BAB 2
LANDASAN TEORI

2.1 Bina Marga Kota Garut

Menurut Peraturan Pemerintah Nomor 107 Tahun 2017 tentang tugas fungsi dan tata kerja Dinas Pekerjaan Umum dan Penataan Ruang (PUPR), Bina Marga merupakan suatu bidang yang dipimpion oleh seorang Kepala Bidang yang berada di bawah dan bertanggung jawab kepada Kepala Dinas PUPR memiliki tugas menyelenggarakan perumusan kebijakan teknis dan menyelenggarakan penyusunan rencana kerja Bidang Bina Marga, meliputi pembangunan jalan, pemeliharaan jalan, dan pemanfaatan jalan. Kantor Dinas Bina Marga Kota Garut bertempatan di Jalan Raya Samarang No.117.

Gambar 2.1 Logo Bina Marga

2.1.1 Tugas Pokok dan Fungsi Bina Marga

Menurut Peraturan Pemertintah No 107 tahun 2017 Bidang Bina Marga mempunyai fungsi dan tugas sebagai berikut :

1. Fungsi

1) Penyelenggaraan perumusan, pelaksanaan dan evaluasi kebijakan teknis operasional bidang Bina Marga, meliputi pembangunan jalan, pemeliharaan jalan, dan pemanfaatan jalan
2) Penyelenggaraan rencana kerja bidang Bina Marga, meliputi pembangunan jalan, pemeliharaan jalan, dan pemanfaatan jalan
3) Penyelenggaraan koordinasi, integrasi dan sinkronisasi sesuai dengan lingkup tugasnya
4) Penyelenggaraan monitoring, evaluasi dan pelaporan capaian kinerja bidang Bina Marga.

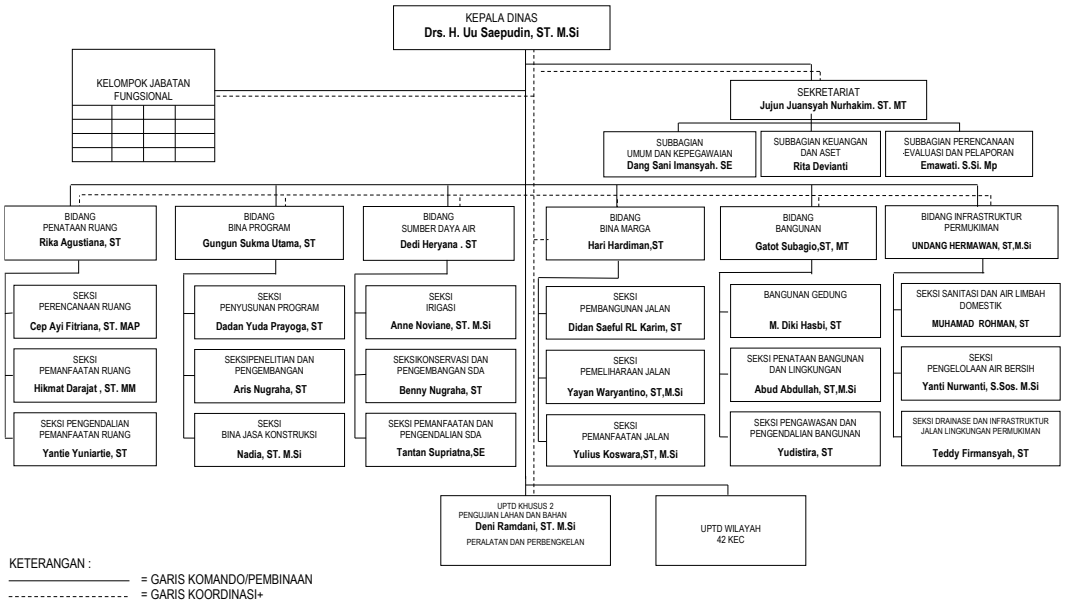
2. Uraian Tugas

1) Menyelenggarakan perumusan, pelaksanaan dan evaluasi kebijakan teknis bidang Bina Marga
2) Menyelenggarakan penyusunan rencana kerja bidang Bina Marga berdasarkan sasaran, kebijakan teknis, strategi dan program kerja dinas serta kondisi dinamis lingkungan dan masyarakat
3) Menyelenggarakan penyusunan rencana teknis pembangunan, peningkatan jalan dan jembatan
4) Menyelenggarakan pembangunan, peningkatan dan rehabilitasi jalan dan jembatan serta bangunan pelengkap berdasarkan prioritas yang telah ditetapkan
5) Menyelenggarakan pengadaan lahan untuk kepentingan pembangunan jalan dan jembatan
6) Menyelenggarakan pengawasan dan pengendalian pelaksanaan pembangunan dan peningkatan jalan serta pelengkapnya
7) Menyelenggarakan pembinaan jalan desa serta pelengkapnya berdasarkan prioritas yang telah ditetapkan
8) Menyelenggarakan fasilitasi teknis dalam pelaksanaan pembangunan jalan desa oleh masyarakat, swasta, pemerintah dan instansi lainnya
9) Menyelenggarakan inventarisasi data kondisi jalan dan jembatan

10) Menyelenggarakan pemeliharaan rutin jalan dan jembatan serta pelengkapnya berdasarkan prioritas yang telah ditetapkan
11) Menyelenggarakan penanganan darurat kerusakan jalan dan jembatan Kelurahan/Desa akibat bencana dan/atau karena sebab lainnya
12) Menyelenggarakan koordinasi untuk relokasi jalan dan jembatan Kelurahan/Desa yang diakibatkan bencana dan/atau karena sebab lainnya
13) Menyelenggarakan pengawasan serta pengendalian pelaksanaan pemeliharaan rutin jalan dan jembatan Kabupaten/Kota dan pelengkapnya serta penanganan darurat kerusakan jalan Kabupaten/Kota akibat bencana dan/atau karena sebab lainnya
14) Menyelenggarakan pengadaan, penyimpanan dan distribusi bahan/material pemeliharaan rutin jalan dan jembatan
15) Menyelenggarakan pemeliharaan rutin/pengelolaan serta penyelenggaraan sistem manajemen jalan dan jembatan
16) Menyelenggarakan penyusunan bahan pengaturan teknis pemanfaatan ruang milik jalan, ruang manfaat jalan dan ruang pengawasan jalan terhadap aktivitas masyarakat di luar kepentingan lalu lintas dan melaksanakan penyusunan kajian teknis untuk bahan pertimbangan perijinan
17) Menyelenggarakan pengaturan teknis, pengawasan dan pengendalian terhadap pemanfaatan ruang milik jalan dan ruang manfaat jalan berdasarkan standar yang telah ditetapkan
18) Menyelenggarakan fasilitasi kajian dan/atau saran teknis proses pelayanan perizinan penerbitan rekomendasi pemerintah daerah untuk perizinan pemanfaatan ruang milik jalan, ruang manfaat jalan dan ruang pengawasan jalan
19) Menyelenggarakan penyusunan bahan pengaturan teknis ruang milik jalan, ruang manfaat jalan dan ruang pengawasan jalan
20) Menyelenggarakan monitoring, evaluasi dan pelaporan bidang Bina Marga

21) Menyelenggarakan koordinasi dengan unit kerja lain dan/atau lembaga/organisasi terkait dalam lingkup tugasnya
22) Mendistribusikan tugas kepada staf sesuai dengan bidang tugasnya
23) Mengkoordinasikan seluruh kegiatan seksi dalam melaksanakan tugas
24) Memberi petunjuk kepada staf untuk kelancaran pelaksanaan tugasnya
25) Menyelenggarakan kegiatan staf dalam lingkup Bidang Bina Marga untuk mengetahui kesesuaiannya dengan rencana kerja
26) Mengarahkan dan mengendalikan pelaksanaan tugas staf berdasarkan rencana kerja yang telah ditetapkan
27) Menyusun dan memeriksa konsep surat dinas berdasarkan tata naskah dinas yang berlaku
28) Mengevaluasi pelaksanaan tugas staf melalui penilaian Sasaran Kerja Pegawai (SKP) untuk mengetahui prestasi kerjanya dan sebagai bahan pembinaan serta upaya tindak lanjut;
29) Melaporkan pelaksanaan tugas dalam lingkup Bina Marga secara lisan, tertulis, berkala atau sesuai dengan kebutuhan kepada pimpinan;
30) Memberikan saran dan pertimbangan kepada pimpinan sesuai dengan bidang tugasnya
31) Melaksanakan tugas-tugas kedinasan lain yang diberikan oleh pimpinan sesuai dengan bidang tugasnya.

2.1.2 Struktur Organisasi Bina Marga Kota Garut



Gambar 2.2 Struktur Organisasi Bina Marga Kota Garut

2.1.3 Unit Pelaksana Teknis Daerah (UPTD)

Unit Pelaksan Teknis Daerah merupakan unit pelaksana dari Penataan Ruang yang memiliki Wilayah Kerja di lingkup Kecamatan dipimpin oleh seorang Kepala Unit Pelaksana Teknis (UPT) yang mempunyai tugas memimpin mengkoordinasikan dan mengendalikan pekerjaan umum dan penataan ruang dilingkup Kecamatan. Adapun salah satu fungsi dari UPT ini adalah melaksanakan monitoring, evaluasi dan pelaporan serta capaian kinerja sesuai dengan tugas dan fungsi.

2.2 Landasan Teori

2.2.1 Analisa Urutan Prioritas

Berdasarkan tata cara penyusunan program pemeliharaan jalan kota No.018/T/BNKT/1990, Urutan Prioritas dihitung dengan memakai rumus sebagai berikut :

Urutan Prioritas = ***17 - (Kelas LHR + Nilai - Kondisi Jalan)***

Kelas LHR = Kelas lalu-lintas untuk pekerjaan pemeliharaan

Nilai Kondisi Jalan = Nilai yang diberikan terhadap kondisi jalan

2.2.2 Aplikasi *Mobile*

Mobile Applications atau Aplikasi *Mobile* adalah aplikasi perangkat lunak yang dibuat khusus untuk dijalankan di dalam tablet dan juga *smartphone*. Umumnya, *developer mobile apps* memerlukan IDE atau *Intergrated Development Environments* dan juga SDK untuk pengembangan dari *mobile apps* itu sendiri. Pada saat ini, pada smartphone dan juga tablet, ada satu aplikasi yang berguna untuk menyediakan berbagai macam aplikasi yang dapat dijalankan di device tersebut. Aplikasi ini sering disebut *store*. Contoh *store* yaitu *apple apps store, samsung apps, amazon kindlefire, windows store dan google playstore* [6].

Gambar 2.3 *Store Mobile*

Jika membahas tentang *Mobile Apps*, umumnya kita tidak bisa tidak menyinggung soal Developer. Developer ialan badan usaha yang membuat aplikasi yang nantinya akan dijalankan dalam device. Pada dasarnya, aplikasi akan berjalan menggunakan tenaga baterai dan juga mendapat dukungan dari prosesor yang ukurannya lebih kecil dibanding dengan prosesor komputer. Sebelum dilempar ke pasar, umumnya mobile *apps* akan dites terlebih dahulu menggunakan emulator. Emulator merupakan salah satu cara untuk menghemat biaya yang dikeluarkan oleh developer untuk membuat *mobile apps*(Irwansyah & V.Moniaga:61-62) [6].

2.2.3 REST Web *Service*

Transfer representasional State (REST) style adalah gaya arsitektur yang pertama kali didefinisikan oleh Roy T. Fielding dalam tesis doktoralnya. Fielding berpartisipasi dalam mengembangkan HTTP1.0. Dia juga adalah arsitek utama HTTP 1.1, dan mengarang sintaksis generik pengenal sumber daya (URI). Dia melihat REST sebagai cara untuk membantu mengomunikasikan konsep dasar yang mendasari Web. Untuk memahami REST, perlu memahami definisi sumber daya,

representasi dan negara. Sumber daya bisa apa saja. Sumber daya dapat berupa objek fisik atau konsep abstrak. Selama sesuatu itu cukup penting untuk dirujuk sebagai sesuatu itu sendiri, itu dapat diekspos sebagai sumber daya. Biasanya sumber daya adalah sesuatu yang dapat disimpan di komputer dan direpresentasikan sebagai aliran bit. Representasi adalah informasi yang berguna tentang keadaan sumber daya. Sumber daya mungkin memiliki beberapa representasi yang berbeda. Di REST ada dua jenis negara. Salah satunya adalah status sumber daya yang merupakan informasi tentang sumber daya, dan yang lainnya adalah status aplikasi, yaitu informasi tentang jalur yang telah diambil klien melalui aplikasi. Status sumber daya tetap di server dan status aplikasi hanya tinggal di klien. REST menyediakan seperangkat kendala arsitektur yang, ketika diterapkan secara keseluruhan, menekankan skalabilitas interaksi komponen, generalisasi antarmuka, penyebaran komponen independen, dan komponen perantara untuk mengurangi latensi interaksi, menegakkan keamanan, dan merangkum sistem warisan [7].

(*Sumber : REST : An Alternative to RPC for Web Services Architecture[Feng, Xinyang; Shen, Jianjing; Fan, Ying,2009*])

Gambar 2.4 Arsitektur REST Web *Service* [7]

2.2.4 Teknologi Geotagging

Geotagging adalah proses menambahkan metadata dengan identifikasi geografis ke konten. Hal ini juga didefinisikan sebagai bentuk metadata geospasial. Proses geotagging berasal dari *Global Positioning System* (GPS), yang didasarkan pada model koordinat lintang dan bujur. Dengan demikian, posisi yang diasumsikan oleh layanan informasi yang diaktifkan *geotag* berasal dari pola ini [8]

Geotagging memungkinkan pengindeksan spasial konten. Oleh karena itu, hal ini adalah proses lokasi geografis yang diakui dapat meningkatkan

pengembangan basis data geografis, sumber daya web yang dirujuk secara geografis, dan konten multimedia yang direferensikan secara geografis. Akibatnya, geotagging adalah praktik yang sepenuhnya mengubah cara pengguna berinteraksi dengan konten dan pengguna lain di ruang digital. Sistem dengan fitur geotagging biasanya menambah lintang, bujur, dan nama tempat konten media. Data yang ditambahkan terdiri dari fitur tekstual dan visual.

Standar penandaan geografis dalam format file elektronik tertanam dalam informasi dalam metadata. Namun, untuk setiap jenis media, ada protokol yang berbeda untuk melampirkan informasi *geospasial* karena informasi *geotag* dapat dibaca oleh beberapa sistem media. Opsi utama sistem penandaan geografis adalah penangkapan informasi GPS pada saat publikasi atau posisi lokal yang ditambahkan oleh pengguna. Layanan berbasis lokasi memiliki pendekatan berbeda terhadap privasi, tetapi sebagian besar pengaturan default yang ada.

2.2.5 Teknologi GPS

Global *Positioning System* atau sering disingkat dengan GPS adalah sistem navigasi yang menggunakan satelit yang didesain agar dapat menyediakan posisi secara instan, kecepatan dan informasi waktu di hampir semua tempat dimuka bumi, setiap saat dan dalam kondisi cuaca apapun. Pada dasaranya, GPS merupakan aplikasi yang harus menunggu terlebih dahulu permintaan dari pengguna. Aplikasi ini menyediakan akurasi positioning atau penentuan posisi yang berkisar antara 100 meter(95% dari waktu), hingga 5 sampai 10 meter, juga sampai akurasi relatif pada submeter, dan bahkan tingkat subcentimeter. Secara umum, semakin tinggi akurasi yang dihasilkan akan memerlukan infrastruktur yang lebih canggih dan tentunya berhubungan dengan biaya yang harus dikeluarkan [9].

Pengguna GPS untuk penentuan posisi saat ini diantaranya adalah navigasi untuk kegiatan pribadi (hiking, pelayaramn, berburu, petunjuk ketika mengemudi dan lain sebagainya), navigasi di pesawat, survei di lepas pantai dan navigasi kapal, fleet tracking, pengendalian mesin, teknik sipil, survey daratan, GIS dan pemetaan, analisis deformasi dan sebagainya [9].

Program GPS dan operasionalnya saat ini sebagian besar bersumber pada Department of Defense (DoD). Amerika serikat yang dapa dikatakan merupakan pembuat sistem ini. Manajmennya sendiri dilaksanakan oleh US *Air Force* dengan panduan dari komite eksekutif DoD Positioning/Navigation. Komite ini menerima masukan dari komite yang sama dari Department of *Transportation* (DoT) yang bertindak sebagai suara sipil untuk urusan atauran GPS(NAPA, 1995). GPS asli hasil desain oleh US Department of Defense (DoD) terdiri atas tiga komponen utama, yaitu *control segment, space segment,* dan *user segment* [9].

1. GPS *Control Segment*

   Control segment terdiri atas lima stasiun yang terletak di pangkalan Falcon Air Force, Colorado Springs, Hawaii, Ascension Island, Diego Garcia dan Kwajalein. Stasiun-stasiun ini adalah mata dan telinga bagi GPS, bertugas memonitor satelit-satelit tersebut sebagaimana mereka mengirimkan data overhead dengan mengukr jarak antarsatelit setiap 1.5 detik (Hofmann-Wellenhof, 1992). Data ini kemudian diperhalus dengan menggunakan informasi ionospheric dan meteorologi dan dikirikan ke Master Control Station yang berada di fasilitas US *Air Force Space Command* yang ada di Colorado Spring. Disinilah parameter-parameternya baru dapat menggambarkan perkiraaan dari orbit satelit dan clock performance. Hal hampir sama terjadi pada peristiwa ketika kita memperkirakan status kondisi dari satelit dan menetapkan apakah re-positioning perlu dilakukan lagi. Informasi ini selanjutnya akan dikembalikan ketiga buah uplink station (ko-lokasi di stasiun pemonitor Ascension Island, Diego Garcia dan Kwajalein) yang akan mengirimkan informasi tersebut ke satelit. Dengan mengacu pada luasnya daerah penyebaran dari control station, semua satelit GPS di-tracking selama 92% dari waktu [9].

2. GPS *Space Segment*

   Space segment terdiri atas sebuah jaringan satelit dalam orbit lingkaran yang terdekat dengan tinggi nominar sekitar 20,183 km di atas permukaan bumi serta dengan periode selama 12 jam. Konstelasi yang sesungguhnya

adalah untuk 24 satelit, dalam 3 ruang orbit dan condong ke equator(Spilker, 1980). Akan tetapi, skenario ini telah mulai berubah dan satelit-satelit saat ini ditempatkan dalam enam ruang orbit yang berbeda, dengan emapt satelit di masing-masing ruang [9].

3. GPS *User Segment*

   User *segement* teridiri atas antenna dan prosessor *receiver* yang menyediakan positiong, kecepatan dan ketepatan waktu ke pengguna. Bagian ini menerima data dari satelit-satelit melalui sinyal radio yang dikirim-kan setelah mengalami koreksi oleh stasiun pengendali di daratan [9].

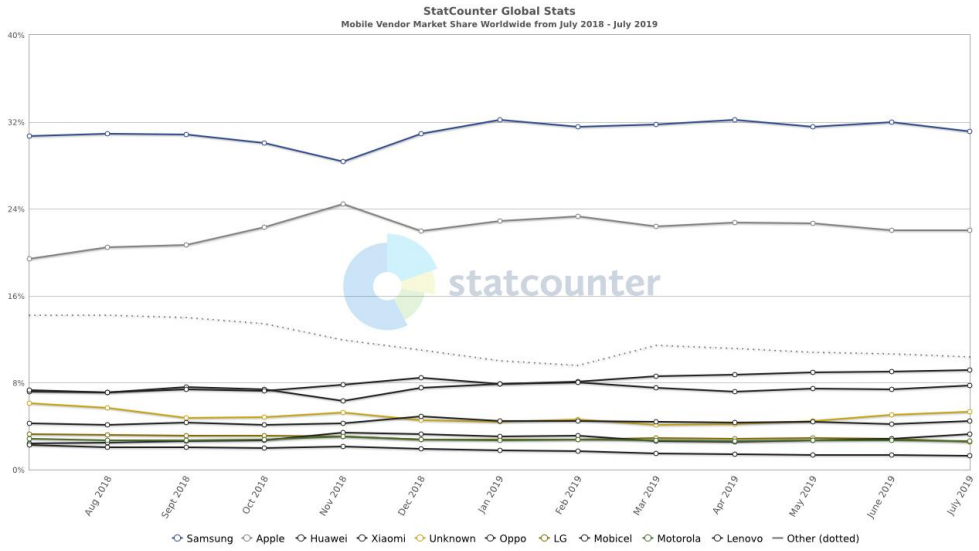
Kesimpulan menurut penulis penggunaan GPS sampai saat ini sangat dibutuhkan ditambah penulis membuat program aplikasi yang tujuannya untuk mengetahui posisi pengguna sebagai pelapor agar informasi lokasi yang dikirim lebih akurat.

2.2.6 Android

Gambar 2.5 Logo Android

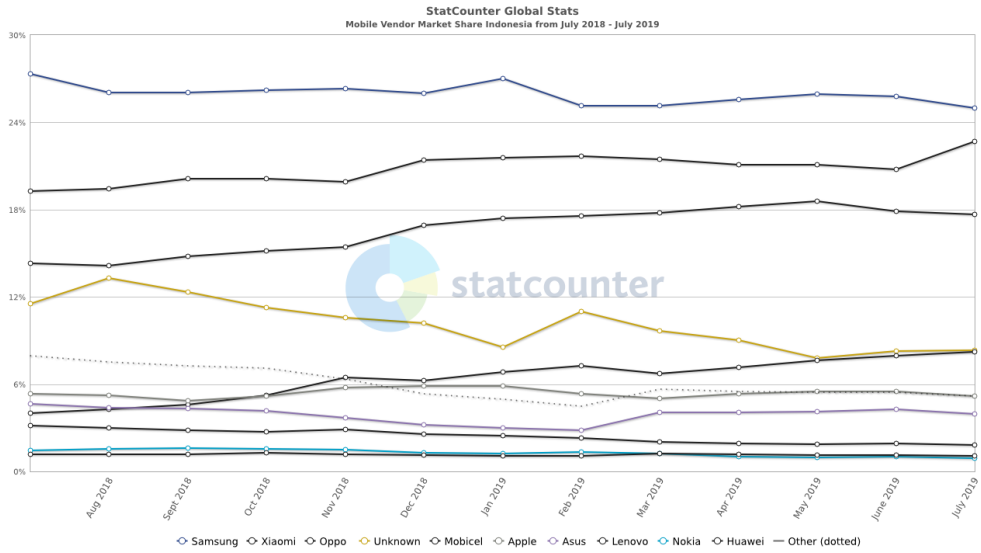
Android adalah salah satu sistem operasi yang pada awalnya, kemudian berkembang menjadi bahasa pemrograman yang banyak dicari dan digunakan oleh programmer. Pada dasarnya android adalah sistem operasi yang berbasis linux. Pengguna android pada awalna hanya digunakan untuk melengkapi sistem operasi pada *gadget-gadget* seluler seperti smartphone yang menggunakan layar sentuh. Tetapi karena sistem yang dikembangkan open source, mau tidak mau perkembangan dan penerimaan didunia industri IT menjadi lebih cepat juga [10].

Perusahaan android dibawah bendera android.inc, adalah pengembang pertama kali sistem operasi ini. Android.inc, pertama kali berdiri dikota Alto, salah satu kota erkenal di California Amerika Serikat, tepatnya pada bulan Oktober tahun 2003. Pendirinya terdiri dari tiga orang yang ahli dalam bidangn pengembangan aplikasi, mereka adalah Andy Rubin, Rich Miner, dan Chris White. Kemudian sejak tahun 2005, Google membeli dan selanjutnya di kembangkan oleh sumber daya Google sendiri sehingga mudah digunakan oleh khayalayak ramai. Dan sejat tahun 2007 secara resmi Google meluncurkan android sebagai sebuah sistem operasi baru khususnya untuk digunakan pada smartphone atau gadget, dan sistem operasi ini bersifat open source alias tidak diperjualbelikan. Sejak saat itu android bisa berkembang sedemikian rupa dan bisa menghasilkan berbagai macam aplikasi yang dibutuhkan oleh masyarakat untuk membantu kehidupan sehari-hari [10].



(Sumber : statcounter.com)

Gambar 2.6 Statistik Pengguna OS *Mobile* di Seluruh Negara



(Sumber : statcounter.com)

Gambar 2.7 Statistik Pengguna OS *Mobile* di Indonesia

Kesimpulan dari teori diatas, bahwa penulis memutuskan untuk membuat program Aplikasi berbasis android, salah satu faktor utama penulis memilih android adalah karena jumlah pengguna android jauh lebih banyak ketimbang sistem operasi lainnya.

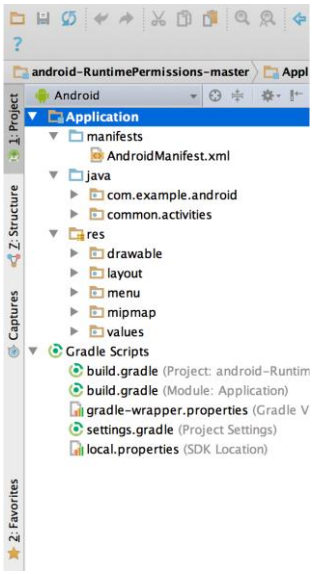
2.2.7 Android Studio

Android Studio adalah Lingkungan Pengembangan Terpadu - Integrated Development Environment (IDE) untuk pengembangan aplikasi Android, berdasarkan IntelliJ IDEA. Selain merupakan editor kode IntelliJ dan alat pengembang yang berdaya guna, Android Studio menawarkan fitur lebih banyak untuk meningkatkan produktivitas Anda saat membuat aplikasi Android, misalnya:

1. Sistem versi berbasis Gradle yang fleksibel
2. Emulator yang cepat dan kaya fitur
3. Lingkungan yang menyatu untuk pengembangan bagi semua perangkat Android
4. Instant Run untuk mendorong perubahan ke aplikasi yang berjalan tanpa membuat APK baru

5. Template kode dan integrasi GitHub untuk membuat fitur aplikasi yang sama dan mengimpor kode contoh
6. Alat pengujian dan kerangka kerja yang ekstensif
7. Alat Lint untuk meningkatkan kinerja, kegunaan, kompatibilitas versi, dan masalah-masalah lain
8. Dukungan C++ dan NDK
9. Dukungan bawaan untuk Google *Cloud Platform*, mempermudah pengintegrasian Google *Cloud Messaging* dan *App Engine.*

1. Struktur Proyek pada Android Studio



Gambar 2.8 Tampilan File Struktur Android Studio

Setiap proyek di Android Studio berisi satu atau beberapa modul dengan file kode sumber dan file sumber daya. Jenis-jenis modul mencakup:

1. Modul aplikasi Android
2. Modul Pustaka
3. Modul *Google App Engine*

Secara default, Android Studio akan menampilkan file proyek Anda dalam tampilan proyek Android, seperti yang ditampilkan dalam gambar 1.

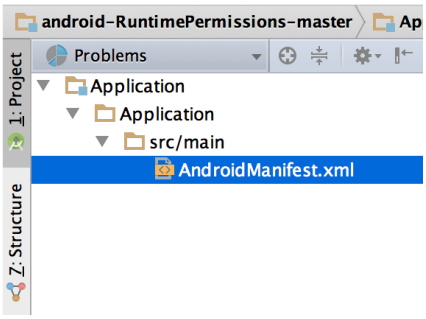
Tampilan disusun berdasarkan modul untuk memberikan akses cepat ke file sumber utama proyek Anda.

Semua file versi terlihat di bagian atas di bawah *Gradle Scripts* dan masing-masing modul aplikasi berisi folder berikut:

1. manifests: Berisi file AndroidManifest.xml.
2. java: Berisi file kode sumber Java, termasuk kode pengujian JUnit.
3. res: Berisi semua sumber daya bukan kode, seperti tata letak XML, string UI, dan gambar bitmap.

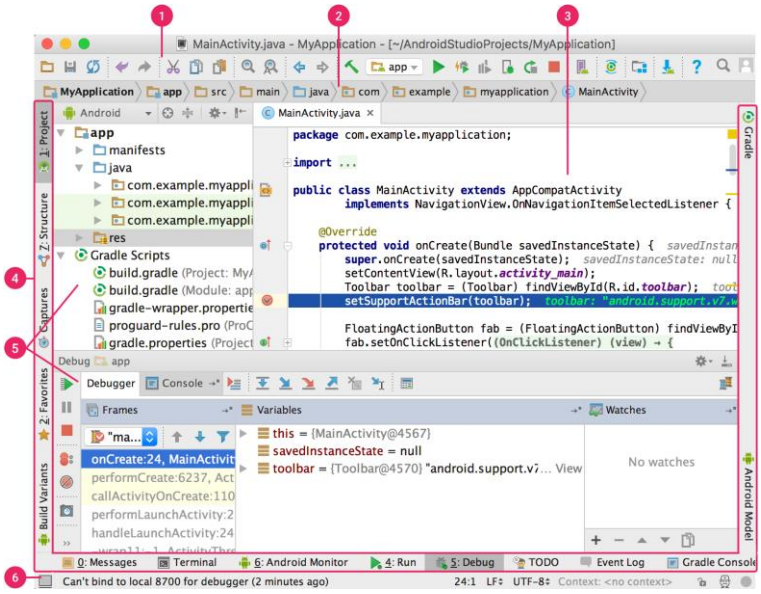
Struktur proyek Android pada disk berbeda dari representasi rata ini. Untuk melihat struktur file sebenarnya dari proyek ini, pilih *Project* dari menu tarik turun *Project* (dalam gambar 1, struktur ditampilkan sebagai Android).

Anda juga bisa menyesuaikan tampilan file proyek untuk berfokus pada aspek tertentu dari pengembangan aplikasi Anda. Misalnya, memilih tampilan *Problem* dari tampilan proyek Anda akan menampilkan tautan ke file sumber yang berisi kesalahan pengkodean dan sintaks yang dikenal, misalnya tag penutup elemen XML tidak ada dalam file tata letak.



Gambar 2.9 Tampilan File Proyek Android Studio

2. Antarmuka Pengguna pada Android Studio



Gambar 2.10 Tampilan Jendela utama Android Studio

1. Bilah alat memungkinkan Anda untuk melakukan berbagai jenis tindakan, termasuk menjalankan aplikasi dan meluncurkan alat Android.
2. Bilah navigasi membantu Anda bernavigasi di antara proyek dan membuka file untuk diedit. Bilah ini memberikan tampilan struktur yang terlihat lebih ringkas dalam jendela Project.
3. Jendela editor adalah tempat Anda membuat dan memodifikasi kode. Bergantung pada jenis file saat ini, editor dapat berubah. Misalnya, ketika melihat file tata letak, editor menampilkan Layout Editor.
4. Bilah jendela alat muncul di luar jendela IDE dan berisi tombol yang memungkinkan Anda meluaskan atau menciutkan jendela alat individual.
5. Jendela alat memberi Anda akses ke tugas tertentu seperti pengelolaan proyek, penelusuran, kontrol versi, dan banyak lagi. Anda bisa meluaskan dan juga menciutkannya.

6. Bilah status menampilkan status proyek Anda dan IDE itu sendiri, serta setiap peringatan atau pesan.

(sumber : https://www. developer.android.com/studio/intro/ 18-April-2019)

Dari teori yang didapat penulis memutuskan untuk menggunakan Android Studio sebagai alat dalam pembuatan program penulis, dalam mencari referensi tentang penggunaan Android Studio juga sangat banyak situs penyedia tutorial yang menggunakan Android Studio sebagai alat atau *tools* dalam pembuatan aplikasi.

2.2.8 WAMP Server

WampServer adalah platform pengembangan web berbasis Windows untuk aplikasi web dinamis menggunakan server Apache 2, bahasa *Scripting* PHP dan database MySQL, dengan PHPMyAdmin untuk mengelola database dengan mudah(sumber: https//www.Wampserver.com).

Gambar 2.11 Logo WAMP Server

Dibawah ini adalah beberapa langkah-langkah instalasi WampServer itu sendiri, sebagai berikut:

1. Klik dua kali pada file yang diunduh dan ikuti saja instruksinya. Semuanya otomatis. Paket WampServer dikirimkan dengan rilis terbaru dari Apache, MySQL dan PHP.
2. Setelah WampServer diinstal, Anda dapat secara manual menambahkan versi Apache, Php atau MySql (hanya VC9, VC10 dan VC11 yang dikompilasi) versi. Penjelasan akan diberikan di forum

3. Setiap rilis Apache, MySQL dan PHP memiliki pengaturan dan file sendiri (data untuk MySQL).

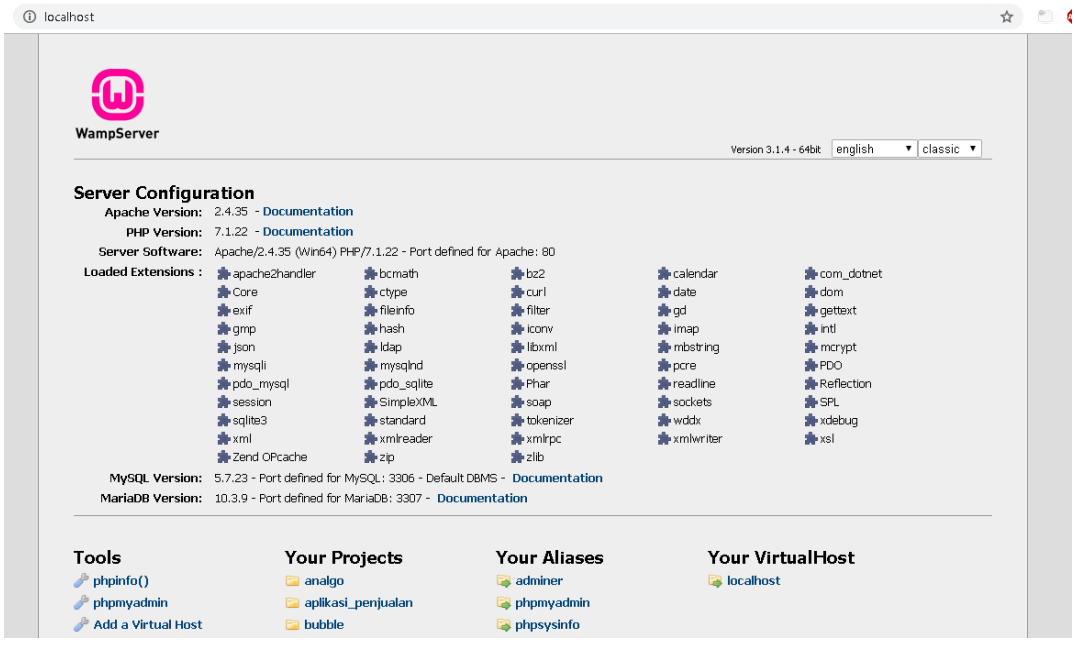
Dalam penggunaa WampServer di website resmi (https//www.wampserver.com) menjelaskan tentang cara penggunaannya, sebagai berikut:

1. Direktori "www" akan dibuat secara otomatis (biasanya c: \ wamp \ www)
2. Buat subdirektori dalam "www" dan letakkan file PHP Anda di dalamnya.
3. Klik tautan “localhost” di menu WampSever atau buka browser internet Anda dan buka URL: http: // localhost.

Beberapa fitur yang tersedia di WampServer juga bisa dilihat dibawah ini:

1. Kelola layanan Apache dan MySQL Anda
2. Beralih online / offline (berikan akses ke semua orang atau hanya localhost)
3. Instal dan alihkan rilis Apache, MySQL dan PHP
4. Kelola pengaturan server Anda
5. Akses log Anda
6. Akses file pengaturan Anda
7. Buat alias

Dibawah ini adalah tampilan awal ketika ingin menjalankan dan melihat database di PHPMyAdmin:



Gambar 2.12 Tampilan Awal WAMP Server

Dari semua penjelasan di atas, maka penulis untuk menggunakan WampServer sebagai salah satu alat pendukung dalam pembuatan program aplikasi. Alasan penulis lebih menggunakan WampServer karena menurut penulis, bahwa menggunakan WampServer lebih mudah dan simpel dalam penggunaannya, ketika ingin menjalankan Aplikasi WampServer pun tidak perlu mengkonfigurasi banyak agar bisa dijalankannya seperti Aplikasi lain yang serupa fungsinya. Tetapi beda dengan WampServer ketika penulis ingin menjalankannya tinggal sekali klik maka semua fungsi yang dibutuhkan umumnya akan langsung otomatis berjalan sendiri.

2.2.9 JAVA

Gambar 2.13 Logo Java

Menurut *Jubilee* di dalam bukunya berjudul "Java Komplet" mengatakan bahwa java adalah bahasa pemrograman yang powerful dan serbaguna untuk pengembangan perangkat lunak yang berjalan di perangkat seluler, komputer *desktop*, dan *server*. Awalnya, java dipanggil dengan sebutan Oak, java di rancang pada tahun 1991 untuk digunakan dalam chip tertanam pada peralatan elektronik yang ada di pasaran [11].

Pada tahun 1995 bahasa pemrograman ini berganti nama menjadi java yang lantas didesain ulang untuk mengembangkan aplikasi *web*. Keunggulan java utamanya bahwa kita bisa menulis sebuah program sekali saja dan hasil pemrograman itu bisa dijalankan dimana saja. Seperti yang dinyatakan oleh pengacaranya, java sederhana, berorientasi objek, terdistribusi, bersifat interpreter, kokoh, aman, portabel, berkinerja tinggi, multi *threaded*, dan dinamis [11].

Fitur yang tersedua di java, sebagai berikut :

1. **Berorientasi objek :** dalam java, semua adalah objek.

2. **Bersifat Platform Independent :** Java di-*compile* dalam bit kode *platform* independen dan bukan pada mesin *paltform* spesifik seperti pada C dan C++.
3. **Sederhana :** Java didesain untuk dapat dengan mudah dipelajari.
4. **Aman :** Dengan fitur kemanan Java, Anda dapat membuat sistem yang bebas virus dan powerful.
5. **Bersifat *Architectural-neutral* :** *Compiler* Java membuat format file objek yang *architectural-neutral*, yang membuat kode yang *decompile* dapat dieksekusi pada berbagai prosesor yang memiliki sistem *runtime* Java.
6. **Portabel :** Java bersifat *portable* karena adanya fitur *platform* independent dan *architectural-neutral*.
7. **Kuat dan powerful :** Java mengeliminasi error dengan menjalankan pengecekan pada waktu *compile* dan *runtime*.
8. **Multithreaded :** Dengan fitur multithread Java. Anda dapat membuat program yang dapat mengerjakan banyak tugas sekaligus.
9. **Terinterprestasi :** Kode bit Java ditranslasi secara langsung pada intruksi mesin dan tidak disimpan.
10. **Performa tinggi :** Java memiliki performa yang tinggi karena menggunakan compiler langsung.
11. **Terdistribusi :** Java didesain untuk lingkungan disribusi internet.
12. **Dinamis :** Java lebih dinamis dari C dan C++ karena Java didesain untuk beradaptasi dengan lingkungan pengembangan.

Kesimpulan dari teori tentang JAVA itu sendiri, penulis lebih memilih menggunakan bahasa pemrograman ini, karena menurut penulis yang masih awam untuk mencari referensi sangat mudah, kebanyakan programmer juga masih menggunakan bahasa pemrograman JAVA dan sampai saat ini masih populer.

2.2.10 XML

XML (*Extensible Markup Language*) adalah bahasa markup untuk keperluan umum yang disarankan oleh W3C untuk membuat dokumen markup keperluan pertukaran data antar sistem yang beraneka ragam. XML merupakan kelanjutan dari HTML (*HyperText Markup Language*) yang merupakan bahasa standar untuk melacak Internet.

XML didesain untuk mampu menyimpan data secara ringkas dan mudah diatur. Kata kunci utama XML adalah data (jamak dari datum) yang jika diolah bisa memberikan informasi.

XML menyediakan suatu cara terstandarisasi namun bisa dimodifikasi untuk menggambarkan isi dari dokumen. Dengan sendirinya, XML dapat digunakan untuk menggambarkan sembarang *view database*, tetapi dengan satu cara yang standar.

1. Tipe XML
    1) XML, merupakan standar format dari struktur berkas (*file*) yang ada.
    2) XSL, merupakan standar untuk memodifikasi data yang diimpor atau diekspor.
    3) XSD, merupakan standar yang mendefinisikan struktur d*atabase* dalam XML.
2. Keunggulan XML
    1) Pintar (*Intelligence*). XML dapat menangani berbagai tingkat (level) kompleksitas.
    2) Dapat beradaptasi. Dapat mengadaptasi untuk membuat bahasa sendiri. Seperti *Microsoft* membuat bahasa MSXML atau Macromedia mengembangkan MXML.
    3) Mudah pemeliharaannya.
    4) Sederhana. XML lebih sederhana.
    5) Mudah dipindah-pindahkan (*Portability*). XML mempunyai kemudahan perpindahan (portabilitas) yang lebih bagus.

    (sumber : https://id.wikipedia.org/wiki/XML/ 18-April-2019)

2.2.11 PHP

Menurut Beta Sidik dalam bukunya yang berjudul "Pemrograman *WEB* dengan PHP 7", PHP secara umum dikenal sebagai bahasa pemrograman *script-script* yang membuat dokumen HTML secara in the fly yang di eksekusi di *server web*, dokument HTML yang dihasilkan dari suatu aplikasi bukan dokumen HTML yang dibuat dengan menggunakan editor teks atau editor HTML. Dikenal juga sebagai bahasa pemrograman *server side*. Dengan menggunakan PHP maka *maintenance* suatu situs *web* menjadi lebih mudah. Proses update data dapat dilakukan dengan menggunakan aplikasi yang dibuat dengan menggunakan script PHP [10].

PHP/FI merupakan nama awal dari PHP. PHP – Personal Home Page, FI adalah *Form Interface*. Dibuat pertama kali oleh *Rasmus Lerdoff*. PHP, awalnya merupakan program CGI yang dikhususkan untuk menerima input melalui *form* yang ditampilkan dalam browser *web*. *Software* ini disebarkan dan dilisensikan sebagai perangkat lunak *Open Source*. CGI (*Common Gateway Interface*) sendiri menurut Beta Sidik adalah suatu standar yang menghubungkan (*interface*) aplikasi eksternal dengan *server web* [10].

Kini, PHP adalah kependekan dari PHP:*HyperText Preprocessor*(rekursif mengikut gaya penamaan di *nix), merupakan bahasa utama *script server side* yang disisipkan pada HTML yang dijalankan di *server*, dan juga bisa digunakan untuk

2.2.12 *Software Engineering* (RPL)

Rekayasa Perangkat Lunak (*software engineering*) merupakan pembangunan dengan menggunakan prinsip atau konsep rekayasa dengan tujuan menghasilkan perangkat lunak yang bernilai ekonomi yang dipercaya dan bekerja secara efisien menggunakan mesin.

Proses perangkat lunak (*software process*) adalah sekumpulan aktifitas yang memiliki tujuan untuk mengembangkan atau mengubah perangkat lunak. Secara umum proses perangkat lunak terdiri dari [13]:

1. Pengumpulan Spesifikasi (*Specification*)

   Mengetahui apa saja yang harus dapat dikerjakan sistem perangkat lunak dan batasan pengembangan perangkat lunak.

2. Pengembangan (*Development*)

   Pengembangan perangkat lunak untuk menghasilkan sistem perangkat lunak.

3. Validasi (*Validation*)

   Memerika apakah perangkat lunak sudah memnuhi keinginan pelanggan (customer).

4. Evolusi (*Evolution*)

   Mengubah perangkat lunak untuk memenuhi perubahan kebutuhan pelanggan(*Customer*)

Kesimpulannya adalah bahwa setiap pengembangan perangkat lunak hal dasar yang harus dipahami konsen rekaya perangkat itu sendiri, pamenurut penulis hal seperti bagi setiap pengembang pasti memahami dan menjadi salah satu faktor dalam membuat sebuah perangkat lunak.

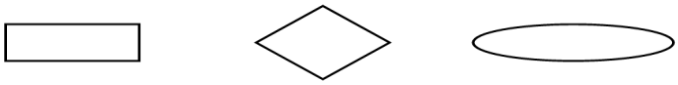
2.2.13 Pemrograman Terstuktur

Pemrograman tersetruktur adalah konsep atau paradigma atau sudut pandang pemrograman yang membagi-bagi program berdasarkan fungsi-funsi atau prosedur-prosedur yang dibutuhkan program komputer [13].

2.2.14 Entiti Relationship Diagram (ERD)

ERD adalah gambaran mengenai berelasinya antar entitas. Sistem adalah kumpulan elemen yang setiap elemennya memiliki fungsi masing-masing dan secara bersama-sama mencapai tujuan dari sistem tersebut [12].

Komponen-komponen ERD tediri dari :



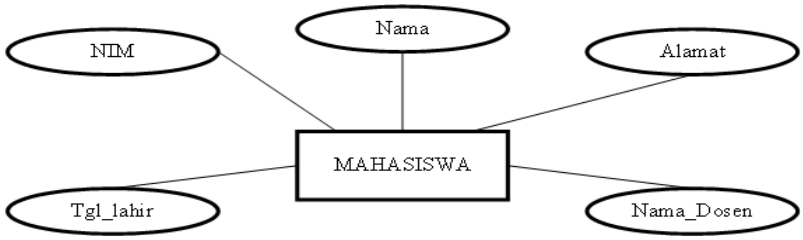
Gambar 2.14 Komponen-komponen ERD [14]

1. Entitas dan Atribut

   Entitas adalah tempat penyimpanan data, maka entitas yang digambarkan dalam ERD ini merupadakan data store yang ada di DFD dan akan menjadi

file data dikomputer. Entitas disebut juga suatu objek dan memiliki nama. Secara sederhananya dapat dikatakan bahwa jika objek ini tidak ada disuatu enterprise (lingkungan tertentu), maka enterprise tersebut tidak dapat berjalan nirmal. Contoh, entitas ‘Mahasiswa’ harus ada dilingkungan perguruan tinggi, begitu juga dengan entitas ‘Dosen;’Mata Kuliah, dan sebagainya.

Dibawah ini contoh penggambaran entitas dan atributnya.



Gambar 2.15 Hubungan Atribut dan Entitas [12]

2. Relasi

   Relasi adalah penghuung antara satu entitas (master file) dengan entitas lainnya didalam sebuah sistem komputer. Pada akhirnya, relasi akan menjadi file transaksi di komputer.

   Secara kalimat logis, contoh relasi yang terjadi sebuah perpustakaan adalah:”Anggota meminjam buku” atau “Anggota mengembalikan buku”, dalam hal ini, Anggota dan buku adalah entitas, meminjam dan mengembalikan adalah transaksi(relasi antara anggota dan buku) [14].

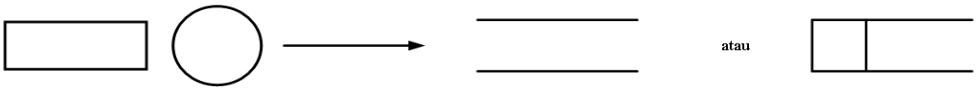
3. Derajat Kardinalitas (*Cardinality Degree*)

   Hubungan antara entitas ditanai pula oleh derajat kardinalitas. Fungsi dari derajat kardinalitas ini adalah untuk menentukan entitas kuat dan entitas lemah. Tiga jenis derajat kardinalitas adalah :(1) *one to one*, dilambangkan dengan 1:1, *one to many* dan sebaliknya, dilambangkan dengan 1:M dan sebaliknya, *many to many*, dilambangkan dengan M:M atau M:N. Entitas dengan derajat kardinalitas 1adalah entitas lemah sehingga entitas tersebut boleh digabung saja dengan entitas yang kuat (M) [14].

2.2.15 DFD

DFD merupakan salah satu komponen dalam serangkaian pembuatan perancangan sebuah sistem terkomuterisasi. DFD menggambarkan aliran data dari sumber pemberi data (*input*) ke penerima (*output*). Aliran data itu perlu diketahui agar si pembuat sistem tahu persis kapan sebuah data harus disimpan, kapan harus ditanggapi (proses), dan kapan harus didistribusikan ke bagian lain [14].

Komponen-komponen DFD terdiri dari:

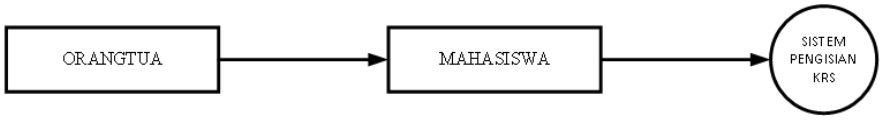


(Sumber : Pemodelan Visual dengan UML (Munawar))

Gambar 2.16 Komponen-komponen DFD [14]

1. Terimnator

   Suatu unit kerja/jabatan, atau sejenisnya yang berada diluar sistem tetapi memberi andil atas pemberian atau penerimaan data dari sistem secara langsung. Terimnator dapat pula disebut dengan ‘sumber pemberi data (*input*)’ maupun ‘tujuan pemberi data(*output*)’. Pemberi data dan penerima data yang dimaksud adalah pihak yang sangat dekat dan memiliki hubungan langsung dengan sistem. Adapun pihak luar yang berhubungan dengan pihak luar lainnya tidak boleh digambarkan. Misalkan, dalam pengisian KRS, mahaiswa berhubungan dengan sistem. Orang tua berhubungan dengan mahasiswa, tetapi tidak berhubungan dengan sistem, karenanya, kesatua luar ‘orang tua’, tidak boleh digambarkan [14].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.17 Contoh Hubungan Terimantor yang salah [14]

2. Proses

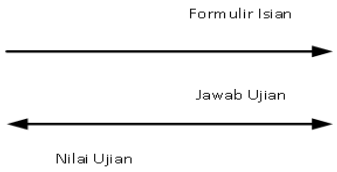
   Proses adalah suatu tindakan yang akan diambil terhadap data yang masuk. Karena proses adalah tindakan, maka proses berisi kata kerja. Proses diberikan identifikasi (nomor) agar mempermudah sekuen untuk diagram detailnya [10].

(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.18 Contoh Proses [14]

3. Alur Data

   Alur data menggambarkan data yang mengalir dari terminator ke proses atau dari proses ke proses lainnya. Data yang dibawa oleh alur data harus disebutkan dan diletakkan di atas lambang alur data dan bila alur data digambar panjang, sebaiknya penulisan data mendekati lambang anak panahnya [14].



(Sumber : Pemodelan Visual dengan UML [Munawar])

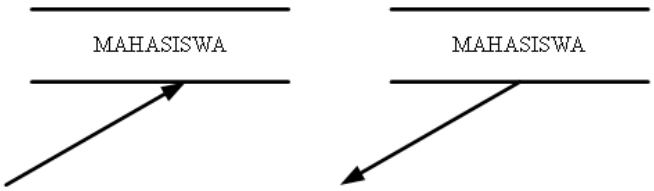
Gambar 2.19 Contoh Alur Data [14]

   Data yang menempati alur data dapat berupa elemen data tunggal, maupun elemen data. Misalkan, pada kumpulan elemen data: ‘Jawaban Ujian’, dapat ditulisa secara lengkap dengan menyebutkan setiap elemen data yang ada di sana, yaitu:’lembar jawaban’, dan ‘naskah soal’ [14].

4. Penyimpanan Data (Data *Store*)

   Data yang akan disimpan perlu ditempatkan ke satu tempat penyimpanan data. Data yang disimpan dapat berupa data manual maupun data digital. Untuk data digital, penyimpanan data tersebut kelak akan dijadikan file data di komputer. Alur data yang anak panahnya menuju ke roses dari penyimpanan data, kegiatannya adalah ‘membaca’ data, sehingga isi file data tidak akan berubah karenanya.

   Penyimpanan data harus diberi nama, misalkan data yang berisi biodata mahasiswa diberi nama ‘Mahasiswa’ [14].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.20 Contoh Penyimpan Data (Data Store) [14]

DFD digambarkan secara bertingkat, dari tingkat yang global berturut-turut hingga yang sangat detail. Tingkat yang global (umum) disebut dengan 'Diagram Konteks' atau '*Context Diagram'* , ini termasuk level 0. Selanjutnya dari diagram konteks, proses dijabarkan lebih rinci lagi 'Diagram Nol' atau '*Zero Diagram'* ini disebut level 1. Pada diagram nol ini yang berkembang hanya proses dan alur data yang menghubungkan proses-prosesnya, sedangkan jumlah terminator dan alur data yang masuk atau keluar dari terminator, tetap. Jika dirasa masih belum cukup makan perlu merinci proses berikutnya yaitu disebut 'diagram detail', ini disebut juga level 2. Dalam diagram detail, yang diagmbarkan cukup proses (nomor berapa) yang perlu di detailkan saja, selain itu (proses lainnya, atau terminatornya) tidak perlu digambarkan. Bila masih dapat lebih didetailkan lagim maka level 3 [14].

2.2.16 Kamus Data

Kamus data (data *dictionary*) dipergunakan untuk memperjelas aliran data yang digambarkan pada DFD. Kamus data adalah kumpulan daftar elemen data yang mengalir pada sistem perangkat lunak sehingga (input) dan keluaran (output) dapat dipahami secara umum (memiliki standar cara penulisan). Kamus data dalam implementasi program dapat menjadi parameter masukan atau keluaran dari sebuah fungsi atau prosedur [13]. Kamus data biasanya berisi :

1. Nama-nama dari data
2. Digunakan pada-merupakan proses-proses yang terkait data
3. Deskripsi-merupakan deskripsi data
4. Informasi tambahan-seperti tipe data, nilai data, batas nilai data, dan komponen yang membentuk data

Kamus data memiliki beberapa simbol untuk menjelaskan informasi tambahan sebagai berikut :

Tabel 2.1 Kamus Data

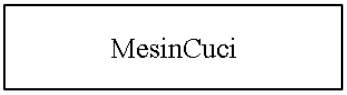
Simbol	Keterangan
=	Disusun atau terdiri dari
+	Dan
[|]	Baik...atau...
$\{\}^n$	N kali diulang/bernilai banyak
()	Data opsional
...	Batas komentar

2.2.17 Pemodelan dan Uml

UML(*Unified Modelling Language*) merupakan kesatuan dari bahasa pemodelan yang dikembang oleh Booch, *Object Modelling Technique* (OMT) dan *Object Oriented Software Engineering* (OOSE).metode *Booch* dari *Grady Booch* sangat terkenal dengan nama metode *Design Object Oriented.* Metode ini menjadikan proses analisis dan desain ke dalam empat tahapan iteratif, yaitu:identifikasi kelas-kelas dan obyek-obyek, identifikasi semantik dari hubungan obyek dan kelas tersebut, pencarian *interface* dan implementasi. Keunggulan metode *Booch* adalah pada detail dan kayanya dengan notasi dan elemen [15].

2.2.18 *Class* Diagram

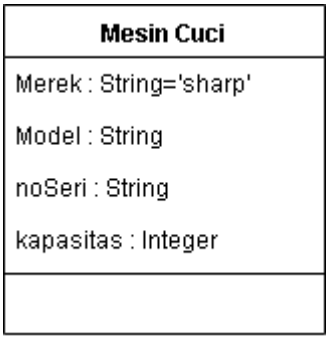
Class, dalam notasi UML digambarkan dengan kotak. Nama *class* menggunakan huruf besar di awal kalimatnya. Dan diletakkan di atas kotak. Bila *class* mempunyai nama terdiri dari 2 suku kata atau lebih, maka semua suku kata digabungkan tanpa spasi dengan huruf awal tiap suku kata menggunakan huruf besar [15].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.21 Notasi *Class* [15]

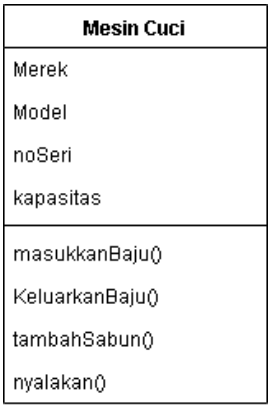
1. *Attribute*
   Attribute adalah properti dari sebuah *class*. *Attribute* ini melukiskan batas nilai yang mungkin ada pada obyek dari *class*. Sebuah *class* mungkin mempunyai nol atau lebih *attribute*. Secara konvensi, jika nama *attribute* terdiri atas satu suku kata, maka ditulis dengan huruf kecil. Akan tetapi jika nama *attribute* mengandung lebih dari satu suku kata maka semua suku kata digabungkan dengan suku kata pertama menggunakan huruf kecil dan awal suku kata berikutnya menggunakan huruf besar [15].



(Sumber : Pemodelan Visual dengan UML (Munawar))

Gambar 2.22 *Attribute* [15]

2. *Operation*
   Operation adalah suatu yang bisa dilakukan oleh sebuah *class* atau yang anda (atau *class* yang lain) dapat dilakukan untuk sebuah *class.* Seperti halnya *attribute,* nama *operation* juga menggunakan huruf kecil semua jika terdiri dari satu suku kata. Akan tetapi jika lebih dari satu suku kata, maka semua suku kata digabungkan dengan suku kata pertama huruf kecil dan huruf awal tiap suku berikutnya dengan huruf besar [15].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.23 *Operation* [15]

Tabel 2.2 *Visibility* [15]

Operator	Visibility	Keterangan
+	*Public*	Sebuah *operation* atau *attribute* bisa di akses oleh siapapun
-	*Private*	Fitur yang hanya digunakan oleh *instance* dari *class*
#	*Protected*	Seperti private tetapi boleh di akses oleh anak-anaknya
-	*Package*	Bisa diakses langsung oleh *instance* dengan *package* sama

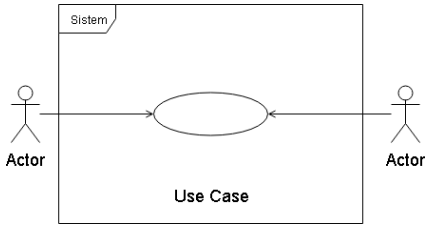
2.2.19 *Use Case* Diagram

Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara user (pengguna) sebuah sistem dengan sistemnya sendiri melalu sebuah cerita bagaiamana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut skenario. Setiap skenario mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatakan *use case* adalah serangkaian skenario yang digabungkan bersama-sama oleh tujuan umum pengguna [13].

Dalam use case, pengguna biasanya disebut dengan aktor, aktor adalah sebuah peran yang bisa dimainkan oleh pengguna dalam interaksinya dengan sistem [15]. Model *use case* adalah bagian dari mode *requirement*(*Jacobson et all*,1992). Termasuk disini adalah *problem domain object model* dan penjelasan tentang user *interface*. *Use case* memberikan spesifikasi fungsi-fungsi yang di tawarkan oleh sistem dari perspektif user [15].

1. Notasi *Use Case*

   Diagram *use case* menunjukkan 3 aspek dari sistem yaitu *actor*, *use case*, dan sistem/sub sistem boundari. *Actor* mewakili peran orang, sistem yang lain atau alat ketika berkomunikasi dengan *use case*. Gambar 2.21 mengilustrasikan *actor*, *use case* dan boundari [15].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.24 *Use Case* Model [15]

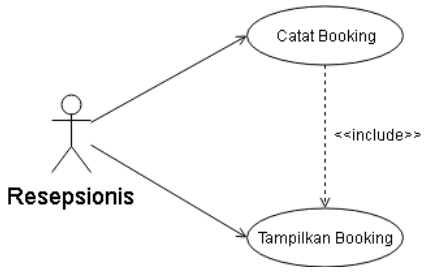
2. Identifikasi *Actor*

   Dalam mengidentifikasi *actor*, terlebih dahulu menentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks target sistem. *Actor* adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Perlu dicatat bahwa *actor* berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case* [15].

3. *Stereotype*

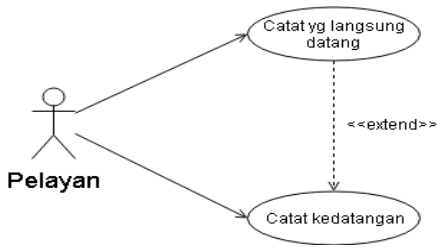
   Stereotype adalah sebuah model khusus yang terbatas untuk kondisi tertentu. Untuk menunjukkan stereotype digunakan simbol “<<” di awalnya dan ditutup ”>>” di akhirnya. <<*extend*>> digunakan untuk menunjukkan bahwa satu *use case* merupakan tambahan fungsional dari *use case* yang lain jika kondisi atau syarat tertentu yang dipenuhi. Sedangkan <<*include*>>

digunakan untuk menggambarkan bahwa suatu *use case* seluruhnya merupakan fungsionalitas dari use case lainnya. Biasanya <<*include*>> digunakan untuk menghindari pengkopian suatu *use case* karena sering dipakai [15].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.25 *Use Case Inclusion* [13]

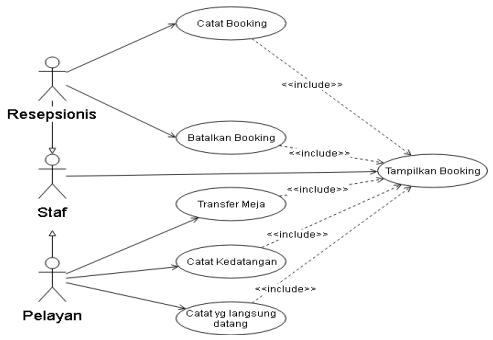


(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.26 *Use Case Extension* [15]

4. Level *Use Case*

   Masalah umum dengan *use case* adalah dengan memfokuskan pada interaksi diantara dua user dan sistem. Sering kali perubahan pada proses bisnis adalah jalan terbaik untuk menyelesaikan masalah. Oleh karena itu *use case* perlu dibedakan menjadi sistem *use case* dan *business* use case. Sistem *use case* adalah sebut interaksi dengan *software*, sedangkan *businees use case* lebih menekankan kepada bagaimana sebuah bisnis merespon ke pelanggan atau kejadian/*event* [15]. Berikut contoh pada gambar 2.27.



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.27 *Use Case Extension* [15]

5. Skenario *Use case* / Dokumentasi *Use Case*

   Setiap *use case* harus dideskripsikan dalam dokumen yang disebut dengan dokumen *flow of event.* Dokumen/skenario ini mendefinisikan apa yang harus dilakukan oleh sistem ketika aktor mengaktifkan *use case*. Struktur dokumen/skenario *use case* ini bisa bermacam-macam, tetapi umumnya deksripsi ini paling tidak harus mengandung: (sumber Quatrani,2000) [15].

   1) *Brief Description* yaitu deskripsi singkat.
   2) *Actor* yang terlibat
   3) *Precondition* yang penting bagi use case untuk memulai
   4) *Main flow* dari kejadian yang bisa dirinci lagi menjadi *sub flow* yang lebih kecil agar dokumen lebih mudah di baca dan dimengerti.
   5) *Alternatif flow* untuk mendefinisikan situasi perkecualian
   6) *Postcondition* yang menjelaskan *state* dari sistem setelah *use case* berakhir.

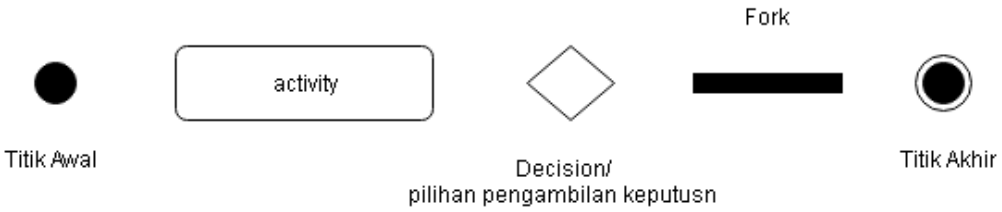
Tabel 2.3 Contoh Dokumen/Skenario *use case* [13]

Use Case	Namaa use case
Biref Description	Deskripsi singkat
Actor	Aktor yang terlibat
Precondition	Yang penting bagi use case untuk memulai
Main flow	Kejadian yang bisa di rinci
Alternatif flow	Mendefinisikan situasi alternatifnya/pengecualian
Postcondition	Menjelaskan hasil dari *state* yang sudah dilakukan

2.2.20 *Acitivity* Diagram

Diagram aktivitas atau *activity* diagram adalah teknik untuk mendeskripsikan logika prosedural, proses bisnis dan workflow (aliran kerja) dalam banyak kasus. *Activity* diagram mempunyai peran seperti halnnya *flowcar*t, akan tetapi perbedaannya dengan flowchart adalah activity diagram bisa mendukung perilaku paralel sedangkan flowcart tidak bisa [15].

Dibawah ini adalah contoh beberapa simbol yang umum dipakai dan tentu digunakan dalam penelitian penulis, sebagai berikut:



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.28 Contoh simbol umum *activity* diagram [15]

2.2.21 Sequence Diagram

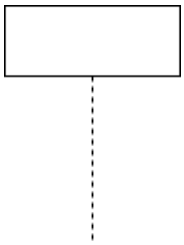
Diagram *sequence* atau *Sequence* Diagram digunakan untuk menggambarkan perilaku pada sebuah skenario. Diagram ini menunjukkan sejumlah contoh objek dan *message* (pesan) yang diletakkan diantara obyek-obyek ini didalam *use case*.

Komponen utama *sequence* diagram terdiri atas obyek yang dituliskan dengan kotak segiempat bernama. *Message* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan *progress vertical* [15].

1) Obyek/*Participant*

   Obyek diletakkan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram. Pengertian obyek hanya ada di UML 1, sedangkan di UML 2 istilah obyek diganti dengan *participant* [15].

   Setiap *participant* terhubung dengan garis titil-titik yang disebut *lifeline*. Sepanjang *lifeline* ada kotak yang disebut *activation*. *Activation* mewakili sebuah eksekusi operasi dari *participant.* Panjang kotak ini berbanding lurus dengan durasi *activation* [15].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.29 Obyek/*Participant* pada *sequence diagram* [15]

2) *Message*

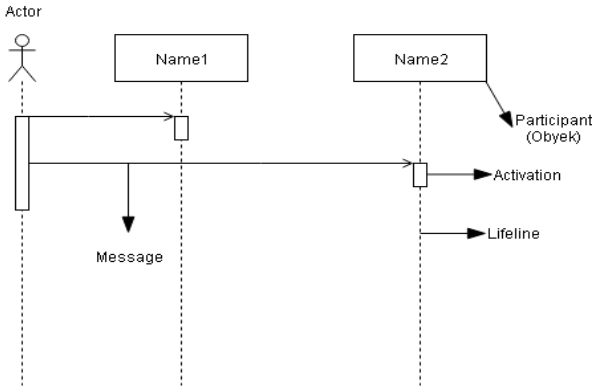
Sebuah *message* bisa jadi *simple*, *synchronous* atau *asynchronous*. *Message* yang simpel adalah sebuah perpindahan (transfer) kontrol dari satu *participant* ke *participant* yang lainnya. Jika sebuah atas *message* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *message asynchronous* yang dikirimkan, maka jawaban atas *message* pada *sequence diagram* bisa dilihat sebagai berikut [15]:

(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.30 Simbol-simbol *message* [15]

3) *Time*

Time adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas kebawah. *Message* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibanding *message* yang lebih dekat kebawah [13].



(Sumber : Pemodelan Visual dengan UML [Munawar])

Gambar 2.31 Simbol-simbol yang ada pada *sequence diagram* [15]

Berikut Tabel 2.4 Operator umum yang sering digunakan di *interaction frame*:

Tabel 2.4 Operator-operator umum pada *interaction frame* [15]

Operator	Keterangan
Alt	Alternatif fragment. Hanya kondisi True yang akan dijalankan
Opt	Optional fragment. Jika kondisi mendukungnya True
Par	Paralel. Setiap fragment dijalankan secara paralel
Loop	Looping. Dijalankan berulang kali, guard menunjukan basis iterasi
Region	Fragment hanya punya satu thread untuk menjalankannya
Neg	Negative, fragment menunjukkan interaction yang salah
Ref	Rerences, menunjukkan interaksi pada diagram class lain
Sd	Sequence diagram