# BAB 2

# LANDASAN TEORI

## Internet

Jaringan adalah kumpulan komputer dan perangkat yang terhubung bersama, seringkali secara nirkabel, melalui perangkat komunikasi dan media transmisi (Cashman, 2012, p. 8). Ketika komputer terhubung ke jaringan, hal tersebut dinamakan *online*. Jaringan memungkinkan komputer berbagi sumber daya, seperti perangkat keras, perangkat lunak, data, dan informasi. Internet merupakan kumpulan jaringan di seluruh dunia yang menghubungkan jutaan bisnis, lembaga pemerintah, lembaga pendidikan, dan individu seperti pada gambar 2.1 .



**Gambar 2. 1** Kumpulan Jaringan di Seluruh Dunia(Cashman, 2012:8)

Lebih dari satu miliar orang di seluruh dunia menggunakan Internet setiap hari untuk berbagai tujuan, seperti berkomunikasi dan bertemu dengan orang lain, mengakses banyak informasi, berita, dan temuan penelitian, berbelanja barang dan jasa, mengambil kelas *online*; mengakses media hiburan, seperti *game online*, musik, video, buku, dan majalah, mengunduh musik dan video, berbagi informasi dan masih banyak lagi.

Pada awalnya , internet berakar pada proyek jaringan yang dimulai oleh agen Departemen Pertahanan AS. Tujuannya adalah :

1. Untuk membangun jaringan yang memungkinkan para ilmuwan di lokasi yang berbeda untuk berbagi informasi dan bekerja sama dalam proyek militer dan ilmiah dan

2. Dapat berfungsi bahkan jika bagian dari jaringan itu dinonaktifkan atau dihancurkan oleh bencana seperti nuklir menyerang.

Jaringan tersebut, yang disebut ARPANET, mewabah pada September 1969, menghubungkan para peneliti ilmiah dan akademis di seluruh Amerika Serikat. Jaringan asli terdiri dari empat komputer utama, masing-masing terletak di *University of California* di Los Angeles, Universitas California di Santa Barbara, Stanford Research Institute, dan Universitas Utah. Setiap komputer berfungsi sebagai tuan rumah di jaringan. Suatu *host* atau *server* adalah setiap komputer yang menyediakan layanan dan koneksi ke komputer lain dalam suatu jaringan. Pada 1984, jaringan itu memiliki lebih dari 1.000 komputer individu yang terhubung sebagai tuan rumah. Hari ini, lebih dari 350 juta penghuni terhubung ke jaringan ini, yang kemudian dikenal sebagai Internet (Cashman, 2012, p. 51).

## World Wide Web

*World Wide Web* adalah sistem dokumen hiperteks yang saling terkait yang diakses melalui Internet. Dengan *browser web*, seseorang dapat melihat halaman web yang mungkin berisi teks, gambar, video, multimedia lain dan menavigasi di antaranya melalui *hyperlink* (Pranay, 2015, p. 134). Setiap dokumen elektronik di *Web*, yang disebut halaman *Web*, dapat berisi teks, grafik, audio, dan video. Selain itu, halaman web biasanya memiliki koneksi *built-in* ke dokumen lain.

Situs *Web* adalah kumpulan halaman *Web* terkait dan item terkait, seperti dokumen dan gambar, yang disimpan di *server Web*. *Server Web* adalah komputer yang mengirim halaman *Web* yang diminta ke komputer. Beberapa pakar industri menggunakan istilah *Web 2.0* untuk merujuk ke situs *Web* yang memungkinkan pengguna untuk memodifikasi konten situs *web*, menyediakan sarana bagi pengguna untuk berbagi informasi pribadi, dan memiliki perangkat lunak aplikasi yang dibangun ke dalam situs untuk digunakan para pengunjung.

Pengunjung yang membuka suatu *Web* yang bersifat statis, maka pengunjung tersebut akan melihat isi konten yang selalu sama. Sedangkan perbedaan terlihat pada halaman *Web* yang bersifat dinamis karena pengunjung dapat menyesuaikan beberapa atau semua konten yang terlihat.

## Web Browser

*Browser Web* adalah perangkat lunak aplikasi yang memungkinkan pengguna untuk mengakses dan melihat halaman *Web* (Cashman, 2012, p. 54). Untuk menjelajah *Web*, Anda memerlukan komputer yang terhubung ke Internet dan yang memiliki *browser Web*.

*Browser Web* yang lebih banyak digunakan untuk komputer pribadi adalah *Internet Explorer, Netscape, Firefox, Opera,* dan *Safari*. Peramban mengambil dan menampilkan halaman *Web* awal, biassanya disebut halaman beranda *browser.*

## Web Address

Alamat *Web* terdiri dari protokol, nama domain, dan jalur ke halaman *Web* atau lokasi tertentu di halaman *Web*. Banyak alamat halaman *Web* dimulai dengan http: //. *Http*, yang merupakan singkatan dari *Hypertext Transfer Protocol*, adalah seperangkat aturan yang menentukan cara transfer halaman di Internet.

Bagian pertama dari nama domain mengidentifikasi jenis *server* Internet. Misalnya, *www* menunjukkan *server Web*.Untuk membantu meminimalkan kesalahan, banyak *browser* dan situs *web* tidak mengharuskan Anda memasukkan bagian http: // dan *www* dari alamat *Web* (Cashman, 2012, p. 56).

## Web Application

*Web Application* adalah sebuah aplikasi yang tersimpan didalam *web server* dan dapat diakses melalui *browser* (Cashman, 2012, p. 65). Biasanya seorang *user* akan menggunakan *web app*langsung dari *website*. *Web application host* menyimpan data-data pengguna dan informasi pada *server*-nya.

Pengguna sering berinteraksi dengan perangkat lunak berbasis *Web*, kadang-kadang disebut aplikasi *Web*, langsung di situs *Web host*. Namun, beberapa situs *web* mengharuskan *user* mengunduh perangkat lunak ke komputer atau perangkat lokal. Situs perangkat lunak berbasis *web* sering menyimpan data dan informasi pengguna di *server* mereka. Situs perangkat lunak berbasis *web* dapat memberi pengguna pilihan untuk menyimpan data secara lokal di komputer pribadi atau perangkat seluler mereka sendiri. Misalnya, pengolah kata, *e-mail*, manajer informasi pribadi, perangkat lunak, pengeditan foto, pengarang halaman *web*, persiapan pajak, *clip art* / galeri gambar, referensi, pendidikan, dan lainnya.

## Mobile Application

*Mobile application* adalah proses dimana pengembangan aplikasi untuk perangkat genggam seperti telepon genggam atau PDA. Selama manufaktur aplikasi mobile sudah ada atau bisa di download oleh pemakai sesuai dengan platform perangkat lunaknya. Menurut (Lee, Schneider & Schall ,2004), berikut beberapa penjelasan mengenai *mobile application* :

1. Masyarakat menganggap pengembangan mobile application lebih mudah untuk perangat *mobile* padahal kenyataanya lebih sulit dari yang dibayangkan.
2. Pengembangan aplikasi untuk *mobile* tergolong lebih cepat. Ada gagasan bahwa mengembangkan aplikasi pada perangkat mobile lebih cepat. Kenyataannya semua aplikasi bergantung pada kerumitan aplikasi itu sendiri.
3. Pengembangan aplikasi *mobile* tergolong tidak lebih murah dimana dibandingkan dengan biaya *PC* *pocket* atau *tablet* *PC* dengan komputer desktop yang terhubung

***2.3 Native***

Aplikasi *native* adalah aplikasi yang dibangun dengan bahasa pemrograman yang spesifik untuk platform tertentu. Contoh populernya yakni penggunaan bahasa pemrograman *Objective*-*C* atau *Swift* untuk platform *iOS* (*Apple*). Adapun platform *Android* yang menggunakan bahasa pemrograman *Java*.

Sedangkan Aplikasi hybrid adalah aplikasi web yang ditransformasikan menjadi kode *native* pada platform seperti *iOS* atau *Android*. Aplikasi *hybrid* biasanya menggunakan browser untuk mengijinkan aplikasi web mengakses berbagai fitur di *device* *mobile* seperti *Push* *Notification*, *Contacts*, atau *Offline* *Data* *Storage*. Beberapa tools untuk mengembangkan aplikasi *hybrid* antara lain *Phonegap*, *Rubymotion* dan lain-lain. (Eisenman, 2016, p. 234).  
  
Mengapa *Native* App*?*

1. *Native* *Code* masih lebih cepat daripada *HTML* dan *JavaScript*. Kualitas user *experience* yang cepat dan dapat diandalkan
2. Standar *WebGL* membantu aplikasi *browser* dan *hybrid* untuk aplikasi *gaming* memenuhi kinerja, tetapi masih asli memiliki keunggulan.
3. *Native* *SDK* memungkinkan untuk mengakses fitur perangkat tanpa melibatkan kompleksitas *plugin* asli dan fitur perangkat baru akan tersedia bersama dengan *SDK*. Contohnya dapat mengakses semua fitur perangkat seperti mikrofon, kamera, daftar kontak, dan lainnya.
4. Tidak banyak ketergantungan pada perpustakaan dan *platform* *open* *source* seperti *Cordova* dan *Ionic*.

# *2.4 Android*

**2.4.1Sejarah *Android***

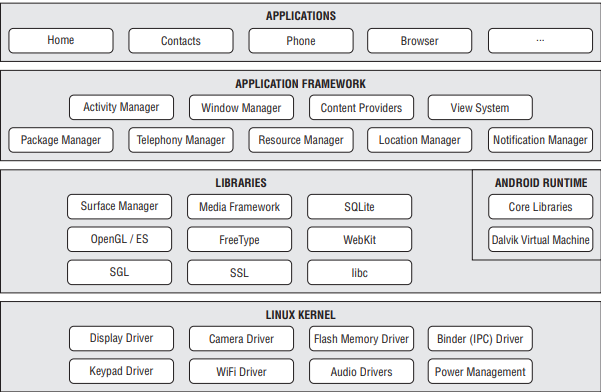
*Android* adalah sistem operasi seluler yang didasarkan pada modifikasi *Linux* (Wei, 2011 , p. 2). Awalnya dikembangkan oleh *startup* dengan nama yang sama, *Android*, Inc. Pada tahun 2005, *Google* membeli *Android* dan mengambil alih pekerjaan pengembangannya (serta tim pengembangannya).

*Google* ingin *Android* terbuka dan gratis, maka dari itu sebagian besar kode *Android* dirilis di bawah lisensi *Apache open source*, yang berarti bahwa siapa pun yang ingin menggunakan *Android* dapat melakukannya dengan mengunduh kode sumber *Android* secara lengkap

Berikut karakteristik *Android* yang menjadi salah satu ciri dari sistem operasi ini:

1. *Complete* : Sistem operasi *Android* adalah sistem operasi yang aman dan mudah diperbarui dengan *framework* yang lengkap dan tampilan yang mudah dipahami.
2. *Open*: Keseluruhan sistem operasi *Android* telah dikembangkan dan disediakan dibawah ketentuan lisensi *open source Apache*.
3. *Free* : Sistem operasi *Android* tidak mengenakan biaya atas lisensi atau keanggotaan untuk mengembangkan aplikasi. Kode sumber dan alat pengembangan perangkat lunak (*Software Development Kit* / *SDK*) .

**2.4.2 Arsitektur Android**



**Gambar 2.2** Arsitektur Android (Lee, 2011:3)

OS Android secara kasar dibagi menjadi lima bagian dalam empat lapisan utama:

1. *Kernel Linux* merupakan *kernel* yang berbasis Android. Lapisan ini berisi semua driver perangkat tingkat rendah untuk berbagai komponen perangkat keras dari perangkat Android.
2. *Libraries* berisi semua kode yang menyediakan fitur utama dari OS Android. Sebagai contoh, pustaka *SQLite* menyediakan dukungan basis data sehingga aplikasi dapat menggunakannya untuk penyimpanan data. Perpustakaan *WebKit* menyediakan fungsionalitas untuk penjelajahan *web*.
3. Android *runtime* menyediakan satu set pustaka inti yang memungkinkan pengembang untuk menulis aplikasi Android menggunakan bahasa pemrograman *Java. Runtime Android* juga termasuk mesin virtual *Dalvik*, yang memungkinkan setiap aplikasi Android untuk berjalan dalam prosesnya sendiri, dengan mesin virtual *Dalvik* sendiri (aplikasi Android dikompilasi ke dalam eksekusi *Dalvik*). *Dalvik* adalah mesin virtual khusus yang dirancang khusus untuk Android dan dioptimalkan untuk perangkat seluler bertenaga baterai dengan memori dan *CPU* terbatas.
4. Kerangka aplikasi mengekspos berbagai kemampuan OS Android ke pengembang aplikasi sehingga mereka dapat menggunakannya dalam aplikasi mereka.
5. Aplikasi yang dikirimkan bersama perangkat Android (seperti Telepon, Kontak, Browser, dll.), Serta aplikasi yang diunduh dan diinstal dari *Android* *Market*. Aplikasi apa pun yang dikembangkan berada di lapisan ini.

***2.5 React Native***

*React Native adalah framework* yang memungkinkan *web developer* untuk membuat aplikasi *mobile* yang kuat dengan menggunakan bahasa pemrograman *JavaScript* (Eisenman, 2016, p. 23). *React native* menyediakan pengembangan *mobile* yang lebih cepat, dan berbagi kode yang lebih efisien di *iOS, Android, dan Web.*

*React Native* didasarkan pada *React*, perpustakaan *JavaScript Facebook* untuk membangun *user interface* , tetapi *React Native* bukan menargetkan *browser*, melainkan menargetkan *platform mobile.* Dengan kata lain, pengembang *web* sekarang dapat membuat aplikasi *mobile* yang terlihat dan terasa nyata, karena sebagian besar kode yang ditulis dapat dibagikan antar-*platform*. *React Native* mempermudah pengembangan aplikasi secara bersamaan baik untuk *Android maupun iOS*. Metode yang ada untuk mengembangkan aplikasi *mobile* menggunakan kombinasi *JavaScript*, *HTML*, dan *CSS*, biasanya dirender menggunakan *WebView*.

**2.6 Lelang**

Jual beli merupakan kegiatan yang berlaku di dunia ekonomi dan perdagangan. Dalam bidang ekonomi dan usaha, jual beli dianggap sebagai bagian terpenting dalam sebuah aktivitas usaha karena langsung berhubungan dengan konsumen. Jual beli adalah sebuah proses pemindahan hak milik berupa barang atau harta kepada pihak lain dengan menggunakan uang sebagai salah satu alat tukarnya. Salah satu bentuk jual beli yaitu dengan cara pelelangan. Lelang adalah suatu bentuk penjualan barang yang dilakukan secara terbuka untuk umum dengan harga penawaran yang semakin meningkat atau menurun untuk mencapai harga tertinggi, yang diajukan secara tertulis maupun secara lisan, sebelumnya didahului pemberitahuan tentang akan adanya pelelangan atau penjualan barang (Ressha, 2018, p. 1) . Pelelangan yang telah dilaksanakan sesuai dengan ketentuan yang berlaku tidak dapat dibatalkan. Pembatalan pelelangan hanya dapat dilakukan sebelum pelaksanaan lelang.

Pada saat tahapan pelaksanaan lelang penentuan harga limit oleh penjual dan diserahkan kepada pejabat lelang sebelum lelang dimulai. Cara penawaran harus diumumkan di depan calon pembeli (media, selebaran, internet). Penawaran yang diajukan tidak dapat diubah atau dibatalkan oleh peserta lelang. Dalam pelaksanaan lelang pada awalnya dilaksanakannya pengumuman atas penawaran barang yang dijadikan sebagai objek lelang secara terbuka kepada calon peserta lelang pada waktu yang bersamaan. Pada saat hari yang telah ditentukan lelang tersebut dilaksanakan oleh para peserta lelang.

## 2.7 Scrum Model

Menurut Schwaber and Sutherland (2011,p5), *Scrum* adalah suatu metodologi atau kerangka kerja yang terstruktur untuk mendukung pengembangan produk yang kompleks. Scrum terdiri dari sebuah tim yang memiliki peran dan tugas masing-masing. Setiap komponen dalam kerangka melayani tujuan tertentu dan sangat penting untuk kesuksesan penggunaan scrum.

Beberapa istilah yang akan dipakai dalam *Scrum* yaitu:

1. *Product Backlog*: Merupakan suatu dokumen formal yang berisikan *Requirement*, keinginan dan ekspektasi dari *Product Owner* dan Stakeholders.
2. *Scrum Board*: Merupakan suatu papan tulis dimana Tugas yang harus dikerjakan akan ditulis atau ditempelkan *Post-It*.
3. *Scrum Master*: Sama seperti *Agile Coach*,  *Scrum Master* bertugas sebagai jembatan antara *stakeholders* dan team dan bertanggung jawab untuk menjaga kebiasaan dan performa team dalam menjalankan *Scrum*.
4. *Sprint*: Merupakan suatu periode waktu (biasanya 2 minggu) dimana *Developer* harus dapat menyelesaikan tugas dan merilis produk saat *Sprint* tersebut berakhir.
5. *Sprint Backlog*: Merupakan suatu dokumen yang berisikan tugas apa yang harus dikerjakan dalam minggu *Sprint*

Keuntungan dari menggunakan sistem *Scrum* adalah sebagai berikut:

1. Flexibel : Dengan memvisualisasikan perkerjaan yang dilakukan, kita dapat mengetahui bottleneck dan kesalahan yang dapat dengan mudah diubah.
2. Pembetulan yang mudah : Karena *Scrum* menggunakan waktu *Sprint* yang singkat dan memungkinkan feedbacks secara konstan, tim dapat dengan mudah melakukan Pembetulan, dan Perubahan pada setiap *Sprint*.
3. Safe & Good *Product*: *Scrum* memungkinkan tim untuk membuat *Product*yang baik dikarenakan pengulangan proses testing, review yang konstan, dan *Daily Scrum* setiap hari.

Ada tiga peran berbeda didalam metode *Scrum*, yaitu:

1. *Master Scrum* memastikan prosedur diikuti, memastikan semua berjalan lancar, dan melindungi tim dari gangguan. Master Scrum berbeda dari manajer proyek tradisional dalam banyak hal, termasuk peran ini tidak memberikan arahan sehari-hari kepada tim dan tidak memberikan tugas kepada individu.
2. *Product Owner* (Pemilik Produk), biasanya merupakan orang yang dianggap paling penting dari sebuah proyek. Bagian dari tanggung jawab pemilik produk adalah memiliki visi tentang apa yang ingin dia buat dan menyampaikan visi tersebut kepada tim *Scrum*. Tugas utama Pemilik Produk adalah untuk menjadi nilai bagi *stakeholder* atau pemegang saham.
3. Tim *Scrum* . Sebuah tim *Scrum* adalah tim yang bisa mengatur pekerjaan mereka sendiri dan merupakan sebuah tim yang lintas fungsional. Member tim akan melakukan analisis, implementasi, perancangan, pengujian, dan lain-lain. Meskipun individu dapat bergabung dengan tim dengan berbagai jabatan, di Scrum, judul tersebut tidak signifikan. Metodologi scrum menyatakan bahwa setiap orang berkontribusi dengan cara apa pun yang mereka bisa untuk menyelesaikan pekerjaan pada setiap sprint. Individu dengan demikian akan menghabiskan sebagian besar (dan terkadang semua) waktu mereka bekerja dalam disiplin apa pun yang mereka ketahui, baik itu analisis, desain, pengembangan, tes dan lain lain.

Langkah-langkah implementasi *Scrum:*

1. Tentukan tim *Scrum*

Tim terdiri dari 5-9 anggota. Semua anggota ini memiliki kombinasi kompetensi dan dapat mencakup untuk menjadi developer, penguji, pendukung, perancang, analisis bisnis, dan lain-lain.

1. Tentukan lama *sprint*

*Sprint* adalah tenggang waktu yang berlangsung antara 7 dan 30 hari, dan biasanya tetap sama panjang selama proyek berlangsung. Tentukan tenggang waktunya dalam sebuah meeting perencanaan dan tim tersebut harus berkomitmen untuk menyelesaikan pekerjaan ini. Pada akhir sprint, sebaiknya mengadakan sebuah review atau ulasan (bisa berupa meeting) dengan demonstrasi hasil kerja.

1. Tunjuk *Scrum Master*

Scrum Master adalah seorang katalisator untuk sebuah kelompok scrum. Mereka memastikan bahwa kelompok scrum bekerja secara efektif dan progresif. Jika terjadi hambatan, master Scrum akan menindaklanjuti dan menyelesaikan masalah untuk tim tersebut.

Master Scrum memang bisa dianggap sebagai manajer proyek untuk tim tersebut, kecuali orang tersebut tidak boleh mendikte apa yang tim kerjakan dan seharusnya tidak terlibat dalam pengelolaan mikro sekalipun. Master Scrum akan membantu tim dalam merencanakan pekerjaan sprint yang akan datang.

1. Tunjuk Pemilik Product ( *Product Owner* )

Pemilik Produk harus merupakan seseorang yang dapat bertanggung jawab memastikan tim menghasilkan sebuah produk yang bisa dipresentasikan dan pasarkan ke bisnis, klien atau siapapun yang menginginkan hasil proyek tersebut (pembeli akhir). Pemilik Produk biasanya menuliskan persyaratan sehubungan dengan apa yang diinginkan dari produk tersebut dalam bentuk sebuah cerita, kemudian memprioritaskan item-item dalam proses pembuatan, dan memasukkannya ke backlog.

1. Buat Backlog Product Awal

Produk backlog adalah daftar keinginan yang berupa semua cerita pengguna yang diharapkan akan dibuat dan diselesaikan dalam proyek tersebut. Cerita yang paling penting harus berada di urutan teratas daftar, jadi keseluruhan simpanan tersusun secara teratur berdasarkan urutan kepentingan cerita.

Sebuah simpanan biasanya berisi 2 jenis item pekerjaan:

**Epics** – Cerita tingkat tinggi yang sketsanya sangat kasar tanpa banyak detail.

**Stories** – Persyaratan lebih rinci untuk apa yang harus dilakukan (mungkin dilakukan).

Sebuah epics biasanya dapat dibagi menjadi beberapa cerita.

Sebuah cerita biasanya akan dipecah menjadi beberapa tugas terpisah sehingga tim dapat bekerja dan melaporkan progressnya nanti. Stories juga bisa memiliki beberapa tipe seperti development, bug, tugas, dan lain lain. Stories baru dapat ditulis dan ditambahkan ke backlog produk kapan saja dan oleh siapa saja.

Jika sebuah epic meningkat dalam prioritas, sebaiknya masukkan detail yang lebih rinci agar tim dapat mulai mengerjakannya. Pemilik Produk bebas untuk memprioritaskan kembali backlog sesuai dengan keinginan dia.

1. Rencanakan dan mulai sprint

Berdasarkan prioritas backlog, tim sekarang mengambil item dari daftar (biasanya dari paling atas). Tim lalu melakukan brainstorming dan memutuskan apa dan berapa banyak yang bisa mereka selesaikan dalam sprint mendatang. Ini disebut rapat perencanaan sprint. Begitu tim setuju, sprint dimulai dan tim dapat mulai mengerjakan proyeknya.

1. Tutup arus dan mulai Sprint berikutnya

Bila akhir tenggat waktu sudah tercapai dan semua pekerjaan yang direncanakan sudah lengkap dilakukan, tergantung pada tim untuk memutuskan apakah pekerjaan yang tersisa harus ditransfer ke sprint berikutnya atau dimasukkan kembali ke dalam backlog.

Setiap sebuah tim menyelesaikan tugasnya, tim disarankan untuk melakukan retrospektif di mana mereka mendiskusikan apa yang berjalan dengan baik dan apa yang bisa diperbaiki untuk sprint berikutnya. Setelah itu, pertemuan perencanaan sprint untuk sprint berikutnya dimulai dan prosesnya diulang.

Tidak ada batasan untuk jumlah sprint kecuali jika ditetapkan oleh deadline (berdasarkan budget atau waktu) atau seluruh backlog selesai. Jika tidak satu pun kriteria ini terpenuhi, sprint terus berlanjut tanpa batas waktu. (Natashya, 2019)

**2.8 *Object Oriented Programming ( OOP )***

Pemrograman berorientasi objek atau *object oriented programming* (OOP) merupakan suatu pendekatan pemrograman yang menggunakan *object* dan *class* (Retroningsih, 2017, p. 96). *Object Oriented Programming* memberikan kemudahan dalam pembuatan sebuah program. Manfaat pengembangan program menggunakan *Object Oriented Programming* antara lain :

1) *Reusability*, dimana kode yang dibuat dapat digunakan kembali

2) *Extensibility*, dimana pemrogram dapat membuat metode baru atau mengubah yang sudah ada sesuai yang diinginkan tanpa harus membuat kode dari awal

3) *Maintainability*, dimana kode yang sudah dibuat lebih mudah untuk dikelola apabila aplikasi yang dibuat berskala besar yang memungkinkan adanya *error* dalam pengembangannya hal tersebut dapat diatasi dengan OOP karena pemrograman OOP sudah menggunakan konsep modularitas

***2.9 Unified Modeling Language ( UML )***

Banyak model grafis yang digunakan dalam pengembangan sistem digambar sesuai dengan notasi yang ditentukan oleh *Unified Modeling Language (UML). UML* adalah standar konstruksi model dan notasi yang didefinisikan oleh *Object Management Group (OMG),* organisasi standar untuk pengembangan sistem (Satzinger, 2012, p. 46). Dengan menggunakan *UML*, analis maupun pengguna akhir dapat menggambarkan dan memahami berbagai diagram khusus yang digunakan dalam proyek pengembangan sistem. Beberapa *UML* yang sering digunakan untuk pengembangan dan perancangan perangkat lunak adalah *use case diagram, activity diagram, sequence diagram, dan class diagram.*

## Use Case Diagram

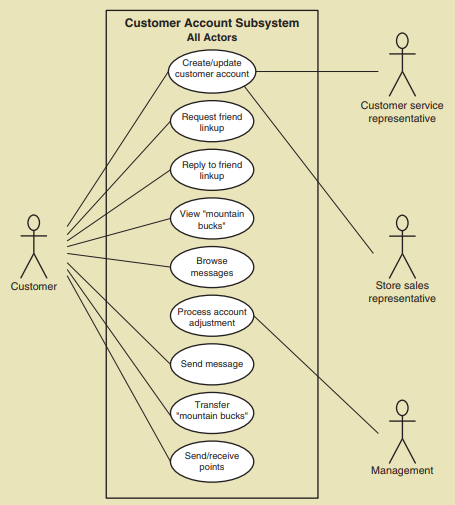
**2.9.1.1 Penjelasan *Use Case Diagram***

*Use case Diagram* merupakan suatu aktivitas yang dilakukan oleh sistem, biasanya merupakan sebuah respon permintaan dari pengguna sistem atau *user* terhadap sistem yang digunakan (Satzinger, 2012, p. 78).

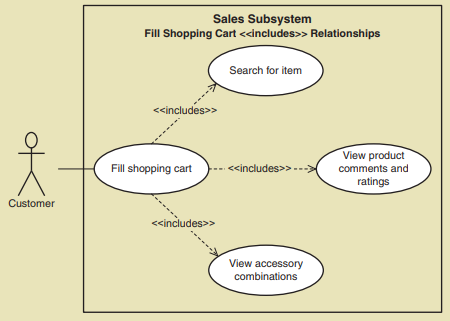
**2.9.1.2 Komponen *Use Case Diagram***

**Table 2.1** Komponen *Use Case Diagram*

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | Seorang aktor selalu di luar batas *boundary sistem* tetapi dapat menjadi bagian dari sistem. Terkadang, aktor dapat digunakan menjadi sistem atau perangkat lain yang menerima layanan dari sistem. Aktor merupakan *user* yang berinteraksi dengan *use case* |
|  | Association merupakan penhubung antara *user* dengan *use case* |
|  | *Use Case* mengidentifikasi dan menggambarkan fungsi-fungsi sistem atau proses yang dilakukan actor |
|  | *Boundary System* merupakan perbatasan antara aplikasi dan actor |
|  | *Include* merupakan relasi *use case* tambahanke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya |



**Gambar 2.3** Contoh *use case diagram* (Satzinger, 2012:82)

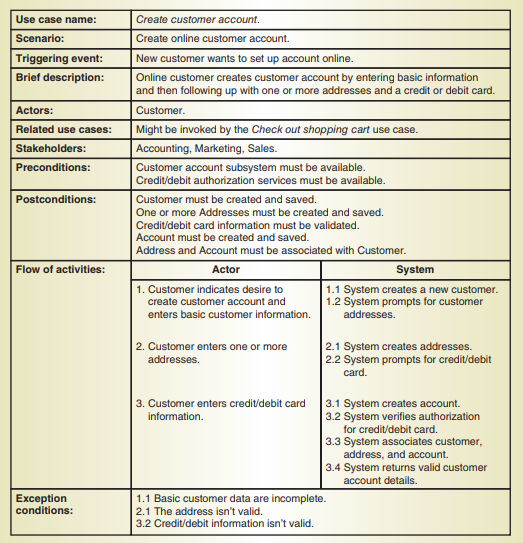


**Gambar 2.4** Contoh *use case diagram* dengan *include*

(Satzinger, 2012:84)

## Use Case Narrative / Description

*Use case narrative* memberikan gambaran umum tentang semua *use case* untuk suatu sistem (Satzinger, 2012, p. 121). Informasi terperinci tentang setiap *use case* dijelaskan dengan deskripsi *use cas*e. Deskripsi *use case* dan mendeskripsikan detail pemrosesan untuk *use case*.



**Gambar 2.5** Contoh *Use Case Narrative*  (Satzinger, 2012:123)

## Activity Diagram

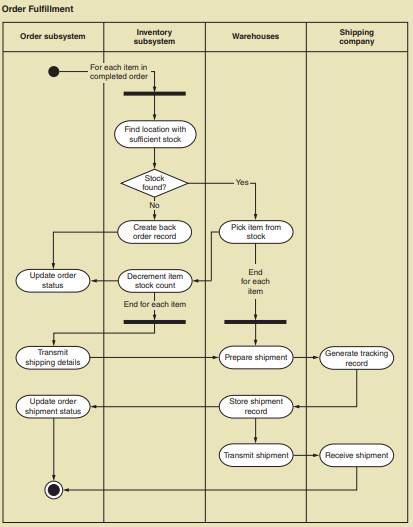
**2.9.3.1 Penjelasan *Activity Diagram***

Diagram aktivitas menggambarkan berbagai aktivitas pengguna (atau sistem), orang yang melakukan setiap kegiatan, dan urutan dari kegiatan proses (Satzinger, 2012, p. 57)

**2.9.3.2 Komponen *Activity Diagram***

**Table 2.2** Komponen *Activity Diagram*

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | Titik Mulai / *Initial Node* |
|  | Titik Akhir / *Final Node* |
|  | Aktivitas / *Actions* yang dilakukan |
|  | *Split* merupakan percabangan dari sebuah *flow* menjadi dua atau lebih *flow* yang berjalan secara parallel |
|  | *Join* merupakan proses penggabungan *split* menjadi *flow* tunggal |
|  | Kondisi yang membutuhkan pilihan *actions* yang akan dilakukan selanjutnya (*Decision node*) |
|  | Menunjukkan alur dari activity diagram (*Flow*) |



**Gambar 2.6** Contoh *Activity Diagram*  (Satzinger, 2012:59)

## Sequence Diagram

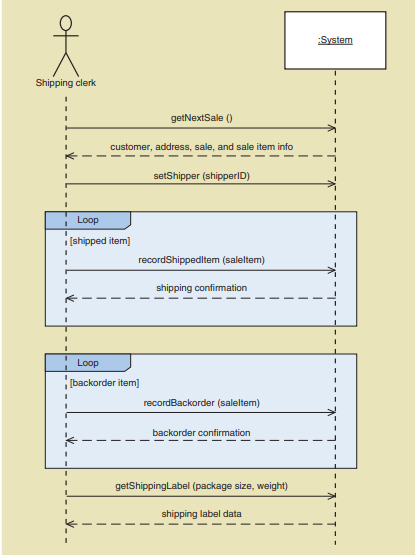
* + - 1. **Penjelasan *Sequence diagram***

*Sequence diagram* menggambarkan bagaimana objek-objek berinteraksi satu sama lain berdasarkan urutan waktu dengan pesan atau umpan balik ketika sebuah *use case* dijalankan (Satzinger, 2012, p. 126).

**2.9.4.2 Komponen *Sequence diagram***

**Table 2.3** Komponen *Sequence Diagram*

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | *Actor*  merupakan *user* yang berinteraksi dengan *object* |
| *\* | *Object* merupakan sebuah turunan daripada *class* yang didalamnya tertulis nama *object* diawali dengan titik koma |
|  | *Lifeline* merupakan garis putus-putus yang menggambarkan basis waktu suatu *object* |
|  | *Message* menggambarkan suatu komunikasi antara *object – object* |
|  | F*rame* digunakan sebagai penanda area pada diagram yang mengalami perulangan (*looping*), seleksi (*alternative*), atau memiliki aturan tertentu (*option* |
|  | Output message merupakan  Tanda panah mendatar dan putus-putus, setiap *behavior* harus mengembalikan sesuatu, paling tidak pesan *true* atau *false* yang menunjukan apakah *behavior* tersebut berhasil |
| NOTE UML | Note merupakan penanda untuk menjelaskan suatu proses dan bersifat opsional |



**Gambar 2.7** Contoh *Sequence Diagram* (Satzinger, 2012:13) .

## 2.9.5 Class Diagram

**2.9.5.1 Penjelasan *Class Diagram***

Diagram ­­­kelas *UML* digunakan untuk menunjukkan kelas objek untuk suatu sistem (Satzinger, 2012, p. 101).

**2.9.5.2 Komponen *Class Diagram***

Pada diagram kelas, persegi panjang mewakili kelas, dan garis yang menghubungkan persegi panjang menunjukkan asosiasi di antara kelas. Nama kelas dan nama atribut menggunakan notasi *camelback*, di mana kata-kata ditulis tanpa spasi atau garis bawah. Nama kelas dimulai dengan huruf kapital; nama atribut dimulai dengan huruf kecil

**Table 2.4** Komponen *Class Diagram*

|  |  |
| --- | --- |
| **Simbol** | **Keterangan** |
|  | *Class* merupakan blok-blok pembangun dalam pemrograman berorientasi objek yang terdiri dari 3 bagian atas yaitu nama *class,* tengah yaitu atribut dan akhir yaitu *method* |
|  | *Association* menggambarkan  *relationship* antara 2 *class* |
|  | *Composition* merupakan sebuah *relationship* yang digunakan untuk menggambarkan ketergantungan antar satu *class* dengan *class* lainnya |
|  | *Dependency* merupakan *relationship* yang digunakan untuk menggambarkan operasi suatu *class* yang menggunakan *class* lain |
|  | *Inheritance* memungkinkan *subclass* untuk memakai karakteristik *superclass* mereka. |

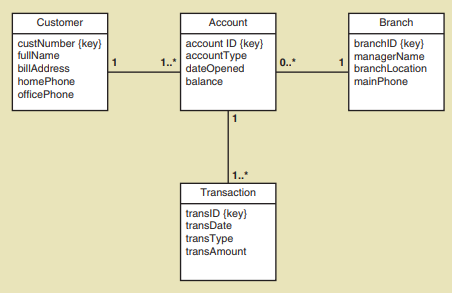
*Visibility* menunjukkan apakah objek lain dapat mengakses atribut secara langsung (Satzinger, 2012, p. 310)*. UML* memiliki jenis visibilitas berikut:

1. Elemen publik dapat dilihat oleh semua elemen konten namespace yang memilikinya. Visibilitas publik diwakili oleh simbol '+' .
2. Elemen package dimiliki oleh *namespace* yang bukan *package* , dan dapat terlihat oleh elemen-elemen yang berada dalam *package* dan *namespace* yang sama. Hanya elemen yang tidak dimiliki oleh *package* yang dapat ditandai sebagai visibilitas *package* . Setiap visibilitas elemen adalah *Visible* untuk semua elemen dalam paket. Di luar *package* terdekat, visibilitas elemen menjadi tidak terlihat. Visibilitas *package* diwakili oleh simbol '~' .
3. Elemen protected terlihat oleh elemen yang memiliki hubungan generalisasi ke *namespace* yang memiliki. Visibilitas *protected* diwakili oleh simbol '#' .
4. Elemen private hanya terlihat di dalam *namespace* yang memilikinya. Visibilitas *private* diwakili oleh simbol '-' .

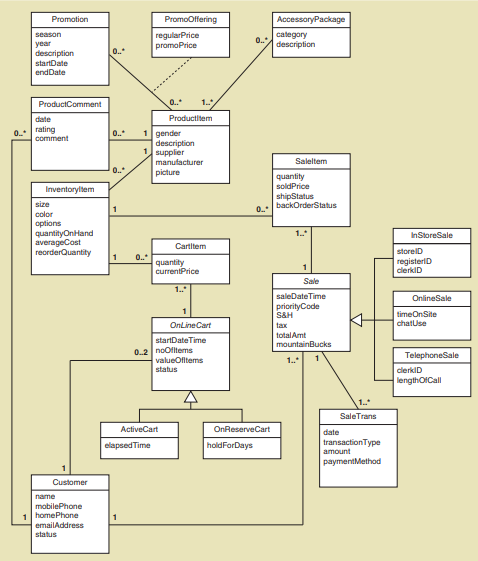
Saat membuat *class diagram* terdapat beberapa hubungan antar *class* yang terbuat dari sebuah garis lurus yang dinamakan Assosiasi, Berikut adalah notasi dari *multiplicity* class diagram:

**Tabel 2.5** Notasi UML untuk *Multiplicity* dari Asosiasi (Satzinger, 2012:102)

|  |  |
| --- | --- |
| ***Multiplicity*** | **Deskripsi** |
| 0..1 | Nol atau satu |
| 1 | Hanya satu |
| 0..\* atau \* | Nol atau lebih |
| 1..\* | Satu atau lebih |



**Gambar 2.8** Contoh *Class Diagram*  (Satzinger, 2012:103)

******

**Gambar 2.9** Contoh *Class Diagram*  (Satzinger, 2012:109)

## 2.10 Interaksi Manusia dan Komputer

Desain antarmuka pengguna menciptakan media komunikasi yang efektif antara manusia dan komputer. Mengikuti serangkaian prinsip desain antarmuka, desain mengidentifikasi objek dan tindakan antarmuka dan kemudian membuat tata letak layar yang membentuk dasar untuk *prototype* antarmuka pengguna.

Hal ini sangat penting karena jika perangkat lunak sulit digunakan, atau jika itu menghambat upaya *user* untuk mencapai tujuannya, *user* tidak akan menyukainya, terlepas dari konten yang diberikannya, atau fungsi yang ditawarkannya. Antarmuka harus benar karena ini membentuk persepsi pengguna terhadap perangkat lunak. Interaksi Manusia dan Komputer (IMK) memfokuskan desain sistem pada pengguna atau biasa disebut dengan *user center design* (UCD). UCD yaitu filosofi perancangan yang menempatkan pengguna sebagai pusat dari proses pengembangan system (Saifulloh, 2015, p.55). Dan prinsip-prinsip dalam merancang *user interface* adalah sebagai berikut :

a. *User Familiarity* / mudah dikenali

b. *Consistency* / Konsistensi

c. *Minimal Surprise* / Tidak membuat pengguna terkejut

d. *Recoverability* / Pemulihan

e. *User guidance* / bantuan

Perhatian utama dari interaksi manusia dan komputer adalah mengkombinasikan metode pengumpulan data dan intelektual *framework* dengan pengalaman secara psikologis dengan kekuatan dan penggunaan secara luas dari pengembangan ilmu komputer.

**2.10.1 Eight Golden Rules**

Untuk merancang tampilan antarmuka, terdapat 8 aturan yang harus dipenuhi (Shneiderman, 2016, p. 74), yaitu :

1. **Mengusahakan konsistensi**

Konsistensi dibutuhkan antar halaman dalam satu aplikasi ataupun antara aplikasi yang masih berhubungan. Hal ini bertujuan agar *user* tetap dapat mengenali halaman yang dilihat tersebut masih dalam lingkup atau masih memiliki hubungan dengan aplikasi yang digunakan. Dengan demikian akan membuat *user* nyaman dalam mengeksplorasi aplikasi tanpa takut berpindah aplikasi.

1. **Menyediakan penggunaan yang bersifat universal**

Dalam merancang antarmuka aplikasi, seorang *interface designer* harus memperhitungkan jenis variasi *user* nya. Baik itu dari segi latar belakang budaya dan bahasa, juga variasi tingkat pemahaman *user* terhadap aplikasi. Pada poin ini , yang lebih sering dipikirkan adalah perbedaan kebudayaan *user*, sehingga aplikasi harus dirancang dalam berbagai macam bahasa. Akan lebih efektif jika *universal usability* diterapkan pada variasi tingkat pemahaman *user* terhadap aplikasi.

1. **Memberikan umpan balik yang informatif**

Umpan balik yang informatif tidak harus selalu dengan jawaban dari aplikasi ke *user*, tetapi dapat berupa perubahan antarmuka setiap *user* melakukan aksi, dengan demikian *user* paham bahwa aksinya sudah direspon oleh aplikasi.

1. **Merancang dialog untuk menghasilkan keadaan akhir**

Untuk poin ini sebenarnya termasuk dari poin nomor 4 yaitu dengan menyampaikan bahwa proses yang dijalankan oleh *user* sudah selesai, *user* paham bahwa dia tidak perlu menunggu apakah masih akan ada tahapan lain setelah menyelesaikan suatu proses

1. **Mencegah terjadinya kesalahan**

Poin ini bertujuan untuk menjaga agar *user* tidak melakukan kesalahan dalam menjalankan proses. Sangat diperlukan supaya *user* tidak merasa jenuh dalam mencoba menggunakan aplikasi, karena tidak bisa menemukan format / aksi yang tepat pada saat mencoba suatu fungsi.

1. **Memungkinan pengembalian tindakan dengan mudah**

Contoh paling nyata dari poin ini adalah pada aplikasi *online shop*, *user* dapat mencoba melakukan pembelian, tetapi pada saat selesai memilih barang, *user* dapat membatalkan barang yang ingin dibeli. Dengan demikian *user* merasa nyaman saat mencoba untuk melakukan eksplorasi pada aplikasi, karena barang yang dicoba untuk dibeli tidak langsung terproses , tetapi *user* dapat melakukan pembatalan barang yang sudah dipilih.

1. **Mendukung pusat kendali**

Poin ini terutama sangat disukai oleh *user* yang sudah terbiasa menggunakan aplikasi, karena biasanya *user* ingin memiliki tampilan yang bisa diatur oleh *user* sendiri sesuai preferensi dari *user*. Hal ini dapat meningkatkan kepuasan *user* terhadap aplikasi yang sangat mempengaruhi *UX* terhadap aplikasi tersebut.

1. **Mengurangi beban memori jangka pendek**

Dengan menerapkan poin ini, maka *user* tidak perlu mengingat data yang harus dimasukkan ke sistem. Karena data yang harus dimasukkan , sudah disediakan oleh sistem. Misalnya pada *online shop*, *user* memilih barang yang diinginkan, dan dari situ sistem langsung menerima input kode barang yang diinginkan oleh *user* untuk diproses pada proses pembayaran nantinya.

**2.10.2 Lima Faktor Manusia Terukur**

Perancangan *interface* sebaiknya juga mengikutsertakan evaluasi terhadap lima (5)  faktor terukur dari manusia sebagai berikut (Shneiderman, 2016, p. 162):

1. **Waktu untuk belajar**

Waktu untuk belajar merupakan berapa lama seorang *user* dapat mempelajari fungsi-fungsi di dalam sebuah aplikasi hingga pada akhirnya dapat menggunakan aplikasi tersebut dengan baik.

1. **Kecepatan performa**

Kecepatan performa merupakan berapa lama suatu fungsi di dalam aplikasi tersebut dilakukan.

1. **Tingkat *error* yang dilakukan pengguna**

Tingkat *error* yang dilakukan pengguna merupakan berapa banyak jenis *error* yang dilakukan oleh *user* di dalam melakukan serangkaian tugas

1. **Daya ingat pengguna**

Daya ingat *user* merupakan berapa lama *user* dapat mempertahankan ingatan dan pengetahuannya setelah beberapa jam, hari, atau bahkan minggu.

1. **Kepuasan subjektif**

Kepuasan subjektif merupakan seberapa puas *user* atas berbagai aspek dari suatu sistem

***2.11 Visual Studio Code***

*Visual Studio Code* hadir dengan tata letak yang sederhana dan intuitif yang memaksimalkan ruang yang disediakan untuk editor dan meninggalkan ruang yang cukup untuk menelusuri dan mengakses konteks penuh folder atau proyek. *User Interface*-nya dibagi menjadi empat area, yaitu:

1. *Editor* : Area utama untuk mengedit *file*. User dapat membuka hinga tiga editor secara berdampingan.
2. *Side Bar* : Berisi tampulan berbeda seperti *Explorer* untuk membantu user saat mengerjakan proyek.
3. *Status Bar*: Bertujuan untuk menunjukan ifnormasi tentang proyek yang dibuka dan file yang *user* edit.
4. *View Bar :*Berada di sisi paling kiri, memungkinkan user beralih di antara tampilan dan memberi konteks tambahan Indikator spesifik, seperti jumlah perubahan keluar ketika *Git* diaktifkan.

Setiap kali user membuka *Visual Studio Code*, kode itu terbuka dalam kondisi yang sama seperti ketika user terakhir kali menutupnya. *Folder*, tata letak dan *file* yang dibuka dipertahankan.

***2.12 ASP .NET***

*ASP.NET* merupakan *toolkit* pemrograman *web Microsoft* yang merupakan bagian dari *.NET* dengan tujuan untuk membantu *developer* membangun berbagai aplikasi. *Developer* dapat menggunakan *framework .NET* untuk membangun aplikasi *windows*, membuat *service*, dan bahkan *line command*. Salah satu bahasa pemrograman yang dapat digunakan adalah *C#* (Spaanjaars, 2014, p. 2).

## 2.13 Internet Information Service ( IIS )

*Internet Information Service* merupakan komponen yang digunakan untuk mengelola *web,Ghoper, File Transfer Protocol (FTP), dan NNTP.* Komponen *IIS* terdapat pada sistem operasi *Windows NT, Windows 2000, Windows XP dan Windows Server 2003* (Iqbal, 2010, p. 3)*.*

## 2.14 HyperText Markup Language ( HTML )

*Hypertext Markup Language* atau yang biasa disebut dengan *HTML* merupakan bahasa yang paling umum digunakan dalam pembangunan *web* (Duckett, 2009, p. 3).

*HyperText Markup Language (HTML)* memungkinkan pengguna untuk menghasilkan halaman *web* yang berisi dengan teks, grafik, video, penunjuk ke halaman *web* lain, dan lainnya. *HTML* merupakan bahasa untuk mendeskripsikan bagaimana bentuk format dari sebuah dokumen. Dengan *HTML*, hal utama yang perlu diingat adalah menambahkan *tag* untuk menunjukkan struktur dokumen (bukan bagaimana yang ingin disajikan),misalnya bagian dokumen mana yang merupakan *heading*, bagian mana yang merupakan paragraf, apa yang termasuk dalam tabel, dan sebagainya. *Browser* seperti *Internet Explorer, Firefox, dan Safari* menggunakan *markup* ini untuk membantu menyajikan teks mirip dengan aplikasi pengolah kata seperti *Word* karena judul utama lebih besar dari teks di paragraf, ada ruang di atas dan di bawah setiap paragraf ,dan lainnya. Sesuai dengan namanya, *markup language* yang berarti bahasa penandaan, *HTML* digunakan untuk menandai secara khusus bagian – bagian yang terdapat dalam sebuah halaman *web*. Penandaan pada setiap bagian menggunakan *tag*.

# *2.15 CSS*

*CSS (Cascading Style Sheet)* berguna untuk merubah tampilan *HTML* menjadi lebih menarik dan untuk menentukan bagimana elemen-elemen yang didalam *HTML* akan ditampilkan, seperti menentukan posisi, merubah warna teks, *background* atau yang lainnya (Duckett, 2009, p. 244).

# *2.16 JavaScript*

*JavaScript* merupakan sebuah bahasa pemrograman yang berinteraksi dengan semua aspek *web browser* ,termasuk isi atau konten dari halaman *web* tersebut (Duckett, 2009, p. 481). *JavaScript* dapat ditambahkan di dalam dokumen *HTML* dan dijalankan diatas *web browser*. *JavaScript* diimplementasikan di dalam dokumen *HTML* menggunakan elemen. Beberapa keuntungan *javascript* adalah:

1. Dapat melakukan validasi *input* dari pengguna sebelum dikirim ke *server* ,artinya dapat mengurangi beban server.
2. Pengunjung *web* tidak perlu menunggu sebuah halaman *web* untuk dimuat untuk memastikan apakah mereka lupa memasukkan sesuatu atau tidak
3. Menginformasikan kesalahan kecil kapada pengunjung *web* Sebagai contoh, apabila pengunjung *website* mengisi *form* pada halaman tersebut, *JavaScript* dapat menyediakan umpan balik secara instan atas *form* yang telah diisi. Jika sebuah *field* yang harus diisi terlewatkan oleh pengguna, halaman *website* tersebut dapat memberitahu pengguna sebelum data-data dikirim ke *server*.
4. Meningkatkan *usability* dengan memungkinkan pengguna untuk mengubah dan berinteraksi dengan antarmuka tanpa harus memuat ulang halaman *web* tersebut. Sebagai contoh, dengan menyembunyikan atau memunculkan suatu bagian dari dokumen *HTML*.
5. Pengembang dapat menciptakan antarmuka yang bereaksi ketika pengguna mengarahkan *pointer* dari *mouse* di atas antarmuka tersebut atau dapat pula diaktifkan dengan *keyboard*. Hal ini memungkinkan dengan adanya bantuan dari *CSS (Cascading Style Sheet) dan HTML*
6. *JavaScript* dapat digunakan untuk memasukkan komponen yang bersifat *drag-and-drop ataupun slider* agar antarmuka menjadi lebih menarik dan interaktif
7. *JavaScript* memiliki ukuran yang kecil dan memiliki tempat penyimpnan sementara *(temporary cache)* begitu selesai dimuat. *JavaScript* juga menggunakan *browser* untuk mengontrol fungsionalitasnya seperti *Flash atau Java Applets*

# 2.17 Router Flux

Navigasi adalah komponen penting dalam sebuah aplikasi, karena sebagian besar aplikasi pasti memiliki lebih dari satu halaman. Untuk berpindah dari satu halaman ke halaman lainnya kita memerlukan komponen navigasi yaitu *react-native-router-flux*. *React-native-router-flux* adalah *API* yang membantu pengguna untuk menentukan semua rute di satu tempat pusat dan menavigasi serta berkomunikasi antara layar yang berbeda dengan cara yang mudah.

## 2.18 Redux

*Redux* adalah penampung yang terprediksi untuk aplikasi *JavaScript*. *State* *management* adalah salah satu aspek tersulit dari pengembangan sebuah *software*. *State* *mismanagement* adalah sumber dari sebagian besar masalah. *Redux* adalah implementasi sederhana dari *Flux* *architecture* *facebook* yang berfungsi sebagai *Model*-*View*-*Controller* *Framework*. *Redux* meminimalisir kompleksitas dengan menggunakan *reducers*. *Redux* *reducers* adalah fungsi tanpa efek samping yang melakukan perhitungan kepada aplikasi *state*. (Thung Kuan, 2016, p. 22).

*Redux* dibuat dengan tiga prinsip :

1. *State* aplikasi tersimpan didalam satu objek. *Redux* menyimpan satu objek *JavaScript* agar proses *map* *out* dan pengiriman data ke semua aplikasi dapat dilakukan dengan mudah.
2. *State* aplikasi bersifat *immune*. Didalam *Redux*, *states* tidak bisa di modifikasi. Cara satu-satunya agar *state* bisa di-modifikasi adalah dengan menyediakan *action*. *Actions* adalah objek *javascript* *immune* yang menjelaskan tentang perubahan *state*. *Actions* di-eksekusi dengan tujuan untuk mencegah kondisi saling mendahului.
3. *Reducers* menentukan bagaimana *action* mengubah sebuah *state*. *Reducers* adalah fungsi *JavaScript* yang membuat sebuah *state* baru dengan menggunakan *state* dan *action* yang sudah ada. Mereka terpusat dalam memutasi data dan dapat melakukannya kepada semua *state*. *Reducers* juga dapat di kombinasi dan digunakan kembali.

Arsitektur ini meningkatkan skalabilitas untuk aplikasi yang kompleks dan besar dengan baik. Arsitektur ini juga memberikan developer fitur yang bagus karena memungkinkan developer untuk melacak semua mutasi kepada action yang menyebabkannya. Dengan *state* dan *action*, aplikasi selanjutnya dapat diprediksi dengan pasti. (Redux)

## 2.19 C#

*C#* Dibuat berdasarkan *C++, C#* adalah bahasa pemrograman berorientasi objek yang dikembangkan terutama oleh Anders Hejlsberg, arsitek kepala di *Microsoft*. *C#* merupakan bahasa pemrograman yang kuat, fleksibel dan berorientasi objek (*OOP*) yang digunakan untuk menciptakan berbagai aplikasi. *C#* telah digunakan untuk berbagai proyek seperti situs *web* dinamis, perangkat pengembangan, dan bahkan *compiler*. *C#* merupakan bahasa yang berasal dari *C dan C++* dengan menambahkan fitur baru agar bahasa lebih mudah digunakan (Cashman, 2012, p. 13)

***2.20 Web Service***

Seringkali perlu merancang sistem terdistribusi, di mana kode untuk menjalankan aplikasi tersebut tersebar di beberapa komputer. Misalnya, untuk membuat sistem pemrosesan transaksi besar, Anda mungkin memiliki *server* terpisah untuk bisnis, *server* basis data, dan sebagainya, yang semuanya perlu berkomunikasi satu sama lain. *Web service* adalah salah satu bentuk sistem perangkat lunak yang didesain untuk mendukung interaksi mesin ke mesin melalui jaringan. *Web services* menyediakan *interoperating* antara aplikasi perangkat lunak yang sedang berjalan pada berbagai *platform* dan *framework* (Fawcett, 2012, p. 544)*.* Sebuah aplikasi bertindak sebagai *service provider* (*server*) dan yang lainnya bertindak sebagai *service consumers* (klien).Untuk model seperti ini untuk bekerja, kode pada satu komputer perlu memanggil kode di komputer lain.

Sebagai contoh, kode di *server web* mungkin memerlukan daftar pesanan untuk ditampilkan pada halaman *web*, dalam hal ini akan memanggil kode pada *server* objek bisnis untuk memberikan daftar pesanan. Kode itu mungkin perlu berkomunikasi dengan *database*. Ketika kode pada satu komputer memanggil kode di komputer lain, ini disebut panggilan prosedur jarak jauh (*RPC*).

Terdapat beberapa keuntungan yang dihasilkan oleh *web service*, antara lain:

1. *Platform*-*independent* : *web service* tersedia pada berbagai *platform hardware (mainframe*, *midrange*, perangkat *personal* dan *mobile*) dan sistem operasi (*UNIX, Windows, Mainframe OS, Android, dan iPhone*).
2. Penggunaan ulang infrastruktur jaringan yang telah ada : *HTTP, SMTP*, dan protokol *JMS*.
3. Mengurangi biaya integrasi dan meningkatkan kecepatan integrasi.

***2.21 WEB API***

*ASP.NET* *Web API* adalah *framework* yang memungkinkan jangkauan luas klien, dari browser hingga perangkat *mobile*, yang menggunakan servis *HTTP*. Di sisi server *Web* *API* membantu dalam pembangunan servis *HTTP* yang mudah digunakan. Dari segi bagaimana itu berbeda dari *MVC*, secara umum, jawabannya dapat diberikan dalam istilah bagaimana memanfaatkan *HTTP*. *MVC* menggunakan notasi berbasis *REST* untuk mengidentifikasi sumber daya sisi server yang diambil. Notasi *REST* menggunakan kata kerja *HTTP* ( *GET*, *PUT*, *DELETE*, dan *POST*) untuk melakukan operasi mereka. *Web* *API* memanfaatkan semua kemampuan *HTTP* (termasuk *headers*, *body*, *full* *URI* *addressing*) untuk menciptakan cara yang kaya dan dapat dioperasikan untuk mengakses sumber daya. (Johnson, 2017,p.568)

## 2.22 REST

*REST (REpresentational State Transfer)* merupakan standard dalam arsitektur *web* yang menggunakan Protokol *HTTP* untuk pertukaran data (Oracle , 2013, p.381). Konsep *REST* pertamakali diperkenalkan oleh Roy Fielding pada tahun 2000. Cara kerjanya, *REST* *server* menyediakan jalur untuk akses *resource* atau data, sedangkan *REST* *client* melakukan akses *resource* dan kemudian menampilkan atau menggunakannya. *Resource* yang dihasilkan sebenarnya berupa teks, namun formatnya bisa bermacam-macam tergantung keinginan *developer*, umumnya adalah *JSON* dan *XML*.

Dalam mengakses sebuah *resource*, *REST* juga menggunakan konsep *URI* dimana ada *method* yang digunakan, secara *default* adalah *GET*. Berikut ini *method-method* yang mendukung *REST*:  
- *GET*, untuk *resource* yang hanya perlu dibaca saja (*read only*)  
- *PUT*, digunakan untuk membuat *resource* baru.  
- *DELETE*, digunakan untuk menghapus suatu *resource*.  
- *POST*, digunakan untuk mengupdate suatu *resource*.  
- *OPTIONS*, digunakan untuk mendapatkan operasi yang didukung pada *resource*.

## 2.23 JavaScript Object Notation ( JSON )

*JSON* adalah format data yang sangat ringan berdasarkan *subset* dari sintaks *JavaScript* (Smith, 2015, p. 37). Karena menggunakan sintaks *JavaScript*, definisi *JSON* dapat dimasukkan dalam *file JavaScript* dan diakses tanpa parsing tambahan untuk mempresentasikan struktur data data sederhana dan menggunakan bahasa pemograman yang fleksibel.

Bentuk *JSON* yang umumnya digunakan adalah objek, dimana terdapat sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan kurung kurawal buka ( **{** ) dan diakhiri dengan kurung kurawal tutup ( **}** ). Setiap nama diikuti dengan titik dua ( **:** ) dan setiap pasangan nama/nilai dipisahkan oleh koma ( **,** ).

Sintaks *JSON* sebenarnya tidak lebih dari campuran objek dan literal *array* untuk menyimpan data. Satu-satunya perbedaan dari contoh di bagian terakhir adalah bahwa *JSON* tidak memiliki variabel.*JSON* hanya mewakili data; tidak memiliki konsep variabel, tugas, atau persamaan. Contoh:

{

  "availableColors": ["merah", "putih", "biru"],

  "availableDoors": [2, 4]

}

***2.24* Sistem Basis Data**

**2.24.1 Pengertian Basis Data**

*Database* adalah kumpulan data yang diatur dan memungkinkan untuk mengakses, pengambilan, dan penggunaan data tersebut. Data adalah kumpulan item yang diproses, dapat mencakup teks, angka, gambar, audio, dan video. Informasi adalah data yang diproses secara bermakna dan berguna. Komputer mengolah data dalam suatu *database* menjadi informasi (Connolly, 2015, p. 63).

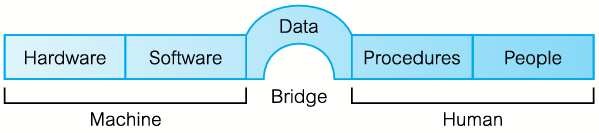
Dengan perangkat lunak *database*, sering disebut sistem manajemen basis data (*DBMS*), pengguna membuat basis data terkomputerisasi seperti menambah, mengubah, dan menghapus data dalam database, menyortir dan mengambil data dari *database*. Kesimpulannya adalah, basis data merupakan kumpulan simbol yang terorganisir dan dapat dibaca oleh mesin, yang harus ditafsirkan sebagai nilai yang benar dari sebuah perusahaan. Basis data juga adalah mesin yang dapat diperbarui, dan dengan begitu juga menjadi koleksi dari variabel.

Dalam sistem basis data juga terdapat beberapa istilah *key* yang dikenal dengan *relational key* :

* + - 1. *Super key* : Sebuah atribut, atau kelompok atribut yang secara unik mengidentifikasi *tuple* dalam relasi.
      2. *Candidate key* : Sebuah *super key* yang unik dan dicalonkan menjadi *primary key*.
      3. *Primary key* : *Candidate key* yang terpilih untuk mengidentifikasi *tuple* secara unik dalam relasi.
      4. *Alternate key* : *Candidate key* yang tidak terpilih untuk menjadi *primary key*.
      5. *Composite key* : Gabungan dua *key* atau lebih yang secara unik dapat mengidentifikasi sebuah *tuple*.
      6. *Foreign key* : Sebuah atribut, atau kumpulan atribut, dalam sebuah relasi yang cocok dengan *candidate key* dari beberapa (mungkin juga sama) relasi.

## 2.24.2 Database Management System (DBMS)

Dengan perangkat lunak *database*, sering disebut sistem manajemen basis data (*DBMS*), pengguna membuat basis data terkomputerisasi seperti menambah, mengubah, dan menghapus data dalam *database* menyortir dan mengambil data dari *database*. Lima komponen utama *DBMS* : perangkat keras (*hardware*), perangkat lunak (*software*), data, prosedur, dan manusia (Connolly, 2015, p. 66)



**Gambar 2.10** Lingkungan *DBMS* (Connolly, 2015:66)

* + - * 1. ***Hardware* (Perangkat Keras)**

*DBMS* membutuhkan perangkat keras untuk bekerja. Perangkat keras dapat beragam seperti komputer personal, *mainframe* dan jaringan beberapa komputer. Perangkat keras tertentu bergantung pada kebutuhan organisasi dan *DBMS* yang digunakan. Beberapa *DBMS* hanya dapat bekerja pada sistem operasi dan perangkat keras tertentu, sementara yang lain dapat bekerja pada berbagai perangkat keras dan sistem operasi.

* + - * 1. ***Software* (Perangkat Lunak)**

Komponen perangkat lunak meliputi perangkat lunak *DBMS* itu sendiri dan program aplikasi bersama dengan sistem operasi seperti jaringan perangkat lunak jika *DBMS* digunakan. Umumnya, program aplikasi dibuat dalam bahasa pemrograman generasi ketiga (*3GL*), seperti *C, C++, Java, Visual Basic, COBOL, Fortran, Ada, dan Pascal,* atau dengan bahasa pemrograman generasi keempat (*4GL*), seperti *SQL*, yang disisipkan dalam bahasa pemrograman generasi ketiga.

# Data

Komponen yang paling penting dari lingkungan *DBMS* adalah data. Data bertindak sebagai jembatan antara sistem dan manusia. *Database* berisi data operasional dan *metadata*. Struktur *database* disebut *schema*.

# Prosedur

Prosedur berisi instruksi dan aturan yang mengatur rancangan dan penggunaan *database*. Contoh dari instruksi-instruksi ini :

Bagaimana masuk ke *DBMS*.

Bagaimana menggunakan fasilitas *DBMS* atau program aplikasi tertentu

Bagaimana memulai dan mengakhiri *DBMS*.

Bagaimana membuat salinan cadangan (*backup*) dari basis data.

Bagaimana menangani kegagalan perangkat keras dan lunak.

Bagaimana mengubah struktur tabel, menyusun ulang basis data di beberapa *disk*, meningkatkan kinerja, atau menyimpan data ke tempat penyimpan cadangan.

# Manusia

Ada lima tipe berbeda dari manusia yang berperan dalam *DBMS* :

## Data Administrator

Bertugas untuk mengelola data dari sumber termasuk perencanaan basis data, pengembangan dan pemeliharaan standar, kebijakan dan prosedur, dan perancangan konseptual atau logikal basis data.

## Database Administrator

Bertugas untuk realisasi fisik dari basis data, termasuk perancangan basis data fisikal dan implementasi, keamanan dan integritas, pemeliharaan sistem operasional, dan memastikan kepuasan kinerja aplikasi bagi user.

## Database Designer

Terdapat dua tipe perancang basis data yaitu perancang basis data logikal dan perancang basis data fisikal.

## Application Developer

Bertugas untuk mengimplementasikan program aplikasi yang menyediakan fungsi-fungsi yang dibutuhkan bagi pengguna akhir. Setiap program mengandung perintah yang meminta DBMS untuk menjalankan suatu operasi pada sistem basis data.

## End-User

*End user* merupakan klien bagi basis data yang telah dirancang dan diimplementasikan, dan dipelihara untuk menyediakan informasi yang dibutuhkan.

**2.25 Data Manipulation**

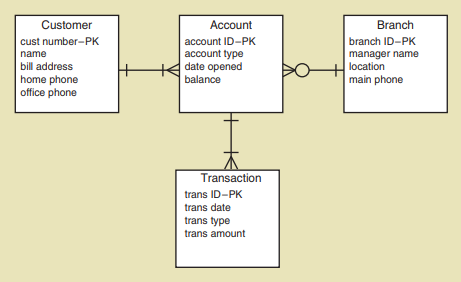
*Data Manipulation Language* (*DML*) adalah sebuah bahasa yang menyediakan kumpulan operasi untuk mendukung operasi manipulasi data dasar pada data di dalam basis data (Connolly, 2015, p. 90).

Beberapa operasi manipulasi data, antara lain :

1. Penambahan data baru ke dalam basis data (*Insert*).
2. Perubahan data yang tersimpan di dalam basis data (*Update*).
3. Pengambilan data yang terdapat di dalam basis data (*Select*).
4. Penghapusan data dari basis data (*Delete*).

***2.26 ERD ( Entity Relationship Diagram )***

*Entity Relationship Diagram* (ERD) digunakan untuk menggambarkan hubungan antara satu entitas dengan entitas yang lain (Connolly, 2015, p. 406). *ERD* awalnya diusulkan oleh Peter Chen untuk desain sistem *database* relasional dan telah diperpanjang oleh yang lain. Satu *set* komponen utama diidentifikasi untuk *ERD*: objek data, atribut, hubungan, dan berbagai jenis indikator. Tujuan utama *ERD* adalah untuk merepresentasikan objek data dan hubungannya. Objek data diwakili oleh persegi panjang berlabel. Hubungan ditunjukkan dengan garis berlabel menghubungkan objek. Dalam beberapa variasi *ERD*, garis penghubung mengandung berlian yang diberi label hubungan. Koneksi antara objek data dan hubungan dibuat menggunakan berbagai simbol khusus yang menunjukkan kardinalitas dan modalitas



**Gambar 2.11** Contoh *ERD* (Pressman, 2010:100)

*ERD* memiliki 4 komponen yaitu :

1. *Entity*

*Entity* adalah benda yang yang memiliki identifikasi yang berbeda. *Entity* dapat digambarkan sebagai persegi yang berisi nama dari *entity* tersebut.

1. *Relationship*

*Relationship* adalah hubungan antara *entity*. *Entity* merupakan pengikut dari *relationship*. *Relationship* dapat digambarkan dalam bentuk belah ketupat yang mana berisi nama dari relasi tersebut. *Relationship* dapat berupa hubungan *one* –*to-one*, *one-to-many* atau *many-to-many.*

1. *Attributes*

*Attributes* merupakan sifat-sifat (property) dari sebuah entitas atau tipe *relationship*. *Attribute Domain* adalah himpunan nilai yang diperbolehkan untuk satu atau lebih atribut.

1. *Structural Contraints*

Tipe utama sebuah *constraint relationship* disebut *multiplicity*. *Multiplicity* adalah jumlah suatu kejadian sebuah tipe entitas yang berhubungan dengan kejadian tunggal dari tipe entitas melalui hubungan tertentu. Pada *multiplicity* terdapat 3 tipe *relationship* yang menggunakan *intergity constraints*, yaitu :

* 1. *One to one Relationships* (1 : 1)
  2. *One to many Relationships* (1 : \*)
  3. *Many to many Relationships* (\* : \*)

***2.27 SQL***

*SQL* (diucapkan sebagai huruf S-Q-L atau sebagai sekuel) adalah singkatan untuk *Structured Query Language. SQL* adalah bahasa yang dirancang khusus untuk berkomunikasi dengan basis data (Connolly, 2015, p. 193). Tidak seperti bahasa lain (bahasa lisan seperti bahasa Inggris, atau bahasa pemrograman seperti *C, Java, atau Python*), *SQL* terdiri dari sangat sedikit kata. *SQL* dirancang untuk melakukan satu hal dan menyediakan developer cara yang mudah dan efisien untuk membaca dan menulis data dari database. Kelebihan dari *SQL*, diantaranya:

1. *SQL* bukan bahasa kepemilikan yang digunakan oleh *vendor database* tertentu. Hampir setiap *DBMS* besar mendukung *SQL*, jadi mempelajari bahasa yang satu ini memungkinkan developer berinteraksi dengan hampir semua *database* yang akan dijalankan.
2. *SQL* mudah dipelajari. Semua pernyataan terdiri dari kata-kata bahasa Inggris deskriptif
3. Meskipun kesederhanaannya jelas, *SQL* adalah bahasa yang sangat kuat, dan dengan menggunakan elemen bahasanya secara cerdik, *developer* dapat melakukan operasi basis data yang kompleks dan canggih.

**2.28 SQL Server**

*SQL Server* dikenal dengan peran sebagai *RDBMS* tradisional, *SQL Server* juga menyediakan kemampuan *reporting*, analisis data yang kuat dan *data mining* (Leiter, 2009, p. 3). *SQL Server* tentunya juga mempunyai fitur-fitur yang mendukung aplikasi data yang berbedai. Semua sistem dan pengguna basis data (termasuk sumber daya basis data) disimpan dalam *file*. Selalu ada minimal dua *file* penyimpanan yaitu satu file data dan satu *file* transaksi. Ekstensi *default* untuk *file data* adalah .*MDF*, dan *default* untuk *file log* transaksi adalah .*LDF*.

**2.29 Black-Box Testing**

Pengujian menggunakan sekumpulan aktifitas validasi, dengan pendekatan *black box testing*. *Black box testing* adalah menguji perangkat lunak dari segi spesifikasi fungsional tanpa menguji desain dan kode program. Pengujian dimaksudkan untuk mengetahui apakah fungsi-fungsi, masukan, dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan. Pengujian kotak hitam dilakukan dengan membuat kasus uji yang bersifat mencoba semua fungsi dengan memakai perangkat lunak apakah sesuai 25 dengan spesifikasi yang dibutuhkan. Kasus uji yang dibuat untuk melakukan pengujian *black box testing* harus dibuat dengan kasus benar dan kasus salah. (Shalahuddin dan Rosa, 2011)

Pengujian *black-box* mengacu pada pengujian yang dilakukan di antarmuka perangkat lunak. Pengujian *black-box* menguji beberapa aspek mendasar dari suatu sistem dengan tidak memperhatikan struktur logis internal perangkat lunak (Pressman, 2010, p. 484)