BAB II
KAJIAN LITERATUR

2.1 Aplikasi *Chat* Berbasis *Web*

Aplikasi *chat* berbasis *web* merupakan sistem komunikasi daring yang memungkinkan pengguna untuk saling bertukar pesan teks melalui antarmuka *web*. Sistem ini biasanya dibangun dengan teknologi *web* seperti HTML, CSS, dan JavaScript di sisi klien, serta bahasa pemrograman *server*-side seperti Node.js, PHP, atau Python untuk menangani pengolahan data dan komunikasi antar pengguna [12]. Untuk mempercepat proses desain antarmuka, *framework* CSS seperti Tailwind CSS sering digunakan karena pendekatan *utility-first*-nya yang fleksibel. Keunggulan utama aplikasi ini terletak pada aksesibilitasnya yang tinggi, karena pengguna tidak perlu mengunduh aplikasi tambahan dan cukup menggunakan peramban *(browser)* untuk terhubung.

Secara umum, aplikasi *chat* berbasis *web* terdiri atas dua komponen utama, yaitu klien dan *server*. Sisi klien bertugas sebagai antarmuka pengguna dan menangani interaksi secara langsung. Sementara itu, *server* berfungsi untuk menerima, memproses, dan meneruskan pesan yang dikirimkan dari satu pengguna ke pengguna lainnya. Agar komunikasi antar pengguna dapat berlangsung secara instan, sistem *chat* modern umumnya menggunakan protokol komunikasi seperti *WebSocket* yang mendukung koneksi *full-duplex*, memungkinkan data dikirim dan diterima secara bersamaan dalam satu jalur koneksi [11].

Saat ini, berbagai aplikasi *chat* populer seperti Slack, Rocket.Chat, dan WhatsApp *Web* telah memanfaatkan pendekatan berbasis *web*. Meskipun demikian, aplikasi-aplikasi tersebut umumnya masih menghadapi keterbatasan dalam hal dukungan terhadap komunikasi lintas bahasa secara otomatis. Misalnya, WhatsApp *Web* dan Telegram *Web* memungkinkan pengguna untuk saling bertukar pesan dengan cepat, tetapi tidak menyediakan fitur penerjemahan pesan secara otomatis tanpa mengandalkan aplikasi pihak ketiga [13]. Kekurangan ini menjadi tantangan tersendiri bagi pengguna yang berkomunikasi dalam bahasa yang berbeda.

Dalam konteks ini, pengembangan aplikasi *chat* berbasis *web* yang dilengkapi dengan fitur penerjemahan menjadi penting untuk mengatasi hambatan bahasa dalam komunikasi global. Hal ini sejalan dengan meningkatnya kebutuhan komunikasi multibahasa akibat globalisasi, mobilitas pelajar dan pekerja antarnegara, serta pertumbuhan sektor pariwisata internasional. Aplikasi *chat* yang memiliki kemampuan untuk secara otomatis

menerjemahkan pesan antar pengguna dalam bahasa yang berbeda akan memberikan nilai tambah yang signifikan, baik dalam konteks sosial maupun profesional [3].

2.2 Teknologi *WebSocket*

WebSocket merupakan salah satu protokol komunikasi berbasis *web* yang dirancang untuk menyediakan kanal komunikasi dua arah (*full-duplex*) antara klien dan *server* melalui satu koneksi TCP yang persisten. Berbeda dengan pendekatan HTTP yang bersifat *stateless* dan berbasis *request-response*, *WebSocket* memungkinkan pertukaran data secara langsung tanpa perlu membuka koneksi baru setiap kali data dikirim atau diterima [14]. Hal ini menjadikan *WebSocket* sangat ideal digunakan dalam aplikasi yang memerlukan latensi rendah dan komunikasi berkelanjutan, seperti sistem *chat*, *game* daring, dan layanan pemantauan data secara langsung.

Proses kerja *WebSocket* dimulai dengan *handshake* awal menggunakan protokol HTTP, setelah itu koneksi beralih menjadi protokol *WebSocket*. Selama koneksi tetap terbuka, klien dan *server* dapat saling mengirim data secara langsung tanpa perlu overhead tambahan dari protokol HTTP. Keunggulan ini tidak hanya mempercepat waktu respons, tetapi juga mengurangi beban jaringan secara keseluruhan [15]. Dalam konteks pengembangan aplikasi *chat* berbasis *web*, *WebSocket* menjadi komponen kunci untuk menghadirkan pengalaman komunikasi yang responsif dan interaktif. Dengan menggunakan *WebSocket*, setiap pesan yang diketik oleh pengguna dapat langsung dikirim ke *server* dan diteruskan ke pengguna lain tanpa jeda yang signifikan, menciptakan sensasi percakapan layaknya obrolan langsung [15].

Implementasi *WebSocket* biasanya dilakukan menggunakan *library* atau *framework* seperti *Socket.IO* untuk *Node.js*, yang mempermudah pengelolaan koneksi, autentikasi pengguna, serta penyiaran pesan ke berbagai klien secara bersamaan. Selain kecepatan, *WebSocket* juga menawarkan efisiensi dalam penggunaan sumber daya. Karena koneksi tidak perlu dibuka dan ditutup berulang kali, penggunaan CPU dan bandwidth menjadi lebih hemat, terutama saat volume pesan yang dipertukarkan sangat tinggi [16]. Namun demikian, *WebSocket* juga memiliki tantangan tersendiri, seperti kebutuhan pengelolaan koneksi yang stabil serta keamanan data yang dipertukarkan secara terus-menerus. Oleh karena itu, dalam implementasi praktis, protokol ini sering dikombinasikan dengan teknologi keamanan seperti TLS/SSL untuk mengenkripsi data yang dikirimkan.

Implementasi *WebSocket* memberikan berbagai keuntungan dalam pengembangan aplikasi *chat*, antara lain [11]:

1. Latensi Rendah: Karena tidak perlu membuka dan menutup koneksi berulang-ulang, respons data menjadi lebih cepat.
2. Efisiensi *Bandwidth*: Tidak ada *header* HTTP tambahan untuk setiap pesan, sehingga penggunaan *bandwidth* menjadi lebih hemat.
3. Komunikasi Dua Arah (*Full-Duplex*): Klien dan *Server* dapat saling mengirim data kapan pun tanpa harus menunggu permintaan.
4. Pengalaman Pengguna Lebih Baik: Respons instan terhadap tindakan pengguna menciptakan interaksi yang lebih mulus dan responsif.
5. Dukungan Fitur Tambahan: Memungkinkan implementasi fitur seperti sinkronisasi multi-perangkat dan integrasi dengan layanan lain seperti *API* penerjemah.

Secara keseluruhan, teknologi *WebSocket* telah menjadi standar industri dalam membangun sistem komunikasi berbasis *web* yang efisien, cepat, dan interaktif. Integrasinya dalam aplikasi *chat* dengan fitur tambahan seperti penerjemahan otomatis akan semakin memperluas potensi penggunaan *WebSocket* dalam mendukung komunikasi global lintas bahasa.

2.3 Model Bahasa Besar (LLM)

Model Bahasa Besar (Large Language Models atau LLM) merupakan sebuah pendekatan canggih dalam bidang kecerdasan buatan yang menjadi fokus utama dalam penelitian ini untuk mengatasi kesenjangan kualitas pada layanan penerjemahan yang umum digunakan. LLM dilatih pada data teks dalam jumlah yang sangat masif, memungkinkannya untuk memahami bahasa secara lebih mendalam, tidak hanya kata per kata, tetapi juga keseluruhan kalimat dan konteksnya.

Berbagai studi terdahulu telah menunjukkan keunggulan kualitatif dari LLM. Sebagai contoh, studi yang menganalisis kemampuan ChatGPT menemukan bahwa kualitas terjemahannya lebih unggul dibandingkan Google Translate dan DeepL, terutama dalam mengaplikasikan metode penerjemahan semantis dan komunikatif [7]. Penelitian lain juga menyimpulkan bahwa ChatGPT, khususnya dengan mesin GPT-4, merupakan penerjemah yang sangat baik [8]. Keunggulan ini terletak pada kemampuan LLM untuk memahami konteks percakapan secara menyeluruh, menginterpretasi bahasa non-formal dan idiom, serta menjaga nada atau gaya bicara [8].

Untuk dapat memanfaatkan kemampuan canggih ini, pengembang menggunakan Application Programming Interface (API). API memungkinkan sebuah aplikasi untuk berkomunikasi dan menggunakan fungsionalitas dari sistem eksternal lain tanpa harus

membangunnya dari awal, yang merupakan pendekatan umum dalam pengembangan aplikasi web modern untuk mendukung efisiensi dan modularitas [17]. Dalam penelitian ini, OpenAI API akan berfungsi sebagai jembatan yang menghubungkan aplikasi chat yang dikembangkan dengan model bahasa canggih milik OpenAI. Alur kerja yang akan diimplementasikan adalah ketika server aplikasi menerima sebuah pesan, ia akan memanggil API OpenAI untuk menerjemahkan teks tersebut sebelum mengirimkan hasilnya kepada pengguna tujuan melalui koneksi *WebSocket*. Proses ini menuntut pengelolaan keamanan yang baik, di mana API key yang bersifat sensitif harus dijaga kerahasiaannya di sisi server dan tidak boleh terekspos ke pengguna akhir [18].

2.4 API OpenAI

Dalam pengembangan aplikasi multibahasa modern, layanan penerjemahan otomatis menjadi komponen yang sangat penting. Secara umum, teknologi yang digunakan adalah *Neural Machine Translation* (NMT), seperti yang dimanfaatkan oleh Google Translate. Teknologi NMT merupakan sebuah kemajuan besar karena mampu menerjemahkan kalimat secara utuh, tidak lagi kata per kata, sehingga menghasilkan terjemahan yang lebih natural [19]. Namun, untuk percakapan *chat* yang santai dan seringkali tidak formal, teknologi NMT ini masih memiliki keterbatasan. Hasil terjemahannya terlalu literal, dan seringkali gagal menangkap maksud atau idiom dari sebuah obrolan.

Untuk mengatasi masalah kualitas tersebut, penelitian ini menggunakan pendekatan yang lebih canggih, yaitu *Model Bahasa Besar (Large Language Models* atau LLM). LLM adalah model kecerdasan buatan yang dilatih pada data teks dalam jumlah yang sangat besar, memungkinkannya untuk memahami bahasa seperti bahasa gaul ataupun idiom dengan konteks dan pola kalimat yang rumit. Berbagai penelitian telah membuktikan keunggulan LLM. Sebagai contoh, sebuah studi menemukan bahwa kualitas terjemahan ChatGPT lebih unggul dibandingkan Google Translate dan DeepL, terutama dalam hal pemahaman makna [7]. Penelitian lain juga menyimpulkan bahwa ChatGPT, khususnya dengan mesin GPT-4, adalah penerjemah yang sangat baik [8]. Keunggulan ini terletak pada kemampuan LLM untuk memahami alur obrolan, mengartikan bahasa gaul, serta menjaga gaya bicara [13].

Untuk dapat memanfaatkan kemampuan canggih ini, pengembang menggunakan *Application Programming Interface* (API). API ibaratnya adalah sebuah "jembatan" yang memungkinkan aplikasi kita untuk berkomunikasi dan menggunakan fitur dari sistem lain tanpa harus membangun semuanya dari awal. Ini adalah pendekatan umum dalam pengembangan aplikasi *web* modern agar lebih efisien [17]. Dalam penelitian ini, OpenAI

API akan berfungsi sebagai jembatan yang menghubungkan aplikasi *chat* kita dengan model bahasa canggih milik OpenAI.

Alur kerja API OpenAI dalam aplikasi ini dirancang agar aman dan efisien, dengan *server* aplikasi bertindak sebagai perantara. Proses ini memastikan bahwa kunci API (*API key*) yang bersifat rahasia tidak pernah terekspos ke pengguna [20]. Alur kerja penerjemahan pesan dapat diuraikan sebagai berikut:

1. Pengiriman Pesan Awal: Pengguna mengetik pesan di aplikasi (*frontend*) dan mengirimkannya ke *server backend* kita melalui koneksi *WebSocket*.
2. Pemanggilan API OpenAI: *Server* aplikasi kita menerima pesan tersebut, lalu membuat sebuah permintaan baru ke *server* OpenAI. Permintaan ini berisi teks dari pesan asli beserta instruksi untuk menerjemahkannya.
3. Penerimaan Hasil Terjemahan: OpenAI memproses permintaan tersebut menggunakan LLM dan mengirimkan kembali hasil terjemahannya ke *server* aplikasi kita.
4. Penerusan ke Pengguna Tujuan: Setelah menerima hasil terjemahan, *server* aplikasi kita akan meneruskan pesan yang sudah diterjemahkan ke pengguna tujuan melalui koneksi *WebSocket* yang sama.

Tabel 2.1 [21] merangkum perbandingan dari ketiga layanan API tersebut berdasarkan informasi yang ada.

Tabel 2. 1 Perbandingan Layanan API Penerjemahan

Aspek	Google Translate	OpenAI	DeepL
Akurasi & Konteks	Cenderung menerjemahkan secara literal (kata per kata)	Mampu menafsirkan makna mendalam.	Lebih fokus pada terjemahan literal dan kesulitan menafsirkan konteks yang rumit.
Keunggulan Utama	Cakupan bahasa yang sangat luas.	Pemahaman konteks yang mendalam.	Bagus dalam menerjemahkan bahasa-bahasa Eropa.
Keterbatasan	Kesulitan dengan nuansa, idiom, dan gaya bahasa.	Masih bisa salah menafsirkan	Kurang unggul dalam pemahaman konteks yang dalam

2.5 Integrasi Layanan Eksternal API dalam Aplikasi *Web*

Dalam pengembangan aplikasi *web* modern, sangat umum bagi sebuah aplikasi untuk menggunakan fitur atau layanan dari aplikasi lain. Proses ini dimungkinkan oleh *Application Programming Interface* (API), yang dapat diibaratkan sebagai sebuah "jembatan" yang memungkinkan dua program perangkat lunak untuk berkomunikasi dan bertukar data satu sama lain. Menggunakan API adalah pendekatan yang sangat efisien karena pengembang tidak perlu membangun setiap fitur kompleks dari nol. Sebaliknya, mereka dapat mengintegrasikan layanan yang sudah jadi dan teruji dari pihak ketiga, seperti sistem pembayaran, layanan peta, atau dalam kasus ini, penerjemahan [17].

Pada konteks aplikasi *chat* yang dikembangkan dalam penelitian ini, integrasi API menjadi sangat krusial untuk menyediakan fitur penerjemahan. Alur kerjanya adalah ketika seorang pengguna mengirim pesan, *server* aplikasi akan memanggil API dari OpenAI untuk menerjemahkan teks tersebut secara otomatis. Proses ini perlu berjalan sangat cepat agar tidak mengganggu kelancaran percakapan. Oleh karena itu, mekanisme pemanggilan API ini sering dikombinasikan dengan teknologi komunikasi seperti *WebSocket* untuk memastikan bahwa pesan asli dan hasil terjemahannya dapat ditampilkan kepada pengguna tanpa jeda yang terasa [16].

Proses integrasi API juga melibatkan beberapa pertimbangan teknis dan keamanan yang penting. Untuk menggunakan layanan eksternal, aplikasi memerlukan sebuah kunci API (*API key*), yang berfungsi seperti kata sandi khusus untuk aplikasi. Kunci ini bersifat sangat rahasia dan tidak boleh sampai terekspos ke pengguna akhir di sisi *browser* (*client*). Oleh karena itu, semua proses pemanggilan ke API eksternal harus dikelola secara aman di sisi *server* [18]. Selain keamanan, pengembang juga perlu memperhatikan aspek performa, seperti latensi (jeda waktu) dari respons API, agar tidak memperlambat pengalaman pengguna secara keseluruhan [22].

2.6 Firebase

Dalam pengembangan aplikasi modern, seringkali tantangan terbesar bukan hanya pada pembuatan antarmuka yang menarik, tetapi juga pada pembangunan sisi *server* (*backend*) yang andal. Proses ini mencakup pengelolaan *database*, sistem *login* pengguna, dan infrastruktur lainnya yang rumit dan memakan waktu. Untuk mengatasi tantangan ini, muncul pendekatan yang disebut *Backend-as-a-Service* (BaaS), di mana pengembang dapat menggunakan layanan *backend* yang sudah jadi dari penyedia pihak ketiga. *Firebase*, yang

dikembangkan oleh Google, adalah salah satu platform BaaS yang paling populer saat ini [26].

Firebase menyediakan serangkaian *tools* dan layanan siap pakai yang memungkinkan pengembang untuk mempercepat proses pengembangan secara signifikan. Dengan menggunakan *Firebase*, pengembang tidak perlu lagi pusing memikirkan konfigurasi *server* atau membangun sistem *database* dari nol. Sebaliknya, mereka dapat langsung fokus pada hal yang paling penting, yaitu membangun fitur dan pengalaman pengguna (*user experience*) di sisi klien (*frontend*). Platform ini sangat cocok untuk pengembangan aplikasi seperti aplikasi *chat* yang menjadi fokus dalam penelitian ini [27].

Ekosistem *Firebase* sangat luas, mencakup berbagai layanan yang saling terintegrasi, mulai dari *database*, autentikasi, penyimpanan *file*, hingga *hosting*. Untuk penelitian ini, dua layanan utama dari *Firebase* akan menjadi fondasi dari arsitektur *backend* aplikasi, yaitu *Cloud Firestore* sebagai *database* untuk menyimpan data percakapan, dan *Firebase* Authentication untuk mengelola seluruh proses yang berkaitan dengan akun pengguna. Kombinasi kedua layanan ini menyediakan solusi *backend* yang kuat, aman, dan dapat diskalakan [23].

2.6.1 Cloud Firestore

Cloud Firestore adalah layanan *database* NoSQL yang disediakan oleh *Firebase*. *Firestore* menyimpan data dalam format yang lebih fleksibel, yaitu dalam bentuk dokumen dan koleksi. Dokumen dapat diibaratkan sebagai sebuah catatan yang berisi pasangan kunci-nilai (mirip format JSON), sementara koleksi adalah wadah untuk kumpulan dokumen. Struktur ini sangat cocok untuk menyimpan data yang dinamis dan tidak terstruktur, seperti data percakapan dalam sebuah aplikasi *chat* [24].

Keunggulan utama dan yang paling relevan dari *Cloud Firestore* untuk aplikasi ini adalah kemampuannya untuk melakukan sinkronisasi data secara langsung. Di mana aplikasi klien akan "mendengarkan" perubahan pada data tertentu di *database*. Setiap kali ada data yang ditambahkan, diubah, atau dihapus, *server Firestore* akan secara otomatis "mendorong" (*push*) pembaruan tersebut ke semua klien yang sedang mendengarkan. Hal ini terjadi hampir secara instan, tanpa klien perlu terus-menerus bertanya ke *server* apakah ada data baru [28].

Dalam konteks aplikasi *chat*, fitur sinkronisasi secara langsung ini sangat penting. Ketika seorang pengguna mengirim pesan, pesan tersebut akan disimpan di *Firestore*. Seketika itu juga, *Firestore* akan mengirimkan pembaruan ke pengguna penerima, sehingga

pesan baru tersebut langsung muncul di layarnya tanpa perlu memuat ulang halaman. Kemampuan ini telah terbukti andal dan efektif dalam berbagai penelitian pengembangan aplikasi *chat*, karena mampu menciptakan pengalaman percakapan yang lancar dan instan, seolah-olah terjadi secara langsung [25].

2.6.2 Firebase Authentication

Firebase Authentication adalah layanan yang menyediakan sistem manajemen pengguna yang lengkap dan aman. Layanan ini mengambil alih semua tugas rumit yang berkaitan dengan autentikasi, seperti proses pendaftaran, *login*, pengaturan ulang kata sandi, dan verifikasi email. Dengan menggunakan *Firebase Authentication*, pengembang tidak perlu lagi membangun sistem ini dari awal, yang seringkali memakan waktu dan rawan terhadap celah keamanan [28].

Layanan ini mendukung berbagai metode *login* yang fleksibel, mulai dari metode klasik seperti email dan kata sandi, hingga integrasi dengan penyedia identitas pihak ketiga yang populer (dikenal sebagai *federated identity providers*), seperti Google, Facebook, dan GitHub. Fleksibilitas ini memberikan kemudahan bagi pengguna untuk mendaftar atau masuk ke aplikasi menggunakan akun yang sudah mereka miliki.

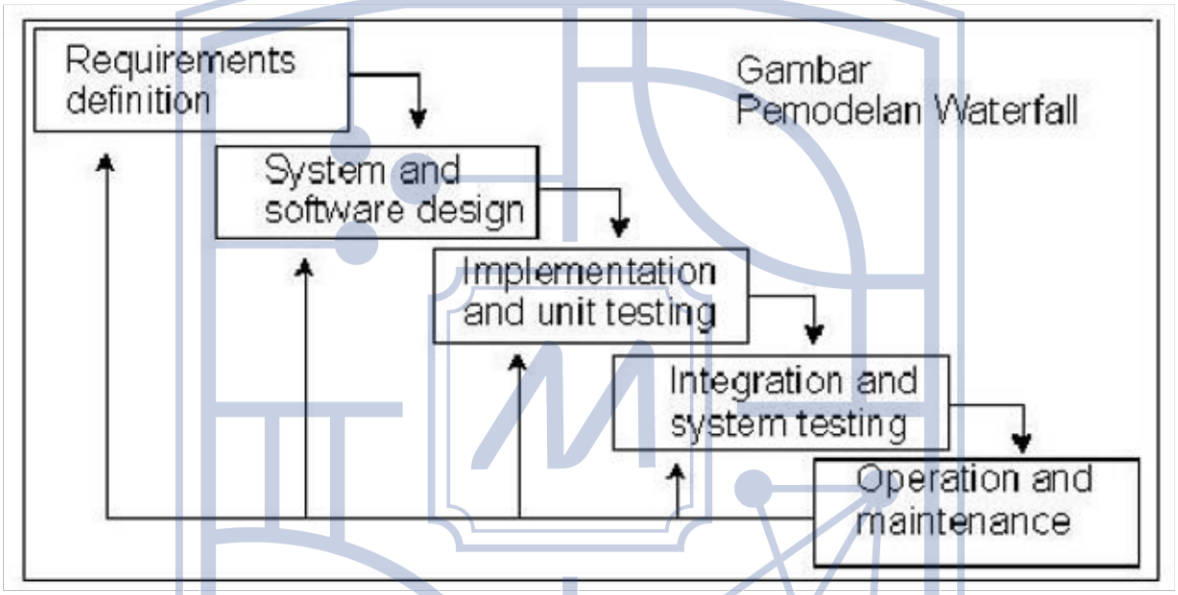
Dari sisi keamanan, *Firebase Authentication* menangani semua aspek penting secara otomatis. Ketika pengguna mendaftar, kata sandi mereka tidak disimpan dalam bentuk teks biasa, melainkan di-*hash* menggunakan algoritma standar industri untuk melindunginya. Selain itu, layanan ini juga mengelola sesi pengguna menggunakan token yang aman, serta menyediakan fitur keamanan tambahan seperti autentikasi multi-faktor untuk lapisan perlindungan ekstra. Dengan menyerahkan tugas-tugas keamanan ini kepada *Firebase*, pengembang dapat memastikan bahwa sistem autentikasi aplikasi mereka aman dan andal, sesuai dengan standar yang ditetapkan oleh Google [28].

2.7 Metode Waterfall

Untuk membuat aplikasi ini, metode kerja yang dipakai adalah metode *Waterfall*. Metode ini adalah salah satu cara paling dasar untuk membuat perangkat lunak, yang prosesnya sangat teratur dan berurutan. Disebut *Waterfall* (air terjun) karena pengerjaannya mengalir ke bawah, dari satu tahap ke tahap berikutnya.[29] Aturan utamanya adalah satu tahap harus benar-benar selesai dan diperiksa sebelum bisa pindah ke tahap selanjutnya, tanpa bisa kembali lagi ke tahap sebelumnya. Karena prosesnya yang kaku, metode ini

sangat cocok untuk proyek yang tujuannya sudah sangat jelas dari awal, seperti aplikasi ini [25].

Pilihan metode *Waterfall* didasarkan pada kebutuhan proyek ini yang fiturnya sudah ditentukan secara spesifik. Dengan alur yang jelas, setiap langkah pengerjaan, mulai dari analisis hingga pengujian, dapat direncanakan dengan matang. Struktur yang teratur ini memberikan kontrol yang baik terhadap proyek dan memastikan semua proses didokumentasikan dengan rapi di setiap tahapannya [25]. Tahapan dalam metode *Waterfall* dapat dilihat seperti Gambar 2.1 [25].



Gambar 2. 1 Metode *Waterfall* [25]

Tahapan dalam metode *Waterfall* yang diterapkan dalam penelitian ini terdiri dari lima fase utama yang saling berkelanjutan, di mana hasil dari satu fase menjadi masukan wajib untuk fase berikutnya [30]:

1. Analisis Kebutuhan (*Requirement Analysis*)

   Tahap pertama ini adalah tentang perencanaan. Di sini, kami mengumpulkan semua informasi untuk mengerti apa saja yang dibutuhkan oleh aplikasi. Kami menentukan fitur-fitur apa saja yang harus ada (kebutuhan fungsional), seperti *login* dan kirim pesan, serta seberapa bagus aplikasi ini harus berjalan (kebutuhan non-fungsional), seperti harus cepat dan aman.

2. Perancangan Sistem (*System Design*)

   Tahap selanjutnya adalah membuat rancangan atau "cetak biru" dari aplikasi. Di tahap ini, kami mendesain *database* di *Firebase*, dan membuat desain tampilan (UI/UX) menggunakan Figma. Kami juga membuat diagram seperti *Use Case Diagram* dan

Activity Diagram untuk menggambarkan bagaimana pengguna akan memakai aplikasi dan bagaimana alur kerjanya.

3. Implementasi (*Implementation*)

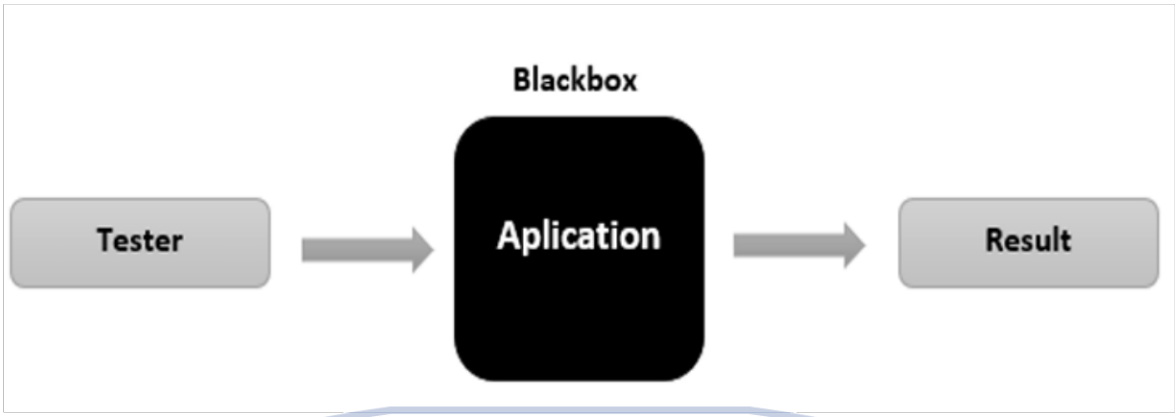
   Tahap ini adalah proses "pembangunan" di mana para *programmer* mulai menulis kode berdasarkan rancangan yang telah dibuat. Ini meliputi pembuatan sisi *backend* (mesin aplikasi) menggunakan Node.js, sisi *frontend* (tampilan yang dilihat pengguna) menggunakan React.js dan Tailwind CSS, serta menghubungkan semuanya dengan layanan luar seperti *Firebase* dan API OpenAI hingga menjadi sebuah aplikasi yang berfungsi.

4. Pengujian (*Testing*)

   Setelah aplikasi selesai dibuat, kami melakukan pengujian untuk memastikan semuanya berjalan dengan benar dan tidak ada *error*. Kami menggunakan metode *Blackbox Testing*, di mana kami menguji aplikasi dari sudut pandang pengguna biasa, tanpa melihat kode di dalamnya. Setiap fitur, dari mendaftar hingga menerjemahkan pesan, akan dicoba untuk memastikan semuanya bekerja sesuai harapan.

2.8 Blackbox Testing

Blackbox testing adalah sebuah metode pengujian perangkat lunak yang berfokus pada fungsionalitas aplikasi dari sudut pandang pengguna akhir, tanpa perlu mengetahui atau memeriksa struktur kode internalnya. Dalam pendekatan ini, sistem yang diuji dianggap sebagai sebuah "kotak hitam" yang misterius. Penguji tidak peduli bagaimana proses di dalam kotak tersebut bekerja; mereka hanya fokus pada *input* (aksi yang diberikan, seperti mengklik tombol) dan *output* (hasil yang ditampilkan) untuk memastikan bahwa aplikasi berjalan sesuai dengan yang diharapkan [31]. Tahapan ini dapat dilihat seperti Gambar 2.2 [33]



Gambar 2. 2 *Blackbox testing* [33]

Tujuan utama dari *blackbox testing* adalah untuk menemukan kesalahan dalam beberapa kategori, seperti fungsi yang salah atau hilang, kesalahan pada antarmuka pengguna, dan kesalahan dalam perilaku sistem secara umum [32]. Karena pengujian dilakukan dari perspektif pengguna, metode ini sangat efektif dalam menemukan *bug* yang mungkin terlewatkan oleh pengembang yang sudah terlalu akrab dengan kode programnya. Dalam konteks penelitian ini, *blackbox testing* akan digunakan untuk memastikan setiap fitur, mulai dari registrasi hingga pengiriman pesan dan penerjemahan, berfungsi dengan benar dalam skenario penggunaan nyata [25].

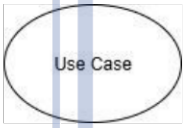
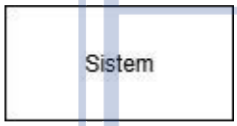
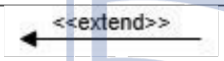
Secara lebih rinci, pengujian akan mencakup beberapa bagian utama aplikasi. Pertama adalah alur manajemen akun, yang meliputi proses pendaftaran, verifikasi email, *login*, dan *logout*. Kedua adalah fungsionalitas inti percakapan, seperti kemampuan untuk menambah teman, mengirim dan menerima pesan. Terakhir dan yang paling penting adalah fitur penerjemahan, di mana pengujian akan memvalidasi bahwa pesan yang diterima secara otomatis diterjemahkan sesuai dengan preferensi bahasa pengguna penerima.

2.9 *Use Case Diagram*

Use case diagram adalah salah satu jenis diagram dari *Unified Modeling Language* (UML) yang digunakan untuk menggambarkan interaksi fungsional antara aktor (pengguna atau sistem eksternal) dengan sistem yang akan dibangun [34]. Tujuan utama dari diagram ini adalah untuk memberikan gambaran tingkat tinggi yang jelas dan mudah dipahami mengenai "apa" saja yang bisa dilakukan oleh sistem dari sudut pandang pengguna, tanpa perlu merinci "bagaimana" sistem melakukannya secara teknis. Diagram ini sangat penting pada tahap analisis dan perancangan karena membantu dalam mendefinisikan ruang lingkup dan kebutuhan fungsional dari sebuah proyek [35].

Diagram *use case* terdiri dari beberapa komponen utama:

Tabel 2. 2 Simbol *Use Case Diagram*

No.	Simbol	Keterangan
1.		Aktor, entitas di luar sistem yang berinteraksi dengan sistem. Dalam penelitian ini, aktor utamanya adalah *User* dan Admin.
2.		*Use Case*, menggambarkan sebuah fungsionalitas atau layanan spesifik yang disediakan oleh sistem, seperti "Mendaftar Akun" atau "Mengirim Pesan".
3.		Sistem, direpresentasikan dengan sebuah kotak besar yang melingkupi semua *use case*, menunjukkan batasan dari sistem yang dikembangkan.
4.		Asosiasi, garis yang menghubungkan antara Aktor dan *Use Case*, yang menunjukkan bahwa aktor tersebut dapat menggunakan fungsionalitas tersebut.
5.		*Extend,* hubungan di mana sebuah *use case* dapat menyertakan fungsionalitas dari *use case* lain dalam kondisi tertentu (opsional).
6.		*Include*, hubungan di mana sebuah *use case* akan **selalu** menyertakan fungsionalitas dari *use case* lain.

Dengan memvisualisasikan interaksi ini, tim pengembang dapat memastikan bahwa semua kebutuhan pengguna telah teridentifikasi dan tidak ada fitur penting yang terlewat. Diagram ini juga berfungsi sebagai alat komunikasi yang efektif bagi sebuah tim untuk menyamakan persepsi mengenai fungsionalitas sistem yang akan dibangun [2].

2.10 Metode PIECES

Untuk menganalisis kebutuhan sistem secara terstruktur, terutama yang berkaitan dengan kualitas dan performa, penelitian ini menggunakan kerangka kerja PIECES. Metode PIECES adalah sebuah alat analisis yang digunakan untuk mengidentifikasi masalah, peluang, dan arahan dalam sebuah sistem informasi. Nama PIECES sendiri merupakan

singkatan dari enam kategori utama yang dievaluasi, yaitu *Performance, Information, Economics, Control, Efficiency,* dan *Service*. Kerangka kerja ini membantu analis sistem untuk melihat sebuah sistem dari berbagai sudut pandang yang berbeda, sehingga dapat mendefinisikan kebutuhan non-fungsional dengan lebih lengkap dan terarah [36].

Pendekatan PIECES sangat berguna pada tahap awal pengembangan untuk memastikan bahwa sistem yang akan dibangun tidak hanya fungsional, tetapi juga benar-benar memberikan nilai tambah dan solusi terhadap masalah yang ada. Dengan mengevaluasi setiap kategori, tim pengembang dapat mengidentifikasi area mana yang perlu ditingkatkan dan menetapkan standar kualitas yang jelas untuk aplikasi yang akan dibuat. [37]

Berikut adalah penjelasan untuk setiap kategori dalam kerangka kerja PIECES [36, 37, 38]:

- *Performance* (Kinerja): Kategori ini berfokus pada seberapa baik sistem menyelesaikan pekerjaannya.
- *Information* (Informasi): Kategori ini mengevaluasi kualitas data dan informasi yang dihasilkan oleh sistem. Ini termasuk akurasi, relevansi, dan ketersediaan informasi bagi pengguna.
- *Economics* (Ekonomi): Kategori ini melihat sistem dari sudut pandang biaya dan manfaat. Ini mempertimbangkan apakah sistem dapat membantu mengurangi biaya operasional atau memberikan keuntungan finansial.
- *Control & Security* (Kontrol & Keamanan): Kategori ini berfokus pada aspek keamanan sistem, seperti perlindungan data dari akses yang tidak sah dan memastikan bahwa hanya pengguna yang berwenang yang dapat melakukan aksi tertentu.
- *Efficiency* (Efisiensi): Kategori ini mengukur seberapa baik sistem memanfaatkan sumber daya (waktu, tenaga, atau sumber daya komputasi) untuk menghasilkan *output*. Sistem yang efisien dapat menyelesaikan tugas dengan usaha seminimal mungkin.
- *Service* (Layanan): Kategori ini menilai kualitas layanan yang diberikan oleh sistem kepada penggunanya. Ini mencakup aspek-aspek seperti kemudahan penggunaan (*usability*), keandalan (*reliability*), dan fleksibilitas sistem.