BAB 2
TINJAUAN PUSTAKA

2.1 Landasan Teori

Landasan teori menjelaskan beberapa definisi dan teori yang berkaitan dengan perencanaan aplikasi sebagai sumber dasar pemahaman dalam sebuah sistem serta metode yang digunakan untuk kegiatan pembangunan aplikasi tersebut.

2.1.1 Apotek

Apotek adalah sarana pelayanan kefarmasian tempat dilakukan praktik kefarmasian oleh apoteker. Standar pelayanan kefarmasian di apotek ditetapkan sebagai acuan pelaksanaan pelayanan kefarmasian di apotek. Untuk keberhasilan pelaksanaan standar pelayanan kefarmasian di apotek diperlukan komitmen dan kerjasama semua pemangku kepentingan. Hal tersebut akan menjadikan pelayanan kefarmasian di apotek semakin optimal dan dapat dirasakan manfaatnya oleh pasien dan masyarakat yang pada akhirnya dapat meningkatkan mutu pelayanan kesehatan [3].

Menurut Keputusan Menkes RI No.1332/Menkes/SK/X/2002 Apotek merupakan suatu tempat tertentu untuk melakukan pekerjaan kefarmasian dan penyaluran obat kepada masyarakat. Sedangkan definisi apotek menurut PP 51 Tahun 2009. Apotek merupakan suatu tempat atau terminal distribusi obat perbekalan farmasi yang dikelola oleh apoteker sesuai standar dan etika kefarmasian.

Adapun tugas dan fungsi Apotek sebagai berikut :

1. Tempat pengabdian profesi seorang apoteker yang telah mengucapkan sumpah jabatan.
2. Sarana farmasi untuk melaksanakan peracikan, pengubahan bentuk pencampuran dan penyerahan obat atau bahan obat.
3. Sarana penyaluran perbekalan farmasi dalam menyebarkan obat – obatan yang diperlukan masyarakat secara luas dan merata.

2.1.2 Aplikasi

Menurut Jogiyanto (1999:12) adalah penggunaan dalam suatu komputer, instruksi (*instruction*) atau pernyataan (*statement*) yang disusun sedemikian rupa sehingga komputer dapat memproses *input* menjadi *output*. Sedangkan menurut Kamus Kamus Besar Bahasa Indonesia (1998 : 52) adalah penerapan dari rancang sistem untuk mengolah data yang menggunakan aturan atau ketentuan bahasa pemrograman tertentu. Menurut Rachmad Hakim S, Aplikasi adalah perangkat lunak yang digunakan untuk tujuan tertentu, seperti mengolah dokumen, mengatur *Windows*, permainan (*game*), dan sebagainya. Dari beberapa pendapat beberapa ahli diatas ditarik kesimpulan bahwa, aplikasi adalah adalah program siap pakai yang dapat digunakan untuk menjalankan perintah-perintah dari pengguna aplikasi tersebut dengan tujuan mendapatkan hasil yang lebih akurat sesuai dengan tujuan pembuatan aplikasi tersebut, aplikasi mempunyai arti yaitu pemecahan masalah yang menggunakan salah satu teknik pemrosesan data aplikasi yang biasanya berpacu pada sebuah komputasi yang diinginkan atau diharapkan maupun pemrosesan data yang diharapkan.

2.1.3 Android

Android adalah sistem operasi bersifat *open source* berbasis Linux dirancang untuk perangkat seluler layar sentuh seperti telepon pintar dan komputer tablet. Android awalnya dikembangkan oleh Android, Inc., dengan dukungan finansial dari Google, yang kemudian membelinya pada tahun 2005. Sistem operasi ini dirilis secara resmi pada tahun 2007, bersamaan dengan didirikannya Open Ponsel Android pertama mulai dijual pada bulan oktober 2008 [4].

Berikut merupakan beberapa versi sistem operasi yang ada di android. Karena pembangunan aplikasi *Smart Apotek* ini menggunakan aplikasi berbasis *mobile*, dan setelah mengetahui hasil dari pertanyaan kuesioner yang disebarkan ke beberapa responden. Ternyata penggunaan *smartphone mobile* android sangat banyak peminatnya, maka dari itu untuk pembangunan aplikasi yang tepat untuk digunakan oleh apoteker dan pelanggan ini bersifat *mobile* android. Penjelasan tentang versi android terdapat pada Tabel 2.1 Daftar Versi Android sebagai berikut :

Tabel 2.1 Daftar Versi Android

Sistem Operation Android	Versi
Tanpa Codename	1.1
Cupcake	1.5
Donut	1.6
Éclair	2.0 / 2.1
Froyo	2.2
Gingerbread	2.3
Honeycomb	3.0 / 3.1
Ice Cream Sandwich	4.0
Jelly Bean	4.1 / 4.2 / 4.3
Kitkat	4.4
Lollipop	5.0
Marshmallow	6.0
Nougat	7.0 / 7.1

2.1.4 *Global Positioning System*

Global Positioning System adalah sistem navigasi berbasis satelit terdiri dari jaringan 24 satelit ditempatkan ke orbit oleh Departemen Pertahanan Amerika Serikat yang pertama kali diperkenalkan mulai tahun 1978. Layanan GPS dahulu hanya dipergunakan untuk keperluan militer namun mulai terbuka untuk publik. 24 satelit GPS tersebut berada sekitar 12.000 mil di atas bumi bergerak mengelilingi bumi 12 jam dengan kecepatan 7.000 mil per jam. Satelit GPS berkekuatan energi sinar matahari, memiliki baterai cadangan untuk menjaga agar tetap berjalan pada saat gerhana matahari atau pada saat tidak ada energi matahari dan memiliki roket penguat kecil pada masing-masing satelit agar dapat mengorbit tepat pada tempatnya.

Satelit-satelit GPS harus selalu berada pada posisi orbit yang tepat untuk menjaga akurasi data yang dikirim ke GPS *receiver*, sehingga harus selalu dipelihara agar posisinya tepat. Posisi satelit-satelit tersebut selalu dipantau oleh stasiun

pengendali. Stasiun-stasiun pengendali di bumi ada di Hawaii, Ascension Island, Diego Garcia, Kwajalein dan Colorado Spring. Untuk dapat mengetahui posisi seseorang maka diperlukan alat yang diberi nama GPS *receiver* yang berfungsi untuk menerima sinyal yang dikirim dari satelit GPS. GPS *receiver* mengambil informasi tersebut dan melakukan perhitungan *triangulation* untuk menentukan lokasi pengguna dengan tepat. GPS *receiver* membandingkan waktu sinyal dikirim dengan waktu sinyal tersebut diterima untuk mengetahui jarak satelit. Dengan mengetahui jarak tersebut, GPS *receiver* dapat melakukan perhitungan dan menentukan posisi pengguna dan menampilkan dalam peta elektronik. Setelah menentukan posisi pengguna, selanjutnya GPS dapat menghitung informasi lain, seperti kecepatan, arah yang dituju, jalur, tujuan perjalanan, jarak tujuan, matahari terbit dan matahari terbenam dan masih banyak lagi.

Satelit GPS sangat presisi dalam mengirim informasi karena satelit tersebut menggunakan jam atom. Jam atom yang ada pada satelit merupakan partikel atom yang diisolasi, sehingga dapat menghasilkan jam yang akurat dibandingkan dengan jam biasa. Keistimewaan GPS adalah mampu bekerja dalam berbagai kondisi cuaca, siang atau malam. Keakuratan sebuah perangkat GPS bisa mencapai 15 meter, bahkan model terbaru yang dilengkapi teknologi *Wide Area Augmentation System* (WAAS) keakuratannya sampai 3 meter. Karena GPS bekerja mengandalkan satelit, maka penggunaannya disarankan di tempat terbuka. Penggunaan di dalam ruangan, atau di tempat yang menghalangi arah satelit (di angkasa), maka GPS tidak akan bekerja secara akurat dan maksimal. Perhitungan waktu yang akurat sangat menentukan akurasi perhitungan untuk menentukan informasi lokasi. Semakin banyak sinyal satelit yang dapat diterima maka akan semakin presisi data posisi yang dihasilkan. Selain itu, ketinggian juga mempengaruhi proses kerja GPS, karena semakin tinggi maka semakin bersih atmosfer, sehingga gangguan semakin sedikit [5].

2.1.4.1 Akurasi Global Positioning System

Posisi yang ditunjukkan oleh suatu GPS mempunyai faktor kesalahan atau juga disebut tingkatan akurasi. Sebagai contoh suatu alat GPS menunjukkan titik koordinat dengan tingkat akurasi 5 meter, itu berarti posisi pengguna bisa berada dalam

range radius 5 meter dari titik yang ditunjukkan tersebut. Mengapa tingkat akurasi yang terlihat bisa berubah-ubah? Kadang terlihat 10 meter, 15 meter atau 5 meter. Ada beberapa hal yang mempengaruhi tingkat akurasi tersebut, antara lain :

1. Kesalahan *Ephemeris*. Terjadi jika satelit tidak dapat mentransmisikan posisi diorbit dengan tepat.
2. Keadaan *lonosphere*. *Lonosphere* berada pada jarak sekitar 43-50 mil di atas permukaan bumi. Satelit yang melewati *ionosphere* akan menjadi lambat dikarenakan adanya plasma (gas dengan tingkat kepadatan rendah). Walaupun GPS *receiver* berusaha untuk mengkoreksi atau memperbaiki faktor keterlambatan yang terjadi tetap saja aktivitas tertentu dari plasma bisa menyebabkan kesalahan perhitungan.
3. Keadaan *Troposphere*. *Troposphere* adalah bagian terendah dari atmosfer sampai dengan ketinggian sekitar 11 mil dari permukaan tanah. Variasi pada temperatur, tekanan dan kelembaban bisa menyebabkan perbedaan kecepatan penerima gelombang radio.
4. Kesalahan Waktu. Kesalahan waktu dari GPS *receiver* yang tidak presisi dapat menimbukan ketidakakurasian.
5. Kesalahan *Multipath*. Terjadi karena sinyal satelit membentur permukaan keras (seperti bangunan atau tebing) sebelum mencapai GPS *receiver*. Hal tersebut bisa menyebabkan terjadinya *delay* sehingga perhitungan jarak menjadi tidak akurat.
6. Buruknya Sinyal Satelit. Keadaan sekitar atau keadaan lingkungan dapat juga menyebabkan GPS sulit untuk menerima data satelit [6].

2.1.5 *Location Based Service*

Location Based Service (LBS) atau layanan berbasis lokasi merupakan layanan informasi yang dapat diakses melalui *mobile device* dengan menggunakan *mobile network*, yang dilengkapi dengan kemampuan untuk memanfaatkan lokasi dari *mobile device* tersebut. *Location Based Service* dapat berfungsi sebagai layanan untuk

mengidentifikasi lokasi dari seseorang atau suatu objek tertentu, seperti menemukan tempat wisata atau lokasi lainnya.

Dalam menggunakan *Location Based Service* atau layanan berbasis lokasi terdapat beberapa elemen yang diperlukan. Elemen yang diperlukan adalah sebagai berikut:

1. *Mobile Devices*

   Sebuah alat yang digunakan untuk meminta informasi yang dibutuhkan seperti perangkat *mobile smartphone* yang memiliki fasilitas navigasi.

2. *Communication Network*

   Jaringan seluler yang mengirimkan data pengguna dan permintaan layanan.

3. *Positioning Component*

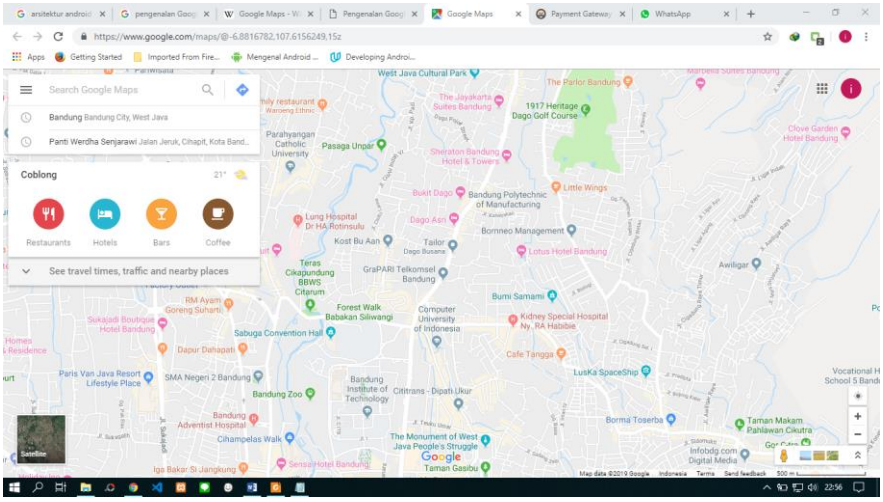
   Untuk dapat mengolah layanan harus menentukan lokasi pengguna. Posisi pengguna ini dapat diperoleh menggunakan jaringan komunikasi atau dengan menggunakan *Global Positioning System* (GPS).

4. *Service and Application Provider*

   Penyedia layanan pengguna seluler yang bertanggung jawab untuk memproses suatu layanan [6].

2.1.6 Google *Maps*

Google *Maps* adalah layanan *mapping online* yang disediakan oleh google. Layanan ini dapat diakses melalui situs http://maps.google.com.



Gambar 2.1 Google Maps

Pada situs tersebut kita dapat melihat informasi geografis pada hampir semua wilayah di bumi. Google *Maps* dibuat dengan menggunakan kombinasi dari gambar peta, database, serta obyek-obyek yang interaktif yang dibuat dengan Bahasa pemrograman HTML, Javascript dan AJAX, dan beberapa bahasa pemrograman lainnya . Bahasa pemrograman dari Google *Maps* yang terdiri dari HTML dan Javascript, memungkinkan untuk menampilkan Google *Maps* di website lain. Kostumisasi dari aplikasi ini dimungkinkan dengan disediakannya *client-side scripts* dan *server-side hooks.*

Google *Maps Application Programming Interface* (API) merupakan suatu fitur aplikasi yang dikeluarkan oleh google untuk memfasilitasi pengguna yang ingin mengintegrasikan Google *Maps* ke dalam website masing-masing dengan menampilkan data point milik sendiri. Agar aplikasi Google *Maps* dapat muncul di website tertentu, diperlukan adanya API *key*. API *key* merupakan kode unik yang digenerasikan oleh google untuk suatu website tertentu, agar *server* Google *Maps* dapat

mengenali. Google *Maps* API telah menyediakan template dasar yang dapat digunakan oleh pengguna untuk dikembangkan lebih lanjut.

Menggunakan atau memprogram Google *Maps* API sedikit menyulitkan karena yang kita butuhkan adalah pengetahuan tentang HTML dan JavaScript, serta koneksi Internet. Dengan menggunakan Google *Maps* API kita dapat menghemat waktu dan biaya kita untuk membangun aplikasi peta digital yang handal, sehingga kita dapat fokus hanya pada data-data kita. Biarkan data peta-peta dunia menjadi urusan Google saja [7].

2.1.7 Java

Java adalah nama untuk sekumpulan teknologi untuk membuat dan menjalankan perangkat lunak pada komputer personal ataupun pada lingkungan jaringan. Java 2 adalah generasi kedua dari java *platform* (generasi awalnya adalah *Java Development Kit*). Java berdiri di atas mesin *interpreter* yang diberi nama *Java Virtual Machine* (JVM). JVM inilah yang akan membaca bytecode dalam file.*class* dari suatu program sebagai sebuah representasi langsung program yang berisi bahasa mesin. Oleh karena itu, bahasa java disebut sebagai bahasa pemrograman yang *portable* karena dapat dijalankan pada berbagai sistem operasi, asalkan pada sistem operasi tersebut terdapat JVM. *Platform* java memiliki tiga buah edisi yang berbeda, yaitu J2EE (*Java2 Enterprise Edition*), J2ME (*Java2 Micro Edition*), dan J2SE (*Java2 Second Edition*) [8].

Java memiliki tiga *platform* yang masing-masing mengarahkan sebagai tujuan tertentu dan sebagai lingkungan komputasi yang berbeda-beda :

1. *Standard Edition (J2SE)*

J2SE merupakan inti dari sebuah bahasa pemrograman Java. J2SE didesain sebagai jalan pada komputer desktop dan komputer *workstations*.

2. *Enterprise Edition (J2EE)*

J2EE berfungsi sebagai built-in dan mendukung untuk *servelts*. JSP dan XML, edisi ini bertujuan untuk aplikasi yang berbasis *server.*

3. *Micro Edition (J2ME)*

J2ME didesain untuk piranti dengan keterbatasan memori, layar *display* terbatas dan power pemrosesannya terbatas juga.

2.1.8 *Web Service*

Web service adalah suatu sistem perangkat lunak yang dirancang untuk mendukung interoperabilitas dan interaksi antar sistem pada suatu jaringan. *Web service* digunakan sebagai suatu fasilitas yang disediakan oleh suatu web site untuk menyediakan layanan (dalam bentuk informasi) kepada sistem lain, sehingga sistem lain dapat berinteraksi dengan sistem tersebut melalui layanan-layanan *(service*) yang disediakan oleh suatu sistem yang menyediakan *web service. Web service* menyimpan data informasi dalam format XML, sehingga data ini dapat diakses oleh sistem lain walaupun berbeda *platform*, sistem operasi, maupun bahasa *compiler.*

Web service bertujuan untuk meningkatkan kolaborasi antar pemrogram dan perusahaan, yang memungkinkan sebuah fungsi di dalam *Web Service* dapat dipinjam oleh aplikasi lain tanpa perlu mengetahui detail pemrograman yang terdapat di dalamnya.

Beberapa alasan mengapa digunakannya *web service* adalah sebagai berikut:

1. *Web service* dapat digunakan untuk mentransformasikan satu atau beberapa bisnis *logic* atau *class* dan objek yang terpisah dalam satu ruang lingkup yang menjadi satu, sehingga tingkat keamanan dapat ditangani dengan baik.
2. *Web service* memiliki kemudahan dalam proses *deployment*-nya, karena tidak memerlukan registrasi khusus ke dalam suatu sistem operasi. *Web service* cukup di-*upload* ke *web server* dan siap diakses oleh pihak-pihak yang telah diberikan otorisasi.
3. *Web service* berjalan di port 80 yang merupakan protokol standar HTTP, dengan demikian *web service* tidak memerlukan konfigurasi khusus di sisi *firewall* [9].

2.1.9 *HyperText Markup Language* (HTML)

HyperText Markup Language (HTML) adalah bahasa yang digunakan untuk menulis halaman web. HTML merupakan pengembangan dari standar pemformatan dokumen teks, yaitu *Standard Generalized Markup Language* (SGML). HTML pada dasarnya merupakan dokumen ASCII atau teks biasa, yang dirancang untuk tidak tergantung pada suatu system operasi tertentu. HTML dibuat oleh Tim Berners-Lee ketika masih bekerja untuk CERN, dan dipopulerkan pertama kali oleh browser Mosaic. Selama awal tahun 1990, HTML mengalami perkembangan yang sangat pesat. Setiap pengembangan HTML, pasti akan menambahkan kemampuan dan fasilitas yang lebih baik dari versi sebelumnya [10].

2.1.10 *Cascading Style Sheet* (CSS)

CSS (*Cascading Style Sheet*) adalah suatu bahasa *stylesheet* yang digunakan untuk mengatur tampilan suatu *website*, baik tata letaknya, jenis huruf, warna, dan semua yang berhubungan dengan tampilan. Pada umumnya CSS digunakan untuk memformat halaman web yang ditulis dengan HTML atau XHTML. Ada dua cara yang bias diterapkan untuk menggunakan CSS pada web. Cara yang pertama dengan membuat CSS langsung didalam satu file HTML (*internal/ inline style sheet*). Cara yang kedua dengan memanggil CSS tersebut dari file CSS tersendiri (*external style sheet*) [10].

2.1.11 JSON (*JavaScript Object Notation*)

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 Pemprograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman 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 :

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

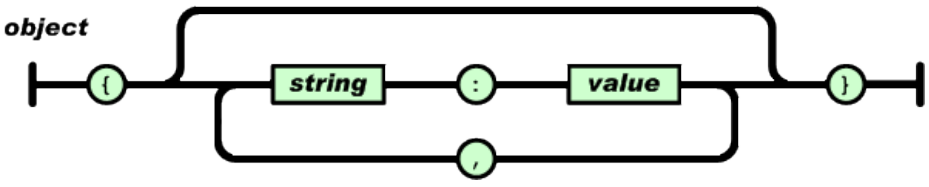
Struktur-struktur data ini disebut sebagai struktur data universal. Pada dasarnya, semua bahasa pemprograman *modern* mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemprograman yang juga berdasarkan pada struktur data ini [11].

2.1.11.1 Bentuk JSON

Berikut ini adalah penjelasan mengenai bentuk JSON, adapun penjelasannya adalah sebagai berikut ;

1. Objek

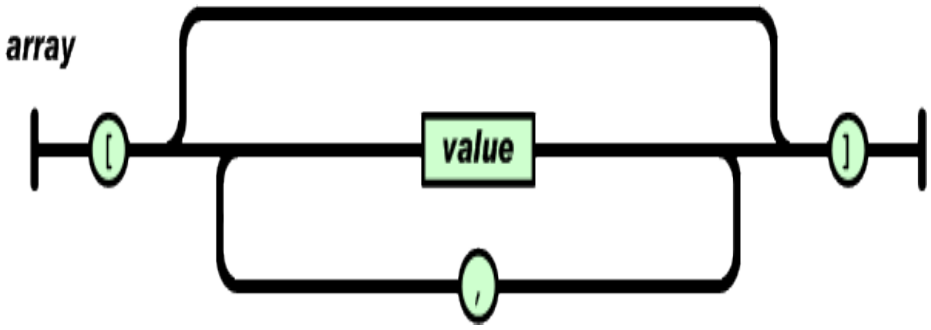
   Objek adalah sepasang nama/nilai yang tidak terurutkan. Objek dimulai dengan { (kurung kurawal buka) dan diakhiri dengan } (kurung kurawal tutup). Setiap nama diikuti dengan : (titik dua) dan setiap pasangan nama/nilai dipisahkan oleh , (koma).



Gambar 2.2 Objek JSON

2. Larik

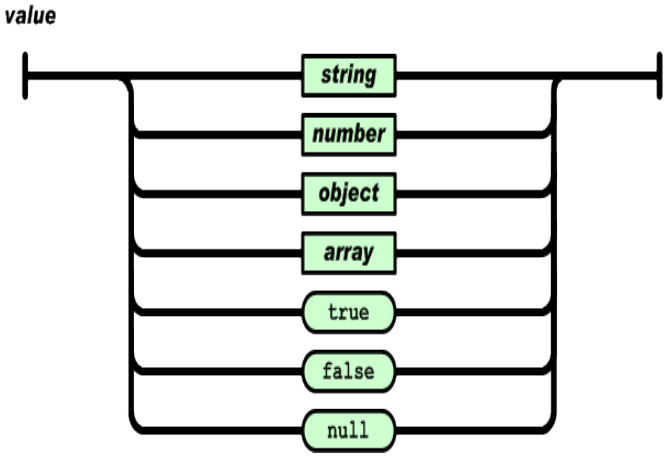
Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh, (koma).



Gambar 2.3 Larik JSON

3. Nilai

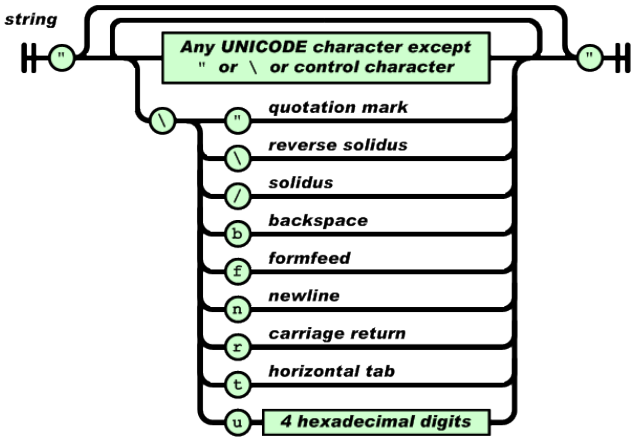
Nilai (*value*) dapat berupa sebuah string dalam tanda kutip ganda, atau angka, atau true atau false atau *null*, atau sebuah objek atau sebuah larik. Struktur-struktur tersebut dapat disusun bertingkat.



Gambar 2.4 Nilai Objek

4. *String*

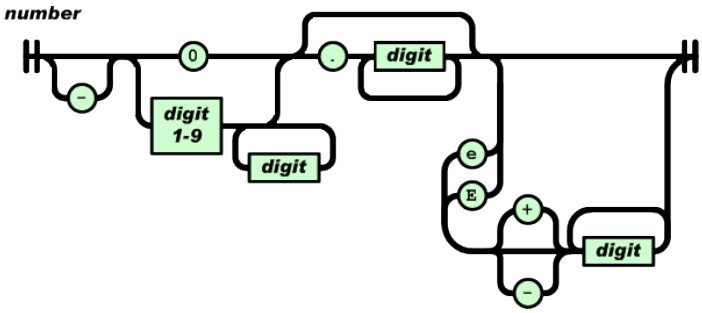
String adalah kumpulan dari nol atau lebih karakter *unicode*, yang dibungkus dengan tanda kutip ganda. Di dalam *string* dapat digunakan *backslash escapes* "\" untuk membentuk karakter khusus. Sebuah karakter mewakili karakter tunggal pada *string*. *String* sangat mirip dengan *string* C atau Java.



Gambar 2.5 String JSON

5. Angka

Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan [11].



Gambar 2.6 Angka JSON

2.1.12 *Payment Gateway*

Internet E-commerce Payment Gateway adalah komponen infrastruktur penting untuk memastikan transaksi berlangsung tanpa hambatan dan terlindungi total melalui jaringan internet. *Payment Gateway* adalah sebuah akses poin ke dalam jaringan perbankan nasional. Semua transaksi secara *online* harus melalui *Payment Gateway* untuk diproses. Secara teorinya, *payment gateway* bertindak sebagai jembatan antara pemilik website dan institusi keuangan yang melakukan proses transaksi. *Payment gateway* membuktikan dan mengarahkan detail pembayaran didalam lingkungan teraman antara berbagai pihak dan bank yang terkait. Fungsi *payment gateway* pada dasarnya sebagai saluran yang terenkripsi, yang secara aman mengirimkan detail transaksi dari pembeli yang menggunakan *personal computer* ke bank untuk disetujui [12].

2.1.13 API Midtrans

Veritrans atau Midtrans adalah *payment gateway* yang mendukung *E-Commerce* (*online shop*) di Indonesia untuk menerima pembayaran dari pelanggan dengan mudah. Dengan Veritrans atau Midtrans , anda dapat membuat proses pembayaran semakin mudah di website maupun *mobile app*. Veritrans atau Midtrasn merupakan *tools* jenis API *Key* yang dapat digunakan dalam mengintegrasikan pada website *e-commerce* maupun *mobile apps* sehingga website ataupun perangkat *mobile* yang terintegrasi dengan API *Key* Veritrans atau Midtrans dapat melakukan pembayaran secara *online.* API *Key* Veritrans atau Midtrans dapat diunduh secara gratis pada situs resminya veritrans.co.id. atau midtrans.com. Namun veritrans atau midtrans juga memberikan beberapa layanan yang disediakan berbayar karena memberikan fitur premium dalam kemudahan melakukan transaksi secara online [12].

2.1.14 *Object Oriented Programming* (OOP)

Pemrograman Berorientasi Obyek (*Object Oriented Programming* – OOP) adalah *programming* paradigma yang menggunakan *object* dan interaksinya untuk merancang aplikasi dan program komputer. OOP tidak banyak digunakan sebelum awal tahun 1990an. Tapi sekarang menjadi sesuatu yang sudah lumrah digunakan. Bahasa-bahasa pemrograman seperti keluarga dotNet dari Microsoft (Visual Basic.Net, Visual C#, dan Visual J), Borland Delphi, Java, Phyton, PHP versi 5 keatas, C++ dan banyak lainnya merupakan bahasa pemrograman yang mendukung konsep OOP.

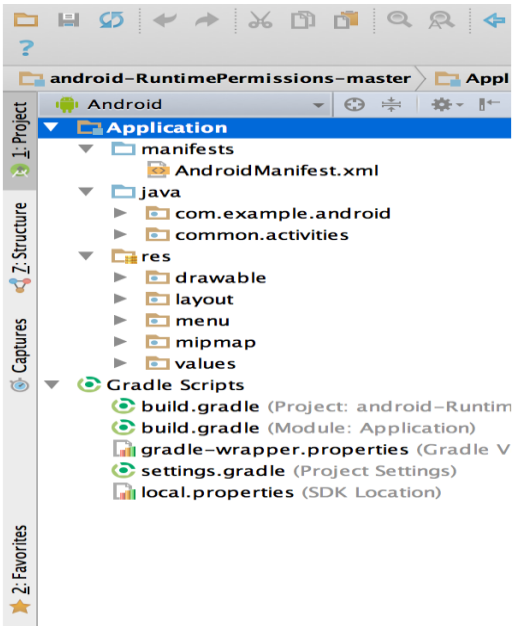
Object Oriented Programming (OOP) dalam Bahasa Indonesia dikenal dengan pemrograman berorientasi *object* (PBO), sementara pada OOP penyelesaiian masalah dilakukan dengan cara memetakan sub bagian program kedalam beberapa class. OOP adalah solusi untuk program besar karna OOP mudah dianalisis pada tiap class yang bersesuaian sehingga jika ada suatu perubahan pada suatu *class*, maka *class* lainnya tidak terganggu [13].

2.1.15 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* [14].



Gambar 2.7 Tampilan *File* Proyek Android Studio

2.1.16 *Unified Modeling Language*

UML adalah bahasa yang digunakan untuk mem-visualisasikan, mendefinisikan, membangun dan membuat dokumen dari arsitektur perangkat lunak. UML dapat digunakan pada semua proses melalui metodologi pengembangan perangkat lunak dan melakukan implementasinya pada teknologi yang berbeda. UML dapat digunakan untuk menggambarkan proses bisnis dengan *actor* dan *use case*, menggambarkan interaksi dengan *interaction* diagram, menggambarkan struktur statis dari sistem yang dibangun dengan menggunakan *class* diagram dan beberapa diagram lainnya yang dapat membantu menggambarkan pengembangan sistem secara *object*

oriented. *Object-oriented* dapat dengan mudah dipahami dengan bantuan UML, sehingga pada tim *development* dapat saling mengerti dengan mudah dan cepat; dan hal ini juga berdampak pada proses analisa yang dilakukan dan proses pengerjaan pengembang perangkat lunak tersebut [15].

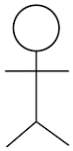
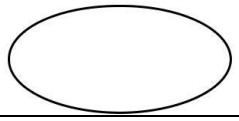
Karena aplikasi yang akan dibangun ini beorientasi obyek maka untuk perancangan konsep pembangunan aplikasi ini menggunakan pendekatan sistem *Object Oriented* yang Dijelaskan dengan UML. Didalam UML memiliki beberapa jenis diagam diantaranya adalah sebagai berikut: *Use Case Diagram, Activity Diagram, Class Diagram, Sequence Diagram.*

2.1.16.1 *Use Case Diagram*

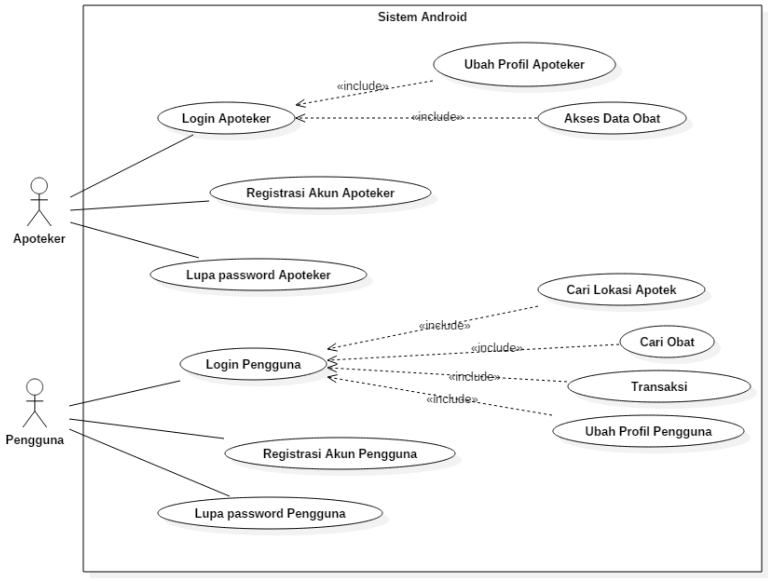
Use case diagram merupakan gambaran fungsionalitas dari beberapa atau semua *actor, use case*, dan interaksi untuk memperkenalkan suatu sistem. *Use case diagram* tidak menjelaskan secara detail tentang penggunaan *use case*, tetapi hanya memberikan gambaran singkat tentang hubungan *use case, actor*, dan sistem. Didalam *use case* hanya diketahui fungsi-fungsi yang berkaitan dengan pembangunan sistem yang akan dibuat.

Sebuah *use case* dapat di *include* oleh lebih dari suatu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan menarik keluar fungsionalitas yang *common*. *Use case* juga dapat meng*extend use case* lain dengan *behavior*nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialis dari yang lain. *Use case diagram* memiliki beberapa *element* yang digunakan untuk menggambarkan rancangan dan penghubung antar komponen, baik *actor* dengan *use case* atau *use case* dengan *use case* yang lainnya. Berikut ini adalah penjelasan dari notasi *use case diagram* yang terdapat pada Tabel 2.2 Notasi *Use Case Diagram* berikut ini :

Tabel 2.2 Notasi *Use Case Diagram*

Bentuk	Nama	Keterangan
	Asosiasi	Hubungan dan ketergantungan setiap actor dengan use case yang dijalani
--- <include> --▯	Include	Digunakan untuk menambahkan use case dengan use case yang lain.
--- <extend> --▯	Extend	Digunakan untuk menyederhanakan satu use case yang digunakan dan harus dijalankan pada use yang lain.
	Actor	Berfungsi untuk mempresentasikan seseorang atau sesuatu (seperti perangkat, system lain) yang berinteraksi dengan system. Actor hanya berinteraksi dengan sistem. Actor hanya berinteraksi dengan use case tetapi tidak memiliki control atas use case
	Use Case	Gambaran fungsionalitas dari suatu sistem

Sedangkan untuk contoh *use case diagram* dijelaskan pada Gambar 2.8 Contoh *Use Case Diagram* berikut ini :



Gambar 2.8 Contoh *Use Case Diagram*

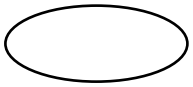
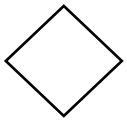
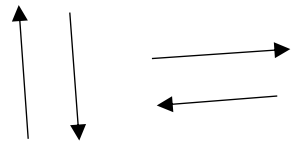
2.1.16.2 *Activity Diagram*

Activity Diagram atau Diagram Aktivitas digunakan untuk mendokumentasikan alur kerja pada sebuah sistem yang akan dibangun, yang dimulai dari pandangan *business level* hingga ke operasional *level*, adapun penjelasannya antara lain :

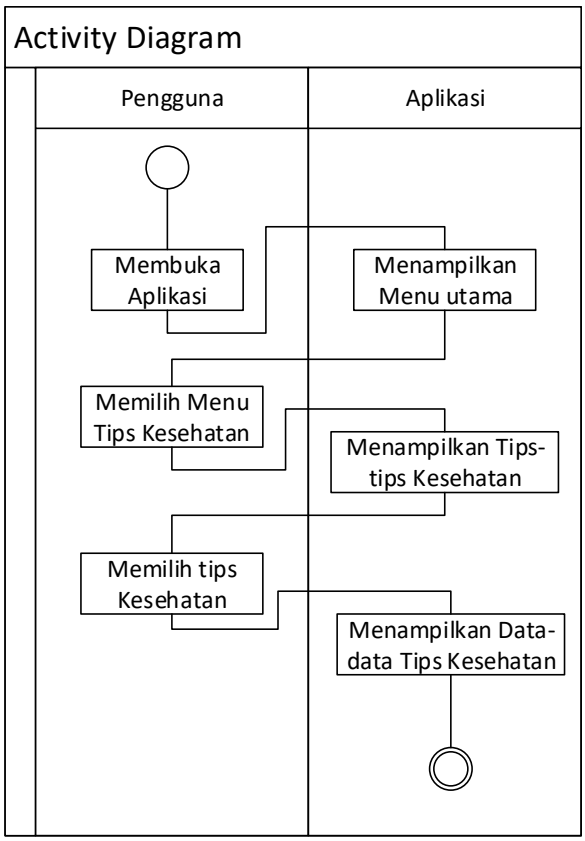
1. *Activity Diagram* merupakan cara lain untuk memodelkan aliran kejadian.
2. Berfungsi untuk menggambarkan *workflow* atau aliran kerja proses bisnis.
3. *Activity Diagram* menggambarkan berbagai aliran aktivitas dalam sebuah sistem yang sedang dirancang bagaimana masing aliran berawal, *decision* yang mungkin terjadi dan bagaimana alur berakhir.
4. *Activity Diagram* dapat menggambarkan proses *parallel* yang mungkin dapat terjadi pada beberapa eksekusi.

Berikut ini adalah penjelasan dari *activity diagram* yang terdapat pada Tabel 2.3 Notasi *Activity Diagram* berikut ini:

Tabel 2.3 Notasi *Activity Diagram*

Bentuk	Nama	Keterangan
	Initial Node	Memperlihatkan bagaimana masing-masing kelas antar muka saling berinteraksi satu sama lain.
	Action	State dari sistem yang mencerminkan eksekusi dari suatu aksi.
	Activity	Bagaimana objek dibentuk atau diawali
	Activity Final Node	Bagaimana objek dibentuk dan diakhiri
	Decision	Digunakan untuk menggambarkan suatu keputusan atau tindakan yang harus diambil pada kondisi tertentu.
	Line Connector	Digunakan untuk menghubungkan satu simbol dengan simbol lainnya.

Sedangkan untuk contoh *activity diagram* dijelaskan pada Gambar 2.9 Contoh *Activity Diagram* berikut ini :

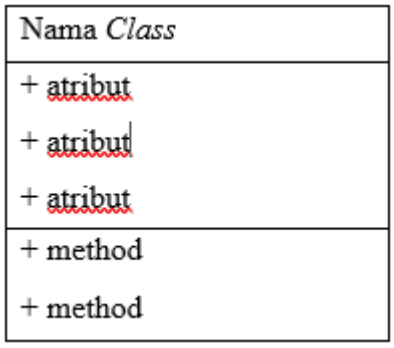


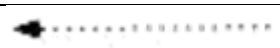
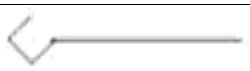
Gambar 2.9 Contoh *Activity Diagram*

2.1.16.3 *Class Diagram*

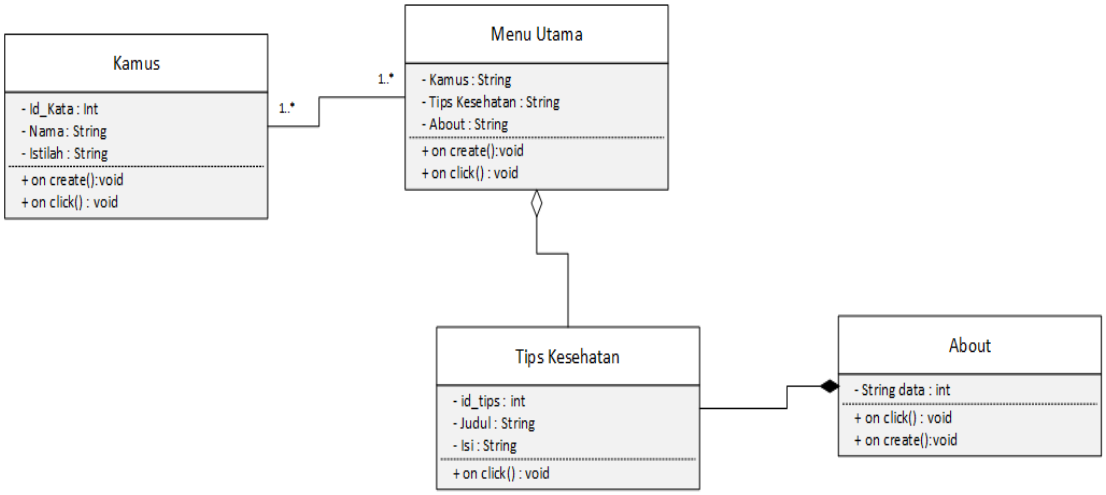
Class Diagram merupakan elemen terpenting dalam sistem berorientasi objek, kelas mendeskripsikan satu blok pembangun sistem. *Class Diagram* memiliki fitur yang memodelkan multiplisitas, ketampakan, penanda, *polymorphism*, dan karakter-karakter lainnya. UML menggunakan istilah fitur sebagai istilah umum yang meliputi *property* dan operasi sebuah kelas Terdapat empat bagian masing-masing bagan kelas, yaitu : nama, atribut, perilaku, dan keterangan kelas. Berikut merupakan penjelasan dari notasi class diagram yang terdapat pada Tabel 2.4 Notasi *Class Diagram* berikut ini :

Tabel 2.4 Notasi *Class Diagram*

Bentuk	Nama	Keterangan
	Class	Class adalah blok-blok pembangunan pada pemrograman berorientasi obyek. Sebuah class digambarkan sebagai sebuah kotak yang terbagi atas 3 bagian. Bagian atas adalah bagian nama dari suatu class. Bagian tengah mendefinisikan method-method dari sebuah class.
	Association	Sebuah asosiasi merupakan sebuah relationship paling umum antara 2 class dan dilambangkan oleh sebuah garis yang menghubungkan antara 2 class. Melambangkan Garis ini bisa tipe-tipe relationship dan juga dapat menampilkan hukum-hukum multiplisitas pada sebuah relationship.
	Composition	Jika sebuah class tidak bisa berdiri sendiri dan harus merupakan bagian dari class yang lain, maka class tersebut memiliki relasi Composition terhadap class tempat dia bergantung tersebut. Sebuah relationship composition digambarkan sebagai garis dengan ujung berbentuk jajaran genjang berisi/solid.

	Dependency	Dependency digunakan untuk menunjukkan operasi pada suatu class yang menggunakan class yang lain. Sebuah dependency dilambangkan sebagai sebuah panah bertitik-titik.
	Aggregation	Aggregation mengindikasikan keseluruhan bagian relationship dan biasanya disebut sebagai relasi.

Sedangkan untuk contoh *class diagram* dijelaskan pada Gambar 2.10 Contoh *Class Diagram* berikut ini :

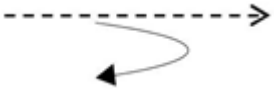


Gambar 2.10 Contoh *Class Diagram*

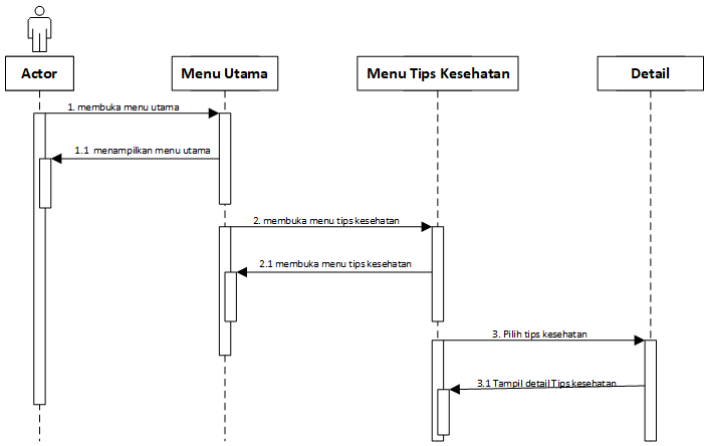
2.1.16.4 *Sequnce Diagram*

Sequence diagram digunakan untuk memberikan gambaran interaksi antar objek di dalam dan di sekitar sistem berupa pesan yang digambarkan terhadap waktu. *Sequence diagram* terdiri antar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). Dan *sequence diagram* ini bisa digunakan untuk membuat rangkaian langkah-langkah yang dilakukan sebagai respon dari event menjadi keluaran (output) tertentu. Dengan diawali dengan men-*trigger* aktivitas, proses dan perubahan yang terjadi secara internal dan *output* apa yang dihasilkan. Setiap objek termasuk aktor memiliki *lifeline* vertikal. Pesan digambarkan sebagai garis panah dari satu objek ke objek yang lainnya. Berikut merupakan komponen-komponen yang digunakan untuk perancangan *sequence diagram* yang terdapat pada Tabel 2.5 Notasi *Sequence Diagram* :

Tabel 2.5 Notasi *Sequence Diagram*

Bentuk	Nama	Keterangan
	Object	Menambah Objek baru pada diagram
	Object Message	Menggambarkan pesan antar dua objek
	Message to Self	Menggambarkan pesan yang menuju dirinya sendiri
	Return Message	Menggambarkan pengembalian dari pemanggilan prosedur
	Time Active	Menyatakan objek dalam keadaan aktif dan berinteraksi dengan pesan

Sedangkan untuk contoh *sequence diagram* dijelaskan pada Gambar 2.11 Contoh *Sequence Diagram* berikut ini :



Gambar 2.11 Contoh *Sequence Diagram*

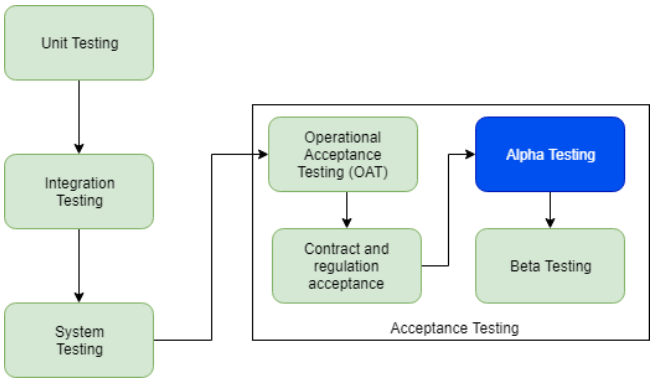
2.2 Metode Pengujian

Metode pengujian adalah suatu teknik menguji aplikasi, yang mempunyai mekanisme untuk menentukan data uji yang dapat menguji aplikasi secara lengkap dan mempunyai kemungkinan tinggi untuk menemukan kesalahan. *Software* atau aplikasi dapat diuji dengan dua cara, yaitu pengujian *alpha* dan pengujian *beta*.

2.2.1 Pengujian *Alpha*

Pengujian *Alpha* adalah salah satu strategi pengujian perangkat lunak yang paling umum digunakan dalam pengembangan perangkat lunak, hal ini khusus digunakan oleh organisasi pengembangan produk dengan tujuan agar sistem yang dikembangkan terhindar dari cacat atau kegagalan penggunaan.

Pengujian *alpha* berlangsung di situs pengembang oleh tim internal, sebelum rilis kepada pelanggan eksternal. Agar nantinya ketika pelanggan menggunakan sistem ini tidak kecewa karena masalah cacat atau kegagalan aplikasi. Pengujian ini dilakukan tanpa keterlibatan tim pengembangan [16].



Gambar 2.12 Diagram Alur Pengujian *Alpha*

Adapun 2 metode dalam pengujian *alpha* sebagai berikut :

1. White Box

Metode perancangan *test case* yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan *test case*. Dengan menggunakan metode *white box*, analis sistem akan dapat memperoleh keuntungan uji kasus sebagai berikut:

a. Menjamin seluruh *independent path* di dalam modul yang dikerjakan sekurang-kurangnya sekali
b. Mengerjakan seluruh keputusan *logical*
c. Mengerjakan seluruh *loop* yang sesuai dengan batasannya
d. Mengerjakan seluruh struktur data internal yang menjamin validitas

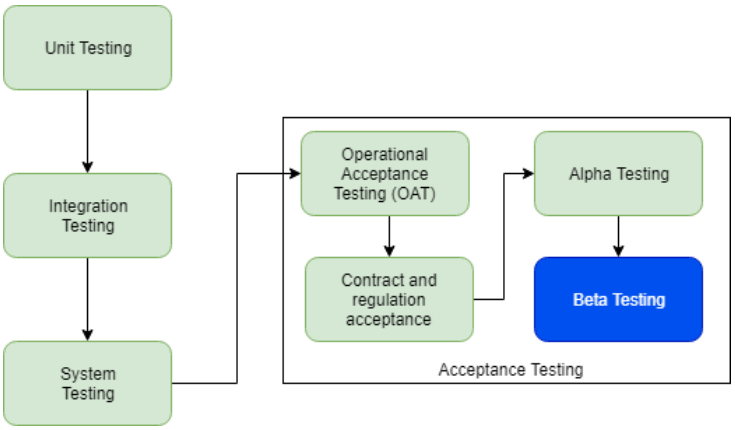
2. Black Box

Metode ujicoba *black box* memfokuskan pada keperluan fungsional dari *software*. Karna itu ujicoba *black box* memungkinkan pengembang *software* untuk membuat himpunan kondisi input yang akan melatih seluruh syarat-syarat fungsional suatu program. Ujicoba *black box* bukan merupakan alternatif dari ujicoba *white box*, tetapi merupakan pendekatan yang melengkapi untuk menemukan kesalahan lainnya, selain menggunakan metode *white box*.

2.2.2 Pengujian *Beta*

Pengujian *beta* juga dikenal sebagai pengujian pengguna berlangsung di lokasi pengguna akhir oleh pengguna akhir untuk memvalidasi kegunaan, fungsi, kompatibilitas, dan uji reliabilitas dari *software* yang dibuat. Aktifitas pengujian *beta* menambah nilai siklus hidup pengembangan perangkat lunak karena memungkinkan pelanggan sebenarnya kesempatan untuk memberikan masukan ke dalam desain, fungsi, dan kegunaan dari produk. Masukan ini tidak hanya penting untuk keberhasilan produk tetapi juga investasi ke produk masa depan ketika data yang dikumpulkan dikelola secara efektif.

Hal ini juga dikenal sebagai uji lapangan, ini terjadi di lokasi pelanggan. Ini mengirimkan sistem untuk pengguna yang menginstal dan menggunakannya di bawah kondisi kerja dunia nyata. Tes *beta* merupakan tahap kedua dari pengujian perangkat lunak di mana pengguna mencoba produk. Awalnya, tes *alpha* berarti tahap pertama pengujian dalam proses pengembangan perangkat lunak. Tahap pertama meliputi unit *testing*, pengujian komponen, dan pengujian sistem. Pengujian *beta* dapat dianggap "pengujian pra-rilis artinya sebelum produk tersebut dilempar ke pasaran maka harus dipastikan dari sisi pelanggan bahwa perangkat lunak tersebut terbebas dari cacat atau kegagalan [17].



Gambar 2.13 Diagram Alur Pengujian *Beta*