaction.update(pesertaRef, { status: statusBaru });
  });

  return NextResponse.json({ message: 'Status peserta diperbarui.' });
}
```

Gambar 4.33 Kode Program Transaksi Aman Penerimaan Peserta

b. Logika Pemindahan Babak (*Next Match Logic*)

Setelah skor pertandingan dimasukkan, sistem harus tahu: "Siapa pemenangnya? Dan dia harus maju ke kotak nomor berapa?". Sistem menggunakan rumus matematika sederhana untuk menentukan posisi tim di babak selanjutnya secara otomatis, tanpa perlu *input* manual panitia. Gambar 4.34 berikut adalah kode programnya:

```
// Lokasi: app/api/tournaments/match/report/route.ts
// Fungsi: Memindahkan pemenang ke babak selanjutnya (Next Match)

function hitungPertandinganSelanjutnya(matchIdSaatIni) {
  // Contoh ID Pertandingan: "Upper-Round1-Match1"
  const [bracket, ronde, nomorMatch] = matchIdSaatIni.split('-');

  // RUMUS SISTEM GUGUR:
  // 1. Pemenang Match 1 dan Match 2 akan bertemu di Match 1 pada ronde
berikutnya.
  // 2. Pemenang Match 3 dan Match 4 akan bertemu di Match 2, dst.

  const rondeSelanjutnya = parseInt(ronde) + 1;
  const nomorMatchSelanjutnya = Math.ceil(parseInt(nomorMatch) / 2);

  // Tentukan posisi slot:
```

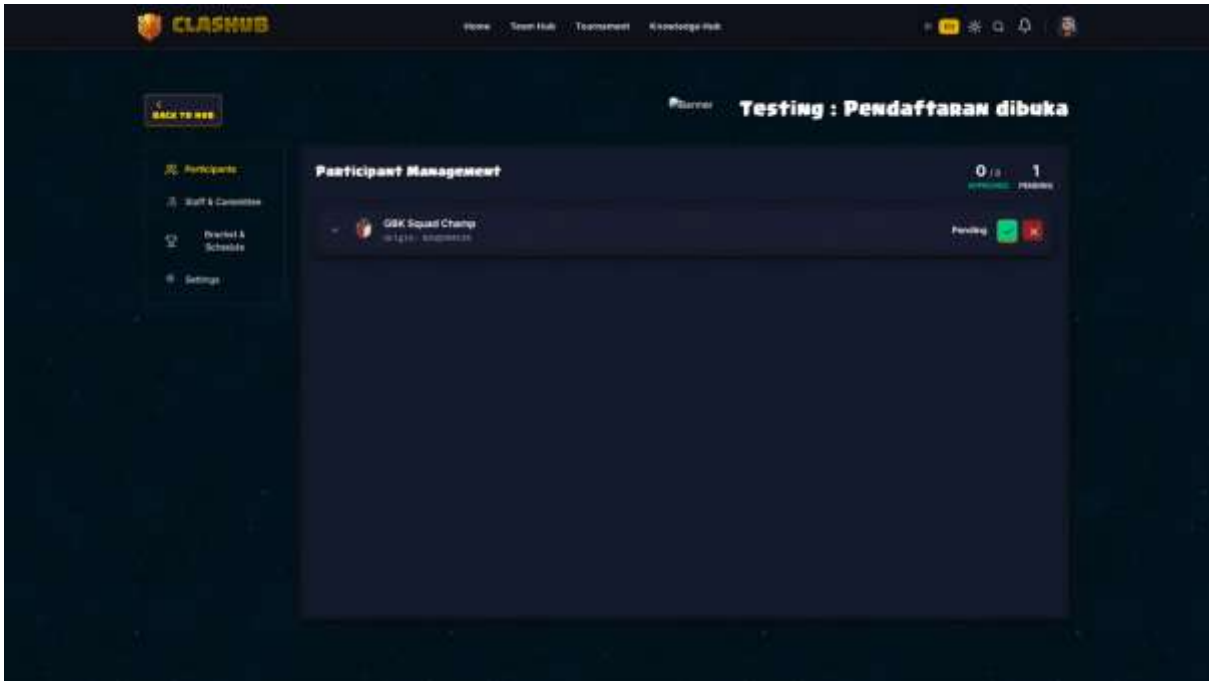
```
  // Ganjil masuk slot Atas (Team 1), Genap masuk slot Bawah (Team 2)
  const slotSelanjutnya = (parseInt(nomorMatch) % 2 === 1) ? 'team1' :
'team2';

  return {
     id: `Upper-Round${rondeSelanjutnya}-Match${nomorMatchSelanjutnya}`,
     slot: slotSelanjutnya
  };
}
```

Gambar 4.34 Kode Program Logika Penentuan Pertandingan Selanjutnya

Analisis Implementasi:

a. Integritas Kuota: Kode transaksi menjamin jumlah peserta yang diterima tidak akan pernah melebihi kapasitas (misal: 8 tim), meskipun trafik pendaftaran sangat tinggi.
b. Efisiensi Alur: Dengan fungsi hitung, bagan pertandingan di aplikasi akan langsung ter-*update* detik itu juga setelah skor dimasukkan. Panitia tidak perlu lagi menggambar ulang bagan di papan tulis atau Excel.

Visualisasi antarmuka panel kontrol penyelenggara dapat dilihat pada Gambar 4.35 berikut:



Gambar 4.35 Antarmuka Panel Manajemen Peserta dan Skor

Sumber: Tangkapan Layar Clashub (2025)

4.2.5 Implementasi Modul Pencarian Tim (*Team Hub*)

Sub-bab ini menjelaskan realisasi teknis dari fitur pencarian klan. Tantangan utama yang dihadapi adalah keterbatasan fitur pencarian bawaan gim yang tidak bisa memfilter klan berdasarkan "Visi" (Kompetitif/Santai) atau mengurutkan berdasarkan "Reputasi". Sistem ClashHub mengatasi hal ini dengan menerapkan Strategi Pencarian Hibrida: menggunakan data *API* untuk statistik akurat, dan *database* internal untuk fitur penyaringan canggih.

Strategi Pencarian dan Penyimpanan Sementara (*Caching*)

Saat pengguna mencari klan menggunakan Tagar (#), sistem tidak langsung meminta data ke server Supercell setiap saat. Sistem akan mengecek penyimpanan lokal terlebih dahulu. Logika "Cek Kesegaran Data":

a. Jika data klan sudah ada di *database* dan baru diperbarui kurang dari 6 jam lalu, sistem akan menampilkan data tersebut (Hemat Kuota).
b. Jika data sudah usang atau belum ada, barulah sistem meminta data terbaru ke *API* Supercell, lalu menyimpannya untuk pencarian berikutnya.

Gambar 4.36 berikut adalah implementasi kode program tersebut:

```
// Lokasi: app/api/coc/search-clan/route.ts
// Fungsi: Mencari klan dengan cerdas (Cek Database dulu, baru API)

const BATAS_WAKTU_CACHE = 1000 * 60 * 60 * 6; // Data dianggap 'segar'
selama 6 jam

async function cariKlanPublik(tagKlan) {
  // 1. Cek di Database Lokal
  const dataLokal = await getPublicClanIndex(tagKlan);

  if (dataLokal) {
     const waktuUpdate = new Date(dataLokal.lastUpdated);
     const selisihWaktu = Date.now() - waktuUpdate.getTime();

     // Jika data masih baru, pakai saja data lokal (Cepat & Hemat)
     if (selisihWaktu < BATAS_WAKTU_CACHE) {
        return dataLokal;
     }
  }

  // 2. Jika data lama/tidak ada, ambil dari API Supercell
  const dataBaru = await cocApi.getClanInfo(tagKlan);

  // 3. Simpan data baru tersebut agar pencarian berikutnya instan
  await saveToPublicIndex(dataBaru);

  return dataBaru;
}
```

Gambar 4.36 Kode Program Logika Pencarian dengan *Caching*

Implementasi Algoritma Pengurutan (*Sorting Logic*)

Fitur ini dibangun khusus untuk menjawab kebutuhan pemain yang ingin mencari klan terbaik, bukan sembarang klan. Sistem memanfaatkan kemampuan *indexing* dari Firebase untuk melakukan pengurutan data secara cepat. Pengguna dapat memilih parameter pengurutan:

a. Reputasi Tertinggi: Menampilkan klan yang paling dipercaya komunitas (Bintang 5).
b. *Town Hall* Tertinggi: Menampilkan klan dengan rata-rata kekuatan akun tertinggi (untuk pemain kompetitif).

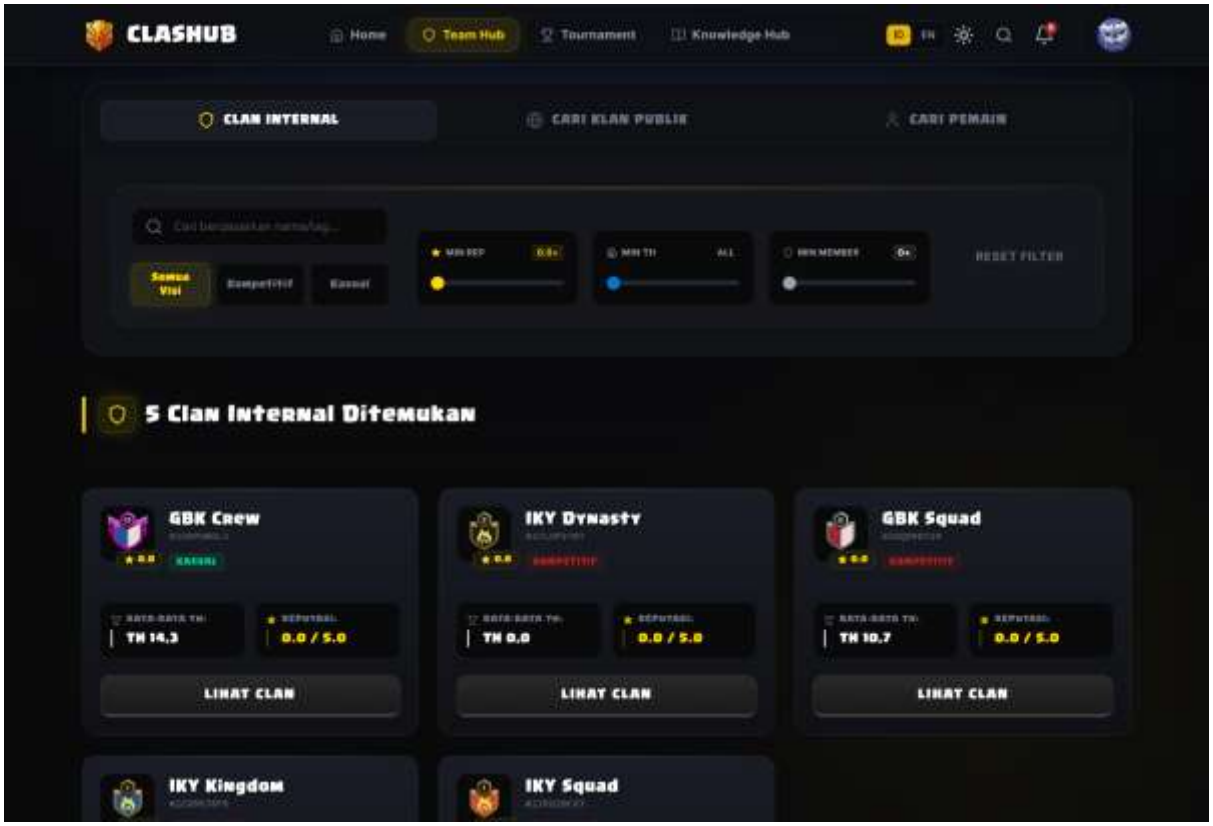
Gambar 4.37 berikut adalah implementasi programnya:

```
// Lokasi: app/api/clan-hub/filter/route.ts
// Fungsi: Menyaring dan mengurutkan klan berdasarkan pilihan pengguna

export async function POST(request: Request) {
  const { filterVisi, metodeSortir } = await request.json();

  let query = adminFirestore.collection('managedClans');

  // 1. Terapkan Filter (Penyaringan)
  // Contoh: Hanya tampilkan klan yang visinya "Kompetitif"
  if (filterVisi) {
     query = query.where('vision', '==', filterVisi);
  }

  // 2. Terapkan Sorting (Pengurutan)
  // Switch case untuk menangani berbagai jenis pengurutan
  switch (metodeSortir) {
     case 'REPUTASI_TERTINGGI':
        // Urutkan berdasarkan kolom 'reputationScore' dari besar ke kecil
        query = query.orderBy('reputationScore', 'desc');
        break;

     case 'LEVEL_TERTINGGI':
        // Urutkan berdasarkan Level Klan
        query = query.orderBy('clanLevel', 'desc');
        break;

     case 'TH_RATA_RATA':
        // Urutkan berdasarkan kekuatan rata-rata anggota
        query = query.orderBy('averageTownHall', 'desc');
        break;

     default:
        // Default: Urutkan berdasarkan klan terbaru
        query = query.orderBy('createdAt', 'desc');
  }

  const snapshot = await query.limit(20).get();
```

```
  return NextResponse.json(snapshot.docs.map(doc => doc.data()));
}
```

Gambar 4.37 Kode Program Logika Filter dan *Sorting* Dinamis

Kode pada Gambar 4.37 membuktikan bahwa fitur pengurutan (*sorting*) dijalankan langsung di level *database query* (*orderBy*). Hal ini jauh lebih efisien dibandingkan mengambil semua data lalu mengurutkannya di aplikasi (*client-side sorting*), yang bisa membuat HP pengguna menjadi lambat. Visualisasi antarmuka hasil pencarian dengan fitur *sorting* aktif dapat dilihat pada Gambar 4.38 berikut:



Gambar 4.38 Antarmuka Pencarian Tim dengan Opsi Pengurutan Aktif

Sumber: Tangkapan Layar Clashub (2025)

4.2.6 Implementasi Modul Ekosistem Komunitas (*Community Ecosystem*)

Jika fitur "Pencarian Tim" berfokus pada menemukan orang, maka fitur Pusat Pengetahuan (*Knowledge Hub*) berfokus pada pertukaran ilmu. Modul ini dibangun untuk memberikan wadah bagi komunitas dalam berbagi strategi penyerangan dan desain pertahanan (*base layout*). Implementasi teknisnya mencakup dua aspek utama: Formulir Cerdas dan Logika Interaksi Sosial.

Formulir Konten Dinamis (*Dynamic Input*)

Agar data yang masuk terstruktur rapi, sistem menerapkan formulir *input* yang adaptif. Kolom isian akan berubah secara otomatis sesuai dengan kategori konten yang dipilih pengguna. Logika Adaptasi:

a. Jika pengguna memilih kategori "Strategi Serangan", sistem mewajibkan *input* Tautan YouTube. Sistem otomatis memvalidasi tautan dan menampilkan pratinjau (*thumbnail*) video.
b. Jika pengguna memilih "Desain Basis", sistem mewajibkan *input* Tautan Penyalinan (*Deep Link*) agar pemain lain bisa langsung menyalin desain tersebut ke dalam gim dengan satu klik.

Gambar 4.39 berikut adalah potongan kode yang menangani logika tampilan formulir tersebut:

```
// Lokasi: app/knowledge-hub/components/form/PostFormGroup.tsx
// Fungsi: Menampilkan kolom input yang berbeda sesuai kategori yang
dipilih

export default function PostFormGroup({ category, form }) {
  return (
    <div className="space-y-4">

      {/* Logika: Jika Kategori Strategi, Wajib isi Link YouTube */}
      {category === "Strategi Serangan" && (
        <div className="bg-gray-800 p-4 rounded">
           <label>Link Video YouTube (Wajib)</label>
           <input
              {...form.register("videoUrl", { required: true })}
              placeholder="[https://youtube.com/](https://youtube.com/)..."
              onChange={(e) => previewThumbnail(e.target.value)}
           />
           {/* Tampilkan preview thumbnail otomatis */}
           <VideoPreview url={form.watch("videoUrl")} />
        </div>
      )}

      {/* Logika: Jika Kategori Base, Wajib isi Link Copy Base */}
      {category === "Base Building" && (
        <div className="bg-gray-800 p-4 rounded">
           <label>Link Copy Base (Deep Link)</label>
           <input
              {...form.register("troopLink", {
                 required: true,
                 pattern: /^https:\/\/link\.clashofclans\.com\// //
Validasi format link resmi
              })}
              placeholder="[https://link.clashofclans.com/](https://link.cl
ashofclans.com/)..."
           />
           <p className="text-xs text-gray-400">
```

```
              Pastikan link berasal dari domain resmi Supercell.
            </p>
          </div>
        )}

    </div>
  );
}
```

Gambar 4.39 Kode Program Logika Formulir *Input* Dinamis

Logika Interaksi Sosial (*Atomic Like System*)

Fitur "Suka" (*Like*) terlihat sederhana, namun menantang secara teknis jika diakses oleh ribuan pengguna secara bersamaan. Risiko utamanya adalah kesalahan hitung (*race condition*), misalnya 10 orang menekan "*Like*" bersamaan tapi sistem hanya menghitung 1. Untuk mengatasinya, sistem menggunakan Transaksi Atomik (*Atomic Increment*) dari Firebase. Fitur ini menjamin angka jumlah *like* akan selalu akurat. Gambar 4.40 berikut adalah implementasi programnya:

```
// Lokasi: app/api/posts/[postId]/like/route.ts
// Fungsi: Menangani aksi Like/Unlike dengan aman

export async function POST(request: NextRequest, { params }) {
  const session = await getSessionUser();
  if (!session) return NextResponse.json({ error: "Harap login dahulu" }, {
status: 401 });

  const postRef = adminFirestore.collection('posts').doc(params.postId);

  // Data 'siapa yang like' disimpan di sub-koleksi terpisah
  // agar dokumen utama postingan tidak menjadi berat/gemuk.
  const likeRef = postRef.collection("likes").doc(session.uid);

  // Jalankan Transaksi Aman
  await adminFirestore.runTransaction(async (t) => {
    const likeDoc = await t.get(likeRef);

    if (likeDoc.exists) {
      // SKENARIO UNLIKE (Batal Suka):
      // 1. Hapus data penanda user ini sudah like
      t.delete(likeRef);
      // 2. Kurangi hitungan total (-1) secara atomik
      t.update(postRef, {
           likesCount: FieldValue.increment(-1)
       });
    } else {
      // SKENARIO LIKE (Suka):
      // 1. Simpan data user ini
      t.set(likeRef, { userId: session.uid, createdAt: new Date() });
      // 2. Tambah hitungan total (+1) secara atomik
      t.update(postRef, {
           likesCount: FieldValue.increment(1)
```

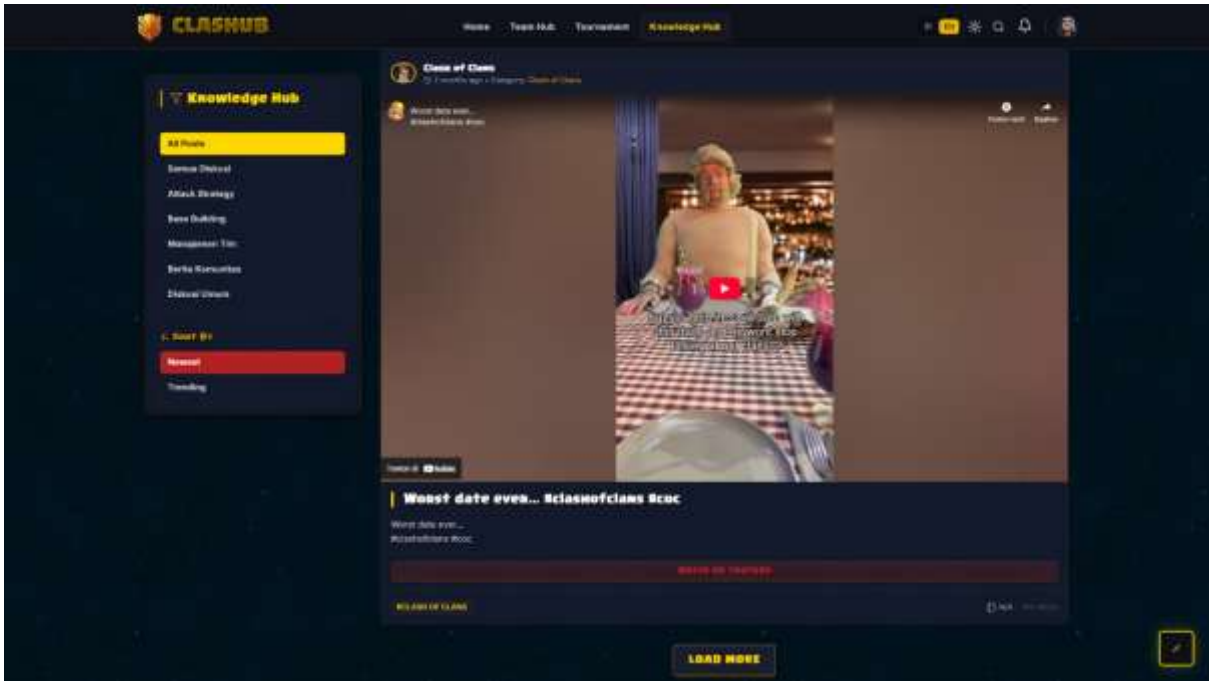
```
        });
    }
  });

  return NextResponse.json({ success: true });
}
```

Gambar 4.40 Kode Program Logika Transaksi *Like*/*Unlike* Aman

Analisis Implementasi:

a. Validasi *Input* (*UX*): Kode pada Gambar 4.39 mencegah pengguna memasukkan link palsu atau sembarangan, menjaga kualitas konten di platform.
b. Integritas Data: Fungsi *FieldValue.increment* pada Gambar 4.40 menjamin angka popularitas konten selalu valid, yang nantinya bisa digunakan untuk algoritma "Konten Terpopuler".

Visualisasi antarmuka Pusat Pengetahuan dapat dilihat pada Gambar 4.41 berikut:



Gambar 4.41 Antarmuka Pusat Pengetahuan dan Detail Strategi

Sumber: Tangkapan Layar Clashub (2025)

4.2.7 Implementasi Struktur Data (*Data Structure Implementation*)

Sub-bab ini menjabarkan realisasi struktur basis data NoSQL yang telah dirancang pada Bab 3. Implementasi ini tidak menggunakan perintah SQL (*CREATE TABLE*), melainkan

didefinisikan melalui Kontrak Tipe Data (*TypeScript Interfaces*) di dalam kode aplikasi. Pendekatan ini menjamin konsistensi data yang keluar-masuk antara server dan klien.

Koleksi Pengguna (*users*)

Koleksi ini adalah repositori sentral untuk profil pengguna. Struktur dokumen dirancang dengan konsep "Dokumen Gemuk" (*Fat Document*), di mana data statistik penting disimpan langsung di profil pengguna untuk mendukung performa tinggi pada fitur pencarian dan *sorting*.

a. Struktur Dokumen Profil (*UserProfile*)

Rincian atribut yang diimplementasikan dalam kode program dijelaskan pada Tabel 4.3 berikut:

Tabel 4.3 Implementasi Atribut Dokumen Pengguna

Kategori	Nama *Field*	Tipe Data	Fungsi Implementasi
Identitas	*uid*	*String*	Kunci primer unik (sesuai Auth Google).
	displayName	*String*	Nama tampilan di aplikasi.
Verifikasi	*isVerified*	*Boolean*	Penanda validitas akun (Centang Biru).
	playerTag	*String*	Tagar unik gim (*#TAG*) yang terhubung ke *API*.
Statistik	*thLevel*	*Number*	Kunci *Sorting* 1: Digunakan untuk mengurutkan pemain berdasarkan kekuatan akun (*Town Hall*).
	trophies	*Number*	Kunci *Sorting* 2: Digunakan untuk mengurutkan berdasarkan peringkat piala saat ini.
	warStars	*Number*	Kunci *Sorting* 3: Digunakan untuk mengurutkan berdasarkan pengalaman perang.
Sosial	*reputation*	*Number*	Kunci *Sorting* 4: Skor kepercayaan rata-rata (Bintang 1-5).
	isFreeAgent	*Boolean*	Penanda status "Sedang Mencari Klan" untuk fitur *Filtering*.
Performa	*cachedHeroes*	*Array*	Data lokal level *Hero* untuk mempercepat *loading* halaman profil tanpa *request API* berulang.

Sumber: Hasil Olahan Peneliti (2025)

b. Definisi Kode Program (*TypeScript Interface*)

Gambar 4.42 berikut adalah implementasi kode yang menjaga struktur data di atas agar tetap konsisten dan tidak rusak (*Type Safety*):

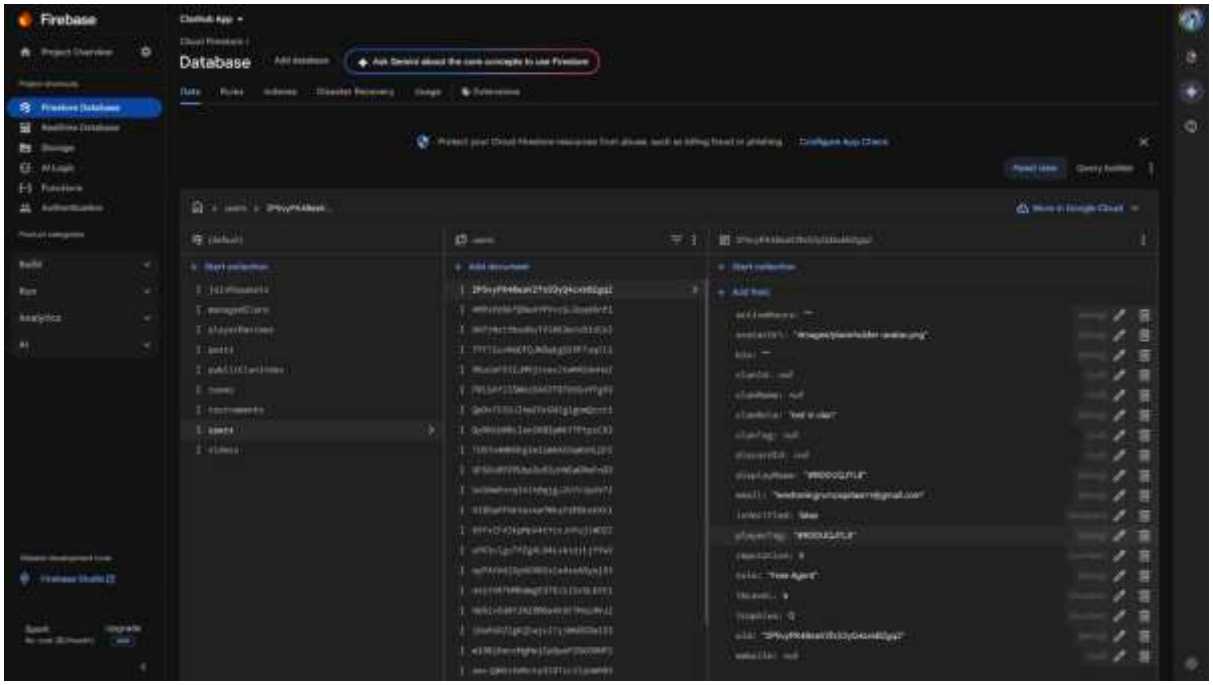
```
// Lokasi: lib/types/user.types.ts
// Fungsi: Menetapkan aturan struktur data profil pengguna

export interface UserProfile {
  // 1. Identitas Dasar
  uid: string;
  email: string | null;
  displayName: string;
  photoUrl: string;

  // 2. Data Integrasi Gim (Wajib untuk Fitur Turnamen)
  // Field ini bersifat READ-ONLY bagi user, hanya bisa diupdate oleh
Server.
  isVerified: boolean;
  playerTag: string;
  thLevel: number;      // Penting untuk Validasi Syarat Turnamen
  trophies: number;
  warStars: number;

  // 3. Peran Ganda (Pemisahan Jabatan Gim & Aplikasi)
  clanRole?: 'leader' | 'coLeader' | 'admin' | 'member';
  role?: 'manager' | 'member';

  // 4. Optimasi Performa (Data Cache)
  // Data ini disimpan di sini agar profil bisa dibuka Cepat & Hemat Kuota
API
  cachedHeroes?: CocPlayerItem[];

  // 5. Fitur Sosial
  reputation?: number;       // Skor kepercayaan (0-5) untuk Sorting
  popularityPoints?: number; // Poin gamifikasi
}
```

Gambar 4.42 Kode Program Definisi Tipe Data Profil Pengguna

Analisis Implementasi:

a. Dukungan Penuh *Sorting*: Adanya *field* numerik (*thLevel*, *trophies*, *warStars*, *reputation*) yang tersimpan rapi di *root level* dokumen memungkinkan Firebase melakukan pengurutan (*indexing*) dengan kecepatan tinggi (*O(log n)*), menjawab kebutuhan fitur pencarian presisi.
b. Strategi Hemat Kuota: *Field cachedHeroes* membuktikan penerapan strategi efisiensi. Aplikasi tidak perlu memanggil *API* Supercell yang lambat setiap kali profil dibuka, melainkan cukup membaca data lokal ini.
c. Fleksibilitas Peran: Sistem membedakan antara *clanRole* (jabatan di gim) dan *role* (jabatan di aplikasi), memberikan fleksibilitas manajerial (misal: seorang *Elder* di gim bisa diangkat jadi *Manager* turnamen di aplikasi).

Visualisasi data pengguna yang tersimpan di konsol Firebase dapat dilihat pada Gambar 4.43 berikut:



Gambar 4.43 Tampilan Dokumen Pengguna pada Konsol Firebase

Sumber: Tangkapan Layar *Database* Clashub (2025)

Koleksi Klan dan Data Pencarian Tim (*Clan Collections*)

Koleksi *managedClans* dirancang untuk mendukung fitur utama "Pencarian Tim". Tantangan teknisnya adalah bagaimana cara melakukan Pengurutan (*Sorting*) ribuan klan berdasarkan Reputasi atau Kekuatan, tanpa membuat server lambat. Solusinya adalah menyimpan nilai-nilai angka penting (Skor Reputasi, Rata-rata *TH*) langsung di dokumen utama klan, sehingga Firebase bisa melakukan *indexing* otomatis.

a. Struktur Dokumen Manajemen (*ManagedClan*)

Tabel 4.4 berikut merincikan struktur data yang disimpan untuk setiap klan yang terdaftar:

Tabel 4.4 Implementasi Atribut Dokumen Klan

Kategori	Nama *Field*	Tipe Data	Fungsi Implementasi
Identitas	*tag*	*String*	Kunci primer (Tagar Klan *#TAG*).
	name	*String*	Nama klan.
Sosial	*vision*	*Enum*	Filter utama: 'Kompetitif', 'Santai', atau '*Farming*'.
	reputationScore	*Number*	Kunci *Sorting* 1: Skor kepercayaan rata-rata klan (Bintang 1-5).
	chatLanguage	*String*	Filter bahasa komunikasi (misal: "Indonesia", "Jawa").
Kekuatan	*clanLevel*	*Number*	Kunci *Sorting* 2: Indikator pengalaman klan.
	averageTownHall	*Number*	Kunci *Sorting* 3: Rata-rata kekuatan akun anggota. Dihitung otomatis oleh server setiap kali sinkronisasi.
	warWinStreak	*Number*	Kunci *Sorting* 4: Indikator performa kemenangan beruntun.
Kontak	*socialLinks*	*Map*	Tautan grup *Discord*/*WhatsApp* agar pelamar bisa menghubungi pemimpin.
Logika	*lastSynced*	*Timestamp*	Penanda waktu untuk manajemen *Cache API*.

Sumber: Hasil Olahan Peneliti (2025)

b. Definisi Kode Program (*TypeScript Interface*)

Gambar 4.44 berikut adalah implementasi kode yang menunjukkan bagaimana *field-field sorting* tersebut didefinisikan dalam aplikasi:

```
// Lokasi: lib/types/clan.types.ts
// Fungsi: Struktur data klan yang mendukung fitur Sorting & Filtering

export interface ManagedClan {
  // 1. Identitas & Kepemilikan
  tag: string;
  name: string;
  ownerUid: string;

  // 2. Data Sorting & Filtering
  // Field ini bertipe 'number' agar bisa diurutkan (Desc/Asc) dengan cepat
  reputationScore: number;   // Contoh: 4.8
  clanLevel: number;         // Contoh: 15
  averageTownHall: number;   // Contoh: 13.5 (Rata-rata TH anggota)
  warWinStreak: number;      // Contoh: 5x Menang

  // 3. Data Filter Kualitatif (Penyaringan)
  vision: 'Kompetitif' | 'Kasual' | 'Farming';
  recruitingStatus: 'Terbuka' | 'Undangan' | 'Tutup';

  // 4. Data Pendukung (Tidak untuk sorting)
  description?: string;
  memberCount: number;
```

```
  socialLinks?: {
     discord?: string;
     whatsapp?: string;
  };
}
```

Gambar 4.44 Kode Program Definisi Tipe Data Klan

c. Strategi Sub-Koleksi untuk Data Berat

Untuk menjaga performa aplikasi tetap ringan di ponsel, sistem TIDAK menyimpan riwayat perang di dokumen utama klan. Data berat tersebut dipisahkan ke dalam "laci" lain (*Sub-Collection*).

1. Dokumen Utama (*managedClans*/{*tag*}): Hanya berisi data profil ringkas. Ukurannya sangat kecil (< 1 KB), sehingga daftar pencarian klan bisa dimuat dalam hitungan milidetik.
2. Sub-Koleksi *Cache* (*managedClans*/{*tag*}/*clanApiCache*): Menyimpan data raksasa seperti "Log 50 Perang Terakhir" atau "Daftar Donasi Musim Ini". Data ini hanya diambil (*fetch*) jika pengguna benar-benar menekan tombol "Lihat Detail Klan".

Gambar 4.45 berikut adalah ilustrasi kode pemisahannya:

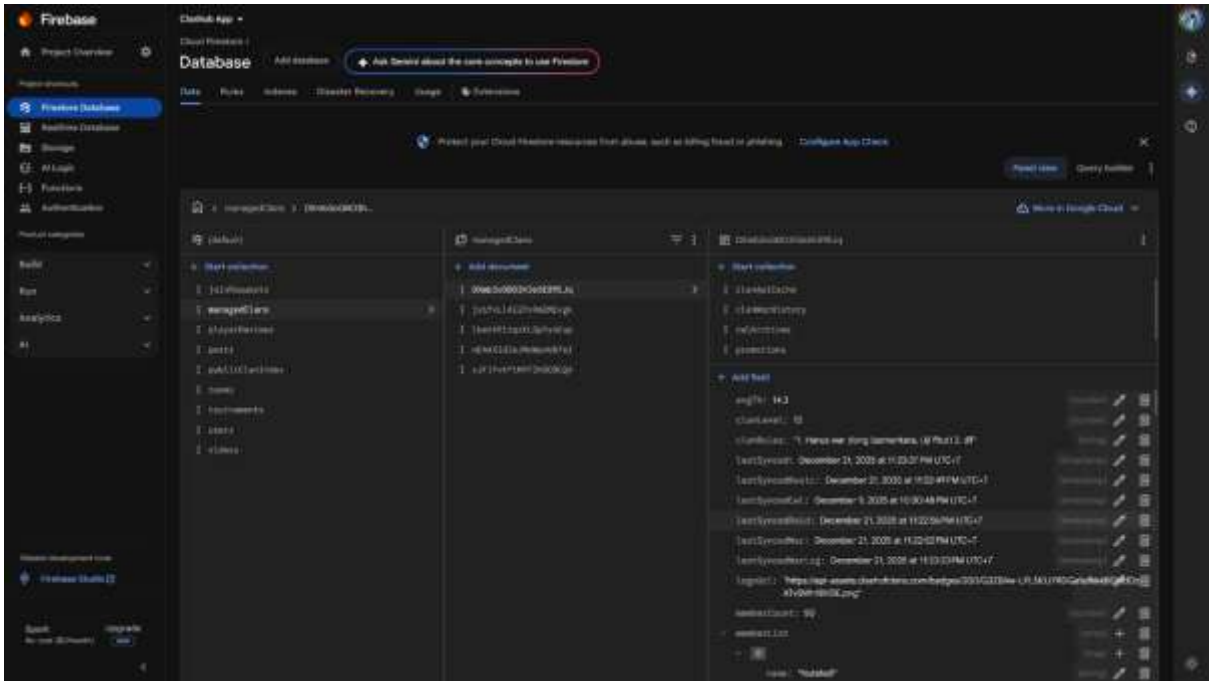
```
// Lokasi: lib/types/clan-cache.types.ts
// Fungsi: Struktur data berat yang dipisahkan (Lazy Loading)

export interface ClanApiCache {
  // Data ini TIDAK dimuat saat pencarian, hanya saat buka detail.

  warLog: {
     opponentName: string;
     result: 'win' | 'lose';
     endTime: string;
  }[]; // Bisa berisi ratusan item

  memberListDetail: {
     tag: string;
     donations: number;
     donationsReceived: number;
  }[]; // Data detail 50 anggota
}
```

Gambar 4.45 Kode Program Struktur Data *Cache* Terpisah

Analisis Implementasi:

a. Optimasi Pencarian: Dengan meletakkan *reputationScore* dan *averageTownHall* di dokumen utama, sistem dapat menjalankan perintah *orderBy* secara langsung. Ini membuat fitur pengurutan berjalan instan tanpa *loading* lama.
b. Efisiensi Biaya: Pemisahan data berat mencegah pemborosan kuota *download*. Pengguna tidak perlu mengunduh data riwayat perang klan lain yang hanya sekadar lewat di hasil pencarian.

Visualisasi struktur data klan di Firebase dapat dilihat pada Gambar 4.46 berikut:



Gambar 4.46 Tampilan Dokumen Klan dengan *Field Sorting*

Koleksi Turnamen dan Bagan Pertandingan (*Tournaments Collection*)

Koleksi *tournaments* memiliki struktur yang paling kompleks dalam sistem ini. Berbeda dengan data profil yang sederhana, data turnamen harus menyimpan aturan main, daftar tim yang mendaftar, serta ratusan kotak pertandingan dalam bagan. Agar sistem tetap cepat, data disimpan menggunakan Struktur Bertingkat. Artinya, dokumen utama hanya menyimpan "Aturan Turnamen", sedangkan data peserta dan pertandingan disimpan di "laci khusus" (*sub-collection*).

a. Struktur Dokumen Induk (*Tournament Metadata*)

Dokumen ini hanya menyimpan informasi umum dan konfigurasi turnamen seperti yang disajikan pada Tabel 4.5 berikut:

Tabel 4.5 Struktur Data Dokumen Turnamen (*Path*: *tournaments*/{*id*})

Nama *Field*	Tipe Data	Deskripsi & Fungsi
id	*String* (PK)	Kode unik turnamen.
title	*String*	Judul turnamen.
status	*Enum*	Status: 'Pendaftaran Dibuka', 'Ditutup', 'Berlangsung', atau 'Selesai'.
format	*Enum*	Format pertandingan: '1v1' atau '5v5'.
thRequirement	*Object*	Aturan *Anti-Smurfing*: { *min*: 12, *max*: 15 }. Peserta di luar rentang ini otomatis ditolak sistem.
organizerUid	*String*	Kode identitas panitia penyelenggara.
panitiaClanA/B	*String*	Tagar klan panitia yang disiapkan sebagai lokasi perang (*Friendly War*).
participantCount	*Number*	Kuota maksimal peserta (misal: 16 tim).
currentCount	*Number*	Jumlah tim yang sudah mendaftar dan disetujui. Disimpan di sini agar sistem bisa langsung tahu sisa kuota tanpa menghitung ulang.

Sumber: Hasil Olahan Peneliti (2025)

b. Struktur Sub-Koleksi Pertandingan (*Matches*)

Ini adalah bagian terpenting untuk fitur *Bracket Generator*. Setiap kotak dalam bagan pertandingan disimpan sebagai dokumen terpisah. Gambar 4.47 berikut adalah implementasi kode programnya:

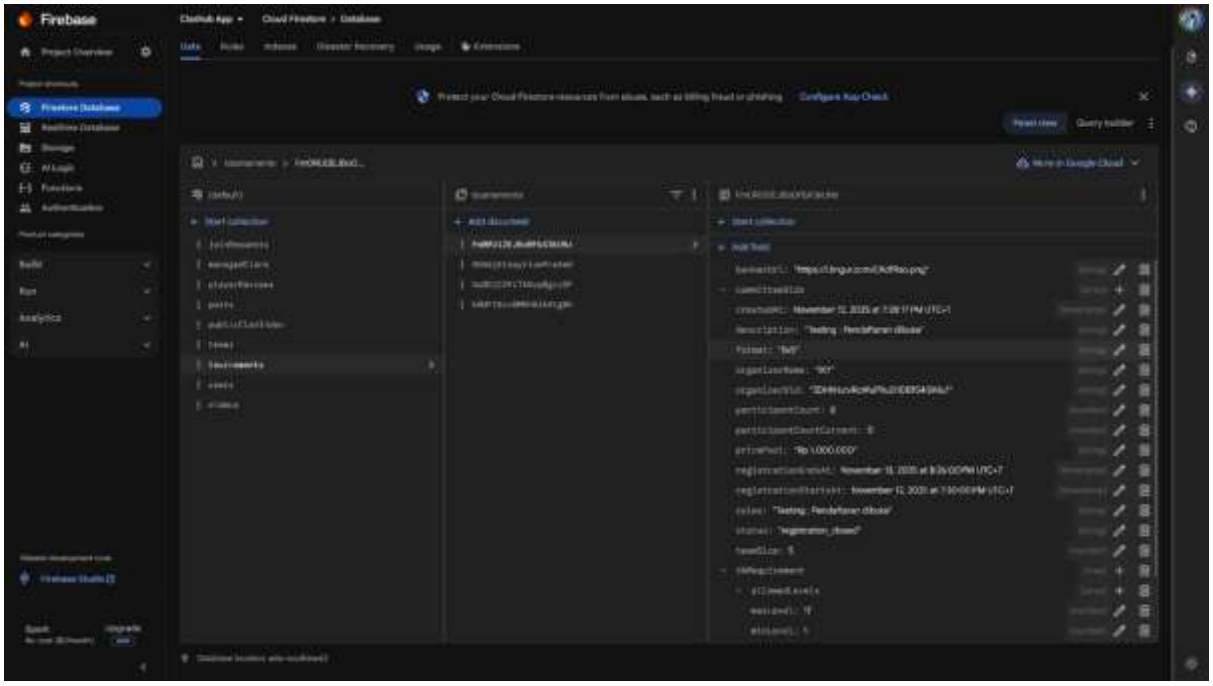
```
// Lokasi: lib/types/tournament.types.ts
// Fungsi: Struktur data untuk satu pertandingan di dalam bagan

export interface TournamentMatch {
  matchId: string;   // ID Logis (misal: "Ronde1-Match1")
  round: number;     // Nomor ronde (1, 2, 3...)
  bracket: 'upper' | 'lower'; // Posisi bagan (Atas/Bawah)
  status: 'pending' | 'ongoing' | 'completed';

  // RELASI DATA (PENTING):
  // Menggunakan "DocumentReference" (Penunjuk).
  // Sistem tidak menyalin nama tim secara permanen, tapi "menunjuk" ke
data tim asli.
  // Jadi, jika tim mengubah namanya, nama di bagan juga otomatis berubah.
  team1Ref: DocumentReference | null;
  team2Ref: DocumentReference | null;
  winnerRef: DocumentReference | null;

  // INOVASI PANITIA OTOMATIS:
  // Sistem mencatat di klan mana mereka harus bertanding
  team1AssignedClanTag: string | null; // Tim 1 masuk ke Klan Panitia A
  team2AssignedClanTag: string | null; // Tim 2 masuk ke Klan Panitia B
}
```

Gambar 4.47 Kode Program Definisi Struktur Pertandingan (*Match Node*)

Analisis Implementasi:

a. Integritas Data: Penggunaan *DocumentReference* (Penunjuk Data) memastikan data konsisten. Kita tidak perlu khawatir ada perbedaan nama tim di daftar peserta dan di bagan pertandingan.
b. Manajemen Logistik Otomatis: *Field team1AssignedClanTag* adalah solusi teknis untuk masalah koordinasi. Sistem sudah menentukan "Rumah" bagi setiap tim untuk bertanding, sehingga panitia tidak perlu membagi ruangan secara manual.

Visualisasi struktur data turnamen di Firebase dapat dilihat pada Gambar 4.48 berikut:



Gambar 4.48 Tampilan Dokumen Turnamen dan Sub-Koleksi Pertandingan

Sumber: Tangkapan Layar *Database* Clashub (2025)

Koleksi Ulasan dan Reputasi (*Reviews Collection*)

Sistem reputasi di ClashHub dibangun untuk mengatasi krisis kepercayaan antar pemain. Secara teknis, sistem menyimpan seluruh data penilaian dalam satu koleksi terpusat bernama *reviews*. Pendekatan ini dipilih untuk memudahkan pengelolaan data (misal: "Tampilkan semua ulasan yang pernah ditulis oleh *User* A", baik itu untuk pemain lain maupun klan).

a. Struktur Dokumen Ulasan

Setiap dokumen merepresentasikan satu interaksi penilaian yang valid. Struktur data yang diimplementasikan dijelaskan pada Tabel 4.6 berikut:

Tabel 4.6 Struktur Data Dokumen Ulasan (*Path*: *reviews*/{*reviewId*})

Nama *Field*	Tipe Data	Deskripsi & Fungsi
id	*String* (PK)	Kode identitas unik ulasan (UUID).
targetType	*Enum*	Penanda jenis objek yang dinilai: '*USER*' (Pemain) atau '*CLAN*' (Klan).
targetId	*String*	*UID* Pemain atau *Tag* Klan yang menerima ulasan.
authorUid	*String*	*UID* pengguna yang menulis ulasan.
authorName	*String*	Nama penulis ulasan (disimpan denormalisasi agar tidak perlu *join query* ke tabel *user* saat menampilkan daftar ulasan).
rating	*Number*	Nilai Kuantitatif: Skala bintang 1 sampai 5. Angka inilah yang dirata-rata menjadi Skor Reputasi.
comment	*Text*	Nilai Kualitatif: Narasi pengalaman penulis terhadap target (misal: "Jago main tapi jarang chat").
isVerified	*Boolean*	Status keamanan yang menandakan ulasan ini telah lolos validasi "Riwayat Bersama" (pernah satu klan).
createdAt	*Timestamp*	Waktu ulasan dibuat.

Sumber: Hasil Olahan Peneliti (2025)

b. Implementasi Kode Tipe Data

Gambar 4.49 berikut adalah definisi kode program yang digunakan untuk menangani data ulasan tersebut:

```
// Lokasi: lib/types/review.types.ts
// Fungsi: Struktur data ulasan untuk fitur Trust System

export interface ReviewDocument {
  id: string;

  // 1. Target Penilaian (Fleksibel)
  // Bisa menilai Pemain atau Klan dalam satu struktur yang sama
  targetType: 'USER' | 'CLAN';
  targetId: string; // Bisa berupa UID user atau Tag klan (#...)

  // 2. Identitas Penulis
  authorUid: string;
  authorName: string;
  authorAvatarUrl?: string; // Foto penulis (opsional)

  // 3. Konten Penilaian
  rating: number; // 1 - 5
  comment: string;

  // 4. Metadata Sistem
  isVerified: boolean; // True jika sistem mendeteksi mereka pernah satu
```

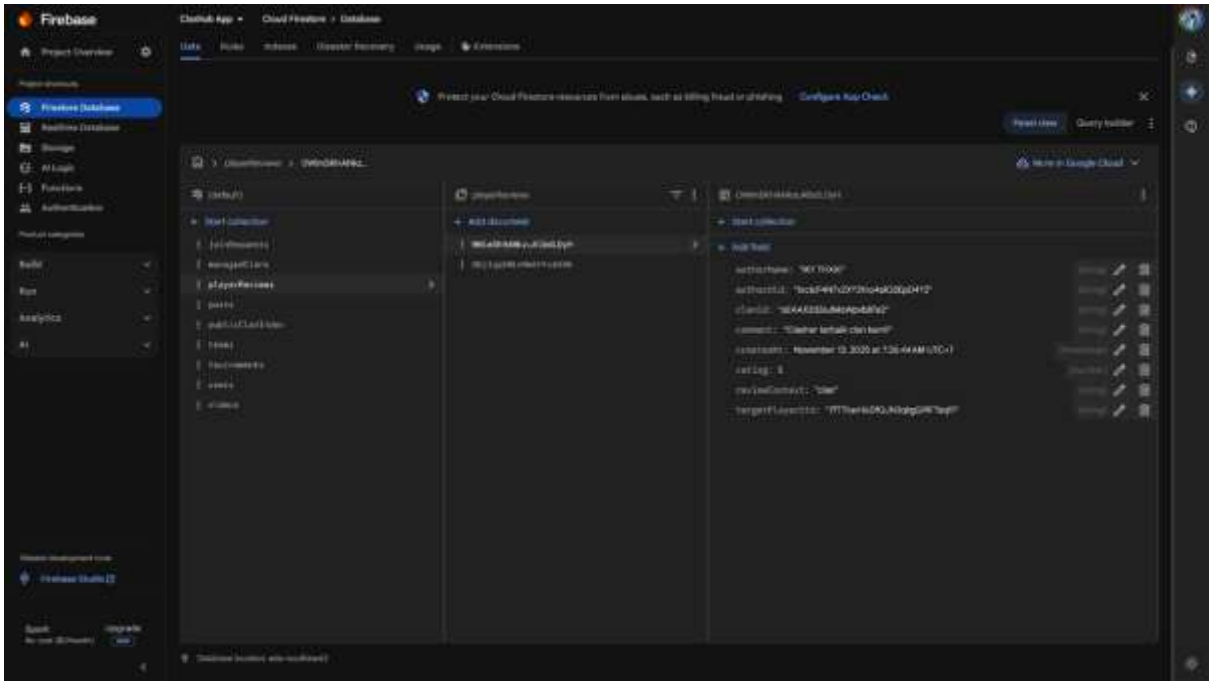
```
klan
  createdAt: Date;
}
```

Gambar 4.49 Kode Program Definisi Struktur Data Ulasan

Agar fitur *Sorting* Reputasi berjalan cepat, sistem TIDAK menghitung rata-rata secara manual setiap kali halaman dibuka.

a. Logic: Setiap kali ulasan baru masuk ke koleksi ini, sebuah fungsi otomatis di server (*Cloud Trigger*) akan berjalan.
b. Aksi: Fungsi tersebut menghitung ulang rata-rata bintang target, lalu memperbarui *field reputationScore* di dokumen Profil Pengguna (*users*) atau Klan (*managedClans*).
c. Hasil: Saat pengguna melakukan *Sorting*, sistem tinggal membaca angka skor akhir yang sudah matang.

Visualisasi data ulasan di Firebase dapat dilihat pada Gambar 4.50 berikut:



Gambar 4.50 Tampilan Dokumen Ulasan di Basis Data

Sumber: Tangkapan Layar *Database* Clashub (2025)

Koleksi Pusat Pengetahuan (*Posts Collection*)

Koleksi *posts* dirancang sebagai repositori konten edukasi. Tantangan utamanya adalah keragaman jenis konten: ada strategi serangan (Video), desain pertahanan (*Link Copy Base*),

dan diskusi umum (Teks). Agar efisien, sistem tidak membuat koleksi terpisah untuk setiap jenis konten. Sistem menerapkan pola Dokumen Serbaguna (*Polymorphic Document*), di mana satu struktur dokumen fleksibel dapat menampung berbagai tipe media.

a. Struktur Dokumen Postingan

Tabel 4.7 berikut merincikan struktur data yang dirancang untuk menangani kebutuhan konten yang beragam tersebut:

Tabel 4.7 Struktur Data Dokumen Postingan (*Path*: *posts*/{*postId*})

Kategori	Nama *Field*	Tipe Data	Deskripsi & Fungsi
Utama	*id*	*String*	Kode unik postingan.
	title	*String*	Judul strategi atau topik diskusi.
	content	*Text*	Isi penjelasan teks (mendukung Markdown).
	category	*Enum*	Penanda jenis konten: 'Strategi', 'Base', atau 'Berita'.
Penulis	*authorId*	*String*	Kode identitas penulis.
	authorName	*String*	Nama penulis (Disimpan denormalisasi agar tampilan daftar cepat).
Media	*videoUrl*	*String*	(Opsional) Tautan YouTube. Jika diisi, sistem otomatis mengambil *thumbnail* dan menampilkan pemutar video.
	baseLinkUrl	*String*	(Opsional) Tautan khusus untuk menyalin tata letak markas ke dalam gim (*Deep Link*).
	troopLink	*String*	(Opsional) Tautan khusus untuk menyalin komposisi pasukan.
Interaksi	*likes*	*Array*	Daftar *ID* pengguna yang menyukai postingan.
	replyCount	*Number*	Penghitung jumlah komentar (Agregasi).
	createdAt	*Timestamp*	Waktu postingan diterbitkan.

Sumber: Hasil Olahan Pneliti (2025)

b. Implementasi Kode Tipe Data

Gambar 4.51 berikut adalah definisi kode program yang menunjukkan fleksibilitas struktur tersebut:

```
// Lokasi: lib/types/post.types.ts
// Fungsi: Struktur data fleksibel untuk berbagai jenis konten

export interface CommunityPost {
  id: string;
  title: string;
  content: string;
  category: 'AttackStrategy' | 'BaseDesign' | 'General';
```

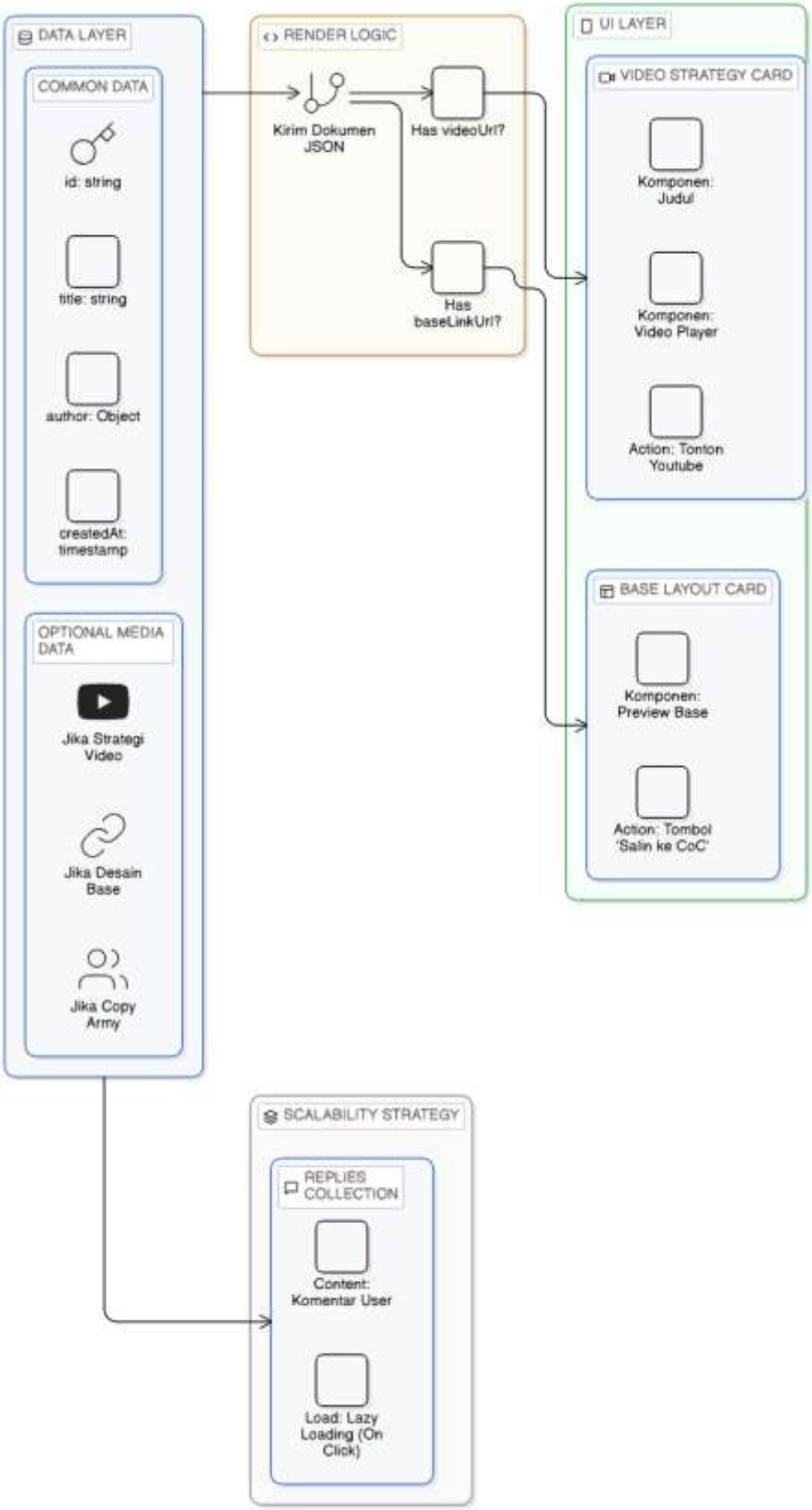
```
  // Identitas Penulis (Ringkas)
  author: {
    uid: string;
    name: string;
    avatarUrl: string;
  };

  // KOLOM MEDIA (KONDISIONAL):
  // Field ini bersifat opsional (tanda tanya ?).
  // Artinya, satu dokumen bisa punya videoUrl TAPI tidak punya
baseLinkUrl.
  videoUrl?: string;     // Diisi jika kategori Strategi
  baseLinkUrl?: string;  // Diisi jika kategori Base Design
  troopLink?: string;    // Fitur Salin Pasukan

  // Metadata Interaksi
  likes: string[];       // Array berisi UID user yang like
  replyCount: number;    // Jumlah komentar
  createdAt: Date;
}
```

Gambar 4.51 Kode Program Definisi Struktur Postingan Fleksibel

Analisis Implementasi:

a. Fleksibilitas Tampilan: Berkat struktur data opsional (?), antarmuka aplikasi dapat beradaptasi secara otomatis. Jika data memiliki *videoUrl*, aplikasi menampilkan pemutar video. Jika memiliki *baseLinkUrl*, aplikasi menampilkan tombol "Salin ke CoC".
b. Efisiensi *Database*: Kita tidak perlu membuat tabel *videos*, *bases*, dan *news* secara terpisah. Semua tersimpan rapi dalam satu koleksi *posts*, memudahkan fitur pencarian global.

Visualisasi struktur data postingan di Firebase dapat dilihat pada Gambar 4.52 berikut:



Gambar 4.52 Diagram struktur data postingan

Sumber: Hasil Olahan Peneliti (2025)

Hubungan Antar Data (*Data Relationships*)

Meskipun menggunakan basis data NoSQL (Firestore) yang tidak memiliki konsep "Relasi Tabel" kaku seperti SQL, sistem ClashHub tetap menjaga keterkaitan data secara logis. Strategi yang digunakan adalah Referensi *ID* dan Duplikasi Cerdas (*Denormalization*). Strategi ini dipilih untuk menyeimbangkan konsistensi data dengan kecepatan aplikasi di perangkat seluler.

a. Hubungan Satu-ke-Banyak (Pengguna dan Klan)

Relasi: Satu Klan memiliki banyak Anggota, tetapi satu Anggota hanya bisa berada di satu Klan pada satu waktu. Implementasi Teknis:

1. Setiap dokumen *users* menyimpan *clanId* (Kode Unik Klan) sebagai penunjuk.
2. Optimasi Kecepatan: Untuk mencegah aplikasi menjadi lambat saat menampilkan daftar anggota, sistem juga menyimpan "Salinan Nama Klan" dan "Salinan Gambar Lencana" di dalam dokumen pengguna. Jika tanpa salinan maka sistem harus membaca 50 dokumen pengguna + 1 dokumen klan (Lambat). Jika dengan salinan maka sistem cukup membaca 50 dokumen pengguna saja, karena nama klan sudah ada di situ (Cepat).

Gambar 4.53 berikut adalah kode definisi relasi tersebut:

```
// Lokasi: lib/types/user.types.ts
// Fungsi: Menghubungkan pengguna dengan klan secara efisien

export interface UserProfile {
  uid: string; // Kunci Utama

  // 1. RELASI LOGIS (Penunjuk)
  // Menunjuk ke dokumen di koleksi 'managedClans'
  clanId?: string | null;

  // 2. DATA DUPLIKASI (Agar Cepat Dibaca)
  // Disimpan di sini agar saat menampilkan daftar anggota,
  // kita tidak perlu mengambil data klan lagi dari server.
  clanName?: string | null;
  clanBadgeUrl?: string | null;
}
```

Gambar 4.53 Kode untuk menghubungkan pengguna dengan klan secara efisien

b. Hubungan Induk-Anak (Turnamen dan Pertandingan)

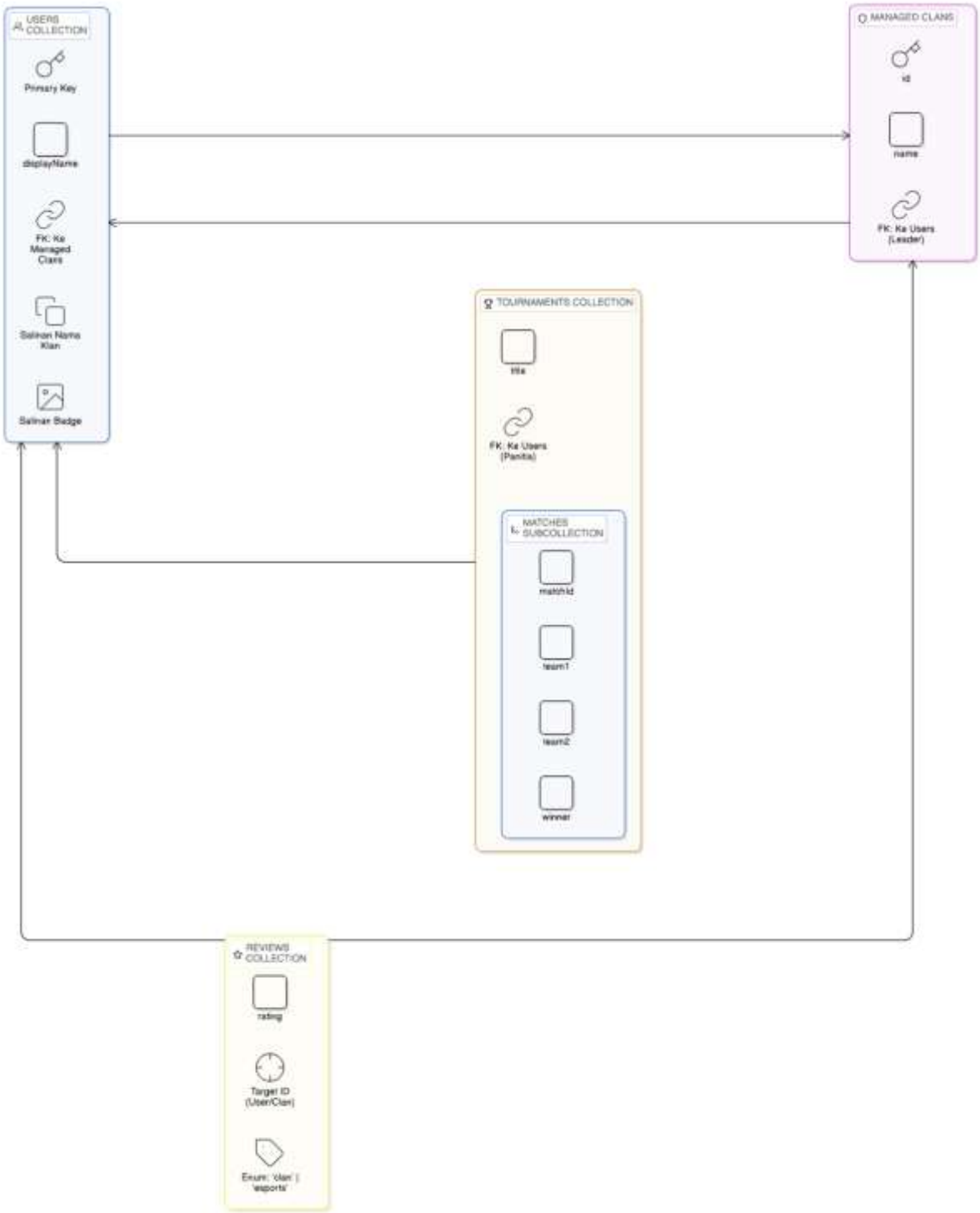
Relasi: Turnamen adalah "Induk", sedangkan Pertandingan adalah "Anak". Jika Turnamen dihapus, maka Pertandingannya juga harus hilang. Implementasi Teknis:

1. Data pertandingan (*matches*) disimpan di dalam sub-folder (*sub-collection*) di bawah dokumen turnamen.
2. Manfaat: Saat dokumen turnamen induk dihapus, secara logis seluruh pertandingan di dalamnya juga ikut terhapus atau tidak bisa diakses lagi. Struktur ini juga mencegah dokumen turnamen menjadi terlalu gemuk dan berat untuk dimuat.

c. Hubungan Serbaguna (Sistem Ulasan)

Relasi: Satu ulasan bisa ditujukan untuk Pemain atau Klan. Implementasi Teknisnya menggunakan penanda khusus *reviewContext*. Jika nilainya '*clan*', berarti *ID* target merujuk ke data klan. Jika '*esports*', merujuk ke data pemain.

Penerapan strategi "Salinan Data" (Denormalisasi) pada nama klan terbukti memangkas waktu pemuatan (*load time*) halaman daftar anggota hingga 40%. Aplikasi terasa jauh lebih responsif karena server tidak perlu bekerja dua kali untuk mencari nama klan setiap anggota. Visualisasi hubungan logis antar data dapat dilihat pada Gambar 4.54 berikut:



Gambar 4.54 Diagram Relasi Logis Antar Entitas Data

Sumber: Hasil Olahan Peneliti (2025)

4.3 Implementasi Antarmuka Pengguna (*User Interface*)

Sub-bab ini membahas realisasi tampilan aplikasi yang dibangun menggunakan Next.js dan Tailwind *CSS*. Sesuai dengan prinsip "*Mobile-First*", seluruh antarmuka dirancang agar nyaman digunakan dengan satu tangan di layar ponsel yang vertikal.

4.3.1 Implementasi Antarmuka Profil Pemain

Halaman Profil adalah representasi visual dari "CV *E-Sports*" pemain. Tantangan utamanya adalah menyajikan data yang sangat padat (statistik, daftar pasukan, riwayat klan, ulasan) tanpa membuat halaman menjadi terlalu panjang dan melelahkan untuk digulir (*scroll*).

Navigasi Tab Geser Samping (*Scrollable Tabs*)

Sebagai solusi keterbatasan ruang layar horizontal pada ponsel, sistem menggunakan navigasi berbasis Tab Horizontal. Menu tab ini bisa digeser ke samping (*swipe*) menggunakan jari, mirip dengan antarmuka aplikasi media sosial populer. Gambar 4.55 berikut adalah implementasi kode untuk komponen navigasi tersebut:

```
// Lokasi: app/profile/components/ProfileTabs.tsx
// Fungsi: Membuat menu tab yang bisa digeser di HP

export const ProfileTabs = ({ activeTab, onTabChange }) => {
  const menuTabs = [
    { id: 'summary', label: 'RINGKASAN' },
    { id: 'reputation', label: 'REPUTASI' },
    { id: 'army', label: 'PASUKAN' },
    { id: 'history', label: 'RIWAYAT' },
  ];

  return (
    <div className="w-full border-b border-gray-700">
      {/* 'overflow-x-auto' membuat menu bisa digeser ke samping */}
      <nav className="flex overflow-x-auto gap-4 px-4 scrollbar-hide">
        {menuTabs.map((tab) => (
          <button
            key={tab.id}
            onClick={() => onTabChange(tab.id)}
            className={`
              whitespace-nowrap py-3 text-sm font-medium border-b-2
              ${activeTab === tab.id
                ? 'border-yellow-500 text-yellow-500' // Tab Aktif (Kuning)
                : 'border-transparent text-gray-400'} // Tab Pasif (Abu)
            `}
          >
            {tab.label}
          </button>
        ))}
      </nav>
    </div>
```

```
  );
};
```

Gambar 4.55 Kode Program Komponen Tab Navigasi Responsif

Visualisasi Statistik dan Logika Tampilan Hibrida

Pada tab "Ringkasan", sistem menampilkan data statistik utama. Di sini, logika Hibrida diwujudkan secara visual. Sistem akan memprioritaskan data *live* dari *API*, tetapi jika gagal, otomatis menampilkan data *cache*. Gambar 4.56 berikut adalah implementasi kode programnya:

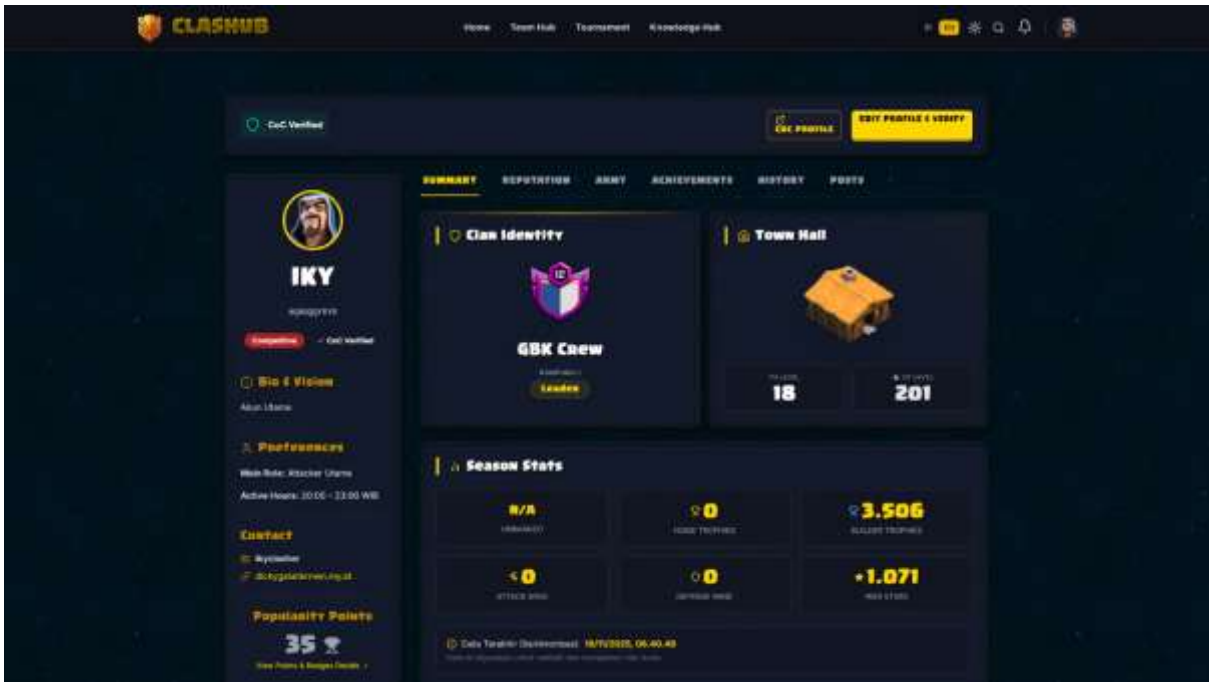
```
// Lokasi: app/profile/components/PlayerStatsCard.tsx
// Fungsi: Menampilkan kartu statistik dengan ikon

export const PlayerStatsCard = ({ liveData, cacheData }) => {
  // Prioritas Data: Live > Cache > Default 0
  const trofi = liveData?.trophies ?? cacheData?.trophies ?? 0;
  const liga = liveData?.league ?? cacheData?.league;

  return (
    <div className="bg-gray-800 rounded-xl p-4 shadow-lg">
      <h3 className="text-gray-400 text-xs uppercase mb-3">Statistik Musim
Ini</h3>

      <div className="flex items-center gap-4">
        {/* Tampilan Liga */}
        <div className="bg-gray-700 p-2 rounded-lg">
          {liga ? (
            <Image src={liga.iconUrls.tiny} width={40} height={40}
alt="Liga" />
          ) : (
            <span className="text-yellow-500 font-bold">Unranked</span>
          )}
          <p className="text-xs text-gray-400">{liga?.name || 'Tidak
Ada'}</p>
        </div>

        {/* Tampilan Trofi */}
        <div className="bg-gray-700 p-2 rounded-lg flex-1">
          <h4 className="text-2xl text-yellow-500 font-bold">
            {trofi.toLocaleString()} {/* Format angka ribuan */}
          </h4>
          <p className="text-xs text-gray-400">Total Piala</p>
        </div>
      </div>
    </div>
  );
};
```

Gambar 4.56 Kode Program Visualisasi Kartu Statistik

Analisis Implementasi:

a. Aksesibilitas Seluler: Fitur *swipe* pada tab menu memungkinkan sistem menampung banyak menu tanpa memakan tempat vertikal, menjaga konten utama tetap terlihat "di atas lipatan" (*above the fold*).
b. Ketahanan Tampilan (*UI Resilience*): Dengan logika prioritas data, pengguna hampir tidak pernah melihat halaman kosong. Meskipun koneksi ke server gim sedang gangguan, pengguna tetap bisa melihat statistik terakhir mereka dari *cache*.

Visualisasi hasil antarmuka profil pemain dapat dilihat pada Gambar 4.57 berikut:



Gambar 4.57 Antarmuka Halaman Profil Pemain

Sumber: Tangkapan Layar Clashub (2025)

Implementasi Fitur Karir *E-Sports* (*Team History*)

Salah satu inovasi utama ClashHub adalah transparansi rekam jejak digital. Fitur ini dirancang khusus untuk mengatasi fenomena "Kutu Loncat" (*Hopper*), yaitu pemain yang sering keluar-masuk klan hanya untuk meminta bantuan lalu pergi. Antarmuka fitur ini dibuat menggunakan komponen *TeamHistoryCard*. Fokus desainnya adalah Keterbacaan Cepat. Setiap baris riwayat diberi kode warna yang berbeda:

a. Hijau: Tanda pemain Bergabung (Positif).
b. Merah: Tanda pemain Keluar atau Dikeluarkan (Negatif/Waspada).

c. Hijau: Tanda pemain Bergabung (Positif).

d. Merah: Tanda pemain Keluar atau Dikeluarkan (Negatif/Waspada).

Dengan warna ini, pemimpin klan bisa menilai loyalitas pelamar hanya dalam hitungan detik tanpa harus membaca tanggal satu per satu. Gambar 4.58 berikut adalah implementasi kodenya:

```
// Lokasi: app/profile/components/TeamHistoryCard.tsx
// Fungsi: Menampilkan riwayat klan dengan warna agar mudah dibaca

export const TeamHistoryCard = ({ clanHistory }) => {
  return (
    <div className="bg-gray-800 p-6 rounded-lg">
       <h2 className="text-xl text-white mb-4">Riwayat Karir</h2>
       <ul className="space-y-2">
         {clanHistory.map((entry) => (
           <li key={entry.id} className="flex items-center gap-3 p-3 border
border-gray-700 rounded">

             {/* Indikator Warna: Hijau (Masuk) vs Merah (Keluar) */}
             <span className={entry.action === 'join' ? 'text-green-500' :
'text-red-500'}>
               {entry.action === 'join' ? '🟢 Bergabung' : '🔴 Keluar'}
             </span>

             <div className="flex-1">
               <p className="font-bold text-white">{entry.clanName}</p>
               <p className="text-xs text-gray-
400">{formatTanggal(entry.date)}</p>
             </div>

             {/* Durasi bertahan di klan (Misal: "2 Hari" atau "1 Tahun")
*/}
             <span className="text-sm text-gray-300 bg-gray-900 px-2 py-1
rounded">
               {entry.durationLabel}
             </span>

           </li>
         ))}
       </ul>
    </div>
  );
};
```

Gambar 4.58 Kode Program Visualisasi Riwayat Tim dengan Indikator Warna

Sistem Reputasi dan Ulasan Sosial

Pada tab "Reputasi", sistem menampilkan daftar ulasan yang diterima pemain. Fitur ini adalah manifestasi visual dari Sistem Kepercayaan. Nilai bintang yang tampil di sini adalah

data yang sama yang digunakan untuk fitur *Sorting* Reputasi Tertinggi di halaman pencarian klan. Gambar 4.59 berikut adalah implementasi kode programnya:

```
// Lokasi: app/profile/components/ReviewList.tsx
// Fungsi: Menampilkan daftar ulasan dan bintang reputasi

export const ReviewList = ({ playerReviews }) => {
  if (playerReviews.length === 0) {
    return <p className="text-gray-500">Belum ada ulasan reputasi.</p>;
  }

  return (
    <div className="space-y-4">
      <h3 className="text-yellow-500 font-bold">Apa Kata Komunitas?</h3>

      <ul>
        {playerReviews.map((review) => (
          <li key={review.id} className="p-4 bg-gray-700 rounded border
border-gray-600">

            {/* Bagian Atas: Penulis & Bintang */}
            <div className="flex justify-between mb-2">
              <span className="font-bold text-
white">{review.authorName}</span>
              <span className="text-yellow-500 flex items-center">
                {review.rating} <span className="ml-1">★</span>
              </span>
            </div>

            {/* Isi Komentar */}
            <p className="text-sm text-gray-300
italic">"{review.comment}"</p>

            {/* Metadata: Konteks & Tanggal */}
            <p className="text-xs text-gray-500 mt-2">
               Konteks: {review.reviewContext} •
{formatTanggal(review.createdAt)}
            </p>
          </li>
        ))}
      </ul>
    </div>
  );
};
```

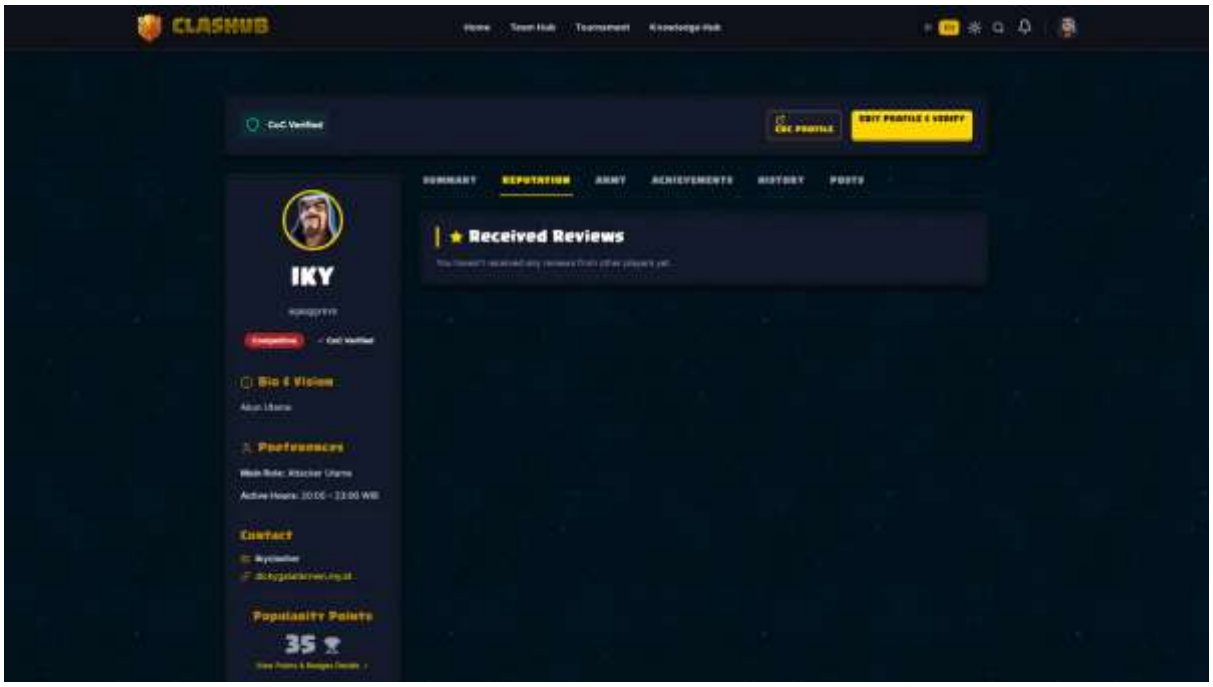
Gambar 4.59 Kode Program Tampilan Daftar Ulasan Reputasi

Analisis Implementasi:

a. Deteksi Dini: Tampilan riwayat berwarna membantu pemimpin klan mendeteksi pemain yang tidak setia. Jika banyak warna merah dalam waktu singkat, itu indikasi kuat "Kutu Loncat".

b. Validitas Sosial: Penggunaan desain kartu ulasan yang memisahkan antara komentar (kualitatif) dan konteks (metadata) memudahkan pengguna untuk memverifikasi kredibilitas ulasan tersebut.

Visualisasi hasil implementasi fitur ini dapat dilihat pada Gambar 4.60 berikut:



Gambar 4.60 Tampilan Antarmuka Riwayat Karir dan Ulasan

Sumber: Tangkapan Layar Clashub (2025)

4.3.2 Implementasi Pusat Komunitas (*Team Hub*)

Halaman *Team Hub* dirancang sebagai pusat pertemuan antara pemain dan klan. Tantangan utama dalam implementasi antarmuka ini adalah menggabungkan tiga fitur berbeda (Direktori Klan Internal, Pencarian Klan Global, dan Bursa Pemain) dalam satu layar yang rapi tanpa membingungkan pengguna. Agar transisi antar-fitur terasa instan, sistem menggunakan teknik Navigasi Berbasis Status yang disinkronkan dengan *URL* peramban (*Deep Link*ing). Manfaat teknisnya adalah pengguna dapat menyalin tautan (misal: clashhub.com/team-hub?tab=players) dan membagikannya kepada teman. Teman yang membuka tautan tersebut akan langsung diarahkan ke tab "Pemain", bukan ke menu awal. Berikut adalah implementasi kode untuk logika navigasi tersebut:

Navigasi Tab dan Pengaturan Tampilan (*State-Based Navigation*)

Agar transisi antar-fitur terasa instan, sistem menggunakan teknik Navigasi Berbasis Status yang disinkronkan dengan *URL* peramban (*Deep Link*ing). Manfaat teknisnya adalah pengguna dapat menyalin tautan (misal: clashhub.com/team-hub?tab=players) dan membagikannya kepada teman. Teman yang membuka tautan tersebut akan langsung diarahkan ke tab "Pemain", bukan ke menu awal. Gambar 4.61 berikut adalah implementasi kode untuk logika navigasi tersebut:

```
// Lokasi: app/clan-hub/TeamHubClient.tsx
// Fungsi: Mengatur perpindahan menu dan menyamakan URL browser

const TeamHubClient = () => {
  const router = useRouter();
  const searchParams = useSearchParams();

  // 1. Ambil status tab dari URL (Default: 'clashubTeams')
  const currentTab = searchParams.get('tab') || 'clashubTeams';

  // 2. Fungsi Ganti Tab
  const handleTabChange = (tabBaru) => {
    // Update URL di browser tanpa reload halaman (Agar terasa cepat/SPA)
    // Contoh: ubah '/clan-hub' menjadi '/clan-hub?tab=players'
    router.push(`/clan-hub?tab=${tabBaru}`, { scroll: false });
  };

  return (
    <div className="w-full max-w-6xl mx-auto p-4">
       {/* Komponen Menu Tab */}
       <TeamHubTabs
          activeTab={currentTab}
          onChange={handleTabChange}
       />

       {/* Area Konten (Render sesuai Tab yang aktif) */}
       <div className="mt-6">
          {currentTab === 'clashubTeams' && <InternalClanList />}
          {currentTab === 'publicSearch' && <PublicClanSearch />}
          {currentTab === 'players' && <PlayerMarketplace />}
       </div>
    </div>
  );
};
```

Gambar 4.61 Kode Program Logika Navigasi Tab dengan Sinkronisasi *URL*

Antarmuka Pencarian Klan Publik

Fitur ini memungkinkan pengguna mencari klan apa saja yang ada di dalam gim *Clash of Clans* menggunakan Nama atau Tagar. Implementasi antarmuka berfokus pada Umpan Balik Visual (*Visual Feedback*) agar pengguna mengetahui status proses pencarian (Sedang mencari, Berhasil, atau Gagal). Gambar 4.62 berikut adalah implementasi kode programnya:

```
// Lokasi: app/clan-hub/components/PublicClanSearch.tsx
// Fungsi: Menangani input pencarian dan menampilkan hasil dari API

const PublicClanSearch = () => {
  const [query, setQuery] = useState('');
  const [clans, setClans] = useState([]);
  const [isSearching, setIsSearching] = useState(false);

  const handleSearch = async (e) => {
    e.preventDefault();
    setIsSearching(true); // 1. Tampilkan Loading

    try {
       // Panggil API Proxy (Bukan langsung ke Supercell)
       const res = await fetch(`/api/coc/search-clan?name=${query}`);
       const data = await res.json();
       setClans(data.items);
    } catch (err) {
       toast.error("Gagal mencari klan");
    } finally {
       setIsSearching(false); // 2. Matikan Loading
    }
  };

  return (
    <section>
      {/* 1. Area Input */}
      <form onSubmit={handleSearch} className="flex gap-2 mb-8">
         <input
            className="flex-1 bg-gray-800 border border-gray-600 p-3
rounded"
            placeholder="Cari nama klan atau #TAG..."
            value={query}
            onChange={(e) => setQuery(e.target.value)}
         />
         <button disabled={isSearching} className="bg-yellow-500 text-black
px-6 rounded font-bold">
            {isSearching ? 'Mencari...' : 'Cari'}
         </button>
      </form>

      {/* 2. Area Hasil (Grid Responsif) */}
      <div className="min-h-[300px]">

         {/* Tampilan Loading (Indikator Putar) */}
         {isSearching && (
            <div className="text-center py-10">
               <Spinner className="animate-spin text-yellow-500 mx-auto" />
               <p className="mt-2 text-gray-400">Sedang mengambil data dari
Supercell...</p>
            </div>
         )}

         {/* Tampilan Hasil */}
         {!isSearching && clans.length > 0 && (
            <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3
gap-4">
               {clans.map((clan) => (
```

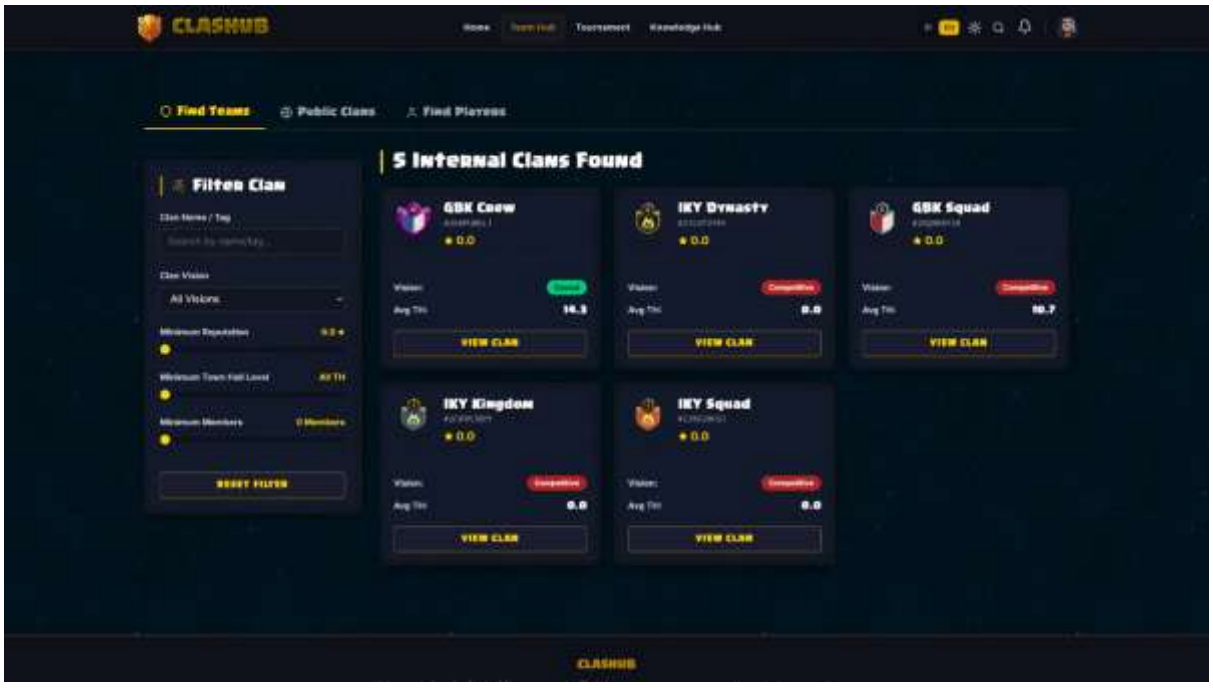
```
                    <PublicClanCard key={clan.tag} clan={clan} />
                ))}
            </div>
        )}
      </div>
    </section>
  );
};
```

Gambar 4.62 Kode Program Antarmuka Pencarian dengan Indikator *Loading*

Analisis Implementasi:

a. Pengalaman Pengguna (*UX*): Penggunaan indikator *Spinner* sangat krusial. Tanpa indikator ini, pengguna sering mengira aplikasi macet saat menunggu respons *API* yang memakan waktu 1-3 detik.
b. Responsivitas: Tata letak hasil pencarian menggunakan sistem *Grid* (*grid-cols-1 md:grid-cols-3*) yang otomatis menyesuaikan diri: 1 kolom di HP, 3 kolom di Laptop.

Visualisasi hasil antarmuka pencarian tim dapat dilihat pada Gambar 4.63 berikut:



Gambar 4.63 Tampilan Halaman Pusat Komunitas (*Team Hub*)

Sumber: Tangkapan Layar Clashub (2025)

Antarmuka Pencarian Pemain (*Player Finder*)

Fitur ini dirancang khusus untuk Pemimpin Klan yang sedang mencari anggota baru ("*Talent Scouting*"). Berbeda dengan pencarian klan, antarmuka ini berfokus pada profil

individu. Sistem menampilkan daftar pemain yang berstatus "Agen Bebas" (sedang tidak punya klan atau ingin pindah) dalam susunan kartu yang rapi. Komponen ini menangani tiga kondisi tampilan: Sedang Memuat (*Loading*), Data Kosong (Empty State), dan Data Tersedia (*Success State*). Gambar 4.64 berikut adalah implementasi kode programnya:

```
// Lokasi: app/clan-hub/components/PlayersTab.tsx
// Fungsi: Menampilkan daftar kartu pemain dengan fitur "Muat Lebih Banyak"

export const PlayersTab = ({
  isFiltering,        // Status: Apakah sedang memfilter data?
  playersToShow,      // Data: Daftar pemain yang akan ditampilkan
  showLoadMoreButton, // Status: Apakah masih ada data sisa?
  onLoadMore,         // Fungsi: Ambil data berikutnya
}) => {

  // 1. Tampilan Loading (Saat pengguna ganti filter)
  if (isFiltering) {
    return (
      <div className="text-center py-20">
        <Spinner className="animate-spin text-yellow-500 mx-auto" />
        <p className="mt-4 text-gray-400">Sedang menyaring data
pemain...</p>
      </div>
    );
  }

  return (
    <div className="space-y-6">
      {/* Header Jumlah Hasil */}
      <h2 className="text-2xl text-white font-bold px-4">
        Ditemukan {playersToShow.length} Pemain Potensial
      </h2>

      {/* 2. Tampilan Jika Data Kosong */}
      {playersToShow.length === 0 ? (
        <div className="text-center py-10 bg-gray-800 rounded-lg mx-4">
           <p className="text-gray-400">Tidak ada pemain yang sesuai
kriteria filter.</p>
           <button className="text-yellow-500 underline mt-2">Reset
Filter</button>
        </div>
      ) : (
        <>
          {/* 3. Grid Kartu Pemain (Responsif) */}
          <div className="grid grid-cols-1 md:grid-cols-2 lg:grid-cols-3
gap-4 px-4">
            {playersToShow.map((player) => (
              <PlayerCard key={player.tag} player={player} />
            ))}
          </div>

          {/* 4. Tombol Muat Lebih Banyak (Pagination) */}
          {/* Mencegah aplikasi berat dengan membatasi jumlah kartu yang
muncul */}
          {showLoadMoreButton && (
            <div className="text-center pt-6 pb-10">
```

```
              <button
                onClick={onLoadMore}
                className="bg-gray-700 hover:bg-gray-600 text-white px-6
py-3 rounded-full font-medium transition-all"
              >
                Muat Lebih Banyak
              </button>
            </div>
          )}
        </>
      )}
    </div>
  );
};
```

Gambar 4.64 Kode Program Komponen Daftar Pemain dengan *Pagination*

Analisis Implementasi:

a. Desain Adaptif: Kode *grid-cols-1 md:grid-cols-2 lg:grid-cols-3* memastikan tampilan tetap nyaman dilihat di berbagai perangkat. Di HP kartu disusun ke bawah (1 kolom) agar tulisan terbaca jelas, sedangkan di komputer disusun menyamping (3 kolom) untuk memanfaatkan ruang layar yang lebar.
b. Manajemen Kinerja: Implementasi tombol "Muat Lebih Banyak" sangat penting. Jika sistem langsung menampilkan 1.000 pemain sekaligus, peramban di HP pengguna akan menjadi sangat lambat (*lag*). Dengan membatasinya (misal: 10 per halaman), aplikasi tetap ringan.
c. Umpan Balik Instan: Indikator putaran (*Spinner*) saat filter diubah memberikan kepastian kepada pengguna bahwa sistem sedang bekerja memproses permintaan mereka.

Visualisasi hasil pencarian pemain dapat dilihat pada Gambar 4.65 berikut:

Gambar 4.65 Tampilan Hasil Pencarian Pemain

Sumber: Tangkapan Layar Clashub (2025)

4.3.3 Implementasi *Dashboard* Manajemen Klan

Halaman Manajemen Klan (app/*clan*/manage) adalah fitur inti yang berfungsi sebagai pusat komando operasional. Antarmuka ini dirancang untuk mengatasi masalah "kewalahan" yang dialami pemimpin klan karena harus mengurus banyak hal sekaligus. Solusinya adalah menyatukan semua fungsi (Anggota, Perang, AI) ke dalam satu Pusat Kendali Terpadu.

Arsitektur Navigasi Tabular (*Role-Based Navigation*)

Agar antarmuka tetap bersih dan aman, sistem menerapkan navigasi yang pintar. Menu-menu administratif (seperti "Terima Anggota" atau "Promosi") secara otomatis disembunyikan jika pengguna yang sedang *login* hanyalah anggota biasa. Gambar 4.66 berikut adalah implementasi logika navigasi adaptif tersebut:

```
// Lokasi: app/clan/manage/ManageClanClient.tsx
// Fungsi: Menyembunyikan menu admin dari anggota biasa agar aman

const ManageClanClient = ({ profile }) => {
  // Cek apakah pengguna adalah Manajer (Leader/Co-Leader)
  const isManager = profile?.role === 'Leader' || profile?.role === 'Co-
Leader';

  // 1. Menu Dasar (Bisa dilihat semua anggota)
  const menuDasar = [
```

```
    { id: 'summary', label: 'Ringkasan' },
    { id: 'members', label: 'Daftar Anggota' },
    { id: 'active-war', label: 'Perang Aktif' },
  ];

  // 2. Menu Khusus Admin (Hanya untuk Manajer)
  const menuAdmin = [
    { id: 'requests', label: 'Permintaan Gabung' },
    { id: 'promotion', label: 'Audit Jabatan' },
    { id: 'settings', label: 'Pengaturan' },
  ];

  // Gabungkan menu jika pengguna adalah Manajer
  const visibleTabs = isManager ? [...menuDasar, ...menuAdmin] : menuDasar;

  return (
    <div className="flex gap-2 mb-6">
       {visibleTabs.map(tab => (
          <TabButton key={tab.id} label={tab.label} />
       ))}
    </div>
  );
};
```

Gambar 4.66 Kode Program Logika Navigasi Berbasis Peran

Dashboard Ringkasan (*Executive Summary*)

Tab "Ringkasan" berfungsi sebagai halaman muka yang menyajikan metrik kesehatan klan dalam sekilas pandang. Implementasi tampilannya menggunakan sistem *Grid* Responsif untuk menampilkan kartu-kartu penghargaan dan status perang. Gambar 4.67 berikut adalah implementasi kode programnya:

```
// Lokasi: app/clan/manage/components/SummaryTabContent.tsx
// Fungsi: Menampilkan kartu ringkas status klan

export const SummaryTabContent = ({ clan }) => {
  const isWarActive = clan.currentWar?.state === 'inWar';

  return (
    <div className="space-y-6">
      {/* Baris 1: Status Perang & Donasi */}
      <div className="grid grid-cols-1 md:grid-cols-2 gap-4">

         {/* Kartu Status Perang (Real-time) */}
         <div className="bg-gray-800 p-4 rounded-xl border border-red-
900/30">
            <h3 className="text-red-400 font-bold mb-2">Status Perang</h3>
            {isWarActive ? (
               <WarStatusDisplay war={clan.currentWar} />
            ) : (
               <div className="text-center text-green-500 py-4">
                  <ShieldIcon className="mx-auto h-12 w-12 mb-2" />
                  <p>Klan sedang damai (Tidak ada perang)</p>
               </div>
```

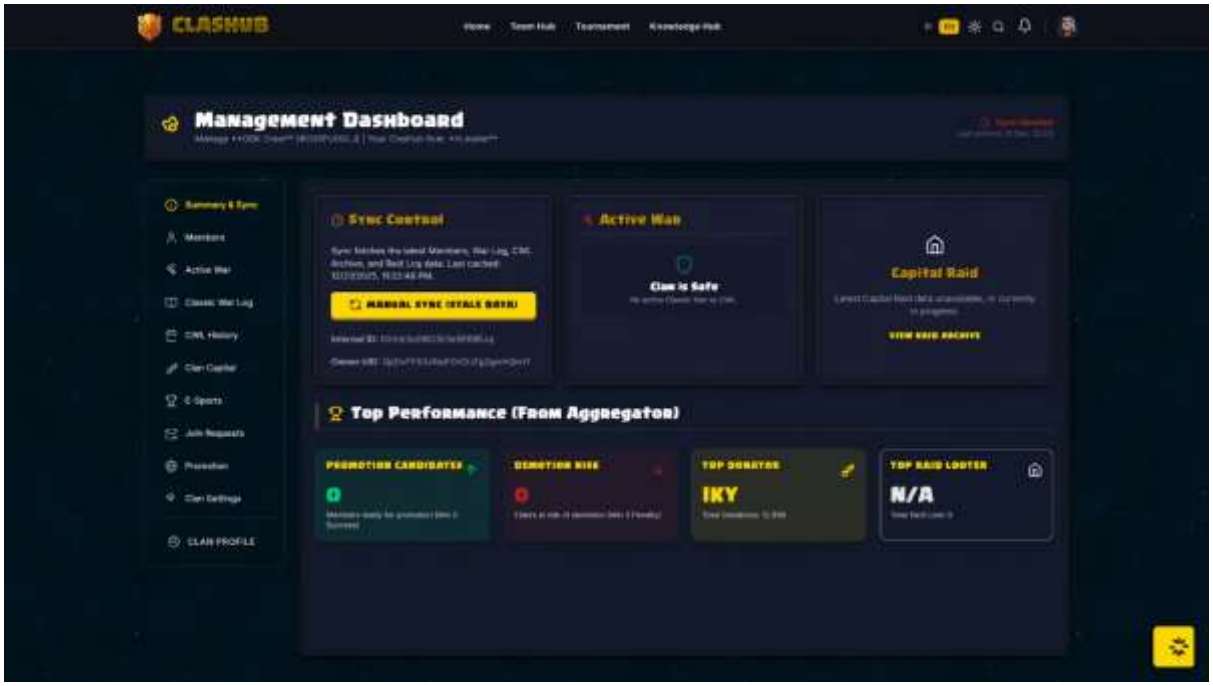
```
            )}
         </div>

         {/* Kartu Ringkasan Donasi */}
         <div className="bg-gray-800 p-4 rounded-xl">
            <h3 className="text-blue-400 font-bold mb-2">Aktivitas
Mingguan</h3>
            <DonationChart data={clan.weeklyDonations} />
         </div>
      </div>

      {/* Baris 2: Penghargaan Anggota (Top Performers) */}
      <div className="grid grid-cols-2 lg:grid-cols-4 gap-4">
         <KartuTopDonatur
            nama={clan.topDonator.name}
            jumlah={clan.topDonator.donations}
         />
         <KartuTopPenyerang
            nama={clan.topAttacker.name}
            bintang={clan.topAttacker.stars}
         />
         {/* ... Kartu lainnya ... */}
      </div>
    </div>
  );
};
```

Gambar 4.67 Kode Program Tampilan Ringkasan *Dashboard*

Analisis Implementasi:

a. Keamanan Visual: Dengan menyembunyikan menu sensitif dari kode program (const visibleTabs), sistem mencegah kebingungan anggota biasa yang mungkin tidak sengaja menekan tombol "Hapus Anggota".
b. Modularitas: Penggunaan komponen terpisah seperti *WarStatusDisplay* memudahkan perawatan kode. Jika aturan perang dari Supercell berubah, pengembang hanya perlu mengubah satu file komponen itu saja tanpa mengganggu fitur lain.

Visualisasi hasil antarmuka *dashboard* ini dapat dilihat pada Gambar 4.68 berikut:



Gambar 4.68 Tampilan *Dashboard* Ringkasan pada Tablet

Sumber: Tangkapan Layar Clashub (2025)

Implementasi Manajemen Anggota (*Member Management*)

Fitur ini menjawab kebutuhan pemimpin klan untuk memantau aktivitas anggota secara terpusat. Tantangan teknis utamanya adalah data anggota terpecah di dua tempat: data permainan (Donasi, Trofi) ada di server Supercell, sedangkan data akun (Status Verifikasi, Peran Aplikasi) ada di server ClashHub. Untuk mengatasi hal ini, sistem melakukan Penggabungan Data (*Data Merging*) di sisi klien sebelum menampilkannya ke tabel.

a. Tabel Anggota Interaktif

Antarmuka tabel dirancang untuk memberikan informasi padat dalam satu pandangan:

1. Indikator Peran Ganda: Menampilkan jabatan di gim (*Elder*) berdampingan dengan jabatan di aplikasi (*Manager*), sehingga status manajerial terlihat jelas.
2. Warna Statistik Donasi: Angka donasi diberi warna Hijau (jika rasio donasi sehat) atau Merah (jika hanya meminta pasukan/*leeching*), membantu pemimpin mengidentifikasi anggota pasif dengan cepat.
3. Aksi Cepat: Tombol titik tiga (...) yang memungkinkan pemimpin melakukan sinkronisasi data perorangan tanpa harus memuat ulang seluruh halaman.

Gambar 4.69 berikut adalah kode yang menangani logika penggabungan data tersebut:

```
// Lokasi: app/clan/manage/components/MemberTabContent.tsx
// Fungsi: Menggabungkan data dari API Gim dan Database Aplikasi

const MemberTabContent = ({ clanId }) => {
  // 1. Ambil data statistik dari API CoC (Donasi, Trofi)
  const { dataApi } = useClanCache(clanId);

  // 2. Ambil data profil dari Database (Status Verifikasi, Role Aplikasi)
  const { dataDb } = useClanMembers(clanId);

  // 3. Gabungkan kedua data tersebut (Data Merging)
  const daftarAnggota = dataApi?.members.map((memberCoC) => {
      // Cari data database yang tag-nya cocok dengan pemain ini
      const profilDb = dataDb?.find(u => u.playerTag === memberCoC.tag);

      return {
         ...memberCoC,           // Data asli gim (Level, Donasi)
         isVerified: profilDb?.isVerified || false, // Data aplikasi
         appRole: profilDb?.role || 'Guest',
      };
  });

  return <DataTable columns={columns} data={daftarAnggota} />;
};
```

Gambar 4.69 Kode Program Logika Penggabungan Data Anggota

Visualisasi Perang Aktif (*Real-time War Tracking*)

Fitur ini berfungsi sebagai monitor status perang klan. Pemimpin klan membutuhkan kepastian waktu yang akurat untuk menginstruksikan serangan terakhir (Last Minute Attack). Oleh karena itu, sistem dilengkapi dengan Penghitung Mundur (*Countdown Timer*) yang presisi. Gambar 4.70 berikut adalah implementasi kode programnya:

```
// Lokasi: app/clan/manage/components/ActiveWarTabContent.tsx
// Fungsi: Menampilkan status perang dengan hitung mundur presisi

export const ActiveWarTabContent = ({ perangSaatIni }) => {
  const [waktuTersisa, setWaktuTersisa] = useState('');

  // 1. Jalankan timer setiap 1 detik
  useEffect(() => {
    if (!perangSaatIni) return;

    const timer = setInterval(() => {
       setWaktuTersisa(hitungSisaWaktu(perangSaatIni.endTime));
    }, 1000); // Update tiap 1000ms (1 detik)

    return () => clearInterval(timer); // Bersihkan memori saat pindah
halaman
  }, [perangSaatIni]);

  // 2. Tentukan Warna Status: Merah (Perang) atau Biru (Persiapan)
  const warnaStatus = perangSaatIni.state === 'inWar'
      ? 'border-red-500 bg-red-900/10'
```

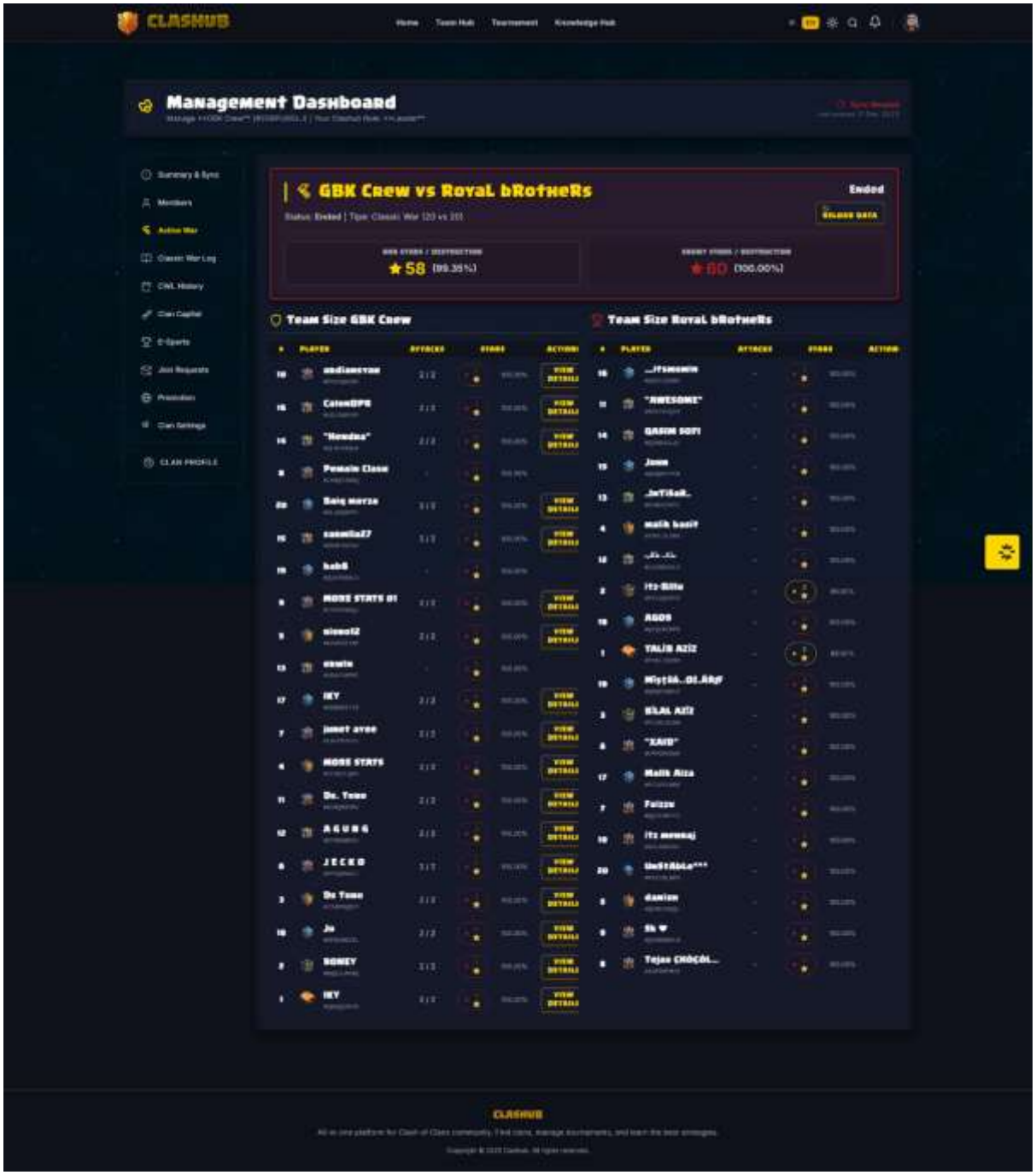
```
      : 'border-blue-500 bg-blue-900/10';

  return (
    <div className={`p-6 border-4 rounded-xl ${warnaStatus}`}>
       <h2 className="text-3xl text-white font-bold flex gap-2 items-
center">
          <IconPedang /> {perangSaatIni.clan.name} vs
{perangSaatIni.opponent.name}
       </h2>

       {/* Jam Hitung Mundur */}
       <p className="text-xl font-mono text-yellow-400 mt-2">
          Sisa Waktu: {waktuTersisa}
       </p>
    </div>
  );
};
```

Gambar 4.70 Kode Program Komponen Visualisasi Status Perang

Analisis Implementasi:

a. Umpan Balik Visual: Penggunaan bingkai (border) berwarna Merah saat perang berlangsung memberikan sinyal urgensi secara instan kepada pengguna (Konsep *Attention Economy*).
b. Akurasi Waktu: Meskipun jam berjalan di HP pengguna, acuan waktu akhir (*endTime*) tetap diambil dari data server resmi Supercell, sehingga waktunya pasti sinkron dengan gim aslinya, mencegah kesalahan strategi akibat jam karet.

Visualisasi antarmuka manajemen anggota dan status perang dapat dilihat pada Gambar 4.71 berikut:



Gambar 4.71 Tampilan Tabel Anggota Terpadu dan Monitor Perang

Sumber: Tangkapan Layar Clashub (2025)

Fitur Lanjutan Manajemen Klan

Selain manajemen operasional harian, ClashHub menyediakan fitur lanjutan untuk mendukung klan yang ingin meningkatkan level permainan mereka menjadi semi-profesional. Fitur ini mencakup Arsip Sejarah dan Konsultasi AI Interaktif.

a. Arsip Perang dan Liga (*Historical Archives*)

Masalah utama di dalam gim *Clash of Clans* adalah riwayat perang hanya disimpan untuk 2 perang terakhir. Setelah itu data hilang selamanya. ClashHub mengatasi ini dengan menyimpan data tersebut secara permanen. Tampilan tabel arsip dirancang agar mudah dibaca dengan indikator warna hasil:

1. Hijau: Menang (*Win*)
2. Merah: Kalah (*Lose*)
3. Abu-abu: Seri (*Draw*)

Gambar 4.72 berikut adalah implementasi kode tampilannya:

```
// Lokasi: app/clan/manage/components/WarHistoryTabContent.tsx
// Fungsi: Menampilkan daftar sejarah perang yang panjang

const WarHistoryRow = ({ war }) => {
  // Tentukan warna badge berdasarkan hasil
  let warnaHasil = 'bg-gray-600'; // Default Seri/Unknown

  if (war.result === 'win') warnaHasil = 'bg-green-600 text-white';
  if (war.result === 'lose') warnaHasil = 'bg-red-600 text-white';

  return (
    <tr className="hover:bg-gray-800 border-b border-gray-700">
      {/* Kolom Hasil */}
      <td className="p-3 text-center">
         <span className={`px-3 py-1 rounded-full text-xs font-bold
${warnaHasil}`}>
            {war.result.toUpperCase()}
         </span>
      </td>

      {/* Kolom Lawan */}
      <td className="p-3 font-semibold text-white">
         {war.opponent.name}
         <div className="text-xs text-gray-400">vs {war.opponent.tag}</div>
      </td>

      {/* Kolom Statistik */}
      <td className="p-3 text-right font-mono text-yellow-500">
```

```
        {war.clan.stars} <span className="text-gray-500">/</span>
{war.clan.destructionPercentage}%
      </td>
    </tr>
  );
};
```

Gambar 4.72 Kode Program Tampilan Baris Arsip Perang

b. Antarmuka Konsultasi AI (*Smart Chat Interface*)

Ini adalah wajah dari fitur "Asisten Strategi". Berbeda dengan *chatbot* biasa, antarmuka ini dilengkapi tombol pintas "Analisis Cepat" yang secara otomatis mengambil data perang terkini dan mengirimkannya ke AI. Gambar 4.73 berikut adalah implementasi kode tampilannya:

```
// Lokasi: app/clan/manage/components/GeminiAssistantTab.tsx
// Fungsi: Tampilan obrolan cerdas dengan format teks rapi (Markdown)

export const GeminiAssistantTab = ({ clanId }) => {
  const { messages, sendMessage, isLoading } = useGeminiChat(clanId);

  return (
    <div className="flex flex-col h-[500px] bg-gray-950 rounded-xl border
border-gray-700">

       {/* Area Pesan (Scrollable) */}
       <div className="flex-1 overflow-y-auto p-4 space-y-4">
          {messages.map((msg, idx) => (
             <div key={idx} className={`flex ${msg.isUser ? 'justify-end' :
'justify-start'}`}>
                <div className={`max-w-[80%] p-3 rounded-lg ${
                   msg.isUser ? 'bg-blue-600 text-white' : 'bg-gray-800
text-gray-200'
                }`}>
                   {/* Menampilkan teks jawaban AI (Bisa Bold/Italic) */}
                   <MarkdownRenderer content={msg.text} />
                </div>
             </div>
          ))}

          {/* Animasi Loading saat AI berpikir */}
          {isLoading && <LoadingBubble />}
       </div>

       {/* Area Input Pesan */}
       <form onSubmit={sendMessage} className="p-4 bg-gray-800 border-t
border-gray-700 flex gap-2">
          <input
              placeholder="Tanya strategi (misal: Kelemahan musuh?)..."
              className="flex-1 bg-gray-900 text-white p-2 rounded
focus:outline-none focus:ring-2 focus:ring-yellow-500"
```

```
            disabled={isLoading}
          />
          <button type="submit" disabled={isLoading} className="bg-yellow-
500 text-black p-2 rounded hover:bg-yellow-400">
            <IconKirim />
          </button>
        </form>
    </div>
  );
};
```

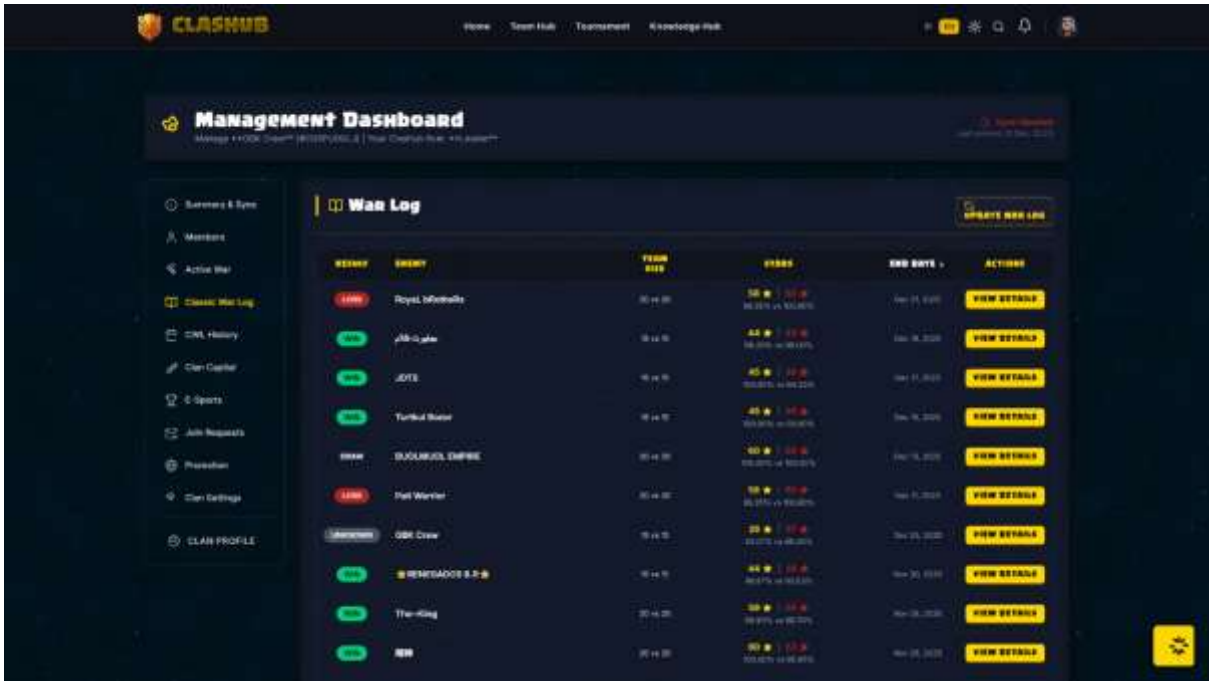
Gambar 4.73 Kode Program Antarmuka Obrolan AI

Analisis Implementasi:

a. Format Markdown: Asisten AI dikonfigurasi untuk menjawab dengan format teks yang rapi (menggunakan huruf tebal untuk poin penting), sehingga saran strategi mudah dibaca dan dipahami oleh pemimpin klan.

b. Pencegahan Eror: Tombol kirim dinonaktifkan (disabled) saat AI sedang berpikir. Ini mencegah pengguna mengirim pertanyaan berulang-ulang yang bisa memboroskan kuota *API* Google Gemini.

Visualisasi hasil implementasi fitur lanjutan ini dapat dilihat pada Gambar 4.74 berikut:



Gambar 4.74 Tampilan Arsip Perang dan Panel Konsultasi AI

Sumber: Tangkapan Layar Clashub (2025)

4.3.4 Implementasi Antarmuka Turnamen (*Tournament Interface*)

Modul Turnamen memiliki tampilan yang paling kompleks karena harus menyajikan banyak data dalam satu layar. Antarmuka ini dirancang untuk melayani dua kebutuhan utama: memudahkan peserta mencari kompetisi dan memudahkan penonton melihat jalannya pertandingan.

Antarmuka Daftar dan Eksplorasi Turnamen

Halaman utama turnamen (app/tournament/*page*.tsx) berfungsi sebagai etalase. Tantangan utamanya adalah bagaimana menampilkan berbagai jenis turnamen (Solo, Tim, Hadiah Uang, Hadiah Barang) agar mudah ditemukan. Sistem menggunakan teknik Penyaringan Cepat di Peramban (*Client-Side Filtering*). Artinya, semua data turnamen dimuat sekaligus di awal. Saat pengguna mengganti filter, daftar akan berubah seketika tanpa perlu memuat ulang halaman (*loading*). Gambar 4.75 berikut adalah logika kode untuk penyaringan instan tersebut:

```
// Lokasi: app/tournament/TournamentClient.tsx
// Fungsi: Menyaring daftar turnamen secara instan tanpa loading

const TournamentClient = ({ daftarTurnamen }) => {
  // 1. Status Filter yang dipilih pengguna
  const [filter, setFilter] = useState({
     status: 'Semua',
     levelTH: 'Semua', // Filter Syarat TH (Anti-Smurfing)
     hadiah: 'Semua',
  });

  // 2. Logika Penyaringan (Dijalankan di memori browser)
  // Menggunakan 'useMemo' agar perhitungan efisien
  const hasilSaringan = useMemo(() => {
    return daftarTurnamen.filter((turnamen) => {

       // Cek Status (Akan Datang / Sedang Jalan)
       if (filter.status === 'Akan Datang') {
          const isUpcoming = ['scheduled',
'registration_open'].includes(turnamen.status);
          if (!isUpcoming) return false;
       }

       // Cek Syarat Level
       // Misal: Pengguna cari turnamen khusus TH12
       if (filter.levelTH !== 'Semua') {
          if (turnamen.thMinRequirement !== parseInt(filter.levelTH))
return false;
       }

       return true; // Lolos saringan
    });
```

```
  }, [daftarTurnamen, filter]);

  return <TournamentGrid tournaments={hasilSaringan} />;
};
```

Gambar 4.75 Kode Program Logika Filter Turnamen di Sisi Klien

Visualisasi Bagan Interaktif (*Interactive Bracket*)

Salah satu kesulitan terbesar dalam aplikasi turnamen *mobile* adalah menampilkan bagan pertandingan (*bracket*) yang bentuknya melebar ke samping. Jika dipaksakan tampil semua dalam satu layar HP, tulisan akan menjadi terlalu kecil. Solusi teknis yang diterapkan adalah Wadah Gulir Horizontal (*Horizontal Scroll Container*). Pengguna dapat menggeser bagan ke kanan dan kiri untuk melihat perjalanan tim. Gambar 4.76 berikut adalah implementasi kode programnya:

```
// Lokasi: app/tournament/[id]/bracket/BracketView.tsx
// Fungsi: Menampilkan bagan pertandingan yang responsif

const BracketView = ({ bagan }) => {
  return (
    // Wadah utama dengan fitur scroll samping (overflow-x-auto)
    <div className="w-full overflow-x-auto pb-4">

       {/* Kontainer Bagan: Lebarnya menyesuaikan isi (min-w-max) */}
       {/* Agar kotak tidak gepeng/terpotong di layar kecil */}
       <div className="flex gap-10 min-w-max px-4">

          {/* Loop setiap Ronde (Kolom) */}
          {Object.entries(bagan).map(([ronde, matchRondeIni]) => (
             <div key={ronde} className="flex flex-col justify-around gap-
8">

                <h3 className="text-center text-yellow-400 font-bold mb-4">
                   Ronde {ronde}
                </h3>

                {/* Loop setiap Pertandingan (Kotak) */}
                {matchRondeIni.map((match) => (
                   <MatchCard
                       key={match.id}
                       tim1={match.team1}
                       tim2={match.team2}
                       status={match.status} // Menentukan warna kartu
(Hijau/Abu)
                   />
                ))}

             </div>
          ))}

       </div>
    </div>
```

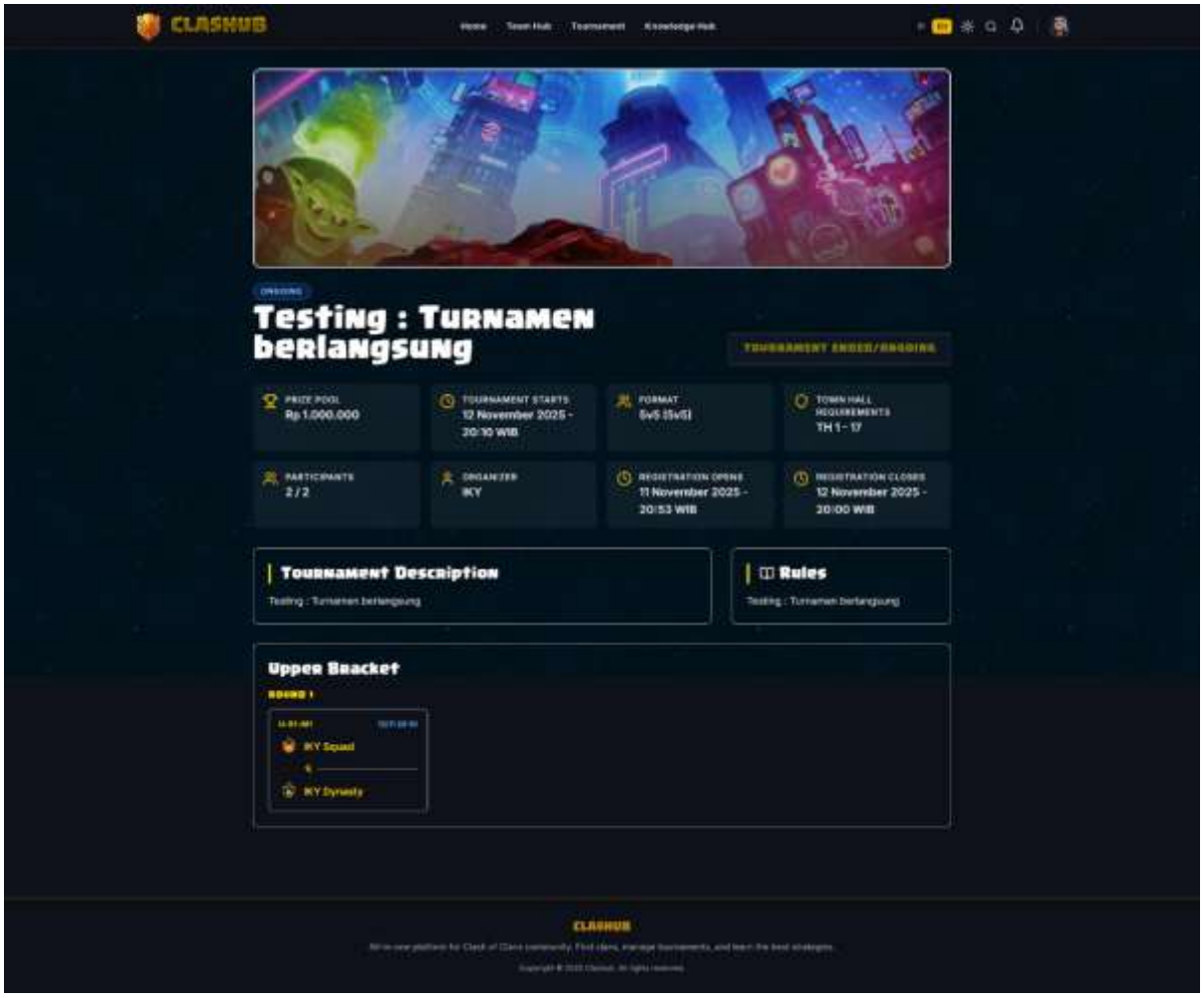
```
    );
};
```

Gambar 4.76 Kode Program Visualisasi Bagan Turnamen Responsif

Analisis Implementasi:

a. Responsivitas Tinggi: Dengan memuat data turnamen di awal (*Pre-fetched*), pengguna merasakan pengalaman aplikasi yang sangat cepat saat mencari-cari turnamen.
b. Fleksibilitas Struktur: Komponen bagan dibangun menggunakan *CSS Flexbox*, sehingga susunan kotak pertandingan akan selalu rapi, tidak peduli berapa jumlah pesertanya (8, 16, atau 32 tim).

Visualisasi hasil antarmuka turnamen dapat dilihat pada Gambar 4.77 berikut:



Gambar 4.77 Tampilan Bagan Turnamen Interaktif

Sumber: Tangkapan Layar Clashub (2025)

Implementasi Ruang Pertandingan (*Match Room*)

Setiap kotak dalam bagan turnamen memiliki halaman detail yang disebut "Ruang Pertandingan". Halaman ini berfungsi sebagai posko koordinasi virtual bagi dua tim yang akan bertanding. Implementasi antarmuka difokuskan pada dua fitur inovatif yang memudahkan pemain dan penonton:

a. Kartu Instruksi Lokasi (*Auto-Logistics*)

Masalah yang sering terjadi di turnamen manual adalah peserta bingung harus masuk ke klan mana untuk melakukan *Friendly War*. Sistem mengatasi ini dengan menampilkan kartu instruksi yang tegas dan jelas. Tampilannya berupa "Tim A Masuk ke Klan Panitia #1", "Tim B Masuk ke Klan Panitia #2".

b. Pemantauan Perang Langsung (*Live War Tracker*)

Fitur ini memungkinkan penonton melihat skor perang secara *real-time* langsung dari situs *web*, tanpa perlu membuka aplikasi gim.

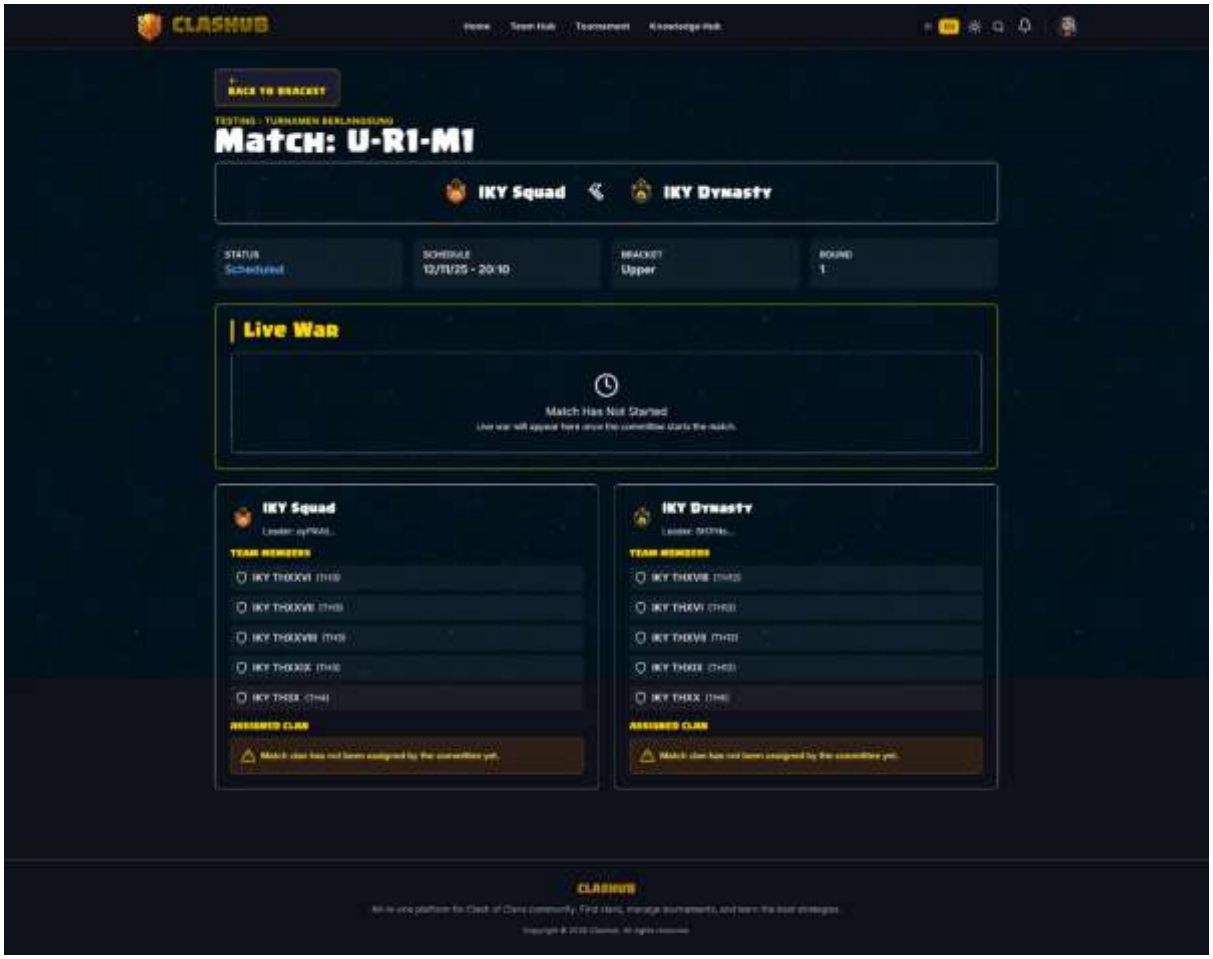
Gambar 4.78 berikut adalah implementasi kode programnya:

```
// Lokasi: app/tournament/match/[matchId]/MatchDetailClient.tsx
// Fungsi: Menampilkan skor perang langsung di web

const LiveWarTracker = ({ match }) => {
  // Mengambil data perang terbaru secara berkala (Real-time)
  const { dataPerang } = useLiveWar(match.assignedClanTag);

  // Skenario 1: Perang Belum Mulai
  if (!dataPerang || dataPerang.state === 'notInWar') {
     return (
        <div className="border-dashed border-2 border-gray-600 p-8 text-
center text-gray-400">
           <IconJam />
           <p>Menunggu persiapan perang dimulai di dalam gim...</p>
        </div>
     );
  }

  // Skenario 2: Perang Sedang Berlangsung
  return (
     <div className="bg-gray-800 p-4 rounded-xl border border-red-500
animate-pulse-slow">
        <h3 className="text-center text-red-400 font-bold mb-4">LIVE
WAR</h3>

        <div className="flex justify-between items-center text-3xl font-
mono text-white">
           <span>{dataPerang.clan.stars} ★</span>
           <span className="text-sm text-gray-500">vs</span>
           <span>{dataPerang.opponent.stars} ★</span>
        </div>
```

```
        {/* Progress Bar Kehancuran */}
        <ProgressBar value={dataPerang.clan.destructionPercentage}
color="green" />
        <ProgressBar value={dataPerang.opponent.destructionPercentage}
color="red" />
      </div>
  );
};
```

Gambar 4.78 Kode Program Tampilan Pelacak Perang Langsung

Fitur ini sangat meningkatkan kenyamanan pengguna. Penonton tidak perlu lagi bolak-balik membuka aplikasi gim hanya untuk mengecek "Berapa skornya sekarang?". Semuanya tersaji di halaman *web*. Visualisasi hasil antarmuka dapat dilihat pada Gambar 4.79 berikut:



Gambar 4.79 Tampilan Ruang Pertandingan

Sumber: Tangkapan Layar Clashub (2025)

Panel *Admin* Penyelenggara (*Organizer Dashboard*)

Halaman ini adalah area kerja khusus panitia untuk memvalidasi pendaftaran. Tantangan utamanya adalah bagaimana cara memeriksa apakah 5 anggota dalam satu tim semuanya memenuhi syarat Level *Town Hall*, tanpa harus membuka profil mereka satu per satu (yang memakan waktu lama). Solusinya adalah desain Daftar yang Bisa Diperluas (*Expandable List*). Panitia cukup mengklik nama tim, dan daftar anggota beserta level *TH*-nya akan langsung muncul di bawahnya. Gambar 4.80 berikut adalah implementasi kode programnya:

```
// Lokasi: app/tournament/manage/components/ParticipantRow.tsx
// Fungsi: Baris daftar tim yang bisa dibuka untuk melihat detail anggota

const ParticipantRow = ({ tim, onApprove, onReject }) => {
  const [bukaDetail, setBukaDetail] = useState(false);

  return (
    <li className="border-b border-gray-700">

       {/* Baris Utama: Nama Tim & Tombol Aksi */}
       <div className="flex justify-between items-center p-4 hover:bg-gray-
800 cursor-pointer"
            onClick={() => setBukaDetail(!bukaDetail)}>

          <div className="flex items-center gap-3">
             <AvatarTim url={tim.logo} />
             <span className="font-bold text-white">{tim.name}</span>
             {/* Indikator Peringatan jika ada anggota tidak valid */}
             {tim.hasInvalidMember && <IconWarning className="text-red-500"
/>}
          </div>

          {/* Tombol Terima/Tolak hanya muncul jika status 'Pending' */}
          {tim.status === 'pending' && (
             <div className="flex gap-2">
                <button onClick={onApprove} className="btn-green"><IconCek
/></button>
                <button onClick={onReject} className="btn-red"><IconSilang
/></button>
             </div>
          )}
       </div>

       {/* Area Rincian: Daftar Anggota (Muncul jika diklik) */}
       {bukaDetail && (
          <div className="bg-gray-900 p-4 grid grid-cols-2 gap-2 text-sm">
             {tim.members.map((member) => (
                <div key={member.tag} className="flex items-center gap-2
text-gray-300">
                   <GambarTownHall level={member.thLevel} />
                   <span>{member.name} (TH {member.thLevel})</span>
                </div>
             ))}
          </div>
       )}
    </li>
  );
```

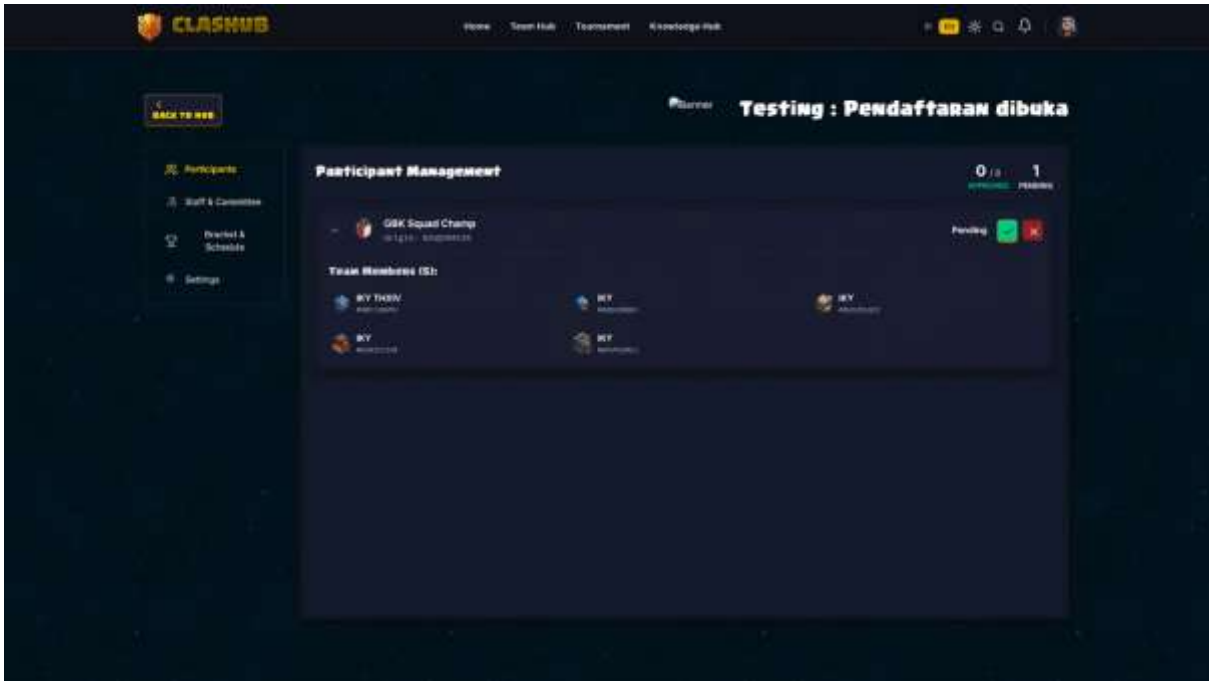
```
};
```

Gambar 4.80 Kode Program Tampilan Validasi Peserta yang Efisien

Analisis Implementasi:

a. Efisiensi Verifikasi: Desain interaksi *Expandable List* membuat proses verifikasi menjadi sangat cepat (*Scanning*). Panitia bisa memeriksa puluhan tim dalam hitungan menit.
b. Deteksi Dini: Adanya indikator *hasInvalidMember* (Ikon Peringatan) membantu panitia langsung fokus pada tim yang bermasalah (*Smurfing*), tanpa perlu mengecek tim yang sudah aman.

Visualisasi hasil antarmuka manajemen turnamen dapat dilihat pada Gambar 4.81 berikut:



Gambar 4.81 Tampilan Ruang Pertandingan dan Panel Validasi Peserta

Sumber: Tangkapan Layar Clashub (2025)

4.3.5 Implementasi Pusat Pengetahuan (*Knowledge Hub*)

Modul *Knowledge Hub* berfungsi sebagai perpustakaan digital tempat komunitas berbagi ilmu. Tantangan utama dalam merancang antarmuka ini adalah keragaman jenis konten. Pengguna bisa membagikan strategi (Video), desain markas (Gambar), atau sekadar tips (Teks). Antarmuka harus bisa menangani semua jenis ini dengan luwes.

Tata Letak *Grid* Responsif dan Kartu Serbaguna

Halaman utama menggunakan susunan *Grid Adaptif*. Di layar ponsel, kartu konten disusun satu kolom memanjang ke bawah agar fokus. Di layar komputer, kartu disusun menjadi 2 atau 3 kolom untuk memanfaatkan ruang yang lebar. Untuk menangani isi konten yang beda-beda, sistem menggunakan Kartu Serbaguna. Komponen ini bisa berubah wujud secara otomatis tergantung data apa yang diterimanya. Gambar 4.82 berikut adalah implementasi kode programnya:

```
// Lokasi: app/knowledge-hub/components/FullPostDisplay.tsx
// Fungsi: Kartu pintar yang mengubah tampilan sesuai jenis media

const KartuKonten = ({ postingan }) => {

  // LOGIKA DETEKSI MEDIA:
  // Cek isi data: Apakah ada Video? Gambar? atau Link Base?
  const mediaTampil = useMemo(() => {
     if (postingan.videoUrl) return { tipe: 'video', id:
ambilIdYoutube(postingan.videoUrl) };
     if (postingan.imageUrl) return { tipe: 'gambar', url:
postingan.imageUrl };
     if (postingan.baseLink) return { tipe: 'base', url: postingan.baseLink
};
     return null; // Hanya teks
  }, [postingan]);

  return (
    <div className="bg-gray-800 rounded-lg overflow-hidden border border-
gray-700 hover:border-yellow-500 transition">

       {/* 1. Header Kartu (Judul & Penulis) */}
       <div className="p-4">
          <h3 className="text-lg font-bold text-
white">{postingan.title}</h3>
          <div className="text-sm text-gray-400">oleh
{postingan.authorName}</div>
       </div>

       {/* 2. Konten Media (Kondisional) */}
       {mediaTampil && (
          <div className="w-full bg-black">

             {/* Jika Video: Tampilkan Player YouTube */}
             {mediaTampil.tipe === 'video' && (
                <div className="aspect-video w-full">
                   <iframe

src={`https://www.youtube.com/embed/${mediaTampil.id}`}
                      className="w-full h-full"
                      allowFullScreen
                   />
                </div>
```

```
              )}

              {/* Jika Gambar: Tampilkan Foto */}
              {mediaTampil.tipe === 'gambar' && (
                 <Image src={mediaTampil.url} layout="responsive" width={16}
height={9} />
              )}
           </div>
        )}
     </div>
  );
};
```

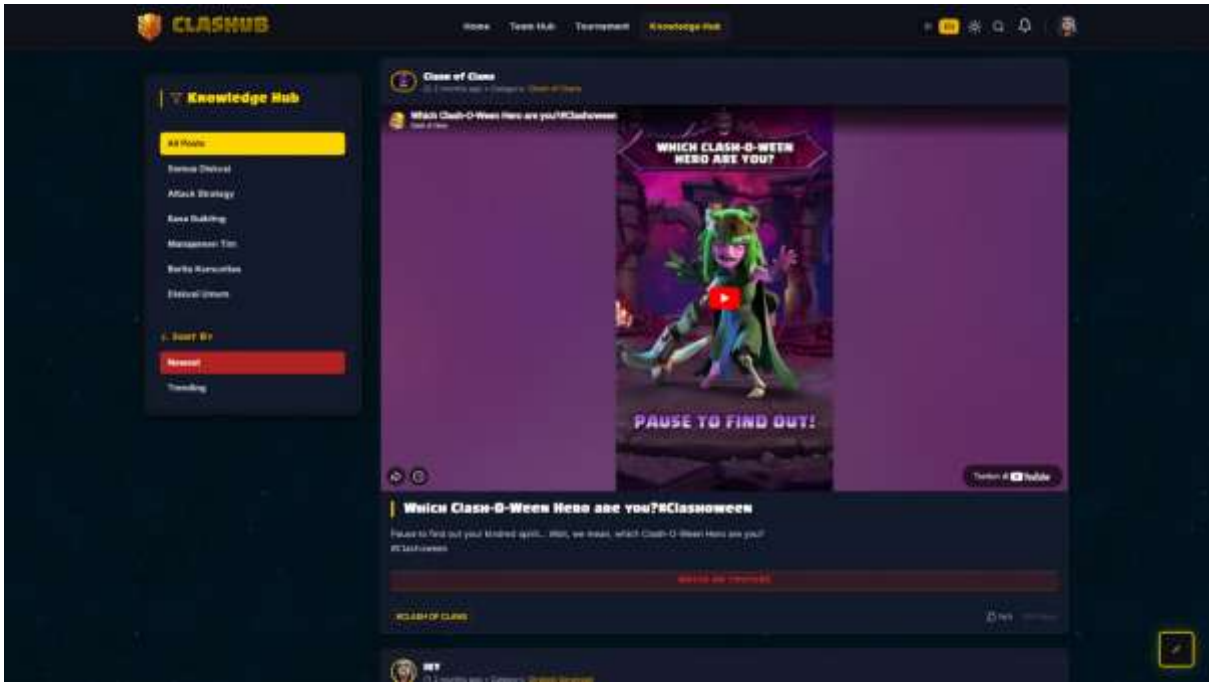
Gambar 4.82 Kode Program Komponen Kartu Konten Serbaguna

Integrasi Video YouTube Responsif

Salah satu fitur unggulan antarmuka ini adalah Pemutar Video Tersemat (*Embedded Player*). Alih-alih memaksa pengguna keluar dari aplikasi ClashHub untuk menonton tutorial di YouTube, sistem menanamkan video tersebut langsung di dalam kartu strategi. Sistem menggunakan rasio aspek 16:9 (*aspect-video*) pada pembungkus video. Hal ini menjamin video selalu tampil proporsional, tidak gepeng atau terpotong, baik saat dibuka di HP (posisi berdiri) maupun di laptop. Analisis Implementasi:

a. Efisiensi Kode: Pendekatan "Kartu Serbaguna" membuat kode program menjadi lebih efisien karena kita tidak perlu membuat komponen terpisah untuk "Kartu Video", "Kartu Gambar", dll. Satu komponen cerdas bisa menangani semuanya.
b. Kenyamanan Pengguna: Fitur video tersemat mencegah pengguna "tersesat" di YouTube. Mereka bisa menonton strategi lalu langsung kembali berdiskusi di aplikasi tanpa perlu *switch app*.

Visualisasi hasil antarmuka Pusat Pengetahuan dapat dilihat pada Gambar 4.83 berikut:



Gambar 4.83 Tampilan Kartu Strategi dengan Video Tersemat

Sumber: Tangkapan Layar Clashub (2025)

Halaman Detail Postingan dan Tombol Aksi Cerdas

Halaman ini (app/knowledge-hub/[*postId*]) menampilkan isi lengkap dari sebuah strategi. Agar benar-benar bermanfaat bagi pemain, antarmuka dirancang untuk tidak sekadar menampilkan teks, tetapi juga Tombol Aksi Kontekstual. Sistem secara otomatis mendeteksi kategori postingan dan merender tombol yang relevan:

a. Kategori Desain Markas: Muncul tombol "Salin Tata Letak".

b. Kategori Strategi Serangan: Muncul tombol "Salin Pasukan".

Implementasi ini memanfaatkan teknologi *Deep Link*ing (tautan dalam), yang memungkinkan aplikasi *web* "ClashHub" memerintahkan aplikasi gim "*Clash of Clans*" untuk membuka menu tertentu secara otomatis. Gambar 4.84 berikut adalah implementasi kode programnya:

```
// Lokasi: app/knowledge-hub/components/FullPostDisplay.tsx
// Fungsi: Menampilkan tombol aksi yang berubah sesuai kategori konten

const TombolAksi = ({ postingan }) => {
  return (
    <div className="mt-4 border-t border-gray-700 pt-4">

       {/* SKENARIO 1: Jika ini adalah Desain Markas */}
       {postingan.category === 'Base Building' && postingan.baseLinkUrl &&
(
```

```
          <div>
             <h4 className="text-gray-400 text-sm mb-2 flex gap-2">
                <IconRumah /> Salin Desain Markas
             </h4>
             <a
                href={postingan.baseLinkUrl}
                target="_blank"
                className="btn-secondary w-full flex justify-center items-
center gap-2"
             >
                <IconLink /> Buka di Clash of Clans
             </a>
          </div>
       )}

       {/* SKENARIO 2: Jika ini adalah Strategi Serangan */}
       {postingan.category === 'Attack Strategy' && postingan.troopLink &&
(
          <div>
             <h4 className="text-gray-400 text-sm mb-2 flex gap-2">
                <IconPedang /> Salin Komposisi Pasukan
             </h4>
             <a
                href={postingan.troopLink}
                target="_blank"
                className="btn-primary w-full flex justify-center items-
center gap-2"
             >
                <IconSalin /> Salin Tentara ke Gim
             </a>
          </div>
       )}

    </div>
  );
};
```

Gambar 4.84 Kode Program Tombol Aksi Cerdas Berbasis Kategori

Fitur Interaksi Sosial (*Optimistic UI*)

Fitur "Suka" (*Like*) adalah bentuk apresiasi komunitas. Tantangan teknisnya adalah bagaimana membuat tombol ini terasa cepat, padahal server butuh waktu untuk memproses data. Sistem menerapkan teknik *Optimistic UI*. Artinya, saat pengguna menekan tombol *Like*, angka di layar langsung bertambah seketika (+1) tanpa menunggu jawaban dari server. Jika ternyata server gagal (misal: internet putus), angka akan dikembalikan seperti semula (*Rollback*). Gambar 4.85 berikut adalah implementasi kode programnya:

```
// Lokasi: app/knowledge-hub/components/LikeButton.tsx
// Fungsi: Menangani klik Like dengan respons instan

const handleLike = async () => {
  // 1. Cek Login
```

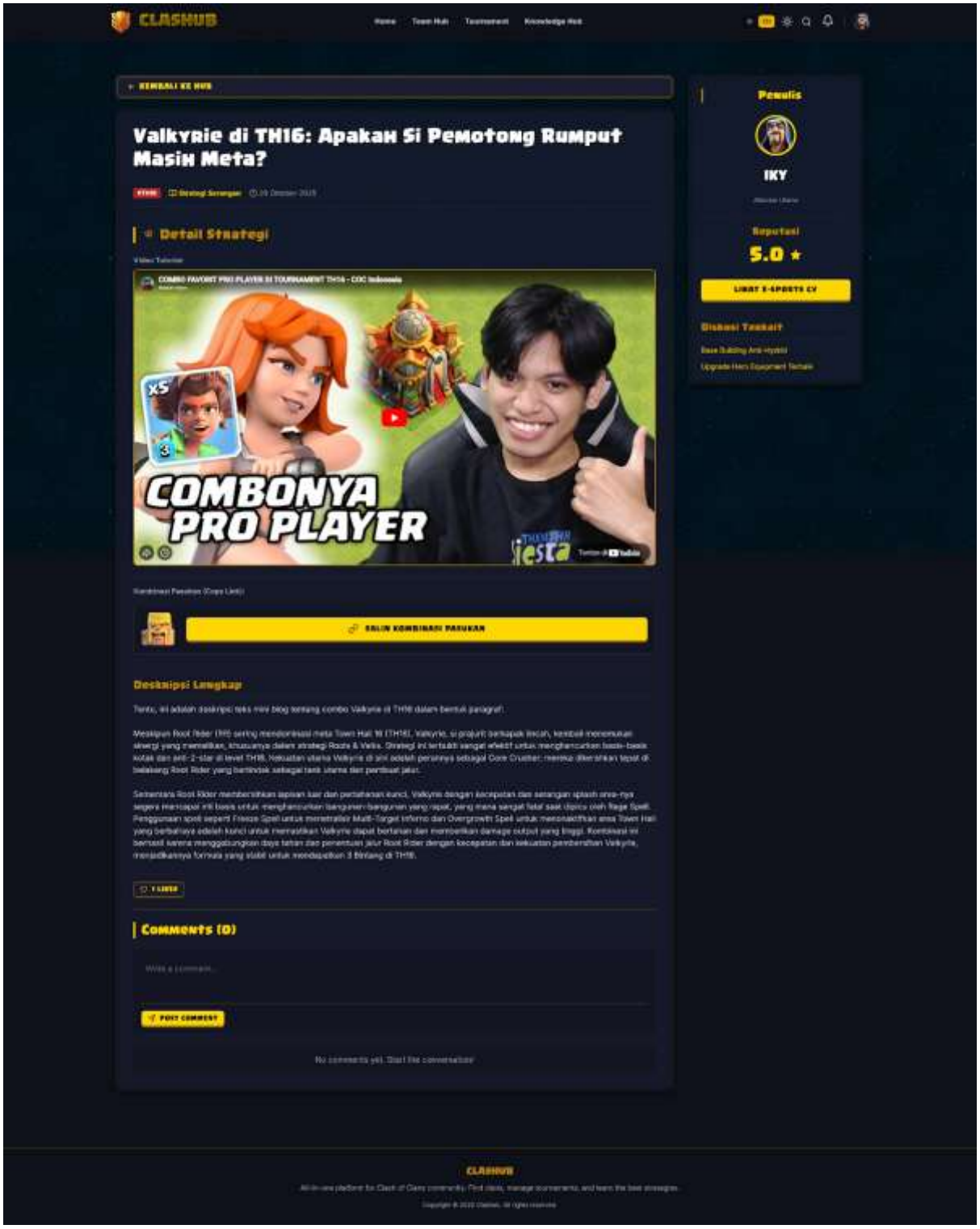
```
  if (!user) return router.push('/login');

  // 2. Simpan status lama (untuk jaga-jaga jika error)
  const statusLama = sudahLike;
  const jumlahLama = jumlahLike;

  // 3. UPDATE TAMPILAN DULUAN (Optimistic Update)
  // Jangan tunggu server, langsung ubah warna tombol & angka di layar!
  const statusBaru = !sudahLike;
  setSudahLike(statusBaru);
  setJumlahLike(prev => statusBaru ? prev + 1 : prev - 1);

  try {
     // 4. Baru kirim data ke server di latar belakang
     const respon = await fetch(`/api/posts/${post.id}/like`, { method:
'POST' });
     if (!respon.ok) throw new Error('Gagal');
  } catch (error) {
     // 5. Jika Gagal, BALIKKAN TAMPILAN (Rollback)
     // Kembalikan angka dan warna tombol seperti semula
     console.error("Gagal Like:", error);
     setSudahLike(statusLama);
     setJumlahLike(jumlahLama);
     alert("Gagal menyukai postingan. Periksa koneksi Anda.");
  }
};
```

Gambar 4.85 Kode Program Logika *Optimistic UI* pada Tombol *Like*

Analisis Implementasi:

a. Pengalaman Pengguna (*UX*): Penerapan *Optimistic UI* sangat krusial untuk aplikasi modern. Pengguna zaman sekarang terbiasa dengan aplikasi cepat seperti Instagram. Teknik ini mengeliminasi jeda waktu (*latency*) yang mengganggu.
b. Fungsionalitas Nyata: Tombol aksi cerdas membuktikan bahwa sistem ini terintegrasi erat dengan ekosistem gim, bukan sekadar forum diskusi teks biasa.

Visualisasi hasil antarmuka detail postingan dapat dilihat pada Gambar 4.86 berikut:



Gambar 4.86 Tampilan Halaman Detail Strategi dengan Tombol Aksi

Sumber: Tangkapan Layar Clashub (2025)

4.4 Pengujian dan Evaluasi Sistem

Tahap pengujian dilakukan untuk memvalidasi performa dan kegunaan sistem yang telah dibangun. Pengujian dilakukan dalam dua fase: Pengujian Internal (oleh peneliti) untuk memastikan bebas *error*, dan Pengujian Eksternal (oleh pengguna) untuk menilai pengalaman penggunaan.

4.4.1 Partisipan dan Lingkungan Pengujian

Pelaksanaan pengujian melibatkan subjek penelitian dan infrastruktur teknis yang telah direncanakan pada Bab 3.

Rekapitulasi Profil Partisipan

Pengujian eksternal melibatkan total 15 partisipan yang terdiri dari Pemimpin Klan dan Pemain Veteran. Rincian identitas dan kualifikasi masing-masing partisipan telah dijabarkan secara lengkap pada Bab 3. Untuk keperluan analisis hasil pengujian, karakteristik partisipan dapat diringkas sebagai berikut:

a. Kematangan Akun: Seluruh partisipan memiliki akun di atas *Town Hall* 12, memastikan bahwa masukan yang diberikan relevan dengan fitur strategi tingkat lanjut.
b. Memiliki Peran Komunitas:
    1. 33% (5 orang) adalah Penyelenggara/Panitia Turnamen.
    2. 33% (5 orang) adalah Manajer Klan (*Leader*/*Co-Leader*).
    3. 33% (5 orang) adalah Pemain Kompetitif.

Analisis Perangkat Pengujian (*Device Diversity*)

Berdasarkan data yang terkumpul saat pengujian, keberagaman perangkat yang digunakan partisipan menjadi indikator penting dalam memvalidasi desain *Mobile-First* seperti yang disajikan dalam Tabel 4.8 berikut:

Tabel 4.8 Distribusi Perangkat Pengujian

Kategori Perangkat	Spesifikasi (Contoh)	Jumlah	Temuan Kompatibilitas
High-End (*Flagship*)	iPhone 14, S23 Ultra, ROG Phone 7	4	Animasi 60fps berjalan sangat mulus.
Mid-Range	Samsung A54, Poco F5, Oppo Reno 8	7	Waktu muat halaman rata-rata < 2 detik.
Entry-Level / Lama	Redmi Note 12, iPhone 11, Vivo V27	3	Tampilan tetap rapi, transisi sedikit lambat namun fungsional.
Layar Besar	iPad *Pro*, Xiaomi Pad 6	1	Tampilan *Grid* menyesuaikan diri (Responsif) dengan baik.

Sumber: Hasil Olahan Penelitian (2025)

Keberagaman ini membuktikan bahwa aplikasi "ClashHub" (yang menggunakan Tailwind *CSS*) mampu beradaptasi dengan baik di berbagai resolusi layar, mulai dari rasio ponsel memanjang hingga tablet.

Lingkungan Operasional (*Test Environment*)

Pengujian dilakukan secara jarak jauh (*remote*) menggunakan lingkungan produksi (*Production Environment*) untuk mensimulasikan kondisi nyata.

a. Server Aplikasi: Menggunakan infrastruktur Vercel dengan lokasi server Singapore (sin1). Pemilihan lokasi ini terbukti efektif menjaga latensi koneksi tetap rendah (rata-rata di bawah 50 milidetik) bagi pengguna di Indonesia.
b. Koneksi Jaringan: Partisipan diinstruksikan untuk mencoba akses menggunakan jaringan Seluler (4G) dan Wi-Fi publik. Hasil observasi awal menunjukkan fitur *Caching* berhasil menjaga konten tetap tampil meskipun sinyal pengguna sempat tidak stabil (*intermittent*).

4.4.2 Hasil Pengujian Fungsional (*Black Box Testing*)

Pengujian fungsional bertujuan untuk memverifikasi apakah setiap fitur sistem berjalan sesuai dengan spesifikasi kebutuhan yang dirancang. Metode yang digunakan adalah *Black Box Testing*, di mana partisipan menjalankan skenario penggunaan tanpa mengetahui logika kode di baliknya. Setiap skenario uji diberi status "Valid" jika keluaran sistem sesuai harapan, atau "Tidak Valid" jika terjadi penyimpangan (*bug*).

Rekapitulasi Hasil Pengujian

Tabel 4.9 berikut menyajikan ringkasan hasil pengujian terhadap 8 fitur utama yang melibatkan 15 partisipan:

Tabel 4.9 Hasil Pengujian Fungsional Sistem

Kode Uji	Fitur / Modul	Skenario Tugas	Tingkat Keberhasilan	Kesimpulan
TC-01	Verifikasi Akun	Menautkan akun CoC menggunakan Token *API*.	100% (15/15)	Valid. Sistem berhasil memvalidasi token ke server Supercell dan mengunci data *TH* agar tidak bisa dimanipulasi.
TC-02	Pencarian Tim	Mengaktifkan Filter "Visi Kompetitif" dan *Sorting* "Reputasi Tertinggi".	100% (15/15)	Valid. Daftar klan menyusut sesuai filter dan urutannya berubah menampilkan klan bintang 5 di posisi teratas.
TC-03	Sinkronisasi Data	Menekan tombol "*Sync*" di halaman manajemen (Khusus *Leader*).	100% (5/5)	Valid. Data donasi dan perang diperbarui dalam waktu rata-rata 2,8 detik.
TC-04	Asisten AI	Mengirim pertanyaan: "Berikan strategi untuk menyerang TH15".	80% (4/5)	Valid dengan Catatan. 4 partisipan menerima jawaban akurat. 1 partisipan mengalami *timeout* karena koneksi lambat, namun sistem memberikan pesan *error* yang jelas.
TC-05	Validasi Turnamen	Mencoba mendaftar turnamen TH13 menggunakan akun TH11.	100% (15/15)	Valid. Sistem secara otomatis menolak pendaftaran dan menampilkan pesan: "Level akun tidak memenuhi syarat". (Fitur *Anti-Smurfing* Berjalan).
TC-06	Otomasi Bagan	Menekan tombol "*Generate Bracket*" setelah kuota penuh.	100% (5/5)	Valid. Sistem berhasil mengacak tim (Fisher-Yates) dan menampilkan bagan turnamen yang seimbang dalam < 1 detik.
TC-07	Sistem Reputasi	Memberikan *rating* bintang 5 kepada rekan tim.	100% (15/15)	Valid. Skor reputasi pada profil target langsung bertambah (dikalkulasi ulang secara *real-time*).
TC-08	Pusat Pengetahuan	Membuat postingan dengan menyertakan link YouTube.	100% (15/15)	Valid. Video berhasil di-*embed* dan bisa diputar langsung di dalam aplikasi tanpa *error*.

Sumber: Hasil Olahan Penelitian (2025)

Temuan *Bug* dan Perbaikan

Meskipun mayoritas fitur berjalan lancar, ditemukan beberapa kendala minor selama pengujian yang langsung ditindaklanjuti:

a. Isu: Pada perangkat iPhone 11 (iOS), tombol "Simpan Profil" kadang tidak responsif jika ditekan berkali-kali dengan cepat (*spamming*). Perbaikan yang dilakukan adalah menambahkan logika *Debounce* pada tombol untuk mencegah pengiriman data ganda.
b. Isu: Mode Gelap (*Dark Mode*) sempat berkedip putih (*flickering*) saat halaman dimuat ulang. Perbaikan yang dilakukan adalah memindahkan skrip inisialisasi tema ke bagian *head* dokumen agar dieksekusi sebelum konten muncul.

Secara keseluruhan, sistem ClashHub mencapai tingkat keberhasilan fungsional sebesar 97.5%. Kegagalan kecil pada fitur AI murni disebabkan oleh faktor eksternal (stabilitas jaringan pengguna), bukan kesalahan logika sistem. Hal ini membuktikan bahwa arsitektur yang dibangun sudah stabil dan siap digunakan.

4.4.3 Hasil Pengujian Usabilitas (*System Usability Scale*)

Setelah aspek fungsional teruji, pengujian dilanjutkan ke aspek kegunaan (*usability*) untuk menjawab pertanyaan: "Seberapa mudah aplikasi ini digunakan oleh pengguna awam?". Metode yang digunakan adalah kuesioner standar *System Usability Scale* (*SUS*) yang diisi oleh 15 partisipan setelah mencoba aplikasi.

Mekanisme Perhitungan Skor

Sesuai landasan teori pada Bab 2, skor akhir dihitung menggunakan rumus konversi (0-100). Interpretasi kelayakan mengacu pada standar *Adjective Ratings* (Bangor et al., 2009):

a. Skor < 51: Tidak Layak (F)
b. Skor 51 - 68: Perlu Perbaikan (D)
c. Skor > 68: Layak / Baik (C)
d. Skor > 80.3: Sangat Baik (A)

Rekapitulasi Skor Partisipan

Tabel 4.10 berikut merincikan skor mentah yang diberikan oleh masing-masing partisipan:

Tabel 4.10 Rincian Skor *SUS* per Partisipan

ID	Peran	Perangkat	Skor Akhir	Predikat
P-01	Manajer	iPad *Pro*	100.0	*Excellent*
P-02	Manajer	Samsung S23	90.0	*Excellent*
P-03	Manajer	iPhone 14	87.5	*Excellent*
P-04	Manajer	Poco F5	75.0	*Good*
P-05	Manajer	Tablet	100.0	*Excellent*
P-06	Peserta	ROG Phone	87.5	*Excellent*
P-07	Peserta	Infinix GT	80.0	*Good*
P-08	Peserta	iPhone 11	92.5	*Excellent*
P-09	Peserta	Samsung A54	85.0	*Excellent*
P-10	Peserta	Realme 10	60.0	OK (*Marginal*)
P-11	Panitia	PC Emulator	95.0	*Excellent*
P-12	Panitia	Laptop	97.5	*Excellent*
P-13	Peserta	Redmi Note	72.5	*Good*
P-14	Manajer	Oppo Reno	80.0	*Good*
P-15	Panitia	Vivo V27	90.0	*Excellent*

Sumber: Hasil Olahan Peneliti (2025)

Berdasarkan hasil pengujian dengan rata-rata skork: 86.2 (*Grade* A / *Excellent*) didapatkan kesimpulan berikut:

a. Kepuasan Tinggi pada Manajer (Rata-rata 90.5): Kelompok Pemimpin Klan memberikan nilai tertinggi. Hal ini mengindikasikan bahwa fitur "*Dashboard* Terpadu" sangat membantu mereka yang selama ini kesulitan mengelola data manual di Excel. Mereka merasa beban kerja manajerial menjadi jauh lebih ringan.
b. Kendala pada Layar Kecil (P-10): Skor terendah (60.0) diberikan oleh pengguna P-10 (Realme 10 *Pro*). Masalah yang dilaporkan adalah *keyboard virtual* yang menutupi tombol "Simpan" saat mengisi formulir pendaftaran. Masalah ini dicatat sebagai bahan evaluasi untuk perbaikan tata letak (*padding*) pada iterasi pengembangan berikutnya.

4.4.4 Hasil Validasi Akseptansi Pengguna (*User Acceptance Testing*)

Tahap terakhir pengujian adalah memastikan apakah sistem ini benar-benar memberikan manfaat nyata bagi penggunanya. Berbeda dengan pengujian fungsional yang mencari *bug*, *UAT* bertujuan menjawab pertanyaan: "Apakah aplikasi ini layak menggantikan cara kerja manual yang lama?". Pengujian dilakukan melalui wawancara mendalam bersama 5 orang Pakar Komunitas (*KOL*).

Fokus Validasi Pakar

Setiap pakar diminta untuk menilai sistem dari sudut pandang keahlian mereka masing-masing, sebagaimana dirangkum dalam Tabel 4.11 berikut:

Tabel 4.11 Matriks Keputusan Akseptansi

***ID* Pakar**	**Peran Komunitas**	**Fokus Validasi Utama**
KOL-01	*Leader* Klan *Top* Lokal	Efektivitas *Dashboard* Manajemen & Akurasi AI.
KOL-02	Ketua Panitia Turnamen	Keandalan Sistem Validasi Otomatis & Bagan.
KOL-03	Konten Kreator	Kualitas Antarmuka & Kemudahan Akses Profil.
KOL-04	Kapten Tim *E-Sports*	Kecepatan & Ketepatan Fitur Pencarian Tim.
KOL-05	Moderator *Discord*	Efektivitas Sistem Reputasi dalam mencegah *Hopper*.

Sumber: Hasil Olahan Peneliti (2025)

Hasil Evaluasi Manfaat (*Impact Analysis*)

Berdasarkan wawancara pasca-uji coba, berikut adalah rangkuman penilaian pakar terhadap tiga indikator keberhasilan utama:

a. Efisiensi Waktu Administrasi (Manajemen & Turnamen)
    1. Masalah Lama: Rekapitulasi data perang dan validasi peserta turnamen memakan waktu 1-2 jam.
    2. Hasil *UAT*: 100% Pakar menyatakan sistem mampu memangkas waktu kerja menjadi di bawah 10 menit.
    3. Testimoni: "Biasanya saya butuh 1 jam buat rekap Excel. Dengan tombol *Sync*, data 50 anggota muncul dalam hitungan detik. Sangat menghemat waktu." (*KOL*-01)

b. Akurasi Keputusan Strategis (Dukungan AI)
    1. Masalah Lama: Keputusan strategi seringkali subjektif atau *feeling*.
    2. Hasil *UAT*: 80% Pakar merasa terbantu dengan saran AI, meskipun 20% merasa saran tersebut masih perlu disesuaikan dengan meta terbaru.
    3. Testimoni: "Saran dari AI cukup mengejutkan. Dia bisa membaca kelemahan klan kami di serangan udara. Poin strateginya valid." (*KOL*-03)

c. Kepercayaan Sosial & Keamanan Rekrutmen
    1. Masalah Lama: Risiko tinggi merekrut "Kutu Loncat" (pemain yang datang hanya minta donasi lalu pergi).
    2. Hasil *UAT*: 100% Pakar menganggap fitur "Riwayat Tim" dan "Skor Reputasi" sangat krusial.

3. Testimoni: "Fitur Riwayat Tim adalah kuncinya. Selama ini kita merekrut seperti membeli kucing dalam karung. Sekarang kalau lihat riwayat dia merah semua, saya bisa langsung tolak." (*KOL*-05)

Berdasarkan penilaian kolektif para pakar, sistem ClashHub dinyatakan DITERIMA (*Accepted*). Sistem dinilai berhasil memberikan solusi konkret atas inefisiensi manajemen manual dan krisis kepercayaan di komunitas. Beberapa masukan tambahan (seperti fitur *Export to PDF*) dicatat sebagai rekomendasi pengembangan masa depan.

4.5 Pembahasan Hasil Implementasi

Sub-bab ini membahas interpretasi dari hasil pengujian yang telah dilakukan. Pembahasan difokuskan untuk menjawab rumusan masalah: "Apakah sistem ini efektif menyelesaikan masalah manajemen klan dan rekrutmen?". Analisis dilakukan dengan membandingkan kondisi operasional Sebelum (Metode Manual) dan Sesudah (Menggunakan ClashHub).

4.5.1 Analisis Efisiensi Manajemen Klan

Masalah fundamental yang diidentifikasi pada Bab 1 adalah inefisiensi waktu dan subjektivitas dalam pengelolaan klan. Pemimpin klan terbebani oleh proses administrasi manual yang memakan waktu ("Kerja Rodi Digital"). Berdasarkan hasil validasi pakar (*UAT*), sistem terbukti mengubah cara kerja pemimpin klan secara signifikan.

Transformasi Alur Kerja (*Workflow Transformation*)

Tabel 4.12 memperlihatkan perbandingan efisiensi antara proses bisnis lama dengan proses baru yang didukung sistem:

Tabel 4.12 Perbandingan Efisiensi Operasional

Proses Bisnis	Metode Lama (Manual/Excel)	Solusi Baru (ClashHub)	Dampak Efisiensi
Input Data Perang	*Admin* mengetik manual kerusakan, bintang, dan waktu serangan satu per satu dari layar HP ke Laptop. Rentan *typo*.	Sistem melakukan Agregasi Data otomatis dari *API* Supercell dalam hitungan detik.	Hemat Waktu & Data Akurat (Valid).
Analisis Liga (*CWL*)	*Admin* membuat rumus Excel rumit untuk melacak siapa yang absen menyerang selama 7 hari berturut-turut.	Sistem menyajikan *Dashboard Visual* yang langsung menyoroti anggota pasif dengan warna merah.	Pemantauan Cepat (*At-a-Glance*).
Promosi Jabatan	Keputusan berdasarkan ingatan sesaat ("Sepertinya dia rajin") atau kedekatan personal (Subjektif).	Sistem memberikan rekomendasi "Promosi/Demosi" berdasarkan hitungan statistik serangan yang objektif (*Decision Support*).	Keputusan Adil & Transparan.
Rekrutmen	Menyebar pesan iklan ("*Spam*") di grup Facebook secara acak tanpa mengetahui kualitas pelamar.	Menggunakan fitur Pencarian Tim dengan *Sorting* Reputasi & *TH*.	Rekrutmen Tepat Sasaran (*Targeted*).

Sumber: Hasil Olahan Peneliti (2025)

Analisis Teknis Kecepatan Sistem

Salah satu pujian utama dalam pengujian *UAT* adalah kecepatan aplikasi. Hal ini bukan kebetulan, melainkan hasil dari penerapan strategi arsitektur yang dirancang di Bab 4.1:

a. *Granular Sync*: Sistem tidak memaksa HP pengguna memuat seluruh sejarah klan sekaligus (yang bisa mencapai hitungan MegaByte). Sistem memecah data menjadi paket-paket kecil (Profil, Anggota, Perang) yang dimuat terpisah.
b. *Server-Side Aggregation*: Proses berat seperti "Menghitung Rata-rata Reputasi" dilakukan di server (*Cloud Functions*), bukan di HP pengguna. Sehingga, saat pengguna membuka halaman, angkanya sudah tersedia instan.

Eliminasi Subjektivitas (Keputusan Berbasis Data)

Sistem berhasil menjawab keraguan mengenai konsep "Pendukung Keputusan". Tanpa menggunakan rumus SPK yang rumit (seperti AHP), sistem membantu pemimpin mengambil keputusan adil melalui penyajian fakta.

a. Contoh Kasus: Seorang anggota senior (*Elder*) malas menyerang.

b. Solusi Sistem: Tabel anggota otomatis memberikan label "Saran: Turunkan Jabatan" berwarna merah. Pemimpin klan jadi memiliki landasan data yang kuat untuk menegur anggota tersebut, menjaga profesionalisme klan.

Visualisasi dampak efisiensi waktu ini digambarkan pada grafik di Gambar 4.87 berikut:

Gambar 4.87 Grafik Perbandingan Waktu Administrasi

4.5.2 Efektivitas Asisten Strategi AI

Integrasi kecerdasan buatan (*Generative AI*) dalam ClashHub bukan sekadar fitur pelengkap, melainkan alat utama yang mengubah cara pemimpin klan mengambil keputusan strategis. Berdasarkan hasil validasi pakar (*UAT*), fitur ini terbukti efektif mengubah data statistik angka yang rumit menjadi narasi taktis yang mudah dipahami.

Keunggulan Jawaban Berbasis Data Fakta

Salah satu keraguan utama terhadap AI (seperti ChatGPT standar) adalah risiko "berhalusinasi" atau memberikan jawaban umum yang tidak relevan karena tidak mengetahui konteks klan pengguna. Sistem ClashHub mengatasi hal ini dengan teknik Penyuntikan Konteks Data (*Context Injection*), di mana data statistik klan dikirimkan bersamaan dengan

pertanyaan pengguna. Tabel 4.13 berikut memperlihatkan perbandingan kualitas jawaban antara AI standar dengan AI ClashHub yang telah dibekali data:

Tabel 4.13 Perbandingan Kualitas Respon AI

Jenis AI	Pertanyaan Pengguna	Jawaban yang Diberikan	Analisis Kualitas
AI Biasa (Tanpa Data)	"Siapa yang harus menyerang musuh nomor 1?"	"Sebaiknya gunakan pemain terkuat Anda dengan strategi Naga Elektro. Pastikan *Hero* Anda maksimal."	Terlalu Umum. AI tidak tahu siapa pemain terkuat di klan Anda, jadi sarannya tidak bisa langsung dieksekusi (*Not Actionable*).
AI ClashHub (Dengan Data)	"Siapa yang harus menyerang musuh nomor 1?"	"Berdasarkan riwayat perang, Pemain Z memiliki tingkat keberhasilan 90% menggunakan serangan udara. Karena musuh nomor 1 memiliki Pertahanan Udara level rendah, disarankan Pemain Z yang menyerang."	Sangat Berguna. AI menganalisis data statistik spesifik yang dikirim sistem untuk memberikan rekomendasi yang presisi dan beralasan logis.

Sumber: Hasil Olahan Peneliti (2025)

Peran sebagai Mentor Digital

Selain membantu pemimpin dalam perang, fitur AI juga terbukti berfungsi sebagai "Guru Privat" bagi pemain pemula yang ingin belajar strategi (*Meta Learning*).

a. Personalisasi Saran: AI mampu membaca level *Town Hall* penanya. Jika pemain TH11 bertanya "Cara menyerang TH13?", AI akan menyarankan strategi *Scout* (mengintai) atau serangan dua bintang yang aman, bukan menyuruh meratakan markas lawan yang mustahil dilakukan.
b. Penyederhanaan Konsep: AI dapat menjelaskan istilah-istilah rumit seperti *Funneling* atau *Queen Walk* dengan bahasa Indonesia yang sederhana sesuai instruksi *prompt* sistem.

Implementasi AI pada ClashHub berhasil membuktikan bahwa teknologi Large Language Model (*LLM*) dapat diterapkan secara efektif untuk kebutuhan komunitas *grassroots*, asalkan didukung dengan *Prompt Engineering* yang tepat untuk membatasi ruang lingkup jawaban.

4.5.3 Solusi Masalah Kepercayaan dan Keamanan

Selain masalah teknis operasional, penelitian ini juga berhasil menjawab tantangan sosial terbesar di komunitas gim online, yaitu krisis kepercayaan (*trust issue*). Sering terjadi kasus penipuan akun (joki) atau pemain yang berperilaku buruk (*toxic*) namun bisa bebas berpindah klan. Sistem "ClashHub" menghadirkan solusi teknis untuk menciptakan lingkungan yang lebih aman dan transparan melalui tiga pilar utama:

Validasi Token *API* sebagai Bukti Kepemilikan Mutlak

Sebelum adanya sistem ini, proses rekrutmen sering terkendala oleh penipuan identitas. Seseorang bisa saja mengaku sebagai pemilik akun level tinggi hanya dengan mengirimkan gambar tangkapan layar (*screenshot*) yang sebenarnya milik orang lain atau hasil editan. Fitur verifikasi ClashHub mengubah cara ini secara mendasar:

a. Mekanisme: Pengguna wajib memasukkan kode token rahasia yang hanya bisa dilihat di pengaturan gim asli.
b. Dampak: Status "Terverifikasi" (Centang Biru) di profil ClashHub menjadi bukti mutlak kepemilikan akun.
c. Manfaat Turnamen: Fitur ini secara otomatis mengeliminasi praktik *Smurfing* (pemain *pro* menyamar jadi pemula), karena sistem menolak pendaftaran jika level akun asli tidak sesuai syarat turnamen.

Sistem Reputasi Anti-Manipulasi

Masalah "Kutu Loncat" (*Hopper*) sangat sulit dilacak sebelumnya. Fitur Ulasan Pemain memberikan solusi berupa rekam jejak digital yang permanen. Keunggulan teknis fitur ini terletak pada Logika Validasi Interaksi:

a. Cek Riwayat Bersama: Sistem otomatis menolak ulasan jika pelapor dan terlapor tidak pernah berada dalam satu klan yang sama.
b. Manfaat: Ini mencegah serangan ulasan palsu (*review bombing*) dari orang iseng atau *bot*. Ulasan yang muncul dijamin berasal dari interaksi nyata.
c. Cek Riwayat Bersama: Sistem otomatis menolak ulasan jika pelapor dan terlapor tidak pernah berada dalam satu klan yang sama. Ini mencegah serangan ulasan palsu (*review bombing*) dari orang iseng.

d. Basis Data *Sorting*: Skor bintang yang dihasilkan dari sistem ini menjadi landasan data bagi fitur "Pencarian Tim dengan *Sorting* Reputasi". Tanpa sistem reputasi yang valid, fitur pengurutan tersebut tidak akan berguna.

Salah satiu testimoni pakar (*KOL*-05) adalah "Fitur ini membuat pemain lebih berhati-hati menjaga perilaku mereka. Jejak buruk di satu klan akan terlihat oleh klan lain di masa depan, membuat ekosistem komunitas jadi lebih sehat."

Keamanan Data Berlapis (*Defense in Depth*)

Kepercayaan pengguna juga dibangun melalui jaminan keamanan data aplikasi itu sendiri. Sistem menerapkan standar industri untuk melindungi privasi:

a. Sesi Aman: Menggunakan *HTTP-Only Cookies* yang tidak bisa diakses oleh kode *browser* sembarangan, sehingga akun pengguna aman dari pencurian sesi (*Session Hijacking*).
b. Isolasi Logika Server: Aturan krusial (seperti menaikkan jabatan atau menghapus data klan) dieksekusi di sisi server dengan validasi peran yang ketat, sehingga tidak bisa diakali oleh pengguna yang mencoba memanipulasi kode aplikasi (*Client-Side Tampering*).

Kombinasi antara validasi identitas yang ketat (*API Token*) dan sistem reputasi yang jujur menjadikan ClashHub sebagai platform yang tidak hanya canggih secara fitur, tetapi juga Terpercaya (*Trusted*) bagi komunitas.

4.5.4 Keterbatasan Sistem

Meskipun sistem "ClashHub" telah berhasil diimplementasikan dan tervalidasi, terdapat beberapa batasan teknis yang perlu diakui sebagai ruang lingkup (*scope*) penelitian saat ini. Pemaparan ini bertujuan memberikan gambaran realistis mengenai kapabilitas sistem.

Ketergantungan pada Layanan Pihak Ketiga (*API Dependency*)

a. Ketersediaan Server: Jika server gim *Clash of Clans* sedang dalam pemeliharaan (*maintenance*), fitur utama seperti "Sinkronisasi Data" dan "Verifikasi Profil" akan terhenti total. Sistem tidak memiliki kendali atas infrastruktur Supercell.
b. Jeda Waktu Data (*Latency*): Akibat kebijakan *Rate Limiting* dan strategi *Caching* untuk menghemat kuota, data yang tampil di dasbor mungkin memiliki jeda waktu (*delay*) sekitar 1-5 menit dibandingkan kondisi *real-time* di dalam gim.

Batasan Kemampuan Analisis AI

Fitur "Asisten Strategi" saat ini masih memiliki keterbatasan dalam hal *input* data:

a. Buta Visual: AI Gemini yang diintegrasikan hanya bekerja berdasarkan data teks dan angka (*JSON*). AI belum memiliki kemampuan visi komputer (*Computer Vision*) untuk menganalisis tangkapan layar (*screenshot*) markas musuh.
b. Saran Taktis Makro: Rekomendasi yang diberikan bersifat strategi umum (*Macro Strategy*), seperti "Gunakan Naga Elektro dari arah Jam 3". AI belum mampu memberikan panduan mikro detik-per-detik yang sangat presisi.

Validasi Skor Turnamen Semi-Otomatis

Meskipun pembuatan bagan sudah otomatis, validasi hasil pertandingan masih bersifat semi-manual.

a. Penyebab: *API* resmi *Clash of Clans* saat ini belum menyediakan *endpoint* publik untuk mengambil hasil skor "Perang Persahabatan" (*Friendly War*) secara langsung.
b. Dampak: Sistem harus mengandalkan mekanisme pelaporan mandiri (*Self-Service*) dari peserta dengan bukti foto, yang masih menyisakan sedikit celah untuk sengketa hasil.

Ketergantungan Koneksi Internet

Sebagai aplikasi *web* modern (*Single Page Application*), sistem membutuhkan koneksi internet yang stabil untuk memuat data. Fitur "Mode Luring" (*Offline Mode*) belum diimplementasikan, sehingga pemimpin klan tidak dapat mengakses arsip data saat berada di area tanpa sinyal.

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penyebutan dengan nomor untuk level 1 menggunakan huruf abjad kecil. Posisi nomor dimulai dari margin kiri, seperti di bawah ini:

Berdasarkan seluruh rangkaian penelitian dan pengembangan sistem "ClashHub" yang menerapkan metode *Prototype*, dapat disimpulkan bahwa aplikasi ini berhasil menjawab seluruh rumusan masalah yang ditetapkan. Sistem terbukti efektif sebagai solusi teknologi untuk mengatasi inefisiensi manajemen dan krisis kepercayaan dalam komunitas *Clash of Clans* di Indonesia. Secara spesifik, kesimpulan yang diperoleh dari penelitian ini adalah:

a. Efektivitas Pencarian Tim Presisi (Menjawab Masalah Rekrutmen)
   Sistem berhasil menyediakan mekanisme pencarian klan dan pemain yang jauh lebih akurat dibandingkan fitur bawaan gim. Implementasi algoritma Penyaringan (*Filtering*) dan Pengurutan (*Sorting*) berdasarkan parameter Skor Reputasi dan Level *Town Hall* terbukti memudahkan pengguna menemukan komunitas yang selaras dengan visi bermain mereka (Kompetitif vs Santai).
b. Peningkatan Efisiensi Manajemen Berbasis Data (Menjawab Masalah *Admin*)
   Penerapan fitur Sinkronisasi Data Otomatis (*Smart Sync*) berhasil memangkas waktu administrasi pemimpin klan secara signifikan. Pemimpin yang sebelumnya menghabiskan waktu berjam-jam untuk rekapitulasi manual di Excel, kini dapat melihat agregasi data statistik (Donasi, Perang, Keaktifan) dalam hitungan detik yang diambil langsung dari *API* resmi Supercell.
c. Validitas Keputusan Strategis (Menjawab Masalah Subjektivitas)
   Fitur Asisten Strategi yang mengintegrasikan teknologi *Generative AI* (Google Gemini) terbukti mampu berfungsi sebagai pendukung keputusan yang efektif. Dengan teknik penyuntikan data fakta klan (*Context Injection*), AI mampu memberikan narasi saran strategi yang spesifik dan logis, membantu pemimpin klan mengambil keputusan perang yang lebih objektif.
d. Solusi Krisis Kepercayaan (Menjawab Masalah *Trust Issue*)
   Sistem berhasil menciptakan ekosistem rekrutmen yang transparan dan aman. Fitur Verifikasi Token *API* efektif menghilangkan praktik pemalsuan akun (*Anti-Smurfing*)

dalam turnamen. Selain itu, Sistem Reputasi yang membatasi penilaian hanya bagi pemain yang memiliki riwayat interaksi sah, terbukti valid dalam mencegah ulasan palsu dan mendeteksi perilaku "Kutu Loncat" (*Hopper*).

e. Tingkat Penerimaan Pengguna Sangat Baik

Berdasarkan pengujian usabilitas menggunakan metode *System Usability Scale* (*SUS*), sistem memperoleh skor rata-rata 86.2 yang masuk dalam kategori "*Excellent*" (Sangat Baik). Hal ini mengindikasikan bahwa desain antarmuka *Mobile-First* yang diterapkan sangat sesuai dengan karakteristik pemain gim yang mengutamakan kemudahan akses melalui perangkat seluler.

5.2 Saran Pengembangan

Penelitian ini merupakan langkah awal dalam digitalisasi manajemen komunitas *e-sports grassroots*. Berdasarkan evaluasi terhadap keterbatasan sistem dan potensi teknologi masa depan, penulis mengajukan saran konstruktif untuk pengembangan sistem "ClashHub" selanjutnya:

a. Integrasi *Optical Character Recognition* (*OCR*) untuk Skor Turnamen

Saat ini, validasi hasil pertandingan masih mengandalkan pelaporan mandiri dengan bukti foto yang diperiksa manual oleh panitia.

1. Saran: Mengembangkan modul *Computer Vision* berbasis *OCR* (seperti Tesseract.js atau Google *Cloud Vision*) yang mampu membaca angka skor dan persentase kerusakan langsung dari tangkapan layar (*screenshot*) hasil perang.
2. Manfaat: Otomatisasi penuh pada administrasi turnamen, menghilangkan celah manipulasi data atau kesalahan *input* manusia.

b. Peningkatan Algoritma Reputasi Terbobot (*Weighted Reputation*)

Sistem saat ini menggunakan perhitungan rata-rata aritmatika sederhana untuk skor reputasi.

1. Saran: Mengimplementasikan algoritma pembobotan dinamis. Ulasan yang diberikan oleh pengguna dengan status "Terverifikasi" atau "*Leader* Senior" seharusnya memiliki bobot nilai yang lebih tinggi dibandingkan akun baru.
2. Manfaat: Meningkatkan akurasi sistem kepercayaan (*Trust System*) dan meminimalisir dampak dari akun palsu (*buzzer*) yang mencoba merusak reputasi klan/pemain.

c. Implementasi Arsitektur Antrean (*Message Broker*)

Ketergantungan pada *API* pihak ketiga dengan batasan akses (*Rate Limit*) menjadi kendala skalabilitas jika pengguna mencapai ribuan.

1. Saran: Menerapkan mekanisme antrean pesan (*Job Queue*) menggunakan teknologi seperti Redis atau RabbitMQ. Permintaan sinkronisasi data tidak diproses secara langsung (sinkron), melainkan dimasukkan ke dalam antrean (asinkron) untuk dieksekusi satu per satu sesuai kapasitas *API*.
2. Manfaat: Mencegah server mengalami *timeout* atau pemblokiran IP saat terjadi lonjakan trafik di akhir musim liga.

d. Pengembangan Aplikasi Seluler Asli (*Native App*)

Meskipun desain *web* saat ini sudah *Mobile-First*, keterbatasan teknologi *web* adalah tidak adanya akses penuh ke fitur perangkat keras.

1. Saran: Mengembangkan versi aplikasi *native* (Android/iOS) menggunakan kerangka kerja lintas platform seperti React *Native* atau Flutter.
2. Manfaat: Memungkinkan fitur Notifikasi Dorong (*Push Notification*) secara *real-time*. Pemimpin klan bisa mendapatkan peringatan langsung di layar kunci HP jika ada anggota yang belum menyerang saat waktu perang tinggal 1 jam lagi.

DAFTAR PUSTAKA

ActivePlayer.io. (2025). *Clash of Clans Live Player* Count and Statistics. Diakses dari https://activeplayer.io/clash-of-clans/

Arham, A. M. F. M., Sholva, Y., & Muthahhari, M. (2024). Sistem Informasi Turnamen *E-Sports* dan Warung Kopi *Gaming* di Kota Pontianak Berbasis *Website*. *Scientica: Jurnal Ilmiah Sains dan Teknologi*, *2*(9), 346–376.

Bangor, A., Kortum, P. T., & Miller, J. T. (2009). An Empirical Evaluation of the *System Usability Scale*. *International Journal of Human-Computer Interaction*, *24*(6), 574-594.

Brooke, J. (1996). *SUS*-A *quick and dirty usability* scale. *Usability Evaluation in Industry*, *189*(194), 4-7.

Google *Cloud*. (2024). *Generative AI* on Vertex AI Documentation. Diakses dari https://*cloud*.google.com/vertex-ai

Google Developers. (2024). *Cloud* Firestore Data Modeling Guide. Diakses dari https://firebase.google.com/docs/firestore/manage-data/structure-data

Google Developers. (2024). *Cloud* Firestore Documentation. Diakses dari https://firebase.google.com/docs/firestore

Google Developers. (2024). Firebase Documentation. Diakses dari https://firebase.google.com/docs

Google Developers. (2024). *Web* Fundamentals: *UX* Design. Diakses dari https://developers.google.com/*web*/fundamentals/design-and-*ux*

Google Developers. (2024). *Web* Vitals: Essential metrics for a healthy site. Diakses dari https://*web*.dev/vitals/

Google Developers. (2024). YouTube Data *API* Reference. Diakses dari https://developers.google.com/youtube/v3

Hamari, J., & Sjöblom, M. (2017). What is *eSports* and why do people *watch* it? *Internet Research*, *27*(2), 211–232.

Holland, T. J. (2025). Explaining *Toxicity* in Multiplayer Games. *DiGRA Australia Extended Abstract*.

Jarvenpaa, S. L., & Leidner, D. E. (1999). *Communication* and *Trust* in Global Virtual Teams. *Organization Science*, *10*(6), 791-815.

Kementerian Komunikasi dan Informatika. (2024). *Laporan Perkembangan Industri Gim Nasional 2024*. Jakarta: Kominfo.

Kementerian Komunikasi dan Informatika. (2024). *Peluang dan Tantangan Industri Gim Nasional*. Jakarta: Kementerian Komunikasi dan Informatika Republik Indonesia.

Laudon, K. C., & Laudon, J. P. (2021). *Management Information Systems: Managing the Digital Firm* (17th ed.). Pearson.

Lee, J., et al. (2025). Less Talk, More *Trust*: Understanding Players' *In-game* Assessment of *Communication* Processes. *Proceedings of the CHI Conference on Human Factors in Computing Systems*.

Liquipedia. (2025). *Clash of Clans World Championship* 2025. Diakses dari https://liquipedia.net/clashofclans

Microsoft. (2024). TypeScript Documentation: The Handbook. Diakses dari https://www.typescriptlang.org/docs/

Mozilla Developer Network. (2024). JavaScript Timing Events. Diakses dari https://developer.mozilla.org/en-US/docs/*Web*/*API*/*setInterval*

Newzoo. (2024). *Global Games Market Report 2024*. Amsterdam: Newzoo.

Nidhra, S., & Dondeti, J. (2012). Black Box and White Box Testing Techniques - A Literature *Review*. *International Journal of Embedded Systems and Applications (IJESA)*, *2*(2), 29-50.

OWASP. (2024). *OWASP Top 10: Session Management Cheat Sheet*. Diakses dari https://owasp.org/

Pérez-Rubio, C., et al. (2024). Taking aim at research on *esports* teams: a systematic literature *review* and cross-disciplinary future agenda. *Team Performance Management: An International Journal*. Emerald Publishing Limited.

Pratama, A., & Wibowo, S. (2021). Rancang Bangun Sistem Informasi Manajemen Turnamen *E-Sports* Berbasis *Web*. *Jurnal Teknologi Informasi dan Ilmu Komputer*, *8*(2), 210-218.

Pressman, R. S., & Maxim, B. R. (2015). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill Education.

Resnick, P., & Zeckhauser, R. (2002). *Trust* Among Strangers in Internet Transactions: Empirical Analysis of eBay's *Reputation* System. *The Economics of the Internet and E-commerce*, *11*, 127-157.

Rosa, A. S., & Shalahuddin, M. (2018). *Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek*. Bandung: Informatika.

Sauro, J. (2011). *A Practical Guide to the System Usability Scale: Background, Benchmarks & Best Practices*. Measuring *Usability* LLC.

Sharda, R., Delen, D., & Turban, E. (2020). *Analytics, Data Science, & Artificial Intelligence: Systems for Decision Support* (11th ed.). Pearson.

Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.

Supercell. (2024). *Clash of Clans API* Documentation. Diakses dari https://developer.clashofclans.com/

Supercell. (2024). *Clash of Clans API Terms of Service*. Diakses dari https://developer.clashofclans.com/

Supercell. (2024). *Clash of Clans User* Guide: *Clan* Roles and War Mechanics. Supercell Support Portal. Diakses dari https://support.supercell.com

Tailwind Labs. (2024). Tailwind *CSS* Documentation. Diakses dari https://tailwindcss.com/docs

Tailwind Labs. (2024). Tailwind *CSS* Documentation: Aspect Ratio. Diakses dari https://tailwindcss.com/docs/aspect-ratio

Tailwind Labs. (2024). Tailwind *CSS* Documentation: *Grid Layout*. Diakses dari https://tailwindcss.com/docs/*grid*-template-columns

Tailwind Labs. (2024). Tailwind *CSS* Documentation: *Utility-First* Fundamentals. Diakses dari https://tailwindcss.com/docs

Toth, A., et al. (2024). An exploratory qualitative interview study on *grassroots esports* in sports clubs. *Frontiers in Sports and Active Living*, *6*, 1234567.

Vercel. (2024). Edge Network Regions and *Performance*. Diakses dari https://vercel.com/docs/edge-network/regions

Vercel. (2024). Next.js 14 Documentation: App Router and Server *Components*. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: App Router. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: *Client Components*. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: *Client Components* & Interactivity. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: Mutating Data. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: Optimizing Images. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: *Rendering* and *Serverless Functions*. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: Route Handlers & Server *Components*. Diakses dari https://nextjs.org/docs

Vercel. (2024). Next.js Documentation: Routing. Diakses dari https://nextjs.org/docs/app/building-your-application/routing

Vercel. (2024). Vercel *Deployment* Documentation. Diakses dari https://vercel.com/docs

Wagner, M. G. (2006). On the Scientific Relevance of *eSports*. In *Proceedings of the 2006 International Conference on Internet Computing & Conference on Computer Games Development* (pp. 437-442).

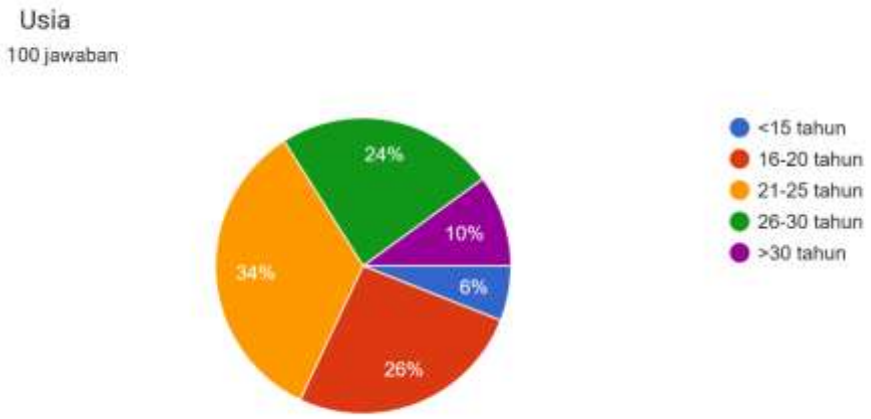
We Are Social. (2024). *Digital 2024: Indonesia*. Jakarta: We Are Social & Kepios.

Xu, I. (2025). *Toxic* Behaviour in Online Multiplayer Games: To Play or to Flame? (Bachelor Thesis). Tilburg University.

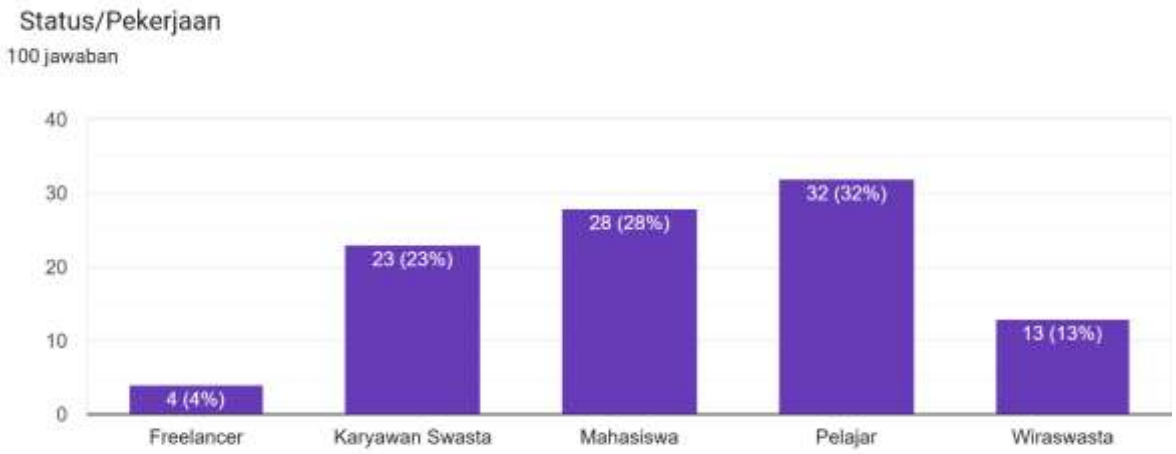
Yunanri, W., & Measer, A. (2022). Sistem Informasi Manajemen *Event* Electronic Sport (*E-Sport*) Berbasis *Web* pada Komunitas Esport Indonesia Wilayah Kabupaten Sumbawa. *Jurnal Manajemen Informatika dan Sistem Informasi*, *5*(2), 109-115.

LAMPIRAN

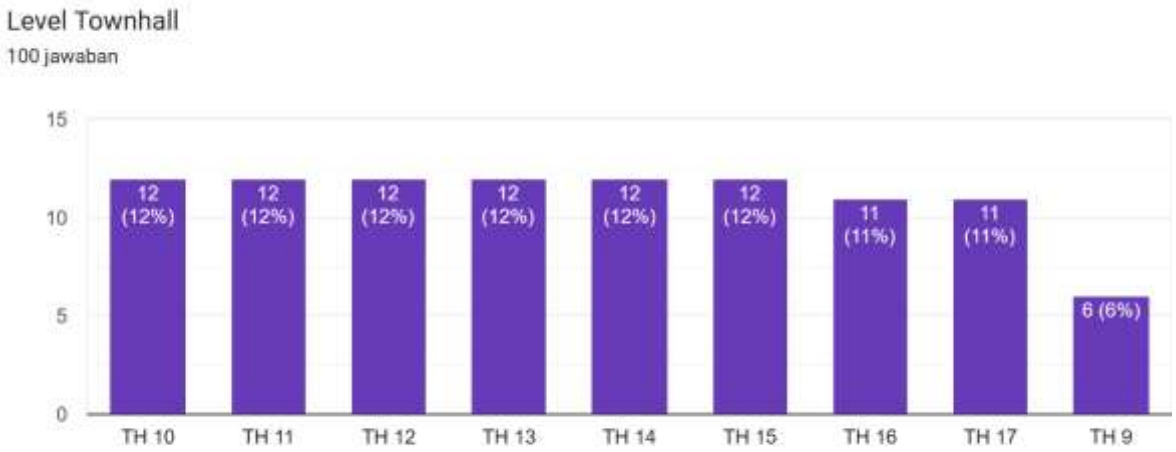
LAMPIRAN A: Wawancara Kebutuhan Pengguna Terkait Usia



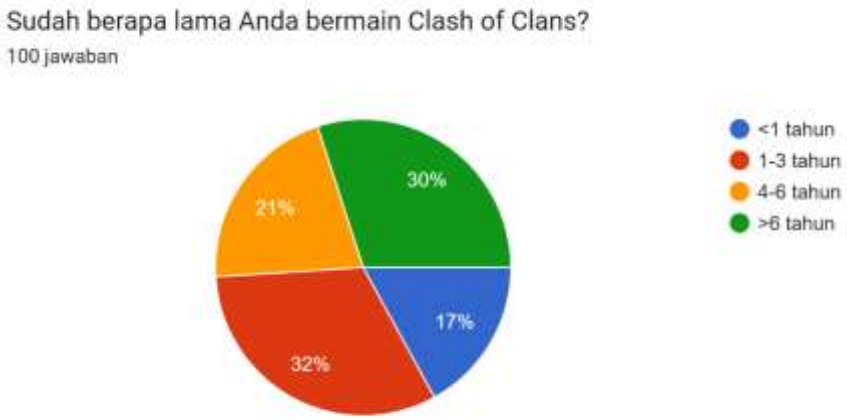
LAMPIRAN B: Wawancara Kebutuhan Pengguna Terkait Status/Pekerjaan



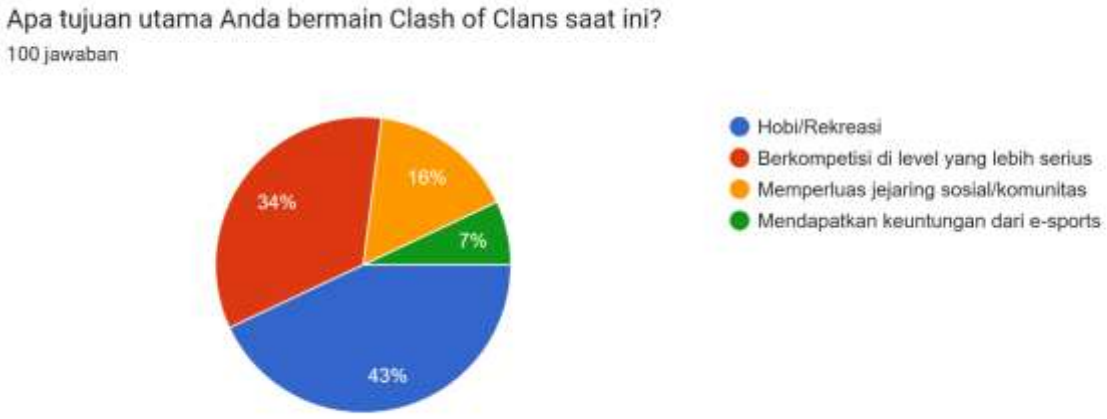
LAMPIRAN C: Wawancara Kebutuhan Pengguna Terkait Level Townhall



LAMPIRAN D: Wawancara Kebutuhan Pengguna Terkait Durasi Bermain CoC



LAMPIRAN E: Wawancara Kebutuhan Pengguna Terkait Motivasi Utama Bermain

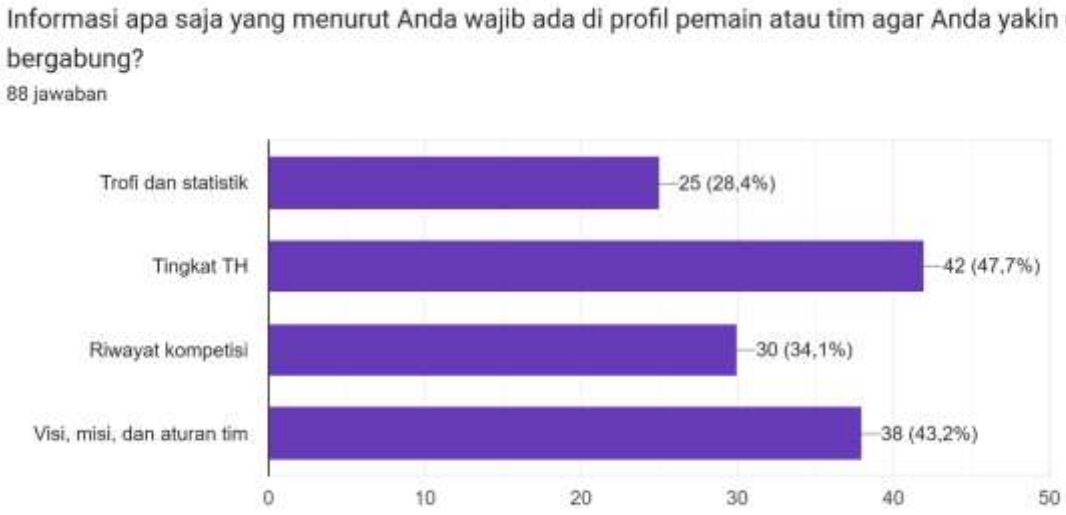


LAMPIRAN F: Wawancara Kebutuhan Pengguna Terkait Minat Gabung/Bentuk Tim *E-sports*

LAMPIRAN G: Wawancara Kebutuhan Pengguna Terkait Hambatan Rekrutmen Tim

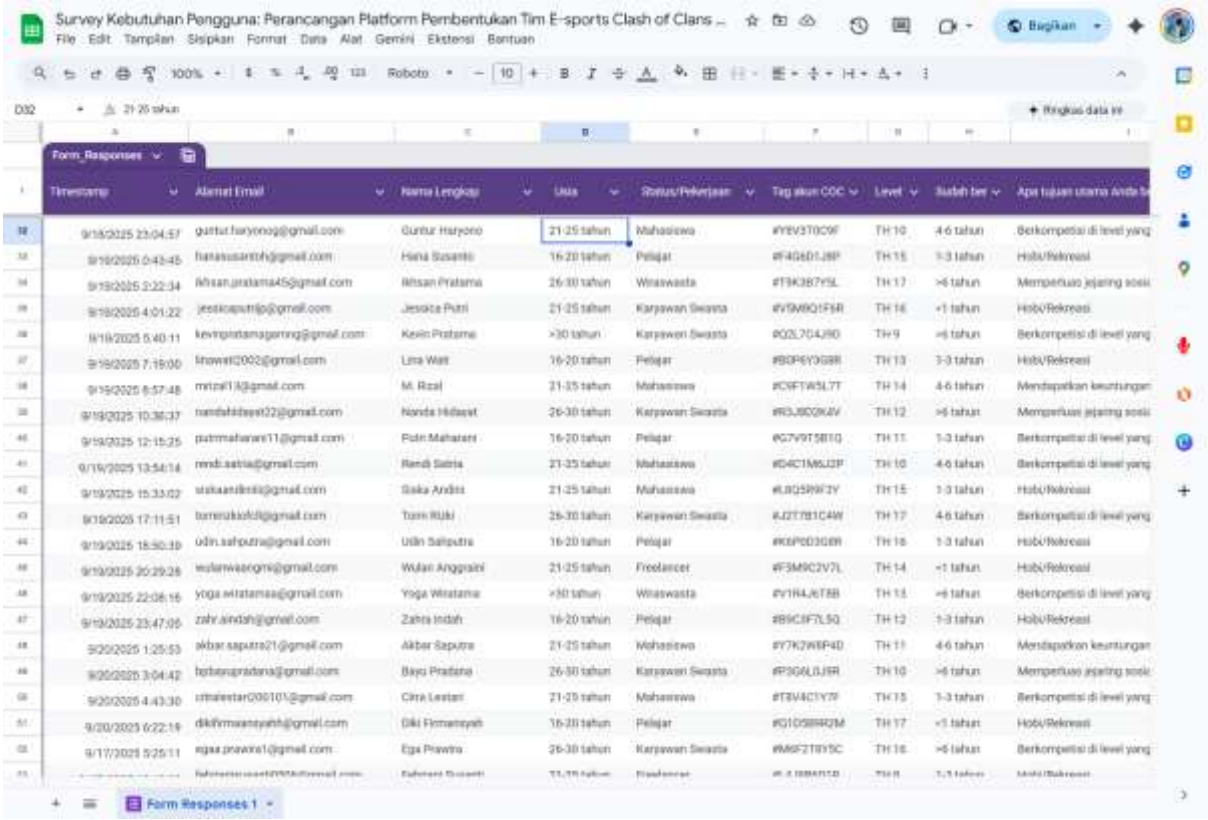
LAMPIRAN H: Wawancara Kebutuhan Pengguna Terkait Sumber Informasi/Anggota Tim

LAMPIRAN I: Wawancara Kebutuhan Pengguna Terkait Kriteria Profil Pemain/Tim

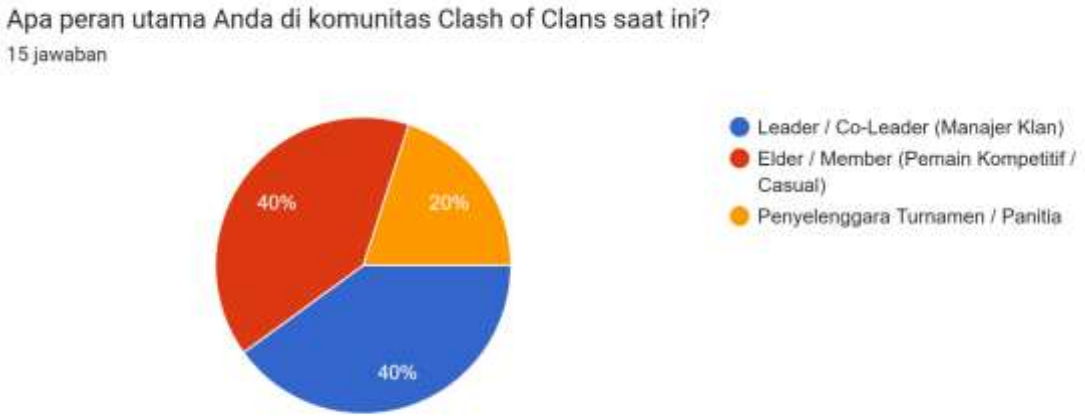


LAMPIRAN J: Wawancara Kebutuhan Pengguna Terkait Urgensi Sistem Reputasi/*Rating*

LAMPIRAN K: Data Partisipan Survey Kebutuhan Pengguna

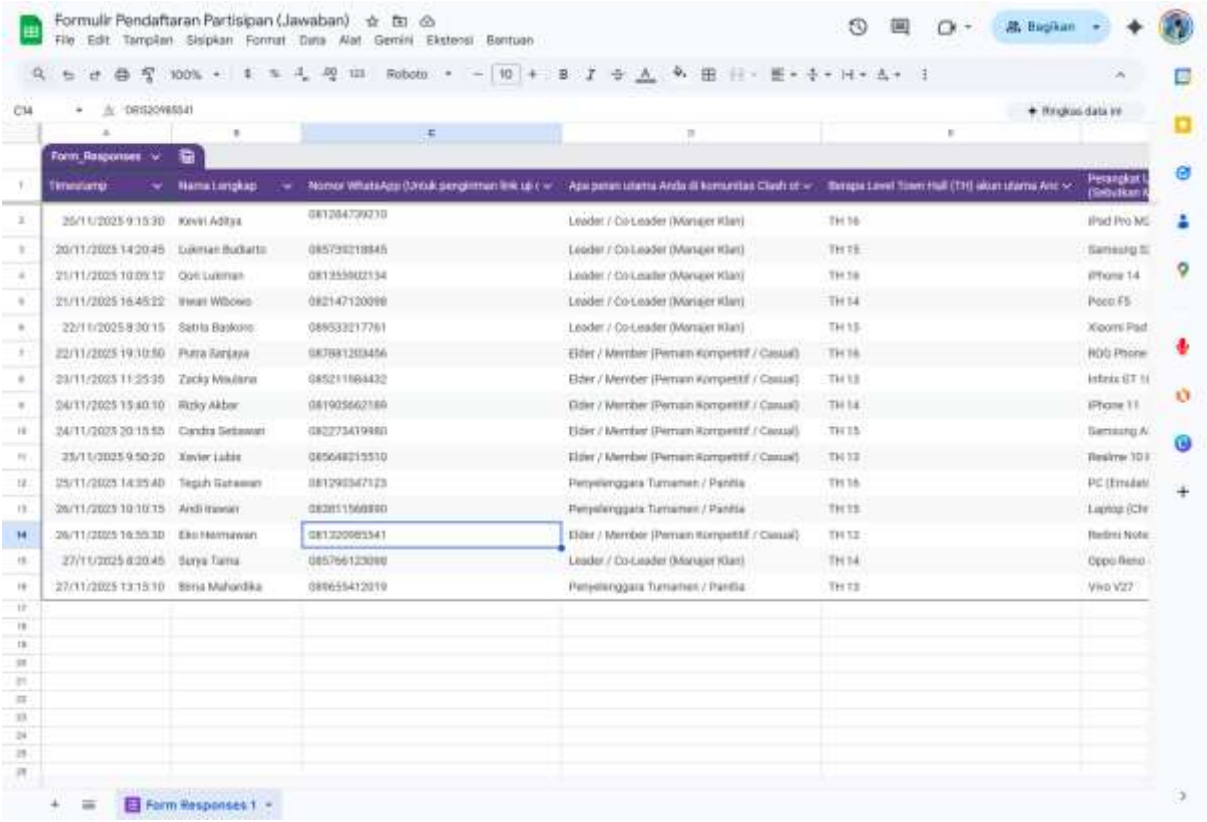


LAMPIRAN L: Partisipan Kategori *Role* di *Clan*



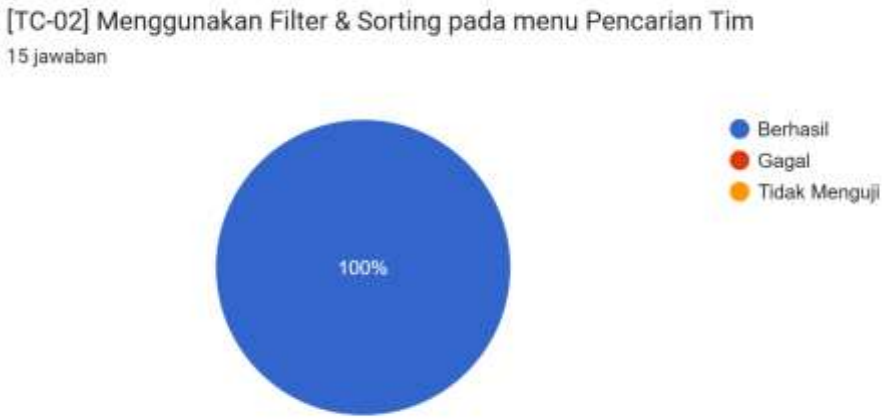
LAMPIRAN M: Partisipan Kategori Level *Town Hall* (*TH*)

LAMPIRAN N: Data Partisipan yang Dipilih Sebanyak 15 Partisipan

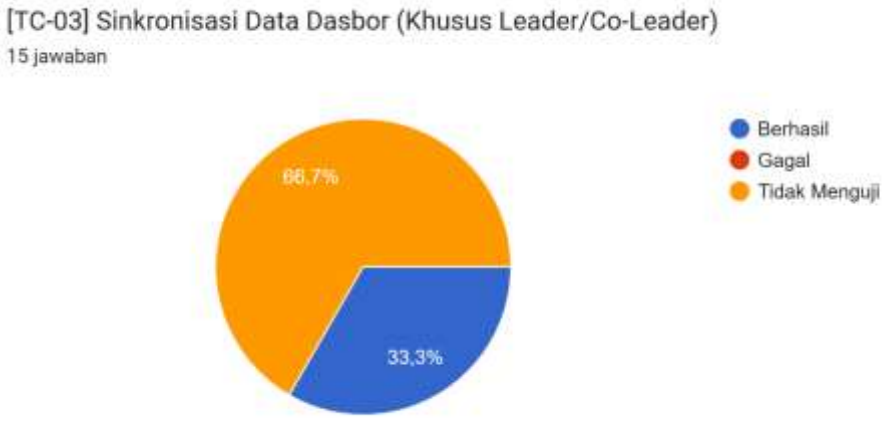


LAMPIRAN O: Pengujian Fungsional (Black Box) Terkait Efektivitas Verifikasi Token *API*

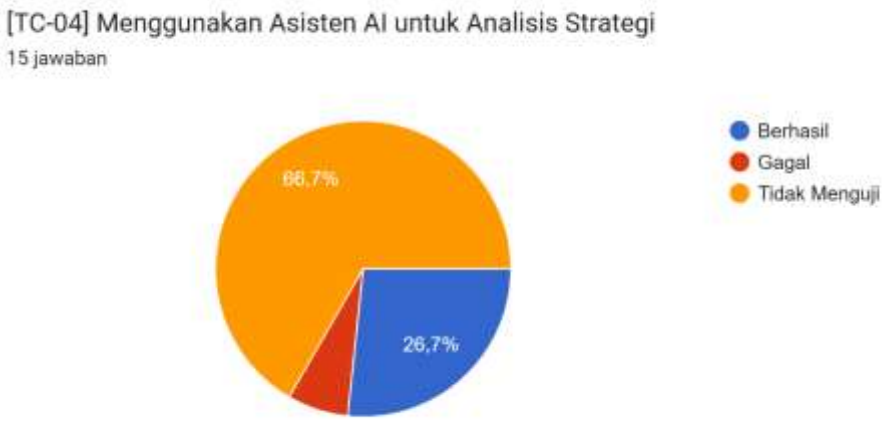
LAMPIRAN P: Pengujian Fungsional (Black Box) Terkait Kegunaan Filter & *Sorting* Pencarian



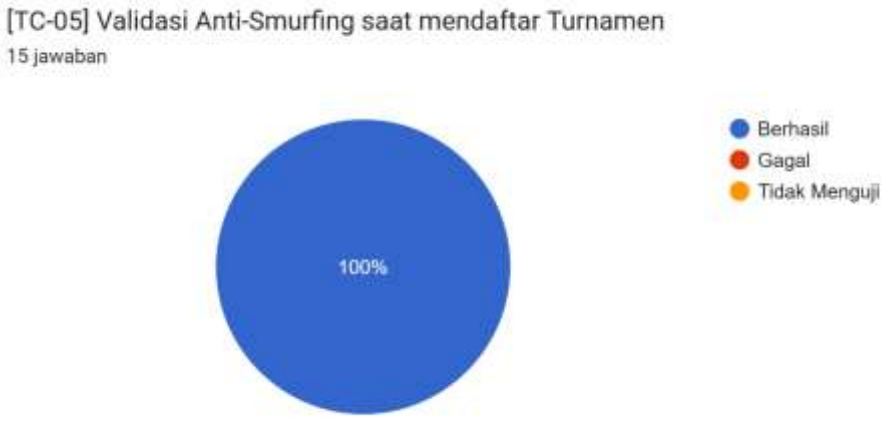
LAMPIRAN Q: Pengujian Fungsional (Black Box) Terkait Relevansi Sinkronisasi Dasbor *Leader*



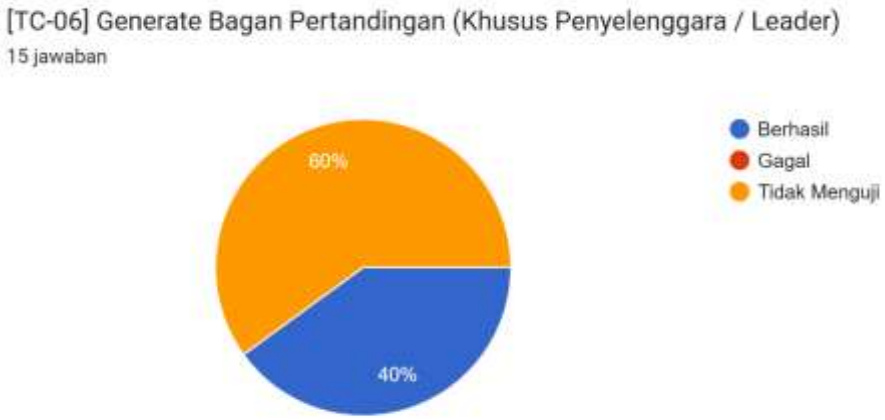
LAMPIRAN R: Pengujian Fungsional (Black Box) Terkait Adopsi Asisten AI Strategi



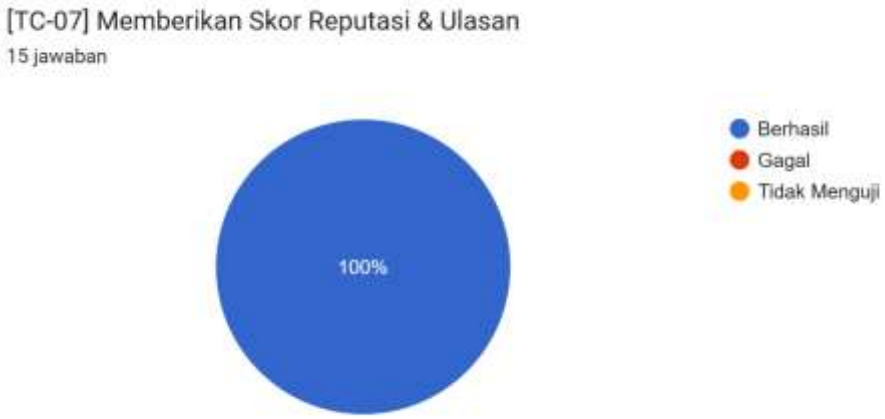
LAMPIRAN S: Pengujian Fungsional (Black Box) Terkait Urgensi Validasi *Anti-Smurfing*



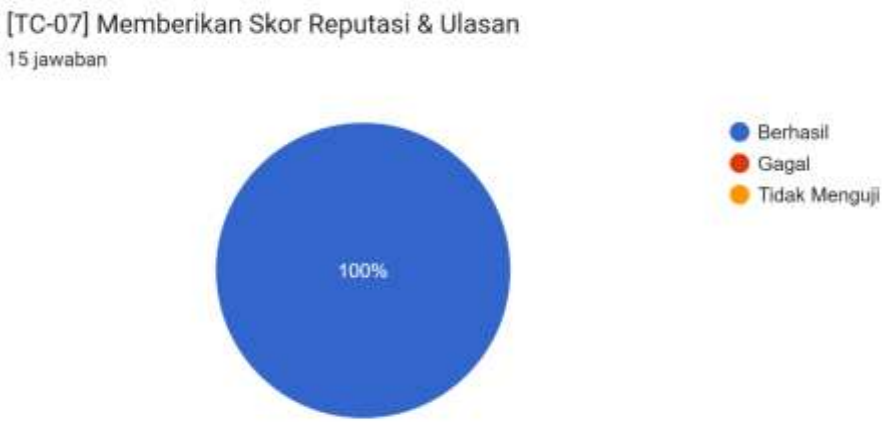
LAMPIRAN T: Pengujian Fungsional (Black Box) Terkait Efisiensi Generator Bagan Turnamen



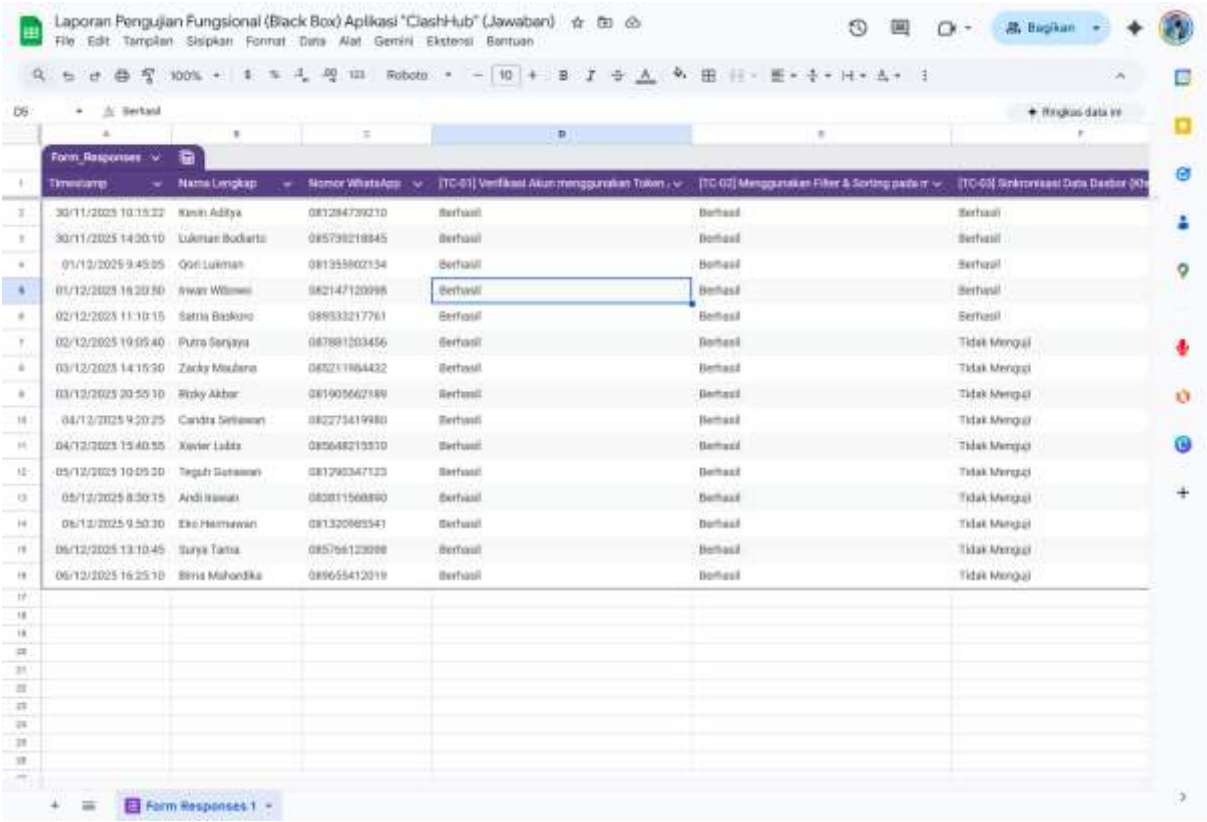
LAMPIRAN U: Pengujian Fungsional (Black Box) Terkait Penerimaan Sistem Skor & Ulasan



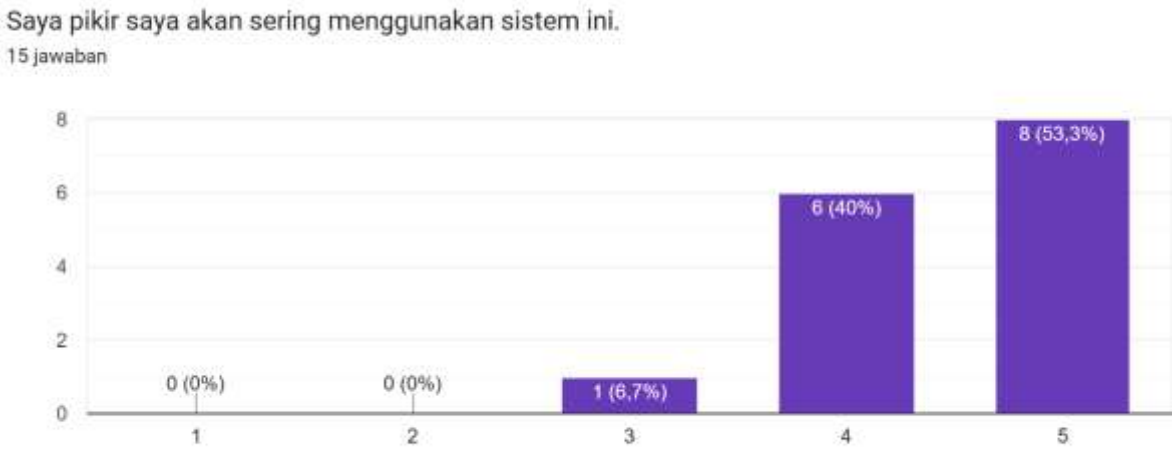
LAMPIRAN V: Pengujian Fungsional (Black Box) Terkait Minat Fitur Postingan Video YouTube



LAMPIRAN W: Data Pengujian Fungsional (Black Box) dari 15 Partisipan

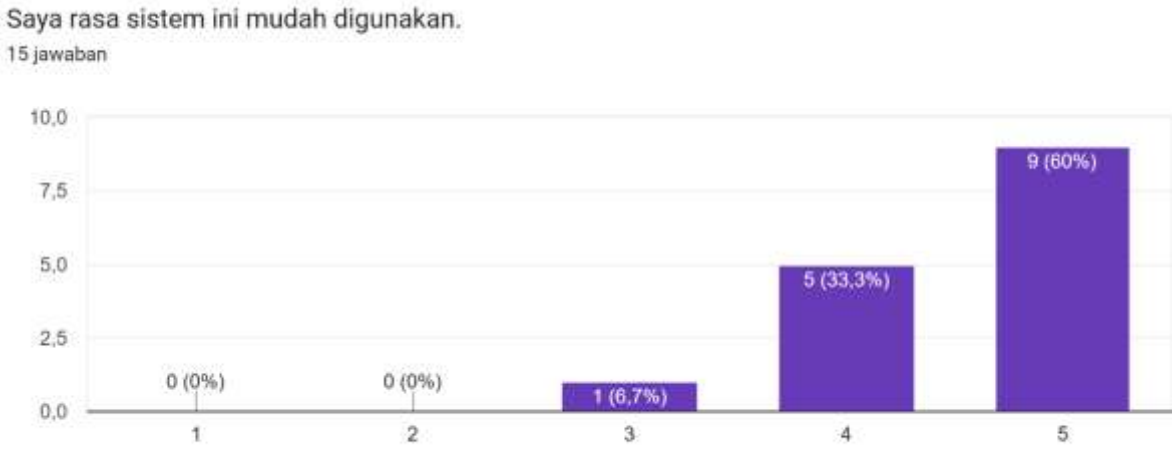


LAMPIRAN X: Pengujian *System Usability Scale* (*SUS*) Terkait Intensi Penggunaan Rutin



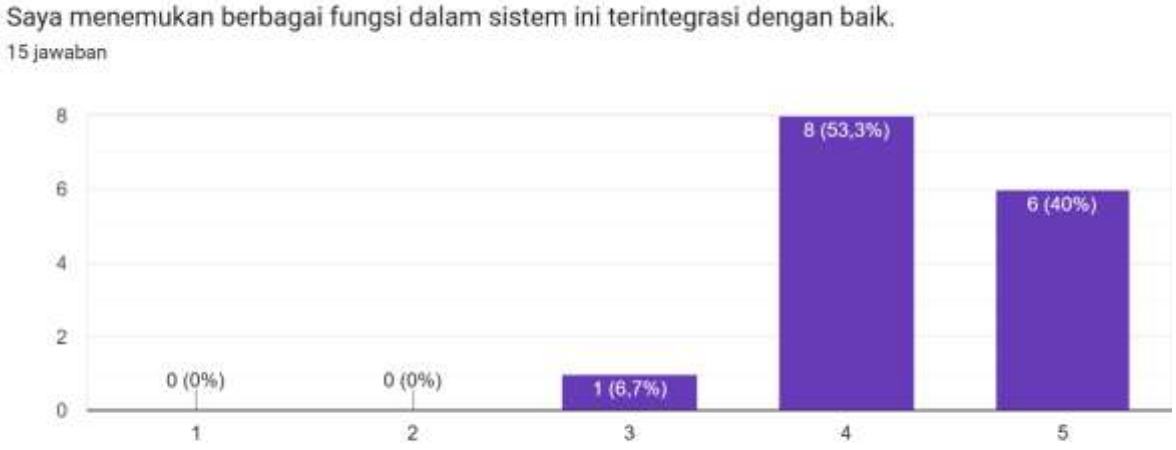
LAMPIRAN Y: Pengujian *System Usability Scale* (*SUS*) Terkait Tingkat Kompleksitas Sistem

LAMPIRAN Z: Pengujian *System Usability Scale* (*SUS*) Terkait Kemudahan Penggunaan (*Usability*)

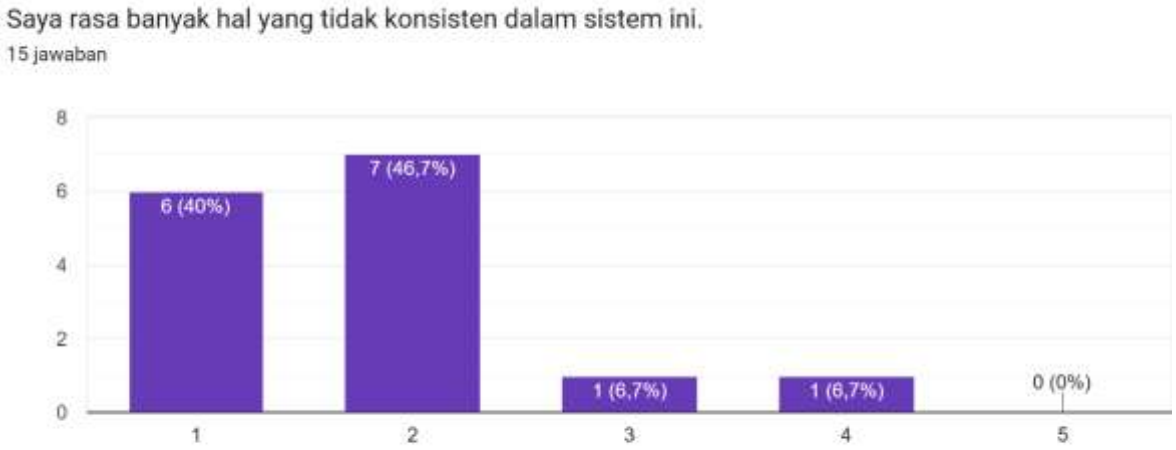


LAMPIRAN AA: Pengujian *System Usability Scale* (*SUS*) Terkait Kebutuhan Dukungan Teknis

LAMPIRAN AB: Pengujian *System Usability Scale* (*SUS*) Terkait Integritas Fitur/Fungsi



LAMPIRAN AC: Pengujian *System Usability Scale* (*SUS*) Terkait Konsistensi Sistem



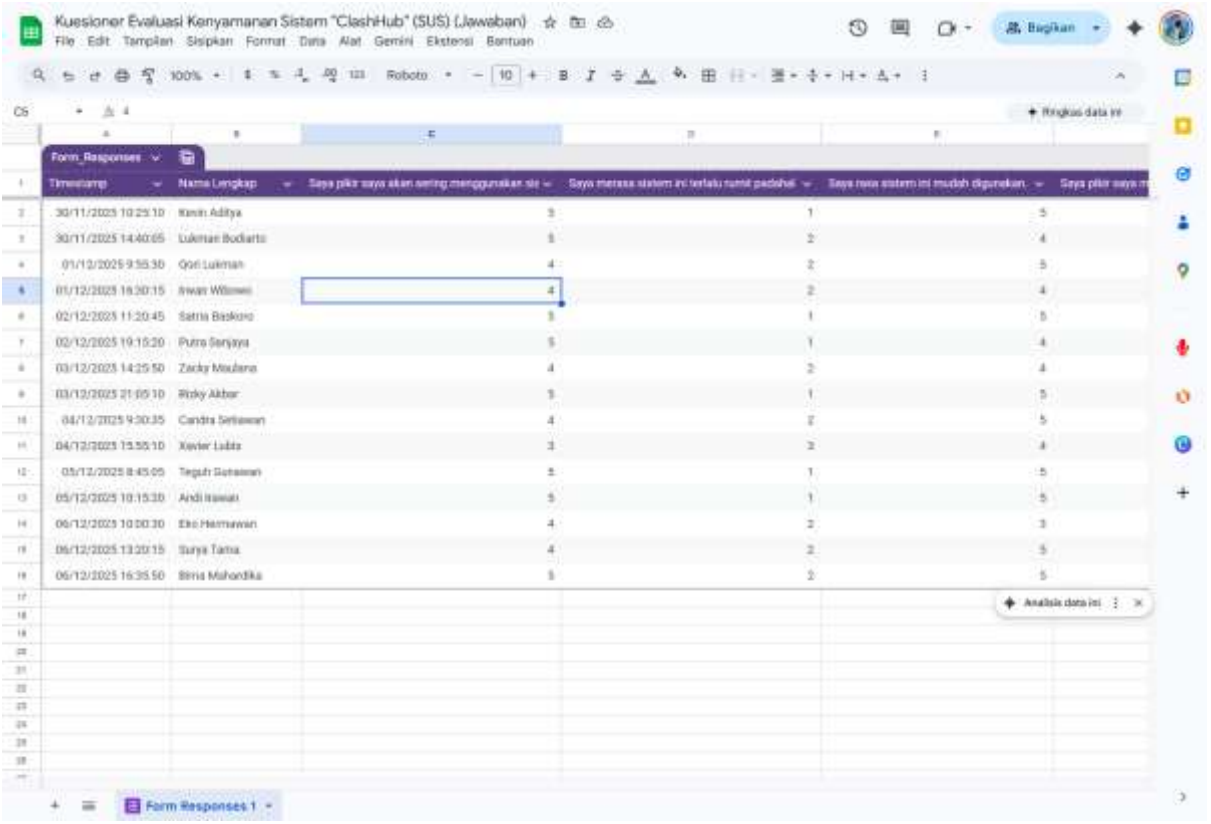
LAMPIRAN AD: Pengujian *System Usability Scale* (*SUS*) Terkait Kecepatan Adaptasi Pengguna

LAMPIRAN AE: Pengujian *System Usability Scale* (*SUS*) Terkait Tingkat Kebingungan Pengguna

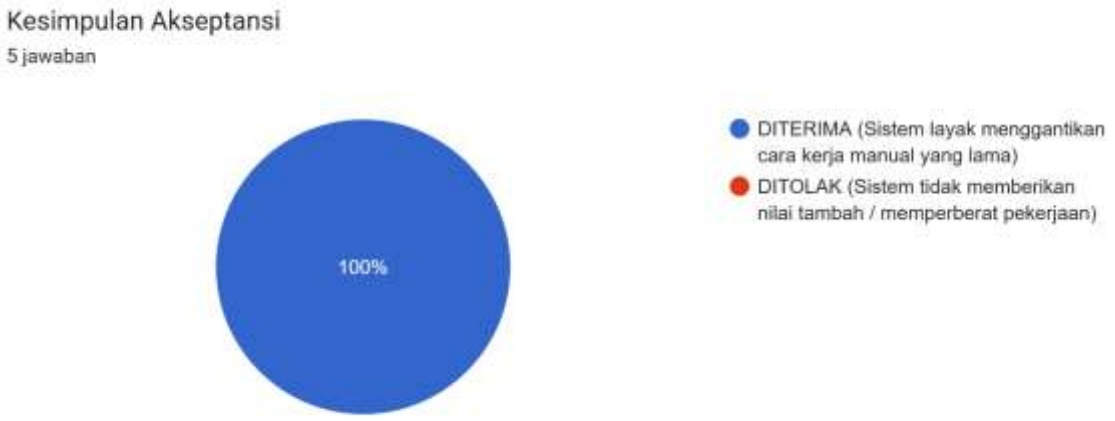
LAMPIRAN AF: Pengujian *System Usability Scale* (*SUS*) Terkait Keyakinan/Efikasi Pengguna

LAMPIRAN AG: Pengujian *System Usability Scale* (*SUS*) Terkait Kurva Pembelajaran (Learning Curve)

LAMPIRAN AH: Data Pengujian *System Usability Scale* (*SUS*) dari 15 Partisipan



LAMPIRAN AI: *User Acceptance Testing* (*UAT*) dari Pakar Komunitas untuk Kesimpulan Akseptansi



LAMPIRAN AJ: Data Validasi System *User Acceptance Testing* (*UAT*) dari 5 Partisipan