BAB 2
TINJAUAN PUSTAKA

2.1. Lansia

2.1.1. Pengertian Lansia

Menurut *World Health Organisation* (WHO), lansia adalah seseorang yang telah memasuki usia 60 tahun keatas. Lansia merupakan kelompok umur pada manusia yang telah memasuki tahapan akhir dari fase kehidupannya. Kelompok yang dikategorikan lansia ini akan terjadi suatu proses yang disebut *aging process* atau proses penuaan.

Proses penuaan adalah siklus kehidupan yang ditandai dengan tahapan-tahapan menurunnya berbagai fungsi organ tubuh, yang ditandai dengan semakin rentannya tubuh terhadap berbagai serangan penyakit yang dapat menyebabkan kematian misalnya pada sistem kardiovaskuler dan pembuluh darah, pernafasan, pencernaan, endokrin dan lain sebagainya. Hal tersebut disebabkan seiring meningkatnya usia sehingga terjadi perubahan dalam struktur dan fungsi sel, jaringan, serta sistem organ. Perubahan tersebut pada umumnya mengaruh pada kemunduran kesehatan fisik dan psikis yang pada akhirnya akan berpengaruh pada ekonomi dan sosial lansia. Sehingga secara umum akan berpengaruh pada aktifitas dalam kehidupan sehari-hari [9].

2.1.2. Batasan Lansia

Menurut para ahli, batasan-batasan umur yang mencakup batasan umur lansia adalah sebagai berikut [10]:

a. Menurut Undang-Undang Nomor 13 Tahun 1998 dalam Bab 1 Pasal 1 ayat 2, lanjut usia adalah seseorang yang mencapai usia 60 (enam puluh) tahun ke atas.
b. Menurut *World Health Organization* (WHO), usia lanjut dibagi menjadi empat kriteria berikut : usia pertengahan (*middle age*) ialah 45-59 tahun, lanjut usia (*elderly*) ialah 60-74 tahun, lanjut usia tua (*old*) ialah 75-90 tahun, usia sangat tua (*very old*) ialah di atas 90 tahun.

c. Menurut Dra. Jos Masdani (Psikolog UI) terdapat empat fase yaitu : pertama (fase *inventus*) ialah 25-40 tahun, kedua (fase *virilities*) ialah 40-55 tahun, ketiga (fase *presenium*) ialah 55-65 tahun, keempat (fase *senium*) ialah 65 hingga tutup usia.
d. Menurut Prof. Dr. Koesoemato Setyonegoro, masa lanjut usia (*getiatric age*) dibagi menjadi tiga batasan umur, yaitu young old (70-75 tahun), old (75-80 tahun), dan very old (> 80 tahun).

2.1.3. Proses Penuaan

Penuaan adalah normal, dengan perubahan fisik dan tingkah laku yang dapat diramalkan yang terjadi pada semua orang pada saat mereka mencapai usia tahap perkembangan kronologis tertentu. Ini merupakan suatu fenomena yang kompleks multidimensional yang dapat diobservasi di dalam satu sel dan berkembang sampai pada keseluruhan sistem [11].

Tahap dewasa merupakan tahap tubuh mencapai titik perkembangan yang maksimal. Setelah itu tubuh mulai menyusut dikarenakan berkurangnya jumlah sel-sel yang ada di dalam tubuh. Sebagai akibatnya, tubuh juga akan mengalami penurunan fungsi secara perlahan-lahan. Itulah yang dikatakan proses penuaan [12].

Aging process atau proses penuaan merupakan suatu proses biologis yang tidak dapat dihindari dan akan dialami oleh setiap orang. Menua adalah suatu proses menghilangnya secara perlahan-lahan (*gradual*) kemampuan jaringan untuk memperbaiki diri atau mengganti serta mempertahankan struktur dan fungsi secara normal, ketahanan terhadap cedera, termasuk adanya infeksi. Proses penuaan sudah mulai berlangsung sejak seseorang mencapai dewasa, misalnya dengan terjadinya kehilangan jaringan pada otot, susunan saraf, dan jaringan lain sehingga tubuh ‘mati’ sedikit demi sedikit. Sebenarnya tidak ada batasan yang tegas, pada usia berapa kondisi kesehatan seseorang mulai menurun. Setiap orang memiliki fungsi fisiologis alat tubuh yang sangat berbeda, baik dalam hal pencapaian puncak fungsi tersebut maupun saat menurunnya. Umumnya fungsi fisiologis tubuh mencapai puncaknya pada usia 20-30 tahun. Setelah mencapai puncak, fungsi alat tubuh akan berada dalam kondisi tetap utuh beberapa saat, kemudian menurun sedikit demi sedikit sesuai dengan bertambahnya usia [13].

Pengaruh proses menua dapat menimbulkan berbagai masalah, baik secara biologis, mental, maupun ekonomi. Semakin lanjut usia seseorang, maka kemampuan fisiknya akan semakin menurun, sehingga dapat mengakibatkan kemunduran pada peran-peran sosialnya [14]. Oleh karena itu, perlu membantu individu lansia untuk menjaga harkat dan otonomi maksimal meskipun dalam keadaan kehilangan fisik, sosial dan psikologis [15].

2.2. *Activity Recognition*

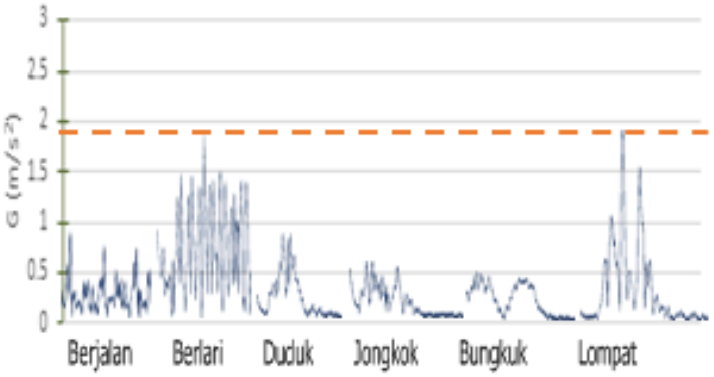
Activity recognition merupakan teknik yang digunakan untuk proses pendeteksian aktivitas fisik user. Tahap-tahap *activity recognition adalah data collection, feature extraction, dan data interpretation*. *Data collection* merupakan proses pengambilan data dengan menggunakan device berupa sensor. Data ini diharapkan dapat digunakan untuk dilakukan proses identifikasi yang bertujuan untuk menggambarkan sebuah entitas tertentu. *Feature extraction* merupakan proses pengolahan data sedemikian hingga agar data tersebut tidak terlalu konvergen atau terlalu divergen untuk dilakukan proses pendeteksian. Data interpretation merupakan proses pendeteksian aktivitas fisik user dengan menggunakan algoritma *fall detection* [16].

2.3. *Threshold*

Threshold merupakan kata dari bahasa Inggris, yang diartikan ke dalam bahasa Indonesia berarti ambang batas. Menurut Kamus Besar Bahasa Indonesia (KBBI), ambang batas diartikan sebagai "tingkatan batas yang masih dapat diterima atau ditoleransi" [17]. Dalam penelitian ini, nilai ambang batas yang digunakan adalah nilai ambang batas akselerasi dan nilai ambang batas kecepatan sudut dari *sensor accelerometer* dan *sensor gyroscope*.

Nilai ambang batas tersebut dipakai untuk membedakan gerak jatuh atau tidak jatuh [18]. Jika nilai akselerasi dan nilai kecepatan sudut melebihi nilai ambang batas yang telah ditetapkan maka terjadi peristiwa jatuh. Sedangkan, pencarian nilai ambang batas dilakukan dengan cara menganalisa semua nilai akselerasi dan kecepatan sudut dari gerak pada aktifitas sehari-hari. Dari semua data nilai akselerasi dan kecepatan sudut diambil nilai maksimalnya untuk dijadikan

nilai ambang batasnya [18]. Salah satu contoh pencarian nilai ambang batas dapat dilihat pada penelitian Mardi Hardjianto, dkk. [18], dimana nilai akselerasi yang didapatkan peneliti digambarkan sebagai berikut :



Gambar 2.1 Pencarian Nilai *Threshold*

Dari nilai akselerasi tersebut didapatkan bahwa nilai ambang batas akselerasi yang digunakan pada penelitian tersebut yaitu 1.8 m/s^2.

2.4. Algoritma *Threshold-Based Fall Detection*

Algoritma *threshold-based fall detection* adalah salah satu algoritma yang digunakan untuk mendeteksi gerakan terjatuh. Dalam penelitian ini, peneliti mengadopsi algoritma *threshold-based fall detection* dari penelitian sebelumnya [6], [19]. Dalam pendeteksian gerakan, algoritma *threshold-based fall detection* pertama-tama memerlukan data akselerasi dari sumbu x, y, z yang didapat melalui sensor *accelerometer* dan kecepatan sudut dari sumbu x, y, z yang didapat melalui *sensor gyroscope*.

$$\text{Ax} = \{ Ax_1, Ax_2, Ax_3, \ldots\ldots\ldots, Ax_n \} \quad (1)$$

$$\text{Ay} = \{ Ay_1, Ay_2, Ay_3, \ldots\ldots\ldots, Ay_n \} \quad (2)$$

$$\text{Az} = \{ Az_1, Az_2, AZ_3, \ldots\ldots\ldots, Az_n \} \quad (3)$$

$$\text{Gx} = \{ Gx_1, Gx_2, Gx_3, \ldots\ldots\ldots, Gx_n \} \quad (4)$$

$$\text{Gy} = \{ Gy_1, Gy_2, \text{G}y_3, \ldots\ldots\ldots, Gy_n \} \quad (5)$$

$$\text{Gz} = \{ Gz_1, Gz_2, GZ_3, \ldots\ldots\ldots, Gz_n \} \quad (6)$$

Data akselerasi kemudian diolah untuk mencari nilai *magnitude*-nya dengan menggunakan rumus :

$$AT_t = \sqrt{Ax^2 + Ay^2 + Az^2} \quad (7)$$

Sedangkan data kecepatan sudut diolah untuk mencari nilai *magnitude*-nya dengan menggunakan rumus :

$$GT_t = \sqrt{Gx^2 + Gy^2 + Gz^2} \quad (8)$$

Kemudian pencarian sudut dilakukan, sudut diperlukan untuk menentukan postur tubuh apakah seseorang sedang berdiri atau terjatuh dengan asumsi terjatuh pada lantai.

$$Pitch = \left(\frac{180}{\pi}\right) * \text{atan}\left(\frac{-Ax}{\sqrt{Ay^2+Az^2}}\right) \quad (9)$$

$$Roll = \left(\frac{180}{\pi}\right) * \text{atan}\left(\frac{Ay}{\sqrt{Ax^2+Az^2}}\right) \quad (10)$$

$$Yaw = \left(\frac{180}{\pi}\right) * \text{atan}\left(\frac{\sqrt{Ay^2+Ax^2}}{Az}\right) \quad (11)$$

Setelah seluruh didapatkan, maka pendeteksian gerakan terjatuh dapat dilakukan. Algoritma *threshold-based fall detection* yang diangkat dalam penelitian ini dibagi menjadi beberapa tahap.

1. Tahap membandingkan nilai *magnitude* akselerasi (AT_t) dengan *threshold* magnitude akselerasi (t_{AT}) dan nilai magnitude kecepatan sudut (GT_t) dengan *threshold* magnitude kecepatan sudut (t_{GT})
2. Tahap membandingkan nilai sudut dengan *threshold* sudut yang sudah ditentukan [6]
3. Jika tahap 1 dan 2 terpenuhi maka gerakan terjatuh pada tahap awal terdeteksi, sehingga memenuhi syarat untuk melanjutkan ke tahap berikutnya
4. Tahap pengecekan pergerakan setelah gerakan terjatuh telah terdeteksi. Tahap ini bertujuan untuk mengecek apakah setelah terjatuh, seseorang kembali melakukan pergerakan apakah tidak. Jika setelah 3 detik atau lebih dari kejadian terjatuh, seseorang tidak melakukan pergerakan apapun, maka dapat disimpulkan bahwa kejadian terjatuh sebenarnya terjadi [19]. Selain itu, tahap ini bertujuan juga untuk meningkatkan nilai akurasi pendeteksian kejadian terjatuh pada algoritma *threshold-based fall detection* [19]. Pengecekan dilakukan dengan cara membandingkan nilai akselerasi maksimum dan

minimum setelah 3 detik dari kejadian terjatuh. Berikut merupakan algoritma dari tahap ini [19] :

```
/* Long lie detection after a detected fall */
/* Assume lowLongLieThres and highLongLieThres are already determined
*/
/* Input: an array of acceleration values extracted in long lie period
*/
/* Output: long lie (true) or not (false) */
bool longlieDetection(float[] a) {
    float min = Math.min(a);
    float max = Math.max(a);
    if (min >= lowLongLieThres && max<= highLongLieThres) {
           return true;
    }
    return false;
}
```

2.5. Android

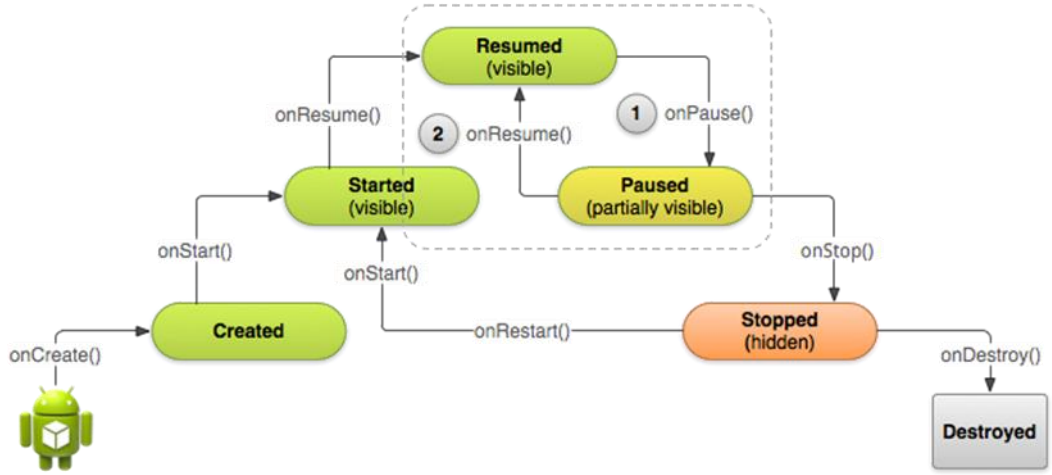
Android adalah sistem operasi untuk telepon seluler yang berbasis linux dan bersifat *Open Source* biasanya digunakan pada perangkat seluler layar sentuh seperti *smartphone*, *smartwatch* dan komputer tablet. Pada sejarahnya, Android dimulai saat perusahaan Android Inc mengembangkan sistem dari linux, pada tahun 2005 Google Inc Membeli Android Inc dan pada tahun 2007 sistem operasi ini dirilis. Kemudian untuk mengembangkan android, dibentuklah open *Handset Alliance*, konsorsium dari perusahaan – perusahaan perangkat keras, perangkat lunak, dan telekomunikasi yang bertujuan untuk memajukan standar yang terbuka pada perangkat seluler.

Android SDK (*Software Developmet Kit*) menyediakan *tools* dan API yang diperlukan untuk mengembangkan aplikasi pada *platform* android dengan menggunakan bahasa pemrograman Java.

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 [20].

Di dalam android terdapat activity dimana komponen ini memberi interaksi antara user dan aplikasi yang dibangun melalui *user interface*. *Activity* ini memiliki siklus hidup yang dinamakan *Android Life Cycle*. Flowchart siklus hidup tersebut dapat dilihat pada gambar berikut :



Gambar 2.2 Android *Life Cycle*

Terdapat beberapa state dalam siklus hidup android yang terjadi pada *Android Life Cycle*, akan tetapi hanya beberapa dari state tersebut yang menjadi statis diantaranya [20]:

1. *Resumed*

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

2. *Paused*

   Dalam 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

Dalam keadaan ini, aplikasi benar-benar tidak ditampilkan dan tidak terlihat oleh pengguna tetapi masih meninggalkan *service* pada *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().

2.6. *Application Programming Interface* (API)

Application Programming Interface, atau API, adalah rangkaian simbol yang diekspor dan tersedia bagi pengguna *library* untuk menulis aplikasinya. Desain API ini bisa dibilang bagian paling penting dari perancangan *library*, karena dapat mempengaruhi desain aplikasi yang dibangun di atas mereka [21]. 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. Keuntungan dengan menggunakan API adalah sebagai berikut [21] :

a. Portabilitas

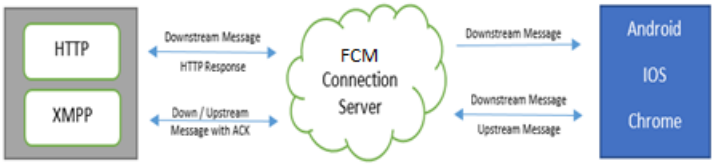
   Developer yang menggunakan API dapat menjalankan programnya dalam sistem operasi mana saja asalkan sudah terinstal API tersebut.

b. Lebih Mudah Dimengerti

   API menggunakan bahasa yang lebih terstruktur dan mudah dimengerti daripada bahasa *system call*. Hal ini sangat penting dalam hal editing dan pengembangan.

2.7. *Firebase Cloud Messaging*

Firebase Cloud Messaging (FCM) adalah layanan gratis dari google yang mengizinkan para pengembang aplikasi untuk mengirim pesan antara server dan aplikasi pada klien. Baik dari server ke aplikasi klien, maupun dari aplikasi klien ke server. Cara penggunaannya cukup mudah. Pada dokumentasi Firebase telah disediakan contoh pemanggilan FCM ke berbagai *platform* yang digunakan.



Gambar 2.3 Arsitektur FCM

Server FCM akan mengambil pesan dari server dan akan mengirimkan pesan tersebut ke aplikasi klien yang sudah dipasang di perangkat masing-masing. Server FCM akan menggunakan protokol HTTP dan XMPP untuk berkomunikasi dengan server FCM. Apabila perangkat klien tidak aktif (*offline*), maka pesan tersebut akan disimpan secara berurutan dan akan dikirimkan pada saat perangkat klien telah aktif kembali (*online*). Aplikasi klien yang menggunakan layanan FCM harus terlebih dahulu didaftarkan untuk mendapatkan unique identifier yang disebut token registrasi agar dapat menerima dan mengirim pesan. Token registrasi bisa didapatkan secara gratis dengan mendaftarkan nama aplikasi beserta nama *package* pada proyek Android.

2.8. SMS (*Short Message Service)*

Menurut Wahana Komputer [22], SMS (*Short Message Service*) merupakan salah satu layanan pesan teks yang dikembangkan dan distandardisasi oleh suatu badan bernama ETSI (*European Telecommunication Standards Institute*) sebagian dari pengembangan GSM (*Global System for Mobile Communication*) Phase 2, yang terdapat pada dokumentasi GSM 03.40 dan GSM 03.38. Fitur SMS ini memungkinkan perangkat Stasiun Seluler Digital (*Digital Cellular Terminal*, seperti ponsel) untuk dapat mengirim dan menerima pesan-pesan teks dengan panjang sampai dengan 160 karakter melalui jaringan GSM. Karakter yang dimaksud adalah alphabet A sampai Z, angka 0 sampai 9 dan spasi. Untuk karakter

non-Latin, seperti Arab, Kanji atau Mandarin dengan panjang sampai dengan 70 karakter.

SMS dapat dikirimkan ke perangkat stasiun seluler digital lainnya hanya dalam beberapa detik selama berada pada jangkauan pelayanan GSM. Lebih dari sekedar pengiriman pesan biasa, layanan SMS memberikan garansi SMS akan sampai pada tujuan meskipun perangkat yang dituju sedang tidak aktif yang dapat disebabkan karena sedang dalam kondisi mati atau berada di luar jangkauan layanan GSM. Dengan adanya fitur seperti ini maka layanan SMS juga cocok untuk dikembangkan sebagai aplikasi-aplikasi seperti : pager, e-mail, dan notifikasi voice mail, serta layanan pesan banyak pemakai (*multiple user*). Namun pengembangan aplikasi tersebut masih bergantung pada tingkat layanan yang disediakan oleh operator jaringan.

2.8.1. Karakteristik SMS

Ada beberapa karakteristik pesan SMS yaitu diantaranya :

1. Sebuah pesan singkat yang terdiri dari 160 karakter.
2. Pesan SMS dijamin sampai atau tidak sama sekali selayaknya e-mail, sehingga jika terjadi kegagalan sistem atau hal lain yang menyebabkan SMS tidak diterima akan diberikan informasi (*delivery report*) yang menyatakan SMS gagal dikirim.
3. Berbeda dengan fungsi *call* (panggilan), sekalipun saat mengirimkan SMS tetapi telepon genggam tujuan tidak aktif bukan berarti pengiriman SMS akan gagal. Namun SMS akan masuk ke antrian dahulu selama waktu belum time out. SMS akan segera dikirimkan jika telepon genggam sudah aktif.
4. *Bandwith* yang digunakan rendah.

2.8.2. Keuntungan SMS

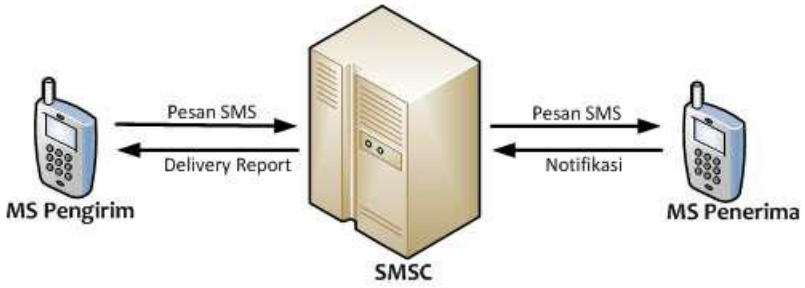
Adapun keuntungan dari SMS yaitu :

1. Pengiriman notifikasi dan peringatan (*alert*).
2. Penyampain pesan yang terjamin, handal dan komunikasi dengan biaya rendah.
3. Kemampuan untuk menyaring pesan dan menanggapi panggilan secara selektif.

4. Tingkat kegagalan kirim sangat kecil sehingga pesan kemungkinan besar akan sampai pada tujuan.
5. Pengiriman pesan ke nomor tujuan yang banyak dan berbeda dapat dilakukan pada waktu yang relatif singkat.

2.8.3. Cara Kerja SMS

Mekanisme dari sistem SMS ialah *store* and *forward*, dimana sistem dapat melakukan pengiriman short message dari satu terminal ke terminal lainnya. Hal ini dapat dilakukan berkat adanya sebuah entitas dalam sistem SMS yang bernama *Short Message Service Center* (SMSC), disebut juga *Message Center* (MC). SMSC merupakan sebuah perangkat yang melakukan tugas *store* and *forward traffic short message* seperti pada gambar berikut :



Gambar 2.4 Mekanisme Kerja SMS

Pengiriman pesan SMS secara *store* and *forward* yaitu pengirim SMS memasukkan pesan SMS dan nomor tujuan dan kemudian mengirimkannya (*store*) ke server SMS (*SMS Center*) yang kemudian bertanggung jawab mengirimkan pesan SMS tersebut (*forward*) ke nomor telepon tujuan.

Hal ini berarti bahwa pengirim dan penerima SMS tidak berada dalam status berhubungan (*connected*) satu sama lain, ketika akan saling bertukar pesan SMS. Pesan yang dikirim oleh pengirim ke SMSC yang kemudian menunggu untuk dapat meneruskan pesan tersebut ke penerima. Ketika status penerima dalam keadaan aktif pesan segera dikirim oleh SMSC ke nomor tujuan beserta isi pesan pengirim. Pengirim akan menerima *delivery report* bahwa pesan telah terkirim (*message sent*).

2.9. *SMS Gateway*

SMS Gateway adalah suatu platform yang menyediakan mekanisme untuk mengantar dan menerima SMS dari peralatan mobile (HP, PDA phone, dan lain - lain) yang menggunakan keyword tertentu. *SMS Gateway* adalah komunikasi SMS dua arah. *SMS Gateway* merupakan salah satu perkembangan fungsi yang dimiliki SMS. Secara umum *SMS Gateway* adalah sebuah sistem yang dipergunakan untuk memudahkan seseorang atau sebuah perusahaan mengirimkan pesan SMS yang sama dalam waktu yang bersamaan pada banyak orang. Selain itu, semakin berkembangnya fungsi SMS, *SMS Gateway* juga dapat dimanfaatkan untuk keperluan lain seperti melakukan polling, transaksi dengan sebuah sistem, pemantauan, dan sebagainya.

2.9.1. Keuntungan *SMS Gateway*

SMS Gateway merupakan pintu gerbang bagi penyebaran informasi dengan menggunakan SMS. *SMS Gateway* dapat menyebarkan pesan ke ratusan nomor secara otomatis dan cepat yang langsung terhubung dengan *database* nomor - nomor ponsel saja tanpa harus mengetik ratusan nomor dan pesan di ponsel karena semua nomor akan diambil secara otomatis dari *database* tersebut sehingga dapat menghemat waktu. Selain itu, kebutuhan untuk SMS Gateway juga tidak terlalu berlebihan karena hanya menggunakan sebuah PC dengan menggunakan sebuah ponsel, kabel data, kartu GSM, *SMS Gateway* dapat mengustomisasi pesan - pesan yang ingin dikirim. Dengan menggunakan program tambahan yang dapat dibuat sendiri, pengirim pesan dapat lebih fleksibel dalam mengirim berita karena biasanya pesan yang ingin dikirim berbeda - beda untuk masing - masing penerimanya.

2.10. Sensor

2.10.1. *Accelerometer*

Accelerometer adalah perangkat yang berfungsi untuk mengukur akselerasi. Akselerasi yang diukur dengan *accelerometer* belum tentu memiliki laju perubahan velositas. Sebaliknya, *accelerometer* mendapatkan akselerasi yang dimaksud dengan fenomena berat yang dialami oleh uji massa pada kerangka acuan

perangkat accelerometer. Sebagai contoh, *accelerometer* di permukaan bumi akan mengukur akselerasi g= 9.81 m/s2 lurus ke atas karena beratnya. Sebaliknya, *accelerometer* jatuh bebas ke bumi mengukur nol untuk akselerasinya [23].

Pada *smartphone*, *Accelorometer* merupakan sensor yang membaca perangkat sehingga dapat mengubah tampilan layar dari posisi *landscape* ke *potrait* atau sebaliknya dengan cukup memiringkan badan ponsel secara otomatis.

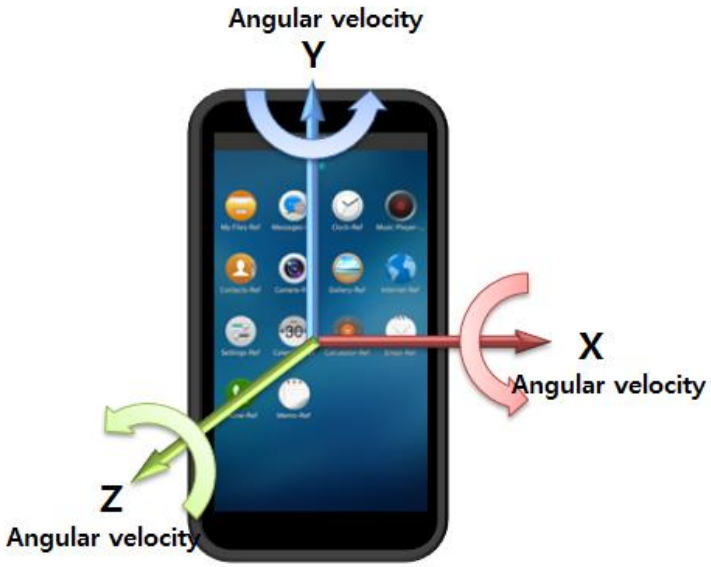
Pada *smartphone* berbasis android terdapat *sensor manager* yang berfungsi untuk mengaktifkan sensor accelorometer dalam mencari nilai kordinat x,y,z dengan kemiringan handphone.

Gambar 2.5 Sumbu *Accelerometer*

2.10.2. *Gyroscope*

Gyroscope adalah sebuah perangkat yang digunakan untuk mengukur orientasi dari sebuah objek. *Gyroscope* merupakan sebuah roda berat yang berputar pada jari-jarinya. Sebuah giroskop mekanis terdiri dari sebuah roda yang diletakkan pada sebuah bingkai. Roda ini berada di sebuah batang besi yang disebut dengan poros roda (*spin axis*). Ketika *gyroscope* digerakkan, maka ia akan bergerak mengitari porosnya. Poros tersebut terhubung dengan lingkaran-lingkaran yang disebut gimbal. Gimbal tersebut juga terhubung dengan gimbal lainnya pada dasar lempengan. Jadi saat piringan itu berputar, unit *gyroscope* itu akan tetap menjaga

posisinya seperti pada saat pertama kali *gyroscope* diputar [23]. Dalam Gambar 2.4 menunjukan poros sensor x, y,z dengan *smartphone*.



Gambar 2.6 Sumbu *Gyroscope*

2.11. GPS

Global Positioning System (GPS) menggunakan sistem satelit yang mengorbit planet untuk membantu penerima (*handset* Android dalam kasus ini) menentukan lokasi saat ini. Istilah GPS mengacu pada keseluruhan sistem GPS, yang terdiri dari satelit, penerima, dan stasiun kontrol yang memantau dan menyesuaikannya. Penerima ada pada telepon tidak berguna tanpa kinerja dari keseluruhan sistem GPS [24].

Secara umum, penerima GPS menggunakan informasi dari satelit GPS yang mengorbit bumi untuk menghitung lokasinya saat ini. Sistem GPS berisi 27 satelit yang terus mengorbit bumi, mengirimkan informasi ke calon penerima. Setiap satelit mengikuti jalur yang ditentukan, memastikan bahwa setidaknya empat satelit "*visible*" dari titik mana pun di bumi pada waktu tertentu. Mampu memiliki "*Line of Sight*" ke setidaknya empat satelit diperlukan untuk menentukan lokasi menggunakan GPS [24].

Untuk menghitung lokasinya, penerima GPS harus dapat menentukan jaraknya dari beberapa satelit. Ini dilakukan dengan menggunakan data ephemeris.

Termasuk dalam data yang ditransmisikan dari satelit, bersama dengan data posisi, adalah waktu di mana transmisi dimulai. Setiap satelit GPS berisi mekanisme ketepatan waktu yang sangat akurat yang memungkinkan satelit untuk menjaga waktu sinkron dengan sisa satelit. Untuk menghasilkan perhitungan lokasi yang akurat, satelit GPS dan penerima GPS harus disinkronkan dengan jam mereka. Bahkan perbedaan sedikit waktu dapat menyebabkan kesalahan besar saat menghitung lokasi.

2.12. Android Studio

Android Studio adalah sebuah IDE untuk Android Development yang diperkenalkan google pada acara Google I/O 2013. Android Studio merupakan pengembangan dari Eclipse IDE, dan dibuat berdasarkan IDE Java populer, yaitu IntelliJ IDE [25]. Android Studio merupakan IDE resmi untuk pengembangan aplikasi Android. Sebagai pengembangan dari Eclipse, Android Studio mempunyai banyak fitur-fitur baru dibandingkan dengan Eclipse IDE. Berbeda dengan Eclipse yang menggunakan Ant, Android Studio menggunakan Gradle sebagai *build environment*. Android studio juga memiliki kelebihan fitur-fitur yang lebih unggul sebagai berikut:

1. Menggunakan *Gradle-based build system* yang fleksibel.
2. Dapat *build multiple* APK.
3. *Template support* untuk Google Services dan berbagai macam tipe perangkat.
4. *Layout editor* yang lebih bagus.
5. *Built-in support* untuk Google Cloud Platform, sehingga mudah untuk integrasi dengan Google Cloud Messaging dan App Engine.
6. *Import library* langsung dari Maven *repository*.

Android Studio adalah kolaborasi antara JetBrains dan Google. Android Studio dibangun di atas IntelliJ JetBrain, dan fungsinya merupakan superset dari IntelliJ. Kebanyakan yang dapat Anda lakukan dengan IntelliJ, Anda juga dapat melakukannya di Android Studio. Android Studio bersifat revolusioner karena menyederhanakan proses pengembangan Android dan membuat pengembangan

Android jauh lebih mudah diakses daripada sebelumnya. Android Studio sekarang adalah IDE resmi untuk Android [25].

2.13. *Object Oriented Programming* (OOP)

Object Oriented Programming (OOP) atau yang dikenal dengan pemrograman berorientasi objek adalah metode implementasi di mana program disusun sebagai kumpulan objek yang kooperatif, yang masing-masing mewakili sebuah instance dari beberapa kelas, dan kelas mana yang menjadi anggota hirarki kelas yang disatukan melalui hubungan warisan. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek. Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar [26].

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bag administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri. Pemrograman orientasi-objek menekankan konsep berikut [26]:

1. *Class* (Kelas)

   Kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh *'Class of laptop'* adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam jenis/karakteristik dari laptop. Sebuah *Class* adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *Class* secara tipikal sebaiknya dapat dikenali oleh seorang non-programmer sekalipun

terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *Class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

2. *Property* (Properti)

   Property (atau disebut juga dengan atribut) adalah data yang terdapat dalam sebuah *class*. Melanjutkan analogi tentang laptop, *property* dari laptop bisa berupa merk, warna, jenis processor, ukuran layar, dan lain-lain. Property ini sebenarnya hanyalah variabel yang terletak di dalam *class*. Seluruh aturan dan tipe data yang biasa diinput kedalam variabel, bisa juga diinput kedalam property. Aturan tata cara penamaan *property* sama dengan aturan penamaan variabel.

3. *Method* (Metode)

   Method adalah tindakan yang bisa dilakukan didalam *class*. Jika menggunakan analogi *class* laptop kita, maka contoh *method* adalah: menghidupkan laptop, mematikan laptop, mengganti cover laptop, dan berbagai tindakan lain. *Method* pada dasarnya adalah *function* yang berada di dalam *class*. Seluruh fungsi dan sifat *function* bisa diterapkan kedalam *method*, seperti argumen/parameter, mengembalikan nilai (dengan *keyword return*), dan lain-lain.

4. *Object* (Objek)

   Object atau Objek adalah hasil cetak dari *class*, atau hasil ‘konkrit’ dari *class*. Jika menggunakan analogi *class* laptop, maka objek dari *class* laptop bisa berupa: laptop_andi, laptop_anto, dan lain-lain. Objek dari class laptop akan memiliki seluruh ciri-ciri laptop, yaitu property dan method-nya. Objek membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program berorientasi objek.

5. *Abstract* (Abstraksi)

   Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat 23 melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.

6. *Encapsulation* (Enkapsulasi)

   Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk mengakses keadaannya. Setiap objek mengakses interface yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

7. *Polymorphism* (Polimorfisme)

   Melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan. Metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.

8. *Inheritance* (Inheritas)

   Mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa harus mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas).

2.14. *Unified Modeling Language* (UML)

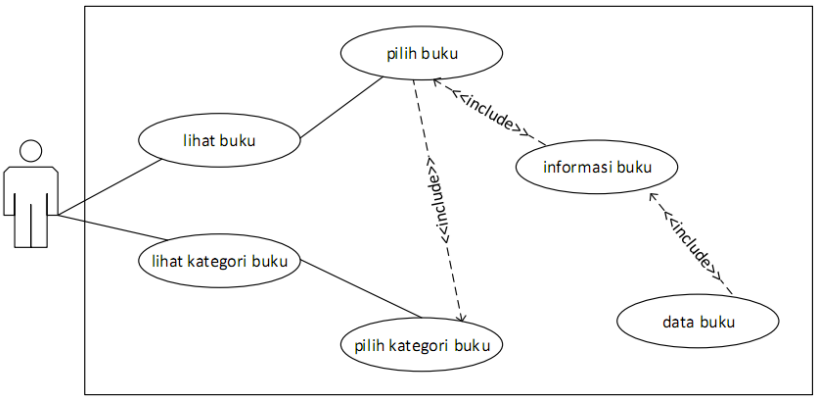
UML (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem. UML (*Unified Modeling Language*) merupkan himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya [27]. UML juga dapat diartikan metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.

UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM. UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson [27]. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan system. UML menyediakan beberapa macam diagram untuk memodelkan aplikasi berorientasi objek, yaitu [27]:

1. *Use case Diagram*

   Use case diagram adalah gambaran *graphical* dari beberapa atau semua Aktor, *use case*, dan interaksi diantara komponen-komponen tersebut yang memperkenalkan suatu sistem yang akan dibangun. *Usecase diagram* menjelaskan manfaat suatu sistem jika dilihat menurut pandangan orang yang

berada di luar sistem. Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar.

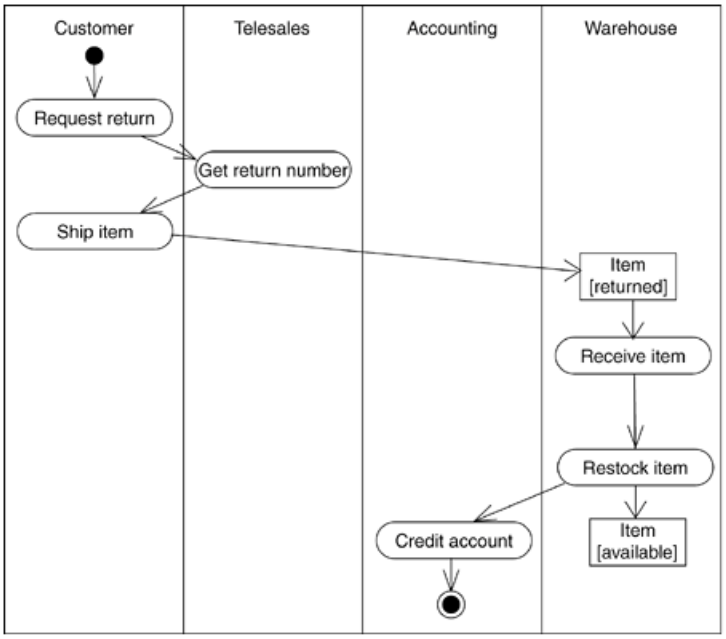


Gambar 2.7 Contoh *Use Case Diagram*

Usecase diagram dapat digunakan selama proses analisis untuk menangkap *requirement* sistem dan untuk memahami bagaimana sistem seharusnya bekerja. Selama tahap desain, *usecase diagram* berperan untuk menetapkan perilaku (*behavior*) sistem saat diimplementasikan. Dalam sebuah model mungkin terdapat satu atau beberapa use-case diagram. Kebutuhan atau *requirements system* adalah fungsionalitas apa yang harus disediakan oleh sistem kemudian didokumentasikan pada model *usecase* yang menggambarkan fungsi sistem yang diharapkan (use case), dan yang mengelilinginya (*Aktor*), serta hubungan antara Aktor dengan use case (use case *diagram*) itu sendiri.

2. *Activity Diagram*

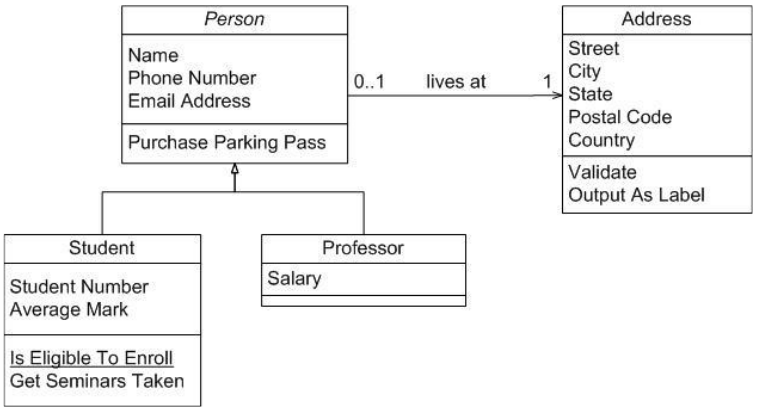
Activity Diagram memiliki pengertian yaitu lebih fokus kepada menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Dipakai pada business modeling untuk memperlihatkan urutan aktifitas proses bisnis. Memiliki struktur diagram yang mirip *flowchart* atau data *flow diagram* pada perancangan terstruktur. Memiliki pula manfaat yaitu apabila kita membuat diagram ini terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan. *Activity diagram* dibuat berdasarkan sebuah atau beberapa use case.



Gambar 2.8 Contoh *Activity Diagram*

3. *Class Diagram*

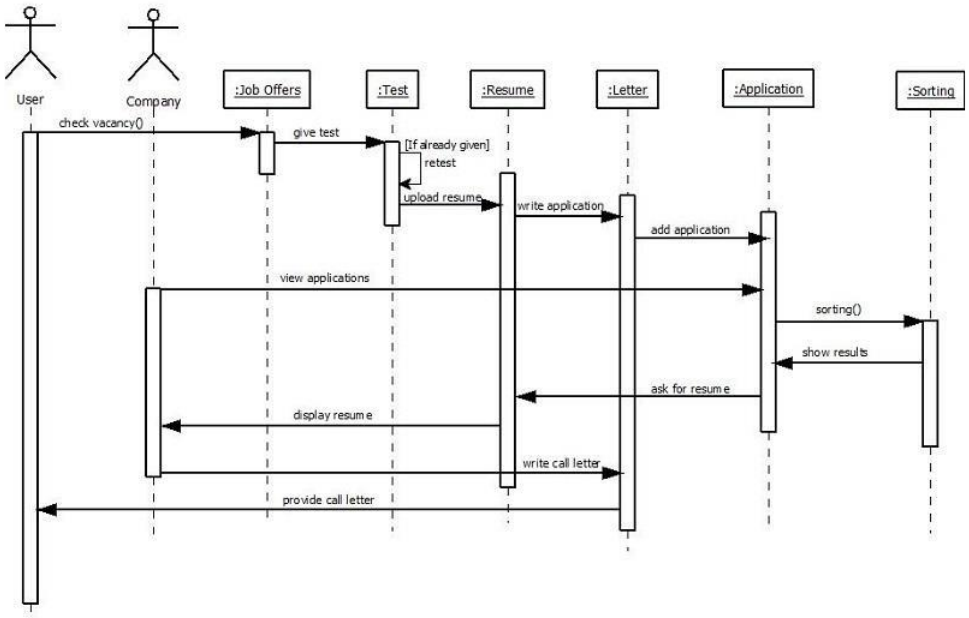
Class diagram adalah sebuah *Class* yang menggambarkan struktur dan penjelasan *Class*, paket, dan objek serta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. *Class diagram* juga menjelaskan hubungan antar *Class* dalam sebuah sistem yang sedang dibuat dan bagaimana caranya agar mereka saling berkolaborasi untuk mencapai sebuah tujuan.



Gambar 2.9 Contoh *Class Diagram*

4. *Sequence Diagram*

Sequence Diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) 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. Sequence diagram menjelaskan secara detil urutan proses yang dilakukan dalam sistem untuk mencapai tujuan dari use case: interaksi yang terjadi antar class, operasi apa saja yang terlibat, urutan antar operasi, dan informasi yang diperlukan oleh masing-masing operasi.



Gambar 2.10 Contoh *Sequence Diagram*

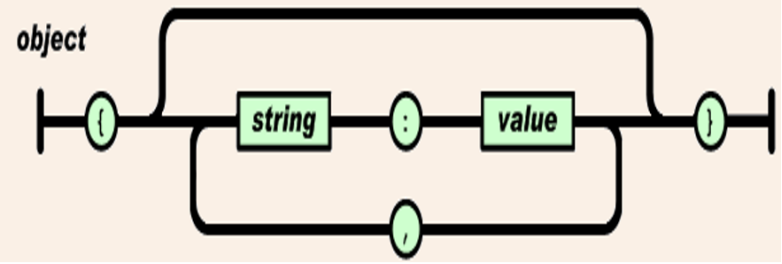
2.15. JSON

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. 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#, Java, JavaScript, Perl, Python dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data. JSON terbuat dari dua struktur [28]:

1. Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel
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*).
3. Struktur-stuktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemrograman moderen mendukung struktur 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:
   a. Objek

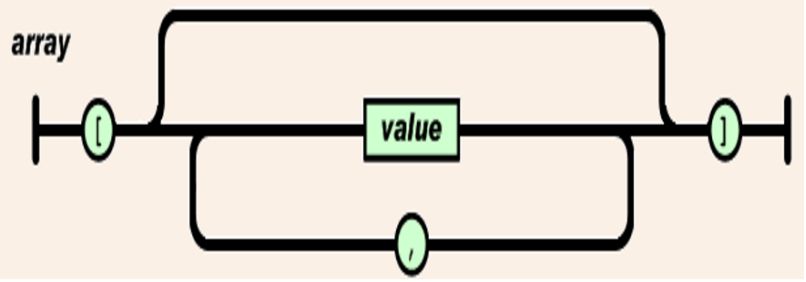
      Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma). *hash* (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.



Gambar 2.11 Objek JSON

b. Larik

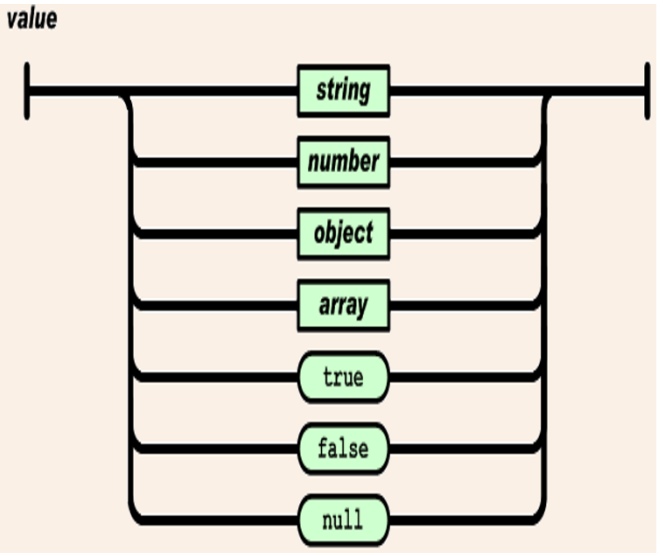
Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung siku buka) dan diakhiri dengan] (kurung siku tutup). Setiap nilai dipisahkan oleh , (koma).



Gambar 2.12 Larik JSON

c. 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.

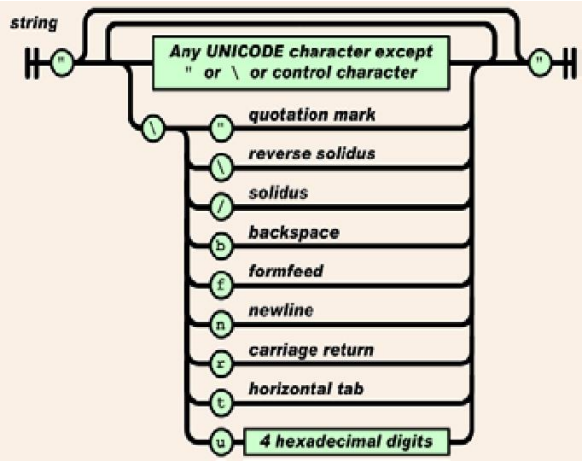


Gambar 2.13 Nilai JSON

d. *String*

String adalah kumpulan dari nol atau lebih karakter *Unicode*, yang dibungkus dengan tanda kutip ganda. Di dalam *string* dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter

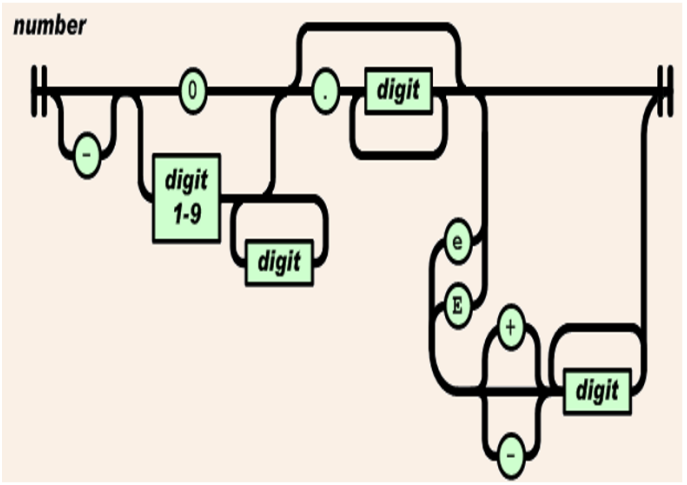
mewakili karakter tunggal pada *string*. *String* sangat mirip dengan *string* C atau Java.



Gambar 2.14 *String* JSON

e. Angka

Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.



Gambar 2.15 Angka JSON

2.16. Java

Java adalah salah satu bahasa pemograman berorientasi objeck (OOP-*Object Oriented Programming*). Paradigma OOP menyelesaikan masalah dengan merepresetasikan masalah ke model objek.

Java pertama kali diluncurkan sebagai bahasa pemrograman umum (*general purpose programming language*) dengan kelebihan dia bisa dijalankan di web *browser* sebagai *applet*. Sejak awal, para pembuat *Java* telah menanamkan visi mereka ke dalam *Java* untuk *small embedded customer device* seperti TV, telepon, radio, dan sebagainya supaya dapat berkomunikasi satu sama lain. Langka pertama yang diambil oleh Sun Microsistem adalah dengan membuat JVM (*JavaVirtual machine*) yang kemudian diimplementasikan dalam bentuk JRE (*Java Runtime Enviroment*). JVM adalah lingkungan tempat eksekusi program Java berlangsung dimana para obyek saling berinteraksi satu dengan lainnya. *Virtual machine* inilah yang menyebabkan *Java* mempunyai kemampuan penanganan memori yang lebih baik, keamanan yang lebih tinggi serta portabilitas yang besar. Apabila kita hanya ingin menjalankan program *Java*, maka kita cukup memiliki JRE saja. Tapi seandainya kita ingin mengembangkan prangkat lunak sendiri, JRE saja tidak cukup [29].

2.17. MySQL

MySQL merupakan *server* basis data dimana pemprosesan data terjadi di *server*, dan *client* hanya mengirimkan data serta meminta data. Oleh karena pemprosesan terjadi di server sehingga pengaksesan data tidak terbatas. MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL atau DBMS yang cepat dan mudah digunakan yang digunakan. Kecepatan adalah fokus utama pengembang MySQL sejak awal. Demi kecepatan, pengembang MySQL membuat keputusan untuk menawarkan fitur yang lebih sedikit daripada pesaing utama mereka (seperti Oracle dan Sybase). Namun, meskipun MySQL kurang berfitur lengkap daripada pesaing komersialnya, namun fitur ini hampir semua fitur yang dibutuhkan oleh pengembang database. Lebih mudah untuk menginstal dan menggunakan daripada pesaing komersialnya, dan selisih harga sangat kuat dengan dukungan MySQL [30].

MySQL termasuk dalam kategori manajemen basis data yaitu basis data yang terstruktur dalam pengolahan dan penampilan data. Ada beberapa alasan mengapa MySQL menjadi program yang *database* yang sangat popular dan digunakan oleh banyak orang. Kelebihan dari MySQL adalah sebagai berikut [30]:

1. MySQL merupakan basis data yang memiliki kecepatan yang tinggi dalam melakukan pemprosesan data, dapat diandalkan dan mudah digunakan serta mudah dipelajari.
2. MySQL mendukung banyak pemrograman seperti C, C++, Perl, Phyton, Java, Visual Basic dan PHP.
3. MySQL dapat menangani basis data dengan skala yang sangat bese dengan jumlah record mencapai lebih dari 50 juta.
4. MySQL merupakan *software* basis data yang bersifat bebas atau gratis tanpa bayaran, jadi kita tidak perlu susah mengeluarkan uang untuk sekedar membayar lisensi.