

BAB II

KAJIAN LITERATUR

2.1 Basis Data

Basis data memiliki peran yang dampaknya sangat besar dalam meningkatkan kinerja bisnis dari berbagai sisi. Dimulai dari meningkatkan efisiensi operasional hingga memungkinkan pengambilan keputusan berbasis data yang tepat, efisien dalam operasional, dan pertumbuhan strategis. Perusahaan dapat membuka peluang baru untuk pengembangan dan mencapai kesuksesan secara konsisten. Jadi, basis data menyediakan alat dan wawasan yang diperlukan dalam pengambilan keputusan penting [5].

Pentingnya efektifitas basis data akan terus meningkat dan menjadi area yang krusial untuk investasi dan fokus. Hal ini disebabkan karena bisnis terus menavigasi lanskap yang semakin didorong oleh data. Dengan kata lain, keputusan dan operasi bisnis terus menghadapi perubahan atau tantangan karena industri masa kini sangat bergantung pada data [5]. Dalam konteks operasional Galaxy Café, ketergantungan ini menuntut adanya arsitektur basis data terpusat guna mengintegrasikan seluruh rekaman data back-office dari berbagai cabang. Implementasi basis data ini menjadi fondasi utama agar pihak manajemen dapat mengelola inventaris, asset, dan laporan keuangan secara konsisten dalam satu ekosistem yang utuh.

2.1.1 Sistem Penyimpanan Data Terpusat

Sistem penyimpanan data terpusat merupakan model pengelolaan basis data di mana seluruh sumber daya data organisasi ditempatkan pada satu lokasi server utama. Dalam arsitektur ini, semua akses data dari berbagai terminal atau cabang akan bermuara ke satu titik penyimpanan yang sama untuk memudahkan pengelolaan. Konsep ini menjadi fundamental dalam merancang sistem yang membutuhkan sinkronisasi tinggi antarpengguna yang tersebar secara luas [6].

Penerapan basis data terpusat memiliki keunggulan utama dalam hal pengendalian logika struktur data dan konsistensi informasi. Dengan memusatkan penyimpanan, aturan bisnis dan validasi data dapat dikelola secara seragam dari satu tempat, yang efektif untuk meminimalisir kesalahan pencatatan. Metode ini memastikan informasi yang diakses oleh setiap pengguna adalah data yang aktual dan valid, serta memudahkan proses penanganan masalah (*troubleshooting*) karena hanya berfokus pada satu server [6].

Sistem ini relevan untuk diterapkan pada perusahaan yang memiliki lebih dari satu lokasi operasional seperti pada Kafe Galaxy. Data dari berbagai lokasi usaha yang berbeda dapat diintegrasikan pada satu lokasi server pusat untuk memudahkan pengelolaan dan analisis [6]. Dengan demikian, proses pengawasan yang dilakukan oleh manajemen pusat dapat lebih terfokus dan efisien.

2.1.2 Basis Data Relasional SQL

SQL merupakan bahasa standar yang digunakan untuk berinteraksi dengan sistem manajemen basis data relasional. SQL memungkinkan pengguna untuk mendefinisikan struktur data, memanipulasi data (menambah, mengubah, menghapus), serta melakukan kueri pengambilan data yang kompleks dari satu atau lebih data secara bersamaan [7]. Secara garis besar, perintah SQL terbagi menjadi beberapa kategori utama, yaitu *Data Definition Language* (DDL) untuk mengelola struktur table, *Data Manipulation Language* (DML) untuk mengelola isi data, dan *Data Control Language* (DCL) untuk mengatur hak akses atau keamanan data.

Keunggulan utama SQL terletak pada efisiensinya dalam menangani relasi antardata yang rumit melalui perintah *join*. Hal ini memungkinkan sistem untuk menghubungkan berbagai entitas data yang berbeda, seperti data stok masuk dengan data asset tetap tanpa harus menyimpan informasi yang sama berulang kali. Selain itu, SQL memiliki skalabilitas yang baik dan didukung oleh hampir semua platform basis data modern, menjadikannya bahasa yang sangat andal untuk menjamin integritas dan konsistensi informasi dalam sebuah sistem informasi manajemen [7].

Selain sebagai media manipulasi data, aspek keamanan dalam SQL merupakan komponen fundamental dalam arsitektur basis data. Penggunaan perintah *Data Control Language* (DCL) memungkinkan administrator untuk melakukan pengawasan dan audit terhadap setiap perintah yang dijalankan pada basis data. Praktik ini memastikan bahwa setiap interaksi, mulai dari pembacaan hingga perubahan data terekam dengan jelas dalam log sistem. Hal ini krusial untuk mendeteksi aktivitas yang tidak sah serta menjaga kepatuhan sistem terhadap standar integritas data perusahaan. Dalam konteks sistem manajemen *back-office*, mekanisme ini menjadi instrumen untuk membatasi akses sensitif hanya kepada pengguna yang memiliki otoritas resmi [8].

Meskipun saat ini berkembang teknologi database NoSQL yang tidak memerlukan skema tetap, SQL dipilih dalam penelitian ini karena keunggulannya dalam menjaga integritas referensial data dalam operasional Galaxy Café. Penggunaan SQL menjamin

bahwa setiap manipulasi data pada sistem *back-office* tetap mengikuti aturan bisnis yang ketat, serta memungkinkan implementasi audit aktivitas yang konsisten untuk meminimalisir risiko terjadinya data yang tidak valid atau akses ilegal antarcabang. Kelebihan lainnya adalah banyak pilihan serta kemudahan *hosting* dan komunitas yang lebih luas untuk saat ini.

2.1.3 Normalisasi Data

Normalisasi data merupakan komponen penting dalam *data tuning* atau penataan struktur basis data. Perannya yaitu mengurangi duplikasi data dan meningkatkan konsistensi data dengan membuang redundansi pada basis data. Seiring dengan bertambahnya volume dan lalu-lintas data, maka kecepatan aplikasi, keteguhan, serta keamanan basis data juga perlu dipertahankan. Salah satu metodenya adalah menerapkan normalisasi data [9].

Cara kerja normalisasi data adalah dengan membagi tabel besar ke beberapa table kecil. Langkah selanjutnya adalah menerapkan *foreign keys* untuk menjamin integritas referensial, menghindari grup yang berulang, serta menghindari nilai null. Peningkatan kinerja *query* dan penurunan biaya penyimpanan juga dapat dilakukan dengan mengurangi redundansi data. Dengan demikian, integrasi data dapat ditingkatkan dan lebih terjamin [9]. Pada sistem *back-office* Galaxy Café, normalisasi data diterapkan untuk menstrukturkan entitas seperti data master inventaris dan detail pencatatan multi-cabang. Hal ini memastikan proses pemanggilan laporan berjalan efisien, akurat, dan terhindar dari duplikasi data.

2.2 Arsitektur Sistem

Arsitektur perangkat lunak merupakan struktur fundamental yang menggambarkan bagaimana elemen-elemen dalam sebuah sistem disusun, berinteraksi, dan diintegrasikan untuk memenuhi kebutuhan fungsional bisnis. Pemilihan gaya arsitektur yang tepat menjadi langkah krusial dalam tahap perancangan sistem informasi, karena arsitektur yang baik akan menjamin kualitas perangkat lunak dari sisi kinerja, keamanan, serta kemudahan dalam pemeliharaan (*maintainability*) jangka panjang [10].

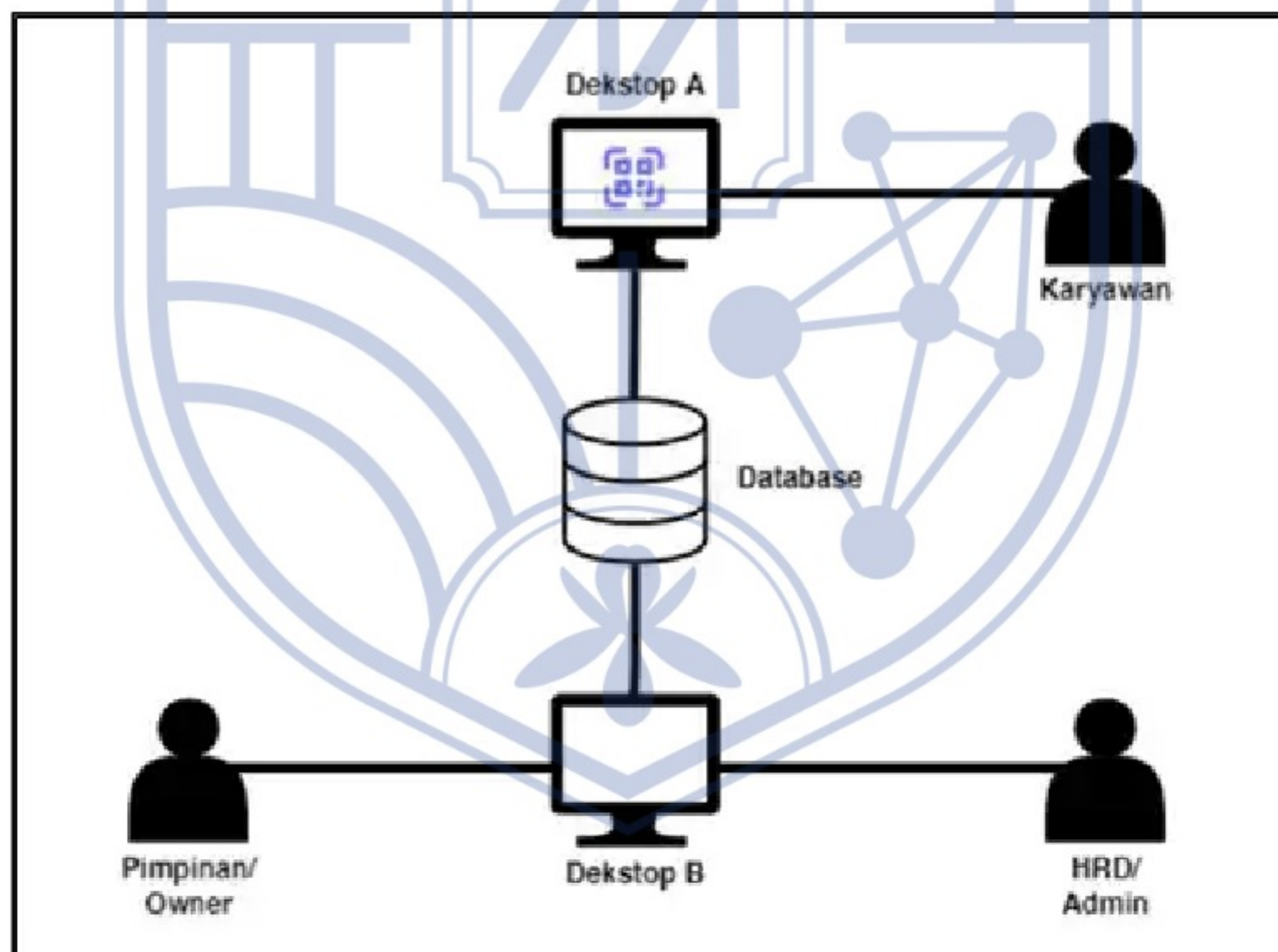
Tahapan dalam merancang arsitektur sistem meliputi pemodelan struktur sistem, perancangan basis data, desain antarmuka pengguna, dan penentuan alur logika aplikasi. Adapun tahap dalam konteks implementasi terdiri dari pemilihan teknologi, pemilihan bahasa pemrograman, dan desain tata letak aplikasi. Hal ini perlu dilakukan untuk merancang solusi yang sesuai dengan kebutuhan, memastikan keamanan, dan memastikan kinerja sistem [10]. Dalam pengembangan sistem *back-office* Galaxy Café, arsitektur

terpusat berbasis web diterapkan untuk menjamin stabilitas performa, kemudahan pemeliharaan, dan efisiensi alur informasi inventaris antar lokasi.

2.2.1 Arsitektur Komunikasi *Client-Server*

Arsitektur *Client-Server* merupakan model pengembangan perangkat lunak terdistribusi yang memisahkan fungsi sistem menjadi dua entitas utama, yaitu penyedia layanan (*server*) dan pengguna layanan (*client*). Dalam model ini, *server* bertugas sebagai pusat penyimpanan data dan pemrosesan logika bisnis, sementara *client* berfungsi sebagai antarmuka pengguna untuk berinteraksi dengan sistem. Pemisahan tugas ini bertujuan untuk mengoptimalkan kinerja jaringan dan pemrosesan data [11].

Gambar 2.1 adalah contoh arsitektur komunikasi *Client-Server* sederhana pada sistem presensi perusahaan. Database berperan sebagai server yang menyimpan data, *Desktop A* dan *B* sebagai *client*, lalu karyawan, pimpinan, dan HRD sebagai pengguna. Pengguna berinteraksi dengan *client* untuk tindakan membaca (*read*) atau menulis (*write*) data ke server. Polanya adalah dari *client*, server, lalu *client* [11].



Gambar 2.1 Arsitektur Komunikasi *Client-Server*

Penerapan arsitektur ini sangat efektif untuk menjaga validitas data dalam lingkungan operasional perusahaan yang memiliki banyak pengguna. Dengan memusatkan basis data di sisi *server*, risiko manipulasi data di sisi *client* dapat diminimalisir secara signifikan. Selain itu, arsitektur ini memudahkan proses pemeliharaan sistem, karena perubahan logika program cukup dilakukan pada server pusat tanpa perlu melakukan instalasi ulang pada setiap komputer pengguna [11].

2.2.2 Arsitektur Aplikasi Monolitik

Arsitektur monolitik adalah pendekatan desain perangkat lunak di mana seluruh komponen fungsional seperti antarmuka pengguna, logika bisnis, dan akses basis data dibangun menjadi satu kesatuan unit program tunggal. Dalam pendekatan ini, semua modul saling terkait erat dan dijalankan dalam satu lingkungan eksekusi yang sama, sehingga komunikasi antarkomponen berjalan sangat cepat karena berada dalam satu memori [10].

Implementasi arsitektur monolitik dinilai sangat efektif untuk rancang bangun sistem informasi yang membutuhkan integrasi kuat dan konsistensi tinggi. Keuntungan utama dari pendekatan ini adalah kemudahan dalam tahap pengembangan awal dan proses *deployment*. Karena aplikasi tidak terpecah menjadi layanan-layanan kecil yang rumit, arsitektur monolitik menjadi solusi yang efisien untuk sistem yang mengutamakan kestabilan fungsi terpusat dibandingkan kompleksitas distribusi layanan [10].

Meskipun arsitektur *microservices* unggul dalam hal skalabilitas untuk aplikasi berskala raksasa, penggunaannya membutuhkan infrastruktur jaringan yang rumit dan biaya operasional yang tinggi. Mengingat sistem informasi manajemen Galaxy Café ini masih berada pada ruang lingkup pemrosesan data internal (*back-office*), maka arsitektur monolitik secara eksplisit dipilih. Pendekatan monolitik dinilai paling ideal dan efisien untuk kasus ini karena mempermudah proses pengembangan, *debugging*, dan *deployment* pada suatu server.

2.3 Object-Oriented Programming (OOP)

Dalam pengembangan perangkat lunak, prinsip OOP menyelesaikan masalah dengan meniru kerangka berpikir manusia dalam kehidupan nyata dengan menggunakan artefak perangkat lunak. Kemampuan ini memungkinkan desain perangkat lunak yang rumit. Konsep ini bukan hanya gaya pemrograman, namun juga merupakan sebuah metodologi untuk menyimpulkan dari konsep umum ke konsep khusus dan menginduksi dari konsep khusus ke konsep umum [12].

Konsep *object-oriented* telah terbukti dalam membantu proses pemecahan masalah, baik dalam bidang edukasi dan industri perangkat lunak. Selama satu dekade terakhir, desain dan analisis berbasis *object-oriented* mendominasi paradigma pengembangan perangkat lunak. Menghubungkan pengalaman dan objek nyata dengan konsep OOP merupakan persyaratan utama dalam identifikasi, pernyataan, dan pemecahan masalah yang kompleks [12].

2.3.1 Penerapan OOP

Prinsip inti OOP terdiri dari *encapsulation*, *inheritance*, *polymorphism*, *abstraction*. *Encapsulation* melindungi data dari perubahan tidak disengaja dengan membuat data menjadi privat. Hal ini meningkatkan keandalan integrasi data ke sistem dan agar modifikasi dan *errors* lebih mudah dikelola. *Inheritance* memudahkan *reusability*, yaitu menggunakan kembali fungsionalitas dibandingkan membuat dari awal dengan mekanisme antar-*subclass* dan *superclass*. *Polymorphism* menjaga ketahanan dan mempermudah adaptasi kode untuk masa depan dengan cara merancang ulang subclass tanpa memodifikasi kode yang memanggilnya. *Abstraction* mengurangi kompleksitas sistem dengan cara hanya menampilkan aspek penting dan menyembunyikan sisanya. Dengan demikian, pengembang dapat mengontrol kompleksitas dengan hanya fokus pada atribut dan operasi yang penting [13].

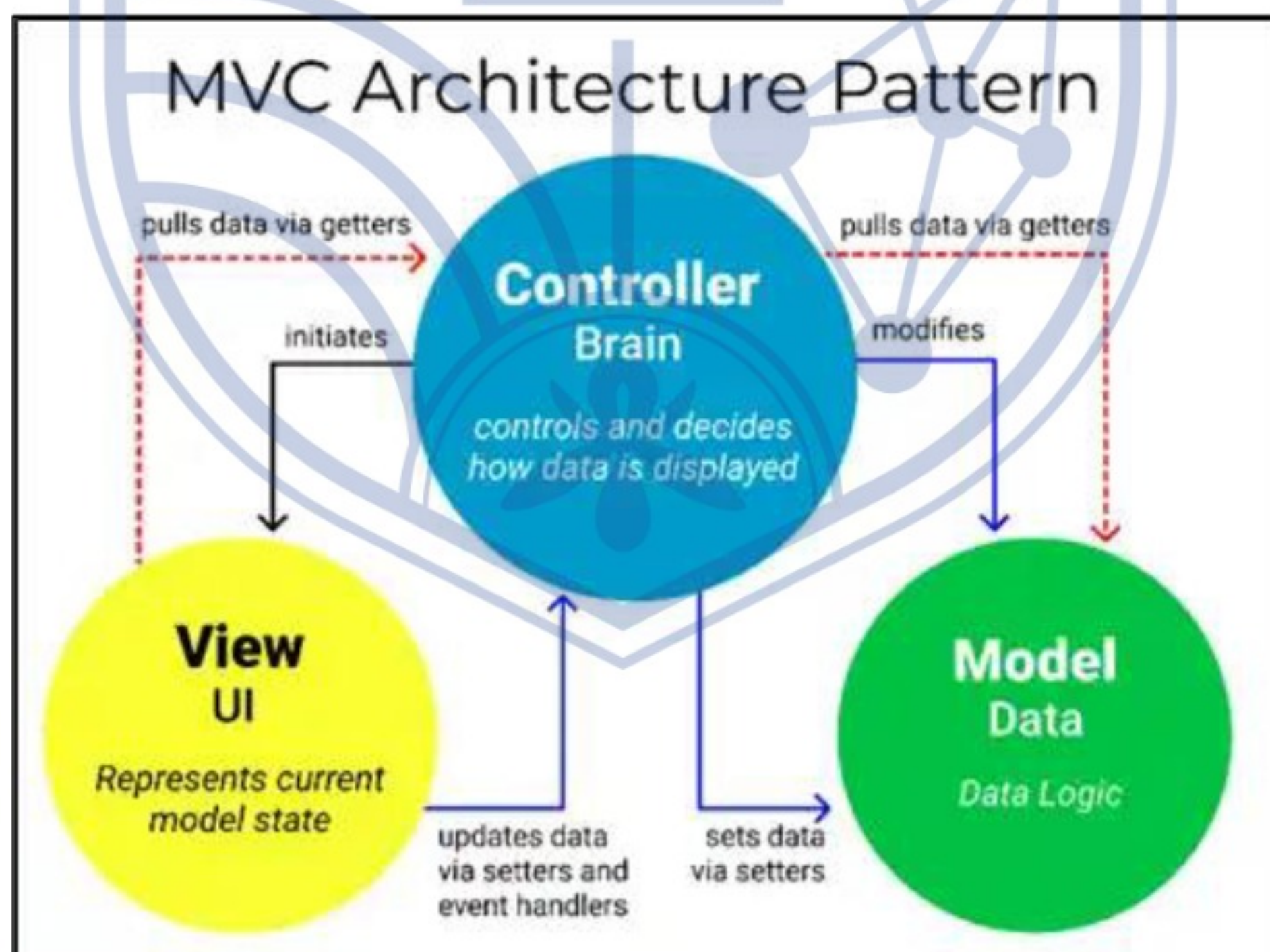
Selain itu, juga terdapat pola desain seperti *singleton*, *factory*, dan *observer* yang sangat penting untuk efisiensi penggunaan sumber daya, pembuatan objek, ketergantungan, serta membuat sistem kompleks lebih portable. Pola *singleton* memastikan bahwa satu kelas hanya mengandung satu *instance* dan satu akses poin ke *instance* tersebut. Hal ini agar terhindar dari masalah pada *shared resource* yaitu *over-usage*, koneksi ke database, serta pengaturan konfigurasi. Pola *factory* mengontrol produksi objek dari sebuah antarmuka dan membiarkan *subclass* menentukan pembentukannya sendiri. Hal ini meningkatkan fleksibilitas karena dapat menentukan jenis pembuatan baru tanpa mengubah keseluruhan kode aplikasi, meminimalkan ketergantungan, serta ideal untuk perubahan yang cukup sering dan pemeliharaan selanjutnya. Pola *observer* meningkatkan interaksi komunikasi dependensi antara satu objek dan lainnya. Semua *observer* akan diberi tahu secara otomatis setiap perubahan subjek. Pola ini ideal untuk sistem kinetik seperti sistem respon terhadap pengguna atau visualisasi yang mengharuskan komponen merespons perubahan keadaan acara [13].

OOP mendorong modularitas dan fleksibilitas untuk pembuatannya, serta kemudahan pemeliharaan program. Hal ini menguntungkan untuk menciptakan sistem perangkat lunak yang besar. Namun, ditemukan juga kelemahan OOP, yaitu ketika berurusan dengan data yang dapat diubah dan transisi keadaan dalam sistem besar [13].

Oleh karena itu, pendekatan pengembangan secara *hybrid* dapat digunakan sebagai opsi tambahan untuk OOP. Hal ini mencakup integrasi pemrograman fungsional ke dalam proses pengembangan. Kombinasi ini mendukung pemrosesan informasi yang lebih intensif dan stabil dalam pengembangan aplikasi. Jadi, OOP masih valid dengan paradigma lain dalam struktur perangkat lunak modern [13].

2.3.2 Pola Arsitektural MVC (*Model-View-Controller*)

Salah satu pola arsitektural yang menerapkan prinsip OOP adalah MVC. MVC membagi aplikasi menjadi tiga komponen utama, yaitu *model* untuk mengelola data dan logika bisnis, *view* untuk menangani tampilan dan antarmuka pengguna, serta *controller* sebagai pengatur interaksi antara *model* dan *view*. Pendekatan MVC membantu penyusunan kode program agar lebih terstruktur. Hal ini meningkatkan efisiensi proses pengembangan serta mempermudah pemeliharaan kode program yang berkelanjutan secara signifikan. Gambar 2.2 merupakan visualisasi pola arsitektural MVC [14].



Gambar 2.2 Pola Arsitektural MVC

Memisahkan logika aplikasi menjadi model, tampilan, dan pengontrol terbukti memberikan keuntungan signifikan dalam pengembangan website menggunakan *framework* Laravel. Pembuatan tampilan antarmuka pengguna dibantu dengan *Blade templating engine*, *controller* memproses input pengguna yang diolah melalui model, kemudian hasilnya kembali dikirimkan ke tampilan. Selain itu, terdapat struktur direktori yang rapi dan fitur bawaan yang mendukung manajemen basis data, *routing*, serta pengujian. Laravel memberikan dukungan yang lengkap dalam penerapan pola arsitektural MVC pada pembuatan *website* [14].

2.4 Role-Based Access Control

Role-Based Access Control (RBAC) adalah pendekatan sistem otorisasi dengan memberikan hak akses atau izin berdasarkan jabatan pengguna, bukan secara langsung pada pengguna itu sendiri. Misalnya ketika pengguna sedang ingin mengakses sebuah fitur maka akan muncul pesan penolakan akses. Hal ini dikarenakan peran atau jabatan pengguna tidak cocok atau tidak diperbolehkan untuk mengakses fitur tersebut [15].

Komponen-komponen RBAC terdiri dari *user*, *roles*, *permission*, dan *session*. *User* adalah pengguna yang mengakses sistem. *Roles* adalah jabatan *user* dalam organisasi yang memiliki wewenang tertentu. *Roles* menghubungkan *user* dengan *permission*. *Permission* adalah hak akses ke fitur dalam sistem. *Session* adalah tempat menyimpan informasi pengguna secara semi-permanen, yang hanya tersedia selama periode waktu tertentu [15].

Dalam implementasinya, model RBAC meningkatkan keamanan sistem dengan memastikan bahwa setiap pengguna hanya memiliki akses yang sesuai tugas dan tanggung jawab masing-masing. Pengaturan peran dan hak akses dapat dilakukan secara terpusat, sehingga mengurangi waktu dan upaya dalam administrasi sistem [15].

RBAC memiliki keterbatasan, contohnya dalam menangani skenario akses yang kompleks, tidak fleksibel dalam mengelola perubahan peran pengguna berbasis konteks, dan keterbatasan jumlah atribut yang dapat digunakan untuk mengatur akses dibandingkan dengan *Attribute-Based Access Control* (ABAC). RBAC tidak efektif untuk situasi pengguna unik yang memiliki tanggung jawab mencakup beberapa peran, ketika menghadapi logika bisnis rumit yang sulit didefinisikan dalam peran, ketika skala sistem yang memerlukan administrasi konstan untuk menyesuaikan izin peran, serta ketika otorisasi didasarkan pada parameter dinamis [15].

Namun RBAC ideal jika fokusnya adalah efisiensi dan efektivitas dalam memberikan dan mencabut hak akses pada pengguna. Implementasi RBAC membawa

manfaat signifikan dalam mengelola hak akses pengguna, meningkatkan keamanan, efisiensi operasional dalam kompleksitas dan beban manajemen, serta kepatuhan dalam lingkungan organisasi yang kompleks [15].

Meskipun model ABAC (*Attribute-Based Access Control*) menawarkan kontrol yang lebih dinamis dan fleksibel dibandingkan RBAC, pengimplementasian ABAC pada studi kasus ini dinilai terlalu kompleks dan tidak efisien. Dalam konteks operasional Galaxy Cafe, struktur organisasi dan pembagian tugas kerja (*job description*) sudah terdefinisi dengan sangat jelas dan statis, yaitu manager pusat, manager cabang, admin keuangan dan admin inventaris. Oleh karena itu, metode RBAC tetap dipilih secara eksplisit karena arsitekturnya yang lebih sederhana, mudah dikelola, dan sudah sangat memadai untuk memenuhi kebutuhan batasan hak akses *back-office* antardivisi di Galaxy Cafe tanpa membebani kinerja sistem.

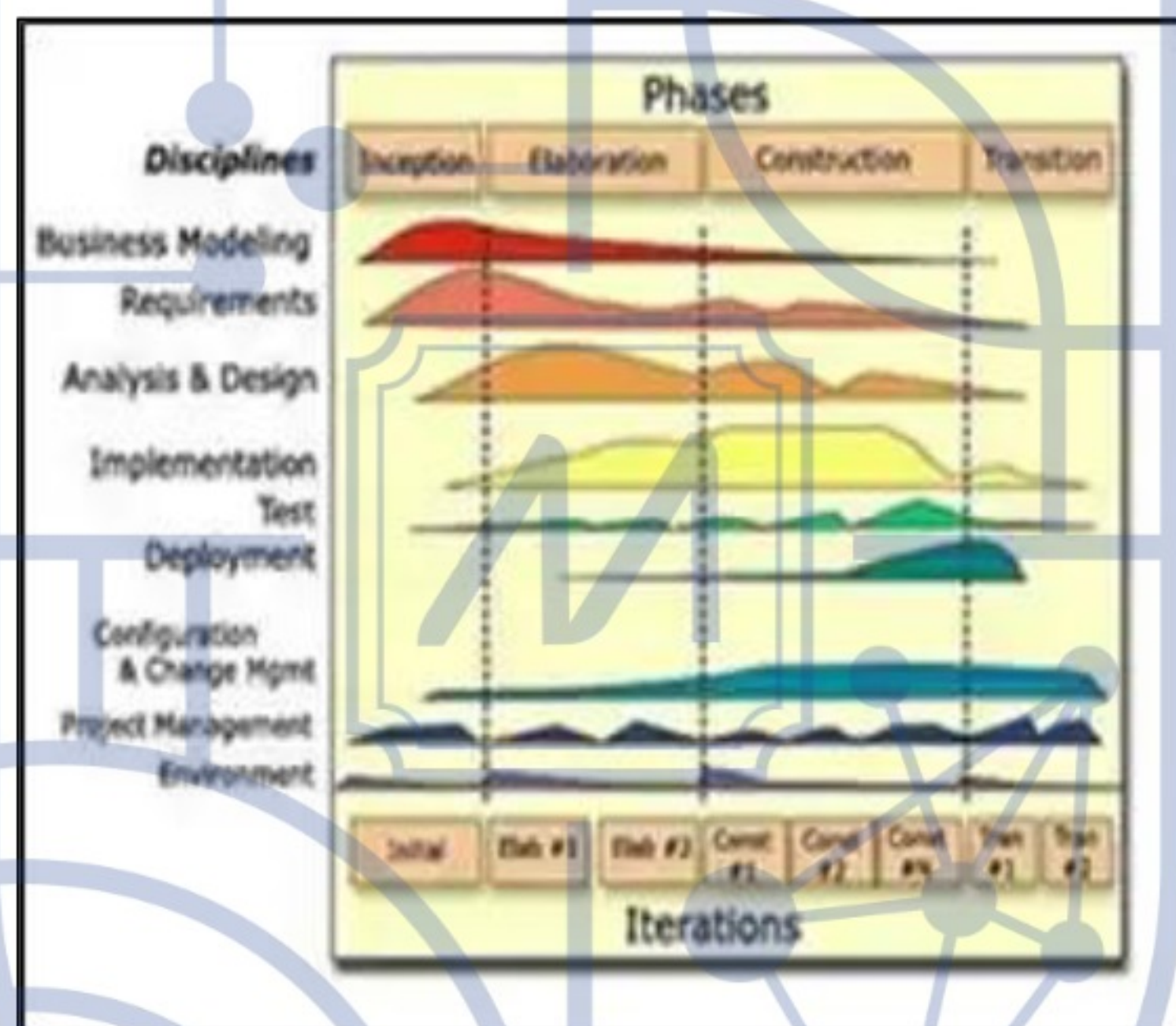
2.5 Metodologi Pengembangan Sistem

Metodologi pengembangan sistem dibutuhkan untuk membangun sebuah sistem informasi. Pemilihan metodologi mempengaruhi banyak hal dan harus disesuaikan dengan kompleksitas proyek, kebutuhan bisnis, dan ketersediaan sumber daya. Adapun bidang yang melakukan implementasi atau pengembangan sistem yaitu bidang pendidikan, bisnis, kepegawaian, kesehatan, dan pemerintah [16].

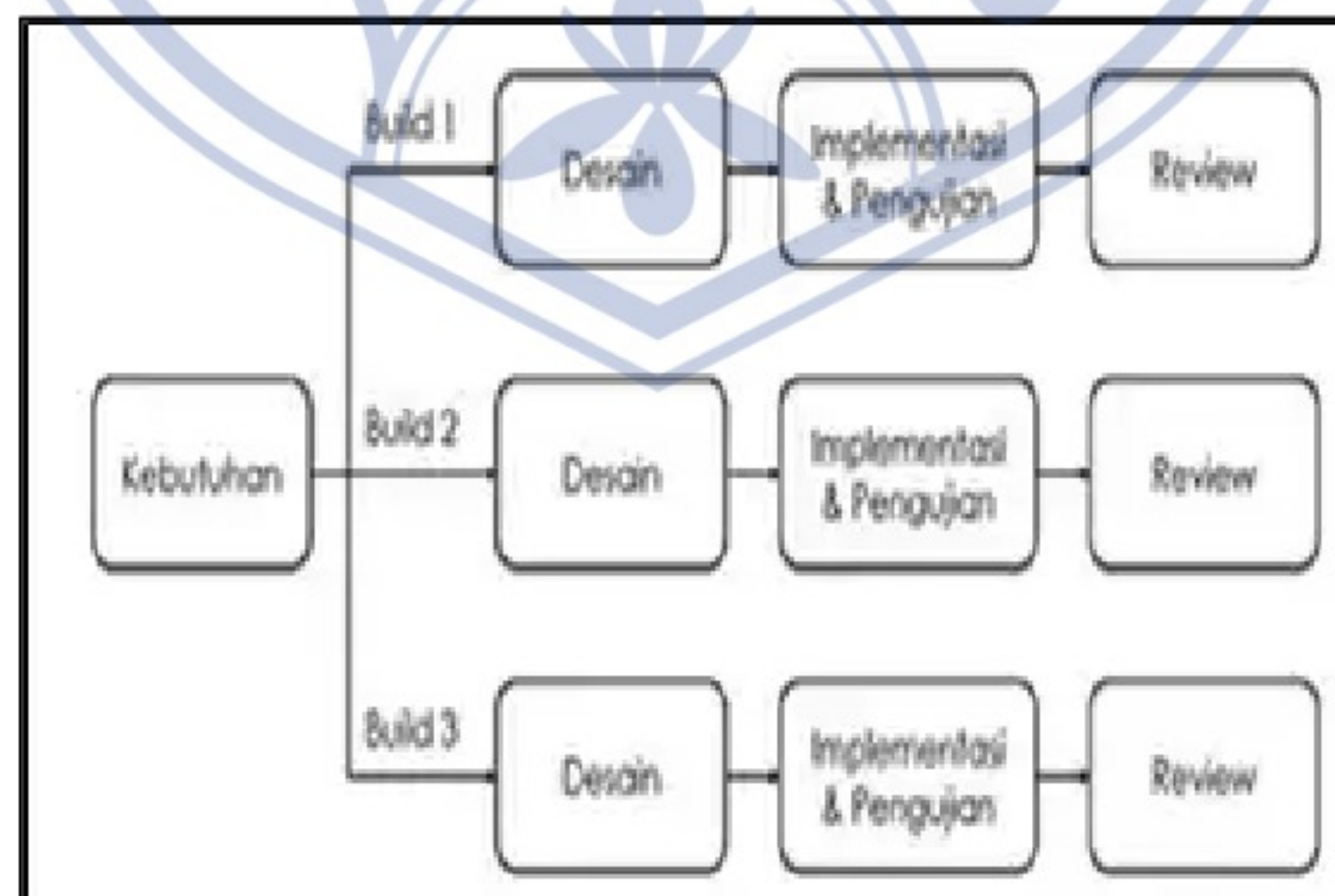
Contoh metodologi yang banyak digunakan meliputi *waterfall*, RUP, dan iteratif. Metodologi *waterfall* dilakukan secara berurutan. Untuk melanjutkan ke tahap selanjutnya, harus terlebih dahulu menyelesaikan tahap sebelumnya. Metodologi *Rational Unified Process* (RUP) memiliki kemampuan untuk menangani risiko terkait kebutuhan sistem yang berdasarkan kepuasan pengguna. Dilakukan pengujian setiap tahap akhir RUP. Metodologi iteratif mengutamakan pembangunan *software* secara iteratif yang berdasarkan pada spesifikasi awal model dasar aplikasi yang dibangun [16].



Gambar 2.3 Tahapan Metodologi *Waterfall*



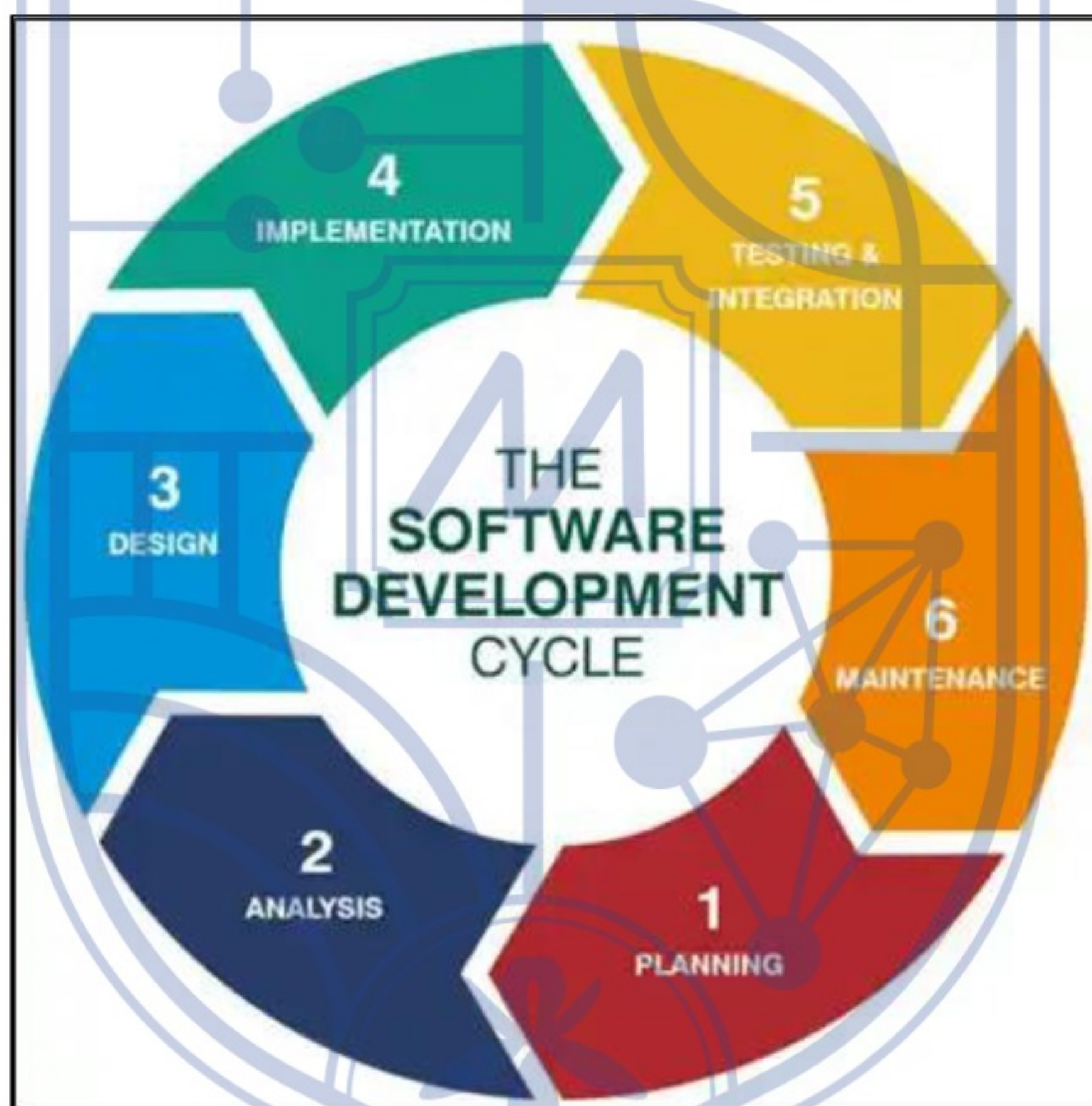
Gambar 2.4 Tahapan Metodologi RUP



Gambar 2.5 Tahapan Metodologi Iteratif

2.5.1 Metodologi Agile

Metodologi *agile* sangat berperan dalam menjaga fleksibilitas dan reaktivitas selama proses pengembangan berlangsung. Cara kerjanya adalah dengan membagi proses pengembangan menjadi beberapa *sprint* yang sanggup dikelola secara paralel. Dengan demikian, tim dapat fokus dalam mengerjakan fitur-fitur spesifik secara tepat waktu, sekaligus memungkinkan umpan balik terus menerus dan adaptasi. Pendekatan ini menjamin alur proyek tetap sesuai jalur dan dengan cepat beradaptasi pada persyaratan baru maupun tantangan yang mungkin muncul. Sifat terstruktur namun fleksibel ini menjamin hasil final aplikasi komprehensif dan sejajar dengan tujuan strategis perusahaan [17].



Gambar 2.6 Alur Metodologi Agile [18]

Kemampuan pengembangan aplikasi *back-office* tidak hanya memenuhi kebutuhan mendesak penggunaannya, namun juga mendukung rencana strategis jangka panjang perusahaan tersebut. Seiring dengan pertumbuhan dan perkembangan organisasi, pendekatan tersebut tetap menjadi aspek penting dalam menjamin solusi teknologi yang selaras dengan tujuan bisnis dan mendorong kesuksesan yang berkelanjutan [17].

Berbeda dengan *waterfall* yang bersifat kaku dan mengharuskan seluruh spesifikasi sudah harus final di awal, metode *agile* dipilih dalam penelitian ini karena sifatnya yang

adaptif. Kondisi operasional Galaxy Café sebagai bisnis F&B yang masih tetap berkembang dan dinamis. Karena itu belum ada ketentuan, spesifikasi, dan desain sistem yang jelas. Hal ini memungkinkan terjadinya perubahan atau penambahan kebutuhan fitur dari pihak manajemen di tengah proses pembuatan perangkat lunak. Dengan *agile*, pengembangan sistem dapat dilakukan secara bertahap, sehingga setiap modul yang selesai dapat langsung dievaluasi oleh mitra untuk memastikan aplikasi yang dibangun benar-benar relevan dengan masalah di lapangan.

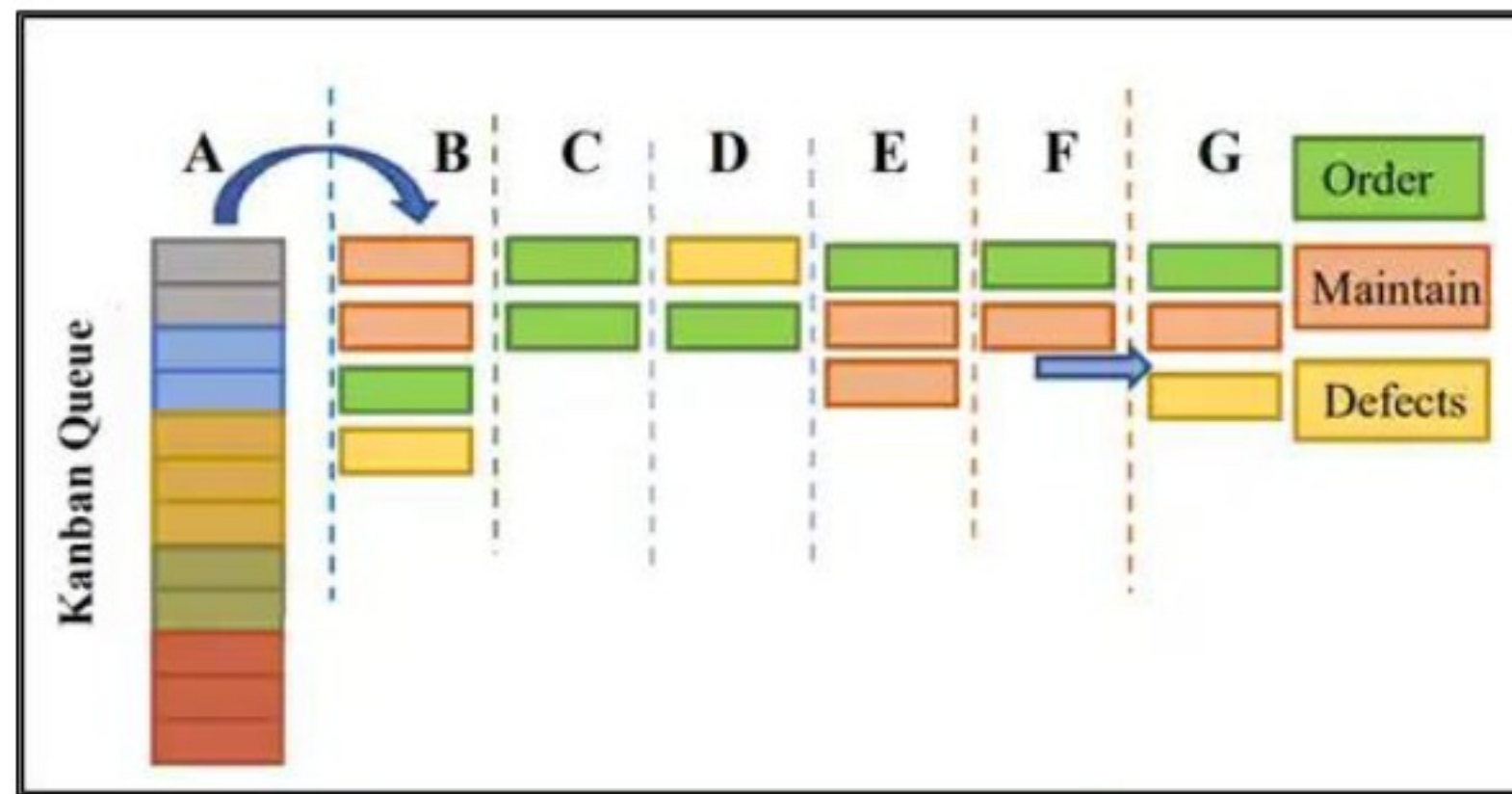
2.5.2 Kanban

Salah satu prinsip yang mengimplementasikan metode *agile* adalah Kanban. Prinsip Kanban menekankan visualisasi alur kerja dan pembatasan tugas yang sedang berlangsung. Penerapan prinsip ini membantu koordinasi tim, meningkatkan transparansi penugasan, serta mengurangi miskomunikasi. Dengan demikian, tim dapat menjaga fokus dan menghindari penumpukan pekerjaan yang sedang dikerjakan. Contoh implementasinya adalah dengan menggunakan papan visual yang membagi tugas ke dalam kolom seperti *To Do*, *In Progress*, *Submitted*, *Testing*, *Revised*, dan *Done*. Kanban adalah metodologi *agile* yang efisien dan membawa paradigma baru dalam pengembangan perangkat lunak. Metode ini berasal dari Toyota Production System dan telah berkembang menjadi struktur *project management* yang solid [19].



Gambar 2.7 Kanban Board Sederhana

Gambar 2.7 menunjukkan tiga tahap mendasar Kanban, yaitu *To Do*, *Doing*, dan *Done*. Namun terdapat cara lain yang menyajikan tampilan yang lebih jelas untuk setiap tahapan alur kerja agar pengerjaan tugas dan proses dapat dilacak secara efektif. Adapun Kanban yang mendetail dapat meliputi tahapan *backlog* atau *input stream* (A), analisis (B), desain (C), pengembangan (D), pengujian (E), *deployment* (F), dan *done* (G). Misalnya pada Gambar 2.8 [19].



Gambar 2.8 Kanban Board Mendetail

Dalam manajemen proyek perangkat lunak, metode Kanban dari *agile* menawarkan pendekatan manajemen proyek secara sistematis, visibilitas jelas, dan pengiriman berkelanjutan. Berdasarkan survei yang dilakukan pada tahun 2024, Kanban dapat meningkatkan *Software Project Management* (SPM) secara signifikan, dari segi kualitas, biaya, jangkauan, *timeline*, risiko, dan manajemen HR. Performa dengan menerapkan Kanban lebih baik dari metodologi *agile* lainnya seperti *scrum*, XP, dan DSDM. Hasil survei juga menunjukkan bahwa Kanban berkontribusi terhadap siklus pengiriman yang cepat, komunikasi tim yang lebih baik, dan peningkatan efisiensi [20]. Metode ini diterapkan sebagai kerangka kerja manajemen proyek utama. Penggunaan papan visual ini secara efektif membantu pengembang dalam melacak progress pengerjaan setiap modul fitur, mengelola prioritas tugas, serta memastikan proses penyelesaian sistem berjalan dengan terstruktur, tepat waktu, dan efisien.

2.6 Metode Pengujian Sistem

Proses evaluasi perangkat lunak secara terstruktur untuk memastikan bahwa sistem yang dikembangkan memenuhi persyaratan dan spesifikasi yang telah ditentukan. Pengujian ini bertujuan untuk mengidentifikasi kesalahan (*bug*), memvalidasi fungsionalitas, dan memastikan kualitas perangkat lunak sebelum diimplementasikan kepada pengguna akhir. Dalam rekayasa perangkat lunak, terdapat berbagai pendekatan pengujian, di mana dua metode yang digunakan dalam penelitian ini adalah *Black Box Testing* dan *White Box Testing*.

2.6.1 *Black Box Testing*

Black Box Testing adalah metode pengujian yang meninjau fungsionalitas perangkat lunak tanpa mempertimbangkan struktur internal atau implementasi kodenya. Metode ini dilakukan oleh penguji yang hanya berinteraksi dengan UI dan memverifikasi *output* yang dihasilkan input spesifik berdasarkan ketentuan dan spesifikasi perangkat lunak. Dengan kata lain, *testing* dilakukan dari perspektif *end-user*, terpisah dari pengembangan kode, dan *test case* dibuat berdasarkan skenario terdefinisi. Kegiatan ini membantu menemukan cacat pada tampilan, perilaku fungsional, dan proses *input/output* [21].

Kelemahan dari metode ini meliputi inefisiensi dalam mendeteksi *error* tersembunyi pada kode program, keterbatasan jangkauan, ketergantungan pada kelengkapan perangkat lunak maupun dokumentasinya, serta kesulitan menentukan skenario *test case* yang tepat, lengkap, terutama untuk skenario yang kompleks dengan banyak kondisi input [21].

Adapun kelebihan adalah eksekusi *testing* yang sederhana dan tidak memerlukan pengetahuan ahli tentang kode, sehingga semua individu dengan berbagai keahlian dapat melakukannya. Kemudian fokus terhadap fungsionalitas yang mendukung keseimbangan fitur-fitur pada perangkat lunak dengan kebutuhan dan spesifikasi pengguna. Lalu *testing* juga lebih cepat karena tidak memerlukan analisis mendalam terhadap kode. Terakhir, karena *testing* didasarkan pada spesifikasi perangkat lunak, maka hal ini mempermudah pelacakan dan pengulangan *testing* di masa mendatang [21].

Adapun rekomendasi praktis dalam implementasi *Black Box Testing* meliputi pembuatan skenario yang merefleksikan pola pemakaian *end-user* yang sebenarnya ketika berinteraksi dengan perangkat lunak, melakukan pengujian integrasi antarfungsi untuk menjamin keakuratan data, serta memverifikasi bahwa semua fungsi perangkat lunak dan *error handling* untuk kesalahan *user* sudah diuji [21].

Meskipun *Black Box Testing* memiliki keterbatasan dalam mendeteksi kesalahan pada logika internal kode program, metode *Black Box Testing* dipilih dalam penelitian ini karena relevansinya dengan kebutuhan sistem informasi yang berfokus pada *user experience*. Dengan metode ini, peneliti dapat memastikan bahwa seluruh fitur fungsional pada operasional *back-office* dapat berjalan dengan akurat dan sesuai dengan ekspektasi mitra tanpa harus terganggu oleh kompleksitas teknik di balik layar. Kompleksitas teknik di balik layar akan ditangani dengan *White Box Testing*.

2.6.2 White Box Testing

Berbeda dengan *Black Box Testing*, *White Box Testing* adalah metode pengujian perangkat lunak yang berfokus pada analisis struktur internal, logika program, dan implementasi kode secara detail. Pengujian ini tidak dilakukan dari sudut pandang *end-user*, melainkan dari sudut pandang pengembang (*developer*). Penguji yang melakukan metode ini dituntut untuk memiliki pemahaman teknis yang mendalam terkait bahasa pemrograman dan arsitektur sistem, karena pengujian dilakukan dengan menelusuri jalur eksekusi kode, struktur kontrol (*looping* dan percabangan), serta aliran data di dalam sistem [22].

Kelebihan utama dari *White Box Testing* adalah kemampuannya untuk mendeteksi *error* atau celah keamanan yang tersembunyi di dalam kode, mengoptimalkan penulisan struktur algoritma, dan memastikan bahwa seluruh baris kode telah dieksekusi dan diuji. Metode ini sangat efektif untuk mendeteksi *bug* pada tingkat unit atau komponen dasar perangkat lunak [22].

Namun, metode ini juga memiliki kelemahan yang signifikan, di antaranya adalah membutuhkan sumber daya spesialis dengan keahlian pemrograman tingkat lanjut, memakan waktu yang jauh lebih lama, dan membutuhkan biaya pengujian yang relatif tinggi. Selain itu, *White Box Testing* murni berfokus pada kebenaran logika kode, sehingga bisa saja sistem dinyatakan lolos uji secara internal, namun tidak memberikan fungsionalitas visual atau alur kerja operasional yang sesuai dengan kebutuhan manajemen [22]. Oleh karena itu, dalam konteks pengembangan sistem informasi manajemen pada penelitian ini, *White Box Testing* tidak dijadikan sebagai metode pengujian utama. Jadi, *Black Box Testing* dilakukan untuk menguji tampilan dan interaktivitas secara umum dan *White Box Testing* menguji logika sistem.

2.7 Penelitian Terdahulu

Penelitian yang dilakukan Dea & Nurmi dan dipublikasikan pada Januari 2026 tentang implementasi RBAC pada sistem informasi manajemen memiliki persamaan dengan penelitian ini [23]. Termasuk dalam hal penggunaan *framework* Laravel dengan *blade template* beserta *package* Spatie Permission untuk implementasi mekanisme RBAC, kelola stok, cetak laporan, dan juga metode *Black Box Testing* untuk pengujian fungsional sistem. Namun pada penelitian ini kami menggunakan metode *agile* untuk mengelola data *back-office* kantor secara lebih detail dan dilengkapi dengan tambahan fitur pemulihan sistem, pencatatan log aktivitas, serta visualisasi grafik. Sedangkan penelitian tersebut menggunakan metode pengembangan *waterfall* dan tidak ada fitur yang disebutkan.

Perbedaan paling mendasar adalah sistem ini mengadopsi prinsip RBAC tersebut dan mengembangkannya untuk arsitektur skala multi cabang, artinya hak akses pengguna dibatasi tidak hanya pada level fitur fungsional menu, tetapi juga secara spesifik dibatasi pada level lokasi cabang mana yang diizinkan untuk diakses. Perbedaan kecil lainnya adalah versi Laravel yang digunakan pada penelitian tersebut adalah versi 11. Sedangkan pada penelitian ini adalah Laravel versi 12 yang juga didukung dengan TALL *stack*.

Penelitian yang dilakukan Maria & Rouli pada tahun 2024 tentang pengembangan sistem informasi manajemen inventaris ini memiliki persamaan dengan penelitian ini dalam hal pengembangan sistem informasi manajemen inventaris berbasis web, basis data menggunakan MySQL, memiliki fitur ekspor laporan, serta menerapkan pengujian dengan metode *Black Box Testing* [24]. Namun, terdapat perbedaan yang sangat mendasar pada aspek metodologi, teknologi, dan arsitektur kontrol akses. Penelitian tersebut menggunakan metode *Waterfall* dengan PHP Native untuk digitalisasi operasional gudang pada perusahaannya dan sistem pada penelitian tersebut hanya menerapkan autentikasi *login* standar untuk satu entitas admin, sedangkan penelitian ini mengamankan data *back-office* dengan mengimplementasikan RBAC pada arsitektur berskala multi-cabang, yang memastikan hak akses pengguna dibatasi secara spesifik baik pada level fitur fungsional maupun level lokasi cabang, serta didukung oleh fungsionalitas tambahan berupa visualisasi grafik penjualan dan pemulihan sistem.

Penelitian yang dilakukan pada tahun 2025 tentang optimalisasi *spreadsheet* dan sistem informasi manajemen dalam proses audit keuangan memiliki persamaan dengan penelitian ini dalam hal sistem informasi manajemen keuangan multi-cabang. Namun, penerapan pada penelitian tersebut fokus pada audit dan diterapkan menggunakan Google Spreadsheet. Cara tersebut terbukti efisien karena adanya fitur rumus otomatis, validasi data, dan template audit yang meningkatkan efisiensi waktu secara signifikan. Namun cara tersebut juga masih tidak luput dari kesalahan input data, *spreadsheet* yang kurang mampu menangani volume data yang besar dan kompleks, kurangnya sistem keamanan yang membatasi hak akses, dan sulit melacak modifikasi [25]. Hal ini dapat diatasi dalam sistem pada penelitian ini dengan aturan beserta validasi input yang ketat, penggunaan basis data relasional SQL, penerapan RBAC, dan fitur pencatatan log aktivitas pengguna.

Penelitian sebelumnya masih membahas aspek sistem secara terpisah yang sudah terbukti efisien dan efektif dalam meningkatkan kinerja bisnis serta pelaporan dan pencatatan data. Sistem pada penelitian ini akan menyatukan seluruh aspek tersebut dalam

manajemen back-office multi cabang dengan RBAC, log aktivitas, unduh laporan, mekanisme pencadangan data, dan pemulihan data dengan metode *agile*.

2.8 Profil Mitra

Galaxy Café adalah perusahaan yang beroperasi di bidang *food and beverage* (F&B) dan olahraga biliar. Usaha ini pertama kali beroperasi dengan nama *Galaxy Café & Pool* yang terletak di Jl. Pasar III No.145A-145B dan resmi dibuka pada 9 Maret 2025. CV SUKSES BERKAT HOKI mengoperasikan cabang ini. Seiring dengan perkembangan bisnis dan peningkatan jumlah pelanggan, pihak manajemen pusat memutuskan untuk melakukan ekspansi dengan membuka cabang kedua. Cabang baru ini bernama *Galaxy Café & Eatery* yang berlokasi di MMV4+M9R, Jl. Marelان Raya Ps. II, dan mulai beroperasi pada 20 Januari 2026. CV SUKSES BERKAT DOA ditunjuk untuk mengoperasikan cabang ini dengan pengawasan manajemen pusat. Dengan adanya dua lokasi yang berbeda, struktur operasional mitra membutuhkan pembagian tugas yang jelas dalam hal pengelolaan data (*back-office*).



Gambar 2.9 Cabang Tuasan Galaxy Café and Pool



Gambar 2.10 Cabang Marelan Galaxy Café and Eatery

2.9 Alat dan Teknologi yang Digunakan

Pengembangan situs web pada penelitian ini menggunakan *TALL Stack*, yaitu kombinasi dari Tailwind CSS, AlpineJS, Laravel, dan Livewire. Dengan demikian, untuk membangun keseluruhan situs web, tidak perlu menggunakan framework SPA lain secara terpisah. Untuk proses *coding* juga hanya perlu berfokus pada *JavaScript* dan PHP. Pemilihan teknologi ini cocok untuk kebutuhan sistem yang bersifat monolitik dan tidak memerlukan API terpisah. *Tools* ini sangat terintegrasi dan melengkapi satu sama lain sehingga terasa seperti satu ekosistem [26].

Tailwind CSS adalah *framework* CSS berbasis utilitas untuk mendesain UI dengan cepat. Adapun untuk konsistensi komponen-komponen beserta pengaturan tema, penelitian ini juga memanfaatkan *plugin* Tailwind CSS yang bernama daisyUI. Dengan demikian, untuk *styling*, tidak perlu menuliskan custom CSS secara terpisah, cukup tambahkan ke dalam atribut *class* [27].

AlpineJS adalah *framework* JavaScript ringan yang dapat mengontrol interaksi UI kecil untuk bagian *client*. Dengan ini, dapat dilakukan *binding* dan manipulasi DOM langsung tanpa menulis perintah JavaScript yang rumit. Selain itu, kita juga dapat mengatur *state* untuk detail sederhana di UI dan melakukan *event handling* dengan mudah. AlpineJS tidak perlu di-*install* secara terpisah karena sudah dalam satu paket bersama Livewire [26].

Laravel adalah pilar utama dalam sistem ini. Backend utama, manajemen data, logika bisnis, dan semua integrasi dengan Livewire dilakukan melalui Laravel. Artinya sudah tersedia struktur file dan folder yang mengatur *views, components, layouts, routing*, Eloquent ORM, *auth, validation*, fitur-fitur siap pakai, dan keamanan built-in. Dengan demikian, tidak perlu membangun sistem dari nol. Selain itu, tersedia fitur keamanan secara *built-in* seperti CSRF dan hashing. *Framework* ini juga ideal untuk mengembangkan dengan prinsip OOP dan struktur MVC [26]. Adapun versi Laravel yang digunakan untuk pengembangan sistem ini adalah Laravel 12 yang dirilis pada 24 Februari 2025. Penggunaan versi lebih baru agar sistem bersifat lebih *future proof*.

Sama halnya dengan Livewire. Versi yang digunakan adalah versi 4 yang dirilis pada Januari 2026. Sedangkan untuk fungsi Livewire adalah membuat UI yang reaktif tanpa SPA. Jadi tidak terjadi *full page reload* setiap ada perubahan komponen dalam UI. Livewire juga menghubungkan *frontend* dan *backend* secara langsung tanpa API, menggantikan peran *controller* pada Laravel tradisional. Adapun fitur bawaan yang tidak kalah penting adalah Laravel Test untuk melakukan *automated testing* pada komponen Livewire. Fitur ini sangat memudahkan dalam melakukan *testing* dibandingkan alat pihak ketiga yang mengharuskan menggunakan *browser* untuk testing. Fitur ini akan kami gunakan untuk membuat *test case White Box Testing*. Jika dipadukan dengan PCOV, hasil pengujian juga dapat menampilkan cakupan kode yang diuji [26].

Laravel juga dikenal dengan ekosistem dan komunitas yang besar. Karenanya banyak tersedia *library* tambahan yang cocok diterapkan sistem dan sesuai kebutuhan mitra. Beberapa *library* yang dapat diterapkan yaitu *library* untuk membantu proses impor dan ekspor ke format .xlsx dengan Laravel Excel [28]. Mempermudah menerapkan RBAC dengan *library* Laravel Permission oleh Spatie [29]. Ada juga *library* untuk membantu membuat grafik modern dan interaktif, yaitu Apache ECharts [30]. Kemudian *library* Laravel DomPDF untuk membantu *generate* dokumen laporan keuangan dalam format PDF [31]. Seluruh pemilihan *library* didasarkan dari yang paling efektif, sederhana, dan ringan.

Adapun alat tambahan dalam pengembangan sistem selain IDE adalah GitHub untuk penerapan *version control* dan metode agile. Lalu untuk standarisasi *enviroment*, mempermudah setup proyek, serta mendukung kolaborasi antaranggota pengembang, digunakan Docker dengan integrasi melalui Ubuntu [32].

