BAB 2
LANDASAN TEORI

2.1 SIS-PIK

Singapore School Pantai Indah Kapuk (SIS-PIK) adalah bagian dari jaringan SIS *Group of Schools* yang didirikan pada tahun 1996 dengan dukungan dari pemerintah Singapura dan Raffles *Institution*. SIS-PIK sendiri mulai beroperasi pada tahun 2004, berlokasi di Jl. Mandara Indah 4, Pantai Indah Kapuk, Jakarta Utara.

Sejak awal berdirinya, SIS-PIK memiliki visi untuk menggabungkan tradisi pendidikan terbaik dari Singapura, Cambridge, BTEC, dan *International Baccalaureate* (IB). Pendekatan ini dirancang untuk mengembangkan rasa ingin tahu, keterbukaan pikiran, dan kreativitas siswa, sambil menanamkan nilai-nilai seperti ketekunan, kerja keras, dan kerendahan hati.

SIS-PIK menawarkan program pendidikan mulai dari Taman Kanak-Kanak hingga Sekolah Menengah Atas, dengan kurikulum yang dirancang untuk mempromosikan kecintaan terhadap pengetahuan dan motivasi untuk mencapai tujuan akademis serta pribadi yang tinggi. Lingkungan belajar di SIS-PIK dirancang untuk membantu siswa berpartisipasi dalam dunia yang terus berubah sebagai warga global masa depan.

Dengan komitmen terhadap keunggulan akademik dan pengembangan karakter, SIS-PIK terus berupaya mempersiapkan siswa untuk menghadapi tantangan abad ke-21 dan menjadi pemimpin masa depan yang kompeten dan berintegritas.[5].

2.2 Asp.Net

Microsoft .NET *Framework* adalah sebuah komponen yang dapat ditambahkan ke sistem operasi Microsoft *Windows* yang telah diintergrasi ke dalam *Windows*. Kerangka kerja ini menyediakan sejumlah besar solusi-solusi program untuk memenuhi kebutuhan-kebutuhan umum suatu program baru.

ASP.NET adalah kerangka kerja yang dikembangkan oleh Microsoft untuk membangun aplikasi *web* dinamis, interaktif, berkinerja tinggi dan berbasis *cloud*. *Framework* ini berjalan di atas .NET dan memungkinkan pengembang untuk membuat situs *web*, layanan *API*. Dalam pengembangan perangkat lunak, ASP.NET

menawarkan berbagai fungsi dan kegunaan yang signifikan, seperti yang diuraikan dalam beberapa jurnal berikut:

1. **Penerapan Arsitektur MVC (*Model-View-Controller*)**

   ASP.NET mendukung arsitektur MVC, yang membagi aplikasi menjadi tiga bagian utama: *Model*, *View*, dan *Controller*. Pendekatan ini mempermudah proses pengembangan dan pemeliharaan dengan memisahkan logika bisnis, tampilan pengguna, serta alur kontrol. Dengan demikian, pengembang dapat bekerja secara simultan pada setiap komponen tanpa mengganggu bagian lainnya [6].

2. **Pengembangan *Website* Dinamis**

   ASP.NET mendukung pengembangan situs *web* dinamis yang interaktif, memberikan informasi serta layanan kepada pengguna. Dengan memanfaatkan bahasa pemrograman seperti C# dan database seperti SQL *Server*, pengembang dapat menciptakan aplikasi *web* yang responsif serta disesuaikan dengan kebutuhan pengguna [7].

3. **Integrasi dengan Teknologi *Web* Utama**

   ASP.NET menyediakan kemampuan yang andal dalam pengembangan situs *web* dengan dukungan teknologi *web* utama seperti HTML, CSS, dan JavaScript. Dengan fitur seperti *Razor Pages*, proses pengkodean menjadi lebih efisien, memungkinkan pengelolaan halaman yang lebih mudah, serta memastikan struktur proyek yang tertata dengan baik [8].

4. **Dukungan untuk Berbagai Bahasa Pemrograman**

   ASP.NET menyediakan fleksibilitas dalam pemilihan bahasa pemrograman, seperti VB.NET, C#, J#, dan C++, sehingga memungkinkan pengembang untuk memilih bahasa yang paling sesuai dengan kebutuhan dan preferensi proyek. Setiap bahasa yang didukung oleh platform ini memiliki karakteristik dan fitur khusus yang dapat dimanfaatkan untuk meningkatkan efisiensi serta efektivitas dalam proses pengembangan aplikasi. Fleksibilitas ini menjadi keunggulan utama ASP.NET, khususnya dalam proyek pengembangan sistem informasi berbasis web yang menuntut kemampuan adaptasi terhadap berbagai pendekatan teknis.[9].

5. **Keamanan dan Autentikasi**

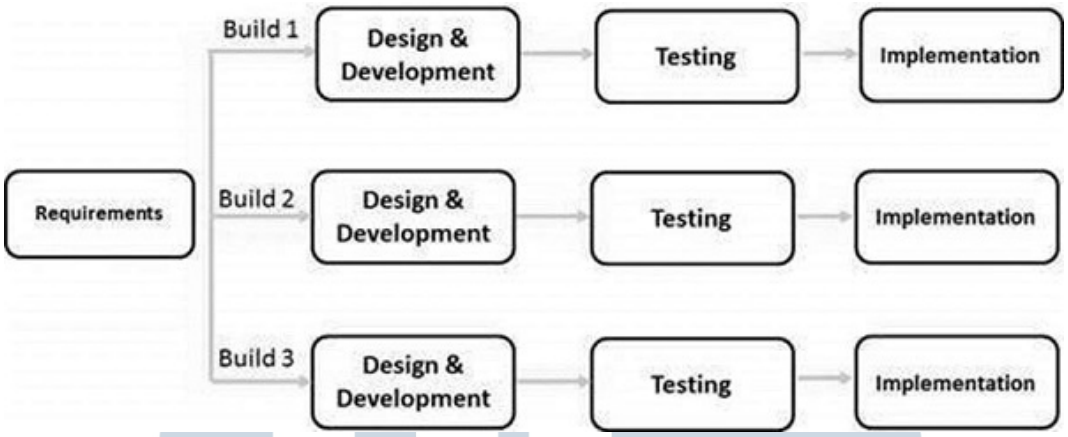
ASP.NET memiliki fitur autentikasi bawaan, termasuk *Windows Authentication*, yang memastikan hanya pengguna yang memiliki izin yang dapat mengakses aplikasi. Dengan mekanisme ini, keamanan aplikasi *web* meningkat, sehingga data sensitif terlindungi dari akses yang tidak sah [10].

Program yang dikembangkan untuk .NET *Framework* dijalankan dalam sebuah lingkungan perangkat lunak yang mengatur berbagai persyaratan *runtime*. Lingkungan ini, yang merupakan bagian dari .NET *Framework*, dikenal sebagai *Common Language Runtime* (CLR). CLR berfungsi sebagai *application virtual machine*, memungkinkan *programmer* untuk menulis kode tanpa perlu menyesuaikan dengan spesifikasi CPU yang digunakan. Selain itu, CLR menyediakan berbagai layanan penting seperti keamanan, manajemen memori, *garbage collection*, serta *exception handling* saat *runtime*. Bersama dengan *class library*, CLR menjadi komponen utama dalam .NET *Framework*. Kerangka kerja ini dirancang untuk mempermudah pengembangan aplikasi, sekaligus meningkatkan keamanan sistem dan mengurangi potensi ancaman terhadap perangkat lunak serta komputer [6].

2.3 Model Iteratif

Model iteratif adalah pendekatan dalam pengembangan perangkat lunak yang dilakukan secara berulang (iteratif) dan bertahap (incremental). Metode ini memungkinkan setiap tahapan pengembangan, mulai dari perencanaan, analisis, perancangan, hingga implementasi, dilakukan secara bertahap sesuai dengan kebutuhan. Setiap iterasi mencakup analisis kebutuhan, desain, implementasi, dan pengujian yang diulang secara terus-menerus hingga sistem mencapai tingkat kesempurnaan yang diinginkan [11].

Pendekatan ini memungkinkan pengembang untuk terlebih dahulu membangun sebagian kecil dari keseluruhan fitur sistem, kemudian menambah dan menyempurnakannya pada iterasi berikutnya. Hal ini memberikan fleksibilitas dalam pengembangan, sehingga pengembang dapat fokus pada bagian-bagian tertentu, memperbaiki dan menambah fitur sistem secara bertahap untuk mencapai hasil akhir yang lebih baik [12].



Gambar 2.1. Gambar Iteratif [1]

.

2.3.1 Tahapan dalam model iteratif sebagai berikut

1. ***Requirements*:**
   Seluruh kebutuhan sistem dikumpulkan di awal. Namun, kebutuhan ini tidak langsung diimplementasikan secara keseluruhan. Sebaliknya, kebutuhan dibagi menjadi bagian-bagian yang lebih kecil yang akan dikembangkan dalam beberapa *build*.

2. ***Build* 1:**
   Pada tahap ini, bagian dasar dari sistem mulai dikembangkan. Tahapannya meliputi:

   - **Design & Development:** Merancang dan mengembangkan bagian awal dari sistem.
   - **Testing:** Melakukan pengujian terhadap bagian yang telah dikembangkan untuk memastikan fungsionalitasnya berjalan dengan baik.
   - **Implementation:** Bagian tersebut diimplementasikan dan siap digunakan.

3. ***Build* 2 dan *Build* 3:**
   Setelah *Build* 1 selesai, proses dilanjutkan ke *Build* 2 dan *Build* 3. Setiap *build* menambahkan fungsi baru pada sistem:

- Setiap *build* memiliki siklus *Design & Development*, *Testing*, dan *Implementation* yang sama.
- Fungsionalitas yang ditambahkan bersifat kumulatif, artinya sistem menjadi lebih lengkap setelah setiap *build*.

Menurut [13], proses perputaran dilakukan apabila ditemukan adanya kekurangan atau kesalahan pada salah satu modul dalam sistem yang sedang dikembangkan. Perputaran ini merujuk pada kegiatan evaluasi dan revisi yang dilakukan secara berulang hingga modul tersebut memenuhi kriteria yang telah ditentukan. Apabila dalam suatu perputaran ditemukan bahwa modul yang sama masih belum memenuhi standar yang diharapkan, maka perputaran selanjutnya akan tetap difokuskan pada modul tersebut. Dengan kata lain, modul tersebut akan terus menjadi objek evaluasi dan perbaikan sampai seluruh kekurangan terselesaikan dan tidak ditemukan lagi permasalahan yang signifikan. Pendekatan ini memastikan bahwa kualitas dari setiap modul benar-benar terjamin sebelum pengembangan berlanjut ke tahap berikutnya atau ke modul yang lain.

Menurut [14], Metode pengembangan yang digunakan dalam proyek ini adalah metode iteratif, yang memungkinkan proses pengembangan dilakukan secara bertahap berdasarkan umpan balik dan evaluasi dari setiap implementasi. Dalam pendekatan ini, satu iterasi terdiri dari tiga tahap utama, yaitu desain, pengujian, dan implementasi.

Proses perputaran atau iterasi dilakukan hanya apabila ditemukan kesalahan atau kekurangan yang bersifat signifikan pada modul yang sedang dikembangkan. Salah satu indikator terjadinya perputaran adalah ketika terjadi perubahan besar, seperti revisi desain antarmuka, perubahan alur sistem, atau ketidaksesuaian pada fungsi utama modul tersebut.

Setelah proses implementasi dalam satu iterasi selesai, sistem akan diuji dan dievaluasi. Jika ditemukan masalah besar atau permintaan perubahan dari pengguna (*user*) yang berdampak langsung terhadap fungsi utama atau desain sistem, maka akan dilakukan perputaran menuju iterasi berikutnya. Revisi yang bersifat minor atau hanya menyangkut estetika visual biasanya dicatat, namun tidak selalu menjadi pemicu dilakukannya iterasi baru, kecuali apabila perubahan tersebut memengaruhi keseluruhan konsistensi antarmuka pengguna.

2.3.2 Kelebihan dan Kekurangan

Setiap metode pasti memiliki kelebihan dan kekurangannya sendiri. Berikut ini adalah kelebihan dan kekurangan dari metode iteratif:

1. **Fleksibilas**

   Metode iteratif memungkinkan perubahan kebutuhan selama proses pengembangan berlangsung. Artinya, jika ada permintaan baru dari pengguna, atau terjadi perubahan dalam prioritas bisnis, pengembang dapat menyesuaikan di iterasi berikutnya tanpa harus mengulang seluruh proyek dari awal. Ini sangat membantu dalam proyek yang dinamis.

2. **Deteksi Dini Masalah**

   Karena sistem dibangun dan diuji secara bertahap, setiap iterasi memberikan kesempatan untuk mengevaluasi hasil dan menemukan bug atau kesalahan lebih awal. Dengan mendeteksi masalah sejak awal, biaya dan dampak perbaikannya jauh lebih kecil dibanding jika ditemukan di akhir proyek.

3. **Manajemen Risiko**

   Risiko proyek, seperti ketidaksesuaian sistem dengan kebutuhan pengguna atau kesalahan teknis, bisa dievaluasi di setiap siklus iterasi. Hal ini memungkinkan tim untuk mengurangi risiko lebih awal, sebelum risiko tersebut berkembang menjadi masalah besar yang sulit diatasi.

4. **Umpan Balik Pengguna**

   Pengguna atau *stakeholder* dapat melihat hasil sementara yang dihasilkan pada akhir setiap iterasi dan memberikan masukan secara langsung. Proses ini memastikan bahwa sistem yang dikembangkan tetap selaras dengan harapan dan kebutuhan pengguna akhir. Selain itu, keterlibatan pengguna secara aktif dalam setiap tahap pengembangan turut mendorong terciptanya sistem yang lebih relevan, adaptif, dan responsif terhadap perubahan kebutuhan selama proses pengembangan berlangsung.

Adapun kekurangan dari metode tersebut adalah:

1. **Memerlukan Sumber Daya Lebih**

   Karena pengembangan dilakukan secara berulang (iteratif), maka dibutuhkan lebih banyak waktu dan tenaga untuk melakukan evaluasi, revisi, dan pengujian di tiap siklus. Ini dapat menyebabkan biaya menjadi lebih tinggi, terutama jika banyak iterasi dilakukan untuk menyempurnakan produk.

2. **Kompleksitas Manajemen**

   Mengelola banyak iterasi memerlukan koordinasi yang kuat antar tim. Setiap siklus menghasilkan versi yang harus dikelola dan dipantau, baik dari sisi teknis maupun dokumentasi. Jika tidak ditangani dengan baik, proyek bisa menjadi kacau atau kehilangan arah.

3. **Kebutuhan Dokumentasi yang Baik**

   Setiap iterasi menghasilkan versi baru, perubahan kode, dan keputusan desain. Jika tidak didokumentasikan dengan rapi dan akurat, tim pengembang bisa kehilangan jejak atas perubahan yang telah dilakukan, yang dapat menghambat proses pengembangan dan menimbulkan kesalahpahaman di antara anggota tim.

2.3.3 Konsep Dasar

Konsep-konsep dasar metode iteratif sebagai berikut:

1. **Pengembangan Bertahap**

   Sistem dikembangkan melalui serangkaian iterasi, di mana setiap iterasi menghasilkan peningkatan atau penambahan fitur pada sistem.

2. **Umpan Balik Berkelanjutan**

   Setelah setiap iterasi, sistem dievaluasi dan umpan balik dari pengguna atau pemangku kepentingan digunakan untuk perbaikan pada iterasi berikutnya.

3. **Manajemen Risiko**

   Dengan pendekatan bertahap, risiko dapat diidentifikasi dan ditangani lebih awal dalam proses pengembangan.

4. **Fleksibilitas**

   Model ini memungkinkan penyesuaian terhadap perubahan kebutuhan atau persyaratan selama proses pengembangan.

2.4 C# (C *Sharp*)

C# dikembangkan sebagai bahasa pemrograman yang dirancang untuk memanfaatkan sepenuhnya kekuatan .NET *Framework*. Bahasa ini mengadopsi dan menggabungkan berbagai elemen terbaik dari bahasa pemrograman sebelumnya, sekaligus memperkenalkan fitur-fitur baru yang lebih modern. Dengan sintaks yang menyerupai bahasa C, C#, C++, dan Java, C# menjadi lebih tidak asing bagi para pengembang yang telah memiliki pengalaman dengan bahasa tersebut.

Sebagai bahasa pemrograman utama dalam .NET *Framework*, C# memiliki cakupan penggunaan yang sangat luas. Bahasa ini memungkinkan pengembangan aplikasi berbasis Windows dengan *user interface*(UI) maupun berbasis konsol. Selain itu, dengan dukungan .NET *Framework* terhadap kode *unmanaged*, C# dapat digunakan untuk berinteraksi dengan pustaka seperti *DirectX* 8.1 dan *OpenGL*. Tidak hanya untuk pengembangan aplikasi *desktop*, C# juga mendukung pembuatan *website* dan *web service*, menjadikannya bahasa yang fleksibel dalam berbagai jenis pengembangan perangkat lunak [15].

C# (dibaca ”C-*Sharp*”) adalah bahasa pemrograman berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif .NET *Framework*. Bahasa ini dirancang untuk memudahkan pengembangan berbagai jenis aplikasi, termasuk aplikasi *desktop*, *web*, *mobile*, dan *game*. C# menggabungkan fitur-fitur dari beberapa bahasa pemrograman lain seperti C++, Java, dan *Visual Basic*, dengan penambahan beberapa penyederhanaan untuk meningkatkan efisiensi dan keamanan dalam pengembangan aplikasi [16].

2.5 *Common Language Runtime* (CLR)

Common Language Runtime (CLR) merupakan komponen inti dari Microsoft .NET *Framework* yang bertanggung jawab dalam mengelola eksekusi program berbasis .NET. CLR menyediakan lingkungan eksekusi yang aman, stabil, dan konsisten untuk aplikasi yang ditulis dalam berbagai bahasa pemrograman yang mendukung .NET. Program yang dijalankan dalam lingkungan .NET *Framework* disebut sebagai program *managed*, sementara program yang berjalan langsung

di atas sistem operasi tanpa pengelolaan dari CLR dikenal sebagai program *unmanaged*. Dalam konsepnya, peran CLR dalam .NET dapat dianalogikan seperti *Java Virtual Machine* (JVM) dalam lingkungan *Java*.

Sebagai bagian dari .NET *Framework*, CLR menangani berbagai aspek penting, seperti pengelolaan memori, manajemen thread, kompilasi kode, pemeriksaan tipe data, serta verifikasi keamanan kode selama *runtime*. Subsistem ini juga mengelola siklus hidup objek dengan *Garbage Collector* (GC) untuk mengoptimalkan penggunaan memori dan mencegah kebocoran memori. Selain itu, CLR mendukung interoperabilitas dengan kode *unmanaged*, memungkinkan aplikasi .NET untuk berkomunikasi dengan komponen lain seperti COM (*Component Object* Model) dan pustaka *native*.

CLR merupakan implementasi dari *Common Language Infrastructure* (CLI), yang telah distandarisasi oleh ECMA, menjadikannya fondasi utama bagi eksekusi aplikasi .NET. Dengan berbagai layanan yang ditawarkannya, CLR memastikan eksekusi program berjalan dengan efisien, aman, dan stabil di berbagai *platform* yang mendukung .NET *Framework*. [9].

CLR merupakan komponen inti dari Microsoft .NET Framework yang bertanggung jawab untuk mengelola eksekusi program yang ditulis dalam berbagai bahasa pemrograman yang mendukung .NET. CLR menyediakan lingkungan eksekusi yang aman, stabil, dan konsisten untuk aplikasi .NET. Berikut adalah penjelasan detail

2.5.1 Manajemen Memori

Sistem manajemen memori pada CLR menggunakan *Garbage Collector* (GC) untuk mengalokasikan dan membebaskan memori secara otomatis. GC berperan dalam mengelola objek yang masih digunakan serta menghapus objek yang tidak lagi diperlukan guna mencegah kebocoran memori. Jurnal yang relevan mengenai manajemen memori dalam CLR antara lain *"A Study of Garbage Collection Algorithms in .NET CLR"* oleh Smith et al. (2018) dan *"Memory Management in .NET Framework"* (Johnson, 2017).

2.5.2 Keamanan

CLR memiliki mekanisme keamanan yang kuat, seperti *Code Access Security* (CAS) dan *Role-Based Security*, yang memastikan kode berbahaya tidak

dapat mengakses sumber daya sistem tanpa izin. Keamanan ini melibatkan pembatasan akses kode, verifikasi keamanan, serta pengelolaan hak akses berdasarkan peran pengguna. Beberapa referensi terkait mencakup *"Security Mechanisms in .NET CLR"* oleh Johnson (2019) dan *"Code Access Security in .NET Framework"* Microsoft *Research* (2020).

2.5.3 Manajemen Eksekusi

CLR mengelola eksekusi kode dengan menerjemahkan kode sumber dalam bahasa .NET menjadi *Intermediate Language* (IL), yang kemudian dikompilasi secara *just-in-time* (JIT) ke dalam kode mesin. Proses ini memungkinkan optimasi eksekusi dan peningkatan kinerja aplikasi. Referensi yang membahas topik ini meliputi *"Just-In-Time Compilation in .NET CLR"* (Lee et al., 2020) dan *"Performance Optimization in .NET CLR"* (Brown, 2016).

2.5.4 Manajemen *Exception*

CLR menyediakan sistem penanganan exception yang konsisten bagi semua bahasa dalam ekosistem .NET. Sistem ini mendeteksi kesalahan saat *runtime*, memberikan informasi debugging melalui *stack trace*, serta memungkinkan penanganan *exception* yang lebih spesifik. Beberapa referensi yang membahas topik ini adalah *"Exception Handling in .NET CLR"* oleh (Brown, 2017) dan *"Debugging and Exception Management in .NET"* (Taylor, 2018).

2.5.5 Interoperabilitas

CLR mendukung interoperabilitas dengan kode yang dikembangkan dalam bahasa lain, termasuk *Component Object Model* (COM) dan kode *unmanaged*. Interoperabilitas ini memungkinkan aplikasi .NET untuk mengakses dan menggunakan kode dari *platform* lain melalui mekanisme seperti *Platform Invocation Services* (P/Invoke). Jurnal yang relevan dengan topik ini mencakup *"Interoperability in .NET CLR"* oleh (Martinez, 2021) dan *"COM Interop in .NET Framework"* (Harris, 2019).

2.5.6 Manajemen *Thread*

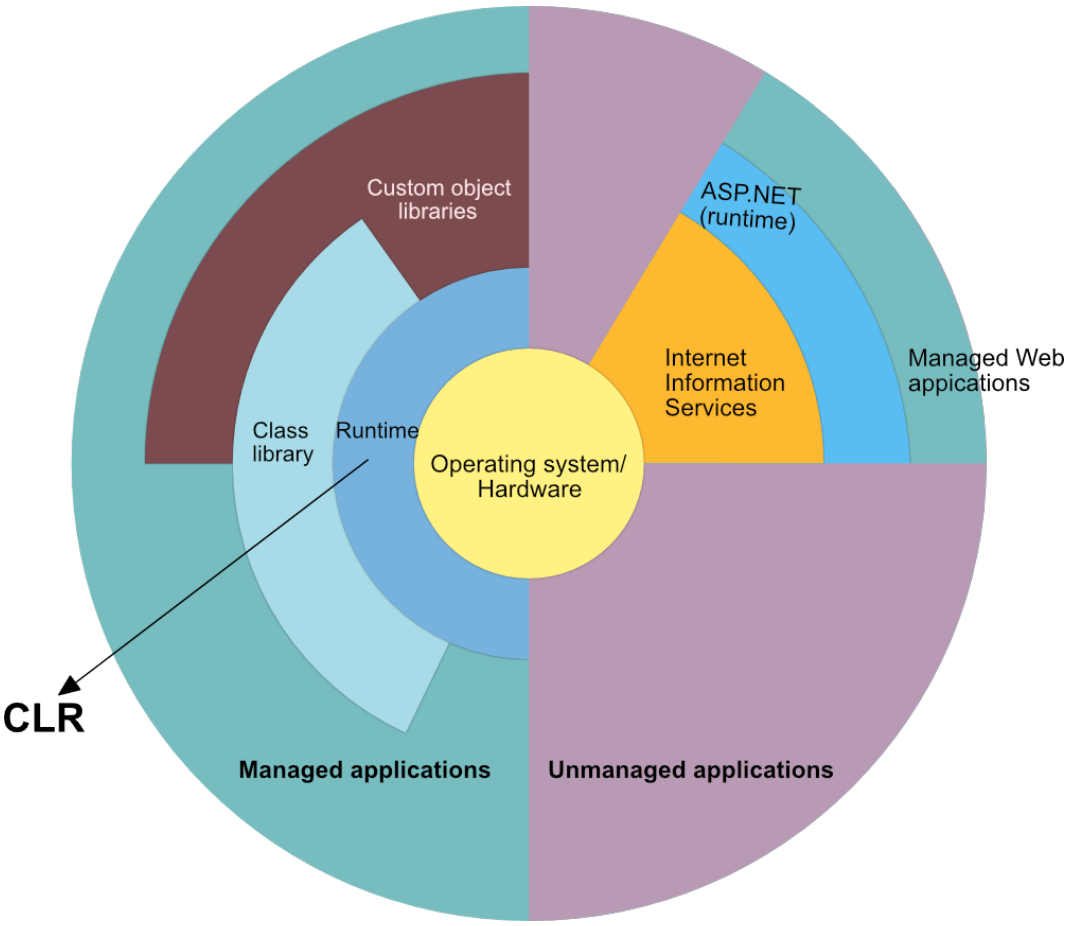
CLR menyediakan dukungan untuk pemrograman *multithreading*, memungkinkan aplikasi menjalankan banyak tugas secara bersamaan. Fitur ini mencakup pembuatan dan pengelolaan *thread* serta mekanisme sinkronisasi untuk menghindari *race condition*. Beberapa referensi yang membahas manajemen *thread* dalam CLR adalah *"Multithreading in .NET CLR"* (Wilson, 2016) dan *"Concurrency in .NET Framework"* (Anderson, 2018).

2.5.7 Manajemen Metadata

Metadata dalam CLR digunakan untuk mendeskripsikan tipe data, metode, dan atribut dalam aplikasi .NET. Metadata ini tersimpan dalam *assembly* (*file* .dll atau .exe) dan dimanfaatkan untuk refleksi (*reflection*) serta pengelolaan eksekusi kode. Beberapa referensi terkait adalah *"Metadata and Reflection in .NET CLR"* (Taylor 2018) dan *"Assembly and Metadata in .NET Framework"* (Lee, 2017).

2.5.8 *Base Class Library* (BCL)

CLR menyertakan *Base Class Library* (BCL), yaitu kumpulan kelas dan metode standar untuk menangani berbagai tugas umum, seperti manipulasi string, akses file, dan komunikasi jaringan. BCL memastikan konsistensi dan kemudahan penggunaan bagi pengembang dalam berbagai bahasa .NET. Referensi mengenai BCL meliputi *"Base Class Library in .NET Framework"* (Harris, 2020) dan *"API Design in .NET Framework"* (Smith, 2019).



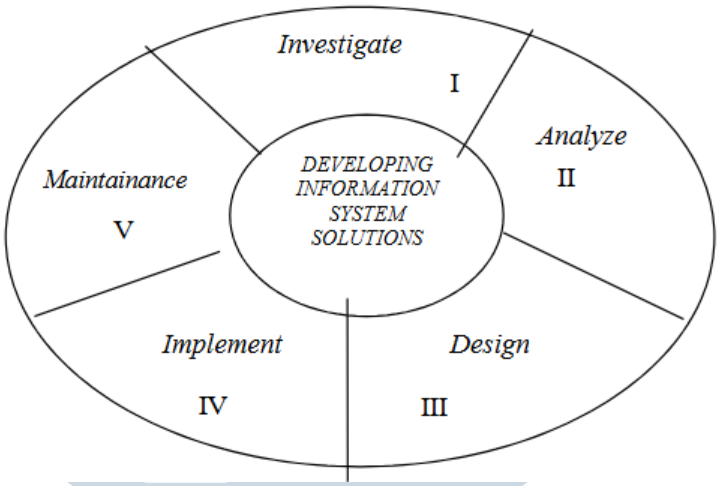
Gambar 2.2. CLR

Gambar 2.2 menggambarkan arsitektur eksekusi aplikasi dalam lingkungan .NET *Framework*, yang memperlihatkan hubungan antara sistem operasi/*hardware* dengan berbagai jenis aplikasi dan layanan. Pada inti diagram terdapat sistem operasi dan perangkat keras yang menjadi dasar dari seluruh proses eksekusi. Di atasnya terdapat lapisan *Common Language Runtime* (CLR), yang merupakan komponen utama dari .NET *Framework* yang berfungsi untuk menjalankan dan mengelola aplikasi secara otomatis, termasuk pengelolaan memori, keamanan, dan pengumpulan sampah (garbage collection). Aplikasi yang dijalankan di bawah pengawasan CLR disebut sebagai *managed applications*, yang mencakup pustaka kelas standar (*class library*), pustaka objek kustom (*custom object libraries*), serta *runtime* aplikasi. Selain itu, terdapat juga *managed web applications* seperti ASP.NET yang berjalan di atas layanan *Internet Information Services* (IIS), yaitu server *web* milik Microsoft yang mendukung eksekusi aplikasi *web* berbasis .NET. Di sisi lain, terdapat pula *unmanaged applications*, yaitu aplikasi yang tidak

dikelola oleh CLR dan langsung berinteraksi dengan sistem operasi, seperti aplikasi *native* yang ditulis menggunakan bahasa pemrograman lama seperti C atau C++. Secara keseluruhan, diagram ini menunjukkan bagaimana CLR memainkan peran penting dalam menyediakan lingkungan eksekusi yang aman dan terkelola untuk berbagai jenis aplikasi, terutama yang berbasis .NET.

2.6 SDLC (*System Development Life Cycle*)

System Development Life Cycle adalah representasi dari suatu proses dalam merancang sistem yang terus berputar layaknya roda, melalui serangkaian tahapan yang mencakup *investigate*, *analyze*, desain, implementasi, serta perawatan [17].



Gambar 2.3. *Sistem Development Life Cycle* dari iteratif

Metode *System Development Life Cycle* (SDLC) memiliki berbagai macam pendekatan, seperti *waterfall*, *scrum*, *iterative*, *DevOps*, *spiral*, *agile*, dan *V-Model*. Setiap metode SDLC memiliki kelebihan dan kekurangannya masing-masing. Meskipun terdapat berbagai pendekatan, SDLC pada dasarnya tetap memiliki tahapan utama, yaitu perencanaan (*planning*), analisis, desain, pengembangan, uji coba, implementasi, dan pemeliharaan.

Pada proyek ini, metode SDLC yang digunakan adalah *iterative*. Model ini diterapkan melalui proses berulang, di mana pengembangan perangkat lunak dibagi menjadi bagian-bagian kecil yang disebut iterasi. Dengan pendekatan ini, tim pengembang dapat menghasilkan komponen perangkat lunak yang siap diuji serta memperoleh umpan balik dari pengguna secara berkala, sehingga setiap iterasi dapat disesuaikan dengan kebutuhan yang berkembang.

Setiap iterasi dalam model ini menyerupai proses *mini-Waterfall*, di mana setiap tahap memberikan masukan yang berharga untuk desain tahap berikutnya. Proses ini melibatkan perulangan dalam desain, pengkodean, pengujian, dan evaluasi untuk setiap bagian atau iterasi proyek. Pada akhir setiap iterasi atau setelah beberapa iterasi, produk perangkat lunak yang dihasilkan dapat langsung

digunakan dalam lingkungan produksi. Hal ini memungkinkan perangkat lunak mengalami peningkatan dan penyempurnaan secara bertahap berdasarkan masukan dari pengguna serta pemangku kepentingan lainnya. Dengan pendekatan ini, perubahan kebutuhan dapat lebih mudah diakomodasi, sehingga memastikan bahwa hasil akhir sesuai dengan ekspektasi pengguna [18].

2.7 Skala Likert

Skala Likert pertama kali diperkenalkan oleh Rensis Likert pada tahun 1932 sebagai metode untuk mengukur sikap masyarakat. Skala ini bersifat *ordinal*, yang memungkinkan penyusunan peringkat tanpa memberikan informasi mengenai sejauh mana perbedaan sikap antarresponden. Dalam implementasinya, instrumen skala Likert terdiri dari pernyataan yang memiliki tingkatan respons dari sangat positif hingga sangat negatif. Pilihan jawaban yang umum digunakan meliputi *sangat setuju* (SS), *setuju* (S), *ragu-ragu* (R), *tidak setuju* (TS), dan *sangat tidak setuju* (STS). Urutan pilihan ini dapat disesuaikan dengan kebutuhan penelitian.

Dalam menyusun skala sikap berbasis Likert, langkah pertama yang dilakukan adalah merancang sejumlah pernyataan yang dapat mengukur sikap individu terhadap suatu objek tertentu. Setiap pernyataan dalam skala ini dapat dikategorikan sebagai mendukung (*favourable*) atau menolak (*unfavourable*) terhadap objek yang sedang diukur. Namun, dalam aspek psikologisnya, tidak dapat diketahui secara pasti seberapa besar perbedaan antara setiap respons yang diberikan responden.

Responden diberikan kebebasan untuk memilih salah satu dari lima kategori jawaban yang telah disediakan, yaitu sangat setuju, setuju, Netral, tidak ditentukan (*undecided*), tidak setuju, dan sangat tidak setuju. Responden yang memiliki sikap sangat positif terhadap suatu pernyataan akan cenderung memilih “sangat setuju” jika pernyataan tersebut bersifat positif [19].

Sebagai contoh, dalam pernyataan ”Apakah Anda setuju bahwa *parent portal* yang dibuat cukup membantu?”, bobot skor diberikan berdasarkan pilihan jawaban sebagai berikut: sangat setuju (SS) = 5, setuju (S) = 4, Netral = 3, tidak setuju (TS) = 2, dan sangat tidak setuju (STS) = 1. Jika terdapat 100 responden dengan distribusi jawaban sebagai berikut:

Tabel 2.1. Perhitungan Skor Responden

Pilihan	Jumlah Responden	Skor
SS	30	$30 \times 5 = 150$
S	30	$30 \times 4 = 120$
KS	5	$5 \times 3 = 15$
TS	20	$20 \times 2 = 40$
STS	15	$15 \times 1 = 15$
Total Skor		**360**

Skor maksimum dihitung sebagai 100 × 5 = 500, sedangkan skor minimum adalah 100 × 1 = 100. Indeks persentase dapat dihitung dengan rumus berikut:

$$\left(\frac{360}{500}\right) \times 100 = 72\% \tag{2.1}$$

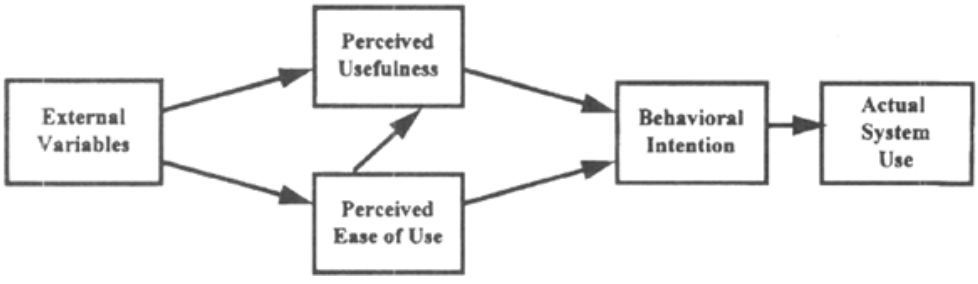
Berdasarkan nilai indeks ini, hasilnya dapat dikategorikan sesuai dengan skala penilaian berikut:

Tabel 2.2. Kategori Berdasarkan Persentase

Persentase	**Kategori**
0% – 19.99%	Sangat Tidak Setuju
20% – 39.99%	Tidak Setuju
40% – 59.99%	Kurang Setuju
60% – 79.99%	Setuju
80% – 100%	Sangat Setuju

Karena indeks yang diperoleh sebesar 72%, maka dapat disimpulkan bahwa mayoritas responden setuju bahwa *web parent portal* yang dibuat cukup membantu orang tua dalam memantau dan mengawasi anak-anak [20].

2.8 TAM (*Technology Acceptance Model)*



Gambar 2.4. Technology Acceptance Model (Davis, et al., 1989)

Gambar 2.4 merupakan model yang dikembangkan oleh Fred D. Davis pada tahun 1989 untuk menjelaskan bagaimana pengguna menerima dan menggunakan suatu teknologi informasi. Model ini didasarkan pada teori psikologi yang menyoroti faktor-faktor yang mempengaruhi penerimaan teknologi oleh pengguna. TAM berfokus pada dua variabel utama, yaitu:

1. ***Perceived Usefulness*** **(PU)**: Sejauh mana seseorang percaya bahwa menggunakan teknologi tertentu akan meningkatkan kinerjanya.

2. ***Perceived Ease of Use*** **(PEOU)**: Sejauh mana seseorang percaya bahwa teknologi tersebut mudah digunakan [21].

Menurut Davis (1989), dua komponen utama dalam model *Technology Acceptance Model* (TAM), yaitu *Perceived Usefulness* dan *Perceived Ease of Use*, berpengaruh secara langsung terhadap sikap dalam menerima dan menggunakan teknologi. Sikap ini kemudian menjadi faktor penentu dalam keputusan untuk mengadopsi atau menolak suatu sistem teknologi. Model ini berakar dari *Theory of Reasoned Action* (TRA) yang dikembangkan oleh Ajzen dan Fishbein (1980), yang menjelaskan bahwa sikap dan niat terhadap suatu perilaku dipengaruhi oleh persepsi serta reaksi yang muncul sebelumnya.

TAM kemudian disempurnakan oleh Venkatesh dan Davis (2000) menjadi TAM2, dengan memperluas kerangka kerja awal melalui penambahan variabel seperti *social influence* dan *cognitive instrumental processes*. Pengembangan ini bertujuan untuk memberikan pemahaman yang lebih menyeluruh mengenai alasan di balik penerimaan terhadap teknologi informasi.

Dalam kerangka TAM, *behavioral intention to use* dipengaruhi oleh dua keyakinan utama: pertama, *perceived usefulness*, yaitu sejauh mana penggunaan sistem dianggap dapat meningkatkan kinerja; dan kedua, *perceived ease of use*, yaitu sejauh mana sistem dianggap mudah digunakan. Di samping itu, variabel eksternal seperti karakteristik sistem, proses pengembangan, dan pelatihan turut memengaruhi niat penggunaan melalui mediasi kedua konstruk utama tersebut. Bahkan, TAM juga menyatakan bahwa *perceived usefulness* turut dipengaruhi secara langsung oleh *perceived ease of use*.

Venkatesh dan Davis menegaskan bahwa TAM merupakan model yang kuat dan andal dalam menjelaskan perilaku pengguna terhadap adopsi sistem informasi baru. Oleh karena itu, TAM banyak digunakan dalam penelitian yang berkaitan dengan penerimaan teknologi, khususnya dalam konteks sistem informasi. [22]

TAM digunakan untuk menganalisis sejauh mana penerimaan dan kemauan menggunakan suatu teknologi, dalam hal ini aplikasi *Parent Portal*. Melalui pendekatan ini, tingkat kepuasan terhadap fitur-fitur yang tersedia serta niat untuk terus menggunakan aplikasi di masa mendatang dapat diukur. TAM mengevaluasi aspek seperti kemudahan penggunaan (*Perceived Ease of Use*) dan kebermanfaatan (*Perceived Usefulness*) yang secara langsung memengaruhi sikap terhadap sistem. Dengan penerapan TAM, efektivitas aplikasi dalam memenuhi kebutuhan dapat dipahami secara lebih mendalam, sekaligus membuka peluang untuk meningkatkan adopsi teknologi secara berkelanjutan. Hasil analisis ini juga menjadi dasar bagi pengembangan lanjutan agar sistem semakin sesuai dengan ekspektasi dan kebiasaan pengguna.

2.8.1 Komponen TAM

TAM memiliki beberapa komponen utama yang membentuk model penerimaan teknologi:

- ***Attitude Toward Using*** **(ATU)**: Sikap pengguna terhadap penggunaan teknologi.
- ***Behavioral Intention to Use*** **(BIU)**: Niat pengguna untuk menggunakan teknologi.
- ***Actual System Use*** **(AU)**: Penggunaan aktual dari sistem teknologi.

2.8.2 Pengaruh dan Pengembangan TAM

TAM telah digunakan secara luas dalam berbagai bidang penelitian, termasuk *e-learning*, sistem informasi manajemen, dan teknologi kesehatan. Beberapa penelitian menunjukkan bahwa model ini dapat membantu dalam merancang sistem teknologi yang lebih mudah diterima oleh pengguna [23].