BAB II
KAJIAN LITERATUR

2.1 Konsep Dasar Sistem

2.1.1 Definisi Sistem

Secara fundamental, sistem bukanlah sekadar kumpulan elemen, melainkan sebuah kesatuan yang kohesif, terdiri dari komponen-komponen yang saling terhubung dan saling bergantung, yang diorganisasikan dengan cara tertentu untuk mencapai suatu tujuan atau serangkaian tujuan yang telah ditetapkan [3]. Pandangan holistik ini menekankan bahwa keseluruhan sistem memiliki nilai lebih besar daripada sekadar penjumlahan bagian-bagiannya, karena interaksi antar komponen menghasilkan properti-properti baru yang tidak dimiliki oleh komponen secara individual. Berbagai definisi dari literatur akademis menyepakati gagasan ini, yaitu adanya bagian-bagian yang saling terkait dan bekerja secara harmonis untuk mencapai tujuan bersama [4].Sebagai analogi, sebuah perusahaan dapat dipandang sebagai sistem di mana berbagai departemen (komponen) seperti pemasaran, keuangan, dan operasional bekerja sama untuk mencapai tujuan profitabilitas [5].

2.1.2 Karakteristik Inti Sebuah Sistem

Untuk memahami konsep sistem secara utuh, penting untuk mengidentifikasi karakteristik universal yang mendefinisikan struktur dan perilakunya. Karakteristik ini menyediakan kerangka kerja untuk menganalisis sistem apa pun, mulai dari perangkat mekanis sederhana hingga organisasi bisnis yang kompleks. Sebuah sistem memiliki beberapa karakteristik kunci yang membedakannya dari sekumpulan elemen yang tidak terorganisir. Karakteristik tersebut meliputi:

1. Komponen (*Components*): Ini adalah bagian-bagian dasar atau subsistem yang membentuk sistem secara keseluruhan. Fungsi sistem yang berhasil sangat bergantung pada operasi dan koordinasi yang efektif dari setiap komponennya.
2. Batasan (*Boundary*): Ini adalah garis pemisah yang mendefinisikan sistem dari lingkungannya. Batasan ini menentukan ruang lingkup sistem, membedakan apa yang bersifat internal dan dapat dikendalikan dari apa yang bersifat eksternal.
3. Lingkungan Luar (*Environments*): Ini mencakup segala sesuatu di luar batasan sistem yang dapat memengaruhi operasinya. Meskipun sistem tidak memiliki kendali langsung atas lingkungannya, ia harus mampu beradaptasi untuk bertahan hidup. Contohnya termasuk pesaing, regulasi pemerintah, dan kondisi ekonomi.

4. Penghubung (*Interface*): Ini merujuk pada titik-titik kontak di mana komponen-komponen di dalam sistem terhubung satu sama lain, atau di mana sistem secara keseluruhan berinteraksi dengan lingkungannya. Penghubung memfasilitasi aliran informasi, material, atau energi.
5. Masukan (*Input*), Proses (*Process*), dan Keluaran (*Output*): Triad ini menggambarkan logika operasional fundamental dari sebuah sistem. Sistem menerima masukan dari lingkungannya, mengubahnya melalui suatu proses internal, dan melepaskan keluaran kembali ke lingkungannya. Ini adalah mekanisme inti yang digunakan sistem untuk mencapai tujuannya.
6. Sasaran atau Tujuan (*Objectives/Goal*): Merupakan hasil akhir yang diharapkan dari keberadaan sebuah sistem, di mana seluruh komponen dan proses di dalamnya diintegrasikan untuk mencapainya.

2.1.3 Informasi

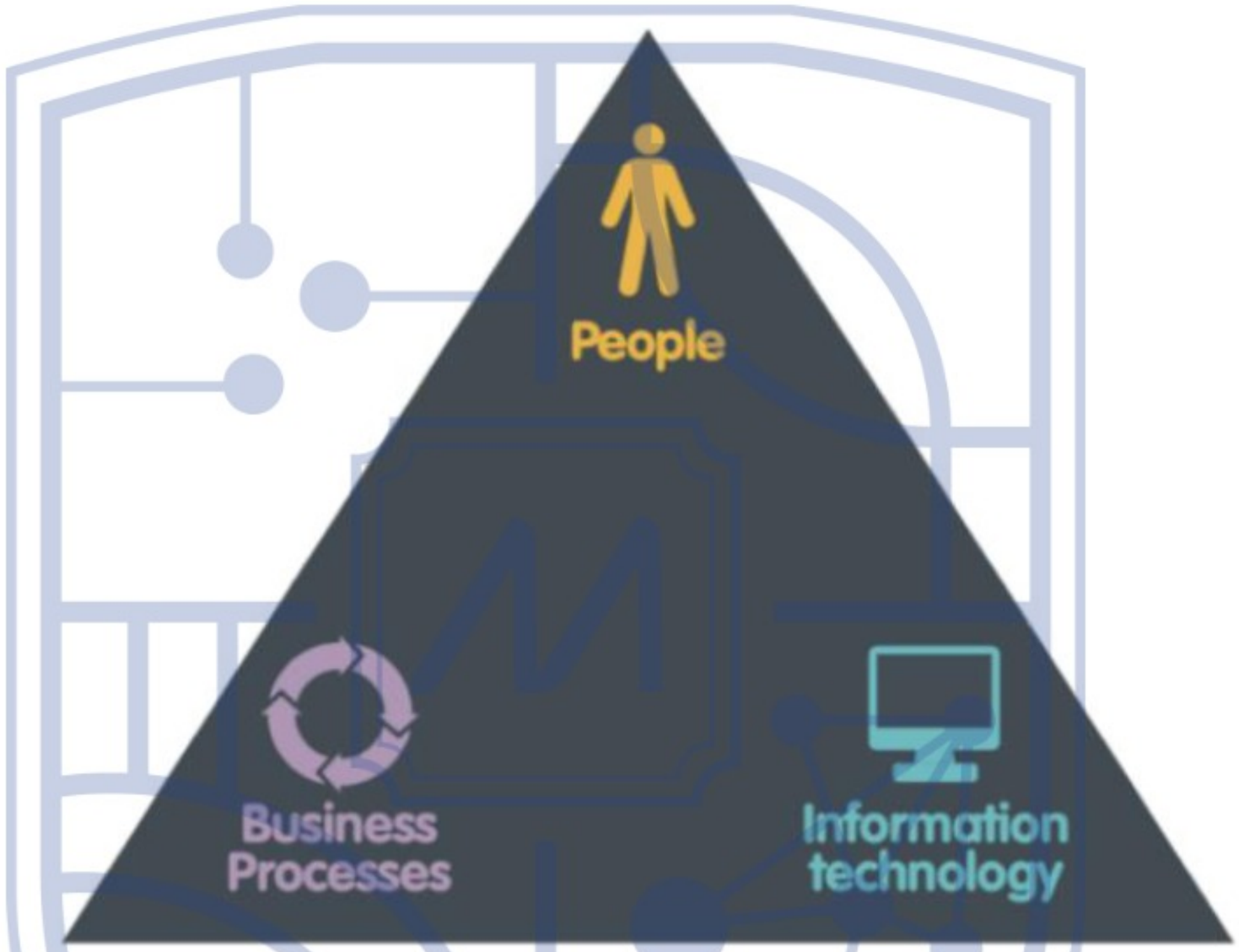
Informasi dapat dipahami sebagai kumpulan data atau fakta yang telah diolah dan diorganisasi sedemikian rupa sehingga memiliki makna dan nilai tertentu bagi pihak yang menerima. Dengan kata lain, informasi merupakan hasil dari proses pengolahan data mentah menjadi bentuk yang lebih terstruktur dan bermanfaat, karena mampu memberikan penjelasan, keterangan, maupun pengetahuan yang relevan bagi penggunanya. Dalam konteks organisasi atau instansi, informasi memiliki peranan yang sangat penting, karena menjadi dasar bagi pengambilan keputusan, perencanaan, serta pengendalian kegiatan agar tujuan yang telah ditetapkan dapat tercapai secara efektif dan efisien [6].

2.1.4 Sistem Informasi

Pada era digital kontemporer, informasi telah menjadi aset krusial yang menopang seluruh aspek kehidupan, terutama dalam konteks operasional bisnis dan organisasi. Sistem Informasi (SI) berperan sebagai fondasi esensial dalam tata kelola dan pemanfaatan informasi secara efektif, yang bertujuan untuk mendukung tiga fungsi vital organisasi: pengambilan keputusan (*decision-making*), pengendalian (*control*), dan aktivitas operasional. Pemanfaatan SI secara strategis memungkinkan organisasi untuk meningkatkan efisiensi, akurasi data, dan daya saing di pasar. Dalam implementasinya, sistem informasi mengikuti sebuah siklus hidup terstruktur yang terdiri dari lima fase utama: perencanaan, pengembangan, implementasi, operasi, dan pemeliharaan.

Sistem informasi dirancang secara spesifik untuk merealisasikan tujuan-tujuan partikular dalam sebuah organisasi. Objektif ini dapat mencakup berbagai sasaran, seperti

optimalisasi efisiensi operasional, peningkatan kualitas pengambilan keputusan, peningkatan produktivitas, serta penyelarasan dukungan teknologi terhadap strategi bisnis yang lebih luas. Keberhasilan implementasi sistem informasi sangat bergantung pada seberapa baik integrasi atau sinergi yang terjalin antara tiga komponen fundamental: manusia (orang), alur kerja (proses), dan infrastruktur (teknologi). Dalam diskursus sistem informasi, keterkaitan harmonis antara ketiga elemen ini sering dirujuk sebagai konsep *golden triangle* atau segitiga emas.



Gambar 2.1 Golden Triangle Sistem Informasi

Konsep *golden triangle* (segitiga emas) merujuk pada sebuah kerangka kerja yang menekankan interdependensi antara tiga elemen kunci: manusia (*people*), prosedur operasional (*process*), dan infrastruktur (*techmology*). Sinergi dan kolaborasi antara ketiga elemen fundamental ini sangat esensial untuk membentuk lingkungan organisasi yang kondusif, yang pada akhirnya mendukung implementasi strategi dan pencapaian tujuan organisasi secara efektif [7].

2.2 Konsep Aplikasi Web

Aplikasi web adalah perangkat lunak yang berjalan di browser web. Bisnis harus bertukar informasi dan memberikan layanan dari jarak jauh. Bisnis menggunakan aplikasi web untuk terhubung dengan customer secara nyaman dan aman. Fitur situs web yang paling umum, seperti keranjang belanja, pencarian dan pemfilteran produk, pesan instan, serta umpan berita media sosial adalah aplikasi web dalam desainnya. Dengan fitur-fitur tersebut,

Pengguna dapat mengakses fungsionalitas yang kompleks tanpa menginstal atau mengonfigurasi perangkat lunak [8].

2.2.1 Website Statis

Website statis merujuk pada jenis situs web yang menyajikan konten bersifat tetap (*fixed*), di mana setiap perubahan atau pembaruan informasi memerlukan intervensi manual langsung pada kode sumber (*source code*) oleh seorang *webmaster* atau pengembang. Arsitektur fundamentalnya dibangun menggunakan HTML untuk struktur, CSS untuk aspek visual (tampilan), dan JavaScript untuk fungsionalitas interaktif yang sederhana. Oleh karena kontennya disimpan langsung dalam bentuk *file*, website statis umumnya tidak memerlukan interdependensi dengan sistem manajemen *database*. Jenis situs ini lazim diimplementasikan untuk kebutuhan seperti halaman arahan (*landing page*), portofolio digital, atau situs informasional sederhana yang tidak menuntut pembaruan konten secara frekuen [9].

2.2.2 Website Dinamis

Website dinamis didesain untuk menampilkan konten yang dapat berubah-ubah secara *real-time*, bergantung pada interaksi pengguna, parameter waktu, atau data yang dikelola di sisi *server*. Kemampuan dinamis ini dicapai melalui penggunaan bahasa pemrograman *server-side* (seperti *PHP, Python*, atau *Node.js*) yang memproses logika dan permintaan sebelum mengirimkan hasil akhir ke peramban pengguna. Berbeda dengan situs statis, website dinamis sangat bergantung pada *database* (basis data) untuk menyimpan, mengelola, dan mengambil data secara terstruktur. Contoh representatif dari website dinamis mencakup platform media sosial, situs *e-commerce*, dan portal berita, yang kesemuanya membutuhkan manajemen konten yang responsif dan pembaruan data yang konstan [9].

2.2.3 Perbedaan Website Statis dan Dinamis

Merujuk pada penjelasan di atas, perbedaan utama antara website statis dan website dinamis terletak pada sifat konten serta cara pengelolaannya. Website statis memiliki konten yang bersifat tetap dan tidak berubah kecuali dilakukan pembaruan secara manual oleh pengembang melalui pengeditan langsung pada kode sumber. Karena tidak terhubung dengan basis data, situs jenis ini lebih sederhana dan cocok untuk kebutuhan yang tidak memerlukan pembaruan informasi secara berkala, seperti portofolio atau halaman profil perusahaan.

Sementara itu, website dinamis mampu menampilkan konten yang berubah secara otomatis sesuai interaksi pengguna, waktu, atau data yang tersimpan di server. Situs jenis ini

bergantung pada bahasa pemrograman sisi server dan sistem basis data untuk mengelola informasi secara real-time. Oleh karena itu, website dinamis lebih sesuai untuk platform yang membutuhkan pembaruan data berkelanjutan, seperti toko daring, portal berita, atau media sosial.

2.3 Konsep Fungsional Bisnis

2.3.1 Sistem Informasi Manajemen Persediaan

Sistem Informasi Manajemen Persediaan adalah sistem terkomputerisasi untuk mencatat, memantau, dan mengendalikan stok barang agar ketersediaan optimal, menghindari kekurangan atau kelebihan stok, dan menyediakan informasi *real-time* bagi pengambilan keputusan operasional dan pembelian ulang. Sistem ini biasanya mencakup proses barang masuk/keluar, master data, pemasok, dan pelaporan otomatis guna meningkatkan akurasi dan efisiensi manajemen gudang atau toko. Tujuan utama Sistem Informasi Manajemen Persediaan meliputi:

1. Menjamin akurasi data stok dan mengurangi risiko kesalahan pencatatan serta ketidaksesuaian jumlah fisik dengan catatan.
2. Meningkatkan efisiensi operasional melalui pelacakan *real-time* dan otomatisasi pembuatan laporan untuk kebutuhan manajemen.
3. Mendukung keputusan pemesanan (berapa dan kapan pesan ulang) untuk mencegah *stockout* dan *overstock* [10].

2.3.2 Sistem Informasi Penjualan

Sistem Informasi Manajemen Penjualan adalah sistem yang mengintegrasikan proses transaksi penjualan dengan pengelolaan dan pelaporan, umumnya terhubung dengan *Point of Sale* (POS) untuk membuat operasi penjualan lebih efisien dan terstruktur. Tujuan utamanya adalah meningkatkan efisiensi pengelolaan penjualan dengan mempermudah proses transaksi serta menghasilkan laporan penjualan dan keuangan bagi pengambilan keputusan operasional.

Tujuan utama sistem ini adalah meningkatkan efisiensi manajemen penjualan dengan mempermudah dan mempercepat proses transaksi di titik penjualan. Sistem ini ditujukan untuk membantu penyusunan laporan penjualan dan keuangan secara otomatis sehingga informasi kinerja lebih cepat tersedia. Dengan integrasi POS, sistem memusatkan data penjualan agar pengelolaan operasional lebih responsif terhadap kebutuhan bisnis [10].

2.3.3 Pengujian Sistem

Pengujian sistem yang digunakan dalam penelitian ini adalah Black Box Testing, yaitu teknik pengujian perangkat lunak yang berfokus pada pemeriksaan fungsionalitas perangkat lunak tanpa memperhatikan struktur internal atau kode programnya. Pengujian ini dilakukan untuk memvalidasi apakah perangkat lunak dapat menerima input yang benar dan menghasilkan output yang diharapkan sesuai dengan spesifikasi yang telah ditentukan. Dalam pengujian ini, penguji tidak perlu memiliki pengetahuan teknis tentang bahasa pemrograman yang digunakan, sehingga metode ini sering digunakan oleh tim pengujian yang terdiri dari orang-orang dengan latar belakang non-teknis [11].

Terdapat beragam teknik dalam pelaksanaan Black Box Testing, di antaranya Equivalence Partitioning, di mana data input dibagi menjadi beberapa partisi yang dianggap setara untuk memastikan setiap partisi menghasilkan output yang sesuai, serta Boundary Value Analysis yang berfokus pada pengujian nilai batas seperti nilai minimum dan maksimum untuk menemukan error yang mungkin muncul di area tersebut. Meskipun metode ini memiliki kelebihan seperti tidak memerlukan pengetahuan teknis dan berorientasi pada pengalaman pengguna akhir, Black Box Testing juga memiliki kekurangan, yaitu kemungkinan error pada bagian back-end tidak terdeteksi karena pengujian hanya berfokus pada input dan output [11].

2.3.4 Program Loyalitas Customer

Program loyalitas customer merupakan strategi perusahaan untuk mempertahankan customer yang telah ada dengan cara membangun hubungan jangka panjang yang saling menguntungkan antara customer dan perusahaan. Melalui program ini, perusahaan berupaya memperkuat ikatan emosional customer terhadap merek, meningkatkan komitmen, serta menumbuhkan rasa keterikatan yang mendorong customer untuk terus menggunakan produk atau layanan yang sama.

Dalam beberapa kasus, program loyalitas diwujudkan melalui sistem poin atau penghargaan atas setiap transaksi yang dilakukan customer. Sistem tersebut memungkinkan customer mengumpulkan poin yang kemudian dapat ditukar dengan hadiah, diskon, atau layanan tambahan. Contoh penerapan nyata dapat ditemukan pada platform transportasi daring yang mengembangkan sistem rewards guna memberikan apresiasi atas frekuensi penggunaan layanan oleh customer. Tujuan utama dari program loyalitas customer adalah sebagai berikut:

1. Meningkatkan loyalitas customer, yaitu mendorong customer untuk terus melakukan pembelian berulang secara konsisten terhadap produk atau layanan yang sama.
2. Memberikan insentif kepada customer, melalui sistem poin atau hadiah yang dapat meningkatkan kepuasan serta minat untuk bertransaksi kembali.
3. Meningkatkan komitmen dan ikatan emosional, dengan menciptakan hubungan yang lebih erat antara customer dan merek.
4. Mendorong pembelian berulang (*repeat purchases*), melalui pemberian manfaat langsung atas setiap transaksi yang dilakukan customer.

Pentingnya program loyalitas terletak pada kontribusinya terhadap keberlanjutan bisnis. Customer yang loyal tidak hanya cenderung melakukan pembelian secara konsisten, tetapi juga memberikan nilai ekonomi yang lebih tinggi dibandingkan customer baru dalam jumlah yang sama. Oleh karena itu, program loyalitas customer menjadi salah satu strategi penting dalam mempertahankan basis customer sekaligus meningkatkan nilai tambah bagi perusahaan maupun konsumen [12].

2.3.5 Jasa Laundry

Jasa laundry merupakan bentuk usaha yang bergerak di bidang pelayanan pencucian pakaian menggunakan mesin cuci, mesin pengering otomatis, serta cairan pembersih dan pewangi khusus. Usaha ini berkembang pesat terutama di wilayah perkotaan karena memberikan kemudahan bagi masyarakat dalam mengelola kebutuhan kebersihan pakaian. Jasa laundry berorientasi pada pelayanan, di mana kepuasan customer bergantung pada ketepatan waktu penyelesaian, kualitas hasil cucian, dan keandalan sistem pengelolaan pesanan. Berdasarkan waktu pengerjaannya, layanan laundry dibedakan menjadi beberapa kategori utama sebagai berikut:

1. Laundry Kilat

   Jenis layanan dengan waktu pengerjaan tercepat, yaitu maksimal 5 jam. Layanan ini ditujukan bagi customer yang membutuhkan hasil cucian dalam waktu singkat.
2. Laundry Two Days

   Jenis layanan dengan waktu pengerjaan selama 2 hari. Layanan ini menawarkan keseimbangan antara kecepatan pengerjaan dan kualitas hasil.
3. Laundry Reguler (Ordinary)

   Jenis layanan dengan waktu pengerjaan 3 sampai 4 hari. Layanan ini biasanya memiliki biaya lebih ekonomis dan ditujukan untuk customer dengan kebutuhan pencucian rutin tanpa batasan waktu ketat [13].

2.4 Konsep Metodologi dan Pemodelan

2.4.1 Metodologi Pengembangan Rapid Application Development (RAD)

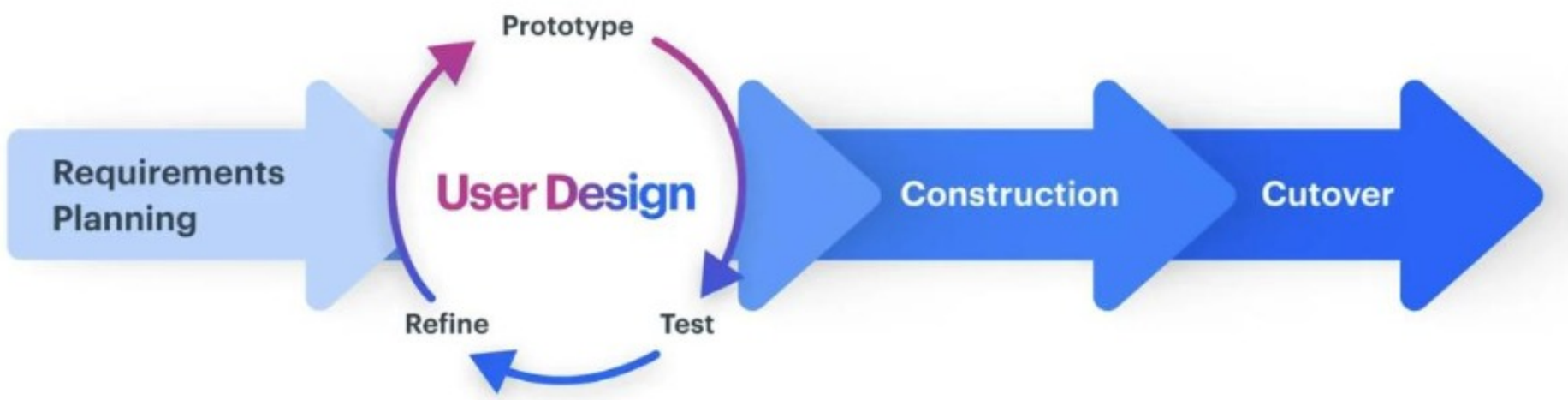
Rapid Application Development (RAD) merupakan model pengembangan perangkat lunak yang bersifat adaptif, berfokus pada pembuatan prototipe dan umpan balik cepat, dengan penekanan yang lebih rendah terhadap perencanaan yang terperinci. Secara umum, pendekatan *RAD* memprioritaskan proses pengembangan dan pembuatan prototipe dibandingkan perencanaan panjang di awal. Melalui pendekatan ini, pengembang dapat melakukan berbagai iterasi dan pembaruan perangkat lunak dengan cepat tanpa harus memulai kembali dari awal. Hal tersebut membantu memastikan bahwa hasil akhir lebih berorientasi pada kualitas serta selaras dengan kebutuhan pengguna akhir.

Konsep *RAD* muncul pada dekade 1980-an, sehingga bukan merupakan pendekatan baru dalam dunia pengembangan perangkat lunak. Namun, berbeda dengan model *waterfall* yang bersifat linear dan kaku, *RAD* bersifat evolusioner dan terus berkembang mengikuti kebutuhan serta konteks teknologi pada masa tertentu.

Gagasan awal RAD dipelopori oleh Barry Boehm, James Martin, dan sejumlah tokoh lain yang menyadari bahwa perangkat lunak tidak harus dibatasi oleh metode rekayasa tradisional. Perangkat lunak dianggap sebagai entitas yang lentur dan dapat disesuaikan dengan kebutuhan pengguna.

Metode ini dipilih karena mampu mempercepat proses pengembangan melalui pembuatan prototipe yang dapat diuji langsung oleh pengguna. Pendekatan RAD menekankan pada iterasi berulang, fleksibilitas terhadap perubahan kebutuhan, serta kolaborasi aktif antara pengembang dan pengguna selama proses pengembangan berlangsung. Dengan demikian, sistem yang dihasilkan diharapkan dapat lebih adaptif terhadap kebutuhan operasional yang nyata di lapangan.

RAD menjadi pendekatan yang ideal untuk mengembangkan prototipe secara cepat guna menguji fungsi perangkat lunak tanpa harus khawatir terhadap dampak langsung pada produk akhir. Banyak organisasi memilih RAD karena fase perencanaannya relatif ringan, sementara tim pengembang dapat dengan cepat merancang, meninjau, dan melakukan iterasi terhadap fitur serta fungsionalitas yang dikembangkan. Berikut adalah tahapan dalam Metodologi Pengembangan *Rapid Application Development (RAD)*



Gambar 2.2 Ilustrasi Metode Pengembangan RAD

1. Requirement Plannning

   Sejak awal, RAD sudah membedakan dirinya dari model pengembangan tradisional. Proses ini tidak menuntut daftar spesifikasi terperinci dari pengguna akhir; yang dibutuhkan hanyalah gambaran kebutuhan secara umum. Pendekatan yang lebih luas ini memungkinkan pengembang menyusun rincian kebutuhan secara bertahap sepanjang siklus pengembangan.

2. Prototyping

   Pada tahap ini proses pengembangan aktual dimulai. Alih-alih mengikuti serangkaian kebutuhan yang kaku, pengembang membuat berbagai prototipe dengan fitur dan fungsi berbeda secepat mungkin. Prototipe tersebut kemudian diserahkan kepada klien untuk memperoleh tanggapan atas fitur yang disukai maupun yang perlu diperbaiki. Biasanya, prototipe ini hanya menampilkan fungsi-fungsi utama dan belum merupakan sistem final. Produk akhir baru akan dibangun setelah pengembang dan klien mencapai kesepakatan penuh terhadap hasil akhir.

3. Construction

   Tahap konstruksi merupakan fase penting di mana prototipe dikembangkan menjadi sistem yang berfungsi penuh. Pada tahap ini, masukan dan umpan balik pengguna sangat menentukan. Pengembang berfokus pada penyempurnaan sistem, perbaikan *bug*, serta penyesuaian berdasarkan hasil uji coba. Durasi tahap ini bisa lebih panjang apabila terdapat banyak revisi atau perubahan arah dari pihak klien.

4. Cutover

   Tahap terakhir adalah penerapan sistem ke lingkungan produksi yang sebenarnya. Fase ini mencakup pengujian skala besar, penyusunan dokumentasi teknis, pelacakan masalah, serta penyesuaian akhir dan simulasi sistem. Tim pengembang juga melakukan proses *debugging*, pembaruan terakhir, serta pemeliharaan awal sebelum sistem resmi dijalankan secara penuh [14].

2.4.2 Use Case Diagram

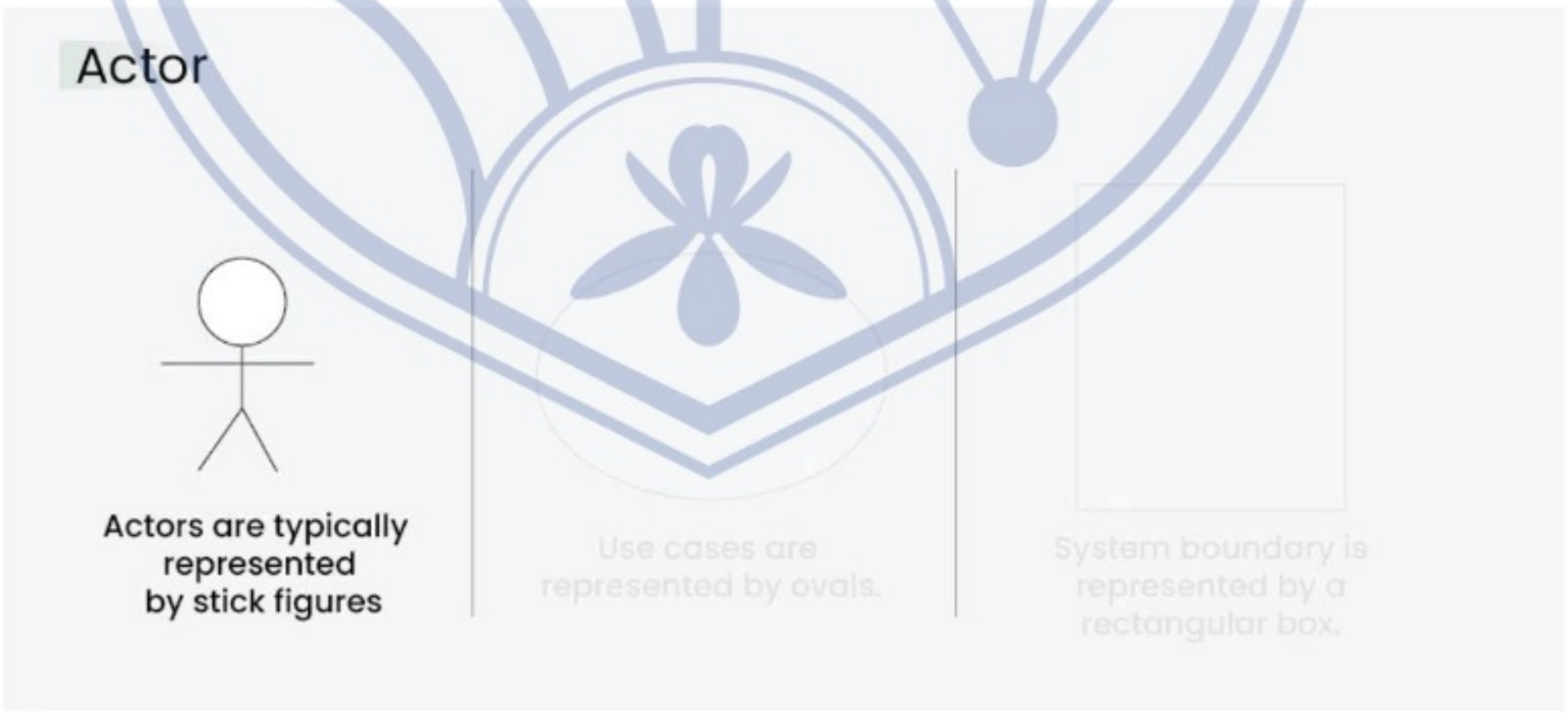
Use Case Diagram merupakan representasi visual yang mengilustrasikan interaksi antara pengguna (*actors*) dan sebuah sistem. Diagram ini menangkap kebutuhan fungsional sistem dengan menunjukkan bagaimana pengguna yang berbeda terlibat dengan berbagai *use cases*, atau fungsionalitas spesifik, di dalam sistem tersebut. *Use Case Diagram* menyajikan gambaran umum tingkat tinggi mengenai perilaku sistem, sehingga berguna bagi pemangku kepentingan (*stakeholders*), pengembang, dan analis untuk memahami bagaimana sistem dimaksudkan untuk beroperasi dari perspektif pengguna, serta bagaimana proses-proses yang berbeda saling berkaitan. Diagram ini sangat krusial untuk mendefinisikan ruang lingkup (*scope*) dan kebutuhan sistem.

Use Case Diagram adalah jenis diagram UML (*Unified Modelling Languge*) yang merepresentasikan interaksi antara aktor (pengguna atau sistem eksternal) dan sistem yang sedang ditinjau untuk mencapai tujuan tertentu. Diagram ini memberikan pandangan tingkat tinggi terhadap fungsionalitas sistem dengan mengilustrasikan berbagai cara pengguna dapat berinteraksi dengannya [15].

2.4.2.1 Notasi Use Case Diagram

1. Actor

Actor adalah entitas eksternal yang berinteraksi dengan sistem. Ini dapat mencakup pengguna manusia, sistem lain, atau perangkat keras. Dalam konteks Diagram Use Case, aktor menginisiasi use cases dan menerima hasilnya. Identifikasi dan pemahaman yang tepat mengenai aktor sangat penting untuk memodelkan perilaku sistem secara akurat.

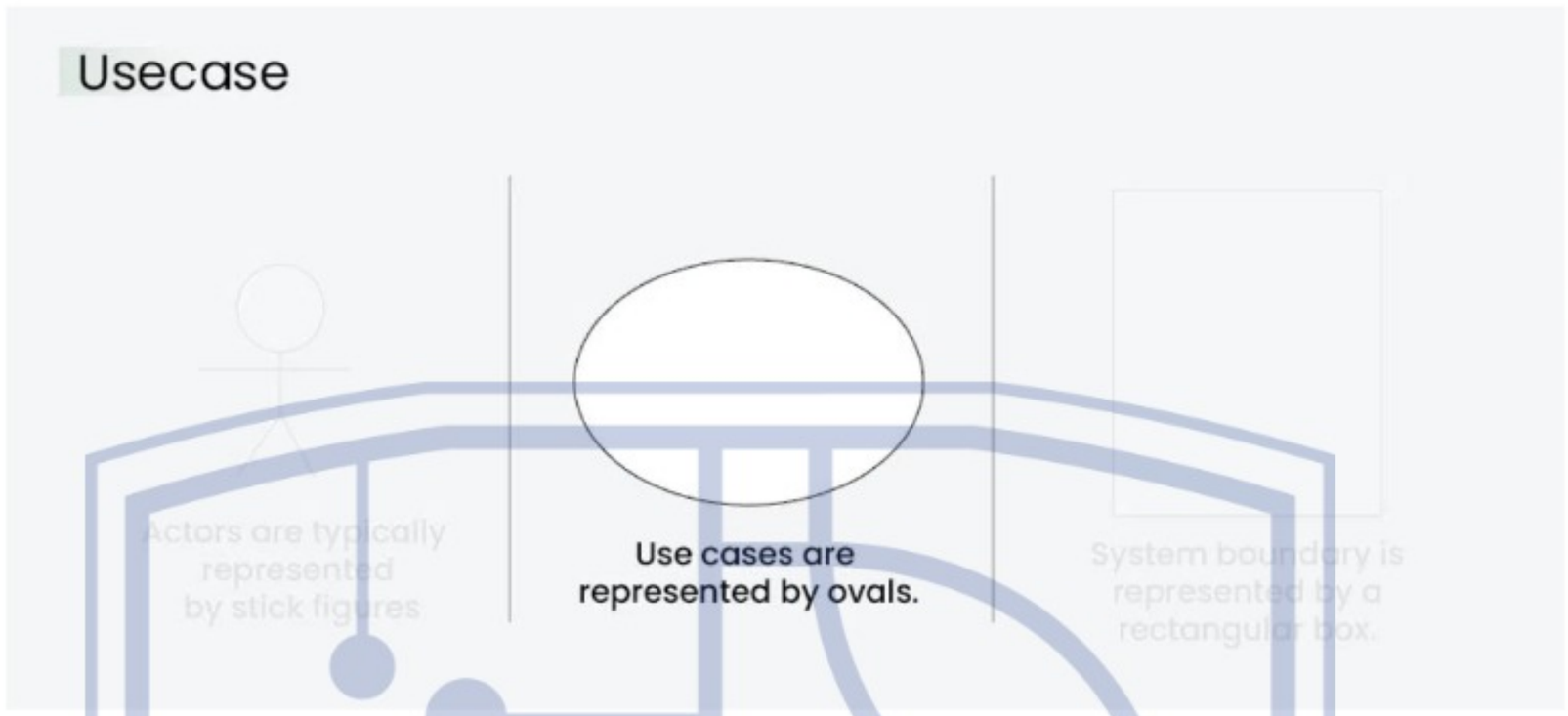


Gambar 2.3 Simbol Aktor pada Use Case Diagram

2. Use Cases

Use cases menyerupai adegan dalam sebuah pertunjukan. Mereka merepresentasikan hal-hal spesifik yang dapat dilakukan oleh sistem Anda. Dalam sistem belanja daring,

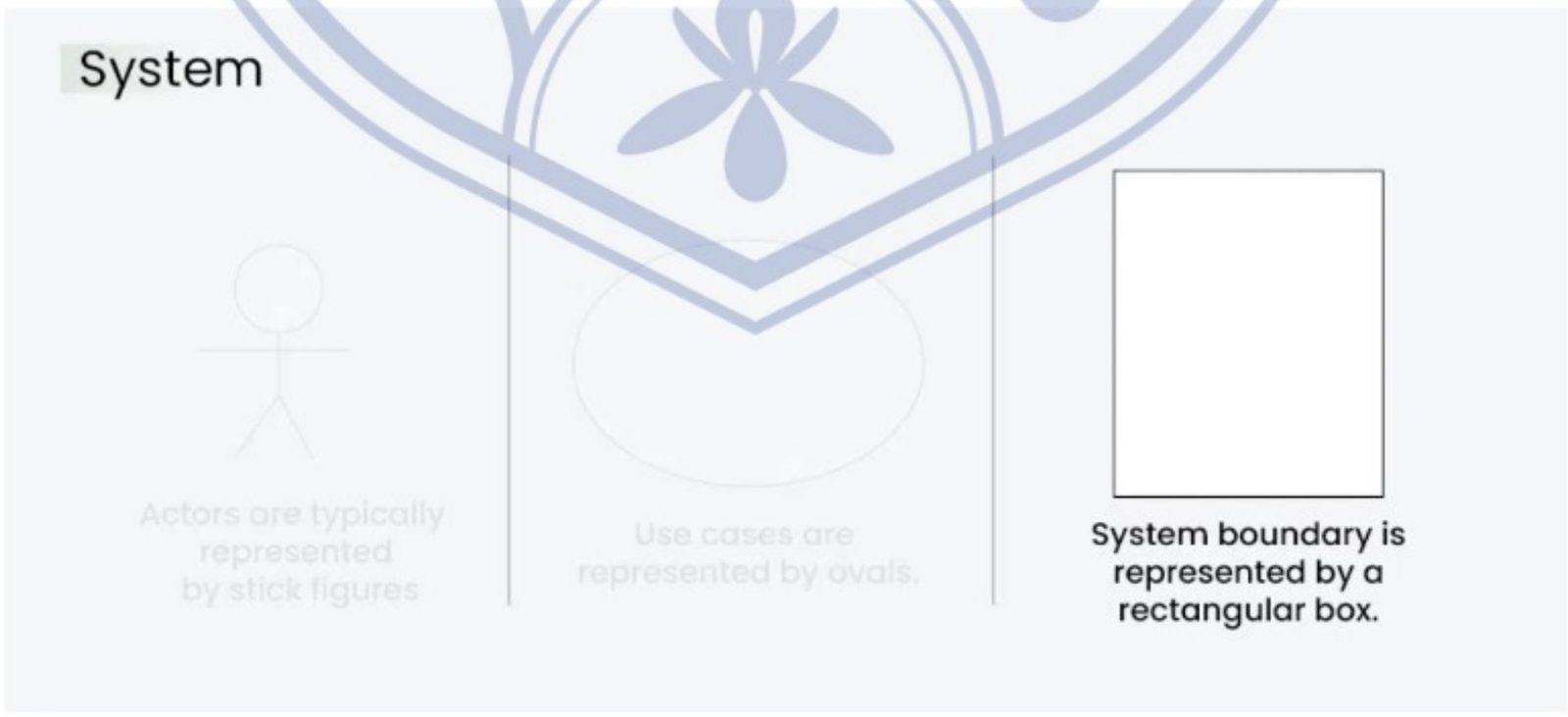
contoh use cases dapat berupa "Melakukan Pemesanan" (*Place Order*), "Melacak Pengiriman" (*Track Delivery*), atau "Memperbarui Informasi Produk" (*Update Product Information*). *Use cases* direpresentasikan dengan bentuk oval.



Gambar 2.4 Simbol Use Cases pada Use Case Diagram

3. System Boundary

System Boundary adalah representasi visual dari ruang lingkup atau limitasi sistem yang sedang Anda modelkan. Hal ini mendefinisikan apa yang berada di dalam sistem dan apa yang berada di luarnya. Batasan ini membantu menetapkan perbedaan yang jelas antara elemen-elemen yang merupakan bagian dari sistem dan elemen-elemen yang bersifat eksternal. Batasan sistem biasanya direpresentasikan oleh kotak persegi panjang yang mengelilingi seluruh use cases dalam sistem tersebut. Tujuan dari batasan sistem adalah untuk secara jelas menguraikan batas-batas sistem, yang mengindikasikan komponen mana yang merupakan internal sistem dan mana yang merupakan aktor atau entitas eksternal yang berinteraksi dengan sistem.



Gambar 2.5 Simbol System Boundary pada Use Case Diagram

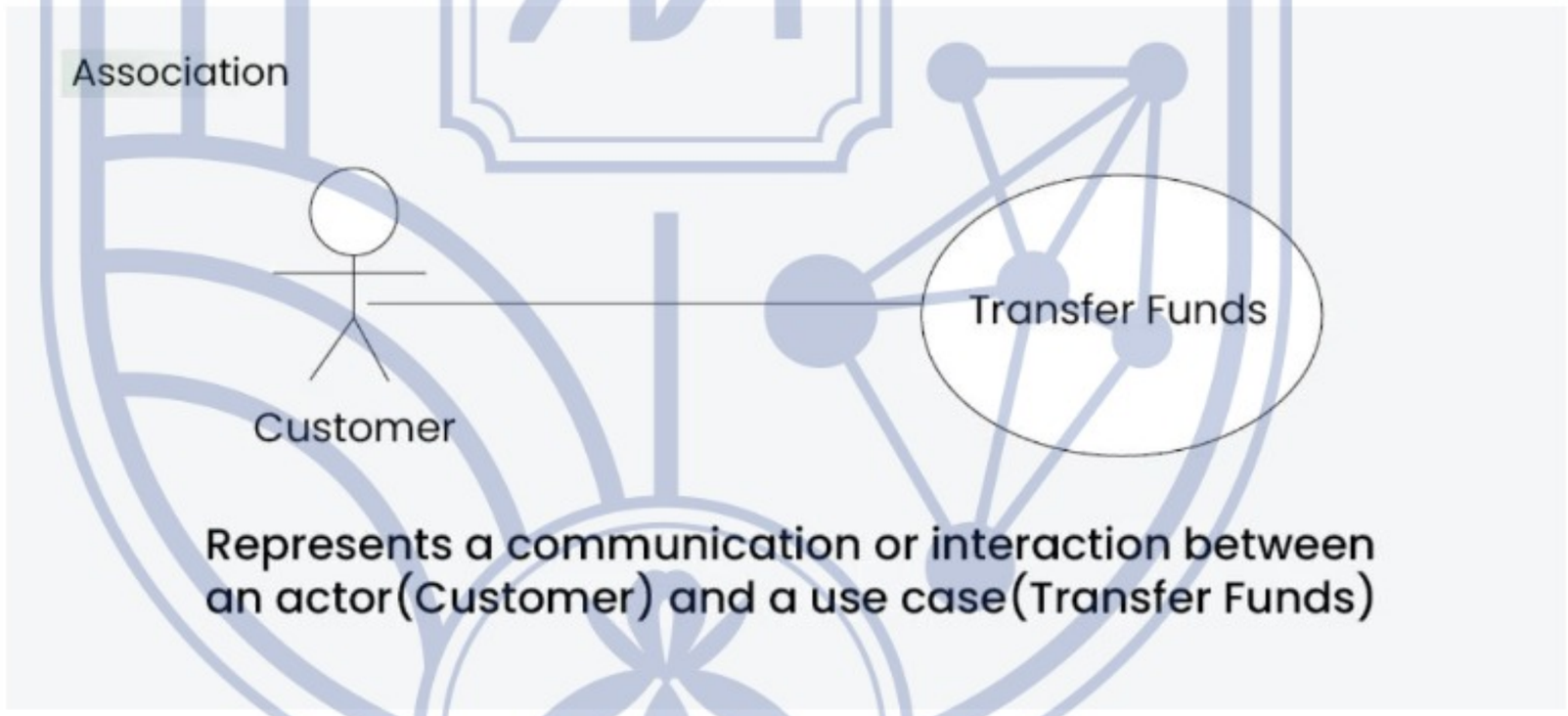
2.4.2.2 Relasi Use Case Diagram

Dalam *Use Case Diagram* Dalam Diagram *Use Case*, relasi memainkan peran krusial dalam menggambarkan interaksi antara aktor dan *use cases*. Relasi ini memberikan pandangan komprehensif mengenai fungsionalitas sistem dan berbagai skenarionya. Berikut adalah jenis-jenis relasi utama beserta contoh penggunaannya.

- Relasi Asosiasi (*Association Relationship*)

  Relasi Asosiasi merepresentasikan komunikasi atau interaksi antara seorang aktor dan sebuah *use case*. Hal ini digambarkan dengan garis yang menghubungkan aktor ke *use case*. Relasi ini menandakan bahwa aktor tersebut terlibat dalam fungsionalitas yang dijelaskan oleh *use case*.

  a. Contoh: Sistem Perbankan Daring
  b. Aktor: Nasabah
  c. Use Case: Transfer Dana
  d. Asosiasi: Sebuah garis yang menghubungkan aktor "Nasabah" ke *use case* "Transfer Dana", yang mengindikasikan keterlibatan nasabah dalam proses transfer dana.



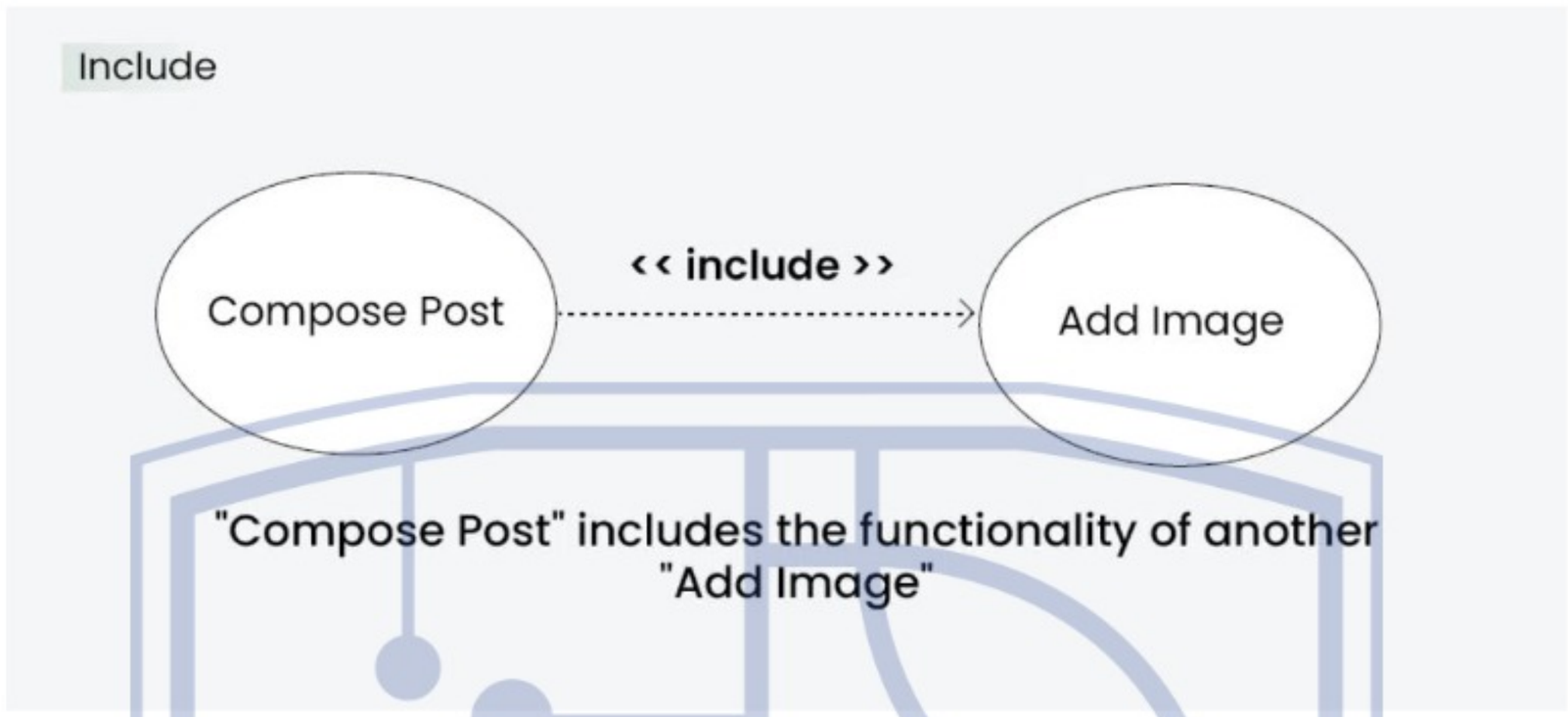
Gambar 2.6 Relasi Association pada Use Case Diagram

- Relasi *Include* (*Include Relationship*)

  Relasi *Include* menunjukkan bahwa sebuah *use case* menyertakan fungsionalitas dari *use case* lain. Hal ini ditandai dengan panah putus-putus yang menunjuk dari *use case* penyerta (induk) ke *use case* yang disertakan. Relasi ini mendukung desain yang modular dan dapat digunakan kembali (*reusable*).

  - Contoh: Sistem Perbankan Daring
  - Aktor: Nasabah
  - Use Case: Transfer Dana

- Asosiasi: Sebuah garis yang menghubungkan aktor "Nasabah" ke *use case* "Transfer Dana", yang mengindikasikan keterlibatan nasabah dalam proses transfer dana.

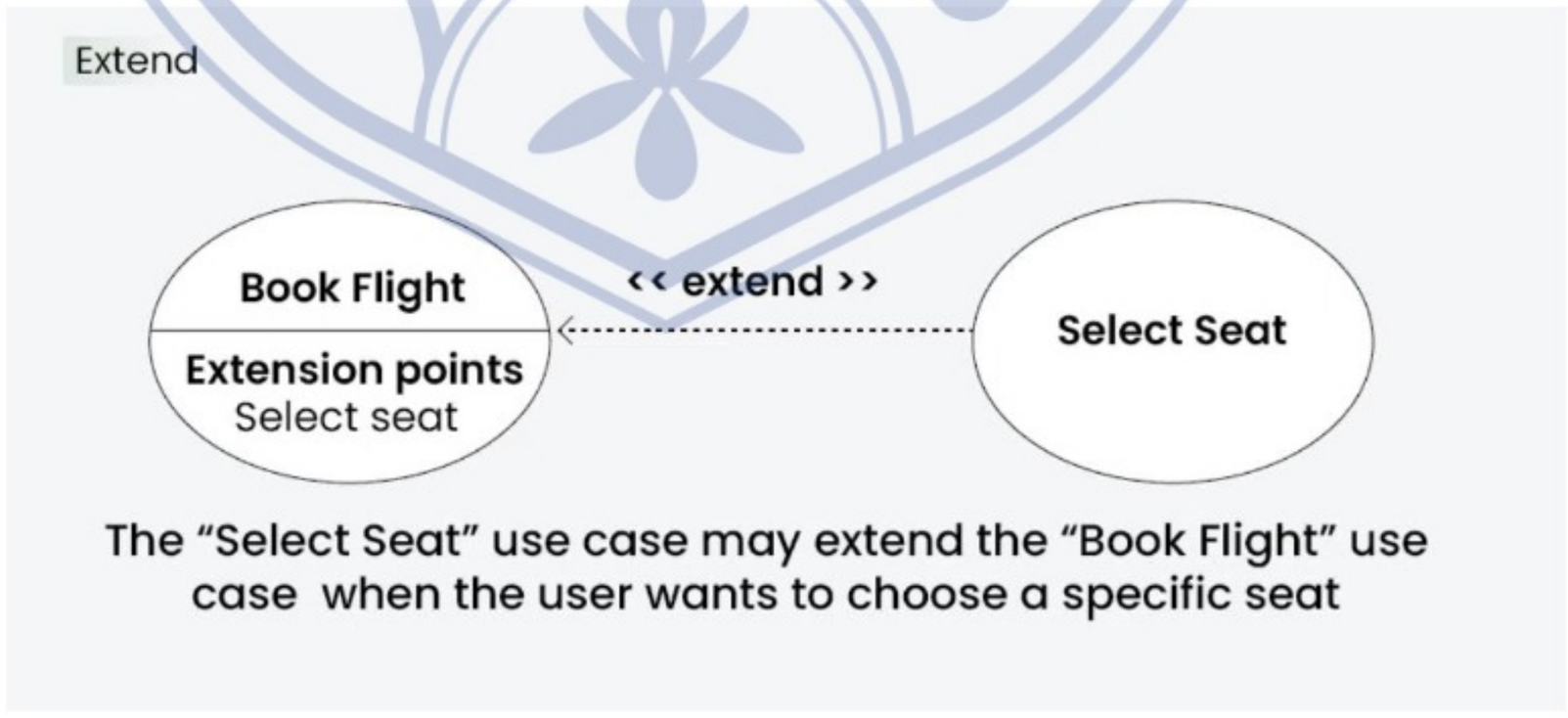


Gambar 2.7 Relasi Include pada Use Case Diagram

- Relasi *Extend* (*Extend Relationship*)

  Relasi *Extend* mengilustrasikan bahwa sebuah *use case* dapat diperluas oleh *use case* lain di bawah kondisi tertentu. Hal ini direpresentasikan oleh panah putus-putus dengan kata kunci "*extend*." Relasi ini berguna untuk menangani perilaku opsional atau pengecualian (*exceptional behavior*).

  1. Contoh: Sistem Pemesanan Penerbangan
  2. Use Cases: Memesan Penerbangan, Memilih Kursi
  3. Relasi Extend: *Use case* "Memilih Kursi" dapat memperluas (*extend*) *use case* "Memesan Penerbangan" ketika pengguna ingin memilih kursi tertentu, namun ini merupakan langkah opsional.



Gambar 2.8 Relasi Extend pada Use Case Diagram

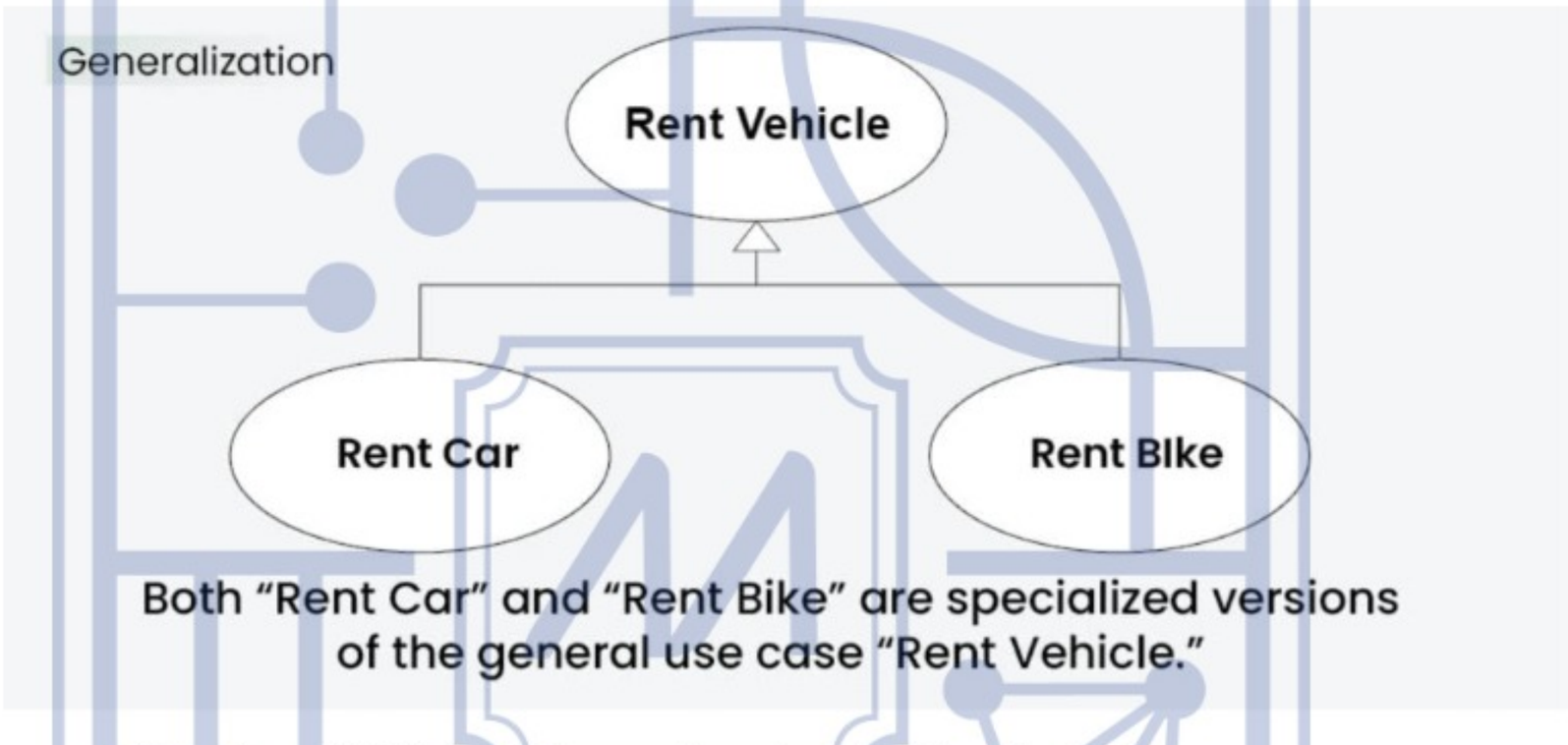
2. Relasi Generalisasi (*Generalization Relationship*)

Relasi Generalisasi menetapkan koneksi "adalah sebuah" (*is-a*) antara dua *use cases*, yang mengindikasikan bahwa satu *use case* merupakan versi spesialisasi dari yang lain. Hal ini direpresentasikan oleh panah yang menunjuk dari *use case* yang terspesialisasi ke *use case* umum.

a. Contoh: Sistem Penyewaan Kendaraan

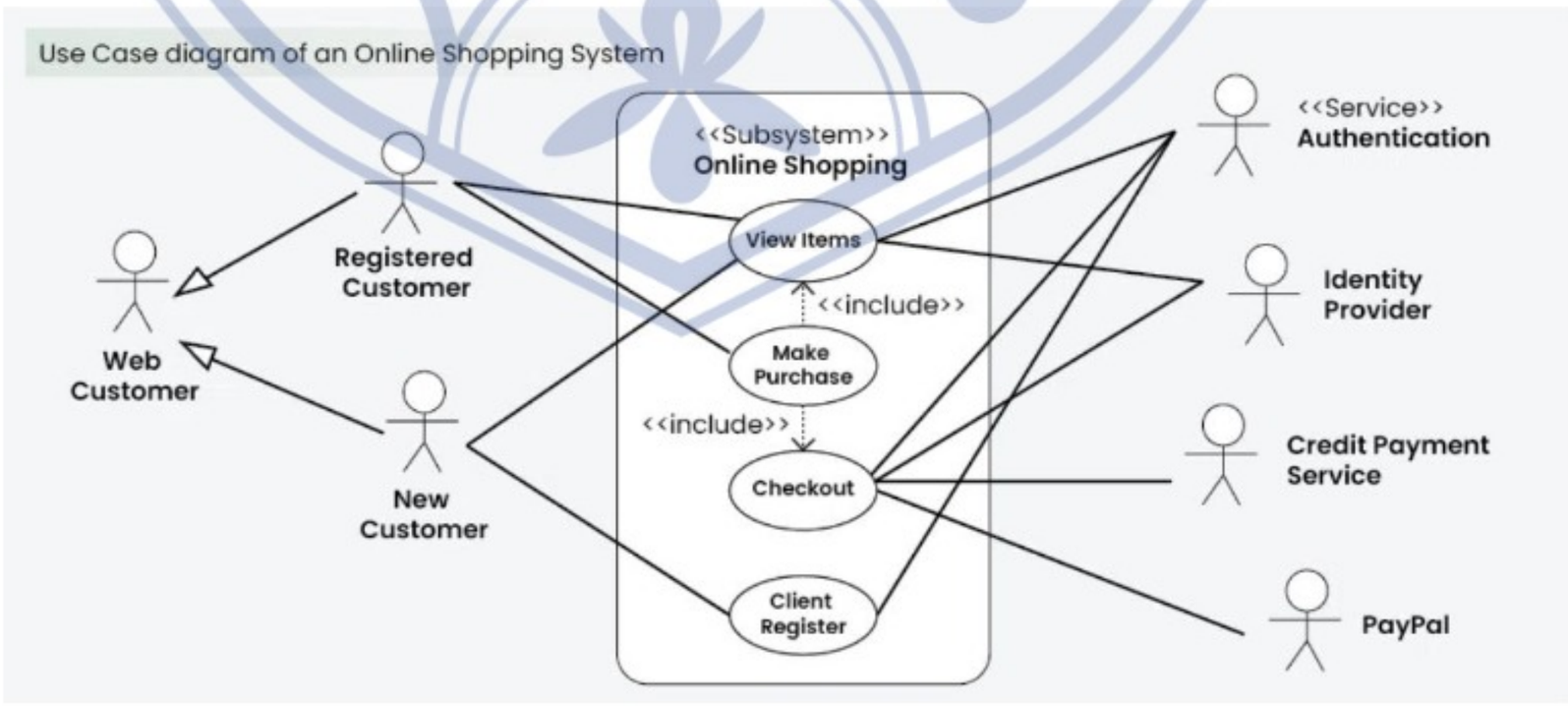
b. Use Cases: Sewa Mobil, Sewa Sepeda

c. Relasi Generalisasi: Baik "Sewa Mobil" maupun "Sewa Sepeda" adalah versi spesialisasi dari *use case* umum "Sewa Kendaraan."



Gambar 2.9 Relasi Generalisasi pada Use Case Diagram

Berikut adalah contoh dari Use Case Diagram untuk Online Shopping System yang berfungsi sebagai artefak utama dalam tahap analisis kebutuhan sistem. Model ini menjembatani pemahaman antara pemangku kepentingan bisnis dan tim teknis dengan menyoroti perilaku sistem dari perspektif pengguna akhir (end-user perspective)



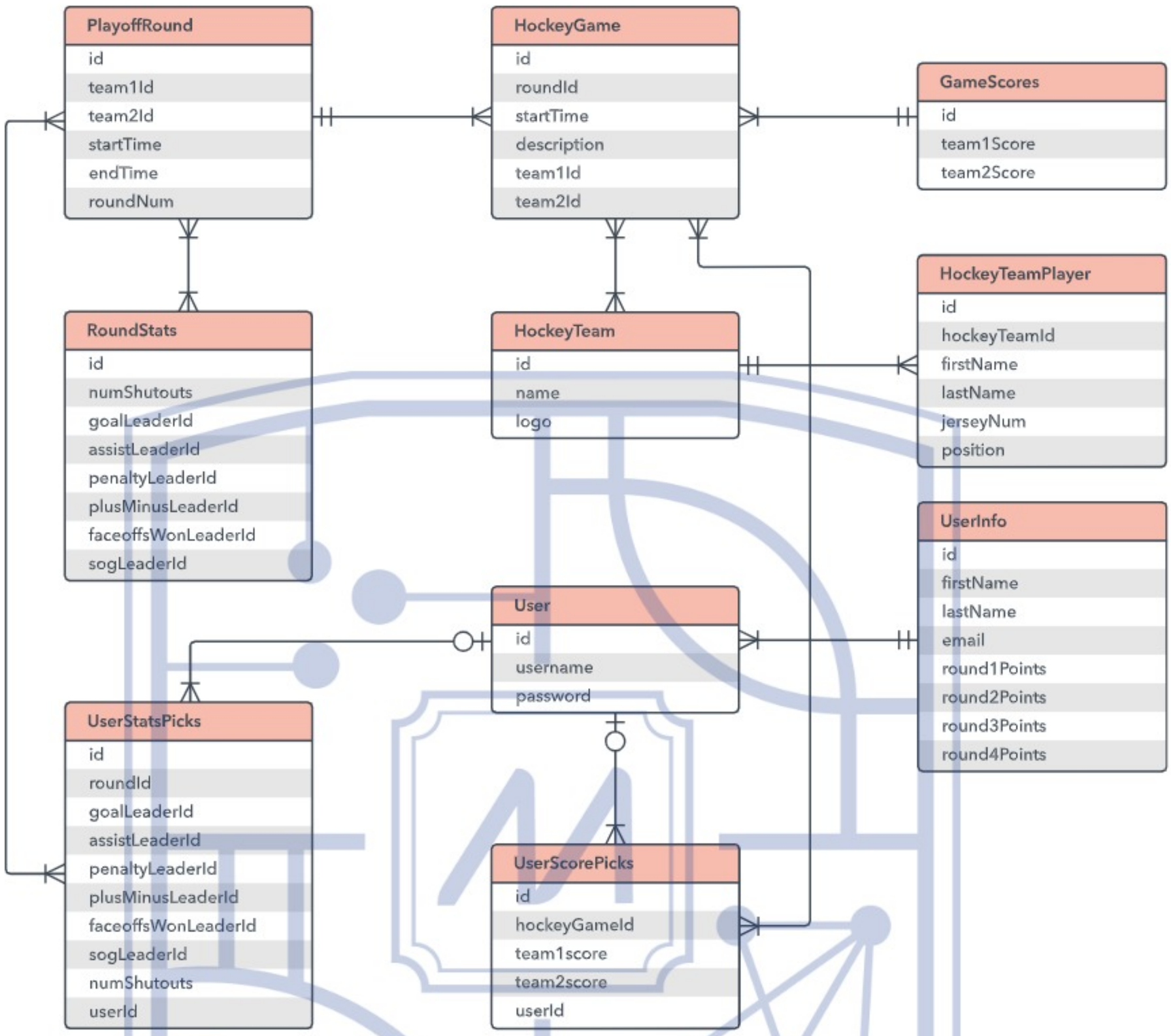
Gambar 2.10 Contoh Use Case Diagram Online Shopping System [15]

2.4.3 Entity Relationship Diagram

Entity Relationship Diagram (ERD) merupakan diagram yang menggambarkan hubungan antar entitas seperti orang, objek, atau konsep dalam suatu sistem. Diagram ini umum digunakan dalam perancangan dan analisis basis data relasional di bidang rekayasa perangkat lunak maupun sistem informasi. ERD memanfaatkan simbol-simbol tertentu seperti persegi panjang, belah ketupat, oval, dan garis untuk menunjukkan keterkaitan antara entitas, relasi, dan atributnya [16].

Sebagai alat pemodelan data, ERD berperan penting dalam menjembatani pemahaman antara kebutuhan konseptual dan implementasi teknis. Melalui representasi visual yang terstruktur, pengembang dapat mengidentifikasi dependensi antar entitas serta memastikan integritas data sebelum sistem dibangun. Dengan demikian, ERD tidak hanya berfungsi sebagai dokumentasi desain, tetapi juga sebagai dasar komunikasi antara analis sistem, pengembang, dan pemangku kepentingan dalam proses perancangan basis data.

Berikut contoh penerapan konsep Entity Relationship Diagram (ERD) yang menggambarkan hubungan logis antar entitas dalam sistem basis data, seperti pengguna, tim, permainan, dan statistik pertandingan. Diagram ini memvisualisasikan keterkaitan data sehingga memudahkan pemahaman, analisis, dan perancangan sistem.

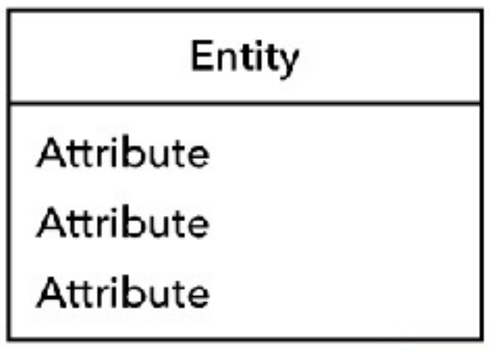
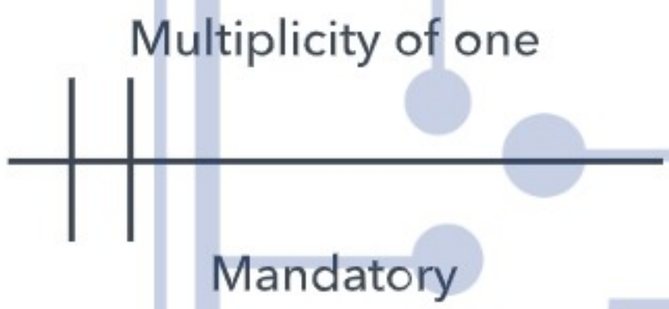
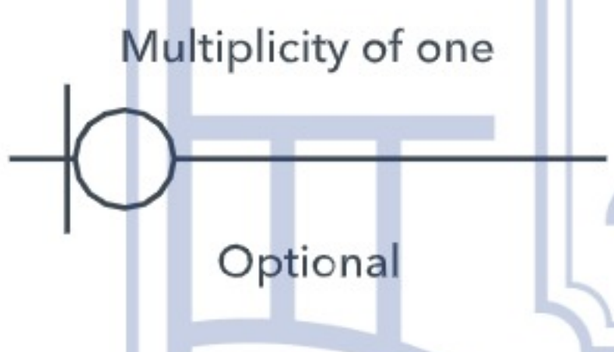
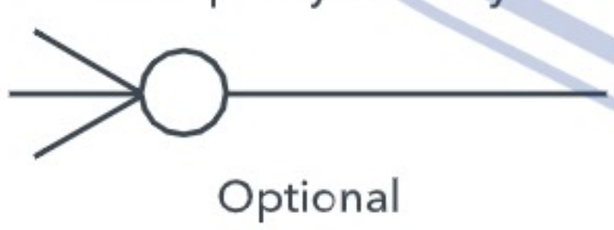


Gambar 2.11 Contoh Entity Relationship Diagram

Sebelum memahami detail simbol yang digunakan dalam ERD, penting untuk mengenali bahwa setiap elemen dalam diagram tersebut memiliki peran konseptual yang saling melengkapi. Hubungan antara entitas, atribut, dan relasi tidak hanya bersifat visual, tetapi juga merepresentasikan struktur logis dari data yang akan diimplementasikan dalam basis data. Dengan memahami keterkaitan ini, perancang sistem dapat memastikan bahwa model yang dibuat benar-benar mencerminkan kebutuhan fungsional dan logika bisnis dari sistem yang dikembangkan.

Setiap simbol dalam ERD memiliki fungsi spesifik untuk memperjelas hubungan dan batas antar entitas, sehingga struktur data dapat disajikan dengan lebih sistematis dan mudah dipahami. Pemahaman terhadap simbol-simbol tersebut membantu perancang sistem merancang basis data secara konsisten dan akurat.

Tabel 2.1 Simbol Entity Relationship Diagram

Simbol	Keterangan
	Simbol Entitas mewakili objek, konsep, atau "sesuatu" di dunia nyata yang datanya penting untuk disimpan oleh sistem.
	Simbol Mandatory One berarti satu instansi entitas wajib terhubung dengan tepat satu instansi entitas lain.
	Simbol Optional One (nol atau satu) berarti satu instansi entitas boleh terhubung dengan paling banyak satu instansi entitas lain.
	Simbol Mandatory Many (satu atau banyak) berarti satu instansi entitas wajib terhubung dengan minimal satu atau lebih instansi entitas lain.
	Simbol Optional Many (nol atau banyak) berarti satu instansi entitas boleh terhubung dengan nol atau lebih instansi entitas lain.

2.4.4 Konsep Desain Antarmuka

2.4.4.1 User Interface (UI)

Antarmuka pengguna (*User Interface*) adalah ruang di mana seorang manusia berinteraksi dengan sebuah teknologi atau produk digital, sering kali melalui layar. Perancang UI menciptakan cara yang cepat dan lugas (*straightforward*) bagi manusia dan

mesin untuk berkomunikasi, menggunakan fitur-fitur interaktif seperti tombol, ikon, menu, navigasi, serta perintah yang dikendalikan oleh suara (*voice-controlled*) dan berbasis gestur (*gesture-based*). Perancang UI menerapkan prinsip-prinsip desain interaksi dan psikologi pengguna untuk menciptakan antarmuka yang ramah pengguna (*user-friendly*), andal, logis, dan menarik (*engaging*).

Desain antarmuka pengguna (*User interface design*) mempertimbangkan keseluruhan tampilan dan nuansa (*look and feel*) dari pengalaman produk digital yang diciptakan, serta menerapkan prinsip-prinsip kebergunaan (usability) dan desain interaksi (interaction design) ke semua fungsi produk dan fitur interaktif. Hal ini membangun koneksi emosional dengan pengguna akhir (end user), melibatkan mereka dengan produk [17].

2.4.4.2 User Experience (UX)

Desain pengalaman pengguna (UX) berfokus pada keseluruhan pengalaman pengguna saat merancang sebuah produk, dengan mempertimbangkan faktor-faktor seperti kemudahan penggunaan (*ease of use*), intuitivitas, dan ekspektasi pengguna.

Desain UX pada intinya adalah tentang keseluruhan pengalaman yang dimiliki pengguna dengan suatu produk. Proses ini melibatkan penelitian dan pengujian mendalam untuk menentukan secara tepat apa yang diharapkan pengguna dari sebuah produk. Perancang UX menggunakan temuan-temuan ini untuk merancang produk yang intuitif dan menyenangkan (*enjoyable*).

Desain UX (*User Experience*) adalah seni sekaligus ilmu pengetahuan, yang berfokus pada penciptaan produk digital yang menarik secara visual, intuitif, dan mudah digunakan. Hal ini berkaitan dengan pemahaman mendalam mengenai kebutuhan, tujuan, dan perilaku pengguna, serta perancangan pengalaman yang berpusat pada mereka.

Perancang UX memprioritaskan pengguna dalam setiap langkah proses desain, membantu tim produk menciptakan solusi yang akan diadopsi, diminati, dan digunakan kembali secara berulang oleh target audiens mereka. Perancang dapat menerapkan strategi UX pada berbagai proyek, meliputi:

2. Desain produk
3. Aplikasi seluler
4. Situs Web
5. Antarmuka perangkat lunak
6. Produk fisik dengan komponen digital [18]

2.4.4.3 User Experience Dalam Konteks Aplikasi Web

Dalam mengembangkan aplikasi web berbasis web, pemastian kualitas User Experience menjadi prioritas utama bagi perancang untuk memastikan bahwa aplikasi yang dibangun dapat digunakan dengan baik oleh pengguna akhir. User Experience mencakup berbagai aspek interaksi antara pengguna dan sistem, menciptakan lingkungan yang nyaman dan efektif melalui kombinasi usability, usefulness, fungsi teknologi, kredibilitas, dan kepuasan emosional. Evaluasi terhadap UX aplikasi web dapat dilakukan melalui berbagai metode yang fokus pada sudut pandang pengguna, seperti System Usability Scale, sehingga hasil pengujian UI lebih sesuai dengan apa yang benar-benar dihadapi dan dirasakan oleh pengguna dalam penggunaan sehari-hari aplikasi tersebut [19].

2.5 Teknologi Pengembangan

2.5.1 Bun

Bun adalah *toolkit* lengkap (*all-in-one*) untuk aplikasi *JavaScript* dan *TypeScript*, yang didistribusikan sebagai satu file *executable* tunggal bernama *bun*. Inti dari *Bun* adalah *Bun runtime*, sebuah *runtime JavaScript* cepat yang dirancang sebagai pengganti langsung (*drop-in replacement*) untuk *Node.js*. *Bun* ditulis dalam bahasa *Zig* dan didukung oleh *JavaScriptCore*, yang secara drastis mengurangi waktu *startup* dan penggunaan memori. Selain *runtime*, *toolkit* ini juga mencakup fungsionalitas lain seperti *Package Manager* yang cepat, *Test Runner* yang kompatibel dengan Jest, dan *Bundler native[20]*.

2.5.2 React.JS

React adalah pustaka (*library*) untuk antarmuka pengguna (*user interface*) web dan *native*. *React* membangun antarmuka pengguna dari bagian-bagian individual yang disebut komponen, yang ditulis dalam *JavaScript*. *React* dirancang agar dapat dengan mulus menggabungkan komponen-komponen yang ditulis oleh orang, tim, dan organisasi yang independen [21].

2.5.3 Next.JS

Next.js adalah sebuah *framework* React untuk membangun aplikasi web *full-stack*. Anda menggunakan Komponen React untuk membangun antarmuka pengguna (*user interface*), dan Next.js menyediakan fitur tambahan serta optimasi. Next.js juga secara otomatis mengonfigurasi alat-alat *lower-level* seperti *bundler* dan *compiler*, memungkinkan Anda untuk fokus membangun produk dan merilisnya dengan cepat. Baik Anda seorang

pengembang individu atau bagian dari tim yang lebih besar, Next.js dapat membantu Anda membangun aplikasi React yang interaktif, dinamis, dan cepat [22].

2.5.4 ElysiaJS

Elysia didefinisikan sebagai sebuah *framework backend* yang dirancang secara ergonomis untuk pengembang (*Humans*), dengan fokus utama pada penyediaan pengalaman pengembang (*Developer Experience*) yang superior. Konsep 'ergonomis' ini diwujudkan melalui sintaks yang familiar dan intuitif, serta jaminan keamanan tipe *end-to-end* (*end-to-end type safety*) yang komprehensif. Fitur keamanan tipe ini memastikan bahwa tipe data tetap konsisten mulai dari basis data, melalui logika *backend*, hingga ke antarmuka *frontend*, sehingga secara drastis mengurangi potensi *bug* dan kesalahan saat *runtime*.

Dibangun di atas *runtime Bun*, Elysia menawarkan kinerja yang sangat cepat. Keunggulan utamanya terletak pada dukungan kelas satu (*first-class support*) untuk TypeScript, yang terintegrasi secara mendalam ke dalam inti *framework*. Elysia juga dirancang dengan integrasi yang matang untuk berbagai layanan esensial, baik itu *tRPC* (untuk komunikasi *API* yang *type-safe* antara *client* dan *server*), *Swagger* (untuk generasi dokumentasi *API* secara otomatis), maupun *WebSocket* (untuk membangun fitur komunikasi *real-time*), menjadikannya pilihan yang tangguh untuk membangun aplikasi web modern yang skalabel dan mudah dipelihara [23].

2.5.5 PostgreSQL

PostgreSQL adalah sistem basis data objek-relasional (*object-relational database system*) *open source* yang andal, yang menggunakan dan memperluas bahasa SQL yang dikombinasikan dengan banyak fitur untuk menyimpan dan menskalakan beban kerja data yang paling rumit secara aman. Asal-usul PostgreSQL dimulai pada tahun 1986 sebagai bagian dari proyek POSTGRES di Universitas California di Berkeley dan telah memiliki hampir 40 tahun pengembangan aktif pada platform intinya.

PostgreSQL telah mendapatkan reputasi yang kuat atas arsitekturnya yang teruji, keandalan (*reliability*), integritas data (*data integrity*), rangkaian fitur yang tangguh, ekstensibilitas (kemampuan untuk diperluas), dan dedikasi komunitas *open source* di balik perangkat lunak tersebut untuk secara konsisten memberikan solusi yang berkinerja tinggi dan inovatif. PostgreSQL berjalan di semua sistem operasi utama, telah patuh-ACID (*ACID-compliant*) sejak tahun 2001, dan memiliki *add-on* (pengaya) yang kuat seperti ekstensi basis data geospasial PostGIS yang populer. Tidak mengherankan jika PostgreSQL telah menjadi basis data relasional *open source* pilihan bagi banyak orang dan organisasi [24].

2.5.6 Nginx

Nginx adalah web server berbasis open-source yang berfungsi ganda sebagai proxy IMAP/POP3, dikenal karena performanya yang tinggi, stabil, dan hemat sumber daya. Nginx juga mendukung fitur reverse proxy untuk berbagai protokol seperti HTTP, Memcached, PHP-FPM, SCGI, dan uwsgi, serta mendukung streaming video HTTP dan HTTP/2 gateway [25].

Berbeda dengan web server lainnya, Nginx bekerja secara asynchronous dan event-driven, di mana setiap permintaan baru dijalankan pada proses yang sudah ada dan terus bercabang menjadi sub-worker hingga batas kapasitasnya. Dengan mekanisme ini, satu worker Nginx mampu menangani hingga 1.024 koneksi sekaligus, menjadikannya pilihan utama untuk server e-commerce, situs dengan traffic tinggi, dan cloud storage [25].

2.6 Perangkat Lunak Pendukung

2.6.1 Visual Studio Code

Visual Studio Code (VS Code) merupakan sebuah editor kode sumber (*source code editor*) yang ringan namun memiliki kapabilitas tinggi (*powerful*), yang beroperasi pada lingkungan desktop serta tersedia untuk sistem operasi Windows, macOS, dan Linux. Aplikasi ini dilengkapi dengan dukungan terintegrasi (*built-in*) untuk JavaScript, TypeScript, dan Node.js, serta memiliki ekosistem ekstensi yang kaya untuk mendukung bahasa pemrograman lain (seperti C++, C#, Java, Python, PHP, Go) maupun berbagai *runtimes* atau lingkungan waktu eksekusi (seperti .NET dan Unity) [26].

2.6.2 Figma

Figma merupakan sebuah platform perancangan (desain) kolaboratif terkemuka yang dirancang untuk memfasilitasi keseluruhan siklus hidup pengembangan produk digital melalui suatu ekosistem terpadu berbasis *cloud*, yang memungkinkan tim untuk bekerja secara simultan dan efisien. Fungsionalitas intinya mengintegrasikan berbagai tahapan krusial, mencakup proses perancangan *mockup*, pembuatan prototipe (*prototyping*) interaktif untuk pengujian pemangku kepentingan, hingga mempermudah proses penyerahan aset desain (*developer handoff*) kepada tim pengembang untuk implementasi. Keunggulan utama platform ini terletak pada kemampuannya mengakselerasi alur kerja secara signifikan dengan memusatkan seluruh proses pengumpulan umpan balik (*feedback*) dan iterasi desain dalam satu lingkungan kerja terpusat, sehingga mengurangi friksi dan mempercepat waktu peluncuran produk ke pasar [27].