# BAB II

# LANDASAN TEORI

## 2.1 Emergency Situation

Secara bahasa, emergency situation berarti situasi darurat. Situasi dimana orang-orang merasa dalam bahaya, dalam tekanan, terancam, terdesak, membutuhkan pertolongan atau berada diantara hidup dan mati. Dalam kehidupan sehari-hari, terdapat banyak keadaan darurat seperti terjadi kecelakaan yang membutuhkan pertolongan secara cepat, terjadi tindakan kriminalitas ataupun terjadi kebakaran pada sebuah bangunan.

Di banyak negara seperti Amerika dan negara lainnya, terdapat satu layanan terintegrasi yang dapat dihubungi secara cepat. Layanan itu berupa nomor telepon yang dapat dihubungi ketika keadaan darurat oleh masyarakat umum untuk mendapatkan bantuan dari berbagai pihak seperti polisi, pemadam kebakaran dan pertolongan medis atau pengangkutan (evakuasi) ke rumah sakit. Di banyak negara hanya ada 1 nomor telepon darurat sehingga mudah diingat. Dengan nomor telepon darurat tunggal ini, masyarakat umum yang memerlukan bisa meminta bantuan dari jasa-jasa penanganan keadaan darurat setempat. Di indonesia belum menerapkan integrasi nomor darurat, sehingga berbagai nomor disediakan untuk membantu masyarakat bergantung pada kebutuhannya. Dalam penggunaanya, ini akan merepotkan dan memakan waktu lebih banyak. Sebab, hitungan detik atau menit akan menentukan kondisi seseorang. Waktu yang singkat itu bahkan jadi penentu hidup mati korban[3]. Dari literatur tersebut dapat disimpulkan bahwa terdapat tiga kategori *emergency* yaitu rumah sakit, kantor polisi dan kantor pemadam kebakaran.

Emergency situation yang akan digunakan pada sistem ini adalah nama sebuah aplikasi yang akan membantu orang-orang saat berada dalam keadaan darurat. Aplikasi akan lebih spesifik pada kebutuhan masing-masing orang. Sehingga dapat menjadi pilihan lain yang dapat digunakan selain nomor darurat yang saat ini telah ada.

## 2.2 Android

Android adalah sebuah sistem operasi untuk perangkat *mobile* berbasis Linux yang mencakup sistem operasi, *middleware*, dan aplikasi. Android menyediakan *platform* yang terbuka bagi para pengembang untuk menciptakan aplikasinya[1].

Salah satu penyebab mengapa sistem operasi Android begitu gampang diterima oleh pasar dan dengan cepatnya berkembang, itu dikarenakan android menggunakan bahasa pemrograman Java serta kelebihannya sebagai *software* yang menggunakan basis kode komputer yang bisa didistribusikan secara terbuka (*open source*) sehingga pengguna dapat membuat aplikasi baru didalamnya. Dan hal tersebut mengakibatkan banyaknya pengembang *software* yang berbondong untuk mengembangkan aplikasi berbasis Android. Sehingga saat ini bila dibanding dengan OS yang lain untuk perangkat handphone dan PC tablet. Android adalah yang mempunyai dukungan aplikasi dan *game* non berbayar terbanyak yang bisa diunduh oleh penggunanya melalui Google Play. Dengan terdapatnya fitur seperti *browser*, MMS, SMS, GPS, dan lain-lain maka sangat memudahkan penggunanya untuk mendapatkan informasi, mengetahui posisi, serta juga komunikasi antar para pengguna[1]. Aplikasi Android dapat dikembangkan pada berbagai sistem operasi, diantaranya adalah:

1. Windows XP/Vista/7
2. Mac OS X (Mac OS X 10.48 atau yang lebih baru)
3. Linux

### 2.2.1 Sejarah

Awalnya, Google Inc. membeli Android Inc., pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan Android, dibentuklah *Open Handset Alliance*, dimana kerjasama tersebut merupakan gabungan dari kurang lebih 34 perusahaan termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile, dan Nvidia[1].

Pada saat perilisan perdananya, 5 November 2007, Android bersama *Open Handset Allia*nce menyatakan akan mendukung pengembangan standar terbuka pada perangkat seluler. Di lain pihak, Google merilis kode - kode Android di bawah lisensi Apache, sebuah lisensi piranti lunak 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) [1].

Telepon selular pertama yang menggunakan sistem operasi Android adalah HTC Dream yang dirilis pada 22 Oktober 2008. Pada 9 Desember 2008, diumumkan anggota baru yang bergabung dalam program kerja Android ARM Holdings, Atheros Communication yang diproduksi oleh Asustek Computer Inc, Garmin Ltd, Softbank, Sony Ericsson, Toshiba Corp dan Vodafone Group Plc[1]. Hingga saat ini terdapat beberapa versi dari sistem operasi Android, antara lain:

1. Android versi 1.1

   Dirilis pada 9 Maret 2009. Android versi ini dilengkapi dengan adanya jam, *alarm*, *voice search*, pengiriman pesan dengan Gmail dan pemberitahuan *email*[1].

2. Android versi 1.5 (Cupcake)

   Dirilis pada Mei 2009. Terdapat pembaruan dari versi 1.1 diantaranya adalah fitur *upload* video ke Youtube dan gambar ke Picasa langsung dari telepon, dukungan bluetooth A2DP, kemampuan terhubung secara otomatis ke *headset* bluetooth, animasi layar, dan *keyboard* pada layar yang dapat disesuaikan dengan sistem[1].

3. Android versi 1.6 (Donut)

   Dirilis pada September 2009. Pembaruan yang terdapat pada versi ini diantaranya adalah proses pencarian yang lebih baik, penggunaan baterai indikator. Fitur lainnya adalah memungkinkan pengguna untuk memilih foto yang akan dihapus, kamera, camcorder dan galeri yang diintegrasikan, CDMA/EVDO, VPN, Gestures, Text-to-speech *engine*[1].

4. Android versi 2.1 (Éclair)

   Dirilis pada 3 Desember 2009. Perubahan yang dilakukan adalah pengoptimalan *hardware*, peningkatan Google Maps 3.1.2, perubahan UI dengan browser baru dan dukungan HTML5, daftar kontak yang baru, dukungan *flash* untuk kamera 3.2 MP, *digital zoom* dan bluetooth 2.1[1].

5. Android versi 2.2 (Froyo)

   Dirilis pada 20 Mei 2010. Versi Android inilah yang cukup banyak digunakan sebagai standar sistem operasi mereka. Terdapat perubahan yang cukup signifikan dari versi sebelumnya diantaranya adalah kerangka aplikasi memungkinkan penggunaan dan penghapusan komponen yang tersedia, Dalvik Virtual Machine (DVM) yang dioptimalkan untuk perangkat mobile, grafik di 2D dan 3D berdasarkan *libraries* OpenGL, SQLite, mendukung berbagai format audio dan video, GSM, bluetooth, EDGE, 3G, Wifi, kamera, Global Positioning System (GPS), kompas dan *accelerometer*[1].

6. Android versi 2.3 (GingerBread)

   Dirilis pada 6 Desember 2010. Beberapa perbaikan fitur dari versi sebelumnya adalah SIP-based VoIP, Near Field Communications (NFC), gyroscope, *multiple cameras support*, *mixable audio effect* dan *download manager*[1].

7. Android versi 3.0 (Honeycomb)

   Dirilis tahun 2011. Android versi ini dirancang khusus untuk tablet, sehingga terdapat perbedaan dari fitur UI (*User Interface*). Honeycomb sengaja dibuat untuk layar yang lebih besar dan juga dapat mendukung *multiprocessor*[1].

8. Android versi 4.0 (Ice Cream)

   Dirilis pada 19 Oktober 2011. Pada versi ini terdapat beberapa perbaikan yang dilakukan google, seperti kemampuan untuk mengakses aplikasi secara langsung dari *lock screen*, fitur yang memungkinkan pengguna untuk membuka perangkat menggunakan perangkat lunak pengenal

wajah(*face detection*), wifi direct, serta kemampuan untuk merekam video 1080p bagi perangkat Android tertentu[1].

9. Android versi 4.1 (Jelly Bean)

   Google mengumumkan Android 4.1 (Jelly Bean) dalam konferensi Google I/O pada tanggal 27 Juni 2012. Berdasarkan kernel Linux 3.0.31, Jelly Bean adalah pembaruan penting yang bertujuan untuk meningkatkan fungsi dan kinerja antarmuka pengguna. Pembaruan ini diwujudkan dalam "Proyek Butter", perbaikan ini termasuk antisipasi sentuh, *triple buffering*, perpanjangan waktu vsync, dan peningkatan *frame rate* hingga 60 fps untuk menciptakan UI yang lebih halus. Android 4.1 Jelly Bean dirilis untuk *Android Open Source Project* pada tanggal 9 Juli 2012. Perangkat pertama yang menggunakan sistem operasi ini adalah tablet Nexus 7, yang dirilis pada 13 Juli 2012[4].

10. Android versi 4.4 (KitKat)

    Dirilis pada 31 Oktober 2013. Pada versi ini tersedia beberapa peningkatan seperti pembaharuan antar muka, Pengembangan NFC, Peningkatan aksesibilitas API dan Mesin *virtual* eksperimental baru yaitu Android Runtime[4].

11. Android versi 5.0 (Lollipop)

    Dirilis pada tanggal 3 November 2014. Pada versi ini terdapat beberapa perubahan dan perbaikan yang dilakukan oleh google. Desain antarmuka baru yang dinamaan "Material Desgin". 64-bit ART *compiler*. Project volta, yang berguna untuk meningkatkan daya hidup baterai 30% lebih tahan lama. Serta *factory reset protection*, yaitu fitur yang berguna ketika *smartphone* hilang, ia tidak bisa direset ulang tanpa memasukan id google dan kata sandi[4].

12. Android versi 6.0 (Marshmallow)

    Android versi ini diumumkan oleh google pada bulan Oktober 2015. Android marshmallow lebih fokus pada pengembangan untuk menambah pengalaman pengguna dalam menggunakan android dan beberapa fitur baru seperti fitur untuk modifikasi *permission* yang diinginkan oleh

aplikasi, metode baru untuk memperpanjang masa hidup baterai serta *support* untuk *fingerprint recognition*[4].

### 2.2.2 Arsitektur Android

Secara garis besar arsitektur android dapat dijelaskan sebagai berikut:

1. Applications dan Widgets

   *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. Di *layer* terdapat aplikasi inti termasuk klien email, program SMS, kalender, peta, browser, kontak, dan lain-lain. Semua aplikasi ditulis menggunakan bahasa pemograman java[1].

2. Applications Frameworks

   *Applications Frameworks* ini adalah *layer* dimana para pembuat aplikasi melakukan pengembangan/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[1].

3. Libraries

   *Libraries* ini adalah *layer* dimana fitur-fitur Android berada, biasanya para pembuat aplikasi mengakses libraries untuk menjalankan aplikasinya[1]. Berjalan diatas kernel, *Layer* ini meliputi berbagai *library* C/C++ inti seperti Libc dan 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 ES 1.0 API's.

4. Android Run Time

*Layer* yang membuat aplikasi Android dapat dijalankan dimana dalam prosesnya mengggunakan implementasi Linux. Dalvik Virtual Machine (DVM) merupakan mesin yang membentuk dasar kerangka aplikasi Android[1]. 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 menerjemahkan bahasa Java / C yang ditangani oleh *Core Libraries*[1].

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[1].

5. Linux Kernel

Linux kernel adalah *layer* dimana inti dari *operating system* dari Android itu berada. Berisi file-file sistem yang mengatur sistem *processing*, memori, *resource*, *drivers*, dan sistem-sistem operasi Android lainnya. Linux kernel yang digunakan Android adalah Linux kernel release 2.6[1].

## 2.3 Bahasa Pemrograman Java

Java adalah suatu teknologi yang merupakan bahasa pemrograman, dan sekaligus suatu *platform*. Sebagai bahasa pemrograman, Java dikenal sebagai bahasa pemrograman tingkat tinggi. Java merupakan bahasa pemrograman berorientasi objek yang merupakan paradigma pemrograman masa depan. Sebagai bahasa pemrograman Java dirancang menjadi handal dan aman. Java juga di rancang agar dapat dijalankan pada semua *platform*[5].

Keunggulan Java dibandingkan dengan bahasa pemrograman yang lain adalah:

1. Berorientasi Objek
2. *Multi Platform*
3. Berbasis GUI (*Graphic User Interface*)
4. Dapat digunakan pada pengembangan *website*.
5. Aman
6. Java menyediakan fitur *multithread*, yang dapat digunakan untuk menjalankan perintah secara bersamaan.
7. Java menyediakan fitur *error-handling*, yaitu penanganan *error* pada program.
8. Dinamis
9. Aplikasi Java dapat didistribusikan dengan mudah

### 2.3.1 Teknologi Java

Teknologi java dibagi menjadi tiga jenis, yaitu:

1. Java 2 Standard Edition (J2SE)

   J2SE adalah pemrograman berbasis *console* dan *dekstop*. Tidak hanya sebatas itu saja, karena J2SE adalah *basic* dari JAVA. J2SE atau yang biasa dikenal sebagai bahasa Java, merupakan teknologi Java yang dirancang untuk berjalan diatas PC dan *workstation* yang dapat berjalan di *platform* sistem operasi Linux, Maxchintos, Windows, dan lain-lain[5].
2. Java 2 Enterprise Edition (J2EE)

   Untuk aplikasi berbasis *web*, aplikasi sistem tersebar dengan beraneka ragam klien dengan komplektivitas yang tinggi. Merupakan *superset* dari Standar Java. Teknologi Java yang satu ini digunakan untuk pengembangan aplikasi - aplikasi *enterprise*. J2EE meliputi beberapa teknologi pendukung, yaitu Java Server Pages (JSP), Java Servlet, Java CORBA dan lain-lain[5].
3. Java 2 Micro Edition (J2ME)

   J2ME digunakan untuk pengembangan sistem mikro dan *embedded*, meliputi *handphone*, *pager*, PDA, dan lain-lain. Teknologi ini kemudian

juga dibagi menjadi dua bagian besar, yaitu CLDC *Technology* (meliputi MIDP yang sangat terkenal, Bluetooth, dan lain-lain) dan CDC *Technology* (meliputi JDBC, yaitu teknologi *database* dan RMI) [5].

### 2.3.2 Eclipse

Eclipse adalah sebuah *Integrated Development Environment* (IDE) untuk mengembangkan perangkat lunak dan dapat dijalankan di semua *platform*. Berikut ini adalah sifat dari Eclipse:

1. *Multi-platform*, target sistem operasi Eclipse adalah Microsoft Windows, Linux, Solaris, AIX, HP-UX dan Mac OS X.
2. *Multi-language*, eclipse dikembangkan dengan bahasa pemrograman Java, akan tetapi Eclipse mendukung pengembangan aplikasi berbasis bahasa pemrograman lainnya, seperti C/C++, Cobol, Python, Perl, PHP, dan lain sebagainya.
3. *Multi-role*, selain sebagai IDE untuk pengembangan aplikasi, Eclipse pun bisa digunakan untuk aktivitas dalam siklus pengembangan perangkat lunak, seperti dokumentasi, tes perangkat lunak, pengembangan *web*, dan lain sebagainya.

Eclipse pada saat ini merupakan salah satu IDE favorit dikarenakan gratis dan *open source*, yang berarti setiap orang boleh melihat kode pemrograman perangkat lunak ini. Selain itu, kelebihan dari Eclipse yang membuatnya populer adalah kemampuannya untuk dapat dikembangkan oleh pengguna dengan komponen yang dinamakan *plug-in*[6].

## 2.4 Object Oriented Programming (OOP)

Dalam penyusunan tugas akhir ini metodologi yang penulis gunakan adalah metodologi yang berbasis OOP. Dengan demikian penjelasan mengenai *system* ini dapat digambarkan dengan *use case diagram, activity diagram* dan cla*ss diagram*[7].

OOP adalah suatu metode dalam pembuatan program dengan tujuan untuk menyelesaikan komplek berbagai masalah program yang terus meningkat. OOP tidak seperti pendahulunya (pemrograman terstruktur), OOP mencoba melihat

permasalahan melalui pengamatan dunia nyata dimana setiap *object* adalah entitas tunggal yang memiliki kombinasi struktur data dan fungsi tertentu. Ini kontras dengan pemrograman terstruktur dimana struktur data dan fungsi didefinisikan secara terpisah dan tidak berhubungan secara erat[7].

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat. OOP terdiri dari *object* yang berinteraksi satu sama lainnya untuk menyelesaikan sebuah tugas. Contoh anggap kita memiliki sebuah perusahaan yang memiliki *manager*, sekretaris, petugas administrasi, dan lain-lain. Misal *manager* ingin memperoleh data dari bagian administrasi maka *manager* tidak harus mengambil berkas tersebut langsung, tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus ini seorang *manager* tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi *manager* bisa mendapatkankan data tersebut melalui *object* petugas administrasi[7].

Jantung pemrograman berbasis obyek adalah konsep *object*, sebuah konstruksi pemograman yang merupakan kombinasi spesifik dan seperangkat kelakuan yang saling berhubungan. Dalam OOP, skenario dimulai dari obyek yang bersangkutan dibandingkan dengan langkah demi langkah yang dilakukan oleh bahasa pemrograman terstuktur[7].

Konsep dasar OOP dalam pemrograman:

1. *Class*, *Object* dan *Instance*

   Istilah yang banyak digunakan dalam pemrograman berbasis obyek adalah *class, object, instance*. Sebuah *class* merupakan *template* bagi *object*. Analogi yang menggambarkan hubungan antara *class*, *object*, *instance* adalah ketika kita memikirkan istilah rumah dan bangunan rumah. Pada analogi tersebut, *class* adalah cetak biru dari rumah dan rumah itu sendiri adalah *object*. Banyak rumah yang dapat dibangun dengan cetak biru yang sama dan banyak *object* yang dapat dibuat dari *class* yang sama, dan setiap *object* yang dibuat dari *class* disebut *instance* dari *class*.

*Object* dibentuk oleh anggota-anggotanya dan anggota dari *object* dapat berupa *properties*, *field*, *method* dan *events* dan disajikan dalam bentuk fungsionalitas yang membentuk *object*. Sedangkan *method* merupakan aksi yang dilakukan oleh sebuah *object* dan *event* merupakan notifikasi sebuah *object* untuk melakukan tindakan tertentu. *Class* dapat mendefenisikan properti yang dapat dimiliki oleh sebuah *instance*. Properti merupakan sebuah nilai yang ada sebagai bagian dari *object* dan kita dapat memanggilnya atau mengaturnya melalui sebuah *object*. Atribut merupakan status *object* dan *behavior* merupakan tingkah laku dari *object* tersebut. *Class* merupakan *prototipe*, *blue p*rint, ataupun *template* dengan kata lain kelas adalah representasi abstrak dari suatu *object*, sedangkan *object* adalah representasi nyata dari kelas ataupun perwujudan (*instance*) dari suatu kelas[7].

2. *Inheritance*

   Mekasnisme *inheritance* mengijinkan kita untuk memanfaatkan fungsionalitas yang terdapat di kelas yang telah didefenisikan kedalam kelas baru dan mengimplementasikan anggota yang berbeda jika diperlukan. Sebuah *class* dapat mengindukkan langsung kesatu kelas yang disebut *base class*. Penggunaan *inheritance* dalam OOP untuk menglasifikasikan *object* dalam program sesuai karakteristik umum dan fungsinya. Hal ini juga membuat *programming* lebih mudah karena memungkinkan kita untuk mengombinasikan karakteristik umum kedalam *object parent* dan mewariskan ini ke *child object*[7].

3. *Override*

   Diperlukan untuk memodifikasi atau menambah metode terhadap metode yang diwariskan oleh kelas induk. Hal tersebut dapat dilakukan, dan mekanisme tersebut dikenal dengan istilah *override*[7].

4. *Constructor*

   *Contsructor* merupakan sebuah *routine* (prosedur) yang dipanggil pada saat suatu kelas *instance* dibuat. Di java, *constructor* disajikan dalam bentuk prosedur “NEW” ketika mendefenisikan *class*. *Constructor* dapat *dioverride*. *Constructor* berguna untuk menginisialisasi sebuah *object*[7].

5. *Overload*

   *Overload* digunakan untuk membuat beberapa metode dengan nama yang sama tetapi memiliki implementasi yang berbeda. Metode-metode yang *overload* harus memiliki ciri khas yang membedakan diantaranya metode - metode tersebut tetapi metode-metode tersebut dapat memiliki *return type* dan akses level yang sama[7].

6. *Encapsulation*

   *Encapsulation* merupakan proses penyembunyian atau pembungkusan tentang detail dari bagaimana sebuah *object* melakukan kegiatannya ketika diminta untuk melakukan tugas-tugas sebuah *client*. Keuntungan dari mekanisme ini adalah:

   a. *Client* dapat menggunakan fungsi dari sebuah *object* tanpa harus mengetahui bagaimana *object* tersebut bekerja.
   b. Setiap perubahan yang terjadi didalam sebuah *object* tidak diketahui oleh *client*.

7. *Polimorfisme*

   *Polimorfisme* sebuah kata dari bahasa Yunani yang mempunyai arti banyak bentuk. Konsep ini dimungkinkan untuk menggunakan suatu *interface* yang sama agar suatu *object* melakukan aksi atau tindakan yang mungkin secara prinsip sama tapi secara proses bisa berbeda-beda. Pada umumnya konsep ini sering kali disebut dalam istilah satu *interface* banyak aksi[7].

## 2.5 SQLite

Sqlite adalah fasilitas yang digunakan untuk membuat *database* yang disediakan oleh android yang secara *default* sudah tersedia di dalam *library* android. Untuk keperluan operasi *database* pada *smartphone* atau tablet android, SQLite sangat memadai karena ukurannya yang kecil, cepat dan ringan dalam hal sumber daya. SQLite merupakan proyek yang bersifat *public domain* yang dikerjakan oleh D. Richard Hipp[8].

Dalam SQLite maupun *database* lainnya terdapat suatu bahasa yang digunakan dalam melakukan modifikasi *database* tersebut, bahasa tersebut yaitu *structured query language* (SQL). SQL merupakan bahasa standar yang digunakan untuk memanipulasi dan memperoleh data dari sebuah *database*[8]. Dalam SQL terdapat beberapa hal yang dapat dilakukan oleh seorang *programmer* dan *database administrator*, yaitu:

1. Mengubah struktur *database*
2. Mengubah pengaturan pengamanan sistem
3. Memberikan hak akses kepada *user* untuk mengakses suatu *database*
4. Memperoleh isi dan informasi dari suatu *database*
5. Mengubah *database*

Sintak SQL terbagi menjadi 2 kategori yaitu:

1. *Data Definition Language* (DDL)

   *Data Definition Language* (DDL) digunakan untuk mendefinisikan, mengubah, serta menghapus basis data dan *object* yang diperlukan dalam basis data, misalnya *table*, *view*, atau *user*. Secara umum DDL yang digunakan adalah "CREATE" untuk membuat *object* baru, "USE" untuk menggunakan *object*, "ALTER" untuk mengubah *object* yang sudah ada, dan "DROP" untuk menghapus *object*. DDL biasanya digunakan oleh *administrator* basis data dalam pembuatan sebuah aplikasi basis data[8].

2. *Data Manipulation Language* (DML)

   *Data Manipulation Language* (DML) merupakan kumpulan perintah SQL yang digunakan untuk proses pengolahan isi data di dalam tabel seperti memasukkan, merubah dan menghapus isi data, dan tidak terkait dengan perubahan struktur dan definisi tipe data dari *object database*. Perintah-perintah DML yang umum digunakan adalah "SELECT" untuk menampilkan data, "INSERT" untuk menambahkan data baru, "UPDATE" untuk mengubah data yang sudah ada, "DELETE" untuk menghapus data[8].

SQLite merupakan *database* yang *powerful*. Beberapa fitur yang dapat digunakan dalam SQLite yaitu:

1. Memiliki transaksi yang bersifat atomic, konsistensi basis data, isolasi, dan durabilitas (dalam Bahasa Inggris lebih sering disebut ACID).
2. Tanpa konfigurasi, tanpa instalasi dan administrasi.
3. Mengimplementasikan hampir seluruh elemen-elemen standar yang berlaku pada SQL-92.
4. Mendukung *database* hingga berukuran *terabyte database* dan berukuran *gigabyte string*.
5. SQLite memiliki jejak kode kecil, membuat efisiensi penggunaan memori, ruang *disk*.
6. Mudah untuk penggunaan API.
7. Mendukung banyak sistem operasi : Unix (Linux dan Mac OS X), OS / 2, dan Windows (Win32 dan WinCE)
8. SQLite pilihan populer untuk mesin *database* di ponsel, PDA, MP3 *player*, set-top box, dan *gadget* elektronik lainnya.

## 2.6 Unified Modeling Language (UML)

UML merupakan bahasa pemodelan yang paling sukses dari tiga metode OO yang telah ada sebelumnya, yaitu Booch, OMT dan OOSE. UML merupakan kesatuan dari ketiga metode pemodelan tersebut dan ditambah kemampuan lebih untuk mengatasi masalah pemodelan yang tidak bisa ditangani ketiga metode tersebut. UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks. UML tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan[9].

1. Use Case Diagram

   *Use case* adalah abstraksi dari interaksi antara *system* dan *actor*. *Use c*ase bekerja dengan cara mendeskripsikan tipe interaksi antara *user* sebuah *system* dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah *system* dipakai. *Use case* merupakan konstruksi untuk mendeskripsikan bagaimana *system* akan terlihat di mata *user*. Sedangkan *use case diagram*

memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*[9].

2. Class Diagram

   *Class diagram* adalah dekripsi kelompok *object* dengan *property*, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class diagram* dapat memberikan pandangan global atas sebuah *system*. Hal tersebut tercermin dari *class* yang ada dan relasinya satu dengan yang lainnya. Sebuah sistem biasanya mempunyai beberapa *class d*iagram. *Class diagr*am sangat membantu dalam visualisasi struktur kelas dari suatu *system*[9].

3. Activity Diagram

   *Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana alir itu berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi[9].

Tujuan Penggunaan UML yaitu:

1. Memberikan bahasa pemodelan yang bebas dari berbagai bahasa pemrograman dan proses rekayasa
2. Menyatukan praktik-praktik terbaik yang terdapat dalam pemodelan
3. Memberikan model yang siap pakai, bahasa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum
4. UML bisa juga berfungsi sebagai sebuah cetak biru (*blue print*) karena sangat lengkap dan detail. Dengan cetak biru ini maka akan bisa diketahui informasi secara detail tentang *coding* program atau bahkan membaca program dan menginterpretasikan kembali ke dalam bentuk diagram (*reserve enginering*).

## 2.7 GPS

GPS merupakan sistem yang digunakan untuk menentukan posisi di permukaan bumi dengan sinkronisasi sinyal satelit. Dengan bantuan GPS

seseorang dapat mengetahui posisi *object* yang diinginkannya dengan bantuan perangkat yang memiliki sensor GPS di dalamnya. Sistem ini dikembangkan oleh Departemen Pertahanan Amerika Serikat. Sistem yang serupa dengan GPS antara lain GLONASS dari Rusia, Galileo dari Uni Eropa dan IRNSS dari India[1].

GPS bekerja ketika sejumlah satelit yang berada di orbit bumi memancarkan sinyalnya ke bumi, kemudian sinyal tersebut ditangkap oleh sebuah alat penerima yang nantinya diubah menjadi informasi berupa titik lokasi dari alat penerima tersebut. Karena alat ini bergantung penuh pada satelit, maka sinyal satelit merupakan hal yang penting untuk mendapatkan informasi posisi *object* yang berupa titik koordinat. Untuk itu perlu diperhatikan hal yang dapat mengganggu sinyal satelit antara lain adalah kondisi geografis, alat yang mengeluarkan gelombang elektromagnetik, gedung, dan sinyal yang memantul[1].

GPS *Tracker* atau sering disebut dengan GPS *Tracking* adalah teknologi *Automated Vehicle Locater* (AVL) yang memungkinkan pengguna untuk melacak posisi kendaraan, armada ataupun mobil dalam keadaan *Real-Time*. GPS *Tracking* memanfaatkan kombinasi teknologi GSM dan GPS untuk menentukan koordinat sebuah *object*, lalu menerjemahkannya dalam bentuk peta digital[1].

### 2.7.1 Arsitektur GPS

Arsitektur GPS terdiri atas 3 segmen, yaitu *space segment, control segment, dan user segment*. *Space segment* berupa 27 satelit yang terus mengorbit bumi. 24 satelit digunakan untuk operasional, sedangkan 3 satelit sisanya digunakan sebagai cadangan. 24 satelit tersebut dibagi atas kumpulan 4 satelit yang mengorbit pada 6 jalur lintasan. Orbit lintasan ini telah diatur, sehingga setiap titik di bumi ini pasti tercakup dalam *Line of Sight* (LOS) dari setidaknya 6 satelit[1].

*Control segment* berupa stasiun di bumi yang memonitor satelit-satelit GPS. Stasiun ini mengontak setiap satelit GPS secara berkala untuk *update* navigasi. *Update* ini berupa sinkronisasi jam atomik satelit dengan satelit lainnya dan mengkoreksi lintasan orbit setiap satelit[1].

*User segment* adalah berupa GPS *receiver*. Secara umum, GPS *receiver* terdiri dari sebuah antena yang dapat menangkap frekuensi yang ditransmisikan

satelit GPS, sebuah *processor*, dan sebuah jam yang sangat stabil. GPS *receiver* memiliki atribut *channel* yaitu jumlah satelit yang dapat dimonitornya dalam suatu waktu (sekarang umumnya jumlah *channel* berkisar antara 12-20). Kebanyakan GPS *receiver* dapat meneruskan datanya ke perangkat yang lain melalui koneksi serial, USB, atau Bluetooth menggunakan protokol *National Marine Electronics Association* (NMEA) [1].

### 2.7.2 Cara Penentuan Lokasi Pada GPS

Satelit GPS mengorbit bumi dua kali dalam sehari dengan lintasan yang sangat presisi dan menransmisikan sinyal secara berkelanjutan ke bumi. GPS *receiver* memanfaatkan informasi ini dengan berperan sebagai sebuah alat pengukur yang menghitung jarak antara antena *receiver* dengan berbagai satelit GPS. Kemudian GPS *receiver* mendeduksi posisi melalui posisi trilaterasi dengan mencari perpotongan tiap vektor satelit-satelit tersebut. Jarak antara antena *receiver* dan satelit diukur dengan membandingkan waktu yang terdapat pada sinyal dengan waktu ketika sinyal diterima[1].

Sebuah GPS *receiver* setidaknya harus dapat menangkap sinyal dari 3 buah satelit untuk menghitung posisi 2 dimensi (latitude dan longitude) dan pergerakannya. Dengan 4 buah satelit atau lebih, *receiver* dapat menghitung posisi 3 dimensi (latitude, longitude, dan altitude) [1].

### 2.7.3 Akurasi GPS

GPS menyediakan posisi dengan ketepatan akurasi hingga 15 meter, yang berarti jika GPS *receiver* memberikan koordinat terhadap suatu lokasi tertentu, maka boleh diharapkan lokasi sebenarnya berada dalam radius 15 meter dari koordinat tersebut. Ketepatan GPS bergantung daripada lokasi GPS *receiver*-nya dan halangan terhadap sinyal satelit GPS[1].

### 2.7.4 Sumber Kesalahan Pada GPS

Berbagai faktor yang dapat mengurangi akurasi GPS yaitu:

1. *Ionosphere* dan *troposphere error*, dikarenakan sinyal mengalami hambatan ketika melewati atmosfer bumi sehingga menyebabkan delay.

2. *Multipath signal*, dikarenakan sinyal GPS dapat direfleksikan oleh berbagai benda sebelum mencapai *receiver* sehingga waktu yang dibutuhkan menjadi semakin lama.
3. Jam GPS *receiver error*, dikarenakan jam internal pada GPS *receiver* tidak seakurat jam atomik pada satelit GPS.
4. *Orbital* atau *ephemeris error*, dikarenakan adanya kesalahan mengenai lokasi satelit.
5. Jumlah satelit yang tertangkap, semakin sedikit satelit yang tertangkap oleh GPS *receiver* semakin rendah akurasinya.
6. *Satellite Geometry error*, dikarenakan posisi satelit tidak ideal dalam perhitungan geometri ketika satelit terlalu berdekatan.

## 2.8 Google Maps

Google Maps adalah sebuah layanan gratis peta digital dari Google berbasis *web* yang dapat digunakan dan ditempatkan pada *website* tertentu dengan menggunakan Google Maps API. Google Maps API adalah suatu *library* yang berbentuk JavaScript. Seiring perkembangan yang pesat, saat ini Google Maps API sudah bisa disematkan pada ponsel anrdoid[6].

Google Maps sendiri mempunyai antara lain navigasi peta dengan *dragging mouse*, *zoom in*, dan *zoom out* untuk menunjukkan informasi peta secara detil, memberi penanda, dan memberi informasi tambahan. Pada Google Maps API terdapat 4 jenis pilihan model peta yang disediakan oleh Google, diantaranya adalah:

1. *ROADMAP*, ini yang penulis pilih, untuk menampilkan peta biasa 2 dimensi
2. *SATELLITE*, untuk menampilkan foto satelit
3. *TERRAIN*, untuk menunjukkan relief fisik permukaan bumi dan menunjukkan seberapa tingginya suatu lokasi, contohnya akan menunjukkan gunung dan sungai
4. *HYBRID*, akan menunjukkan foto satelit yang diatasnya tergambar pula apa yang tampil pada *ROADMAP* (jalan dan nama kota)

## 2.9 Algoritma A*

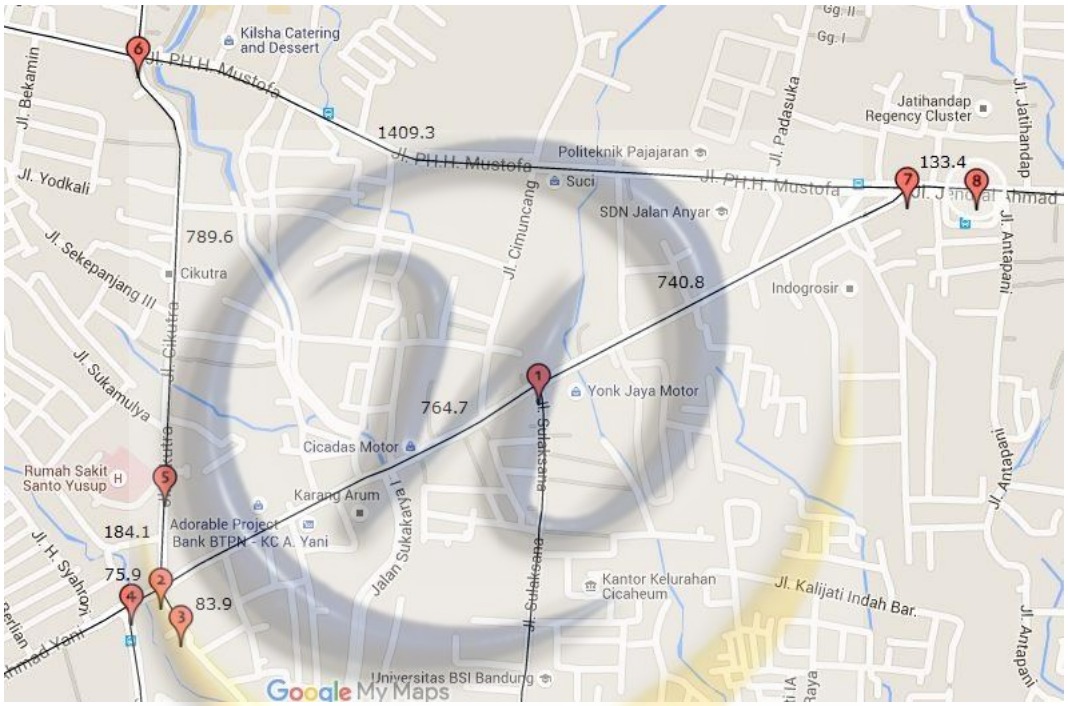
Alogritma A* merupakan salah satu jenis algoritma yang digunakan untuk menyelesaikan kasus yang berhubungan dengan *path finding* (pencarian jalan). Dalam hasil pencariannya Algoritma A* dikatakan *complete* dan *optimal*. Algoritma A* menggunakan cara pencarian yang dilakukan dengan cara melebar ke setiap *node* pada level yang sama, dan nantinya akan menemukan rute terbaik dari titik awal sampai titik tujuan[10].

Algoritma A* memiliki lima komponen utama, yaitu : *node* awal, *node goal*, *open list*, *closed list*, dan *cost*. *Node* awal merupakan titik awal dari posisi saat ini, sedangkan *node goal* merupakan titik akhir atau dapat juga disebut titik tempat tujuan. *Cost* merupakan nilai dari jarak yang telah ditempuh untuk sampai ke tempat tujuan. *Open list* digunakan sebagai penampung alternatif *node* yang tersedia untuk dipilih sebagai *node* selanjutnya. *Closed list* digunakan sebagai penampung *node* yang telah dilewati selama proses berlangsung. *Closed* list ini berupa sebuah *stack*, dimana *node* yang terakhir dimasukkan akan dikeluarkan pertama kali. Selain sebagai penampung *node* yang telah dilewati, *closed list* ini juga digunakan untuk mendapatkan rute terdekat saat *node goal* sudah dicapai[10].

Langkah pertama yang dilakukan adalah mencari tahu lokasi pada peta yang akan dijadikan sebuah *node*. Setelah mendapatkan semua *node*, hubungkan *node-node* tersebut sesuai dengan jalan pada peta. Setelah itu dicari *cost/value* diantara node tersebut dengan bantuan google map. *Cost/value* pada kasus ini merupakan jarak dari satu *node* ke *node* lain. Semua data tersebut disimpan pada *array string* bernama mGraph. Pada mGraph terdapat semua *node*, koordinat *node* serta *cost* yang ada pada peta. Jadi, mGraph merupakan bentuk peta yang telah diubah menjadi *node-node* beserta atributnya sehingga dapat digunakan sebagai bahan utama dalam melakukan pencarian lokasi terdekat.

Algoritma A* dimulai dengan mempersiapkan *open list* dengan *node* awal sebagai isi dan mengosongkan *closed list*. Langkah berikutnya adalah mengambil isi dari *open list*. Jika *open list* kosong, proses akan berakhir dengan hasil rute

tidak ditemukan. *Node* yang diambil dari *open list* dimasukkan ke dalam *closed list*. Jika *node* tersebut merupakan *node goal*, proses akan berakhir dengan hasil rute ditemukan. Jika *node* tersebut bukan merupakan *node goal*, *open list* akan diisi dengan *node* baru yang merupakan *successor node* tersebut. *Successor node* adalah *node* yang dapat dituju dari *node* saat ini. *Successor node* yang akan dimasukkan ke dalam *open list* hanya berupa *node* yang tidak terdapat dalam *closed list*. Proses ini berlangsung sampai mencapai *node goal*[10].




Gambar 2.1 Contoh kasus

Contoh kasus pada gambar 2.1 menunjukan data yang tersimpan pada mGraph. Misalnya posisi user berada pada *node* 1 dan mencari rumah sakit terdekat, yaitu ada pada *node* 5. Dari *node* 1 menuju *node* 7 dengan *cost* 740.8. Setelah itu menuju *node* 8 dengan *cost* 133.4. Karena *node* 8 bukan merupakan *node* tujuan dan tidak memiliki *node* selanjutnya (*node* buntu), maka *node* 8 dihapuskan dalam pencarian. Dimulai kembali dari *node* 1 ke *node* 2 dengan *cost* 764.7. Pada *node* 2 terdapat 3 kemungkinan, yaitu menuju *node* 3 dengan *cost* 83.9, menuju *node* 4 dengan *cost* 75.9 dan menuju *node* 5 dengan *cost* 184.1. *Cost* terkecil adalah menuju *node* 3. Tetapi *node* tiga bukanlah *node* tujuan dan tidak memiliki *node* selanjutnya (*node* buntu), begitu pula dengan *node* 4. Maka *node* 3

dan 4 dihapuskan dalam pencarian. Kemungkinan selanjutnya adalah menuju *node* 5. *Node* 5 merupakan *node* tujuan, maka pencarian dihentikan dengan rute 1-2-5.

## 2.10 Unit Testing

Jika struktur kendali antar modul sudah terbukti bagus, maka pengujian yang tak kalah pentingnya adalah pengujian unit. Pengujian unit digunakan untuk menguji setiap modul untuk menjamin setiap modul menjalankan fungsinya dengan baik. Ada 2 metode untuk melakukan unit testing, yaitu *black box testing* dan *white box testing*[11].

### 2.10.1 Black Box Testing

Terfokus pada apakah unit program memenuhi kebutuhan (requirement) yang disebutkan dalam spesifikasi. Pada black box testing, cara pengujian hanya dilakukan dengan menjalankan atau mengeksekusi unit atau modul, kemudian diamati apakah hasil dari unit itu sesuai dengan proses bisnis yang diinginkan. Jika ada unit yang tidak sesuai outputnya maka untuk menyelesaikannya, diteruskan pada pengujian yang kedua, yaitu *white box testing*[11].

### 2.10.2 White Box Testing

*White box testing* adalah cara pengujian dengan melihat ke dalam modul untuk meneliti kode-kode program yang ada, dan menganalisis apakah ada kesalahan atau tidak. Jika ada modul yang menghasilkan *output* yang tidak sesuai dengan proses bisnis yang dilakukan, maka baris-baris program, variabel, dan parameter yang terlibat pada unit tersebut dicek satu persatu dan diperbaiki, kemudian di-*compile* ulang[11].