BAB II
TINJAUAN PUSTAKA

Dalam bagian ini, peneliti akan menjelaskan aspek-aspek terkait dengan pengkajian kembali terhadap literatur yang terkait dengan topik penelitian. Hal ini dilakukan untuk mendapatkan bahan perbandingan dan referensi terhadap beberapa teori yang relevan dalam kerangka tugas akhir yang berjudul "Rancang Bangun Aplikasi *Monitoring Resource* Server dan *Microservice* Berbasis *Web* dengan Notifikasi Telegram di PT XYZ Menggunakan Pendekatan Waterfall". Aplikasi yang dimaksud mencakup aplikasi dashboard berbasis web dan aplikasi latar belakang yang beroperasi melalui *scheduler*. Oleh karena itu dibawah ini merupakan teori teori dasar yang mendukung pengembangan aplikasi dan perancangan aplikasi.

2.1 Teori Dasar

2.1.1 Rancang Bangun

Woro Isti Rahayu, Ravi Rahmatul Fajri, Parhan Hambali (2019, p 21) mengungkapkan bahwa Rancang bangun adalah produk penelitian yang membantu peneliti menyelesaikan masalah pada objek penelitian. Istilah ini terdiri dari dua kata yaitu rancang dan bangun.

1) Rancang
   Didalam rancang, hasil analisis sistem diterjemahkan ke dalam bahasa pemrograman yang tepat. Hal ini bertujuan untuk menjelaskan implementasi komponen-komponen sistem secara detail dan terstruktur. Tahap rancang ibarat merancang cetak biru sebuah bangunan, di mana setiap elemen dan fungsinya dijabarkan dengan jelas.

2) Bangun
   Setelah rancangan selesai, tahap selanjutnya adalah bangun, yaitu proses mewujudkan sistem tersebut. Tahap ini melibatkan implementasi kode program, pengujian sistem, dan penyempurnaan hingga sistem siap digunakan.

Rancang bangun dalam penelitian ini merupakan produk yang dihasilkan dari penerjemahan data penelitian, seperti observasi, wawancara, studi pustaka, dan dokumentasi. Produk tersebut membantu peneliti dalam membuat perancangan dengan lebih mudah dan efisien. Berikut merupakan beberapa manfaat dari rancang bangun didalam proses penelitian.

1) Mempercepat proses penelitian, Dengan merancang dan membangun sistem secara terstruktur, peneliti dapat menghemat waktu dan tenaga dalam menyelesaikan penelitiannya.

2) Meningkatkan ketepatan dan validitas hasil penelitian, Sistem yang dirancang dan dibangun dengan baik akan menghasilkan hasil penelitian yang lebih akurat dan valid.
3) Memudahkan replikasi penelitian, Rancangan dan bangun yang terdokumentasi dengan baik memungkinkan peneliti lain untuk mereplikasi penelitian dengan mudah.
4) Meningkatkan kolaborasi penelitian, Produk rancang bangun dapat dibagikan dengan peneliti lain, sehingga memungkinkan kolaborasi penelitian yang lebih efektif.

2.1.2 Sistem Informasi

Elisabeth Yunaeti Anggraeni dan Rita Irviani (2017, p. 5) mengungkapkan bahwa Sistem informasi adalah suatu sistem yang menyediakan informasi bagi manajemen untuk pengambilan keputusan serta pelaksanaan operasional perusahaan. Sistem ini merupakan kombinasi dari manusia, teknologi informasi, dan prosedur-prosedur yang terorganisir. Sistem informasi sendiri dirancang untuk mengumpulkan, memproses, menyimpan, dan mendistribusikan informasi. Sistem ini dapat membantu perusahaan untuk meningkatkan efisiensi, efektivitas, dan pengambilan keputusan. Berikut merupakan komponen-komponen dari sistem informasi.

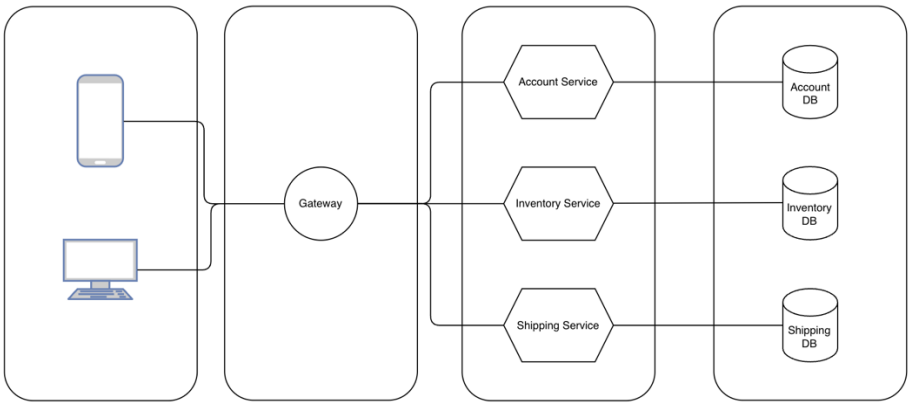
1) Perangkat Keras (*Hardware*) yaitu berupa komputer, server, jaringan, dan perangkat penyimpanan.
2) Perangkat Lunak (*Software*) yaitu berupa sistem operasi, aplikasi, dan database.
3) Orang (*People*) yaitu berupa pengguna, administrator sistem, dan pengembang.
4) Data yaitu berupa fakta, angka, dan teks yang diolah oleh sistem informasi.
5) Prosedur yaitu berupa aturan dan instruksi untuk mengoperasikan sistem informasi.
6) Jaringan Komunikasi (*Communication Networks*) yaitu berupa internet, intranet, dan ekstranet.
7) Keamanan (*Security*) yaitu berupa keamanan fisik, keamanan data, dan keamanan jaringan.
8) Infrastruktur (*Infrastructure*) yaitu berupa pusat data dan peralatan jaringan seperti router, switch, dan firewall.
9) Kebijakan dan Regulasi (*Policies and Regulations*) yaitu berupa kebijakan internal dan regulasi eksternal terkait penggunaan dan perlindungan data.
10) Manajemen Proyek (*Project Management*) yaitu berupa pengelolaan proyek dan metodologi pengembangan seperti Agile, Waterfall, atau DevOps.

Salah satu jenis dari sistem informasi merupakan sebagai sistem pendukung keputusan (SPK). Sistem Pendukung Keputusan (SPK) adalah sebuah sistem informasi yang diciptakan khusus untuk membantu dalam proses pengambilan keputusan dalam situasi yang kompleks dan semi-terstruktur. Dengan menyediakan berbagai alat dan teknik analisis data, serta membangun model, SPK bertujuan untuk menghasilkan solusi yang optimal bagi penggunanya.

Komponen utama dari SPK meliputi basis data yang menyimpan informasi terkait masalah yang akan dipecahkan, model matematis yang merepresentasikan masalah, alat analisis untuk menganalisis data dan membangun model, serta antarmuka pengguna yang memfasilitasi interaksi dengan sistem.

Manfaat yang diperoleh dari penggunaan SPK sangatlah beragam. Pertama-tama, SPK membantu meningkatkan kualitas pengambilan keputusan dengan memungkinkan pengambil keputusan untuk mempertimbangkan semua faktor yang relevan dan memilih solusi yang paling optimal. Selain itu, SPK juga dapat meningkatkan efisiensi pengambilan keputusan dengan mengotomatisasi beberapa tugas yang terkait, sehingga menghemat waktu dan sumber daya. Yang tak kalah pentingnya, SPK juga dapat meningkatkan komunikasi dan kolaborasi antara para pengambil keputusan, memungkinkan mereka untuk berinteraksi dengan lebih mudah dan efisien dalam mencapai tujuan bersama. Dengan demikian, SPK tidak hanya memberikan dukungan teknis dalam pengambilan keputusan, tetapi juga memfasilitasi proses komunikasi dan kerja sama yang lebih baik di antara para pemangku kepentingan.

2.1.3 *Microservice*



Gambar 2. 1 Contoh Arsitektur *Microservice*

Jose Haro Peralta (2023, p. 1) mengungkapkan bahwa *Microservice* merupakan pendekatan arsitektural di mana komponen-komponen dari sebuah sistem dirancang sebagai aplikasi yang mandiri dan dapat dideploy secara

independen. Setiap aplikasi yang berfungsi secara independen berkomunikasi melalui *application programming interface (API).*

Karakteristik utama dari *Microservices* adalah kebebasannya, di mana setiap layanan dapat dikembangkan, diuji, dan dideploy secara terpisah. Layanan ini juga haruslah kecil dan fokus pada satu fungsi spesifik. Komunikasi antar layanan dilakukan melalui *API*, tanpa adanya titik kontrol terpusat, yang memungkinkan sistem untuk tahan terhadap kegagalan.

Manfaat utama dari Microservices termasuk kecepatan dalam pengembangan dan peluncuran aplikasi (*agility*), kemampuan untuk mengatasi peningkatan *traffic* dengan penskalaan horizontal (*scalability*), keandalan dalam menghadapi kegagalan (*reliability*), serta kemudahan pemeliharaan yang lebih baik dibandingkan dengan aplikasi monolitik.

Namun, Microservices juga membawa tantangan tersendiri. Kompleksitas dapat meningkat karena banyaknya layanan yang perlu dikelola, sementara keamanan juga menjadi perhatian karena adanya risiko baru yang muncul. Selain itu, monitoring perlu dilakukan dengan cermat untuk memastikan performa sistem tetap optimal.

2.1.4 Server

Didik Setiawan (2017, p. 9) mengungkapkan bahwa Server adalah sebuah sistem komputer yang menyediakan berbagai jenis layanan untuk klien dalam sebuah jaringan komputer. Server ini dilengkapi dengan sistem operasi khusus yang mengatur akses dan sumber daya di dalamnya, yang umumnya disebut sebagai sistem operasi jaringan atau *Network Operating System*. Server sendiri berfungsi untuk menerima permintaan (*request*) dari klien, memprosesnya, dan memberikan respon atau hasil kembali kepada klien. Beberapa jenis server meliputi:

1) Server *Web*
   Menyediakan halaman *web* dan konten internet. Contoh: Apache, Nginx.
2) Server *Database*
   Menyimpan dan mengelola data. Contoh: MySQL, PostgreSQL, MongoDB.
3) Server File
   Menyimpan dan mengelola file yang dapat diakses oleh klien. Contoh: FTP server, file server.
4) Server Aplikasi
   Menjalankan aplikasi tertentu dan menyediakan layanan aplikasi. Contoh: Java EE server, Microsoft IIS.
5) Server Email
   Menyediakan layanan email. Contoh: Microsoft Exchange Server, Postfix.

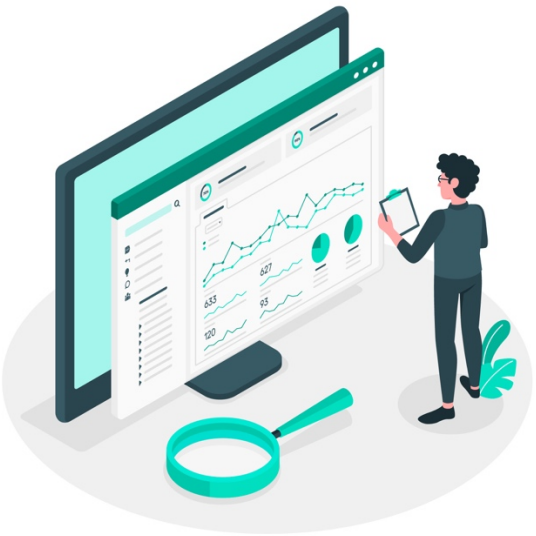
Server dapat berupa perangkat keras (*hardware*) yang berupa komputer fisik atau perangkat lunak (*software*) yang berjalan di atas suatu sistem komputasi.

Secara umum, server dirancang untuk memberikan layanan yang dapat diakses oleh banyak klien secara bersamaan. Keandalan, kecepatan, dan ketersediaan tinggi sering menjadi fokus dalam desain dan pengelolaan server. Didalam Server sendiri tedapat komponen-komponen atau sumberdaya yang dapat digunakan oleh server untuk menjalankan aplikasi atau menyediakan layanan. Beberapa sumber daya utama dalam server melibatkan:

1) *CPU (Central Processing Unit)*
   CPU adalah otak dari server yang menangani pemrosesan instruksi dan eksekusi tugas-tugas komputasi. Kapasitas dan kecepatan *CPU* mempengaruhi kemampuan server dalam menangani beban kerja.
2) *RAM (Random Access Memory)*
   RAM menyimpan data sementara yang sedang digunakan oleh server. Semakin besar kapasitas RAM, semakin banyak data dan aplikasi yang dapat diakomodasi oleh server.
3) *Penyimpanan (Storage)*
   Sumber daya penyimpanan termasuk *hard drive* atau *solid-state drive (SSD)* yang digunakan untuk menyimpan sistem operasi, aplikasi, dan data. Kapasitas dan jenis penyimpanan dapat memengaruhi ketersediaan dan kinerja server.
4) Jaringan (*Network*)
   Sumber daya jaringan mencakup ketersediaan *bandwidth* dan konektivitas. Ketersediaan jaringan yang baik penting untuk menyediakan layanan kepada pengguna dan memastikan konektivitas dengan sumber daya eksternal.
5) *Bandwidth*
   Bandwidth adalah kapasitas maksimum untuk mentransfer data dalam suatu jaringan. Sumber daya ini penting untuk menangani jumlah permintaan dan lalu lintas data yang diterima atau dikirim oleh server.
6) Prosesor Grafis (GPU)
   GPU dapat digunakan untuk akselerasi komputasi tertentu, terutama dalam konteks komputasi paralel, rendering grafis, atau aplikasi kecerdasan buatan.

Pemantauan dan manajemen sumber daya sangat penting dalam menjaga ketersediaan, performa, dan keamanan server. Pemilihan sumber daya yang sesuai dengan kebutuhan aplikasi dan beban kerja tertentu juga menjadi pertimbangan penting dalam merancang dan mengelola server.

2.1.5 *Monitoring*



Sumber : https://www.freepik.com/

Gambar 2. 2 Contoh *Monitoring*

James Turnbull (2014, p. 9) mengungkapkan bahwa *monitoring* merupakan alat dan proses yang digunakan untuk mengukur dan mengelola suatu sistem. Tujuan utama dari *monitoring* adalah untuk memastikan kinerja yang optimal, keandalan, keamanan, dan ketersediaan suatu sistem atau proses. Beberapa aspek umum dari *monitoring* melibatkan:

1) Pengumpulan Data
   Monitoring melibatkan pengumpulan data terkait dengan kinerja, status, atau perilaku suatu sistem. Data ini bisa melibatkan berbagai metrik, log, atau informasi lainnya yang diperlukan untuk evaluasi.
2) Analisis Data
   Data yang dikumpulkan dianalisis untuk mengidentifikasi tren, pola, atau anomali yang dapat menunjukkan masalah atau perubahan kondisi.
3) Notifikasi dan Peringatan
   Berdasarkan hasil analisis, sistem *monitoring* dapat menghasilkan notifikasi atau peringatan kepada administrator atau pengguna ketika terjadi situasi yang memerlukan perhatian atau tindakan.
4) Pelaporan
   Monitoring juga seringkali melibatkan pembuatan laporan yang merinci kinerja, keandalan, atau kejadian tertentu dalam suatu periode waktu. Laporan ini dapat digunakan untuk evaluasi jangka panjang atau pemantauan tren.
5) Pengambilan Keputusan

Informasi yang diberikan oleh *monitoring* dapat digunakan sebagai dasar untuk pengambilan keputusan terkait perbaikan, peningkatan kinerja, atau tindakan lainnya untuk menjaga atau meningkatkan kondisi suatu sistem.

2.1.6 *Dashboard*

Sumber : https://www.freepik.com/

Gambar 2. 3 Contoh *Dashboard*

Dover (2004, p. 44) mengungkapkan bahwa *dashboard* adalah alat yang menyajikan ringkasan sekilas dalam format visual dan intuitif yang sangat mudah dipahami. Ungkapan tersebut menekankan bahwa tujuan utama dari *dashboard* adalah memberikan informasi secara cepat dan efisien kepada pengguna, sehingga mereka dapat dengan cepat membuat keputusan atau mengambil tindakan yang tepat. Berikut merupakan manfaat dari *dashboard* antara lain:

1) Mempermudah pemantauan
   Dashboard memungkinkan pengguna untuk memantau berbagai data penting dalam satu tempat.
2) Meningkatkan pemahaman
   Dashboard menyajikan data secara visual sehingga mudah dipahami dan dianalisis.
3) Mempercepat pengambilan keputusan
   Dashboard membantu pengguna untuk mengambil keputusan yang tepat dengan cepat berdasarkan data yang akurat.
4) Meningkatkan efisiensi
   Dashboard membantu pengguna untuk menghemat waktu dan sumber daya dengan memvisualisasi data yang kompleks.

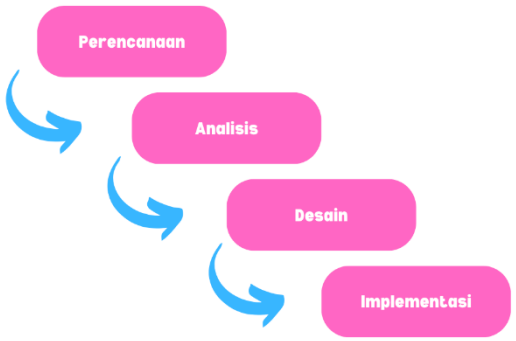
Disamping itu dashboard memiliki beberapa jenis yang masing-masing memliki fungsi dan karakteristiknya, yaitu :

1) *Dashboard* Operasional
   Digunakan untuk memantau kinerja dan kesehatan sistem secara real-time.
2) *Dashboard* Analitik
   Digunakan untuk menganalisis data dan mengidentifikasi tren.
3) *Dashboard* Strategis
   Digunakan untuk memantau kemajuan dan kinerja organisasi secara keseluruhan.

2.1.7 *SDLC*

Arief Yanto Rukmana, dkk (2023, p 125) mengungkapkan bahwa dalam upaya pengembangan sistem informasi, diperlukan langkah-langkah formal untuk mencegah kegagalan dan memastikan kesesuaian dengan kebutuhan organisasi atau perusahaan. Proses formal ini dikenal sebagai Siklus Hidup Pengembangan Sistem atau biasa disebut SDLC *(System Development Life Cycle)*.

SDLC merupakan suatu proses yang melibatkan pemahaman terhadap bagaimana sistem informasi digunakan dan mendukung kebutuhan bisnis dengan merancang sistem, membangunnya, dan mengirimkannya kepada pengguna. Menurut Alan Dannis dkk (2015) mengungkapkan bahwa tahapan *SDLC* berupa perencanaan, analisis, desain dan implementasi.



Gambar 2. 4 Contoh Tahapan *SDLC* (Alan Dannis dkk)

1. Perencanaan (*Planning*)
   Fase perencanaan merupakan awal dari *SDLC* di mana tujuan, ruang lingkup, sumber daya, jadwal, dan risiko proyek ditentukan. Pada tahap ini, tim

proyek dan pemangku kepentingan lainnya merumuskan rencana keseluruhan untuk proyek pengembangan perangkat lunak. Aktivitas yang biasa dilakukan dalam fase perencanaan meliputi :

1) Mendefinisikan tujuan proyek, Apa yang ingin dicapai dengan proyek ini?
2) Menganalisis kebutuhan pengguna, Siapa yang akan menggunakan *software* ini? Apa kebutuhan dan ekspektasi mereka?
3) Menentukan ruang lingkup proyek, Fitur apa yang akan diimplementasikan? Apa yang tidak termasuk dalam proyek?
4) Memperkirakan waktu dan biaya, Berapa lama waktu yang diperlukan untuk menyelesaikan proyek ini? Berapa besar biaya yang dibutuhkan?
5) Memilih metodologi pengembangan, Metodologi apa yang akan digunakan untuk mengembangkan *software*?

2. Analisis (*Analysis*)

Fase analisis bertujuan untuk memahami secara menyeluruh kebutuhan dan persyaratan pengguna untuk sistem yang akan dikembangkan. Tim proyek akan menganalisis persyaratan fungsional dan non-fungsional yang diperlukan oleh pengguna. Selama fase ini, sering kali dilakukan wawancara dengan pengguna potensial, observasi langsung, dan pengumpulan dokumen yang relevan. Aktivitas yang biasa dilakukan dalam fase Analisis meliputi:

1) Mengumpulkan data, Data dikumpulkan dari berbagai sumber, seperti wawancara, survei, observasi, dan dokumentasi.
2) Menganalisis data, Data dianalisis untuk memahami kebutuhan pengguna, fungsionalitas yang dibutuhkan, dan batasan-batasan sistem.
3) Membuat model sistem, Model sistem dibuat untuk menggambarkan bagaimana sistem akan bekerja.
4) Menetapkan spesifikasi sistem, Spesifikasi sistem dibuat untuk mendokumentasikan kebutuhan dan fungsionalitas sistem.

Hasil dari fase analisis ini biasanya berupa dokumen spesifikasi kebutuhan yang menjadi pedoman bagi tahap selanjutnya.

3. Desain (*Design*)

Fase desain melibatkan konversi persyaratan yang telah dianalisis menjadi rancangan teknis yang konkret untuk sistem yang akan dikembangkan. Hal tersebut mencakup perencanaan arsitektur sistem, pemodelan data, desain antarmuka pengguna, dan rancangan algoritma. Tujuan dari fase ini adalah untuk menghasilkan rencana detail tentang bagaimana sistem akan dibangun, termasuk pemilihan teknologi yang tepat. Aktivitas yang biasa dilakukan dalam fase Desain meliputi:

1) Merancang arsitektur sistem, Arsitektur sistem menentukan bagaimana komponen-komponen sistem akan terhubung dan berinteraksi.

2) Merancang antarmuka pengguna (*User Interface*), Antarmuka pengguna dirancang agar mudah digunakan dan dipahami oleh pengguna (*User*).
3) Merancang *database*, *Database* dirancang untuk menyimpan data yang digunakan oleh sistem.
4) Mengembangkan prototipe, Prototipe dibuat untuk menguji desain sistem dan mendapatkan masukan dari pengguna.

Hasil dari fase desain ini adalah serangkaian dokumen desain yang akan digunakan oleh tim pengembang selama implementasi.

4. Implementasi (*Implementation*)

Fase implementasi merupakan titik krusial dalam proses pengembangan perangkat lunak, di mana tim pengembang secara aktif menerjemahkan rancangan menjadi kumpulan kode yang berfungsi. Dalam proses ini, peran masing-masing tim, baik *frontend* maupun *backend*, menjadi sangat penting.

1) Frontend

Pada bagian *frontend*, fokus utamanya adalah pada pembuatan antarmuka pengguna yang responsif dan menarik. Tim *frontend* bertanggung jawab untuk mengimplementasikan desain yang telah dirancang secara visual ke dalam kode HTML, CSS, dan JavaScript. Mereka harus memastikan bahwa pengguna dapat berinteraksi dengan aplikasi secara intuitif dan lancar. Dalam hal ini, kemampuan untuk menguji antarmuka pengguna secara menyeluruh sangat penting untuk memastikan bahwa setiap elemen bekerja seperti yang diharapkan.

2) Backend

Sementara itu, di sisi *backend*, fokusnya adalah pada pengembangan logika bisnis dan pengelolaan data. Tim *backend* bertanggung jawab untuk membuat *API* yang kuat dan efisien untuk mengelola interaksi antara antarmuka pengguna dan basis data. Tim *backend* harus memastikan bahwa sistem dapat menangani permintaan dari klien dengan cepat dan dapat diandalkan, serta memiliki mekanisme keamanan yang kokoh untuk melindungi data sensitif.

Selama fase implementasi, integrasi antara *frontend* dan *backend* menjadi kunci. Tim harus bekerja sama untuk memastikan bahwa antarmuka pengguna terhubung dengan lancar dengan logika bisnis di backend. Hal tersebut melibatkan pengujian integrasi yang cermat untuk memastikan bahwa semua komponen berinteraksi dengan benar dan menghasilkan hasil yang diinginkan.

Selain itu, pengujian unit juga penting di kedua sisi. Tim harus secara rutin menguji setiap komponen perangkat lunak secara terpisah untuk memastikan bahwa mereka beroperasi dengan baik dan tidak menyebabkan masalah saat diintegrasikan ke dalam sistem secara keseluruhan.

Dalam keseluruhan proses implementasi, komunikasi dan kolaborasi yang efektif antara tim *frontend* dan *backend* sangat penting. Hanya dengan kerja sama yang solid, sistem dapat dikembangkan dengan lancar dan efisien, menghasilkan perangkat lunak yang berkualitas tinggi dan memuaskan kebutuhan pengguna dengan baik.

Hasil dari fase ini adalah sistem yang siap untuk diuji lebih lanjut dalam fase pengujian. Aktivitas yang biasa dilakukan dalam fase Implementasi meliputi :

1)) Mengembangkan kode program, Kode program ditulis berdasarkan desain yang telah dibuat.
2)) Menguji *software, Software* diuji untuk memastikan bahwa *software* tersebut bebas dari *bug* dan berfungsi sesuai dengan spesifikasi.
3)) Memperbaiki *bug, Bug* yang ditemukan selama pengujian diperbaiki.
4)) Melakukan *deployment*, Software diinstal pada lingkungan produksi.

Keempat fase dalam *SDLC* saling terkait dan berurutan. Fase perencanaan menentukan arah proyek, fase analisis memahami kebutuhan pengguna, fase desain merancang solusi, dan fase implementasi mengembangkan dan menguji software.

Terdapat dua konsep penting yang harus dipahami tentang *SDLC*. Pertama, pemahaman menyeluruh tentang tahapan, langkah-langkah, dan teknik yang harus dilewati oleh proyek sistem informasi. Bagian ini akan menjelaskan tentang tahapan, langkah-langkah, dan teknik yang terlibat dalam *SDLC*. Kedua, penting untuk menyadari bahwa *SDLC* merupakan proses peningkatan berkelanjutan. Hal tersebut berarti bahwa hasil dari fase analisis memberikan gambaran umum tentang apa yang akan dilakukan oleh sistem baru. Hasil ini kemudian digunakan sebagai masukan dalam fase desain, yang kemudian disempurnakan untuk menghasilkan deskripsi yang lebih rinci tentang bagaimana sistem akan dibangun. Hasil dari fase ini akan digunakan dalam fase implementasi untuk mengarahkan pembuatan sistem yang sebenarnya. Setiap fase bertujuan untuk menyempurnakan dan memperjelas pekerjaan yang telah dilakukan sebelumnya.

Konsep ini secara khas terkait dengan model pengembangan perangkat lunak "*Waterfall*" atau "Air Terjun". Model ini menggambarkan proses pengembangan perangkat lunak secara bertahap, dimulai dari tahap awal hingga tahap akhir, dengan setiap tahap menghasilkan keluaran yang menjadi masukan bagi tahap berikutnya. Analoginya, model Waterfall menyerupai aliran air terjun yang mengalir secara berurutan dari satu fase ke fase berikutnya.

Ronald E. Giachetti (2016, p. 88) mengungkapkan bahwa Model *Waterfall* menggambarkan sebuah metodologi di mana fase-fase dilakukan secara berurutan, menggunakan metafora air yang jatuh di tangga-tangga. Model

waterfall mendefinisikan fase-fase dalam hal aktivitas mereka dimana setiap fase memiliki satu aktivitas. Berikut adalah fase-fase yang ada dalam model *Waterfall*.

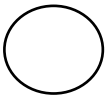
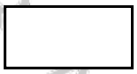
1. *Requirement Analysis* (Analisis Kebutuhan)
    1) Mengumpulkan dan mendokumentasikan semua kebutuhan sistem dari pemangku kepentingan.
    2) Membuat spesifikasi kebutuhan yang jelas dan lengkap.
2. *System Design* (Desain Sistem):
    1) Mengembangkan arsitektur sistem berdasarkan kebutuhan yang telah dianalisis.
    2) Membuat desain detail yang mencakup struktur data, arsitektur perangkat lunak, antarmuka, dan algoritma.
3. *Implementation* (Implementasi):
    1) Mengubah desain menjadi kode sumber.
    2) Mengembangkan modul sesuai dengan rancangan yang telah dibuat pada tahap sebelumnya.
4. *Integration and Testing* (Integrasi dan Pengujian):
    1) Mengintegrasikan semua modul yang telah dikembangkan.
    2) Melakukan pengujian untuk memastikan bahwa sistem bekerja sesuai dengan kebutuhan yang telah ditentukan, termasuk pengujian unit, integrasi, sistem, dan penerimaan.
5. *Deployment* (Penempatan):
    1) Memasang sistem yang telah diuji di lingkungan produksi.
    2) Melakukan konfigurasi yang diperlukan dan memastikan bahwa sistem berjalan dengan baik di lingkungan produksi.
6. *Maintenance* (Pemeliharaan):
    1) Memantau sistem untuk menemukan dan memperbaiki masalah yang muncul.
    2) Melakukan perbaikan bug dan peningkatan sistem berdasarkan umpan balik pengguna dan perubahan kebutuhan.

Kelebihan model *waterfall* mencakup struktur yang jelas, dokumentasi yang rinci, dan kesederhanaan manajemen proyek. Namun, kekurangannya adalah kurangnya fleksibilitas jika perubahan diperlukan setelah tahap dimulai, dan risiko tinggi terkait dengan menunggu hingga akhir untuk melihat produk yang sebenarnya. Model ini sering digunakan dalam proyek-proyek di mana persyaratan diperkirakan cukup stabil dan perubahan tidak terlalu sering terjadi.

2.1.8 Data *Flow Diagram*

Hanif Al Fatta (2007, p. 105) mengugkapkan bahwa data *flow diagram* (DFD) merupakan sebuah diagram yang digunakan untuk menggambarkan proses-proses yang terjadi dalam sistem yang akan dibangun. DFD menggambarkan proses-proses yang terjadi dalam sistem, aliran data antara proses-proses tersebut, penyimpanan data, serta sumber dan tujuan data. Diagram ini membantu dalam memahami, menganalisis, dan mendokumentasikan sistem, sehingga mempermudah pengembangan dan perbaikan sistem tersebut. Berikut merupakan komponen dari data flow diagram (DFD)

Tabel 2. 1 Data *Flow Diagram*

Simbol	Keterangan
	*Proses* (Process): Digambarkan dengan lingkaran atau oval, mewakili operasi atau kegiatan yang mengolah data. Setiap proses harus memiliki input dan output.
	Arus Data (Data *Flow*): Digambarkan dengan panah, menunjukkan arah aliran data antara proses, penyimpanan data, dan entitas eksternal.
	Penyimpanan Data (Data *Store*): Digambarkan dengan dua garis paralel atau persegi panjang terbuka di satu sisi, menunjukkan tempat data disimpan untuk digunakan di kemudian hari.
	Entitas Eksternal (*External Entity*): Digambarkan dengan persegi panjang, mewakili sumber atau tujuan data yang berada di luar sistem yang dimodelkan.

Sumber : https://www.dewaweb.com/blog/data-flow-diagram/

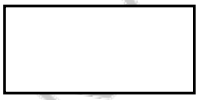
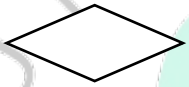
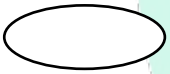
Didalam data flow diagram terdapat tingkatan dimulai dari DFD level 0 hingga DFD level 2 dengan detail sebagai berikut.,

1. Level 0 (*Context Diagram*): Merupakan gambaran umum dari sistem, menunjukkan hubungan antara sistem dengan entitas eksternal tanpa merinci proses internal.
2. Level 1: Menunjukkan rincian lebih lanjut dari proses-proses yang ada di Level 0. Setiap proses di Level 0 dipecah menjadi sub-proses di Level 1.
3. Level 2 dan seterusnya: Menyediakan rincian lebih lanjut dari proses-proses di Level 1, dan seterusnya, hingga mencapai tingkat detail yang diperlukan.

2.1.9 *Entity Relationship Diagram*

Hanif Al Fatta (2007, p. 105) mengugkapkan bahwa *Entity Relationship* Diagram (ERD) merupakan diagram yang menggambarkan bagaimana informasi dibuat, disimpan, dan digunakan dalam suatu sistem bisnis. Diagram ini sangat berguna dalam perancangan basis data dan sistem informasi karena membantu menggambarkan hubungan antara berbagai entitas dalam sistem tersebut. ERD terdiri dari beberapa komponen utama.

Tabel 2. 2 *Entity Relationship Diagram*

Simbol	Keterangan
	Entitas: Objek atau "*thing*" yang dapat diidentifikasi secara unik dalam sistem.
	Hubungan (*Relationship*)**:** Menggambarkan bagaimana entitas-entitas tersebut berinteraksi satu sama lain
	Atribut: Properti atau karakteristik dari entitas.
	Garis : Penghubung antara hubungan dengan entitas atau atribut dengan entitas

Sumber : https://steemit.com/esteem/@ryal/entity-relationship-diagram-erd-ec3f63c1800ff

Dalam ERD, hubungan (relasi) dapat melibatkan beberapa entitas, yang disebut dengan derajat relasi. Derajat relasi maksimum dikenal sebagai kardinalitas relasi, sedangkan derajat minimum disebut modalitas. Kardinalitas relasi menunjukkan jumlah maksimum entitas yang dapat terhubung dengan entitas lain dalam himpunan entitas yang berbeda. Kardinalitas relasi antara dua himpunan entitas (misalnya, A dan B) dapat berupa:

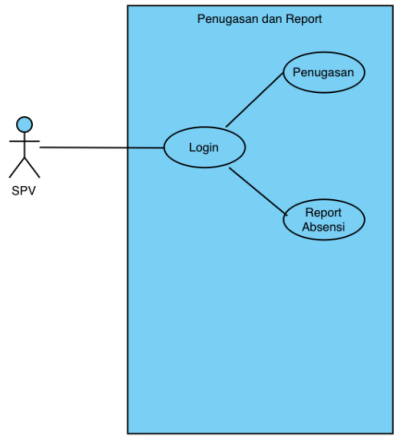
1. One-to-One (1:1)
   Satu entitas dalam set A berhubungan dengan tepat satu entitas dalam set B, dan sebaliknya.
2. One-to-Many (1:M)
   Satu entitas dalam set A berhubungan dengan banyak entitas dalam set B.
3. Many-to-Many (M:M)
   Banyak entitas dalam set A berhubungan dengan banyak entitas dalam set B.

2.1.10 *UML*

Ir. M. Farid Aziz, M.Kom (2005, p. 116) mengungkapkan bahwa *UML (Unified Modeling Language)* merupakan serangkaian simbol dan diagram yang digunakan untuk memodelkan perangkat lunak. Dengan menggunakan *UML*, perancangan perangkat lunak dapat direpresentasikan dalam bentuk

simbol dan diagram. Representasi ini kemudian dapat diubah menjadi kode program. *UML* menyediakan notasi grafis yang dapat digunakan untuk menggambarkan berbagai aspek dari sistem perangkat lunak, mulai dari struktur statis hingga perilaku dinamis. Terdapat 4 model dari *UML* yang sering digunakan diantaranya :

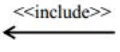
1. *Use Case* Diagram



Gambar 2. 5 Contoh *Use Case* Diagram

Use Case Diagram digunakan untuk menggambarkan interaksi antara sistem dan entitas luar (biasanya pengguna atau sistem lain). Berikut merupakan elemen utama dari *use case* diagram :

Tabel 2. 3. Elemen Use Case Diagram

Simbol	Keterangan
	*Aktor* : Berperan sebagai individu, sistem lain, atau alat saat berinteraksi dengan kasus penggunaan.
	*Use Case* : Representasi abstrak dan interaksi antara sistem dan pelaku.
	*Association* : Abstraksi dari koneksi antara aktor dan *use case*.
	Generalisasi : Menunjukkan bagaimana aktor disesuaikan untuk berpartisipasi dalam *use case* tertentu.
	*Include* : Menandakan bahwa suatu *use case* secara keseluruhan merupakan bagian dari fungsionalitas dari *use case* lain.

<<extend>>	*Extend* : Menunjukkan bahwa suatu *use case* adalah tambahan fungsional dari *use case* lain jika suatu kondisi tertentu terpenuhi.

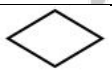
Sumber : https://www.dicoding.com/blog/contoh-use-case-diagram/

2. *Activity* Diagram

Gambar 2. 6 Contoh *Activity* Diagram

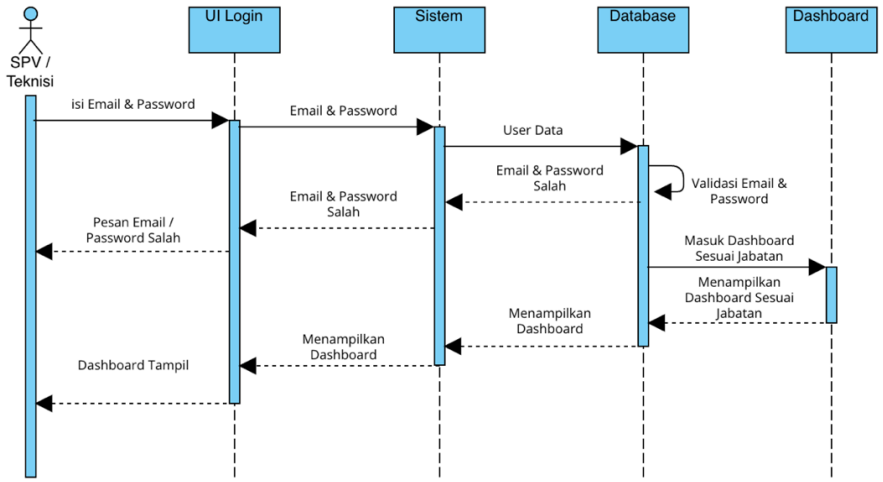
Activity Diagram menunjukkan alur kerja atau aktivitas dalam suatu sistem atau proses bisnis. Berikut merupakan elemen utama dari *activity* diagram :

Tabel 2. 4. Elemen Activity Diagram

Simbol	Keterangan
	Status Awal : Titik awal dari diagram.
	Aktivitas : Tindakan yang dilakukan oleh sistem.
	Percabangan : Titik di mana terdapat jalur alternatif yang dapat diambil oleh sistem dan sistem harus membuat pilihan.
	Penggabungan : Titik di mana dua atau lebih jalur bergabung Kembali.
	Status Akhir : Titik akhir dari diagram.
	*Swimlane* : *Swimlane* mengelompokkan unit bisnis yang bertanggung jawab atas aktivitas yang terjadi.

Sumber : https://www.dicoding.com/blog/apa-itu-activity-diagram/

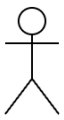
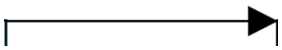
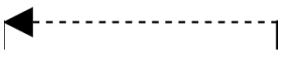
3. *Sequence* Diagram:

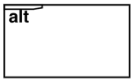


Gambar 2. 7 Contoh *Sequence* Diagram

Sequence Diagram menunjukkan urutan pesan atau interaksi antar objek dalam suatu skenario. Berikut merupakan elemen utama dari *sequence* diagram:

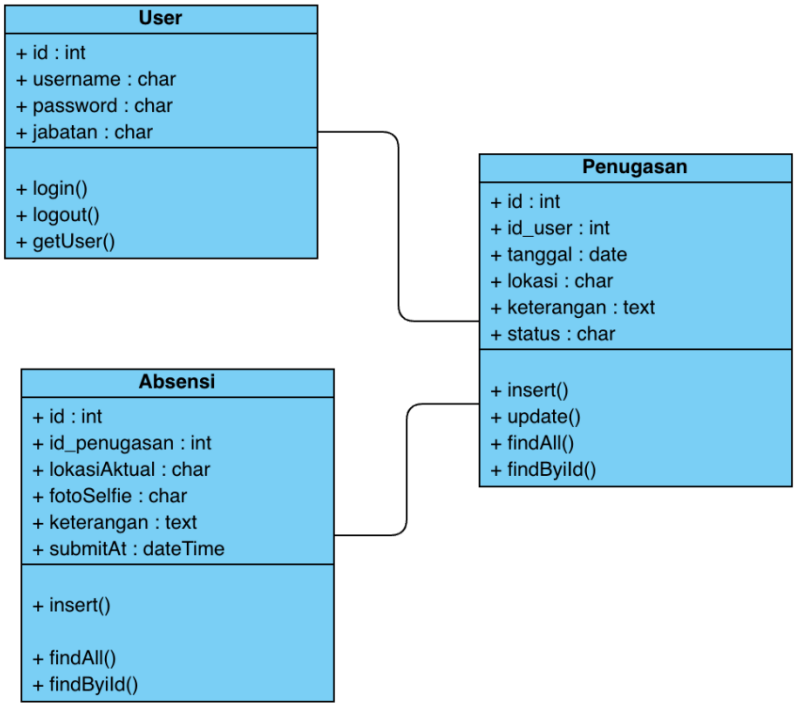
Tabel 2. 5. Elemen *Sequence* Diagram

Simbol	Keterangan
	*Actor* : Representasi struktur sistem yang menjadi dasar dalam pembentukan basis data.
	*Object* : Berfungsi untuk menerima atau mengirimkan sebuah pesan
	Life Line : Komponen yang ditunjukkan dengan garis putus yang terhubung dengan objek.
	*Recursive* : Pesan yang merujuk pada dirinya sendiri.
	*Activation* : Mewakili proses aktivasi operasi dalam jangka waktu tertentu.
	*Message* : sebagai interaksi antar object atau class.
	*Message To Self :* sebagai interaksi balik dari object atau class sebelumnya.

	*Frame* :berfungsi untuk menjelaskan konteks dari diagram urutan

Sumber : https://www.dicoding.com/blog/apa-itu-sequence-diagram/

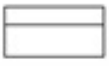
4. *Class* Diagram



Gambar 2. 8 Contoh *Class* Diagram

Class Diagram digunakan untuk menggambarkan struktur kelas dan hubungan antar kelas dalam sistem. Berikut merupakan elemen utama dari *class* diagram:

Tabel 2. 6. Elemen *Class* Diagram

Simbol	Keterangan
	Generalization : Hubungan di mana objek turunan berbagi sifat dan struktur data dari objek di atasnya, yaitu objek yang lebih umum.
	*Nary Association* : Usaha untuk menghindari hubungan asosiatif yang melibatkan lebih dari dua objek.
	*Class* : Kumpulan objek yang memiliki atribut dan operasi yang serupa.
	*Collaboration* : Penjelasan tentang urutan tindakan yang dilakukan oleh sistem yang menghasilkan hasil yang terukur bagi seorang pelaku.

	Realization : Tindakan yang sepenuhnya dilakukan oleh sebuah objek.
	Dependency : Hubungan di mana perubahan pada satu elemen yang mandiri akan berdampak pada elemen yang tergantung padanya.
	Association : Hubungan yang menghubungkan satu objek dengan objek lainnya.

Sumber : https://mesran.blogspot.com/2013/05/class-diagram.html

UML digunakan untuk membantu pemahaman, desain, dan dokumentasi sistem perangkat lunak secara visual. Setiap diagram memiliki tujuan dan konteks penggunaan tertentu dalam proses pengembangan perangkat lunak.

2.1.11 Penjadwalan (*Scheduler*)

Dana Marsetiya Utama (2023, p. 1) mengungkapkan bahwa Penjadwalan adalah perencanaan urutan kerja dan alokasi sumber daya (*resource*) pada setiap operasi. *Scheduler* berfungsi untuk merencanakan dan mengatur alokasi sumber daya atau mesin untuk mengeksekusi serangkaian tugas dalam suatu jangka waktu tertentu. Hal ini melibatkan pengaturan urutan kerja dan alokasi sumber daya, baik dalam hal waktu maupun fasilitas, untuk setiap operasi yang perlu diselesaikan. Dengan kata lain, penjadwalan adalah proses perencanaan yang melibatkan pengorganisasian tugas dan sumber daya guna mencapai tujuan tertentu dalam waktu yang ditentukan.

Penjadwalan lebih dari Sekedar Urutan Kerja, Penjadwalan bukan sekadar menyusun daftar tugas. penjadwalan merupakan seni merancang strategi, memetakan alur kerja, dan mengalokasikan sumber daya secara optimal. penjadwalan adalah kunci untuk mencapai tujuan dalam waktu yang ditentukan, meminimalisir pemborosan, dan memaksimalkan hasil. Berikut merupakan beberapa manfaat dari dilakukannya penjadwalan.

1) Efisiensi
   Penjadwalan membantu menyelesaikan tugas dengan lebih cepat dan tepat waktu, mengantarkan pada gerbang efisiensi.
2) Kualitas
   Dengan fokus pada tugas-tugas penting, hasil yang berkualitas pun tercipta.
3) Produktivitas
   Pencapaian optimal dalam waktu singkat, mengantarkan pada puncak performa.

Untuk membuat suatu penjadwalan yang efektif diperlukan langkah-langkah berikut :

1) Menentukan Tujuan, Apa yang ingin dicapai? Menjadikannya visi yang jelas.

2) Mendefinisikan Tugas, Merinci langkah-langkah yang mengantarkan pada visi.
3) Memperkirakan Waktu, Memetakan durasi setiap langkah, mengukur waktu yang dibutuhkan.
4) Menentukan Sumber Daya, Mengidentifikasi alat, bahan, dan personil yang diperlukan.
5) Menyusun Jadwal, Merajut urutan langkah dan mengalokasikan sumber daya, bagaikan seniman yang melukis *masterpiece* alur kerja.
6) Memantau dan Mengevaluasi, Melacak kemajuan dan melakukan penyesuaian, memastikan alur kerja tetap optimal.

2.1.12 *Api*

Waskitho Cito Adiwiguno, Muhammad Wildan Khalilurrahman, Rolly Maulana Awangga (2023, p. 5) mengungkapkan bahwa *API (Application Programming Interface)* adalah suatu antarmuka yang memungkinkan pengembang perangkat lunak untuk mengakses fungsionalitas yang telah dibuat oleh pengembang lain tanpa perlu mengetahui rincian implementasi internalnya. Dengan kata lain *API* berperan sebagai jembatan komunikasi antara dua aplikasi atau komponen perangkat lunak yang berbeda, memungkinkan mereka untuk saling berinteraksi dan berbagi data atau fungsi tanpa harus mengetahui secara detail bagaimana komponen tersebut diimplementasikan.

Konsep utama dari *API* adalah abstraksi. *API* menyediakan suatu abstraksi atas fungsionalitas atau layanan yang disediakan oleh suatu sistem atau komponen, menyembunyikan detail-detail kompleks tentang bagaimana fungsi-fungsi tersebut diimplementasikan secara internal. Dengan demikian, pengguna *API* hanya perlu memahami cara menggunakan antarmuka yang telah ditentukan oleh pengembang tanpa harus terlibat dalam kompleksitas dari implementasi di balik layar.

API dapat mengambil berbagai bentuk, termasuk *API web*, *hardware API*, dan *Sofware API*. API web adalah antarmuka yang disediakan oleh aplikasi web atau layanan web yang memungkinkan pengguna untuk berinteraksi dengannya melalui protokol komunikasi web seperti HTTP. Contoh dari API web termasuk RESTful API dan GraphQL API yang digunakan untuk mengakses dan mengelola data dari server web.

2.1.13 *Json*

Ben Smith (2015, p. 37) mengungkapkan bahwa *JSON (JavaScript Object Notation)* dikenal sebagai standar pertukaran data, yang secara tersirat menyiratkan bahwa ia dapat digunakan sebagai format data di mana pun pertukaran data terjadi. *JSON* berbasis teks dan terinspirasi oleh sintaks

objek dalam bahasa pemrograman JavaScript, namun *JSON* dapat digunakan dengan berbagai bahasa pemrograman yang mendukung manipulasi *string*.

JSON digunakan sebagai standar pertukaran data menunjukkan fleksibilitasnya didalam beradaptasi dengan lingkungan teknologi yang beragam. Sifatnya yang berbasis teks memungkinkan *JSON* digunakan dalam konteks apapun di mana pertukaran data diperlukan, baik itu dalam aplikasi *web*, perangkat lunak desktop, aplikasi mobile, hingga sistem terdistribusi.

Namun, yang membuatnya lebih menarik adalah kemampuannya untuk digunakan dengan berbagai bahasa pemrograman lainnya. Oleh karena itu *JSON* menjadi pilihan yang populer dan dapat diandalkan dalam ekosistem pengembangan perangkat lunak yang berbeda.

Pentingnya *JSON* dalam pertukaran data tidak hanya terbatas pada lingkungan pengembangan perangkat lunak, tetapi juga merambah ke *domain* lain seperti pertukaran data antar sistem, pengiriman data melalui jaringan, dan penyimpanan data dalam format yang mudah diakses dan dimanipulasi. Dengan segala keunggulan ini, tidak mengherankan jika *JSON* terus menjadi salah satu standar *de facto* dalam pertukaran data di berbagai industri dan aplikasi teknologi saat ini.

2.1.14 *Database*

Ramadhani (2016, p. 105) *Database* merupakan tempat penampungan semua data yang ada didalam sistem komputer. Didalam *database*, data disimpan dalam tabel atau struktur data lainnya dan diorganisir sedemikian rupa sehingga dapat dengan efisien dikelola, diakses, dan dimanipulasi. *Database* digunakan untuk menyimpan informasi yang berkaitan dengan suatu domain atau aplikasi tertentu. Beberapa konsep dasar dalam *database* melibatkan:

1) Tabel

   Tabel adalah struktur dasar dalam *database* yang menyimpan data dalam bentuk baris dan kolom. Setiap baris dalam tabel mewakili satu *record* atau entitas, sedangkan setiap kolom mewakili atribut atau *field* dari *record* tersebut.

2) Relasi

   Hubungan antar tabel dalam database dapat ditentukan melalui kunci asing (*foreign key*). Hal tersebut memungkinkan penggabungan atau pengambilan data dari beberapa tabel sekaligus.

4) *Query*

Query digunakan untuk mengambil atau memanipulasi data dalam *database*. Bahasa pemrograman yang umum digunakan untuk menulis *query* adalah *SQL (Structured Query Language).*

5) Indeks

Indeks digunakan untuk meningkatkan kinerja pencarian data dalam *database*. Indeks mempercepat proses pencarian data dengan menciptakan struktur yang memetakan nilai tertentu ke lokasi fisik data.

6) Integritas Data

Database memiliki mekanisme untuk menjaga integritas data, seperti aturan unik, kunci primer, dan kunci asing, untuk memastikan keakuratan dan konsistensi data.

7) Sistem Manajemen Basis Data (DBMS)

DBMS (*Database Management System*) adalah perangkat lunak yang digunakan untuk membuat, mengelola, dan menyediakan antarmuka untuk bekerja dengan database. Contoh DBMS meliputi MySQL, PostgreSQL, Microsoft SQL Server, dan Oracle *Database*.

Database digunakan dalam berbagai konteks, termasuk bisnis, pemerintahan, pendidikan, dan aplikasi perangkat lunak. Mereka membantu dalam penyimpanan dan pengelolaan data dengan efisien, serta menyediakan mekanisme untuk pencarian, pembaruan, dan analisis data. Selain itu terdapat berbagai macam database antara lain:

1) *Relational Database Management System (RDBMS)*

RDBMS menggunakan struktur tabel yang terdiri dari baris dan kolom untuk menyimpan dan mengorganisir data. Relasi antar tabel ditentukan oleh kunci primer dan kunci asing. Contoh dari *RDMS* yaitu MySQL, PostgreSQL, Oracle Database, Microsoft SQL Server.

2) *NoSQL Database*

NoSQL databases menggunakan model data yang berbeda dari *RDBMS*. Mereka dapat menyimpan data dalam bentuk dokumen, kolom, grafik, atau jenis lainnya. Mereka sering digunakan untuk aplikasi yang membutuhkan skalabilitas horizontal dan fleksibilitas skema. Contoh dari *NoSQL* database yaitu MongoDB, Cassandra, *Couchbase*.

3) *Graph Database*

Graph databases dirancang khusus untuk menyimpan dan mengelola data yang memiliki struktur grafik. Mereka menggunakan *node*, relasi, dan properti untuk merepresentasikan dan menghubungkan entitas dalam basis data. Contoh dari *Graph Database* yaitu Neo4j, Amazon Neptune, OrientDB.

4) *In-Memory Database*

In-Memory database menyimpan data dalam memori utama komputer, yang memungkinkan akses data yang sangat cepat. Mereka sering digunakan untuk mengatasi kinerja dan latensi dalam aplikasi yang membutuhkan respons waktu yang cepat. Contoh dari *In-Memory Database* yaitu Redis, Memcached, VoltDB.

5) *Time-Series Database*

Time-series databases dioptimalkan untuk menyimpan dan menganalisis data yang dikumpulkan dari waktu ke waktu, seperti data sensor, log, atau metrik. Mereka menawarkan fitur-fitur khusus untuk mengelola data berbasis waktu dengan efisien. Contoh dari *Time-Series Database* yaitu InfluxDB, Prometheus, TimescaleDB.

6) *Document-Oriented Database*

Database yang berorientasi pada dokumen menyimpan data dalam bentuk dokumen, sering dalam format seperti *JSON* atau *BSON*. Mereka cocok untuk aplikasi yang membutuhkan fleksibilitas skema dan menyimpan data semi-terstruktur. Contoh dari *Document-Oriented Database* yaitu CouchDB, Firebase Firestore, Amazon DocumentDB.

Setiap jenis *database* memiliki kelebihan dan kekurangan yang berbeda, dan pemilihan yang tepat tergantung pada kebutuhan spesifik aplikasi atau proyek yang sedang dihadapi.

2.1.15 *Web*

Deris Stiawan (2005, p 149) mengungkapkan bahwa *Web* atau *World Wide Web* (WWW) adalah sistem yang memfasilitasi pertukaran informasi melalui jaringan Internet. *Web* dibangun dengan menggunakan program yang dikenal sebagai server *web*. Saat ini, pengaruh *web* telah menjangkau berbagai sektor kehidupan, termasuk dunia bisnis, administrasi pemerintahan, pendidikan, kegiatan keagamaan, interaksi sosial, dan warisan budaya.

Untuk memastikan bahwa informasi yang disajikan dalam halaman web dapat diakses oleh pengguna Internet, diperlukan keberadaan mesin server *web*. Mesin ini bertugas untuk menanggapi permintaan yang datang dari pengguna yang ingin mengakses informasi tersebut. Sebagaimana yang telah diuraikan sebelumnya, beberapa jenis server *web* yang paling umum digunakan adalah Apache dan IIS. Mesin-mesin ini memiliki peran vital dalam memfasilitasi akses pengguna ke berbagai halaman *web* di seluruh dunia, memainkan peranan penting dalam memastikan konektivitas yang efisien dan andal di ranah digital saat ini.

2.2 Tinjauan Studi

Penelitian ini dilakukan setelah penulis mengumpulkan informasi dari berbagai jurnal untuk digunakan sebagai rujukan dalam penulisan ini. Berikut beberapa sumber yang dapat menyokong penelitian ini, adalah sebagai berikut:

1. Tinjauan studi pada jurnal yang berjudul "***MONITORING* SERVER DENGAN PROMETHEUS DAN GRAFANA SERTA NOTIFIKASI TELEGRAM**". Jurnal ini ditulis oleh Dede Rahman, Hidra Amnur, dan Indri Rahmayuni pada tahun 2020 dan diterbitkan oleh Jurnal Ilmiah Teknologi Sistem Informasi. Masalah yang diangkat dalam artikel ini adalah peningkatan kompleksitas jaringan, kelemahan dalam sistem topologi jaringan, dan keterbatasan efisiensi pemantauan jaringan dengan metode tradisional. Administrator jaringan, yang bertanggung jawab untuk memastikan jaringan beroperasi dengan optimal, sering kali menemukan bahwa metode pemantauan konvensional kurang efisien karena memerlukan kehadiran fisik mereka di depan layar. Untuk mengatasi masalah ini, para peneliti memperkenalkan solusi dengan menggunakan aplikasi Prometheus dan Grafana untuk pemantauan jaringan secara real-time. Grafana menyediakan informasi langsung mengenai kondisi komponen jaringan dan mengirimkan pesan ke administrator melalui Telegram. Artikel ini menyimpulkan bahwa pemantauan server berhasil dilakukan menggunakan Prometheus dan Grafana pada server, dengan Ubuntu Server versi 18.04 sebagai sistem operasi yang digunakan. Sistem ini mampu memberikan notifikasi kepada administrator saat terjadi masalah seperti kegagalan CPU, memori, atau layanan Apache dan MySQL. Selain itu, Grafana secara konsisten mengirimkan pemberitahuan melalui Telegram jika kondisi server melebihi batas yang telah ditetapkan atau jika layanan Apache atau MySQL berhenti beroperasi. Pemilihan konfigurasi didasarkan pada spesifikasi minimal untuk membuat instance yang mencakup VCPU, RAM, dan Disk.
2. Tinjauan studi pada jurnal yang berjudul "**DESAIN DAN IMPLEMENTASI SISTEM *MONITORING* SUMBER DAYA SERVER MENGGUNAKAN ZABBIX 4.0**". Jurnal ini ditulis oleh Sulasno Sulasno dan Rakhmat Saleh pada tahun 2020. Artikel ini membahas dampak peningkatan penggunaan aplikasi di instansi pemerintah dan perusahaan swasta yang mengakibatkan peningkatan jumlah server yang digunakan. Tantangan yang dihadapi banyak server tanpa sistem pemantauan yang memadai adalah kesulitan dalam memantau penggunaan sumber daya secara real-time. Ketika terjadi gangguan operasional seperti kapasitas hard disk penuh atau kinerja CPU melebihi batas, pemantauan yang cepat untuk penanganan segera menjadi sulit. Penelitian ini bertujuan untuk mengembangkan sistem pemantauan sumber daya server secara real-time. Metode penelitian meliputi pengumpulan data, desain sistem, instalasi, konfigurasi, dan implementasi menggunakan Zabbix 4.0 di Pusat Pendayagunaan Informatika dan Kawasan Strategis Nuklir (PPIKSN) Badan Tenaga Nuklir Nasional (BATAN). Hasil penelitian mencakup sistem pemantauan penggunaan sumber daya server, seperti kapasitas hard disk,

kinerja CPU, penggunaan memori, besaran paket data ethernet, dan notifikasi email. Sistem ini telah digunakan oleh Administrator server, memberikan manfaat berupa kemudahan pemantauan sumber daya server secara real-time, peningkatan efisiensi dan produktivitas, serta identifikasi dan penanganan masalah sebelum muncul keluhan, yang pada akhirnya meningkatkan kualitas layanan.

3. Tinjauan studi pada jurnal yang berjudul "**RANCANG BANGUN SISTEM INFORMASI *MONITORING* SERVER VIRTUAL BERBASIS WEB MENGGUNAKAN SCRIPT *MONITORING* PADA PROMOX VIRTUAL ENVIRONTMENT**". Jurnal ini ditulis oleh Evan Prima Prasetyo, Joseph Dedy Irawan, dan F. X. Ariwibisono pada tahun 2022. Artikel ini menjelaskan pentingnya monitoring untuk memastikan server dapat beroperasi dengan optimal. Proses monitoring mencakup pemantauan perangkat keras dan perangkat lunak yang digunakan pada server. Informasi yang diambil dari server diolah menjadi data pemantauan yang ditampilkan untuk menunjukkan status kondisi server. Jika terdapat hasil pemantauan yang abnormal, seperti overheating atau kegagalan aplikasi server, administrator sistem dapat segera melakukan perbaikan yang diperlukan. Administrator sistem juga dapat melakukan perawatan server secara efisien berdasarkan data pemantauan yang terkumpul. Pembuatan aplikasi monitoring server virtual berbasis web menggunakan skrip pemantauan memungkinkan pengumpulan data pemantauan dari server virtual dan penampilannya melalui antarmuka web. Hasil akhir dari aplikasi ini adalah kemampuannya untuk menampilkan data pemantauan perangkat keras dan perangkat lunak server virtual, pemantauan aplikasi server, grafik penggunaan sumber daya, fitur notifikasi peringatan, dan fitur pengendalian server virtual berbasis SSH yang dapat diakses melalui browser web. Selama server web Apache2 beroperasi, aplikasi pemantauan berhasil melakukan pemantauan, menjalankan fitur notifikasi peringatan, dan mengendalikan server dengan tingkat keberhasilan 100%. Namun, akses ke aplikasi pemantauan tidak mungkin jika server web Apache2 dimatikan.
4. Tinjauan studi pada jurnal yang berjudul "**APLIKASI *MONITORING* DAN *CONTROLLING* SERVER MENGGUNAKAN PROTOCOL ICMP (*INTERNET CONTROL MESSAGE PROTOCOL*) DAN *SSH (SECURE SHELL)* BERBASIS *WEBSITE***". Jurnal ini ditulis oleh Redo Yanuar Pratama, Mira Orisa, dan FX Ariwibisono pada tahun 2020. Artikel ini menjelaskan bahwa sistem pemantauan server sangat penting bagi seorang administrator. Menjamin setiap server dalam kondisi optimal adalah tugas utama administrator, mengingat server harus selalu online atau aktif setiap saat. Metode pemantauan yang umum digunakan saat ini melibatkan pemeriksaan individual setiap server oleh administrator. Dengan pendekatan ini, administrator dapat mengidentifikasi kondisi fisik dan status server, baik dalam keadaan aktif (up) maupun non-aktif (down). Penggunaan protokol ICMP (Internet Control Message Protocol) memungkinkan pemantauan server secara real-time, sementara protokol SSH (Secure Shell) digunakan untuk mengendalikan server dan mentransfer file ke server. Aplikasi pemantauan dan

kontrol server ini telah diimplementasikan dan diuji coba di jaringan lokal Laboratorium Jaringan Komputer Teknik Informatika S-1 ITN Malang. Melalui aplikasi ini, administrator dapat memonitor kondisi server secara real-time, baik saat server aktif maupun tidak aktif. Jika server tidak aktif, administrator akan menerima notifikasi. Fitur kontrol server dalam aplikasi ini mencakup kemampuan untuk mematikan atau me-restart server, mengelola layanan yang berjalan di server, serta mengirimkan file ke server.

5. Tinjauan studi pada jurnal yang berjudul "***MONITORING* APLIKASI MENGGUNAKAN *DASHBOARD* UNTUK SISTEM INFORMASI AKUNTANSI PEMBELIAN DAN PENJUALAN (STUDI KASUS : UD APUNG)".** Jurnal ini ditulis oleh Sufia Maulida, Fikri Hamidy, dan Agung Deni Wahyudi pada tahun 2020. Artikel ini membahas masalah operasional yang dihadapi oleh UD Apung, sebuah perusahaan dagang yang fokus pada penjualan alat bangunan. Proses manual dalam penjualan dan pembelian, terutama dalam pembuatan nota penjualan dan faktur pengiriman, menyebabkan keterlambatan pengolahan transaksi. Pencatatan manual pembelian barang dan keterbatasan dalam monitoring serta perekapan data menggunakan Microsoft Excel juga menjadi kendala, memperlambat pembuatan laporan. Meskipun telah menerapkan sistem informasi akuntansi, masih terdapat keterbatasan dalam pemrosesan data penjualan dan pembelian. Meskipun sudah mencoba menggunakan dashboard interaktif, masih ada ketergantungan pada Microsoft Excel. Solusi yang diusulkan adalah mengembangkan sistem informasi akuntansi yang lebih canggih, integratif, dan menyeluruh. Implementasi dashboard interaktif yang sepenuhnya terintegrasi dapat meningkatkan akses real-time, mempercepat pengambilan keputusan, dan meningkatkan efektivitas dalam mengelola dan mengoptimalkan kinerja perusahaan. Dengan demikian, UD Apung dapat mengatasi kendala operasional dengan lebih efisien dan adaptif terhadap dinamika pasar. Kesimpulan dari artikel ini adalah pemantauan aplikasi melalui dashboard dilakukan dengan mengamati grafik transaksi penjualan dan pembelian barang. Hal ini memungkinkan pimpinan untuk segera mengetahui total transaksi penjualan dan pembelian perusahaan dengan cepat. Selain itu, proses input data penjualan dan pembelian tidak lagi memerlukan pencatatan manual seperti sebelumnya. Hal ini dapat mengurangi risiko kesalahan dalam memasukkan data penjualan dan pembelian, yang dapat menyebabkan kerugian perusahaan.