BAB III
LANDASAN TEORI

3.1 Konsep Dasar Sistem Informasi

3.1.1 Sistem, Data, dan Informasi

Sistem pada dasarnya merupakan sekumpulan elemen yang saling berhubungan, saling memengaruhi, dan bekerja secara terkoordinasi untuk mencapai tujuan tertentu. Dalam konteks organisasi pendidikan, sistem tidak hanya dipahami sebagai perangkat lunak, melainkan juga mencakup prosedur kerja, sumber daya manusia, aliran data, serta aturan operasional yang membentuk satu kesatuan proses (Herdiansah et al., 2025).

Data merupakan fakta mentah yang belum memiliki makna penuh sebelum diolah, misalnya nama siswa, tanggal kehadiran, tema pembelajaran, atau nomor surat. Data menjadi bernilai ketika diorganisasi, divalidasi, dikelompokkan, dan dihubungkan dengan kebutuhan pengguna (Prayoga, 2024).

Informasi adalah hasil pengolahan data yang telah diberi konteks sehingga mempunyai arti dan nilai guna bagi penerima. Dalam lingkungan sekolah, data presensi harian dapat diolah menjadi rekap kehadiran, data tema pembelajaran dapat diolah menjadi rancangan kegiatan, dan data persuratan dapat diubah menjadi arsip administratif yang mudah ditelusuri kembali (Adittiya et al., 2025).

Berdasarkan uraian tersebut, dapat dipahami bahwa sistem, data, dan informasi memiliki hubungan yang saling berkaitan. Sistem berfungsi sebagai mekanisme pengolah, data menjadi bahan mentah yang dimasukkan, dan informasi merupakan hasil akhir yang digunakan untuk mendukung pekerjaan administratif maupun pengambilan keputusan di lembaga pendidikan.

3.1.2 Sistem Informasi Manajemen

Sistem Informasi Manajemen (SIM) adalah sistem terintegrasi yang digunakan untuk mengumpulkan, mengolah, menyimpan, dan menyajikan informasi bagi kebutuhan operasional maupun manajerial organisasi (Herdiansah et al., 2025).

Dalam bidang pendidikan, SIM berfungsi membantu sekolah mengelola data akademik, administrasi, dan layanan pendukung agar pekerjaan menjadi lebih tertib, efisien, dan mudah dipantau (Adittiya et al., 2025). Pada level manajerial, SIM membantu pimpinan lembaga memperoleh informasi yang lebih cepat dan lebih akurat untuk evaluasi serta pengambilan keputusan (Prayoga, 2024).

Dalam penelitian ini, AISYA-RA dapat diposisikan sebagai implementasi SIM pada bidang administrasi guru di lingkungan RA. Sistem ini dirancang untuk mengintegrasikan penyusunan RPPH, pengelolaan presensi, persuratan, serta layanan tanya jawab berbasis *Chatbot* dalam satu platform (Herdiansah et al., 2025). Dengan demikian, *Chatbot* pada AISYA-RA bukan fitur yang berdiri sendiri, melainkan bagian dari sistem informasi manajemen yang mendukung pekerjaan administratif guru dan kepala RA.

3.2 *Website*

Website adalah sekumpulan halaman atau sumber daya digital yang saling terhubung dan diakses melalui *browser*. Dalam perkembangannya, *Website* tidak lagi hanya digunakan untuk menampilkan informasi secara statis, tetapi juga berkembang menjadi aplikasi berbasis web yang memungkinkan pengguna berinteraksi, mengisi data, mengirim permintaan, dan menerima keluaran secara langsung (Rudi Sanjaya et al., 2022).

Aplikasi berbasis web memiliki sejumlah kelebihan, seperti dapat diakses lintas perangkat, tidak memerlukan instalasi rumit di sisi pengguna, dan lebih mudah dipelihara karena pembaruan dilakukan di sisi server (Adittiya et al., 2025). Bagi lembaga pendidikan, pendekatan web juga memudahkan guru dan pimpinan sekolah menggunakan sistem melalui perangkat yang telah mereka miliki, baik laptop maupun ponsel, selama tersedia *browser* dan jaringan yang memadai (Rudi Sanjaya et al., 2022).

Dalam konteks AISYA-RA, *Website* menjadi bentuk antarmuka utama yang mempertemukan pengguna dengan seluruh layanan sistem. Melalui *browser*, guru dapat mengakses menu administrasi, mengisi formulir, melihat data, dan berinteraksi dengan *Chatbot* pada ruang kerja yang terintegrasi. Dengan demikian, pemilihan bentuk aplikasi berbasis web pada penelitian ini bukan hanya keputusan teknis, tetapi juga keputusan

desain yang bertujuan meningkatkan aksesibilitas, kemudahan penggunaan, dan efisiensi operasional di lingkungan RA (Adittiya et al., 2025).

3.3 Administrasi Guru di Raudhatul Athfal (RA)

Administrasi guru di Raudhatul Athfal (RA) merupakan bagian yang tidak terpisahkan dari proses pendidikan. Guru RA tidak hanya melaksanakan kegiatan pembelajaran, tetapi juga menyusun rencana pembelajaran, mencatat kehadiran, mendokumentasikan perkembangan peserta didik, dan menyiapkan dokumen kelembagaan yang dibutuhkan sekolah (Wardi et al., 2023).

Dengan demikian, administrasi guru harus dipahami sebagai bagian dari profesionalitas kerja guru, bukan sekadar beban tambahan di luar tugas mengajar (Sari & Firmansyah, 2025). Dalam konteks madrasah, administrasi guru RA juga berkaitan erat dengan tata kelola kurikulum, evaluasi pembelajaran, dan kepatuhan terhadap standar pendidikan (Qamaria et al., 2025).

Praktik administrasi yang baik mendukung ketertiban lembaga, mempermudah supervisi kepala RA, dan menjaga keberlanjutan arsip akademik maupun nonakademik. Pada penelitian ini, fokus administrasi guru di RA diwujudkan dalam tiga bidang utama, yaitu penyusunan RPPH, pencatatan presensi siswa, dan manajemen surat resmi. Ketiga bidang tersebut dipilih karena mewakili pekerjaan administratif yang paling sering dilakukan guru dan kepala RA, serta memiliki potensi besar untuk dibantu melalui sistem informasi berbasis web dan AI (Sari & Firmansyah, 2025).

3.3.1 Rencana Pelaksanaan Pembelajaran Harian (RPPH)

RPPH merupakan dokumen perencanaan pembelajaran harian yang menjadi acuan guru dalam melaksanakan kegiatan belajar anak usia dini (Syafwandi & Juairiyah, 2023).

RPPH disusun sebagai penjabaran operasional dari rencana pembelajaran yang lebih luas, sehingga kegiatan yang dilakukan setiap hari tetap selaras dengan tujuan perkembangan anak, tema pembelajaran, dan target penilaian (Sahyan et al., 2023). Secara pedagogik, RPPH penting karena membantu guru menentukan hubungan antara tujuan, materi, alat dan bahan, langkah kegiatan, serta bentuk penilaian (Dwinata et al., 2025).

Dalam praktiknya, RPPH membantu guru menghindari pembelajaran yang terlalu improvisasional dan memastikan setiap kegiatan memiliki tujuan perkembangan yang jelas. Dalam konteks AISYA-RA, teori RPPH menjadi sangat relevan karena salah satu fungsi sistem adalah membantu guru menyusun draf RPPH secara lebih cepat melalui bantuan AI (Syafwandi & Juairiyah, 2023). Namun, bantuan tersebut tetap harus mengikuti struktur pedagogik yang benar, bukan sekadar menghasilkan teks bebas.

3.3.2 Presensi Siswa

Presensi siswa adalah proses pencatatan kehadiran peserta didik pada waktu tertentu sebagai bagian dari administrasi kelas (Joediono et al., 2024).

Kehadiran siswa merupakan data dasar yang penting karena berhubungan dengan kedisiplinan, partisipasi dalam pembelajaran, serta kesinambungan proses belajar (Joediono et al., 2024). Dari sudut pandang manajemen kelas, data presensi membantu guru mengetahui pola kehadiran siswa, mendeteksi ketidakhadiran berulang, dan menentukan tindak lanjut yang diperlukan (Joediono et al., 2024)

Dalam AISYA-RA, presensi siswa menjadi salah satu fitur utama karena mencerminkan kebutuhan administratif harian yang nyata di sekolah (Joediono et al., 2024) Integrasi presensi dalam sistem memungkinkan guru melakukan *input* data dengan cara yang lebih efisien, sedangkan hasilnya dapat langsung disimpan, direkap, dan digunakan sebagai dasar pemantauan kelas. Dengan demikian, presensi digital bukan sekadar fitur teknis, tetapi bagian penting dari manajemen kelas.

3.3.3 Manajemen Surat

Manajemen surat adalah kegiatan mengelola surat masuk dan surat keluar melalui pencatatan, penyimpanan, pengarsipan, pencarian kembali, dan pelaporan sesuai aturan yang berlaku (Faza & Samsudin, 2025).

Dalam administrasi lembaga, surat resmi berfungsi sebagai media komunikasi formal yang memiliki nilai legal dan organisatoris (Wulandari & Ismaya, 2024). Digitalisasi persuratan dapat meningkatkan efisiensi birokrasi karena mempercepat pembuatan draf surat, memudahkan standarisasi format, serta mendukung pengarsipan yang lebih terstruktur (Mahendra Hasibuan & Chansa Thelma, 2025).

Selain itu, sistem persuratan digital membantu sekolah mengurangi ketergantungan pada pencatatan manual yang rawan salah, mudah tercecer, dan sulit ditelusuri kembali (Nabila et al., 2025). Dalam penelitian ini, teori manajemen surat resmi menjadi landasan bagi fitur AISYA-RA yang membantu pembuatan surat secara lebih cepat dan seragam. Otomatisasi pembuatan surat resmi dipandang mampu mempercepat alur administrasi sekolah tanpa mengurangi formalitas dokumen (Faza & Samsudin, 2025).

3.4 Kecerdasan Buatan (*Artificial Intelligence*) dan *Chatbot*

3.4.1 *Artificial Intelligence (AI)*

Artificial Intelligence (AI) adalah cabang ilmu komputer yang mengembangkan sistem yang mampu menjalankan tugas-tugas yang biasanya membutuhkan kecerdasan manusia, seperti memahami bahasa, mengenali pola, membuat prediksi, dan menghasilkan respons berdasarkan data (Yafianto et al., 2025).

Dalam beberapa tahun terakhir, AI berkembang sangat pesat, terutama pada bidang pemrosesan bahasa alami, sistem rekomendasi, dan *generative* AI (Rahmawati et al., 2025). Dalam konteks layanan administrasi, AI dapat digunakan untuk mempercepat penyajian informasi, membantu pekerjaan repetitif, dan menurunkan hambatan interaksi antara pengguna dengan sistem (Wahyu Widhyana & Magdalena Lidya Gultom, 2025)

Nilai AI terletak pada kemampuannya membantu manusia bekerja lebih efisien, bukan semata-mata menggantikan manusia. Pada AISYA-RA, AI diposisikan sebagai komponen pendukung yang membantu guru menyusun dokumen, menjawab pertanyaan administratif, dan mempermudah interaksi melalui bahasa alami (Yafianto et al., 2025).

3.4.2 *Natural Language Processing (NLP)*

Natural Language Processing (NLP) adalah cabang kecerdasan buatan yang berfokus pada kemampuan komputer untuk memahami, menginterpretasikan, dan menghasilkan bahasa manusia secara alami. NLP mencakup berbagai tugas komputasional seperti tokenisasi, penghapusan kata-kata umum *(stop word removal*), pengenalan entitas bernama (*named entity recognition*), klasifikasi intensi, dan pembuatan teks (Puspitasari et al., 2024).

Dalam konteks sistem berbasis *Chatbot*, NLP berperan sebagai komponen inti yang memungkinkan mesin memahami maksud perintah pengguna yang disampaikan dalam bahasa sehari-hari. Proses ini mencakup serangkaian tahapan, mulai dari pra-pemrosesan teks (*preprocessing*) hingga klasifikasi intensi (*intent recognition*) yang mengidentifikasi tujuan dari pernyataan atau permintaan pengguna. Dari hasil klasifikasi tersebut, sistem dapat menentukan tindakan apa yang harus diambil (Hikmah, 2022).

Klasifikasi intensi merupakan komponen paling krusial dalam NLP pada sistem *Chatbot* administratif. Melalui klasifikasi intensi, sistem dapat mengenali bahwa kalimat "Tolong buatkan RPPH untuk hari Senin tema Binatang" memiliki intensi buat_RPPH dengan entitas hari: Senin dan tema: Binatang. Dengan demikian, *Chatbot* dapat secara otomatis meneruskan permintaan ke modul RPPH dengan parameter yang tepat tanpa pengguna harus membuka menu secara manual (Nurmansyah & Zakaria, 2024).

Pada sistem berbasis LLM seperti AISYA-RA, kemampuan NLP klasik sebagian besar sudah terintegrasi di dalam model bahasa besar. Model seperti Gemini API tidak lagi membutuhkan *pipeline* NLP yang dibangun secara terpisah karena pemahaman bahasa, pengenalan intensi, dan ekstraksi entitas sudah dilakukan secara implisit oleh model. Meskipun demikian, pemahaman teoritis tentang NLP tetap penting karena menjelaskan landasan kerja model dalam memahami perintah pengguna dan menghasilkan respons yang relevan (Waleska et al., 2025).

3.4.3 *Chatbot*

Chatbot adalah program yang dirancang untuk mensimulasikan percakapan dengan pengguna melalui bahasa alami, baik dalam bentuk teks maupun suara (Tri Utami Br Lubis et al., 2024).

Dalam bidang pendidikan dan layanan informasi, *Chatbot* berkembang sebagai antarmuka yang memudahkan pengguna memperoleh jawaban, panduan, atau bantuan tugas tanpa harus memahami struktur sistem yang rumit (Yafianto et al., 2025). Salah satu kelebihan *Chatbot* adalah kemampuannya menurunkan beban antarmuka karena pengguna tidak harus selalu menelusuri menu satu per satu, tetapi dapat langsung menyampaikan kebutuhan dalam bentuk kalimat percakapan (Pradana et al., 2025).

Dalam perkembangan terkini, *Chatbot* tidak lagi hanya dipahami sebagai antarmuka tanya-jawab yang memberikan respons berbasis pola atau aturan tetap. *Chatbot* modern yang ditenagai oleh LLM mampu berfungsi sebagai *conversational user interface (CUI)*, yaitu antarmuka percakapan yang memungkinkan pengguna mengendalikan fungsi-fungsi sistem menggunakan kalimat bahasa alami tanpa harus memahami struktur menu yang kompleks (Qaulan et al., 2025). Pendekatan ini secara signifikan menurunkan hambatan penggunaan sistem bagi pengguna yang memiliki literasi digital yang terbatas.

Sebagai CUI, *Chatbot* tidak hanya merespons pertanyaan, tetapi juga menerima instruksi dan menerjemahkannya menjadi operasi nyata di dalam sistem. Misalnya, perintah "Buatkan RPPH untuk hari Senin" akan diproses oleh *Chatbot*, dikenali sebagai perintah administratif, lalu diteruskan ke modul RPPH untuk menghasilkan draf yang sesuai (Hikmah, 2022). Dengan posisi ini, *Chatbot* berperan ganda: sebagai antarmuka percakapan berbasis AI sekaligus sebagai pengendali operasional sistem informasi administrasi guru.

Dalam AISYA-RA, *Chatbot* tidak menggantikan menu konvensional, melainkan melengkapinya. Pendekatan ini penting karena sebagian pengguna masih memerlukan tombol, formulir, dan struktur navigasi yang familier, sementara sebagian lainnya akan terbantu oleh interaksi bahasa alami. Dengan demikian, *Chatbot* pada penelitian ini diposisikan sebagai teori antarmuka percakapan yang berjalan berdampingan dengan antarmuka konvensional dalam satu layar kerja sistem (Tri Utami Br Lubis et al., 2024).

3.4.4 Conversational User Interface *(CUI)*

Conversational User Interface (CUI) adalah paradigma antarmuka yang memungkinkan pengguna berinteraksi dengan sistem melalui bahasa alami, baik dalam bentuk teks maupun suara, alih-alih menggunakan elemen visual statis seperti tombol, menu, atau formulir (Flohr et al., 2021). Dalam CUI, interaksi tidak lagi dibatasi oleh jalur navigasi yang telah ditentukan sebelumnya, melainkan terbuka dan fleksibel karena sistem beradaptasi terhadap bahasa pengguna. Hal ini menjadikan CUI sebagai pergeseran paradigma dalam interaksi manusia-komputer (*Human-Computer Interaction*) dari

pendekatan berbasis klik *(click-based)* ke pendekatan berbasis percakapan (*conversation-based*) (Flohr et al., 2021).

Berbeda dengan *Graphical User Interface (GUI)* yang mengharuskan pengguna mempelajari struktur logika antarmuka sebelum dapat berinteraksi secara efektif, CUI mengandalkan kemampuan bahasa alami yang sudah dikuasai pengguna sejak awal. Hal ini berpotensi menurunkan hambatan penggunaan sistem, terutama bagi pengguna dengan literasi digital yang terbatas (Flohr et al., 2021). Tabel perbandingan karakteristik CUI dan GUI disajikan pada Tabel 3.1.

Tabel 3.1 Perbandingan Karakteristik GUI dan CUI

Karakteristik	**GUI *(Graphical User Interface)***	**CUI *(Conversational User Interface)***
Modalitas interaksi	Klik, sentuh, pilih menu	Bahasa alami (teks/suara)
Jalur navigasi	Terstruktur dan ditentukan sebelumnya	Fleksibel dan terbuka
Kurva pembelajaran	Memerlukan pembelajaran struktur antarmuka	Rendah, menggunakan bahasa sehari-hari
Aksesibilitas pengguna literasi digital rendah	Tinggi hambatan	Rendah hambatan
Penyesuaian terhadap pengguna	Statis, sistem tidak beradaptasi	Dinamis, sistem menyesuaikan konteks percakapan
Pemulihan kesalahan	Terbatas pada pesan error baku	Kontekstual, sistem dapat memandu pengguna

(Sumber:Flohr et al., 2021)

Antarmuka percakapan memiliki tingkat komunikabilitas yang lebih tinggi dibandingkan antarmuka tradisional untuk pengguna dengan keterbatasan paparan teknologi. CUI terbukti mampu meningkatkan efektivitas dan efisiensi komunikasi antara pengguna dan sistem, khususnya pada tugas sederhana seperti pencarian informasi atau pemicuan operasi.

Meskipun demikian, CUI tidak selalu menggantikan GUI secara mutlak. Kedua paradigma antarmuka memiliki keunggulan masing-masing. GUI unggul dalam skenario *"happy path"* di mana pengguna mengetahui dengan pasti langkah-langkah yang harus dilakukan, sedangkan CUI unggul dalam skenario yang membutuhkan fleksibilitas atau perubahan rencana. Oleh karena itu, pendekatan yang paling efektif adalah mengintegrasikan elemen percakapan ke dalam antarmuka berbasis GUI, sehingga pengguna dapat memilih modalitas interaksi yang paling sesuai dengan kebutuhan dan konteks mereka (Flohr et al., 2021).

3.5 Literasi Digital dan Penerimaan Teknologi oleh Pendidik

3.5.1 Literasi Digital Pendidik PAUD/RA

Literasi digital didefinisikan sebagai kemampuan untuk mengakses, menyeleksi, memahami, dan mendistribusikan informasi melalui media digital (Novitasari & Fauziddin, 2022). Dalam konteks pendidikan anak usia dini, literasi digital tenaga pendidik menjadi faktor pendukung yang menentukan keberhasilan integrasi teknologi dalam kegiatan pembelajaran dan administrasi (Novitasari & Fauziddin, 2022).

Literasi digital tenaga pendidik PAUD secara umum berada pada kategori "cukup baik", yang berarti masih terdapat ruang yang signifikan untuk peningkatan. Empat indikator literasi digital yang diukur meliputi kemampuan mengakses informasi digital, menyeleksi informasi yang relevan, memahami konten digital, serta mendistribusikan informasi melalui kanal digital (Novitasari & Fauziddin, 2022). Keterbatasan literasi digital pada pendidik PAUD berdampak langsung pada keengganan mengadopsi sistem informasi berbasis komputer, yang pada gilirannya menghambat efisiensi administrasi lembaga.

Hambatan ini tidak bersifat teknis semata, melainkan bersifat psikologis dan berkaitan dengan ketakutan akan kesalahan sistem (*error*). Oleh karena itu, desain sistem yang ditujukan perlu mempertimbangkan tingkat literasi digital pengguna sebagai faktor desain utama, bukan sekadar variabel tambahan (Novitasari & Fauziddin, 2022).

3.5.2 *Technology Acceptance Model (TAM)*

Technology Acceptance Model (TAM) adalah kerangka teori yang dikembangkan oleh Davis (1989) untuk menjelaskan dan memprediksi penerimaan teknologi oleh pengguna. TAM menyatakan bahwa dua faktor utama yang memengaruhi penerimaan teknologi adalah *Perceived Usefulness* (PU) atau persepsi kegunaan, yaitu sejauh mana seseorang percaya bahwa menggunakan teknologi tertentu akan meningkatkan kinerja kerjanya, dan *Perceived Ease of Use* (PEOU) atau persepsi kemudahan penggunaan, yaitu sejauh mana seseorang percaya bahwa menggunakan teknologi tertentu akan terbebas dari upaya mental dan fisik (Davis, 1989; Nur Amalia et al., 2023)

3.6 Model Bahasa dan Pemrosesan Konteks

3.6.1 *Large Language Model (LLM)*

Large Language Model (LLM) adalah model bahasa berskala besar yang dilatih pada kumpulan data teks yang sangat besar sehingga mampu memahami dan menghasilkan bahasa alami secara luwes (Ardiansyah et al., 2025).

LLM modern banyak digunakan untuk tugas seperti menjawab pertanyaan, pembuatan ringkasan, klasifikasi teks, penyusunan dokumen, dan percakapan berbasis AI (Hidayat et al., 2025). Dalam implementasinya, LLM dapat dibedakan menjadi *local* LLM dan *cloud* LLM. *Local* LLM dijalankan pada perangkat atau server milik pengembang sendiri sehingga memberikan kontrol lebih besar terhadap data dan infrastruktur (Tri Utami Br Lubis et al., 2024).

Sebaliknya, *cloud* LLM diakses melalui layanan API dari penyedia model, sehingga integrasi menjadi lebih cepat dan lebih ringan dari sisi pengembang (Google, 2025a). Pada penelitian ini, teknologi LLM dikaitkan dengan penggunaan layanan Google Gemini melalui API. Pendekatan ini memungkinkan sistem memanfaatkan kemampuan

model bahasa besar untuk membantu penyusunan teks administratif, menjawab pertanyaan pengguna, dan menghasilkan keluaran yang lebih natural (Google, 2025b).

Google Gemini adalah keluarga model bahasa besar yang dikembangkan oleh Google DeepMind dan dapat diakses secara langsung oleh pengembang melalui Gemini API (Google, 2026a). Model ini tersedia dalam beberapa varian yang dirancang untuk kebutuhan yang berbeda, Gemini Flash dioptimalkan untuk kecepatan tinggi dan efisiensi biaya pada tugas-tugas yang membutuhkan respons cepat, sedangkan Gemini Pro dirancang untuk tugas yang membutuhkan penalaran lebih mendalam, penanganan konteks yang panjang, dan keluaran berkualitas tinggi (Google, 2026c). Keduanya mendukung pemrosesan teks dalam berbagai bahasa termasuk Bahasa Indonesia.

Dalam konteks AISYA-RA, Gemini API dipilih karena kemampuannya memahami instruksi kompleks dalam Bahasa Indonesia, menghasilkan teks terstruktur seperti RPPH dan surat resmi, serta merespons perintah percakapan dari *Chatbot* dengan akurasi yang memadai untuk lingkungan administrasi sekolah. Pengguna cukup mengetikkan perintah dalam bahasa sehari-hari, dan Gemini API akan menafsirkan perintah tersebut berdasarkan *system prompt* yang telah dirancang, lalu menghasilkan respons yang relevan secara administratif (Google, 2026b).

3.6.2 *Prompt Engineering*

Prompt engineering adalah praktik merancang dan mengoptimalkan instruksi masukan (*prompt*) yang diberikan kepada model bahasa besar *(LLM)* guna mengarahkan model menghasilkan keluaran yang akurat, terstruktur, dan sesuai konteks, tanpa memodifikasi parameter model yang mendasarinya (Sahoo et al., 2024). Alih-alih mengubah model itu sendiri, *prompt engineering* bekerja dengan cara mengoptimalkan cara berkomunikasi dengan model sehingga respons yang dihasilkan memenuhi kebutuhan spesifik pengguna atau sistem.

Dalam perkembangannya, *prompt engineering* mencakup berbagai teknik yang dapat diterapkan sesuai kebutuhan. *Zero-shot prompting* memberikan instruksi tugas tanpa contoh sebelumnya dan mengandalkan pengetahuan bawaan model. *Few-shot prompting* menyertakan beberapa contoh pasangan masukan-keluaran untuk memandu model

memahami pola yang diinginkan. *System prompt* digunakan untuk menetapkan perilaku umum model secara keseluruhan, termasuk peran, batasan, dan format respons yang diharapkan. Sementara itu, *chain-of-thought prompting* mendorong model untuk menghasilkan penalaran bertahap sebelum memberikan jawaban akhir, sehingga keluaran menjadi lebih akurat pada tugas yang kompleks (Sahoo et al., 2024).

Prompt engineering sangat relevan dalam pengembangan AISYA-RA karena hampir setiap keluaran sistem yang dihasilkan melalui Gemini API bergantung pada kualitas *prompt* yang dirancang. Pembuatan RPPH membutuhkan *prompt* yang memuat struktur pedagogik yang tepat, tema pembelajaran, dan kelompok usia peserta didik. Pembuatan surat resmi membutuhkan *prompt* yang menyertakan kaidah bahasa formal, format kelembagaan, dan informasi yang relevan. Selain itu, *system prompt Chatbot* perlu mendefinisikan peran *Chatbot* sebagai asisten administratif yang terbatas pada fungsi-fungsi sistem, sehingga respons yang dihasilkan tetap relevan dan tidak menyimpang dari konteks administrasi Raudhatul Athfal (Rosidin et al., 2024).

Melalui pendekatan *prompt engineering* yang terstruktur, AISYA-RA dapat memanfaatkan kemampuan generatif LLM secara terkendali. Keluaran yang dihasilkan tidak hanya akurat secara bahasa, tetapi juga sesuai dengan kebutuhan operasional lembaga, seperti format RPPH yang berlaku di RA Al-Islam Gunungcupu dan kaidah surat resmi yang digunakan oleh mitra.

3.6.3 *Function Calling / Tools Use* pada LLM

Function calling (juga disebut *tool use* atau *tool calling*) adalah kemampuan *Large Language Model (LLM)* untuk tidak hanya menghasilkan teks, tetapi juga memilih dan memanggil fungsi terstruktur yang telah didefinisikan sebelumnya berdasarkan maksud pengguna (Mauludin et al., 2026). Dengan *function calling*, LLM berperan sebagai komponen perencanaan (*planning component*) yang menerjemahkan bahasa alami pengguna menjadi instruksi terstruktur yang biasanya dalam format JSON yang dapat dieksekusi oleh sistem *backend* (Mauludin et al., 2026).

Mekanisme *function calling* bekerja melalui tahapan berikut (Mauludin et al., 2026):

1. Pendefinisian alat (*tool declarations*): Mendefinisikan daftar fungsi yang tersedia beserta nama, Deskripsi, dan skema parameter (tipe data, properti wajib) dalam format JSON Schema. Definisi ini disertakan dalam permintaan API ke LLM.

2. Pemilihan alat oleh LLM: Ketika pengguna mengirimkan pesan, LLM menganalisis maksud pengguna dan menentukan apakah ada fungsi yang relevan untuk dipanggil. Jika ya, LLM menghasilkan respons terstruktur yang berisi nama fungsi dan argumen yang sesuai.

3. Eksekusi oleh *backend*: Sistem *backend* menerima respons terstruktur dari LLM, mengeksekusi fungsi yang diminta (misalnya, menyimpan data ke basis data, menghasilkan dokumen, atau melakukan kueri), lalu mengembalikan hasil eksekusi kepada LLM.

4. Sintesis jawaban akhir: LLM menerima hasil eksekusi fungsi dan menyintesiskannya menjadi jawaban bahasa alami yang diterima pengguna.

Pendekatan *function calling* memberikan keunggulan dibandingkan metode tradisional berbasis *intent classification* NLP klasik. Dengan *function calling*, LLM tidak hanya mengenali maksud, tetapi juga mengekstrak parameter kompleks dari kalimat bahasa alami yang tidak terstruktur, memvalidasi argumen berdasarkan skema yang ditentukan, dan menghasilkan output terstruktur yang konsisten untuk diolah oleh sistem (Mauludin et al., 2026).

3.6.4 *Retrieval-Augmented Generation (RAG)*

Retrieval-Augmented Generation (RAG) adalah pendekatan yang menggabungkan kemampuan pencarian informasi dari sumber eksternal dengan kemampuan generatif dari model bahasa besar (Hajar et al., 2025).

Tujuan utama RAG adalah membuat model menjawab berdasarkan konteks yang relevan dan dapat dirujuk, sehingga mengurangi risiko jawaban yang halusinatif atau terlalu umum (Yasmin & Fudholi, 2025). Pada tingkat teknis, RAG biasanya bekerja dengan cara memecah dokumen menjadi potongan teks, mengubah potongan tersebut menjadi *embedding*, yakni representasi matematis dari teks dalam bentuk vektor yang

memungkinkan sistem mengukur kedekatan makna antar-teks, lalu mencocokkan *query* pengguna dengan potongan dokumen yang paling mirip secara semantik (Google, 2025c).

Hasil pencarian tersebut kemudian dimasukkan sebagai konteks tambahan sebelum model menghasilkan jawaban. Dalam konteks AISYA-RA, RAG sangat relevan apabila sistem diarahkan untuk menjawab atau menghasilkan keluaran berdasarkan dokumen lokal sekolah, seperti kalender pendidikan, pedoman administrasi, atau format surat (Mubarak & Sutopo, 2026). Dengan pendekatan ini, sistem dapat memberikan jawaban yang lebih kontekstual dan sesuai kebutuhan RA.

3.6.5 Konsep *Text Embedding*

Text embedding adalah teknik merepresentasikan teks ke dalam bentuk vektor numerik berdimensi tinggi sehingga makna semantik dari teks dapat diukur secara matematis (Rafli Aditya et al., 2025). Dalam proses embedding, setiap kata, kalimat, atau dokumen dipetakan ke dalam ruang vektor berdimensi n, di mana teks-teks yang memiliki makna serupa akan berada berdekatan dalam ruang vektor tersebut. Representasi vektor ini memungkinkan sistem untuk melakukan perbandingan semantik antar teks yang tidak terbatas pada pencocokan kata kunci, melainkan pada kedekatan makna (Rafli Aditya et al., 2025).

3.6.6 Semantic Search dan Cosine Similarity

Semantic search adalah teknologi temu kembali informasi yang mengembalikan hasil paling relevan berdasarkan maksud dan konteks dari kueri pencarian, bukan hanya pencocokan kata kunci (Lubis et al., 2026). Berbeda dengan pencarian tradisional yang bergantung pada kecocokan kata per kata (*keyword matching*), pencarian semantik memanfaatkan representasi vektor untuk mengukur kedekatan makna antara kueri pengguna dengan dokumen yang tersedia dalam basis data (Lubis et al., 2026).

Pengukuran kedekatan antar vektor dilakukan menggunakan metrik cosine similarity, yaitu perhitungan kosinus sudut antara dua vektor dalam ruang berdimensi n. Nilai *cosine similarity* berkisar dari -1 hingga 1, di mana nilai yang mendekati 1 menunjukkan kedekatan makna yang kuat antara dua teks, sedangkan nilai yang mendekati 0

menunjukkan tidak adanya kedekatan semantik (Rafli Aditya et al., 2025). Rumus cosine similarity dituliskan sebagai berikut:

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|}$$

di mana A·B adalah perkalian titik (dot *product*) antara vektor A dan B, sedangkan ‖A‖ dan ‖B‖ adalah norma (panjang) dari masing-masing vektor (Rafli Aditya et al., 2025).

3.6.7 pgvector: *Vector Database* pada PostgreSQL

pgvector adalah ekstensi PostgreSQL yang memungkinkan penyimpanan dan pencarian vektor berdimensi tinggi secara langsung di dalam basis data relasional (Supabase, 2026b). Dengan pgvector, pengembang dapat menyimpan embedding dokumen dalam tabel PostgreSQL, melakukan kueri pencarian kemiripan semantik (*similarity search*) menggunakan operator khusus, serta mengintegrasikan hasil pencarian tersebut dengan data relasional lainnya tanpa memerlukan sistem basis data vektor terpisah (Supabase, 2026b).

3.7 Metode *Agile*

3.7.1 Konsep *Agile*

Agile adalah pendekatan pengembangan perangkat lunak yang menitikberatkan pada kecepatan *delivery*, kolaborasi tim, keterlibatan pemangku kepentingan secara aktif, dan kemampuan beradaptasi terhadap perubahan kebutuhan kapan saja selama proses pengembangan berlangsung (Hidayah Nova et al., 2022). Berbeda dengan metode tradisional seperti *Waterfall* yang bersifat linear dan sekuensial, metode *Agile* membagi siklus pengembangan sistem *(SDLC)* menjadi beberapa iterasi *timebox* yang lebih kecil, di mana setiap iterasi menghasilkan perangkat lunak yang dapat diuji dan dievaluasi secara langsung oleh pengguna (Hidayah & Asnadi, 2024). Pendekatan ini memungkinkan pengembang mendeteksi dan memperbaiki masalah lebih cepat dibandingkan harus menunggu seluruh sistem selesai dibangun.

Metode *Agile* telah terbukti efektif dalam pengembangan aplikasi berbasis web karena melibatkan pengguna secara langsung dalam proses pengembangan, sehingga

menghasilkan produk yang lebih sesuai dengan kebutuhan nyata pengguna (Hidayah Nova et al., 2022). Hal ini dikonfirmasi oleh berbagai penelitian pengembangan sistem informasi di Indonesia yang menyatakan bahwa pendekatan *Agile* cocok diterapkan pada sistem yang menekankan kesederhanaan, fungsionalitas, dan kemampuan adaptasi terhadap perubahan kebutuhan, termasuk pada pengembangan sistem informasi administrasi institusional (R. Saputra et al., 2024).

Dalam pelaksanaannya, metode *Agile* mengandalkan konsep *timeboxing*, yaitu penetapan durasi waktu yang tetap untuk setiap iterasi guna memastikan fokus pengembangan dan ketepatan jadwal rilis produk. Selain itu, untuk menjamin kualitas pada setiap tahap, diperlukan standar yang disebut *Definition of Done* (DoD). DoD merupakan sekumpulan kriteria yang harus dipenuhi oleh sebuah fitur atau modul sebelum dinyatakan benar-benar selesai dan siap untuk ditinjau oleh pengguna. Penerapan kriteria ini penting dalam siklus iteratif untuk mencegah akumulasi kesalahan teknis pada tahap pengembangan berikutnya (Hidayah & Asnadi, 2024).

Metode *Agile* mengedepankan empat nilai inti yang dikenal sebagai *Agile Manifesto*. Keempat nilai tersebut adalah:

1. Individu dan interaksi lebih diprioritaskan daripada proses dan alat.
2. Perangkat lunak yang sudah berjalan lebih diutamakan daripada dokumentasi yang terperinci.
3. Kolaborasi dengan klien lebih diutamakan daripada negosiasi kontrak.
4. Merespons terhadap perubahan lebih diutamakan daripada mengikuti rencana.

Keempat nilai ini bukan berarti mengabaikan hal di sisi kanan, melainkan lebih menghargai hal di sisi kiri. Nilai-nilai ini menjadi fondasi filosofis yang membedakan *Agile* dari pendekatan pengembangan perangkat lunak tradisional (Pratama et al., 2022).

Pengembangan perangkat lunak dengan metode *Agile* merupakan proses iteratif dalam pembuatan perangkat lunak, di mana pengembangan dilakukan secara berulang dan sistematis (Pratama et al., 2022). Pendekatan ini memungkinkan perubahan dan

penyesuaian sesuai dengan kebutuhan yang berkembang sepanjang proses pengembangan. Prinsip utama dari *Agile* adalah kolaborasi tim, interaksi dengan pemangku kepentingan, dan adaptasi terhadap perubahan kebutuhan selama siklus proyek (Hidayah & Asnadi, 2024). Karakteristik ini menjadikan *Agile* sangat sesuai untuk proyek yang menuntut fleksibilitas tinggi dan keterlibatan pengguna secara langsung, seperti pada pengembangan sistem informasi administrasi institusional.

Metode *Agile* telah terbukti memberikan kontribusi positif dalam efisiensi, responsivitas terhadap perubahan, dan kualitas hasil proyek (Hidayah & Asnadi, 2024). Hasil *Systematic Literature Review* yang melibatkan 16 artikel jurnal terindeks menunjukkan bahwa penerapan metode *Agile* terbukti sebagai pendekatan yang efektif dalam berbagai jenis proyek, termasuk pengembangan perangkat lunak, aplikasi seluler, dan sistem informasi. Penerapan metode *Agile* juga meningkatkan kepuasan pengguna dan mempercepat manfaat yang dihasilkan dari proyek-proyek yang menggunakan pendekatan ini (Hidayah & Asnadi, 2024).

3.7.2 Pengembangan Iteratif dan Inkremental

Pengembangan iteratif dan inkremental merupakan inti dari metode *Agile*. Pendekatan ini membagi siklus pengembangan sistem *(SDLC)* menjadi beberapa iterasi *timebox* yang lebih kecil, di mana setiap iterasi menghasilkan perangkat lunak yang dapat diuji dan dievaluasi secara langsung oleh pengguna (Hidayah Nova et al., 2022). Berbeda dengan metode tradisional seperti *Waterfall* yang bersifat linear dan sekuensial, pendekatan iteratif memungkinkan pengembang mendeteksi dan memperbaiki masalah lebih cepat dibandingkan harus menunggu seluruh sistem selesai dibangun (Hidayah & Asnadi, 2024).

Pengembangan inkremental berarti bahwa sistem dibangun secara bertahap (*incremental*), di mana setiap kode atau fitur baru ditambahkan dan diintegrasikan ke dalam basis kode yang sudah ada tanpa merusak fungsionalitas dari iterasi sebelumnya. Setiap iterasi menghasilkan sebuah *increment*, yaitu perangkat lunak yang berfungsi dan memiliki nilai tambah dibandingkan versi sebelumnya (Hidayah & Asnadi, 2024). Sementara itu, pengembangan iteratif berarti bahwa proses pengembangan dilakukan

secara berulang, di mana setiap siklus dapat menyempurnakan, memperbaiki, atau menambah fungsionalitas berdasarkan evaluasi dari siklus sebelumnya (Pratama et al., 2022).

Karakteristik iteratif ini memungkinkan tim pengembang untuk mengevaluasi hasil sementara, menerima umpan balik dari pengguna, dan melakukan perbaikan pada setiap siklus pengembangan (Hidayah & Asnadi, 2024). Jika produk pada iterasi pertama belum cukup memenuhi kebutuhan, maka iterasi berikutnya dapat dikembangkan sesuai dengan sistem evaluasi pengguna (Pratama et al., 2022). Pendekatan ini sangat sesuai untuk proyek yang menuntut adaptasi cepat terhadap perubahan kebutuhan, karena pengembang tidak perlu menunggu seluruh sistem selesai untuk memvalidasi kesesuaian produk dengan kebutuhan pengguna.

Dalam AISYA-RA, pendekatan iteratif dan inkremental direalisasikan melalui beberapa iterasi pengembangan yang masing-masing menghasilkan fungsionalitas baru yang terintegrasi ke dalam sistem secara bertahap. Umpan balik dari pengguna pada akhir setiap iterasi langsung diakomodasi pada iterasi berikutnya, mencerminkan prinsip inti dari adaptabilitas metode *Agile*.

3.7.3 *Product* Backlog dan Sprint

Dalam pelaksanaan metode *Agile*, kebutuhan fungsional dan non-fungsional sistem dikumpulkan dan dikelola dalam sebuah daftar prioritas yang disebut *Product Backlog*. *Product Backlog* berfungsi sebagai daftar prioritas utama yang memuat seluruh fitur yang harus dibangun, di mana semakin penting suatu kebutuhan (*user story*), posisinya semakin tinggi dalam daftar *backlog* (Pratama et al., 2022). *Product Backlog* bersifat adaptif dan dapat diperbarui melalui proses *backlog refinement* pada setiap awal atau akhir iterasi berdasarkan umpan balik pengguna (Pratama et al., 2022).

Sprint atau Iterasi merupakan siklus pengembangan berulang dengan durasi waktu yang tetap (*timeboxed*), umumnya berkisar antara satu minggu hingga satu bulan (Pratama et al., 2022). Pada awal setiap *sprint*, dilakukan sprint planning untuk merencanakan pekerjaan yang akan dilakukan, yang hasilnya disebut *Sprint* Backlog, yaitu subset dari *Product Backlog* yang dialokasikan untuk dikerjakan pada sprint tersebut (Pratama et al.,

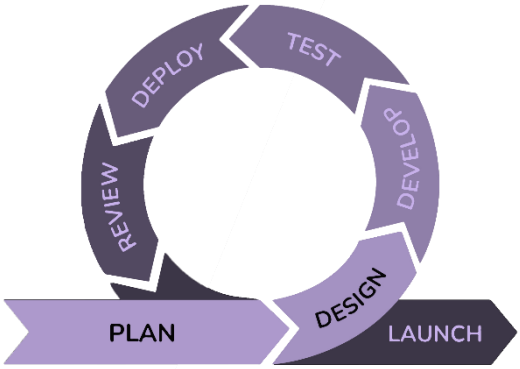
2022). Pada akhir setiap *sprint*, dilakukan *sprint review* untuk meninjau increment yang dihasilkan dan memperbarui *Product Backlog* jika diperlukan, serta *sprint retrospective* untuk menilai proses sprint yang telah selesai agar proses kerja dapat lebih lancar pada sprint berikutnya (Pratama et al., 2022).

Dalam AISYA-RA, konsep *Product Backlog* direalisasikan melalui penyusunan daftar prioritas fitur berdasarkan hasil observasi, wawancara, dan studi dokumentasi. *Sprint* direalisasikan dengan setiap iterasi mengikuti tahapan *Design*, *Develop*, *Test*, *Deploy*, dan *Review*. *Backlog refinement* dilakukan pada akhir setiap iterasi berdasarkan umpan balik dari sesi *Review* dengan Kepala RA dan perwakilan Guru RA, di mana kebutuhan baru yang teridentifikasi dimasukkan ke dalam perencanaan iterasi berikutnya.

3.7.4 Tahapan Metode *Agile*

Metode *Agile* dalam pengembangan sistem informasi umumnya dilaksanakan melalui lima tahapan yang dijalankan secara berulang pada setiap iterasi, Perencanaan (*Planning*), Desain (*Design),* Pengembangan (*Develop*), Pengujian (*Test)* dan Penyebaran (*Deployment*) .

Implementasi *Agile* dipilih karena fleksibilitasnya, keterlibatan pemangku kepentingan, dan proses iterasi yang memungkinkan perbaikan lebih cepat (Reza & Rahman, 2025). Metode *Agile* dalam pengembangan sistem informasi dilaksanakan melalui tahapan yang dijalankan secara berulang pada setiap iterasi, Perencanaan (*Plan*), Desain (*Design*), Pengembangan (*Develop*), Pengujian (*Test*), Penyebaran (*Deployment*), Evaluasi (*Review*), dan Peluncuran (*Launch*) (Reza & Rahman, 2025).



Gambar 3.1 Tahapan Metode
(Sumber:Reza & Rahman, 2025)

3.7.4.1 *Plan* (Perencanaan)

Tahap *Plan* (Perencanaan) merupakan langkah inisiasi dalam siklus *Agile* di mana kebutuhan fungsional dan non-fungsional sistem diidentifikasi dan diterjemahkan ke dalam *Product* Backlog (Hidayah & Asnadi, 2024). Berdasarkan daftar prioritas tersebut, alokasi pengerjaan fitur disusun ke dalam target iterasi. *Product* Backlog bersifat adaptif dan dapat diperbarui (*backlog* refinement) pada setiap awal atau akhir iterasi berdasarkan umpan balik pengguna (Pratama et al., 2022).

3.7.4.2 *Design* (Desain)

Tahap *Design* (Desain) dilakukan untuk merancang arsitektur perangkat lunak dan antarmuka pengguna berdasarkan item *backlog* yang telah ditetapkan pada iterasi yang sedang berjalan (Reza & Rahman, 2025). Pemodelan logika dan interaksi sistem disusun menggunakan standar *Unified Modeling Language (UML)*, sedangkan perancangan struktur data dimodelkan menggunakan *Entity Relationship Diagram (ERD)*. Rancangan visual antarmuka divisualisasikan melalui pembuatan *Wireframe* (Reza & Rahman, 2025). Pembuatan desain dilakukan secara parsial sesuai dengan target fitur pada masing-masing iterasi, sehingga perancangan menjadi lebih fokus dan siap untuk langsung diimplementasikan.

3.7.4.3 *Develop* (Pengembangan)

Tahap *Develop* (Pengembangan) merupakan proses penulisan kode program (*coding*) untuk merealisasikan rancangan dari tahap *Design* menjadi perangkat lunak (*increment*) yang dapat berfungsi (Reza & Rahman, 2025). Proses coding dilakukan secara bertahap (incremental), di mana setiap kode baru ditambahkan dan diintegrasikan ke dalam basis kode yang sudah ada tanpa merusak fungsionalitas dari iterasi sebelumnya (Hidayah & Asnadi, 2024). Karakteristik *incremental* ini memungkinkan sistem berkembang secara bertahap dengan setiap iterasi menambah nilai fungsional baru.

3.7.4.4 *Test* (Pengujian)

Tahap *Test* (Pengujian) dilaksanakan pada setiap akhir siklus pengembangan iterasi guna memvalidasi bahwa fungsionalitas yang baru saja dibangun telah bekerja sesuai dengan

rancangan (Reza & Rahman, 2025). Pengujian utama menggunakan metode *Black Box Testing*, yang berfokus pada kesesuaian antara *input* (masukan) pengguna dengan *output* (keluaran) sistem tanpa harus meninjau struktur internal kode program. Pada iterasi tingkat lanjut juga dilakukan uji regresi (*regression testing*) untuk memastikan bahwa penambahan kode atau fitur baru tidak menyebabkan kerusakan pada fitur-fitur yang telah dirilis pada iterasi sebelumnya.

3.7.4.5 *Deploy* (Penyebaran)

Tahap *Deploy* (Penyebaran) merupakan proses pendistribusian aplikasi agar dapat diakses dan digunakan (Reza & Rahman, 2025). Dalam pelaksanaan metode *Agile*, *deployment* dapat dilakukan secara bertahap, mulai dari *deployment* terbatas pada lingkungan lokal untuk memfasilitasi demonstrasi dan pengujian oleh pengguna, hingga *deployment* operasional penuh (*production*) pada server publik. Pendekatan bertahap ini memungkinkan risiko teknis tetap terisolasi pada tahap awal, sementara *deployment* operasional baru dilakukan ketika sistem telah mencapai stabilitas yang memadai (Reza & Rahman, 2025).

3.7.4.6 *Review* (Evaluasi)

Tahap *Review* (Evaluasi) merupakan sesi evaluasi yang melibatkan pengguna sebagai pemangku kepentingan utama sistem. Tahap ini dilaksanakan segera setelah *deployment* pada masing-masing iterasi selesai dilakukan (Reza & Rahman, 2025). Tujuan utama *review* adalah memberikan kesempatan bagi pengguna untuk berinteraksi dengan purwarupa (*prototype*), memvalidasi kesesuaian alur kerja sistem dengan praktik nyata di lapangan, serta mengevaluasi kemudahan antarmuka (Reza & Rahman, 2025).

Umpan balik yang diperoleh dari pengguna selama sesi *review* dikumpulkan dan diklasifikasikan. Perbaikan minor langsung diselesaikan, sedangkan penemuan celah kebutuhan atau permintaan fitur esensial yang baru diformulasikan menjadi *backlog* tambahan. Kebutuhan baru ini kemudian dimasukkan ke dalam perencanaan iterasi berikutnya, mencerminkan prinsip inti dari adaptabilitas metode *Agile* (Hidayah & Asnadi, 2024). Tahap *review* ini berfungsi sebagai mekanisme *sprint review* dan *sprint*

retrospective dalam kerangka Scrum, di mana hasil iterasi dievaluasi dan proses kerja dinilai untuk penyempurnaan pada iterasi berikutnya (Pratama et al., 2022).

3.7.4.7 *Launch* dan Evaluasi Operasional

Tahap *Launch* menandai berakhirnya seluruh rangkaian iterasi pengembangan, di mana sistem dinyatakan stabil dan diserahkan kepada pengguna untuk digunakan dalam kegiatan operasional yang sesungguhnya. Sistem dioperasikan secara penuh oleh pengguna selama periode tertentu tanpa pendampingan teknis langsung dari pengembang (Reza & Rahman, 2025). Periode penggunaan operasional ini bertujuan untuk memberikan waktu bagi pengguna untuk beradaptasi dengan sistem baru.

Setelah periode *launch* selesai, dilakukan evaluasi sistem secara komprehensif. Evaluasi mencakup pengukuran dampak efisiensi waktu pengerjaan administrasi dengan membandingkan durasi kerja sebelum dan sesudah adanya sistem, serta pengukuran tingkat penerimaan dan kegunaan sistem (*usability*) melalui instrumen *System Usability Scale (SUS)*. Tahap *launch* dan evaluasi operasional ini memastikan bahwa sistem tidak hanya berfungsi secara teknis, tetapi juga memberikan dampak nyata bagi pengguna dalam konteks operasional yang sesungguhnya (Hidayah & Asnadi, 2024).

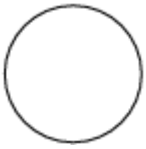
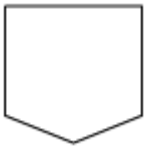
3.8 *Flowchart*

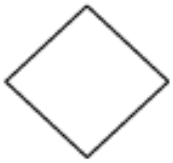
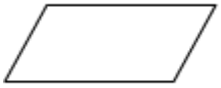
Flowchart atau bagan alir adalah representasi grafis yang menunjukkan urutan langkah, percabangan keputusan, serta arah aliran proses dalam suatu sistem (Rudi Sanjaya et al., 2022).

Flowchart digunakan untuk menyederhanakan logika kerja sistem agar lebih mudah dipahami oleh pengembang, pembimbing, maupun pengguna nonteknis (Adittiya et al., 2025). Pada sistem informasi seperti AISYA-RA, *Flowchart* bermanfaat untuk memodelkan proses prosedural, misalnya alur pembuatan surat, pengisian presensi, atau proses permintaan *Chatbot* yang menghasilkan keluaran administratif.

Dalam Tugas Akhir ini, *Flowchart* dapat digunakan sebagai pendahulu sebelum pembahasan UML yang lebih formal. *Flowchart* membantu pembaca memahami logika kerja dasar sistem, sedangkan UML dipakai untuk memodelkan fungsionalitas, aktivitas, struktur, dan interaksi sistem secara lebih rinci. Oleh karena itu, penggunaan *Flowchart* pada penelitian ini berfungsi sebagai alat visual awal untuk menjelaskan proses utama pada AISYA-RA (Rudi Sanjaya et al., 2022). Simbol *flowchart* disajikan pada Tabel 3.2.

Tabel 3.2 Simbol Flowchart

Simbol	Nama	Keterangan
	Flow	Menunjukkan arah aliran atau urutan langkah antar simbol dalam bagan alir.
	*On-Page Connector*	Menghubungkan dua bagian alur yang berada dalam satu halaman yang sama tanpa menarik garis panjang.
	*Off-Page Connector*	Menghubungkan alur proses yang berlanjut ke halaman atau lembar diagram yang berbeda

Simbol	Nama	Keterangan
	*Terminator*	Menandai titik awal (*Start*) atau titik akhir (*End*) dari sebuah proses atau program.
	*Process*	Menyatakan sebuah proses, perhitungan, atau operasi yang dilakukan oleh sistem secara otomatis.
	*Decision*	Menyatakan titik percabangan berdasarkan kondisi tertentu, biasanya menghasilkan dua atau lebih jalur keluaran (Ya/Tidak).
	*Input/output*	Menyatakan operasi pemasukan data (*input*) ke dalam sistem atau pengeluaran informasi (*output*) dari sistem.
	*Manual Operation*	Menyatakan proses yang dilakukan secara manual oleh pengguna, bukan oleh sistem komputer
	*Document*	Menyatakan data yang berasal dari dokumen fisik atau hasil keluaran yang berbentuk dokumen cetak.
	*Display*	Menyatakan keluaran yang ditampilkan langsung kepada pengguna melalui layar perangkat.
	*Preparation*	Menyatakan tahap inisialisasi atau persiapan, seperti penetapan nilai awal variabel sebelum proses utama dimulai.

3.9 *Unified Modeling Language (UML)*

Unified Modeling Language (UML) adalah bahasa pemodelan standar yang digunakan untuk memvisualisasikan, merancang, dan mendokumentasikan sistem perangkat lunak (Rudi Sanjaya et al., 2022)

UML membantu peneliti menggambarkan sistem dari sisi perilaku, aktivitas, struktur, dan interaksi antar komponen (Rudi Sanjaya et al., 2022). Penggunaan UML pada penelitian ini penting karena AISYA-RA bukan hanya sebuah halaman web sederhana, melainkan sistem yang terdiri atas banyak fungsi, aktor, dan komponen yang saling berinteraksi (Adittiya et al., 2025).

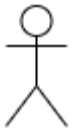
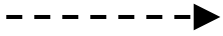
Melalui UML, peneliti dapat menunjukkan apa saja layanan sistem, bagaimana proses berlangsung, bagaimana struktur data dibangun, dan bagaimana komponen bertukar pesan.

3.9.1 *Use Case Diagram*

Use Case Diagram adalah diagram UML yang digunakan untuk menunjukkan hubungan antara aktor dan fungsi-fungsi yang disediakan sistem (Rudi Sanjaya et al., 2022). Fokus utamanya adalah menjawab pertanyaan siapa dapat melakukan apa di dalam sistem (Adittiya et al., 2025). Dalam AISYA-RA, aktor utama yang relevan adalah guru dan kepala RA. Guru dapat menggunakan sistem untuk menyusun RPPH, memasukkan data presensi, membuat surat, dan berinteraksi dengan *Chatbot.*

Kepala RA dapat mengakses fungsi yang lebih luas, misalnya pengelolaan data lembaga, arsip, atau pemantauan administratif. Dengan *Use Case Diagram*, hubungan peran dan layanan ini dapat divisualisasikan secara jelas dan sederhana. Hal ini membantu mempertegas ruang lingkup sistem yang dibangun pada penelitian ini. Simbol-simbol yang digunakan di *Use Case Diagram* disajikan pada Tabel 3.3.

Tabel 3.3 Simbol Use Case Diagram

Simbol	Nama	Keterangan
	*Actor*	Mewakili pengguna atau sistem eksternal yang berinteraksi dengan sistem yang sedang dimodelkan.
Label nama sistem	*System Boundary*	Menandai batas ruang lingkup sistem; *Use Case* yang berada di dalam kotak adalah bagian dari sistem, sedangkan aktor berada di luar.
	*Use Case*	Menggambarkan sebuah fungsionalitas atau layanan yang disediakan sistem kepada aktor, umumnya diberi nama menggunakan kata kerja.
	Association	Menghubungkan aktor dengan *Use Case*, menunjukkan bahwa aktor tersebut terlibat dalam *Use Case* yang dimaksud.
«include» 	*Include*	Menyatakan bahwa sebuah *Use Case* selalu memanggil dan menjalankan *Use Case* lain sebagai bagian wajib dari prosesnya

Simbol	Nama	Keterangan
«extend» 	*Extend*	Menyatakan bahwa sebuah *Use Case* dapat memperluas perilaku *Use Case* lain secara opsional, hanya pada kondisi tertentu.
	*Generalization*	Menyatakan hubungan pewarisan antara aktor atau *Use Case*, di mana elemen turunan mewarisi sifat dari elemen induknya.

3.9.2 *Activity Diagram*

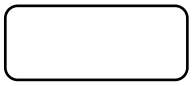
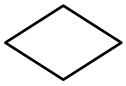
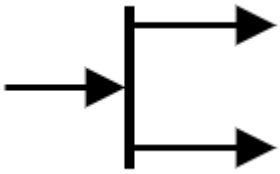
Activity Diagram adalah diagram UML yang digunakan untuk memodelkan alur kerja (*workflow*) atau aktivitas dalam suatu sistem, baik untuk proses bisnis, alur logis program, maupun interaksi pengguna dengan sistem (Rudi Sanjaya et al., 2022). Diagram ini menggambarkan urutan aktivitas dari awal hingga akhir, termasuk keputusan, percabangan, dan aktivitas paralel (Rahmanda Putri et al., 2025).

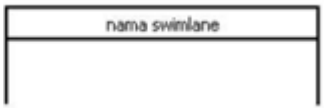
Activity Diagram berfungsi untuk memvisualisasikan tahapan kerja secara runtut, mulai dari input pengguna, validasi sistem, proses internal, hingga keluaran akhir. Komponen utama dalam *Activity Diagram* meliputi *Initial Node* sebagai tanda awal proses, *Activity* sebagai representasi langkah-langkah aktivitas, *Decision Node* untuk percabangan berdasarkan kondisi tertentu, dan *Final Node* yang menandakan akhir dari proses. Selain itu, diagram ini dapat menggunakan *Fork Node* dan *Join Node* untuk menggambarkan aktivitas yang berjalan secara paralel dan kembali bergabung. Untuk memperjelas tanggung jawab, Swimlanes sering digunakan untuk membagi aktivitas berdasarkan aktor atau unit kerja yang terlibat (Rahmanda Putri et al., 2025).

Diagram ini membantu pembaca melihat tahapan kerja secara runtut, mulai dari input pengguna, validasi sistem, proses internal di backend, pemanggilan layanan eksternal, hingga keluaran akhir yang ditampilkan pada antarmuka. Dengan demikian, *Activity*

Diagram menjadi sarana penting untuk menjelaskan proses yang diotomasi oleh sistem. Simbol-simbol yang digunakan di *Activity Diagram* disajikan pada Tabel 3.4.

Tabel 3.4 Simbol Activity Diagram

Simbol	Nama	Keterangan
	*Start*	Menandai titik awal dari suatu alur aktivitas; setiap *Activity Diagram* hanya boleh memiliki satu *initial node.*
	*Activity*	Mewakili sebuah langkah, tindakan, atau tugas yang harus dijalankan dalam proses. Nama aktivitas ditulis di dalam simbol.
	*Decision*	Menandai titik percabangan di mana alur terbagi menjadi dua atau lebih jalur berdasarkan kondisi yang dievaluasi (*guard condition*).
	*Fork / Join Node*	*Fork* memecah satu alur menjadi beberapa alur paralel yang berjalan bersamaan; *Join* menggabungkan kembali alur-alur paralel tersebut menjadi satu.
	*Flow Final*	Menandai berakhirnya satu cabang alur tertentu tanpa menghentikan keseluruhan proses diagram.
	*End Point*	Menandai akhir dari seluruh alur aktivitas dalam diagram; ketika titik ini dicapai, semua proses dinyatakan selesai.

Simbol	Nama	Keterangan
	*Control Flow*	Menghubungkan satu node aktivitas ke *node* berikutnya, menunjukkan urutan atau arah alur proses.
	*Swimlane*	Memisahkan organisasi bisnis yang bertanggung jawab terhadap aktivitas yang terjadi.

(Sumber: Rahmanda Putri, 2025)

3.9.3 *Class Diagram*

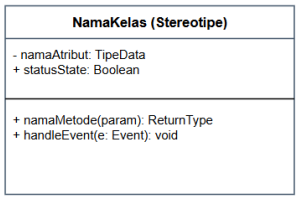
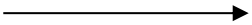
Class Diagram adalah diagram UML yang memodelkan struktur statis sistem, meliputi kelas, atribut, operasi (*metode*), dan hubungan antarkelas (Rudi Sanjaya et al., 2022). *Class Diagram* merupakan visual dari struktur sistem program pada jenis-jenis yang dibentuk dan juga dapat dikatakan sebagai kumpulan dari beberapa kelas beserta relasinya (Ramdany et al., 2024).

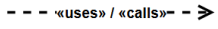
Selama proses analisis, *Class Diagram* memperlihatkan aturan-aturan dan tanggung jawab entitas yang menentukan perilaku sistem. Selama tahap desain, *Class Diagram* berperan dalam menangkap struktur dari semua kelas yang membentuk arsitektur sistem dan membantu memvisualisasikan struktur kelas-kelas dari suatu sistem (Ramdany et al., 2024). *Class Diagram* secara khas meliputi kelas (*Class*), relasi asosiasi (*Association*), generalisasi (*Generalization*) dan agregasi (*Aggregation*), atribut (*Attributes*), operasi/metode (*Operation/Method*), serta visibilitas (*Visibility*) yang merupakan tingkat akses objek eksternal kepada suatu operasi atau atribut. Hubungan antarkelas mempunyai keterangan yang disebut dengan *Multiplicity* atau *Cardinality* (Ramdany et al., 2024).

Sebuah kelas digambarkan sebagai sebuah kotak yang terbagi menjadi tiga bagian, bagian atas berisi nama kelas, bagian tengah berisi atribut, dan bagian bawah berisi metode atau operasi (Ramdany et al., 2024). Dalam penelitian sistem informasi, *Class Diagram* penting karena dapat membantu memvisualisasikan entitas utama dan relasinya sebelum diwujudkan ke dalam basis data atau kode program (Rudi Sanjaya et al., 2022). Pada

AISYA-RA, *Class Diagram* digunakan untuk memodelkan entitas serta membagi arsitektur sistem ke dalam lapisan (*layer*) seperti *View Layer, Logic & Controller Layer,* dan *Data Entity Laye*r. Dengan demikian, *Class Diagram* berperan sebagai jembatan antara analisis kebutuhan dan rancangan data sistem. Simbol-simbol yang digunakan di *Class Diagram* disajikan pada Tabel 3.5.

Tabel 3.5 Simbol Class Diagram

Simbol	Nama	Keterangan
	*Class*	Blok pembangunan pada pemrograman berorientasi objek. Kotak terbagi menjadi tiga bagian: nama kelas (atas), atribut (tengah), dan metode/operasi (bawah)
	*Association*	Hubungan yang menunjukkan adanya interaksi antarkelas. Garis dengan mata panah terbuka di ujungnya mengindikasikan adanya aliran pesan dalam satu arah
	*Aggregation*	Relasi "keseluruhan–bagian" (*whole-part*) yang bersifat lemah; bagian dapat eksis secara independen tanpa keseluruhan
	*Composition*	Relasi "keseluruhan–bagian" yang bersifat kuat; bagian tidak dapat berdiri sendiri tanpa keseluruhan. Digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid

Simbol	Nama	Keterangan
	*Generalization*	Hubungan pewarisan antarkelas yang bersifat dari khusus ke umum; kelas turunan mewarisi atribut dan metode dari kelas induk
	*Dependency*	Menunjukkan bahwa suatu kelas menggunakan kelas lain. Dilambangkan sebagai panah bertitik-titik
+, −, #	*Visibility*	Tingkat akses objek eksternal terhadap atribut atau operasi: + (*public*), − (*private*), # (*protected*)
1, 0.., 1..	*Multiplicity*	Keterangan pada hubungan antarkelas yang menunjukkan jumlah instance yang dapat terlibat dalam relasi, misalnya 1 (satu), 0..* (nol atau lebih), 1..* (satu atau lebih)

(Sumber: Ramdany et al., 2024)

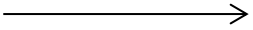
3.9.4 *Sequence Diagram*

Sequence Diagram adalah diagram UML yang digunakan untuk menggambarkan interaksi antarobjek dalam urutan waktu tertentu, yang menunjukkan bagaimana objek saling berkomunikasi untuk menyelesaikan suatu proses atau fungsi tertentu (Shidqi et al., 2025). Diagram ini memetakan alur pesan atau perintah yang dikirim antara objek-objek sistem, serta urutan eksekusi metode yang diperlukan untuk mencapai hasil yang diinginkan. Tujuan utamanya adalah menunjukkan rangkaian pesan yang dikirim antar objek, interaksi antar objek, serta peristiwa yang terjadi pada titik tertentu dalam eksekusi sistem (Shidqi et al., 2025).

Sequence Diagram memiliki dimensi vertikal yang merepresentasikan waktu dan dimensi horizontal yang merepresentasikan objek-objek yang terkait. Komponen utama dari *Sequence Diagram* meliputi objek yang digambarkan dengan kotak bernama, pesan yang diwakili oleh garis dengan panah, dan waktu yang ditunjukkan oleh garis vertikal (Shidqi et al., 2025). Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal berupa garis putus-putus yang menandakan keberadaan objek selama interaksi berlangsung. Message digambarkan sebagai garis berpanah dari satu objek ke objek lainnya, menunjukkan pengiriman pesan atau pemanggilan fungsi. *Activation bar* berupa kotak sempit vertikal pada *lifeline* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah message. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi atau metode dari kelas. Simbol-simbol yang digunakan di *Sequence Diagram* disajikan pada Tabel 3.6.

Tabel 3.6 Simbol Sequence Diagram

Simbol	Nama	Keterangan
Nama	*Object / Participant*	Objek atau komponen yang terlibat dalam interaksi. Diletakkan di bagian atas diagram, diurutkan dari kiri ke kanan
	Lifeline	Garis titik-titik yang terhubung dengan objek, menandakan keberadaan objek selama interaksi berlangsung. Sepanjang *lifeline* terdapat *activation bar*
	Activation Bar	Mewakili eksekusi operasi dari objek; panjang kotak berbanding lurus dengan durasi aktivitas sebuah operasi. Biasanya diawali dengan diterimanya sebuah *message*

Simbol	Nama	Keterangan
	Synchronous Message	Pesan yang dikirim dari satu objek ke objek lain dan menunggu balasan sebelum melanjutkan eksekusi.
	Return Message	Pesan balasan yang dikembalikan dari objek penerima ke objek pengirim setelah operasi selesai.
	*Asynchronous Message*	Pesan yang dikirim tanpa menunggu balasan; pengirim dapat melanjutkan eksekusi tanpa harus menunggu respons.
	*Recursive Message / Message to Self*	Menggambarkan pengiriman pesan yang dikirim untuk dirinya sendiri

(Sumber: Shidqi et al., 2025)

3.10 Basis Data (*Database*)

3.10.1 Konsep Basis Data

Basis data adalah kumpulan data terstruktur yang disimpan secara sistematis agar mudah diakses, dikelola, diperbarui, dan ditelusuri kembali (PostgreSQL, 2026a).

Dalam sistem informasi, basis data menjadi fondasi utama penyimpanan karena memungkinkan data disusun secara konsisten dan saling berhubungan (Supabase, 2026a). Dalam sistem pendidikan, basis data berfungsi menyimpan data siswa, data guru, data kehadiran, dokumen pembelajaran, serta arsip administrasi lain dalam satu struktur yang terpusat (Prayoga, 2024).

Basis data yang baik membantu mengurangi redundansi, menjaga integritas informasi, dan memudahkan penelusuran data historis (Adittiya et al., 2025). Dalam penelitian ini,

basis data menjadi tulang punggung AISYA-RA karena seluruh fungsi administratif bergantung pada penyimpanan data yang rapi dan terhubung. Oleh sebab itu, basis data tidak hanya dipahami sebagai media penyimpanan, tetapi sebagai fondasi integrasi layanan sistem (Supabase, 2026c).

3.10.2 PostgreSQL

PostgreSQL adalah sistem manajemen basis data relasional tingkat lanjut yang dikenal kuat, andal, dan kaya fitur (PostgreSQL, 2026a).

PostgreSQL mendukung relasi data, *foreign key*, *constraint* integritas, indeks, transaksi, dan berbagai tipe data modern (PostgreSQL, 2026b). Keunggulan tersebut membuat PostgreSQL banyak digunakan pada aplikasi yang membutuhkan penyimpanan data yang stabil dan skalabel, termasuk aplikasi web dan sistem informasi pendidikan.

Selain mendukung data relasional, PostgreSQL juga relevan pada pengembangan sistem modern yang memerlukan integrasi dengan teknologi AI, terutama ketika dipadukan dengan ekstensi *vector Database* seperti *pgvector* (Supabase, 2026b). Dalam konteks penelitian ini, PostgreSQL relevan karena merepresentasikan landasan teoritis dari penyimpanan data AISYA-RA, termasuk bila implementasi praktisnya menggunakan layanan seperti Supabase. Dengan PostgreSQL, data siswa, presensi, RPPH, surat, dan konteks dokumen dapat dikelola dalam satu sistem basis data yang kuat dan terstruktur (Supabase, 2026a).

3.10.3 Supabase

Supabase adalah platform *backend*-as-a-*service* yang menyediakan layanan PostgreSQL terkelola (*managed* PostgreSQL) disertai dengan fitur autentikasi, penyimpanan *file*, API RESTful otomatis, dan koneksi real-time melalui satu infrastruktur terintegrasi berbasis *cloud* (Rosidin et al., 2024). Sebagai platform yang bersifat *open-source*, Supabase memungkinkan pengembang mengelola basis data PostgreSQL tanpa harus menyiapkan dan mengkonfigurasi server basis data secara mandiri.

Seluruh operasi basis data pada Supabase dapat diakses melalui koneksi PostgreSQL standar, sehingga kompatibel dengan berbagai *framework backend* modern termasuk (Supabase, 2026a). Dengan pendekatan ini, pengembang dapat memanfaatkan ekosistem

PostgreSQL sepenuhnya, termasuk dukungan ekstensi seperti pg*vector* yang memungkinkan penyimpanan dan pencarian *embedding vektor* untuk keperluan fitur RAG. (Supabase, 2026b).

Dalam pengembangan AISYA-RA, Supabase diposisikan sebagai platform pengelola basis data yang memisahkan infrastruktur penyimpanan data dari server aplikasi. Server aplikasi yang berjalan pada DigitalOcean *Droplet* terhubung ke basis data PostgreSQL yang dikelola melalui Supabase, sehingga pengelolaan sistem menjadi lebih modular, skalabel, dan mudah dipelihara (Supabase, 2026c). Pendekatan pemisahan ini juga sejalan dengan prinsip arsitektur *cloud-native* yang memisahkan antara lapisan aplikasi dan lapisan data.

3.10.4 *Entity Relationship Diagram (ERD)*

Entity Relationship Diagram (ERD) adalah diagram berbentuk notasi grafis yang berada dalam pembuatan basis data yang menghubungkan antara data satu dengan yang lain (Diva et al., 2025). ERD merupakan salah satu alat utama dalam perancangan basis data yang menggambarkan keterhubungan antarentitas, tipe objek mengenai data yang dikelola, serta relasi antara objek tersebut (Raziqin et al., 2025).

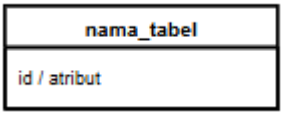
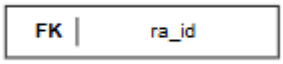
Di dalam ERD terdapat tiga elemen dasar, yaitu entitas, atribut, dan relasi. Entitas merupakan objek yang akan menjadi perhatian dalam suatu basis data, entitas dapat berupa manusia, tempat, benda, atau kondisi mengenai data yang dibutuhkan. Atribut merupakan karakteristik atau properti yang dimiliki oleh suatu entitas, yang memberikan Deskripsi lebih rinci tentang entitas tersebut. Relasi merupakan hubungan yang mengaitkan antara satu entitas dengan entitas lainnya (Diva et al., 2025). Selain ketiga elemen dasar tersebut, ERD juga mengenal konsep kardinalitas (*cardinality*), yaitu batasan jumlah instance suatu entitas yang dapat berhubungan dengan instance entitas lain. Jenis kardinalitas meliputi *one-to-one*, *one-to-many*, dan *many-to-many* (Diva et al., 2025).

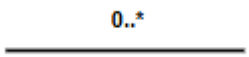
Dalam perancangan sistem informasi, ERD penting untuk mengidentifikasi hubungan antardata sekaligus meminimalkan redundansi data dan potensi anomali saat proses penyimpanan maupun pembaruan data. Hasil penelitian menunjukkan bahwa ERD dapat

meningkatkan efisiensi perancangan sistem, mengurangi kesalahan logis, dan memperbaiki komunikasi antara pengembang dan pemangku kepentingan (Diva et al., 2025). Dengan pemodelan yang baik, ERD menjadi jembatan antara analisis kebutuhan pengguna dan implementasi basis data fisik pada DBMS (Mafirah et al., 2026).

Pada AISYA-RA, ERD relevan karena sistem mengelola data yang saling terhubung. Melalui ERD, relasi antarentitas dapat dirancang secara konsisten sehingga pengembangan basis data pada PostgreSQL menjadi lebih terstruktur, mudah dipelihara, dan mendukung kebutuhan integrasi antarmodul sistem (Mafirah et al., 2026). Simbol-simbol yang digunakan di ERD disajikan pada Tabel 3.7.

Tabel 3.7 Simbol ERD

Simbol	Nama	Keterangan
	*Entitas / Tabel*	Mewakili objek atau entitas basis data yang menyimpan sekumpulan data terstruktur dalam sistem
	*Primary Key (PK)*	Atribut unik utama yang menjadi identitas tunggal untuk setiap baris data di dalam entitas/tabel (contoh: id ber-tipe UUID).
	*Foreign Key (FK)*	Atribut kunci asing yang merujuk pada *Primary Key* entitas lain untuk membangun keterhubungan (relasi) antar entitas.
	*Unique Constraint [UQ]*	Menandakan atribut dengan batasan keunikan (Unique Constraint), sehingga nilainya

Simbol	Nama	Keterangan
		tidak boleh duplikat di seluruh baris tabel (contoh: email).
	*Atribut & Tipe Data*	Mewakili bidang data (kolom) yang dimiliki oleh entitas beserta jenis/tipe data yang disimpannya (contoh: VARCHAR, TEXT, UUID, TIMESTAMP, BOOLEAN, DATE).
	Relationship Line *(Garis Relasi)*	Garis penghubung antar entitas yang menunjukkan adanya asosiasi atau hubungan keterikatan data melalui *Primary Key* dan *Foreign Key*.
	*Kardinalitas One (1)*	Menunjukkan kardinalitas tepat satu (*One*). Setiap satu rekaman pada entitas asal (*Parent*) terikat secara pasti pada satu entitas.
	*Kardinalitas Zero to Many (0..*)*	Menunjukkan kardinalitas nol hingga banyak (*Many*). Satu rekaman pada entitas utama dapat memiliki nol, satu, atau lebih banyak rekaman pada entitas anak (*Child*).

(Sumber: Draw.io;Diva et al., 2025)

3.11 Bahasa Pemrograman

3.11.1 Python

Python adalah bahasa pemrograman tingkat tinggi yang dikenal karena sintaksnya sederhana, mudah dibaca, dan mendukung pengembangan perangkat lunak secara cepat (FastAPI, 2026b).

Python banyak digunakan pada pengembangan *backend*, automasi, analisis data, dan integrasi AI karena memiliki ekosistem pustaka yang luas (FastAPI, 2026a). Keunggulan tersebut menjadikan Python sangat sesuai untuk proyek yang memadukan logika aplikasi dengan pengolahan data dan layanan kecerdasan buatan.

Pada sistem modern, Python sering dipilih karena dapat berfungsi sebagai bahasa orkestrasi yang menghubungkan berbagai komponen, seperti API, basis data, layanan dokumen, dan model AI. Dalam penelitian ini, Python menjadi penting sebagai dasar pengembangan sisi *backend* AISYA-RA. Melalui Python, sistem dapat mengelola logika administrasi, menerima permintaan dari antarmuka web, menghubungkan data dengan basis data, dan berkomunikasi dengan model AI (FastAPI, 2026a).

3.11.2 JavaScript

JavaScript adalah bahasa pemrograman utama yang digunakan untuk membuat halaman web menjadi dinamis dan interaktif (React, 2026c).

Melalui JavaScript, antarmuka web dapat merespons aksi pengguna, memperbarui tampilan tanpa memuat ulang seluruh halaman, serta berkomunikasi dengan *backend* melalui API (React, 2026b). Hal ini menjadikan JavaScript sebagai fondasi penting dalam pengembangan aplikasi web modern.

Keunggulan JavaScript terletak pada kemampuannya mendukung pengalaman pengguna yang responsif. Pada aplikasi administrasi, JavaScript memungkinkan formulir bekerja secara interaktif, data tampil lebih cepat, dan navigasi berlangsung lebih halus. Karena aplikasi berbasis web seperti AISYA-RA menuntut interaktivitas tinggi, JavaScript menjadi bahasa yang sangat relevan untuk membangun pengalaman penggunaan yang nyaman bagi guru dan kepala RA (React, 2026a).

3.12 Perancangan Antarmuka Pengguna

3.12.1 *Wireframe* dan *Mockup*

Wireframe merupakan rancangan awal antarmuka yang menampilkan struktur halaman, tata letak elemen, dan alur navigasi tanpa menekankan detail visual secara penuh (Fatah et al., 2022). *Wireframe* biasanya dibuat dalam fidelitas rendah (*low-fidelity*) sehingga memudahkan pengembang dan pengguna untuk fokus pada susunan fungsi, bukan pada aspek estetika terlebih dahulu (Darmawan et al., 2024). Pembuatan kerangka *Wireframe low-fidelity* dilakukan melalui penggambaran antarmuka secara manual menggunakan media kertas dan bolpoin, atau menggunakan perangkat lunak sederhana (Darmawan et al., 2024).

Mockup merupakan representasi visual antarmuka yang lebih detail dibanding *Wireframe* karena sudah menampilkan elemen warna, tipografi, ikon, dan komponen tampilan yang mendekati kondisi sistem akhir (Fatah et al., 2022). *Mockup* berada pada tingkat fidelitas tinggi *(high-fidelity)* yang merupakan bentuk realisasi dari kerangka antarmuka low-fidelity yang sudah dibuat sebelumnya. Pada tahapan pengembangan rancangan tampilan *high-fidelity*, perangkat lunak seperti Figma atau Draw.io dapat digunakan untuk merealisasikan kerangka antarmuka menjadi desain yang lebih kompleks dan realistis (Darmawan et al., 2024).

Metode *Wireframe Prototyping* menekankan pada perancangan kerangka aplikasi dari mulai sederhana dengan kerangka tulisan tangan (*low-fidelity*) sampai dengan kompleks (*high-fidelity*) sehingga memudahkan untuk merealisasikan keinginan pengguna secara realistis. Salah satu aspek pembuatan tampilan antarmuka adalah harus mencapai standar ramah pengguna atau *user friendly* (Darmawan et al., 2024). Dengan adanya *Wireframe* dan mockup, proses validasi desain menjadi lebih mudah karena pengguna dapat membayangkan pengalaman penggunaan sistem sebelum implementasi dilakukan (Fatah et al., 2022).

3.13 *Framework* (Kerangka Kerja)

3.13.1 React.js

React.js adalah pustaka JavaScript untuk membangun antarmuka pengguna berbasis komponen (React, 2026c).

Dalam React, tampilan aplikasi disusun dari komponen-komponen kecil yang dapat digunakan ulang, sehingga pengembangan antarmuka menjadi lebih modular, terstruktur, dan mudah dipelihara (React, 2026a). Pendekatan ini sangat sesuai untuk aplikasi yang memiliki banyak elemen interaktif dan tampilan yang terus berubah sesuai *input* pengguna.

React juga mendukung pengelolaan *state* antarkomponen, sehingga data pada antarmuka dapat diperbarui secara efisien tanpa harus memuat ulang seluruh halaman (React, 2026b). Dalam konteks AISYA-RA, React relevan karena sistem memadukan *dashboard*, formulir administrasi, data presensi, persuratan, dan *Chatbot* dalam satu aplikasi web. Dengan struktur berbasis komponen, antarmuka dapat dibangun lebih konsisten, mudah dikembangkan, dan tetap responsif untuk pengguna nonteknis.

3.13.2 FastAPI

FastAPI adalah *framework* Python modern yang digunakan untuk membangun API dengan cepat dan efisien (FastAPI, 2026b).

FastAPI mendukung deklarasi rute yang jelas, validasi data, *type hints* Python, serta dokumentasi API otomatis (FastAPI, 2026a). Karakteristik ini menjadikannya sangat sesuai untuk pengembangan *backend* yang harus menghubungkan *frontend*, basis data, dan layanan AI dalam satu sistem yang terintegrasi.

Salah satu keunggulan FastAPI adalah kemampuannya menyederhanakan pengembangan layanan *backend* tanpa mengurangi kualitas struktur aplikasi. Dalam AISYA-RA, FastAPI sangat relevan sebagai kerangka kerja *backend* yang menjembatani antarmuka React dengan basis data dan model AI. Melalui API yang dibangun menggunakan

FastAPI, sistem dapat menerima *input* pengguna, memproses data administratif, mengakses basis data, dan meneruskan permintaan ke layanan AI (FastAPI, 2026a)

3.13.3 REST API

REST API (*Representational State Transfer Application Programming Interface*) adalah gaya arsitektur layanan web yang memungkinkan pertukaran data antara *client* dan server melalui *endpoint* berbasis HTTP. Pendekatan ini banyak digunakan karena sederhana, interoperabel, dan mendukung komunikasi data secara terstandar pada aplikasi modern (Olivia, 2025).

Pada REST API, *resource* diakses melalui URL dan dioperasikan dengan metode standar seperti GET, POST, PUT, dan DELETE. Pola ini memudahkan integrasi antarlayanan karena setiap permintaan bersifat *stateless* dan setiap respons dapat dikirim dalam format data yang ringan seperti JSON (FastAPI, 2026a).

Dalam AISYA-RA, REST API relevan karena antarmuka React, layanan *backend* FastAPI, basis data, dan integrasi Gemini API perlu saling berkomunikasi secara konsisten. Dengan menggunakan pendekatan REST API, proses *login*, pengambilan data, penyimpanan presensi, pembuatan RPPH, dan pemrosesan permintaan *Chatbot* dapat diorkestrasi melalui layanan yang lebih modular dan mudah diuji (Simbulan & Aryanto, 2024).

3.14 Perangkat Lunak Tambahan (*Tools*)

3.14.1 Visual Studio Code (VS Code)

Visual Studio Code adalah editor kode lintas platform yang mendukung banyak bahasa pemrograman, *debugging*, terminal terintegrasi, ekstensi, dan *version control* (Microsoft, 2026c).

Sebagai editor modern, VS Code dirancang untuk membantu pengembang menulis, menavigasi, menguji, dan memelihara kode dalam satu lingkungan kerja (Microsoft, 2026b). Karena sifatnya ringan tetapi kaya fitur, VS Code banyak digunakan dalam pengembangan perangkat lunak modern.

Keunggulan VS Code terletak pada kemampuannya mendukung produktivitas pengembang secara menyeluruh. Fitur seperti *code navigation*, *integrated terminal*, *debugging*, dan dukungan ekstensi membantu pengembang mengelola proyek *full-stack* dengan lebih efisien. Pada penelitian ini, VS Code relevan sebagai perangkat lunak pendukung pengembangan AISYA-RA karena proyek menggabungkan *frontend*, *backend*, API, dan layanan tambahan dalam satu aplikasi (Microsoft, 2026a).

3.14.2 Windows Subsystem for Linux (WSL)

Windows Subsystem for Linux (WSL) adalah fitur bawaan sistem operasi Windows yang memungkinkan pengembang menjalankan lingkungan GNU/Linux secara penuh, termasuk alat baris perintah, utilitas, dan aplikasi langsung di atas Windows, tanpa memerlukan mesin virtual konvensional maupun konfigurasi *dual-boot* (Microsoft, 2026d). Dengan WSL, pengembang dapat menggunakan *shell* Linux seperti *Bash*, menjalankan perintah-perintah Linux, serta mengelola paket menggunakan apt, semuanya dalam satu sistem operasi Windows yang tetap berjalan secara bersamaan.

WSL tersedia dalam dua versi utama yang memiliki perbedaan arsitektur mendasar. WSL 1 bekerja sebagai lapisan kompatibilitas yang menerjemahkan *system call* Linux ke dalam *kernel* Windows NT, sehingga tidak semua program Linux dapat berjalan dengan sempurna. WSL 2, yang merupakan versi *default* sejak Windows 10 *build* 18362, menggunakan *kernel* Linux nyata yang berjalan di dalam mesin virtual ringan berbasis *Hyper-V* (Microsoft, 2026d). Perubahan arsitektur ini menjadikan WSL 2 jauh lebih kompatibel dengan ekosistem Linux, termasuk dukungan penuh terhadap seluruh *system call*, performa *file* I/O yang lebih cepat, serta kompatibilitas penuh dengan Docker Desktop yang mengandalkan fitur virtualisasi WSL 2.

Salah satu keunggulan utama WSL dalam konteks pengembangan perangkat lunak adalah kemampuannya menjaga konsistensi lingkungan antara mesin pengembang dan server produksi. Karena server produksi umumnya berbasis Linux, penggunaan WSL memungkinkan pengembang menggunakan perintah, skrip, dan alat yang identik dengan yang berjalan di server, sehingga risiko perbedaan perilaku aplikasi antara lingkungan pengembangan dan produksi dapat diminimalkan (Microsoft, 2026d). Selain itu, WSL

dapat diintegrasikan langsung dengan Visual Studio Code melalui ekstensi *Remote*–WSL, memungkinkan penulisan kode di Windows dengan eksekusi yang terjadi di dalam lingkungan Linux.

Dalam pengembangan AISYA-RA, WSL 2 dengan distribusi Ubuntu 24.04 LTS digunakan sebagai lingkungan pengembangan utama pada laptop ThinkPad L440 yang menjalankan Windows 11 Pro. Pilihan ini didasarkan pada pertimbangan bahwa server produksi berbasis DigitalOcean *Droplet* juga menjalankan Ubuntu Linux, sehingga perintah-perintah untuk mengelola *container* Docker, menjalankan *backend* FastAPI, dan mengonfigurasi layanan dapat diuji dan dijalankan di lingkungan pengembangan yang secara arsitektur identik dengan server produksi.

3.14.3 Kontainerisasi Aplikasi dengan Docker

Docker adalah platform perangkat lunak *open-source* yang dirancang untuk mempermudah proses pengembangan, pengemasan, dan pendistribusian aplikasi di dalam *container*, unit perangkat lunak yang mengemas kode aplikasi beserta seluruh dependensinya dalam satu lingkungan terisolasi dan portabel (Docker, 2026d). Docker pertama kali diperkenalkan pada tahun 2013 dan sejak saat itu menjadi salah satu teknologi paling fundamental dalam ekosistem pengembangan perangkat lunak modern, khususnya dalam paradigma DevOps dan *cloud-native development* (Ariadi et al., 2020).

Konsep utama yang mendasari Docker adalah kontainerisasi. Sebuah *container* berjalan di atas *kernel* sistem operasi induk yang sama, tetapi memiliki *namespace* dan lapisan sistem *file* yang terisolasi. Hal ini menjadikan *container* jauh lebih ringan dibandingkan mesin virtual (*virtual machine*) konvensional yang memerlukan sistem operasi tersendiri untuk setiap instansnya. *Container* Docker hanya memerlukan sumber daya CPU sebesar 4–12,7% dari kapasitas penuh server, jauh lebih efisien dibandingkan teknik virtualisasi berbasis *hypervisor* (Ariadi et al., 2020).

Docker memiliki beberapa komponen inti yang bekerja secara terpadu dalam siklus pengembangan dan *Deployment* aplikasi:

1. Docker Image adalah *blueprint* atau cetakan *read-only* yang mendefinisikan seluruh isi *container*, termasuk sistem *file* dasar, kode aplikasi, *library*, variabel

lingkungan, dan perintah eksekusi. *Image* dibangun dari instruksi yang ditulis dalam sebuah *file* teks bernama Dockerfile. Setiap instruksi dalam Docker*file* menghasilkan lapisan (*layer*) baru pada *image,* dan layer yang tidak berubah dapat digunakan kembali secara efisien melalui mekanisme layer *caching* (Docker, 2026d).

2. Docker Container adalah instans *image* yang sedang berjalan. Satu *image* yang sama dapat dijalankan menjadi banyak *container* secara bersamaan. Setiap *container* bersifat terisolasi, perubahan yang terjadi di dalam satu *container* tidak memengaruhi *image* aslinya maupun *container* lain. *Container* dapat dihentikan, dihapus, dan dibuat ulang kapan saja, menjadikan proses *Deployment* bersifat *reproducible* (Docker, 2026c).
3. Dockerfile adalah *file* konfigurasi teks yang berisi serangkaian instruksi untuk membangun sebuah Docker Image. Instruksi umum yang digunakan antara lain FROM untuk menentukan *image* dasar, COPY untuk menyalin *file* aplikasi, RUN untuk menjalankan perintah saat proses *build*, EXPOSE untuk mendeklarasikan *port* yang digunakan, serta CMD atau ENTRYPOINT untuk mendefinisikan perintah yang dijalankan saat *container* dimulai (Docker, 2026b).
4. Docker Compose adalah alat yang digunakan untuk mendefinisikan dan menjalankan aplikasi yang terdiri dari beberapa *container* sekaligus (*multi-container application*) melalui satu *file* konfigurasi berformat YAML bernama *docker-compose.yml*. Di dalam *file ini*, setiap layanan (*service*) didefinisikan beserta *image, port,* variabel lingkungan, *volume*, dan ketergantungan antar layanan. Dengan satu perintah *docker compose up*, seluruh layanan dapat dijalankan secara bersamaan dan terkoordinasi (Docker, 2026a).
5. Docker Network adalah mekanisme yang memungkinkan *container-container* yang berjalan dalam satu lingkungan Docker untuk saling berkomunikasi menggunakan nama *service* sebagai *hostname*, tanpa perlu mengetahui alamat IP secara eksplisit. Docker secara otomatis membuat jaringan virtual (*bridge network*) untuk setiap aplikasi yang didefinisikan melalui Docker Compose,

sehingga *container frontend, backend*, dan layanan lain dapat saling mengakses secara internal dengan aman (Docker, 2026b).

6. Docker Volume adalah mekanisme penyimpanan persisten yang memungkinkan data tetap ada meskipun *container* dihentikan atau dihapus. Berbeda dengan sistem *file container* yang bersifat sementara, volume disimpan di sistem *file host* dan dikelola oleh Docker, sehingga data tidak hilang ketika *container* diganti atau diperbarui ke versi baru (Docker, 2026a).

Dalam konteks pengembangan AISYA-RA, Docker digunakan dalam dua fase yang saling terkait. Pada fase pengembangan lokal di lingkungan WSL, Docker digunakan untuk menjalankan layanan pendukung seperti web server lokal tanpa instalasi manual. Pada fase *Deployment* ke DigitalOcean *Droplet*, Docker Compose mengorkestrasi tiga *container* utama, *frontend* React.js, *backend* FastAPI, dan Nginx sebagai *reverse proxy*, sehingga seluruh sistem dapat dijalankan, dihentikan, dan diperbarui melalui satu *file* konfigurasi terpusat (Ariadi et al., 2020).

3.14.4 Draw.io

Draw.io (juga dikenal sebagai diagrams.net) adalah aplikasi pembuatan diagram berbasis web dan desktop yang dikembangkan oleh JGraph Ltd dan dapat digunakan secara gratis tanpa memerlukan pendaftaran akun maupun lisensi berbayar. draw.io adalah sebuah *Website* yang didesain khusus untuk menggambarkan diagram UML secara *online*, dengan tampilan yang sangat responsif dan terintegrasi dengan layanan penyimpanan Google Drive, sehingga draw.io menjadi alternatif pembuatan diagram UML dengan waktu yang lebih singkat (Marthiawati et al., 2024). Aplikasi ini mendukung pembuatan berbagai jenis diagram yang dibutuhkan dalam perancangan sistem informasi, termasuk *Use Case Diagram, Activity Diagram, Class Diagram, Sequence Diagram, Entity Relationship Diagram (ERD), flowchart*, diagram arsitektur jaringan, dan *mockup* antarmuka (draw.io, 2026).

Salah satu keunggulan utama draw.io adalah fleksibilitas format penyimpanan dan ekspor yang dimilikinya. Diagram yang dibuat disimpan dalam format XML yang ringan untuk pengeditan lanjutan, serta dapat diekspor ke berbagai format keluaran termasuk *vector*

(SVG), gambar raster (JPEG, PNG), dokumen PDF, maupun HTML tersemat. Selain itu, draw.io mendukung integrasi penyimpanan dengan berbagai *platform cloud* seperti Google Drive, OneDrive, Dropbox, dan GitHub, sehingga diagram dapat diakses, disunting, dan dibagikan secara kolaboratif dari perangkat mana pun tanpa memerlukan instalasi khusus (draw.io, 2026).

Dalam konteks pengembangan sistem informasi, draw.io digunakan untuk memvisualisasikan konsep-konsep seperti diagram alur kerja, struktur data, dan antarmuka pengguna, sehingga siswa maupun pengembang dapat memperoleh pemahaman yang lebih baik tentang proses perancangan perangkat lunak (Oktafiandi et al., 2024). Penggunaan draw.io dalam pelatihan pembuatan diagram UML terbukti mampu meningkatkan pemahaman peserta terhadap konsep pemodelan perangkat lunak, sekaligus menunjukkan bahwa alat ini mudah dipelajari oleh pengguna dari berbagai latar belakang (Marthiawati et al., 2024).

Dalam pengembangan AISYA-RA, draw.io digunakan untuk menyusun seluruh artefak perancangan UML yang diperlukan pada setiap iterasi *Agile*, meliputi *Use Case Diagram, Activity Diagram, Sequence Diagram, Class Diagram*, dan ERD. Pemilihan draw.io didasarkan pada pertimbangan bahwa draw.io tidak memerlukan lisensi berbayar, dapat dijalankan langsung di *browser* tanpa instalasi tambahan, mendukung ekspor ke format gambar dan PDF untuk keperluan dokumentasi, serta menghasilkan *file* XML yang ringan dan mudah dikelola bersama kode proyek melalui sistem kontrol versi (Oktafiandi et al., 2024).

3.15 Infrastruktur *Deployment* Sistem

Infrastruktur *Deployment* adalah sekumpulan komponen teknis yang diperlukan agar perangkat lunak yang telah dikembangkan dapat dijalankan secara daring dan diakses oleh pengguna nyata. Dalam pengembangan sistem berbasis web, infrastruktur *Deployment* mencakup server *cloud*, *platform containerization*, web server, dan konfigurasi Domain yang bekerja secara terpadu untuk membuat sistem dapat diakses, stabil, dan aman (Raseuki et al., 2024).

3.15.1 *Cloud Computing* dan *Virtual Private Server (VPS)*

Cloud computing adalah model komputasi yang memungkinkan akses jaringan sesuai permintaan ke sumber daya komputasi bersama, seperti server, penyimpanan, jaringan, dan layanan, yang dapat disediakan dan dilepaskan dengan cepat tanpa memerlukan interaksi langsung dengan penyedia layanan (Raseuki et al., 2024). Dalam penerapannya, *cloud computing* dibagi menjadi tiga model layanan utama, *Infrastructure as a Service (IaaS)* yang menyediakan infrastruktur virtual seperti server dan jaringan, *Platform as a Service (PaaS)* yang menyediakan platform pengembangan siap pakai serta *Software as a Service (SaaS)* yang menyediakan aplikasi berbasis web yang langsung dapat digunakan pengguna.

Virtual Private Server (VPS) merupakan salah satu implementasi model IaaS yang memberikan pengembang akses ke sumber daya server virtual yang terisolasi, termasuk CPU, RAM, penyimpanan, dan koneksi jaringan, dengan hak penuh untuk menginstal sistem operasi, perangkat lunak, dan mengonfigurasi lingkungan sesuai kebutuhan (Rahmah & Elyas, 2024). Berbeda dengan *shared hosting* yang membagi sumber daya dengan banyak pengguna lain, VPS memberikan lingkungan yang lebih terkontrol dan terisolasi sehingga lebih sesuai untuk aplikasi web yang membutuhkan konfigurasi khusus.

Dalam pengembangan AISYA-RA, VPS digunakan melalui layanan DigitalOcean *Droplet* yang merupakan nama komersial untuk VPS pada platform DigitalOcean. Server ini berfungsi sebagai lingkungan komputasi utama tempat seluruh komponen aplikasi dijalankan, termasuk layanan *frontend, backend, container* Docker, dan *reverse proxy*. Penggunaan VPS berbasis *cloud* memungkinkan sistem dapat diakses kapan saja melalui internet tanpa bergantung pada ketersediaan perangkat keras lokal (Raseuki et al., 2024).

3.15.2 Web Server dan Reverse Proxy

Web Server adalah perangkat lunak yang berfungsi menerima permintaan (*request*) dari klien melalui protokol HTTP/HTTPS, memprosesnya, dan mengirimkan kembali respons yang sesuai. Web server merupakan komponen inti dari infrastruktur sistem berbasis web karena menjadi titik masuk utama bagi seluruh lalu lintas data antara pengguna dan

aplikasi (Saputra et al., 2023). Di antara web server yang paling umum digunakan saat ini adalah Nginx dan Apache, di mana Nginx dikenal memiliki performa lebih baik dalam menangani banyak permintaan secara bersamaan dengan konsumsi memori yang lebih rendah.

Reverse Proxy adalah server yang berfungsi sebagai perantara antara klien dan satu atau beberapa server *backend*. Ketika klien mengirimkan permintaan, reverse proxy menerimanya terlebih dahulu, kemudian meneruskannya ke server *backend* yang sesuai, dan mengembalikan respons ke klien atas nama server *backend* tersebut (Bukhari & Iqbal, 2024). Dengan arsitektur ini, klien tidak berinteraksi langsung dengan server aplikasi, sehingga memberikan lapisan keamanan tambahan sekaligus memungkinkan pengelolaan lalu lintas yang lebih fleksibel.

Penggunaan *reverse proxy* memberikan beberapa manfaat teknis penting:

1. penyembunyian infrastruktur *backend* dari akses langsung sehingga meningkatkan keamanan server (Azi et al., 2023).
2. kemampuan *load balancing* untuk mendistribusikan permintaan ke beberapa server (Azi et al., 2023).
3. dukungan SSL/HTTPS yang terpusat pada satu titik (Azi et al., 2023).
4. kemampuan mengelola beberapa layanan dari satu alamat IP publik (Azi et al., 2023).

Dalam konteks AISYA-RA yang berjalan pada satu VPS DigitalOcean *Droplet*, Nginx dikonfigurasi sebagai *reverse proxy* yang meneruskan permintaan dari pengguna ke layanan *frontend* dan *backend* yang berjalan di dalam *container* Docker pada *port* yang berbeda.

3.15.3 Domain dan Subdomain

Domain adalah nama unik yang digunakan untuk mengidentifikasi sebuah situs web atau layanan di internet, menggantikan alamat IP numerik yang sulit diingat oleh pengguna (Fendy Nugraha et al., 2025). Domain terdiri dari hierarki nama yang dipisahkan oleh titik, di mana .id adalah *Top-Level Domain (TLD)* untuk Indonesia, .sch menunjukkan

kategori sekolah, dan nama Domain merupakan identitas yang didaftarkan oleh lembaga. Sistem yang menerjemahkan nama Domain ke alamat IP disebut *Domain Name System (DNS)*, yang bekerja secara otomatis di balik layar setiap kali pengguna mengetikkan alamat web di *browser* (Fahmi et al., 2023).

Subdomain adalah bagian dari Domain yang dibuat di bawah Domain utama dengan menambahkan awalan sebelum nama Domain, dipisahkan oleh titik. Sebagai contoh, aisya.ra-alislam.sch.id merupakan subdomain dengan nama aisya yang dibuat di bawah Domain ra-alislam.sch.id. Subdomain dapat diarahkan secara independen ke server atau alamat IP yang berbeda dari Domain utamanya, sehingga memungkinkan sebuah lembaga memiliki beberapa layanan digital dalam satu nama Domain tanpa harus mendaftarkan Domain baru (Fendy Nugraha et al., 2025).

Dalam pengembangan AISYA-RA, penggunaan subdomain aisya.ra-alislam.sch.id bertujuan untuk memberikan alamat akses yang mudah dikenali oleh guru dan Kepala RA sebagai bagian dari ekosistem digital lembaga. Subdomain ini dikonfigurasi melalui pengaturan DNS untuk mengarahkan permintaan ke alamat IP server DigitalOcean *Droplet*. Di sisi server, Nginx sebagai *reverse proxy* menerima permintaan yang masuk melalui subdomain tersebut dan meneruskannya ke layanan aplikasi yang sesuai(Habib et al., 2023).

3.15.4 Arsitektur Sistem Web Berlapis

Arsitektur sistem web berlapis (*layered web architecture*) adalah pendekatan desain sistem yang memisahkan fungsi-fungsi aplikasi ke dalam lapisan-lapisan yang memiliki tanggung jawab yang berbeda dan saling berkomunikasi melalui antarmuka yang terdefinisi. Pemisahan lapisan ini meningkatkan modularitas sistem, memudahkan pemeliharaan, dan memungkinkan setiap lapisan dikembangkan atau diperbarui secara independen tanpa mengganggu lapisan lainnya (Raseuki et al., 2024).

Dalam arsitektur web modern, sistem umumnya diorganisasi ke dalam tiga lapisan utama:

1. lapisan presentasi (*frontend*), yang bertanggung jawab menampilkan antarmuka kepada pengguna.

2. lapisan logika bisnis (*backend*), yang memproses permintaan, menjalankan aturan bisnis, dan mengorkestrasi komunikasi antar layanan.
3. lapisan data (*Database*), yang menyimpan dan mengelola data secara persisten.

Pada sistem yang mengintegrasikan layanan AI eksternal, ditambahkan lapisan layanan AI sebagai komponen tambahan yang dapat dipanggil dari *backend* sesuai kebutuhan.

Pada AISYA-RA, arsitektur berlapis diwujudkan melalui: *frontend* React.js sebagai lapisan presentasi yang berjalan di *browser* pengguna; *backend* FastAPI sebagai lapisan logika bisnis yang mengelola permintaan, autentikasi, dan integrasi layanan; PostgreSQL melalui Supabase sebagai lapisan data; serta Gemini API sebagai lapisan layanan AI eksternal. Seluruh lapisan diikat oleh lapisan infrastruktur yang terdiri dari DigitalOcean *Droplet*, Docker (*containerization*), dan Nginx (*reverse proxy*). Dengan arsitektur ini, setiap lapisan dapat diuji, diperbaiki, dan diskalakan secara independen sesuai kebutuhan operasional sistem.

3.16 Black Box Testing

Black Box Testing adalah metode pengujian perangkat lunak yang berfokus pada pengujian fungsi sistem berdasarkan *input* dan *output* tanpa memeriksa struktur internal kode program (Novianti & Voutama, 2024)

Dalam pendekatan ini, yang diuji adalah apakah sistem memberikan hasil yang sesuai terhadap masukan tertentu. Karena orientasinya pada perilaku eksternal sistem, *Black Box Testing* sangat cocok digunakan pada aplikasi berbasis web yang berinteraksi langsung dengan pengguna (Rudi Sanjaya et al., 2022).

Pada sistem administrasi seperti AISYA-RA, *Black Box Testing* dapat digunakan untuk memeriksa apakah *login* berjalan dengan benar, apakah data presensi tersimpan sesuai *input*, apakah surat berhasil dihasilkan, dan apakah *respons Chatbot* sesuai dengan konteks permintaan pengguna (Novianti & Voutama, 2024). Dalam penelitian ini, *Black Box Testing* menjadi metode pengujian yang tepat karena tujuan utama pengujian adalah memastikan seluruh fitur AISYA-RA dapat digunakan secara benar oleh guru dan kepala RA Al-Islam.

3.17 *System Usability Scale (SUS)*

System Usability Scale (SUS) adalah instrumen evaluasi *usability* yang terdiri atas 10 pernyataan dengan skala *Likert* untuk mengukur persepsi pengguna terhadap kemudahan penggunaan suatu sistem. SUS banyak digunakan karena sederhana, cepat diterapkan, dan mampu memberikan gambaran umum mengenai kualitas pengalaman pengguna secara kuantitatif (Kusuma & Giri, 2024).

Pada metode SUS, item bernomor ganjil merepresentasikan pernyataan positif, sedangkan item bernomor genap merepresentasikan pernyataan negatif. Skor akhir dikonversi ke rentang 0 sampai 100 dan kemudian diinterpretasikan untuk menilai apakah suatu sistem termasuk tidak dapat diterima, marginal, atau dapat diterima dari sudut pandang pengguna .

Dalam penelitian AISYA-RA, SUS relevan digunakan untuk mengukur tingkat penerimaan guru dan kepala RA terhadap sistem setelah digunakan. Melalui SUS, peneliti dapat menilai apakah antarmuka *Chatbot*, modul administrasi, dan alur interaksi yang dikembangkan benar-benar mudah dipelajari, nyaman digunakan, dan sesuai dengan kemampuan digital pengguna di lingkungan RA Al-Islam (Rosalina et al., 2024).

3.17.1 Konsep *Usability*

Usability atau kegunaan merupakan salah satu aspek krusial dalam pengembangan sistem informasi. Menurut ISO 9241-11, *usability* didefinisikan sebagai sejauh mana suatu sistem dapat digunakan oleh pengguna tertentu untuk mencapai tujuan tertentu secara efektif, efisien, dan memuaskan dalam suatu konteks penggunaan (Deluma et al., 2023). Definisi ini menekankan tiga dimensi utama, efektivitas (ketepatan dan kelengkapan pencapaian tujuan), efisiensi (sumber daya yang digunakan dalam mencapai tujuan), dan kepuasan (sikap positif terhadap penggunaan sistem) (Deluma et al., 2023).

Usability mencakup beberapa komponen evaluasi, yaitu *learnability* (kemudahan belajar), *efficiency* (efisiensi penggunaan setelah mempelajari sistem), *memorability* (kemampuan mengingat kembali penggunaan setelah periode tidak menggunakan), *error rate* (tingkat kesalahan), dan *satisfaction* (kepuasan pengguna) (Dyayu et al., 2023). Dalam konteks AISYA-RA, evaluasi *usability* menjadi sangat penting karena pengguna

utama sistem memiliki tingkat literasi digital yang beragam, sehingga kemudahan penggunaan menjadi faktor penentu keberhasilan adopsi sistem.

3.17.2 Sejarah dan Konsep *System Usability Scale (SUS)*

System Usability Scale (SUS) adalah instrumen evaluasi usability yang pertama kali diperkenalkan oleh John Brooke pada tahun 1996 sebagai alat evaluasi yang cepat dan reliabel untuk menilai tingkat kegunaan suatu sistem dari sudut pandang pengguna (Kusuma & Giri, 2024). SUS dirancang sebagai metode "*quick and dirty*" yang memungkinkan penilaian *usability* secara global dengan biaya rendah dan waktu yang singkat. Meskipun istilah "*quick and dirty*" digunakan, SUS telah terbukti memiliki tingkat validitas dan reliabilitas yang tinggi dalam berbagai konteks penelitian (Huda et al., 2023).

SUS terdiri atas 10 butir pernyataan dengan skala *Likert* 1 hingga 5, di mana nilai 1 mewakili "Sangat Tidak Setuju" dan nilai 5 mewakili "Sangat Setuju" (Kusuma & Giri, 2024). Konstruksi kuesioner dirancang secara selang-seling, yaitu terdiri dari 5 pernyataan bernada positif (item bernomor ganjil) dan 5 pernyataan bernada negatif (item bernomor genap). Pola selang-seling ini bertujuan untuk mengurangi bias respons akquiescence, yaitu kecenderungan responden untuk menyetujui semua pernyataan tanpa membaca secara seksama (Dyayu et al., 2023).

Setiap item dalam kuesioner SUS merepresentasikan indikator usability yang berbeda, meliputi *usefulness* (kegunaan), *complexity* (kompleksitas), *ease of use* (kemudahan penggunaan), *need for support* (kebutuhan dukungan teknis), *consistency* (konsistensi), *learnability* (kemudahan belajar), dan *confidence* (kepercayaan diri) (Dyayu et al., 2023). Kuesioner SUS bersifat agnostik terhadap teknologi, artinya dapat digunakan untuk mengevaluasi berbagai jenis produk seperti perangkat lunak, aplikasi web, aplikasi mobile, hingga perangkat keras (Kusuma & Giri, 2024).

3.17.3 Metode Pengukuran Skor SUS

Penilaian akhir kuesioner SUS tidak dilakukan dengan sekadar menjumlahkan nilai mentah dari jawaban responden, melainkan melalui tahapan perhitungan matematis baku (Kusuma & Giri, 2024). Aturan perhitungannya adalah sebagai berikut:

1. Untuk pernyataan bernada positif (item ganjil 1, 3, 5, 7, dan 9), skor kontribusi adalah nilai skala jawaban dikurangi 1, atau dirumuskan sebagai X − 1.
2. Untuk pernyataan bernada negatif (item genap 2, 4, 6, 8, dan 10), skor kontribusi adalah 5 dikurangi nilai skala jawaban, atau dirumuskan sebagai 5 − X.

Nilai keseluruhan (*Total Score*) untuk masing-masing responden didapatkan dengan mengakumulasikan seluruh skor kontribusi, yang kemudian dikalikan dengan konstanta 2,5 untuk mentransformasi skor ke dalam rentang 0 hingga 100 (Kusuma & Giri, 2024). Secara matematis, rumusan tersebut dituliskan sebagai berikut:

$$\text{Skor Akhir SUS} = \left(\sum\left(X_{ganjil} - 1\right) + \sum\left(5 - X_{genap}\right)\right) \times 2{,}5$$

Perkalian dengan konstanta 2,5 mengonversi skor mentah yang berkisar antara 0 hingga 40 menjadi skor akhir dalam rentang 0 hingga 100, sehingga memudahkan interpretasi dan perbandingan antar sistem yang berbeda (Kusuma & Giri, 2024).

3.17.4 Interpreteasi Skor SUS

Hasil skor komprehensif dalam rentang 0–100 diinterpretasikan melalui tiga pendekatan utama*, Acceptability Ranges, Grade Scale*, dan *Adjective Rating* (Kusuma & Giri, 2024)

Acceptability Ranges digunakan untuk menilai apakah sistem yang diuji diterima oleh pengguna atau tidak. Skor di bawah 50 dikategorikan *Not Acceptable* (Tidak Dapat Diterima), skor 50 hingga 70 dikategorikan *Marginal* (Batas Ambang), skor di atas 70 dikategorikan *Acceptable* (Dapat Diterima), dan skor di atas 80 diklasifikasikan ke dalam predikat penerimaan tertinggi (Kusuma & Giri, 2024).

Selain *Acceptability Ranges*, skor SUS juga dapat dipetakan ke dalam *Grade Scale* yang mirip dengan sistem penilaian akademis (A hingga F) dan *Adjective Rating* yang memberikan Deskripsi kualitatif seperti *Worst Imaginable, Poor, OK, Good, Excellent,* hingga *Best Imaginable* (Kusuma & Giri, 2024). Tabel interpretasi skor SUS disajikan pada Tabel 3.8.

Tabel 3.8 Interpretasi Skor SUS (Acceptability Ranges, Grade Scale, dan Adjective Rating)

Skor SUS	***Acceptability Range***	***Grade Scale***	***Adjective Rating***
85 – 100	*Acceptable*	A+	*Best Imaginable / Excellent*
80 – 84	*Acceptable*	A	*Excellent*
75 – 79	*Acceptable*	B+	*Good*
70 – 74	*Acceptable*	B	*Good*
65 – 69	*Marginal*	C+	*OK*
60 – 64	*Marginal*	C	*OK*
55 – 59	*Marginal*	D	*Poor*
50 – 54	*Marginal*	D	*Poor*
0 – 49	*Not Acceptable*	F	*Worst Imaginable*

(Sumber:Kusuma & Giri, 2024)

Selain itu, terdapat patokan umum yang menyatakan bahwa skor rata-rata global SUS adalah 68, yang dianggap sebagai batas minimal untuk sistem yang dapat diterima pengguna. Skor di atas 68 umumnya dianggap baik, sedangkan skor di bawah 68 menunjukkan bahwa sistem perlu perbaikan dalam hal kegunaan (Kusuma & Giri, 2024).

3.17.5 Keunggulan SUS

SUS banyak digunakan dalam penelitian usability karena memiliki beberapa keunggulan (Kusuma & Giri, 2024; Huda et al., 2023):

1. Sederhana dan cepat. Kuesioner hanya terdiri dari 10 item, sehingga dapat diselesaikan oleh responden dalam waktu 3–5 menit.

2. Valid dan reliabel. SUS telah terbukti sebagai alat uji usability yang valid dan reliabel walaupun dengan sampel yang sedikit, menjadikannya cocok untuk penelitian dengan jumlah responden terbatas.
3. Agnostik terhadap teknologi. Dapat digunakan untuk mengevaluasi berbagai jenis sistem, baik perangkat lunak, aplikasi web, aplikasi mobile, maupun perangkat keras.
4. Gratis. SUS tersedia secara gratis untuk penggunaan non-komersial, tanpa memerlukan lisensi khusus.
5. Hasil kuantitatif yang mudah diinterpretasikan. Skor akhir dalam rentang 0–100 memudahkan perbandingan antar sistem dan antar penelitian.

3.18 Pengukuran Efisiensi Waktu Administrasi

3.18.1 Konsep Efisiensi Berbagi Tugas

Pengukuran efisiensi sistem informasi dapat dilakukan melalui pendekatan task *completion time*, yaitu pengukuran waktu yang diperlukan pengguna untuk menyelesaikan tugas tertentu menggunakan sistem dibandingkan dengan metode manual (Muflihah & Yoses, 2024). Standar ISO/IEC 25022 mengatur indikator utama untuk mengukur efisiensi, meliputi *task time* (waktu penyelesaian tugas), *time efficiency* (efisiensi waktu), dan *productive time ratio* (rasio waktu produktif) (Muflihah & Yoses, 2024)

Pengukuran task completion time sering digunakan sebagai indikator efisiensi dalam evaluasi usability karena berkaitan langsung dengan beban kerja pengguna. Sistem yang memungkinkan tugas diselesaikan dalam waktu yang lebih singkat dinilai lebih efisien dan mampu mengurangi beban kerja administratif (Muflihah & Yoses, 2024).

3.18.2 Rumus Penghitungan Persentase Penghematan Waktu

Persentase penghematan waktu dihitung dengan membandingkan waktu penyelesaian tugas menggunakan metode manual dengan waktu penyelesaian tugas menggunakan sistem. Rumus yang digunakan adalah (Muflihah & Yoses, 2024):

$$\text{Penghematan (\%)} = \frac{T_{\text{manual}} - T_{\text{sistem}}}{T_{\text{manual}}} \times 100\%$$

Di mana:

a. T_{manual} = waktu penyelesaian tugas dengan metode manual (sebelum implementasi sistem)

b. T_{sistem} = waktu penyelesaian tugas menggunakan sistem AISYA-RA