BAB 2
TINJAUAN PUSTAKA

2.1 Pengertian Fashion

Fashion adalah gaya berbusana. Busana adalah pakaian atau baju atau kostum yang berfungsi sebagai pelindung tubuh yang dapat dilihat dari kutipan "*for as long as humans have lived in organized societies, clothing has represented for more than a simple means of protecting oneself form the cold or covering the body* [4]*."*

Kata *fashion* terkait dengan bahasa latin, *Factio* yang artinya membuat, karena itu arti asli dari *fashion* adalah sesuatu kegiatan yang dilakukan seseorang, tidak seperti dewasa ini yang memaknai *fashion* sebagai sesuatu yang dikenakan seseorang. *Fashion* juga bisa didefinisikan sebagai gaya yang diterima dan digunakan oleh mayoritas anggota sebuah kelompok dalam satu waktu tertentu [5].

Fashion pada umumnya suatu ekspresi atau ungkapan pribadi yang tidak selalu sama untuk setiap orang. Perubahan mode yang menyangkut busana akan terjadi lebih cepat dibandingkan dengan perubahan kebudayaan secara keseluruhan. Dalam dunia fashion terdapat istilah *fashionable* dan *unfashionable* untuk menjelaskan apakah seseorang tersebut mengikuti perkembangan mode terbaru atau tidak [6].

2.1.1 Perkembangan Fashion

Fashion berubah dari waktu ke waktu secara konstan. Proses perubahan tersebut terkadang mengalami istilah *out of fashion* atau ketinggalan jaman, dengan terjadinya perubahan tersebut busana yang *out of fashion* pada satu saat nantinya akan muncul kembali dengan modifikasi. Tren berbusana sebenarnya hanya berputar dengan disertai modifikasi-modifikasi yang baru [7].

1. Awal Abad 20

Perkembangan *fashion* dimulai di Paris dan London. Tahun 1900 akhir abad 19, industri pakaian telah meluas memenuhi kebutuhan dan gaya hidup yang serba praktis. Sepanjang tahun 1910 busana-busana yang lembut dan feminin mulai berkembang pesat. Peragaan busana dipelopori oleh desainer dari Paris, Jeanne Paquin [7].

2. Masa Peperangan

Periode antara masa peperangan sering disebut sebagai masa kejayaan busana-busana Prancis. Gaya bangsawan ala kerajaan sudah ditinggalkan dan digantikan dengan *Haute couture.* Tahun 1920 setelah perang dunia I, *fashion* mengalami perubahan yang radikal. Para wanita lebih beralih pada penampilan yang lebih *casual*. Busana lebih disesuaikan dengan keadaan lingkungan [7].

3. Pertengahan Abad 20

Perang dunia II menciptakan banyak perubahan-perubahan dalam industri mode setelah perang. Reputasi Paris sebagai pusat mode semakin popular seiring dengan ditemukan tekstil sintetis. Gaya busana tahun 1950 lebih progresif dan mengandung semangat gaya masa lampau. Antara 1960-1969 orang-orang muda mulai menentukan industri mode. Mode tidak hanya untuk orang-orang kaya. Gaya dan busana pada waktu itu adalah sederhana, rapi, dan muda. Bahan-bahan sintetis mulai digunakan secara luas pada tahun 1960. Awal 1960 terjadi kerjasama antara para perancang busana dengan kaum selebritis. Sepanjang tahun 1970 banyak perancang mengadakan revolusi dan inovasi terhadap pakaian pria yang semula formal dan kaku menjadi lebih *casual* [7].

4. Akhir Abad 20

Tahun 1980 industri busana mengalami kejayaan. Peragaan busana mulai banyak ditayangkan di televisi. Keanekaragaman busana terjadi sepanjang tahun 1980, namun dibatasi pada awal tahun 1990 karena faktor ekonomi. Tahun 2000,

perkembangan *fashion* terus meningkat. Inspirasi didapatkan pada tahun-tahun sebelumnya. Gaya busana tahun 60, 70, dan 80-an menjadi sangat popular di pertengahan 2007. Awal tahun 2000 masih terlihat gaya busana yang minimalis berkembang menjadi feminin dan dinamis. Pertengahan 2000 gaya busana menonjolkan sisi yang lebih feminin. Para perancang mulai mengadakan percobaan dengan gaya *tunic*, bentuk-bentuk jubah dengan warna terang. Busana pria juga terdapat sedikit sentuhan feminin terutama setelah pertengahan dekade [7].

2.2 Pengertian OOTD

OOTD adalah singkatan dari *Outfit of The Day*, sebagai padanan kata untuk menunjukan apa yang Anda pakai di hari itu. Konsep OOTD sangat sederhana, dan tidak terlalu dibuat-buat. Dalam artian, itu adalah *outfit* pribadi yang benar-benar mereka pakai untuk beraktivitas dalam satu hari. *Hashtag* ini sudah mulai populer sekitar 2 tahun terakhir.

OOTD bisa diartikan sebuah cara memadu padankan pakaian yang nantinya akan diabadikan kedalam sebuah bentuk foto yang kreatif oleh para pecinta *fashion* ataupun pelaku industri *fashion*, dimana *capture* foto #OOTD tersebut akan diabadikan kedalam foto dengan spot – spot yang menarik, keren dan unik, lalu foto #OOTD tersebut yang menampilkan gaya berpakaian dari atas kepala sampai ujung sepatu akan di unggah ke berbagai *social media*, para pelaku dan pecinta *fashion* biasanya juga menambahkan keterangan tentang semua *brand* yang melekat di tubuh mereka. Tujuannya adalah mereka berupaya untuk membagikan informasi tentang apa yang mereka kenakan sehingga para pecinta *fashion* yang lain juga dapat memiliki *fashion items* yang mereka kenakan, disamping itu mereka akan menjadi *trendsetters* di dalam dunia *fashion*. Fenomena #OOTD yang muncul dan menjadi populer di berbagai *social media* salah satunya adalah instagram, dalam perkembangannya saat ini instagram tidak hanya digunakan untuk sekedar berbagi foto lewat *social media* namun nyatanya

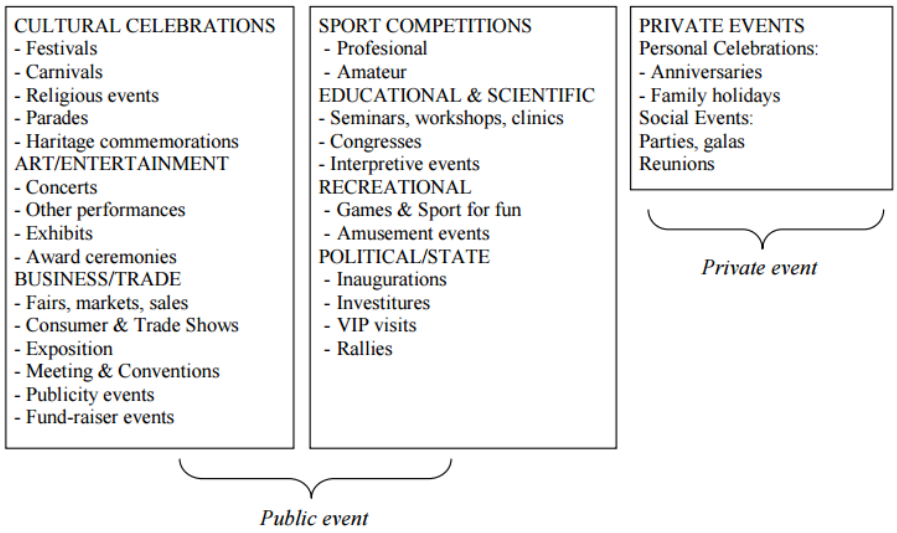
sekarang menjadi sarana promosi, memasarkan dan berkomunikasi yang sangat mumpuni bagi para pelaku industri *fashion* dengan para konsumennya [1].

2.3 Pengertian Event

Definisi *event* adalah sebagai berikut: "*Events a transient and every event is a unique blending of its duration, setting, management, and people." Event* adalah fana, tidak abadi, dan setiap *event* merupakan suatu campuran unik dari durasi, pengaturan,pengurus dan orang-orangnya [8].

Event dibedakan menjadi *public event* dan *private event*, yang termasuk dalam *public event* adalah : perayaan budaya, seni atau hiburan, bisnis atau perdagangan, kompetisi olahraga, pendidikan dan ilmu pengetahuan, rekreasi, serta politik atau kenegaraan. Sedangkan *private event* meliputi perayaan pribadi seperti peringatan hari jadi atau *anniversaries*, liburan keluarga, pesta pernikahan, dan pesta ulang tahun, serta *event-event* sosial seperti pesta-pesta, gala, dan acara reuni [8].

Typology of planned event menurut Getz dapat digambarkan sebagai berikut:



Gambar 2. 1 Typology of Planned Events

2.4 Android

Android adalah sistem operasi untuk *handphone* yang berbasis Linux. Android menyediakan *platform* terbuka bagi para pengembang buat menciptakan aplikasi mereka sendiri untuk digunakan oleh bermacam peranti bergerak. Awalnya, 9 Google Inc. membeli Android Inc. pendatang baru yang membuat peranti lunak untuk *handphone*. Kemudian untuk mengembangkan Android, dibentuklah *Open Handset Alliance*, konsorsium dari 34 perusahaan peranti keras, peranti lunak, dan telekomunikasi [9].

Pada saat perilisan perdana Android, 5 November 2007, Android bersama Open Handset Alliance menyatakan mendukung pengembangan standar terbuka pada perangkat seluler. Di lain pihak, Google merilis kode–kode Android di bawah lisensi Apache, sebuah lisensi perangkat *software* dan standar terbuka perangkat seluler. Di dunia ini terdapat dua jenis distributor sistem operasi Android. Pertama yang mendapat dukungan penuh dari Google atau *Google Mail Services* (GMS) dan kedua adalah yang benar–benar bebas distribusinya tanpa dukungan langsung Google atau dikenal sebagai *Open Handset Distribution* (OHD).

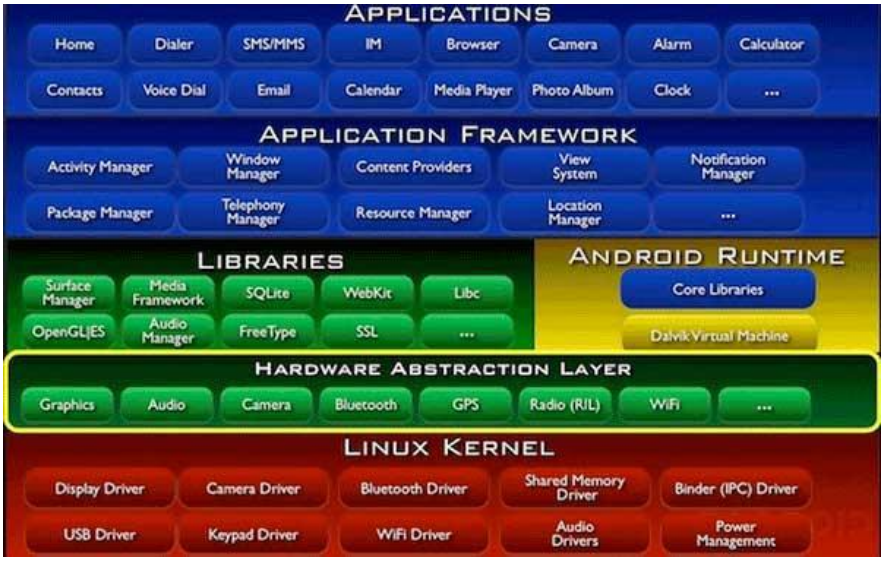
Android menawarkan sebuah lingkungan yang berbeda untuk pengembang. Setiap aplikasi memiliki tingkatan yang sama. Android tidak membedakan antara aplikasi inti dengan aplikasi pihak ketiga. *Application Programming Interface* (API) yang disediakan menawarkan akses ke hardware, maupun data-data ponsel sekalipun, atau data sistem sendiri. Bahkan pengguna dapat menghapus aplikasi inti dan menggantikannya dengan aplikasi pihak ketiga.

Android merupakan sistem operasi yang berkembang dengan pesat, namun tidak menjadikannya sistem operasi yang sempurna ada beberapa kekurangan dari sistem operasi Android diantaranya Android terkesan rumit, karena mempunyai banyak sekali *widget* maupun aplikasi dengan banyak pengaturan sehingga pengguna harus banyak belajar mengenai Android, selain itu Android yang merupakan sistem operasi terbuka sehingga pengguna dapat memasang aplikasi di luar toko aplikasi yang ditawarkan oleh perangkat Android tersebut sehingga

sangat rentan terkena ancaman *malware* atau virus. Tidak semua perangkat Android dapat langsung memperbarui sistem operasi terbaru, karena produsen *smartphone* lebih mementingkan produk baru untuk diberi sistem operasi yang terbaru, dibanding dengan memberi pemberitahuan tentang *update* sistem operasi terbaru sehingga membutuhkan waktu lama untuk memperbarui sistem operasi bagi beberapa perangkat.

2.4.1 Arsitektur Android

Secara garis besar arsitektur android dapat dijelaskan pada gambar sebagai berikut [9]:



Gambar 2. 2 Arsitektur Android

1. Application dan Widget

Application dan *Widgets* ini adalah layer dimana kita berhubungan dengan aplikasi saja, dimana biasanya kita *download* aplikasi kemudian kita lakukan instalasi dan jalankan aplikasi tersebut. Pada layer terdapat aplikasi inti termasuk klien *email*, program SMS, kalender, peta, *browser*, kontak, dan lain-lain. Hampir semua aplikasi ditulis menggunakan bahasa pemrograman Java.

2. Application Frameworks

Applications frameworks Android adalah "*Open Development Platform*" yaitu Android menawarkan kepada pengembang atau memberi kemampuan kepada pengembang untuk membangun aplikasi yang bagus dan inovatif. Pengembang bebas untuk mengakses perangkat keras, akses informasi *resource*, menjalankan *service background*, mengatur alarm, dan menambah status notifikasi, dan sebagainya. Pengembang memiliki akses penuh menuju API *framework* seperti yang dilakukan oleh aplikasi kategori inti. Arsitektur aplikasi dirancang supaya kita dengan mudah dapat menggunakan kembali komponen yang sudah digunakan (*reuse*). Sehingga bisa kita simpulkan *application frameworks* ini adalah *layer* dimana para pembuat aplikasi melakukan pengembangan atau pembuatan aplikasi yang akan dijalankan di sistem operasi Android, karena pada *layer* inilah aplikasi dapat dirancang dan dibuat, seperti *content providers* yang berupa sms dan panggilan telepon.

Komponen-komponen yang termasuk di dalam *application frameworks* adalah sebagai berikut:

a. Views
b. Content Provider
c. Resource Manager
d. Notification Manager
e. Activity Manager

3. Libraries

Libraries ini adalah *layer* dimana fitur-fitur Android berada, biasanya para pembuat aplikasi mengakses *libraries* untuk menjalankan aplikasinya. Berjalan di atas Kernel, layer ini meliputi berbagai library C/C++ inti seperti Libc SSL, serta:

a. *Libraries* media untuk pemutaran media audio dan video..
b. *Libraries* untuk manajemen tampilan.
c. *Libraries Graphics* mencakup SGL dan OpenGL untuk grafis 2D dan 3D.
d. *Libraries* SQLite untuk dukungan *database*.

e. *Libraries* SSL dan WebKit terintegrasi dengan *web browser* dan *security*.

f. *Libraries LiveWebcore* mencakup modern web *browser* dengan *engine embedded web view*.

g. *Libraries* 3D yang mencakup implementasi OpenGL ES1.0 API's.

4. Android Run Time

Layer yang membuat aplikasi Android dapat dijalankan dimana dalam prosesnya menggunakan Implementasi Linux. *Dalvik Virtual Machine* (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android. Di dalam Android *Run Time* dibagi menjadi dua bagian yaitu:

a. *Core Libraries*

Aplikasi Android dibangun dalam bahasa Java, sementara Dalvik sebagai virtual mesinnya bukan Virtual Machine Java,sehingga diperlukan sebuah *libraries* yang berfungsi untuk menterjemahkan bahasa Java/C yang ditangani oleh Core Libraries.

b. *Dalvik Virtual Machine*

Virtual mesin berbasis *register* yang dioptimalkan untuk menjalankan fungsi-fungsi secara efisien, dimana merupakan pengembangan yang mampu membuat Linux Kernel untuk melakukan *threading* dan manajemen tingkat rendah.

5. Linux Kernel

Linux Kernel adalah *layer* dimana inti dari sistem operasi Android itu berada. Berisi *file-file* sistem yang mengatur sistem *processing*, *memory*, *resource*, *drivers*, dan sistem-sistem operasi Android lainnya. Linux Kernel yang digunakan Android adalah Linux Kernel *release* 2.6. Keunikan dari nama sistem operasi (OS) Android adalah dengan menggunakan nama makanan hidangan penutup (*Dessert*). Selain itu juga nama-nama OS Android memiliki huruf awal berurutan sesuai abjad. Adapun beberapa nama dan versi Android yang sudah diluncurkan adalah sebagai berikut :

a. Android v.10 Astro (Alpha)

Sebenarnya sebelum mereka memberikan nama-nama *dessert* sebagai nama untuk versi OS nya, Android sempat memiliki 2 versi awal dengan nama Android Alpha dan Beta. Nama untuk versi pertama ini sendiri sebenarnya adalah Android Astro, namun karena alasan hak cipta (*trademark*), nama ini tidak jadi digunakan. Pada versi awal ini belum ada perangkat dengan sistem operasi Android yang dijual secara komersil.

b. Android v1.1 Bender (beta)

Versi ini dirilis pada tanggal 5 November 2007 yang merupakan versi lanjutan dari Android Astro (Alpha). Sama seperti versi awalnya, nama Bender juga juga tak jadi digunakan karena alasan hak cipta (*trademark*). Kemudian lahirlah 13 telepon seluler pertama dengan sistem operasi Android yang dijual secara komersil yakni HTC Dream.

c. Android v 1.5 Cupcake

Ini merupakan versi pertama yang menggunakan nama *dessert* sebagai kode nama untuk tiap versi Android yang kemudian tradisi untuk menamai versi Android dengan nama *dessert* masih diteruskan hingga saat ini. Android Cupcake dirilis pada tanggal 30 April 2009.

d. Android v1.6 Donut

Dirilis tidak sampai setahun setelah perilisan Android Cupcake, yakni pada tanggal 15 September 2009. Versi ini dihadirkan untuk menutupi *bug* pada versi sebelumnya, sekaligus untuk penambahan beberapa fitur seperti misalnya dukungan untuk perangkat dengan ukuran layar yang lebih besar.

e. Android v2.0 – 2.1 Éclair

Sistem operasi ini juga dirilis tidak sampai setahun setelah perilisan dua versi sebelumnya yakni pada tanggal 26 Oktober 2009. Mereka masih berfokus

untuk menutupi *bug* yang ada dan juga menambahkan beberapa fitur seperti *Bluetooth*, *flash* pada kamera, fitur digital *zoom* pada kamera, *multi-touch*, *live wallpaper*, dan lainnya. Hadirnya perangkat seri Nexus dari Google yang pertama kali muncul yakni HTC Nexus One juga menggunakan versi OS Android Eclair.

f. Android v2.2 Frozen Yoghurt (Froyo)

Dirilis pada tanggal 20 Mei 2010. Perangkat dengan OS Android semakin banyak dan kehadirannya mulai dilirik oleh pasar meski masih jauh dibawah kepopuleran OS lain seperti Symbian dan Windows Mobile.

g. Android v2.3 Gingerbread

Dirilis pada tanggal 6 Desember 2010 bersamaan dengan dihadirkannya Nexus S yang merupakan perangkat *smartphone* seri Nexus yang diproduksi oleh Samsung. Versi OS ini juga mengawali kesuksesan Android di jagad *smartphone* meski masih kalah populer dengan BlackBerry OS. Beberapa vendor mulai serius untuk menggarap perangkat dengan OS Android.Pada saat itu, Samsung dengan Galaxy *series* nya berperan besar dalam kesuksesan Android. Promosi yang luar biasa gencarnya membuat orang awam mulai mengenal sistem operasi Android. Bahkan saat itu sebagian besar orang beranggapan bahwa OS Android adalah milik Samsung karena kuatnya *branding* yang dilakukan oleh Samsung. Ini juga menjadi awal mula kejayaan Samsung di jagad *smartphone*.

h. Android v3.0 – 3.2 Honeycomb

Versi ini dirilis pada tanggal 10 Mei 2011 dan dirancang khusus untuk perangkat tablet, yang kala itu mulai populer di pasaran salah satunya berkat promosi Samsung dan juga kepopuleran Apple iPad.

i. Android v4.0 Ice Cream Sandwich

Dirilis pada 16 Desember 2011. Bisa dibilang merupakan Android Honeycomb yang disempurnakan, dan dioptimalkan untuk penggunaan baik *smartphone* maupun tablet. Perubahan yang paling terlihat dari versi ini dibanding

dengan versi sebelumnya adalah dari segi *User interface* yang nampak lebih bersih dan elegan. Versi ini juga lebih dioptimalkan untuk urusan *multitasking*. Bersamaan dengan diperkenalkannya Android ICS, Google juga memperkenalkan perangkat Galaxy Nexus yang merupakan seri *smartphone* Nexus yang diproduksi oleh Samsung. Setelah versi ini, Google kemudian secara rutin memperkenalkan perangkat seri Nexus pada tiap kali mereka memperkenalkan versi Android terbaru.

j. Android v4.1 – 4.3 Jelly Bean

Dirilis pada 9 Juli 2012. Bersamaan dengan diperkenalkannya versi OS 4.1 pada 27 Juni 2012, Google juga memperkenalkan Nexus 7 yang diproduksi oleh ASUS. Nexus 7 (generasi 1) merupakan seri Nexus pertama yang merupakan perangkat tablet. Jelly Bean mengalami 3x *update* versi yakni 4.1, 4.2 hingga 4.3. Selanjutnya mereka memperkenalkan Android v4.2 bersamaan dengan dihadirkannya Nexus 4, *smartphone* yang diproduksi oleh LG plus Nexus 10, perangkat tablet yang diproduksi oleh Samsung.Pada saat versi 4.3 dirilis, Google juga merilis Nexus 7 generasi 2 yang masih diproduksi oleh ASUS yang mana ia memiliki beberapa peningkatan seperti misalnya penambahan kamera belakang serta dukungan untuk konektivitas internet.

k. Android v4.4 Kitkat

Nama Kitkat diambil dari sebuah produk cemilan wafer berlapis coklat yang dimiliki oleh Nestle. Sebelumnya Android versi “K” ini disebut-sebut sebagai Key Lime Pie, namun atas beberapa pertimbangan akhirnya Google lebih memilih untuk memberi nama Kitkat. Ceritanya, Kitkat adalah salah satu cemilan yang tersedia di dapur kantor yang biasanya juga menemani para *programer* Google. Versi ini diklaim lebih ramah terhadap perangkat dengan spesifikasi seadanya. Bahkan perangkat dengan RAM 512 MB masih bisa menjalankan OS versi ini dengan mulus. Berbeda dengan Jelly Bean yang minimal harus memiliki RAM diatas 756 MB agar dapat berjalan dengan mulus. Bersamaan dengan

dirilisnya Android Kitkat pada tanggal 31 Oktober 2013, Google juga merilis *smartphone* Nexus 5 yang diproduksi oleh LG.

l. Android v5.0 – lollipop

Dirilis pada tanggal 15 Oktober 2014, versi OS ini mengusung perubahan besar dari segi UI yang nampak lebih *flat* dengan konsep material *design*. Versi Android ini sudah mendukung arsitektur 64-bit sehingga sudah memungkinkan untuk penggunaan RAM diatas 3 GB pada *hardware* perangkat. Penggunaan prosesor 64-bit pun makin banyak diadopsi oleh para vendor, mulai dari penerapan pada perangkat *flagship* hingga perangkat kelas menengah kebawah.

m. Android v6.0 Marshmallow

Versi Android ini resmi dirilis pada bulan September tahun 2015. Bersamaan dengan dirilisnya versi ini, untuk pertama kalinya Google juga memperkenalkan 2 perangkat *smartphone* Nexus sekaligus yang diproduksi oleh 2 vendor yang berbeda. Nexus 5X adalah versi *smartphone* Nexus kelas menengah dengan ukuran layar 5.2 inch yang diproduksi oleh LG. Sedangkan yang satunya lagi memiliki bentang layar yang lebih lebar yakni 5.7 inch yang diberi nama Nexus 6P yang merupakan *smartphone flagship* hasil kerjasama Google dengan Huawei.

n. Android v7.0 Nougat

Resmi diperkenalkan pada akhir Juni 2016. Banyak netizen yang berspekulasi bahwa kemungkinan besar, pemberian nama untuk Android versi "N" ini adalah Nutella. Namun Google menepis kabar tersebut setelah resmi memperkenalkannya bersamaan dengan dipamerkannya patung *icon* Android yang berdiri diatas potongan *nougat.* Sebelumnya, Google telah mengundang para penggunanya untuk memberikan ide penamaan pada versi ini. Beberapa nama termasuk Nutella dan Nastar pun muncul, hingga akhirnya Google lebih memilih nama Nutella.

2.4.2 Android Life Cycle

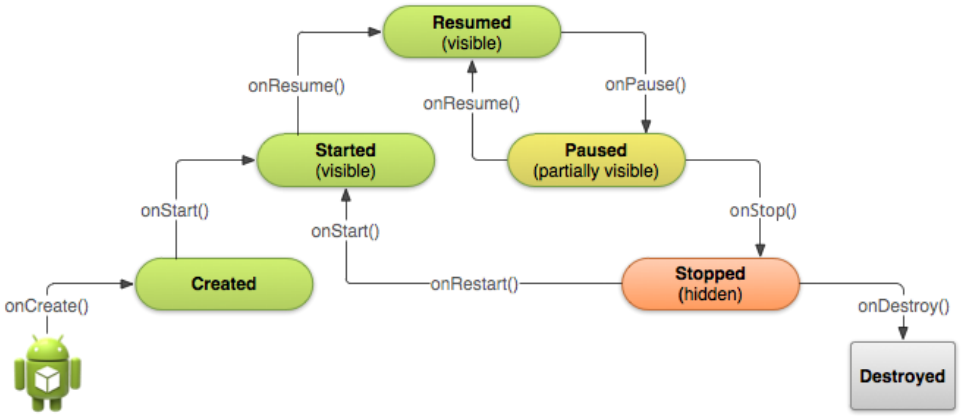
Aplikasi android terdiri dari beberapa fungsi dasar seperti mengubah catatan, memutar *file* musik, membunyikan *alarm*, atau membuka kontak telepon. Fungsi-fungsi tersebut dapat diklasifikasikan ke dalam empat komponen android yang berbeda seperti pada tabel berikut [10].

Tabel 2. 1 Komponen Aplikasi Android

Functionality	Java Base Class	Examples
Focused thing a user can do	*Activity*	*Edit a note, play a game*
Background process	*Service*	*Play music, update weather icon*
Receive messages	*BroadcastReceiver*	*Trigger alarm upon event*
Store and retrieve data	*ContentProvider*	*Open a phone contact*

Setiap aplikasi pasti menggunakan minimal satu dari komponen tersebut, akan tetapi terdapat beberapa komponen yang mengharuskan mencantumkan *specified permission* sebelum digunakan seperti komponen *service, broadcast receiver* dan *content provider* [10].

Android memiliki paradigma pemrograman lain tidak seperti paradigma pemrograman biasa di mana aplikasi yang dijalankan pada fungsi *main*(), sistem android menjalankan kode dalam *method Activity* dengan menerapkan metode *callback* tertentu yang sesuai dengan tahap tertentu dari siklus hidup. Setiap aplikasi yang berjalan dalam sistem operasi android memiliki siklus hidup yang berbeda dengan aplikasi *desktop* atau *web*, Hal ini dikarenakan aplikasi *mobile* memiliki tingkat interupsi proses yang lumayan tinggi seperti ketika *handling* panggilan masuk aplikasi diharuskan menghentikan proses sementara, Penerapan siklus hidup juga berguna untuk memastikan aplikasi tidak menghabiskan sumber daya baterai pengguna [10].



Gambar 2. 3 Siklus Hidup Android

Terdapat beberapa *state* dalam siklus hidup android yang terjadi seperti diilustrasikan pada **Gambar 2.3 Siklus Hidup Android**, akan tetapi hanya beberapa dari *state* tersebut yang menjadi statis diantaranya:

1. Resumed

Resumed terjadi ketika aplikasi berjalan setelah *state paused* . *State* ini akan menjalankan perintah program yang ditulis pada *method* onResume().

2. Paused

Pada keadaan ini aktivitas yang terjadi dihentikan secara sementara tetapi masih terlihat oleh pengguna karena terdapat proses yang memiliki prioritas lebih tinggi seperti panggilan telepon. Aplikasi tidak dapat menjalankan perintah apapun ataupun menampilkan apapun dalam *state* ini.

3. Stopped

Pada keadaan ini, aplikasi benar-benar tidak ditampilkan dan tidak terlihat oleh pengguna tetapi masih meninggalkan *service* di *background*.

State lain seperti *Created* dan *Started* bersifat sementara dan sistem dengan cepat menjalankan *state* berikutnya dengan memanggil metode *life cycle*

callback berikutnya. Artinya, setelah sistem *OnCreate*() dipanggil dengan cepat sistem akan memanggil *method OnStart*(), kemudian diikuti oleh *onResume*().

2.4.3 Fitur Android

Android memiliki beberapa fitur utama yang sering digunakan dalam proses pembangunan aplikasi diantaranya adalah [10]:

1. Multi-proses dan App Widgets

Sistem operasi android tidak melarang prosesor menjalankan lebih dari satu aplikasi dalam satu waktu. Sistem operasi android dapat mengatur aplikasi dan *thread* yang berjalan secara *multitasking.* Keuntungan yang didapat adalah ketika aplikasi berjalan dan berinteraksi dengan pengguna di *layer* depan sistem operasi, proses dari aplikasi lain dapat berjalan untuk melakukan pembaruan informasi. Sebagai contoh misalnya ketika pengguna memainkan *game*, proses lain dapat berjalan di belakang aplikasi seperti memeriksa harga saham dan memunculkan peringatan.

App Widgets adalah mini aplikasi yang dapat *embedded* dalam aplikasi seperti *home screen*. *App widgets* dapat menjalankan proses *request* seperti musik *streaming* atau mendeteksi suhu ruangan secara *background*.

Multi-proses dapat memberikan manfaat berupa *user experience* yang lebih banyak, namun penggunaan fitur tersebut dapat menghabiskan banyak energi baterai jika penggunaan tidak benar.

2. Touch Gestures dan Multi-touch

Touchscreen adalah *user interface* intuitif yang digunakan banyak *smartphone* di dunia. Dengan fitur ini interaksi dapat dibuat lebih mudah karena cukup dengan menggunakan jari tangan. *Multi-touch* adalah kemampuan yang dapat melakukan *tracking* lebih dari satu tangan dalam satu waktu, fitur ini sering digunakan untuk interaksi memperbesar atau memutar objek. Selain itu pengembang dapat membuat interaksi baru dengan memanfaatkan fitur tersebut.

3. Hard dan Soft Keyboard

Salah satu fitur pada perangkat *smartphone* adalah tombol fisik dan non fisik, tombol fisik digunakan untuk navigasi pendukung dalam pengoperasian android.. Pengembang aplikasi tidak perlu secara manual untuk mengintegrasikan tombol tersebut dalam aplikasi. Tombol non fisik adalah tombol yang dibuat oleh sistem operasi seperti *keyboard* virtual, dan tombol navigasi aplikasi.

2.4.4 Prinsip Desain

Android memiliki beberapa prinsip desain yang dapat menjadi acuan dalam membuat desain aplikasi android diantaranya adalah :

1. Multiple Assets

Android mendukung jutaan *smartphone*, tablet dan perangkat lain dalam berbagai ukuran layar dan ukuran, untuk itu *Multiple Assets* sangat disarankan digunakan untuk mengatasi fragmentasi pada android. Seperti ilustrasi pada **Gambar 2.4 Klasifikasi Ukuran Icon**, android menciptakan beberapa klasifikasi ukuran *icon* yaitu MDPI, HDPI, XHDPI, XXHDPI dan XXXHDPI. MDPI dan HDPI dikhususkan untuk *icon* yang akan digunakan pada *device* berukuran *smartphone* sedangkan untuk XHDPI, XXHDPI dan XXXHDPI digunakan pada *device* berukuran tablet.



Gambar 2. 4 Klasifikasi Ukuran Icon

2. Touch Feedback

Touch Feedback dalam android digunakan sebagai respon setiap objek yang ditekan pengguna. Hal ini bertujuan untuk memberi tahu pengguna objek mana yang berinteraksi dengan pengguna.



Gambar 2. 5 Touch Feedback

2.4.5 Android SDK (Software Development Kit)

Android SDK adalah tools API *(Application Programming Interface)* yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* android menggunakan bahasa pemrograman Java. Android merupakan subset perangkat lunak untuk ponsel yang meliputi sistem operasi, *middleware* dan aplikasi kunci yang di *release* oleh Google. Saat ini disediakan Android SDK (*Software Development Kit*) sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java. Sebagai *platform* aplikasi-netral, android *member* anda kesempatan untuk membuat aplikasi yang kita butuhkan yang bukan merupakan aplikasi bawaan *Handphone* atau *Smartphone* [10].

Beberapa fitur-fitur android yang paling penting adalah [10]:

a. *Framework* : Aplikasi yang mendukung pengganti komponen dan *reusable*.
b. *Dalvik Virtual Machine* dioptimalkan untuk perangkat *mobile*.
c. *Integrated Browser* berdasarkan *engine open source* WebKit.
d. Grafis yang dioptimalkan dan didukung oleh *libraries* grafis 2D, grafis 3D berdasarkan spesifikasi opengl ES 1,0 (Opsional Ekselerasi *hardware*)

e. SQLite untuk penyimpanan data.
f. Media *Support* yang mendukung audio, video, dan gambar (MPEG4, H.264, MP3, AAC, AMR, JPG, PING, GIF), GSM *Telephony* (tergantung *hardware*)
g. *Bluetooth*, EDGE, 3G, dan WiFi (tergantung *hardware*)
h. Kamera, GPS, Kompas, dan *Accelerometer* (tergantung *hardware*)
i. Lingkungan *Development* yang lengkap dan termasuk perangkat *emulator*, *tools* untuk *debugging*, profil dan kinerja memori, dan *plugin*.

2.5 Java

Inovasi bahasa komputer dimotivasi oleh dua factor yaitu, perbaikan dalam seni pemrograman dan perubahan dalam lingkungan komputasi, tidak terkecuali Java. Java dibangun di atas warisan yang kaya dari C dan C++, Java menambahkan perbaikan dan fitur yang mencerminkan keadaan seni dalam pemrograman saat ini. Menanggapi munculnya lingkungan *online*, Java menawarkan fitur yang merampingkan pemrograman untuk arsitektur yang sangat terdistribusi.

Java adalah bahasa pemrograman yang dapat dijalankan di berbagai komputer termasuk telepon genggam. Bahasa ini awalnya dibuat oleh James Gosling saat masih bergabung di Sun Microsystems saat ini merupakan bagian dari Oracle dan dirilis tahun 1995. Bahasa ini banyak mengadopsi sintaksis yang terdapat pada C dan C++ namun dengan sintaksis model objek yang lebih sederhana serta dukungan rutin-rutin arus bawah yang minimal. Aplikasi-aplikasi berbasis java umumnya dikompilasi ke dalam p-code (*bytecode*) dan dapat dijalankan pada berbagai *Mesin Virtual Java* (JVM).

Versi awal Java ditahun 1996 sudah merupakan versi rilis sehingga dinamakan Java Versi 1.0. Java versi ini menyertakan banyak paket standar awal yang terus dikembangkan pada versi selanjutnya:

a. java.lang: Peruntukan kelas elemen-elemen dasar
b. java.io: Peruntukan kelas *input* dan *output*, termasuk penggunaan berkas.

c. java.util: Peruntukan kelas pelengkap seperti kelas struktur data dan kelas kelas penanggalan.

d. java.net: Peruntukan kelas TCP/IP, yang memungkinkan berkomunikasi dengan komputer lain menggunakan jaringan TCP/IP.

e. java.awt: Kelas dasar untuk aplikasi antarmuka dengan pengguna (GUI)

f. java.applet: Kelas dasar aplikasi antar muka untuk diterapkan pada penjelajah *web*.

Seperti telah dibahas sebelumnya, banyak jenis komputer dan sistem operasi yang terhubung ke internet. Program-program untuk secara dinamis di *download* ke semua berbagai jenis *platform*, beberapa sarana untuk menghasilkan kode dieksekusi portabel diperlukan. Mekanisme yang sama yang membantu menjamin keamanan juga membantu menciptakan portabilitas karena fungsionalitasnya yang memungkinkan aplikasi java mampu berjalan di beberapa *platform* sistem operasi yang berbeda. Saat ini java merupakan bahasa pemrograman yang populer digunakan dan secara luas dimanfaatkan dalam pengembangan berbagai jenis perangkat lunak aplikasi ataupun aplikasi berbasis *web*.

Java memliki beberapa kelebihan dibandingkan dengan bahasa pemograman lain,diantaranya:

a. Multiplatform

b. OOP (*Object Oriented Programming*)

c. *Library Class* yang lengkap.

d. Mewarisi Kekayaan C/C++

e. Pengumpulan Sampah Otomatis

2.6 Web Service

W3C mendefinisikan *web service* sebagai sebuah *software* aplikasi yang dapat teridentifikasi oleh URL dan memiliki *interface* yang didefinisikan, dideskripsikan, dan dimengerti oleh XML atau JSON dan juga mendukung

interaksi langsung dengan *software* aplikasi yang lain dengan menggunakan *message* berbasis XML atau JSON melalui protokol internet.

Web service adalah sebuah *software* aplikasi yang tidak terpengaruh oleh *platform*, menyediakan *method-method* yang dapat diakses oleh *network*. *Web Service* juga akan menggunakan XML untuk pertukaran data, khususnya pada dua entitas bisnis yang berbeda. Beberapa karakteristik dari web service adalah:

a. Message-based
b. Standards-based
c. Programming language independent
d. Platform-neutral

Beberapa *key* standar di dalam *web service* adalah: JSON, XML, SOAP, WSDL and UDDI.

2.7 JSON

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman JavaScript, Standar ECMA-262 Edisi ke 3 – Desember 1999 [11].

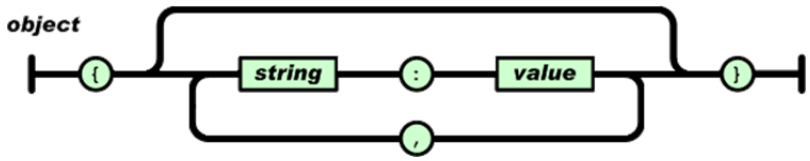
JSON merupakan format teks yang tidak bergantung pada bahasa pemrograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh programmer keluarga C termasuk C, C++, C#, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran data. JSON terbuat dari dua struktur, yaitu [11]:

1. Kumpulan pasangan nama atau nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), daftar berkunci (*keyed list*), atau *associative array.*
2. Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*) .

Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman modern mendukung sruktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemrograman yang juga berdasarkan pada struktur data ini. JSON menggunakan bentuk sebagai berikut:

1. Objek

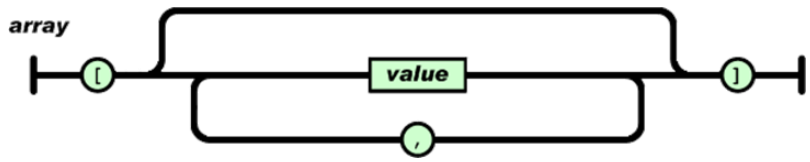
Objek adalah sepasang nama / nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma). Objek biasanya digunakan untuk menyimpan data tunggal dalam bentuk JSON.



Gambar 2. 6 Objek JSON

2. Larik

Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma). Larik dalam JSON dapat digunakan sebagai value dari JSON object hal ini dapat berguna jika JSON menyimpan data bertingkat.

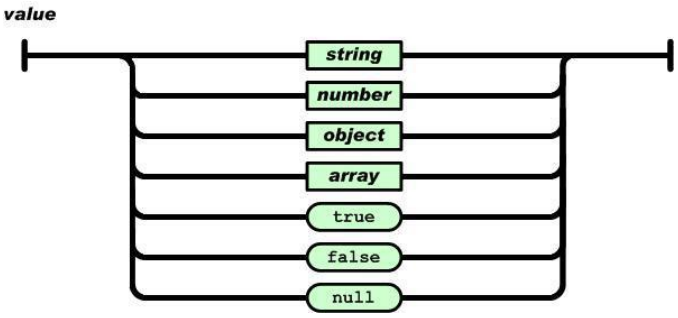


Gambar 2. 7 Array JSON

Bentuk data JSON objek dan larik dapat saling dikombinasikan untuk mendukung struktur data yang lebih kompleks. JSON mendukung beberapa tipe data untuk menjadi *value* seperti Angka, *String*, *Boolean* dan nilai NULL.

3. Nilai

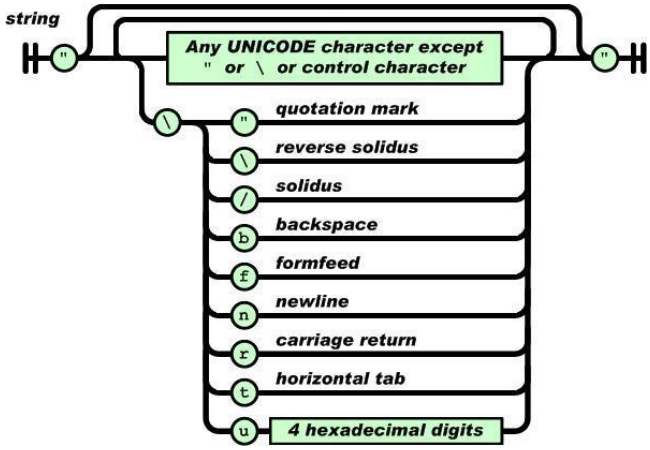
Nilai (*value*) dapat berupa sebuah *string* dalam tanda kutip ganda, atau angka, atau *true* atau *false* atau *null* atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.



Gambar 2. 8 Value JSON

4. String

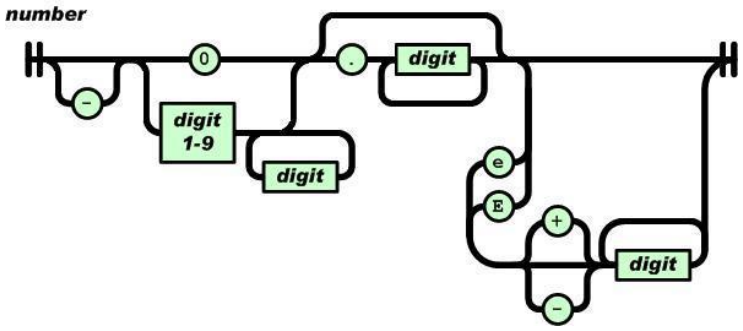
String adalah kumpulan dari nol atau lebih karakter *Unicode*, yang dibungkus dengan tanda kutip ganda. Pada *string* dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada *string*. *String* sangat mirip dengan *string* C atau Java .



Gambar 2. 9 String JSON

5. Angka

Angka, format oktal dan heksadesimla tidak digunakan.



Gambar 2. 10 Number JSON

2.8 API (Application Programming Interface)

API adalah sekumpulan perintah, fungsi, dan protokol yang dapat digunakan saat membangun perangkat lunak untuk sistem operasi tertentu. API memungkinkan *programmer* untuk menggunakan fungsi standar untuk berinteraksi dengan system operasi. API atau *Application Programming Interface* juga merupakan suatu dokumentasi yang terdiri dari antar muka, fungsi, kelas, struktur untuk membangun sebuah perangkat lunak.

Dengan adanya API, maka memudahkan seorang *programmer* untuk membongkar suatu *software* untuk kemudian dapat dikembangkan atau diintegrasikan dengan perangkat lunak yang lain. API dapat dikatakan sebagai penghubung suatu aplikasi dengan aplikasi lainnya. Suatu rutin standar yang memungkinkan *developer* menggunakan *system function*. Proses ini dikelola melalui *operating system*. Keunggulan dari API ini adalah memungkinkan suatu aplikasi dengan aplikasi lainnya untuk saling berinteraksi.

API menggunakan bahasa yang lebih terstruktur dan mudah dimengerti daripada bahasa *system call*. Hal ini sangat penting dalam hal *editing* dan pengembangan. *System call interface* ini berfungsi sebagai penghubung antara API dan *system call* yang dimengerti oleh sistem operasi. *System call interface* ini akan menerjemahkan perintah dalam API dan kemudian akan memanggil *system*

calls yang diperlukan. Untuk membuka suatu *file* tersebut *user* menggunakan program yang telah dibuat dengan menggunakan bantuan API, maka perintah dari user tersebut diterjemahkan dulu oleh program menjadi perintah *open*().

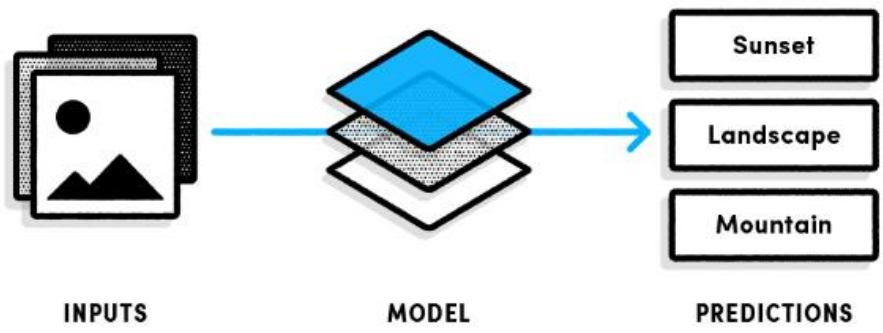
Perintah *open*() ini merupakan perintah dari API dan bukan perintah yang langsung dimengerti oleh kernel sistem operasi. Oleh karena itu, agar keinginan pengguna dapat dimengerti oleh sistem operasi, maka perintah *open*() tadi diterjemahkan ke dalam bentuk *system call* oleh *system call interface*. Implementasi perintah *open*() tadi bisa bermacam-macam tergantung dari sistem operasi yang digunakan. Cara menggunakan API :

1. Dilakukan dengan mengimpor *package*/kelas
2. Ada beberapa kelas bernama sama dipackage yang berbeda, yaitu :
   - *Import* salah satu dan gunakan nama lengkap untuk yang lain
   - Gunakan nama lengkap semua kelas.

Kebanyakan sistem operasi seperti Windows, menyediakan fasilitas API sehingga *programmer* dapat melakukan aktivitas *programming* dengan lebih konsisten. Meskipun API didesain untuk *programmer*, namun API juga baik untuk *user* karena setidaknya dapat menjamin bahwa program tersebut memiliki *interface* yang sama, sehingga lebih mudah untuk dipelajari.

2.9 Clarifai

`

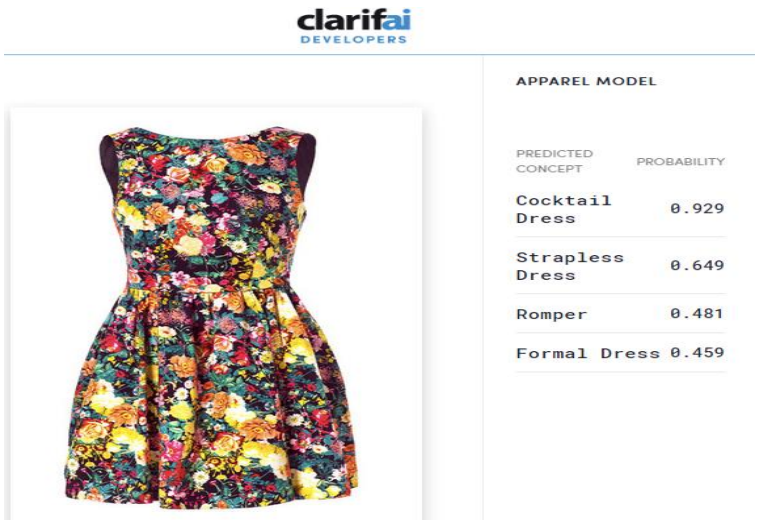


Gambar 2. 11 Proses Pengenalan Clarifai

Model-model yang terdapat pada Clarifai adalah sebagai berikut :

1. Apparel Model

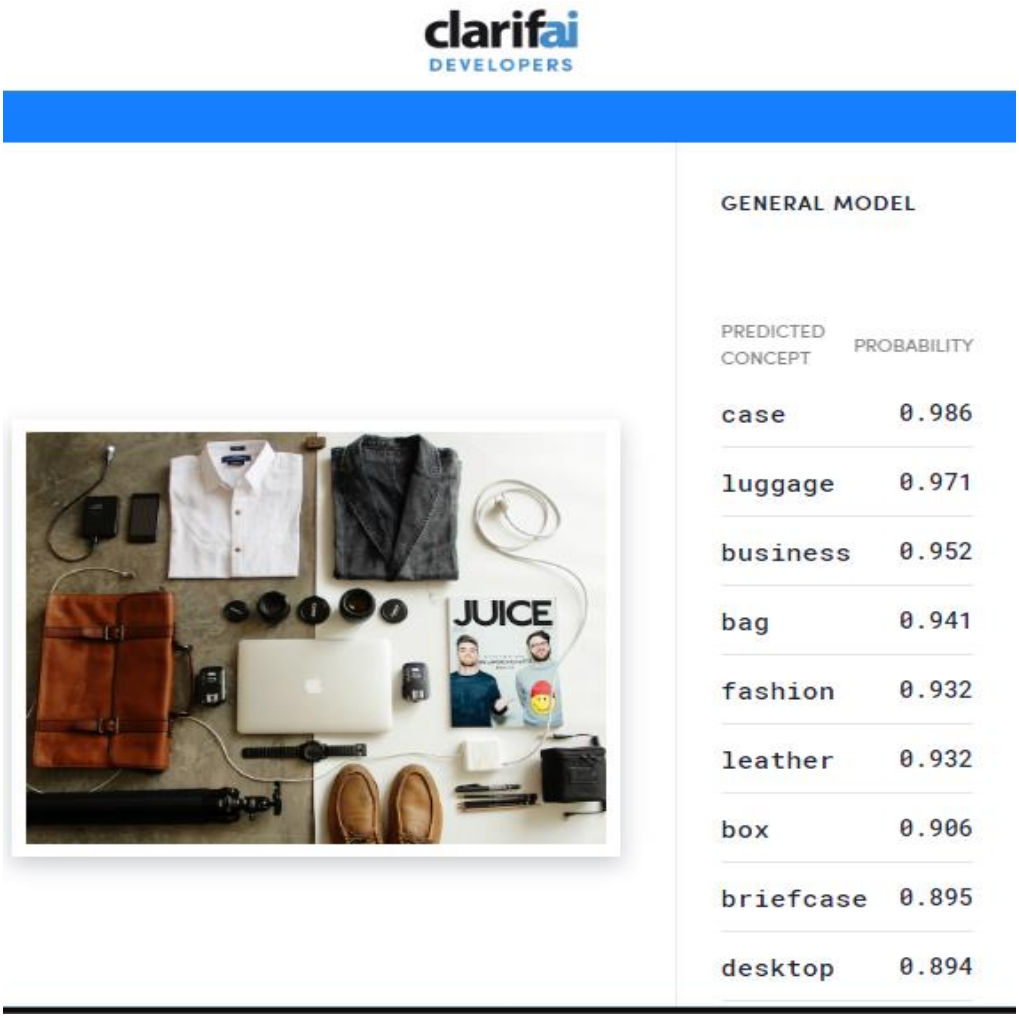
Apparel model bisa mengenali lebih dari 100 konsep yang berhubungan dengan *fashion* termasuk pakaian dan aksesoris. Contoh pengenalan objek menggunakan model *apparel* adalah sebagai berikut:



Gambar 2. 12 Contoh Apparel Model

2. General Model

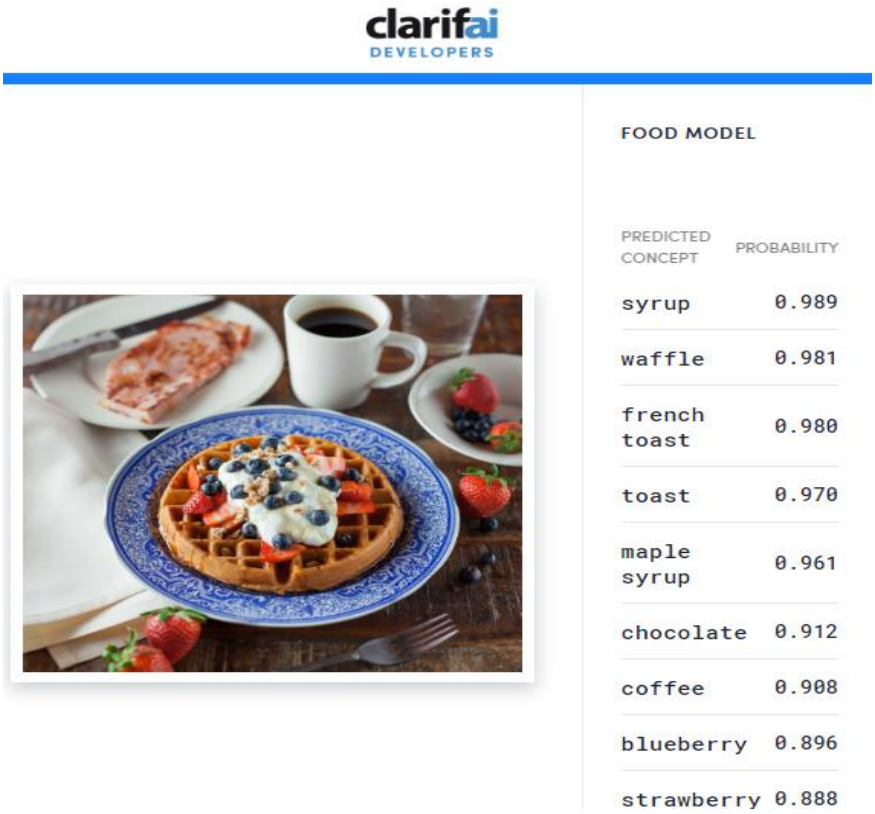
General model berisi berbagai macam *tag* di berbagai topik yang berbeda. Dalam kebanyakan kasus, *tag* dari model umum cukup akan mengenali apa yang ada di dalam gambar.



Gambar 2. 13 Contoh General Model

3. Food Model

Food model mengenali objek dan memberikan prediksi yang terkait dengan objek. Model ini dirancang untuk mengidentifikasi item makanan tertentu dan bahan-bahan yang terlihat.



Gambar 2. 14 Contoh Food Model

2.10 Open Weather

Open Weather adalah layanan *online* yang menyediakan API untuk data cuaca, termasuk data saat cuaca, ramalan, dan data historis untuk para pengembang layanan *web* dan aplikasi *mobile*. Sebagai sumber data, dengan menggunakan layanan meteorologi siaran, data mentah dari stasiun cuaca bandara, data mentah dari stasiun radar, dan data mentah dari stasiun cuaca resmi lainnya. Semua data yang diproses oleh OpenWeatherMap sehingga bisa memberikan

secara *online* data cuaca dan cuaca peta yang akurat, seperti untuk awan atau hujan. Selain itu, layanan ini difokuskan pada aspek sosial dengan melibatkan pemilik stasiun cuaca untuk menghubungkan mereka ke layanan untuk meningkatkan akurasi data cuaca [13].

Open Weather API memiliki beberapa fitur, diantaranya adalah sebagai berikut :

1. *Current weather data*

   Fitur ini memeberikan akses untuk data cuaca pada setiap lokasi melebihi 200.000 kota. Cuaca terkini akan diperbarui secara berkala secara global dan berdasarkan dari data stasiun cuaca. Data cuaca yang diberikan dalam bentuk format JSON, XML, atau HTML.

2. 5 *day* / 3 *hour forecast*

   Fitur ramalan cuaca lima hari bisa diakses pada banyak lokasi atau kota yang meliputi data cuaca setiap 3 jam. Data ramalan cuaca diberikan dalam bentuk firmat JSON dan XML.

3. *16 day / daily forecast*

   Ramalan cuaca 16 hari yang meliputi data cuaca setiap harinya bisa diakses pada banyak kota atau lokasi. Data ramalan cuaca tersedia dalam benuk JSON dan XML.

4. *Historical data*

   Open weather API menyediakan riwayat data cuaca lebih dari 37.000 kota.

5. *History bulk*

6. *Weather map layers*

   Peta cuaca meliputi curah hujan, awan, tekanan cuaca, suhu, angin dan lain-lain.

2.11 Instagram

Instagram adalah aplikasi untuk *photo-sharing* dan layanan jejaring sosial *online* yang memungkinkan penggunanya untuk mengambil gambar, menerapkan

filter digital dan berbagi hasilnya melalui berbagai layanan social media seperti Facebook, Twitter dan situs media lainnya [14].

Instagram pertama-tama dikembangkan di San Francisco oleh Kevin Systrom dan Brasil Michel "Mike" Krieger saat mereka berdua memilih untuk berfokus pada pembuatan aplikasi multi-fitur "HTML5 *check-in*" di proyek Burbn untuk fotografi *mobile* [14].

Pada bulan Januari 2011, Instagram menambahkan fitur "*hashtags*" (atau "tagar" dalam bahasa Indonesia, dikenal dengan lambang "#" untuk membantu pengguna menemukan foto yang bertopik serupa satu sama lain. Fitur *hashtag* ini kemudian dikenal dengan istilah "*tag*". Instagram juga mendorong pengguna untuk membuat *tag* spesifik dan relevan saat mempublikasikan foto mereka [14].

Instagram API menggunakan OAuth 2.0 Protocol yang sangat mudah digunakan daripada *protocol* sebelumnya dan *developer* bisa langsung menggunakan Instagram API dengan seketika [14].

Instagram API memerlukan otentikasi dengan permintaan khusus yang dibuat atas nama pengguna. Permintaan otentikasi memerlukan sebuah *access_token*. Token ini harus bersifat unik dan disimpan dengan aman [14].

2.12 Object Oriented Analysis Design

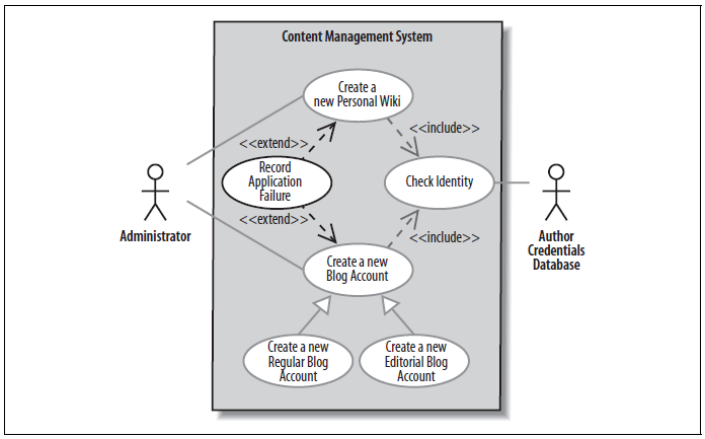
Analisis dan desain berorientasi objek adalah cara baru dalam memikirkan satu masalah dengan menggunakan model yang dibuat menurut konsep sekitar dunia nyata. Tujuan dari analisis berorientasi objek adalah untuk mengembangkan model yang menggambarkan perangkat lunak komputer karena bekerja untuk memenuhi seperangkat persyaratan yang ditentukan *user*. *Tools* yang dapat digunakan pada pendekatan analisis pengembangan sistem secara objek dapat menggunakan UML [15].

Unified Modelling Language (UML) adalah sebuah bahasa yg telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak. UML menggunakan *class* dan *operation object* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan

piranti lunak dalam bahasa bahasa berorientasi objek. Dalam membangun block UML ada 3 hal yang harus diperhatikan, yaitu *object* (memodelkan konsep), *relationship* (mengkoneksikan *object*), dan *diagram* (*grouping* yang saling mengkoneksikan antara *object* dan *relationship* [15].

2.12.1 Use Case Diagram

Use Case Diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem, yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di- *include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang serupa. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Dasar untuk menentukan sebuah *use case* adalah *use case* merupakan sesuatu yang menyediakan beberapa hasil terukur kepada pengguna atau sistem eksternal. *Use case* harus memiliki kriteria berhasil / gagal [16]. Berikut adalah contoh dari *use case* diagram :



Gambar 2. 15 Contoh Use Case Diaram

2.12.2 Use Case Scenario

Sebuah diagram yang menunjukkan *use case* dan aktor mungkin menjadi titik awal yang bagus, tetapi tidak memberikan detail yang cukup bagi desainer sistem untuk benar-benar memahami persis bagaimana sistem dapat terpenuhi. Cara terbaik untuk mengungkapkan informasi penting ini adalah dalam bentuk penggunaan *use case scenario* berbasis teks per *use case*-nya. Berikut adalah dasar format penulisan *use case scenario* [16] :

Tabel 2. 2 Dasar Pembangunan Use Case Scenario

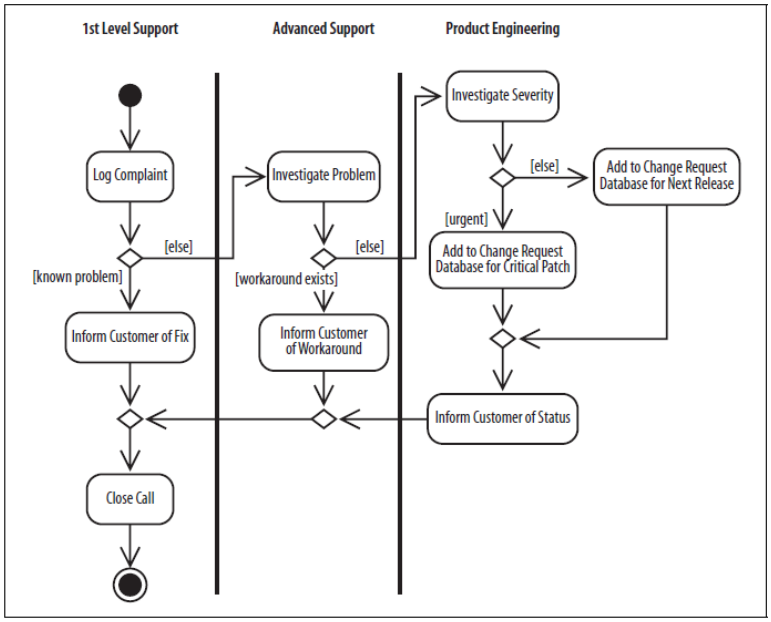
Use Case Name	Berisi nama dari *Use case* yang akan digunakan	
Goal In Context	Menjelaskan apa yang aktor coba untuk dapatkan dari *Use case*	
Description	Menjelaskan gambaran dari *Use case*	
Related Use Case	Daftar *Use case* yang berhubungan dengan *Use case* tersebut	
Successful End Condition	Kondisi *Use case* jika berhasil	
Failed End Condition	Kondisi *Use case* jika gagal	
Actors	Daftar aktor yang dapat mengakses *Use case*	
Trigger	Aktifitas yang dilakukan untuk mengawali *Use case*	
Main Flow	Step	Action
	1	Deskripsi urutan aksi dari aktifitas *Use case*
Extension	Step	Branching Action
	1.1	Deskripsi urutan aksi lain selain urutan aksi utama

2.12.3 Activity Diagram

Activity diagram menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing aliran berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

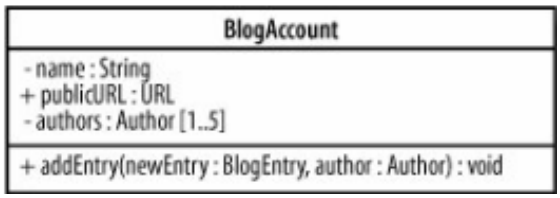
Standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu, untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis *horizontal* atau *vertikal*. *Activity diagram* dapat dibagi menjadi beberapa object *swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu [16].



Gambar 2. 16 Contoh Activity Diagram

2.12.4 Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (atribut/properti) satu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class* diagram menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti cont*a*inment, pewarisan, asosiasi, dan lain-lain [16].



Gambar 2. 17 Contoh Class Diagram

Class memiliki tiga area pokok:

1. Nama dan stereotype
2. Atribut
3. Metoda

Atribut dan metoda dari *class* dapat memiliki salah satu sifat berikut:

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metoda. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Sesuai dengan perkembangan *class* model, *class* dapat dikelompokkan menjadi *package*. *Class* memiliki tipe-tipe *relationship*, diantaranya [17] :

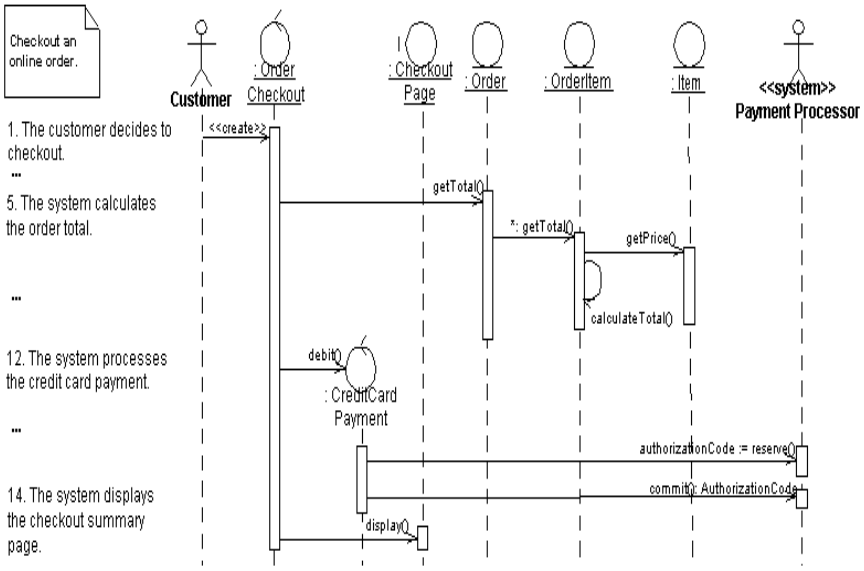
1. Asosiasi, yaitu hubungan statis antar *class*. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau *class* yang harus mengetahui eksistensi *class* lain.
2. Agregasi, yaitu hubungan yang menyatakan bagian terdiri atas dimana ketika satu *class* di *share* atau direferensikan kepada objek yang ada di *class* lain.
2. Pewarisan, yaitu hubungan hirarkis antar *class*. *Class* dapat diturunkan dari *class* lain dan mewarisi semua atribut dan metoda *class* asalnya dan menambahkan fungsionalitas baru, sehingga ia disebut anak dari *class* yang diwarisinya. Kebalikan dari pewarisan adalah generalisasi.
3. Komposisi, yaitu jenis relasi *class* diagram yang kuat dimana jika sebuah *class* tidak bisa berdiri sendiri dan harus merupakan bagian dari *class* yang lain, maka *class* tersebut memiliki relasi Composition terhadap *class* tempat dia bergantung tersebut. Sebuah relationship composition digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid.
4. Depedensi, salah satu jenis relasi class diagram yang lemah dimana objek dalam suatu *class* akan bekerja sangat singkat dengan objek yang ada pada *class* lain.

2.12.5 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem berupa *message* yang digambarkan terhadap waktu. *Sequence* diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence* diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan *Output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara *internal* dan *output* apa yang dihasilkan.

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*.

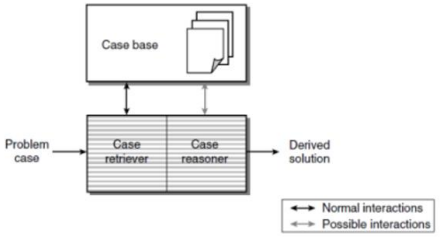
Activation bar menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity* [16].



Gambar 2. 18 Contoh Sequnce Diagram

2.13 Case-Based Reasoning (CBR)

Case-Based Reasoning (CBR) adalah metode penyelesaian masalah dengan menggunakan solusi masalah-masalah sebelumnya yang serupa. CBR sendiri adalah metode yang umum digunakan manusia dalam menyelesaiakan masalah sehari-hari [18].

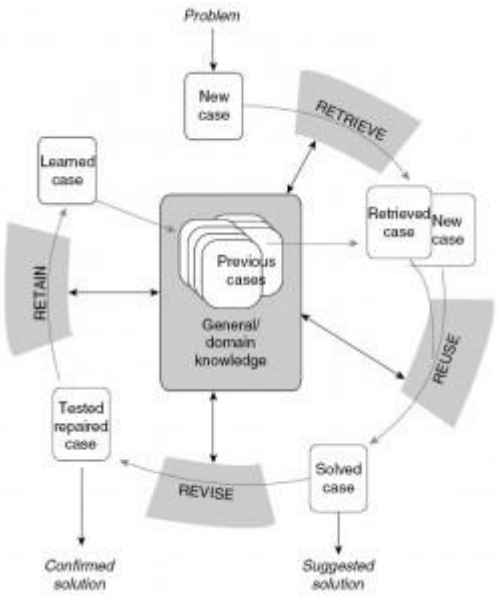


Gambar 2. 19 Komponen Case-Based Reasoning

Komponen penyelesaian masalah CBR memiliki dua bagian utama : *case retriever* dan *case reasoner*. *Case retriever* bertugas untuk mencari masalah-

masalah yang mirip dengan masalah yang dimasukkan pengguna di dalam basis pengetahuan (*case base*), sementara *case reasoner* bertugas untuk mencari solusi masalah pengguna dengan melihat atau menyesuaikan solusi masalah yang ditemukan *case retriever*.

Tahapan pada metode CBR (*Case-Based Reasoning*) ada 4 yaitu, *retrieve, reuse, revise* dan *retain.* Hubungan antara tahapan pada metode CBR (*Case-Based Reasoning*) dapat dilihat pada gambar berikut.



Gambar 2. 20 Siklus Case-Based Reasoning

1. *Retrieve* : yaitu mengambil kembali permaslaahan yang sama. Pada langkah ini dilakukan proses pencarian atau kalkulasi dari kasus-kasus yang memiliki kesamaan.
2. *Reuse* : yaitu menggunakan kembali informasi dan pengetahuan dalam kasus tersebut untuk mengatasi masalah baru. Pada langkah ini dicari solusi dari kasus serupa pada kondisi sebelumnya untuk permasalahan baru.
3. *Revise* : yaitu meninjau kembali solusi yang diberikan. Pada langkah ini dicari solusi dari kasus serupa pada kondisi sebelumnya untuk permasalahan yang terjadi kemudian.

4. *Retain* : yaitu mendalami bagian dari pengalaman sebelumnya untuk digunakan dalam pemecahan masalah berikutnya.

2.14 Algoritma K - Nearest Neighbor

Algoritma *Nearest Neighbor* (k-*nearest neighbor* atau k-NN) sebuah algoritma untuk melakukan klasifikasi terhadap objek berdasarkan data pembelajaran yang jaraknya paling dekat dengan objek tersebut. Kasus khusus di mana klasifikasi diprediksikan berdasarkan data pembelajaran yang paling dekat (dengan kata lain, k = 1) disebut algoritma *nearest neighbor*.

Algoritma *nearest neighbor* berdasarkan pada proses pembelajaran menggunakan analogi / *learning by analogi*. *Training* sampelnya dideskripsikan dalam bentuk atribut numerik n-dimensi. Tiap sampel mewakili sebuah titik pada ruang n- dimensi. Dengan cara ini, semua *training* sampel disimpan pada pola ruang n-dimensi. Ketika diberikan "*unknown*" sampel, k- *nearest neighbor classifier* mencari pola ruang K *training* sampel yang paling dekat "*unknown*" sampel tersebut. K training sampel ini adalah k *nearest neighbor* dari *unknown* sampel. *Unknown* sampel ditetapkan dengan *class* yang paling umum diantara k *nearest neighbors*-nya. Ketika k = 1, i sampel ditetapkan dengan class dari *training* sampel yang paling dekat dengan pola ruangnya. Algoritma *nearest neighbor* menyimpan semua *training* sampel dan tidak. Membangun *classifier* sampai sampel baru (*unlabeled*) perlu diklasifikasikan, sehingga algoritma *nearest neighbor* sering disebut dengan *instance-based* atau *lazy learners*. Rumus untuk menghitung bobot kemiripan (*similarity*) dengan *nearest neighbor* adalah

$$Similarity\ (problem, case) = \frac{(S1*W1)+(S2*W2)+(S3*W3)+(S4*W4)}{W1+W2+W3+W4}$$

Keterangan:

S = *similarity* (nilai kemiripan)

W = *weight* (bobot yang diberikan) .