

2. TINJAUAN PUSTAKA

Prototipe Sistem Lacak Kendaraan ini adalah sebuah prototipe berupa alat dan aplikasi yang terhubung dengan web server untuk memudahkan pemilik kendaraan untuk mengetahui atau melacak posisi kendaraan secara digital. Prototipe ini memanfaatkan mikrokontroler dan aplikasi *mobile* atau bergerak. Pada tahap ini akan dijelaskan teori mengenai pengertian informasi, pengertian kendaraan, pengertian aplikasi *mobile* atau bergerak, pengertian GPS (*Global Positioning System*), mikrokontroler Arduino, modul SIM808, SDLC, Ionic dan komponennya, sistem operasi Android untuk menjalankannya, API, JSON, pemodelan diagram dan struktur navigasi.

2.1 Pengertian Informasi

Informasi adalah sekumpulan fakta-fakta yang telah diolah menjadi bentuk data, sehingga dapat menjadi lebih berguna dan dapat digunakan oleh siapa saja yang membutuhkan data-data tersebut sebagai pengetahuan ataupun dapat digunakan dalam pengambilan keputusan.

Informasi menyandang arti manfaat, apabila bisa memanfaatkan informasi mengandung makna usaha, untuk mendapatkannya, memahaminya, menggunakannya, menyebarkannya, menyimpannya dan memadukannya dengan informasi lain menjadi suatu bentuk informasi baru (Rochim,2002).

Informasi adalah data yang telah diolah menjadi sebuah bentuk yang berarti bagi penerimanya dan bermanfaat dalam pengambilan keputusan saat ini atau mendatang (Fatah,2007). Informasi adalah data yang telah diolah menjadi bentuk yang berguna bagi penerimanya dan nyata, berupa nilai yang dapat dipahami di dalam keputusan sekarang maupun masa depan (Wahyono,2004).

Pada informasi yang terpenting adalah makna, pengertian makna merupakan hal yang sangat penting. Karena berdasarkan maknalah si penerima dapat memahami informasi tersebut dan secara lebih jauh dapat menggunakannya untuk menarik suatu kesimpulan atau bahkan mengambil keputusan dari sebuah informasi (Kadir,2003).

2.2 Pengertian Kendaraan

Kendaraan adalah satu alat yang dapat bergerak di jalan, terdiri dari kendaraan bermotor atau kendaraan tidak bermotor. (Pasal 1 Angka 6 UU Nomor 14 Tahun 1992 Tentang Lalu Lintas Dan Angkutan Jalan). Kendaraan adalah suatu sarana angkut di jalan yang terdiri atas Kendaraan Bermotor dan Kendaraan Tidak Bermotor. (Pasal 1 Angka 7 UU Nomor 22 Tahun 2009 Tentang Lalu Lintas Dan Angkutan Jalan). Kendaraan menurut penggeraknya, dibagi menjadi 2 yaitu:

1. Kendaraan Bermotor

Kendaraan Bermotor adalah semua kendaraan beroda beserta gandengannya yang digunakan di semua jenis jalan darat, dan digerakkan oleh peralatan teknik berupa motor atau peralatan lainnya yang berfungsi untuk mengubah suatu sumber daya energi tertentu menjadi tenaga gerak kendaraan bermotor yang bersangkutan, termasuk alat-alat berat dan alat-alat besar yang dalam operasinya menggunakan roda dan motor dan tidak melekat secara permanen serta kendaraan bermotor yang dioperasikan di air. (Pasal 1 Angka 13 UU Nomor 28 Tahun 2009 Tentang Pajak Daerah Dan Retribusi Daerah).

2. Kendaraan Tidak Bermotor

Kendaraan Tidak Bermotor adalah setiap Kendaraan yang digerakkan oleh tenaga manusia dan/atau hewan. (Pasal 1 Angka 9 UU Nomor 22 Tahun 2009 Tentang Lalu Lintas Dan Angkutan Jalan).

2.3 Aplikasi *Mobile*

Aplikasi bergerak biasa disebut dengan aplikasi *mobile*, sehingga diperoleh pengertian bahwa aplikasi bergerak merupakan aplikasi yang dapat dijalankan walaupun pengguna berpindah. Aplikasi bergerak saat ini sangat dibutuhkan karena alat-alat telekomunikasi yang tersebar di seluruh dunia, membutuhkan aplikasi-aplikasi yang dapat mempermudah pekerjaan penggunanya dimanapun dan kapanpun terutama dalam hal informasi.

Pemrograman aplikasi bergerak tidak banyak berbeda dengan pemrograman konvensional pada *Personal Computer* (PC). Aspek karakteristik dari suatu

perangkat sangat mempengaruhi arsitektur dan implementasi dari aplikasi tersebut. Aplikasi bergerak masih memiliki keterbatasan dibandingkan PC (Darytomo,2007).

Pada aplikasi bergerak sudah pasti di jalankan pada perangkat bergerak. Perangkat bergerak memiliki perbedaan dengan perangkat konvensional seperti PC dalam hal ukuran, *design* dan *layout* dengan perangkat desktop, sebagai berikut :

1. Ukuran yang kecil

Perangkat bergerak memiliki ukuran yang kecil. Konsumen menginginkan perangkat yang terkecil untuk kenyamanan dan mobilitas mereka.

2. Memory yang kecil

Perangkat bergerak juga memiliki *memory* yang kecil, yaitu *primary* (RAM) dan *secondary* (*disk*). Pembatasan ini adalah salah satu faktor yang mempengaruhi penulisan program untuk berbagai jenis dari perangkat ini. Dengan pembatasan jumlah dari *memory*, pertimbangan-pertimbangan khusus harus diambil untuk memelihara pemakaian dari sumber daya yang mahal ini.

3. Daya proses yang terbatas

Sistem bergerak tidaklah setangguh rekan mereka yaitu *desktop*. Ukuran, teknologi dan biaya adalah beberapa faktor yang mempengaruhi status dari sumber daya ini. Seperti *hardisk* dan RAM, anda dapat menemukan mereka dalam ukuran yang pas dengan sebuah kemasan kecil.

4. Mengkonsumsi daya yang rendah

Perangkat bergerak menghabiskan sedikit daya dibandingkan dengan mesin *desktop*. Perangkat ini harus menghemat daya karena mereka berjalan pada keadaan dimana daya yang disediakan dibatasi oleh baterai-baterai.

5. Kuat dan dapat diandalkan

Karena perangkat bergerak selalu dibawa kemana saja, mereka harus cukup kuat untuk menghadapi benturan-benturan, gerakan, dan sesekali tetesan-tetesan air.

6. Koneksi yang terbatas

Perangkat bergerak memiliki *bandwidth* rendah, beberapa dari mereka bahkan tidak tersambung. Kebanyakan dari mereka menggunakan koneksi *wireless*.

7. Masa hidup yang pendek

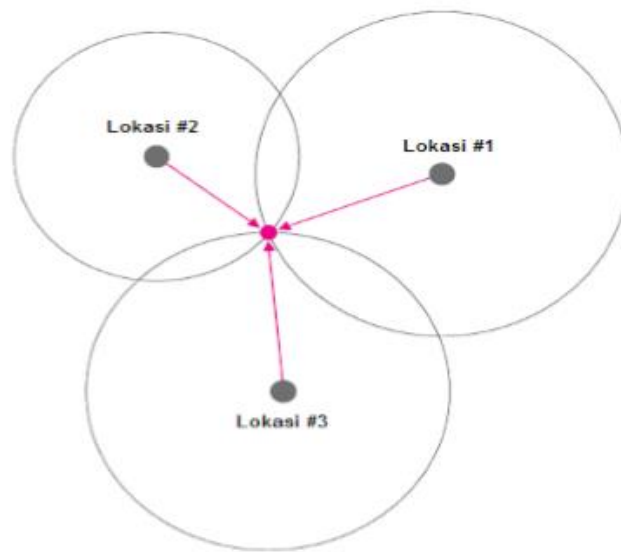
Perangkat-perangkat konsumen ini menyala dalam hitungan detik kebanyakan dari mereka selalu menyala. Coba ambil kasus sebuah *smartphone*, mereka booting dalam hitungan detik kebanyakan orang tidak mematikan *smartphone* mereka bahkan ketika malam hari. PDA akan menyala jika anda menekan tombol power mereka.

2.4 Pengertian GPS

Global Positioning System (GPS) merupakan sebuah alat atau sistem yang dapat digunakan untuk menginformasikan penggunaanya dimana dia berada (secara global) dipermukaan bumi yang berbasis satelit. Data dikirim dari satelit berupa sinyal radio dengan data digital.

Global Positioning System (GPS) adalah sistem navigasi yang berbasis satelit yang saling berhubungan yang berada di orbitnya. Satelit-satelit itu milik Departemen Pertahanan (*Departemen of Defense*) Amerika Serikat yang pertama kali diperkenalkan mulai tahun 1978 dan pada tahun 1994 sudah memakai 24 satelit. Untuk dapat mengetahui posisi seseorang maka diperlukan alat yang diberi nama GPS receiver yang berfungsi untuk menerima sinyal yang dikirim dari satelit GPS. Posisi diubah menjadi titik yang dikenal dengan nama Way-point nantinya akan berupa titik-titik koordinat lintang dan bujur dari posisi seseorang atau suatu lokasi kemudian di layar pada peta elektronik.

GPS adalah satu-satunya sistem satelit navigasi global untuk penentuan lokasi, kecepatan, arah, dan waktu yang telah beroperasi secara penuh di dunia saat ini (undergraduate thesis Wildan Habibi, ITS, Surabaya Januari : 2011). GPS menggunakan konstelasi 27 buah satelit yang mengorbit bumi, dimana sebuah GPS receiver menerima informasi dari tiga atau lebih satelit tersebut seperti terlihat dalam Gambar 2.1 dibawah, untuk menentukan posisi. GPS receiver harus berada dalam *line-of sight* (LoS) terhadap ketiga satelit tersebut untuk menentukan posisi, sehingga GPS hanya ideal untuk digunakan dalam *outdoor positioning*.



Gambar 2.1 Trilaterasi Dalam Global Positioning System (GPS)

(Sumber: Jurnal Andi Sunyoto, STMIK AMIKOM Jogjakarta, 2013:1)

2.4.1 Cara Kerja Global Positioning System (GPS)

Setiap daerah di atas permukaan bumi ini minimal terjangkau oleh 3-4 satelit. Pada prakteknya, setiap GPS terbaru bisa menerima sampai dengan 12 chanel satelit sekaligus. Kondisi langit yang cerah dan bebas dari halangan membuat GPS dapat dengan mudah menangkap sinyal yang dikirimkan oleh satelit. Semakin banyak satelit yang diterima oleh GPS, maka akurasi yang diberikan juga akan semakin tinggi.

Cara kerja GPS secara sederhana ada 5 langkah, yaitu :

1. Memakai perhitungan “*triangulation*” dari satelit.
2. Untuk perhitungan “*triangulation*”, GPS mengukur jarak menggunakan travel time sinyal radio.
3. Untuk mengukur travel time, GPS memerlukan memerlukan akurasi waktu yang tinggi.
4. Untuk perhitungan jarak, kita harus tahu dengan pasti posisi satelit dan ketinggian pada orbitnya.
5. Terakhir harus mengoreksi delay sinyal waktu perjalanan di atmosfer sampai diterima *reciever*.

2.4.2 Cara Satelit Menentukan Posisi Lokasi

Sinyal yang dikirimkan oleh satelit ke GPS akan digunakan untuk menghitung waktu perjalanan (*travel time*). Waktu perjalanan ini sering juga disebut sebagai *Time of Arrival* (TOA). Sesuai dengan prinsip fisika, bahwa untuk mengukur jarak dapat diperoleh dari waktu dikalikan dengan cepat rambat sinyal. Maka, jarak antara satelit dengan GPS juga dapat diperoleh dari prinsip fisika tersebut. Setiap sinyal yang dikirimkan oleh satelit akan juga berisi informasi yang sangat detail, seperti orbit satelit, waktu, dan hambatan di atmosfer. Satelit menggunakan jam atom yang merupakan satuan waktu paling presisi.

Untuk dapat menentukan posisi dari sebuah GPS secara dua dimensi (jarak), dibutuhkan minimal tiga buah satelit. Empat buah satelit akan dibutuhkan agar didapatkan lokasi ketinggian (secara tiga dimensi). Setiap satelit akan memancarkan sinyal yang akan diterima oleh GPS receiver. Sinyal ini akan dibutuhkan untuk menghitung jarak dari masing-masing satelit ke GPS. Dari jarak tersebut, akan diperoleh jari-jari lingkaran jangkauan setiap satelit. Lewat perhitungan matematika yang cukup rumit, *interseksi* (perpotongan) setiap lingkaran jangkauan satelit tadi akan dapat digunakan untuk menentukan lokasi dari GPS di permukaan bumi. (Jurnal Andi Sunyoto, STMIK AMIKOM Jogjakarta, 2013:1)

Dengan mengunci sinyal yang ditransmit oleh satelit minimum 3 sinyal dari satelit yang berbeda, pesawat penerima GPS dapat menghitung posisi tetap sebuah titik yaitu posisi lintang dan bujur bumi (Latitude & Longitude).

Latitude adalah garis yang melintang di antara kutub utara dan kutub selatan, yang menghubungkan antara sisi timur dan barat bagian bumi. Garis ini memiliki posisi membentangi bumi, sama halnya seperti garis equator (khatulistiwa), tetapi dengan kondisi nilai tertentu. Garis lintang inilah yang dijadikan ukuran dalam mengukur sisi utara-selatan koordinat suatu titik di belahan bumi. Sedangkan longitude adalah garis membujur yang menghubungkan antara sisi utara dan sisi selatan bumi (kutub). Garis bujur ini digunakan untuk mengukur sisi barat-timur koordinat suatu titik di belahan bumi. Sama seperti equator pada

latitude yang berada ditengah dan memiliki nilai 0 (nol) derajat, pada longitude, garis tengah yang bernilai 0 (nol) derajat disebut garis prime meridian (garis bujur). Sedangkan garis yang berada paling kiri memiliki nilai -90 derajat, dan yang paling kanan memiliki nilai 90 derajat.

2.4.3 Manfaat Penggunaan Global Positioning System (GPS)

Dengan teknologi GPS dapat digunakan untuk beberapa keperluan sesuai dengan tujuannya. GPS dapat digunakan oleh peneliti, olahragawan, petani, tentara, pilot, petualang, pendaki, pengantar barang, pelaut, kurir, penebang pohon, pemadam kebakaran dan orang dengan berbagai kepentingan untuk meningkatkan produktivitas, keamanan, dan untuk kemudahan. Dari beberapa pemakaiaa di atas dikategorikan menjadi:

1. Lokasi

Digunakan untuk menentukan dimana lokasi suatu titik dipermukaan bumi berada.

2. Navigasi

Membantu mencari lokasi suatu titik di bumi

3. Tracking

Membantu untuk memonitoring pergerakan obyek dan membantu memetakan posisi tertentu, dan perhitungan jaringan terdekat

4. Timing

Dapat dijadikan dasar penentuan jam seluruh dunia, karena memakai jam atom yang jauh lebih presisi di banding dengan jam biasa.

2.4.4 Kelemahan GPS

Penentuan posisi GPS pun tak luput dari faktor kesalahan, kesalahan tersebut bias timbul dari:

- Referensi waktu, untuk ketepatan minimal 4 sinyal satelit harus didapat.
- *Multipath*, GPS receiver tidak hanya menerima sinyal dari satelit tapi bias saja dari pantulan dari perangkat lain di daratan dan sebagainya.

Bertambahnya waktu perjalanan sinyal untuk sampai ke penerima GPS tersebut dapat mempengaruhi ketelitian pengukuran.

- Jumlah satelit yang tampak. Semakin tinggi sinyal satelit GPS, maka semakin baik ketelitiannya. Bangunan, tanah lapang, *interferensi* elektronik dapat menghalangi penerimaan sinyal, sehingga dapat menyebabkan kesalahan penentuan posisi bahkan memungkinkan tidak adanya pendeteksian posisi sama sekali. Oleh karena itu, GPS tidak dapat digunakan di dalam bangunan atau di daerah urban.

2.5 Mikrokontroler

Mikrokontroler adalah sebuah sistem komputer fungsional dalam sebuah chip. Di dalamnya terkandung sebuah inti prosesor, memori (sejumlah kecil, RAM, memori program, atau keduanya), dan perlengkapan input output. Dengan kata lain, mikrokontroler adalah suatu alat elektronika digital yang mempunyai masukan dan keluaran serta kendali dengan program yang bisa ditulis dan dihapus dengan cara khusus, cara kerja mikrokontroler sebenarnya membaca dan menulis data.

Mikrokontroler merupakan komputer didalam chip yang digunakan untuk mengontrol peralatan elektronik, yang menekankan efisiensi dan efektifitas biaya. Secara harfiahnya bisa disebut “pengendali kecil” dimana sebuah system elektronik yang sebelumnya banyak memerlukan komponen-komponen pendukung seperti IC TTL dan CMOS dapat direduksi/diperkecil dan akhirnya terpusat serta dikendalikan oleh mikrokontroler ini.

Mikrokonktroler digunakan dalam produk dan alat yang dikendalikan secara otomatis, seperti sistem kontrol mesin, remote controls, mesin kantor, peralatan rumah tangga, alat berat, dan mainan. Dengan mengurangi ukuran, biaya, dan konsumsi tenaga dibandingkan dengan mendesain menggunakan mikroprosesor memori, dan alat input output yang terpisah, kehadiran mikrokontroler membuat kontrol elektrik untuk berbagai proses menjadi lebih ekonomis. Dengan penggunaan mikrokontroler ini maka :

- Sistem elektronik akan menjadi lebih ringkas
- Rancang bangun sistem elektronik akan lebih cepat karena sebagian besar

dari sistem adalah perangkat lunak yang mudah dimodifikasi

- Pencarian gangguan lebih mudah ditelusuri karena sistemnya yang kompak

2.5.1 ATmega328

ATmega328 adalah mikrokontroler keluaran dari atmel yang mempunyai arsitektur RISC (*Reduce Instruction Set Computer*) yang dimana setiap proses eksekusi data lebih cepat dari pada arsitektur CISC (*Completed Instruction Set Computer*). Mikrokontroler ini memiliki beberapa fitur antara lain :

- 130 macam instruksi yang hampir semuanya dieksekusi dalam satu siklus clock.
- 32 x 8-bit register serba guna.
- Kecepatan mencapai 16 MIPS dengan clock 16 MHz.
- 32 KB Flash memory dan pada arduino memiliki bootloader yang menggunakan 2 KB dari flash memori sebagai bootloader.
- Memiliki EEPROM (*Electrically Erasable Programmable Read Only Memory*) sebesar 1KB sebagai tempat penyimpanan data semi permanent karena EEPROM tetap dapat menyimpan data meskipun catu daya dimatikan.
- Memiliki SRAM (*Static Random Access Memory*) sebesar 2KB.
- Memiliki pin I/O digital sebanyak 14 pin 6 diantaranya PWM (*Pulse Width Modulation*) output.
- Master / Slave SPI Serial interface

2.6 Arduino Uno

Arduino Uno adalah sebuah board mikrokontroler yang berbasis ATmega328. Arduino memiliki 14 pin input/output yang mana 6 pin dapat digunakan sebagai output PWM, 6 analog input, *crystal osilator* 16 MHz, koneksi USB, *jack power*, kepala ICSP dan tombol reset. Arduino UNO memuat semua yang dibutuhkan untuk menunjang mikrokontroler, mudah menghubungkannya ke sebuah computer dengan sebuah kabel USB atau mensuplainya dengan sebuah adaptor AC ke DC atau menggunakan baterai untuk memulainya.



Gambar 2.2 Tampilan Bagian Atas dan Bawah Arduino Uno
(Sumber: <https://www.arduino.cc/en/Main/ArduinoBoardUno/>)

2.6.1 Daya (Power)

Arduino Uno dapat disuplai melalui koneksi USB atau dengan sebuah power suplai eksternal. Sumber daya dipilih secara otomatis.

Suplai eksternal (non-USB) dapat diperoleh dari sebuah adaptor AC ke DC atau battery. Adaptor dapat dihubungkan dengan mencolokkan sebuah center-positive plug yang panjangnya 2,1 mm ke power jack dari board. Kabel lead dari sebuah battery dapat dimasukkan dalam header/kepala pin Ground (Gnd) dan pin Vin dari konektor POWER.

Board atau papan Arduino Uno dapat beroperasi pada sebuah suplai eksternal 6 sampai 20 Volt. Jika disuplai dengan yang lebih kecil dari 7 V, kiranya pin 5 Volt mungkin mensuplai kecil dari 5 Volt dan board Arduino Uno bisa menjadi tidak stabil. Jika menggunakan suplai yang lebih dari besar 12 Volt, voltage regulator bisa kelebihan panas dan membahayakan board Arduino Uno. Range yang direkomendasikan adalah 7 sampai 12 Volt.

Pin-pin dayanya adalah sebagai berikut:

- VIN. Tegangan input ke Arduino board ketika board sedang menggunakan sumber suplai eksternal (seperti 5 Volt dari koneksi USB atau sumber tenaga lainnya yang diatur). Kita dapat menyuplai tegangan melalui pin ini, atau jika penyuplaian tegangan melalui power jack, aksesnya melalui pin ini.

- 5V. Pin output ini merupakan tegangan 5 Volt yang diatur dari regulator pada board. Board dapat disuplai dengan salah satu suplai dari DC power jack (7-12V), USB connector (5V), atau pin VIN dari board (7-12). Penyuplaian tegangan melalui pin 5V atau 3,3V membypass regulator, dan dapat membahayakan board. Hal itu tidak dianjurkan.
- 3V3. Sebuah suplai 3,3 Volt dihasilkan oleh regulator pada board. Arus maksimum yang dapat dilalui adalah 50 mA.
- GND. Pin ground.

Tabel 2.1 Spesifikasi Arduino Uno

(Sumber: <https://www.arduino.cc/en/Main/ArduinoBoardUno/>)

Mikrokontroler	ATmega328
Tegangan pengoperasian	5V
Tegangan input yang disarankan	7-12V
Batas tegangan input	6-20V
Jumlah pin I/O digital	14 (6 di antaranya menyediakan keluaran PWM)
Jumlah pin input analog	6
Arus DC tiap pin I/O	40 mA
Arus DC untuk pin 3.3V	50 mA
Memori Flash	32 KB (ATmega328), sekitar 0.5 KB digunakan oleh bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz

2.6.2 Memori

ATmega328 mempunyai 32 KB (dengan 0,5 KB digunakan untuk bootloader). ATmega 328 juga mempunyai 2 KB SRAM dan 1 KB EEPROM (yang dapat dibaca dan ditulis (RW/read and written) dengan EEPROM library).

2.6.3 Input dan Output

Setiap 14 pin digital pada Arduino Uno dapat digunakan sebagai input dan output, menggunakan fungsi pinMode(), digitalWrite(), dan digitalRead(). Fungsi-fungsi tersebut beroperasi di tegangan 5 Volt. Setiap pin dapat memberikan atau menerima suatu arus maksimum 40 mA dan mempunyai sebuah resistor pull-up (terputus secara default) 20-50 kOhm.

Selain itu, beberapa pin mempunyai fungsi-fungsi spesial:

- Serial: 0 (RX) dan 1 (TX). Digunakan untuk menerima (RX) dan memancarkan (TX) serial data TTL (*Transistor-Transistor Logic*). Kedua pin ini dihubungkan ke pin-pin yang sesuai dari *chip* Serial Atmega8U2 USB-ke-TTL.
- *External Interrupts*: 2 dan 3. Pin-pin ini dapat dikonfigurasi untuk dipicu sebuah interrupt (gangguan) pada sebuah nilai rendah, suatu kenaikan atau penurunan yang besar, atau suatu perubahan nilai. Lihat fungsi attachInterrupt() untuk lebih jelasnya.
- PWM: 3, 5, 6, 9, 10, dan 11. Memberikan 8-bit PWM output dengan fungsi analogWrite().
- SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Pin-pin ini mensupport komunikasi SPI menggunakan SPI library.
- LED: 13. Ada sebuah LED yang terpasang, terhubung ke pin digital 13. Ketika pin bernilai HIGH LED menyala, ketika pin bernilai LOW LED mati.

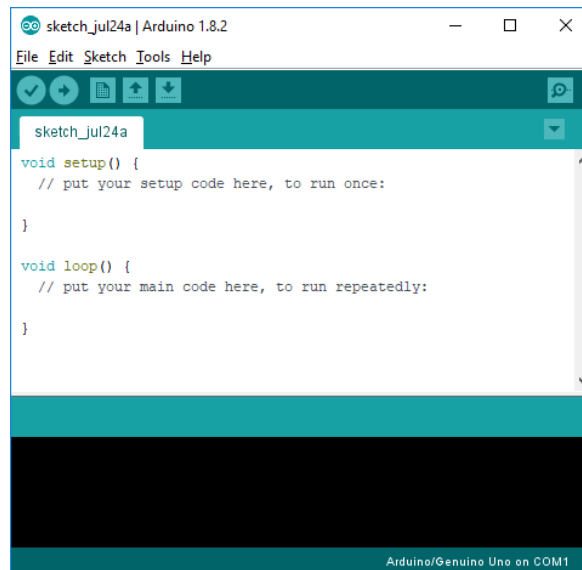
2.6.4 Komunikasi

Arduino Uno mempunyai sejumlah fasilitas untuk komunikasi dengan sebuah komputer, Arduino lainnya atau mikrokontroler lainnya. Atmega 328 menyediakan serial komunikasi UART TTL (5V), yang tersedia pada pin digital 0 (RX) dan 1 (TX). Software Arduino mencakup sebuah serial monitor yang memungkinkan data tekstual terkirim ke dan dari board Arduino. LED RX dan TX pada board akan menyala ketika data sedang ditransmit melalui *chip USB-to-serial* dan koneksi USB pada komputer (tapi tidak untuk komunikasi serial pada pin 0 dan 1). Sebuah *SoftwareSerial library* memungkinkan untuk komunikasi serial pada beberapa pin digital UNO.

2.6.5 Arduino IDE

IDE itu merupakan kependekan dari Integrated Development Environment, atau secara bahasa mudahnya merupakan lingkungan terintegrasi yang digunakan untuk melakukan pengembangan. Disebut sebagai lingkungan karena melalui software inilah Arduino dilakukan pemrograman untuk melakukan fungsi-fungsi yang dibenamkan melalui sintaks pemrograman. Arduino menggunakan bahasa pemrograman sendiri yang menyerupai bahasa C. Bahasa pemrograman Arduino (Sketch) sudah dilakukan perubahan untuk memudahkan pemula dalam melakukan pemrograman dari bahasa aslinya. Sebelum dijual ke pasaran, IC mikrokontroler Arduino telah ditanamkan suatu program bernama Bootlader yang berfungsi sebagai penengah antara compiler Arduino dengan mikrokontroler.

Arduino IDE dibuat dari bahasa pemrograman JAVA. Arduino IDE juga dilengkapi dengan library C/C++ yang biasa disebut Wiring yang membuat operasi input dan output menjadi lebih mudah. Arduino IDE ini dikembangkan dari software Processing yang dirombak menjadi Arduino IDE khusus untuk pemrograman dengan Arduino.



Gambar 2.3 Arduino IDE

2.7 Modul SIM808

Modul SIM808 adalah modul GSM / GPRS Quad-Band lengkap yang menggabungkan teknologi GPS untuk navigasi satelit dan bluetooth. Desain kompak yang terintegrasi GSM/GPRS, GPS dan Bluetooth dalam paket SMT secara signifikan akan menghemat waktu dan biaya bagi pelanggan untuk mengembangkan aplikasi GPS yang digunakan. Menampilkan antarmuka standar industri dan fungsi GPS, ini memungkinkan aset variabel dilacak secara mulus di lokasi manapun dan kapan saja dengan jangkauan sinyal.



Gambar 2.4 Modul SIM808

(Sumber: <http://simcom.ee/modules/gsm-gprs-gnss/sim808/>)

A. Fitur Umum

- Quad-band 850/900/1800/1900MHz
- GPRS multi-slot class 12/10
- GPRS mobile station class B
- Compliant to GSM phase 2/2+
Class 4 (2 W @ 850/900MHz)
Class 1 (1 W @ 1800/1900MHz)
- Bluetooth: compliant with 3.0+EDR
- FM: 76~109MHz worldwide bands with 50KHz tuning step
- Dimensions: 24.0*24.0*2.6mm
- Weight: 3.30g
- Control via AT commands (3GPP TS 27.007, 27.005 and SIMCOM enhanced AT Commands)
- Supply voltage range 3.4 ~ 4.4V
- Low power consumption
- Operation temperature: -40°C ~ 85°C

B. Spesifikasi GPS

- Receiver type
22 tracking /66 acquisition -channel
GPS L1 C/A code
- Sensitivity
Tracking: -165 dBm
Cold starts : -147 dBm
- Time-To-First-Fix
Cold starts: 30s (typ.)
Hot starts: 1s (typ.)
Warm starts: 28s (typ.)
- Accuracy
Horizontal position : <2.5m CEP

C. Spesifikasi GPRS

- GPRS class 12: max. 85.6 kbps (downlink/uplink)
- PBCCH support
- Coding schemes CS 1, 2, 3, 4
- PPP-stack
- CSD up to 14.4 kbps
- USSD

D. Spesifikasi SMS via GSM/GPRS

- Point to point MO and MT
- SMS cell broadcast

- Text and PDU mode

E. Interfaces

- 68 SMT pads including:
- Analog audio interface
- PCM interface (optional)
- SPI interface (optional)
- RTC backup
- Serial interface
- USB interface
- Interface to external SIM 3V/1.8V
- Keypad interface
- GPIO
- ADC
- GSM Antenna pad
- GPS Antenna pad
- Bluetooth Antenna pad

F. Fitur Software

- 0710 MUX protocol
- Embedded TCP/UDP protocol
- FTP/HTTP
- MMS
- POP3/SMTP
- DTMF
- Jamming Detection
- Audio Record
- SSL
- Bluetooth 3.0 (optional)
- TTS CN(optional)
- Embedded AT (optional)

G. Compatibility

- AT cellular command interface

2.8 Software Development Life Cycle

Software Development Life Cycle (SDLC) adalah proses mengembangkan atau mengubah suatu sistem perangkat lunak dengan menggunakan model-model dan metodologi yang digunakan orang untuk mengembangkan sistem-sistem perangkat lunak sebelumnya. Tahapan-tahapan yang ada pada SDLC secara global adalah sebagai berikut (Shalahuddin,2013):

1. Inisialisai (*initiation*)

Tahap ini biasanya ditandai dengan pembuatan proposal proyek *software*.

2. Pengembangan konsep sistem (*system concept development*)

Mendefinisikan lingkup konsep termasuk dokumen lingkup sistem, analisis manfaat biaya, manajemen rencana, dan pembelajaran kemudahan sistem.

3. Perencanaan (*planning*)

Mengembangkan rencana manajemen proyek dan dokumen perencanaan lainnya. Menyediakan dasar untuk mendapatkan untuk mendapatkan sumber daya (*resource*) yang dibutuhkan untuk memperoleh solusi.

4. Analisis Kebutuhan (*requirements analysis*)

Menganalisis kebutuhan pemakai sistem perangkat lunak, mengembangkan kebutuhan pengguna dan membuat dokumen kebutuhan fungsional.

5. Desain (*design*)

Mentransformasikan kebutuhan detail menjadi kebutuhan yang sudah lengkap, dokumen desain sistem fokus pada bagaimana dapat memenuhi fungsi-fungsi yang dibutuhkan.

6. Pengembangan (*development*)

Mengonversi desain ke sistem informasi yang lengkap termasuk bagaimana memperoleh dan melakukan instalasi lingkungan sistem yang dibutuhkan, membuat basis data dan mempersiapkan prosedur kasus pengujian, mempersiapkan berkas atau *file* pengujian, pengkodean pengkompilasian, memperbaiki dan membersihkan program, peninjauan pustaka.

7. Integrasi dan pengujian (*integration and test*)

Mendemonstrasikan sistem perangkat lunak bahwa telah memenuhi kebutuhan yang dispesifikasikan pada dokumen kebutuhan fungsional. Dengan diarahkan oleh staff penjamin kualitas (*quality assurance*) dan pengguna. Menghasilkan laporan analisis pengujian.

8. Implementasi (*implementation*)

Termasuk persiapan implementasi, implementasi perangkat lunak pada lingkungan produksi pengguna dan menjalankan resolusi dari permasalahan yang teridentifikasi dari fase integrasi dan pengujian.

9. Operasi dan pemeliharaan (*operation and maintenance*)

Mendeskripsikan pekerjaan untuk mengoperasikan dan memelihara sistem informasi pada lingkungan produksi pengguna, termasuk implementasi akhir dan masuk pada proses peninjauan.

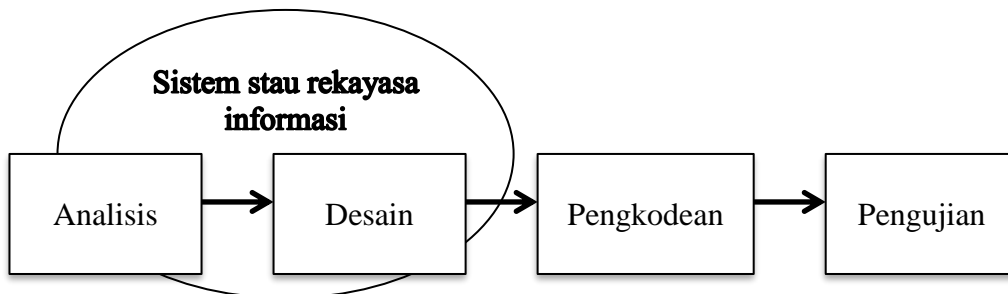
10. Disposisi (*disposition*)

Mendeskripsikan aktifitas akhir dari pengembangan sistem dan membangun data yang sebenarnya sesuai dengan aktifitas pengguna.

Analisis dan desain sering dikelompokkan sebagai proses sistem atau rekayasa informasi karena pada tahapan inilah informasi mengenai kebutuhan perangkat lunak banyak dikumpulkan dan diintegrasikan. SDLC memiliki beberapa model dalam penerapan tahapan prosesnya yang dapat digunakan. Setiap model memiliki kekurangan dan kelebihan dan pada pemilihan model SDLC, pemilihan model harus sesuai dengan karakter pengembang dan karakter pengguna. Pada pembangunan aplikasi informasi gempabumi menggunakan tahapan SDLC model *waterfall*. Penulis harus menganalisis apa saja yang dibutuhkan dalam pembangunan aplikasi, mendesain tampilan dan alur program, melakukan implementasi yaitu penerapan perancangan tampilan dan alur program ke dalam pembangunan aplikasi dan terakhir adalah uji coba aplikasi kedalam tahapan tahap yang telah dibuat.

Model SDLC air terjun (*waterfall*) sering juga disebut model sekuensial linier (*sequential linear*) atau alur hidup klasik (*classic life cycle*). Model *waterfall*

menyediakan alur hidup perangkat lunak secara sekuensial atau terurut dimulai dari analisis, desain, pengkodean, pengujian, dan tahap pendukung (*support*).



Gambar 2.5 Ilustrasi model *waterfall*
(Sumber : Shalahuddin, 2013)

Pada Gambar 2.5 menjelaskan tahapan pada model *waterfall*, terdapat empat tahapan dalam mengembangkan aplikasi (Shalahuddin,2013).

1. Analisis kebutuhan

Proses pengumpulan kebutuhan dilakukan secara intensif untuk mespesifikasikan kebtuhan perangkat lunak agar dapat dipahami perangkat lunak seperti apa yang dibutuhkan oleh pengguna. Spesifikasi kebtuhan perangkat lunak pada tahap ini perlu untuk didokumentasikan.

2. Desain

Desain perangkat lunak adalah proses multi langkah yang fokus pada desain pembuatan program perangkat lunak termasuk struktur data, arsitektur perangkat lunak, representais antarmuka, dan prosedur pengkodean. Tahap ini mentranslasi kebutuhan perangkat lunak dari tahap analisis kebutuhan ke representasi desain agar dapat diimplementasikan jadi program pada tahap selanjutnya. Desain perangkat lunak yang dihasilkan pada tahap ini juga perlu didokumentasikan.

3. Pembuatan kode program

Desain harus ditranslasikan ke dalam program perangkat lunak. Hasil dari tahap ini adalah program komputer sesuai dengan desain yang telah dibuat pada tahap desain.

4. Pengujian

Pengujian fokus pada perangkat lunak secara dari segi logik dan fungsional dan memastikan bahwa semua bagian sudah diuji. Hal ini dilakukan untuk meminimalisir kesalahan dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.

5. Pendukung (*support*) atau pemeliharaan (*maintenance*)

Tidak menutup kemungkinan sebuah perangkat lunak mengalami perubahan ketika sudah dikirim ke pengguna. Perubahan bisa terjadi karena adanya kesalahan yang muncul dan tidak terdeteksi saat pengujian atau perangkat lunak harus beradaptasi dengan lingkungan baru. Tahapan pendukung atau pemeliharaan dapat mengulangi proses pengembangan mulai dari analisis spesifikasi untuk perubahan perangkat lunak yang sudah ada, tapi tidak untuk membuat perangkat lunak baru.

2.9 Ionic Framework

Ionic merupakan kerangka kerja (*framework*) yang menyediakan sejumlah kontrol antarmuka pengguna yang digunakan di aplikasi bergerak. Ionic juga merupakan *framework* HTML5 yang masih baru dirilis pada tahun 2013 oleh Drifty Co. Ionic adalah teknologi yang dikembangkan untuk membangun aplikasi *mobile hybrid* dengan powerful, cepat, mudah dan juga memiliki tampilan yang menarik. Ionic digunakan untuk membangun aplikasi *mobile* menggunakan teknologi *website*, seperti HTML, CSS, dan JavaScript (Aditya,2015).

Pembangunan aplikasi bergerak yang menggunakan kerangka kerja Ionic setidaknya pembuat harus mengerti HTML, CSS dan JavaScript. Bahasa HTML dan CSS pada Ionic berfungsi untuk membangun tampilan-tampilan. Setiap tampilan memiliki aksi-aksi, aksi tersebut dapat terlaksana dengan adanya JavaScript didalam Ionic. Pembuat harus menguasai dasar-dasar JavaScript. Pada Ionic JavaScript terbungkus oleh kerangka kerja Angular. Pengembang *website* pasti sangat terbantu dengan hadir nya Ionic dalam pembangunan aplikasi bergerak.

Dengan menggunakan Ionic, para pengembang *website* tidak perlu belajar bahasa pemrograman Java, Objective C atau C# untuk membuat aplikasi bergerak.

Pengembang *website* hanya perlu menggunakan ilmu mereka dalam membangun aplikasi bergerak. Ionic hanyalah sebuah kerangka kerja, untuk membangun menjadi aplikasi bergerak berbasis android atau iOS Ionic membutuhkan Cordova. Dengan menggunakan Cordova, komponen Ionic yang berupa HTML, CSS dan JavaScript dapat disatukan dengan JDK dan SDK Android untuk menjadikan atau membuat *Android Package* (APK). Dalam membangun aplikasi menggunakan Ionic setidaknya programmer mengerti cara menggunakan beberapa hal dibawah ini :

1. HTML5, CSS, Javascript
2. AngularJS
3. Dasar-dasar nodeJS, *Node Package Manager* (NPM)
4. Cordova
5. *Command Line* (CMD, Bash, Terminal Linux dan Unix)
6. Cara membuat *Android Package* (APK)

2.9.1 Hyper Text Markup Language

Hyper Text Markup Language (HTML) adalah suatu format data yang digunakan untuk membuat dokumen teks pada komputer yang memungkinkan user saling mengirimkan informasi (*hypertext*) (Shalahuddin, 2008). Dokumen HTML harus disimpan dengan ekstensi .htm atau .html. HTML memiliki *tag-tag* yang telah didefinisikan untuk membuat halaman *website*. Penulisan *tag-tag* atau tanda-tanda html dapat menggunakan huruf besar atau huruf kecil, karena HTML tidak membedakan huruf besar dan huruf kecil memiliki maksud yang berbeda (*case sensitive*).

HTML adalah kerangka suatu halaman *website*, pembangunan website tidak akan bias apabila tidak menggunakan HTML. Bahasa HTML dapat diberikan atribut *style* untuk memperindah tampilan, akan tetapi sudah terdapat 28 bahasa pemrograman yang bekerja untuk memberikan *style* pada kerangka HTML yaitu *Cascading Style Sheets* (CSS).

HTML tidak hanya digunakan untuk membangun *website*. Dengan berkembang teknologi saat ini dan telah hadirnya Ionic membuat HTML dapat

digunakan untuk membangun aplikasi bergerak berbasis Android atau iOS. *Tag* HTML pada Ionic menggunakan HTML5 dan elemen HTML yang sudah diatur oleh kerangka kerja Ionic. Pada Ionic HTML yang sudah menjadi kerangkaan *input* adalah HTML5 *input types*. Penjelasan *tag* HTML dapat dilihat pada Table 2.2

Tabel 2.2 Contoh *tag* pada sebuah dokumen HTML (Sumber : Shalahuddin,2008)

Tanda	Keterangan
<html></html>	<i>Tag</i> dasar yang menandakan dokumen yang merupakan dokumen HTML.
<head>....</head>	<i>Tag</i> untuk mengisikan informasi tentang dokumen HTML.
<title>....</title>	<i>Tag</i> yang berada di dalam Tanda <i>head</i> untuk menuliskan judul <i>website</i> pada <i>caption website</i> .
<body>....</body>	<i>Tag</i> untuk mengisikan isi dokumen <i>website</i> yang ingin ditampilkan sebagai halaman <i>website</i> .
<p>....</p>	Menyatakan paragraf
<div>....</div>	Menyatakan divisi
<h1>....</h1> <h2>....</h2> <h3>....</h3>	Untuk mengatur judul, semakin besar angkanya maka ukuran <i>font</i> semakin kecil
<a>....	Untuk membuat tautan (<i>link</i>)
	Untuk menyajikan gambar
<script src=" " "></script>	<i>Tag</i> untuk menghubungkan dengan file JavaScript
<ink rel=" " ">....</link>	<i>Tag</i> untuk menghubungkan dengan file <i>cascading style sheets</i>

Pada pembangunan aplikasi bergerak menggunakan Ionic, diperlukan HTML dalam pembangunan tampilan aplikasi. Beberapa *tag* HTML dapat digunakan, akan tetapi Ionic telah memiliki fungsi pada *tag* HTML tertentu yang telah disesuaikan dengan pembangunan aplikasi. Perintah *tag* HTML dasar atau HTML5 masih bisa digunakan dalam pembangunan aplikasi bergerak menggunakan Ionic.

Pada pembangunan aplikasi bergerak, pengembang dapat melihat dokumentasi Ionic pada *website* resminya <http://ionicframework.com/>. Fungsi tanda HTML pada pembangunan Ionic terdapat pada HTML5 *input types*. Ionic telah menyediakan kerangka kerja dalam penyajian *input* pada aplikasi bergerak, pengembang dapat dengan mudah membuat *input* pada aplikasi yang sedang dibangun.

Tabel 2.3 Contoh *tag* dan kelas pada HTML5 *input types* (Sumber: Ionic,2016)

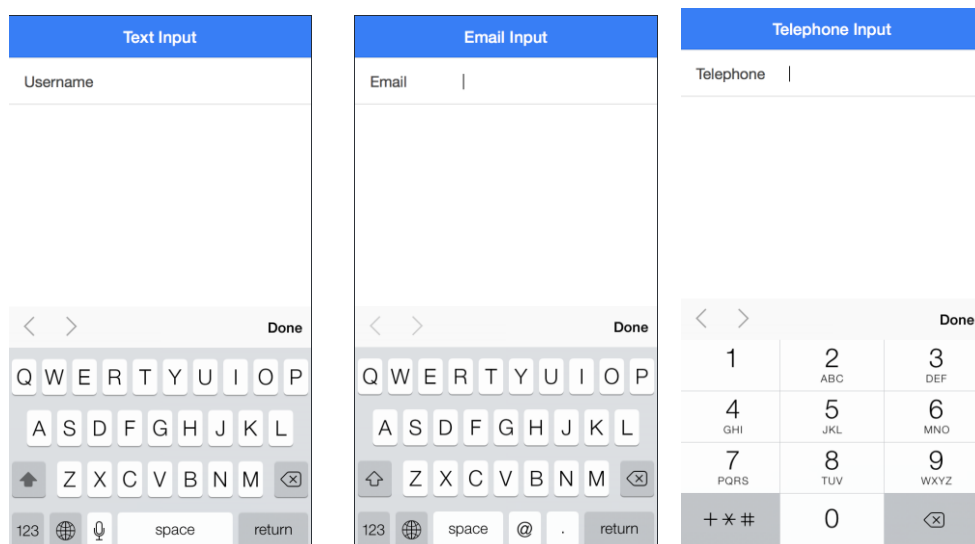
Tanda dan kelas	Keterangan
<code><div class="item item-input"></code> <code></div></code>	Kelas tersebut digunakan untuk <i>input</i>
<code><input type="text"></code>	<i>Tag</i> yang digunakan untuk <i>input</i> teks
<code><input type="email"></code>	<i>Tag</i> yang digunakan untuk <i>input email</i>
<code><input type="tel"></code>	Tanda yang digunakan untuk telepon
<code><i class="icon ion-search placeholder-icon"></i></code>	Kelas <i>icon</i> digunakan untuk memberikan <i>icon</i> pada tanda <i>i</i>
<code><input type="search" placeholder="Cari.." ></code>	<i>Tag</i> yang digunakan untuk pencarian
<code><input type="date"></code>	<i>Tag</i> yang digunakan untuk tanggal
<code><input type="month"></code>	<i>Tag</i> yang digunakan untuk bulan
<code><input type="password"></code>	<i>Tag</i> yang digunakan untuk kata kunci

Penggunaan *tag-tag* tersebut sudah disediakan dan dapat digunakan oleh *programmer* atau pengembang dalam membangun aplikasi bergerak menggunakan Ionic. Pembuat aplikasi tanpa harus memikirkan CSS dan JavaScript dari *tag* tersebut, Ionic sudah menyiapkan semuanya dan hal itulah yang bisa disebut dengan kerangka kerja. Perintah *tag* HTML dasar atau HTML5 masih bisa digunakan dalam pembangunan aplikasi bergerak menggunakan Ionic.

Pada dokumentasi HTML5 *input types* tidak terlalu banyak yang dapat digunakan dalam pembangunan aplikasi. Pengembang diharuskan menguasai bahasa *website*. Penguasaan bahasa *website* sangat bermanfaat pada pembangunan

aplikasi bergerak dengan menggunakan kerangka kerja Ionic. Ketika pengembang menginginkan tampilan yang berbeda dengan sudah disediakan oleh kerangka kerja Ionic, pengembang dapat membuat tampilannya sendiri tanpa harus mengikuti aturan kerangka kerja Ionic dengan cara menggunakan HTML dan CSS *native*. Beberapa bahasa *website* tersebut sangat dibutuhkan dalam pembuatan aplikasi bergerak Android menggunakan kerangka kerja Ionic.

Ionic akan terus berkembang bersamaan dengan Android. HTML5 *input types* akan selalu dikembangkan Ionic. Ionic sudah menyentuh versi 1.7 dan akan memiliki fungsi yang lebih menarik seiring berkembang versi Ionic. Terdapat beberapa contoh tampilan HTML5 *input types* pada Gambar 2.6.

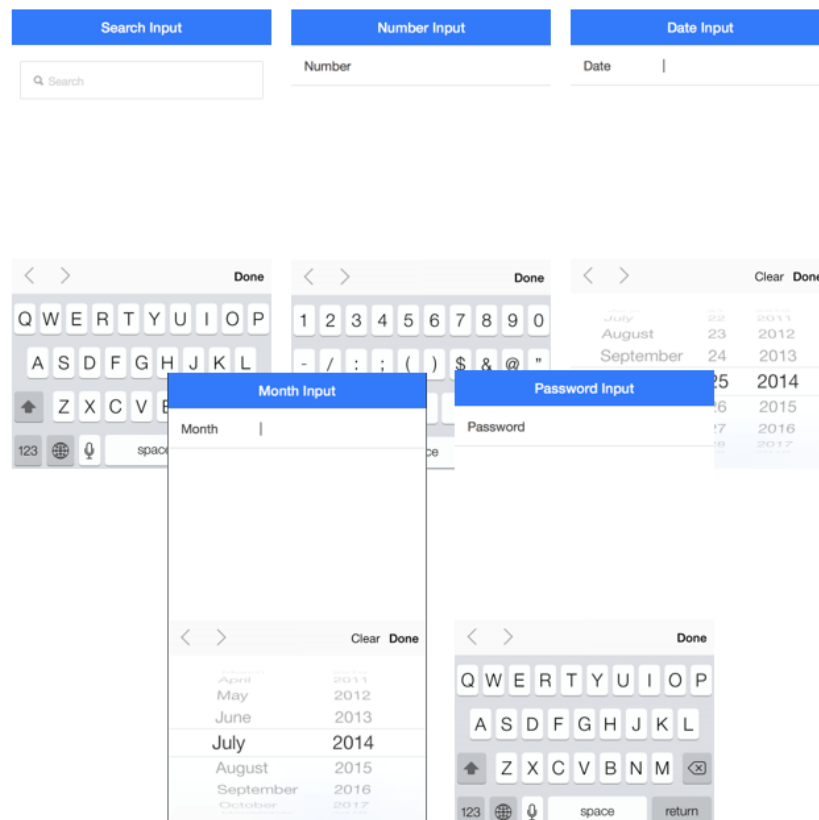


Gambar 2.6 Tampilan HTML5 *input types*

(Sumber: Ionic,2016)

Pada Gambar 2.6 terdapat tiga contoh tampilan HTML5 *input types*. Tampilan *username* biasa digunakan untuk *login* pada aplikasi bergerak atau untuk input registrasi akun pengguna. Tampilan *email* bisa digunakan sebagai validasi ketika membuat akun pengguna dan yang terakhir tampilan telpon. Terdapat tampilan *input* telpon pada pembangunan aplikasi, karena kerangka kerja Ionic ini digunakan untuk membangun aplikasi bergerak berbasis Android atau iOS. Maka dari itu tampilan HTML5 *input type* sangat mirip dengan *smartphone*. Pada pembangunan aplikasi terdapatlah fitur seperti ini dalam pembangunannya. Dengan

seiringnya berkembangnya teknologi, Ionic pasti akan membuat tampilan menjadi semenarik mungkin. Masih terdapat beberapa tampilan pada *HTML5 input types* yaitu kata kunci, tanggal, bulan dan pencarian. Tampilan lain nya dapat dilihat pada Gambar 2.7



Gambar 2.7 Tampilan *HTML5 input types* (lanjutan)
(Sumber: Ionic,2016)

Pada Gambar 2.7 lanjutan contoh tampilan *HTML5 input types* yang terakhir. Tampilan tersebut berguna dalam membuat masukan data pengguna apabila pada pembangunan aplikasi dibutuhkan. Pada kerangka kerja Ionic tidak hanya *input* saja untuk membuat tampilan aplikasi bergerak. Terdapat dokumentasi *CSS Component* pada subbab berikut nya yang akan menjelaskan komponen-komponen tampilan dalam membangun aplikasi bergerak menggunakan Ionic.

2.9.2 Cascading Style Sheet

Cascading Style Sheet (CSS) adalah suatu fasilitas untuk mempermudah pemeliharaan sebuah halaman *website*, dengan menggunakan CSS sebuah halaman *website* dapat diubah tampilannya tanpa harus mengubah dokumen HTML-nya (Shalahuddin,2008). Penulisan CSS yang dihubungkan dengan HTML dapat menghasilkan tampilan yang sangat indah sekali. Karena HTML tanpa CSS bagaikan bangunan tatidak memiliki cat, jendela, bingkai atau pun tembok.

Penggunaan CSS dalam dokumen HTML adalah untuk menciptakan suatu tampilan lebih menarik. File CSS dapat dipisahkan dengan kode-kode tanda HTML agar penulisan kode program lebih rapih dan teratur. Penulisan CSS secara eksternal dapat dilihat pada Gambar 2.8.

```
nama_style_atau_nama_tag_HTML {
    nama_properti : nilai_properti;
    .....
    nama_properti : nilai_properti;
}
```

Gambar 2.8 Penulisan CSS

(Sumber : Shalahuddin,2008)

Sebagai contoh, judul yang dibentuk oleh pasangan *tag* <h1>....</h1> bisa diatur agar diberi warna latar belakang dan merubah warna tulisan dengan menggunakan kode CSS. Penulisan contoh kode CSS terdapat pada Gambar 2.9.

```
h1 {
    background-color : black;
    color : white;
}
```

Gambar 2.9 Contoh kode CSS

(Sumber : Shalahuddin,2008)

Pada Gambar 2.9, h1 berposisi sebagai selektor *tag* pada HTML. Selektor *tag* mempunyai ciri-ciri berupa nama elemen HTML yang akan di format. Kode yang berada didalam {} setelah nama selektor berisi kumpulan pasangan

“nama_properti : nilai_properti;”. Pada contoh diatas terdapat dua properti, yaitu *background-color* dan *color*. Properti *background-color* digunakan untuk mengatur warna latar belakang. Nilai *black* berarti hitam. Properti *color* digunakan untuk menentukan warna teks. Nilai *white* berarti putih.

Selain selektor *tag* pada elemen HTML, terdapat selektor-selektor lain, diantaranya selektor id dan selektor kelas. Selektor id biasa digunakan untuk mengatur sebuah elemen yang mempunyai nilai id tertentu. Selektor id ditandai dengan tanda *kress* (#) dan diikuti dengan nama id. Penulisan kode selektor id terdapat pada Gambar 2.10.

```
#modal {
    height : 100px;
    background-color : lightgray;
}
```

Gambar 2.10 Contoh kode CSS selektor id

(Sumber : Shalahuddin,2008)

Pada contoh selektor id diatas digunakan untuk mengatur elemen yang ber-id “modal”. Properti *height* yang bernilai *100px* menyatakan bahwa tinggi elemen yang diatur adalah 100 *pixel*. Properti *background-color* yang bernilai *lightgray* menyatakan bahwa warna latar belakang berupa abu-abu muda.

Selektor kelas adalah selektor yang memformat elemen-elemen yang mempunyai nama kelas tertentu. Selektor kelas ditandai dengan titik (.) dan diikuti dengan nama kelas. Penulisan kode css dengan selektor kelas terdapat pada Gambar 2.7.

```
.item {
    border : solid 1px;
}
```

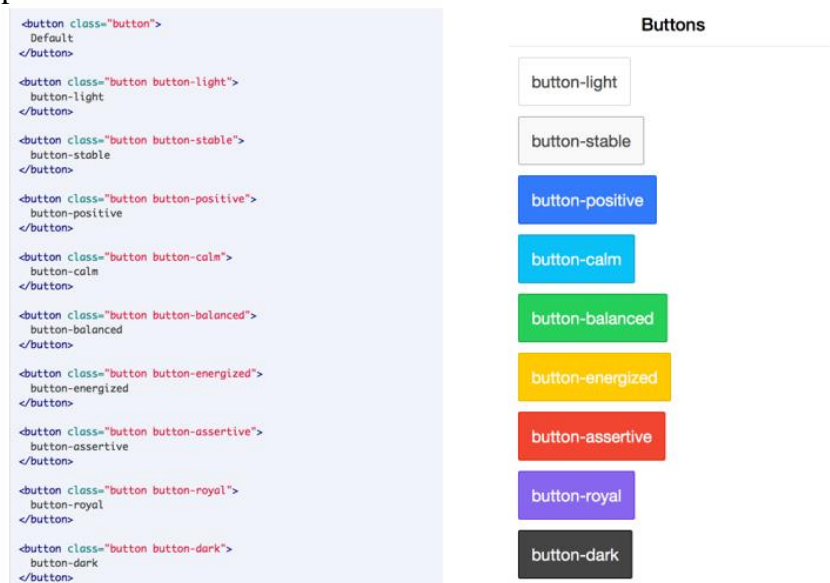
Gambar 2.11 Contoh kode CSS selektor kelas

(Sumber : Shalahuddin,2008)

Pada Gambar 2.11, selektor kelas “item” dan yang diatur adalah *border* yang berfungsi untuk memberikan bingkai garis dengan nilai *solid 1px* yang berarti

akan muncul garis atau bingkai solid 1 *pixel*. Penggunaan divisi kelas sangat berguna dalam pembangunan aplikasi dengan menggunakan kerangka kerja Ionic.

Pada kerangka kerja Ionic untuk memberikan nilai warna tidak perlu menuliskan properti pada selektor CSS, apabila sudah terdapat pada fungsi dalam kelas-kelas yang sudah dibangun oleh Ionic. Terdapat pada contoh *button* yang diberikan warna oleh fungsi Ionic pada kelas-kelas yang sudah ditentukan, dapat dilihat pada Gambar 2.12.



Gambar 2.12 CSS *components* Ionic

(Sumber : Ionic,2016)

Pada Gambar 2.12 penggunaan *value* CSS dengan menggunakan nama kelas, nama kelas yang sudah memiliki *value* CSS dari Ionic tidak dapat diubah dengan menggunakan CSS yang baru. Pada Ionic memberikan *style* pada *tag* HTML yang berbeda terdapat pada file *www/css/style.css*.

Pada pemberian *value* dan *property* pada CSS tidak jauh berbeda dengan *website*, karena Ionic dibangun menggunakan bahasa *website*. Pada *CCS components*, masih banyak nama-nama kelas yang bisa digunakan. Seluruh contoh pada dokumentasi *CCS Components* berisikan nama kelas, bukan *value* dan *property* CSS. Semua itu dibangun oleh Ionic untuk mempermudah pemula *website* dalam membangun aplikasi bergerak.

2.9.3 JavaScript dan Angular

JavaScript adalah bahasa yang kodenya ditulis menggunakan teks biasa (skrip) yang ditempelkan pada dokumen HTML dan diproses pada sisi klien (Kadir,2013). JavaScript merupakan bahasa yang *case sensitive* seperti halnya bahasa pemrograman Java yaitu membedakan penulisan dengan huruf kecil dan huruf besar memiliki arti yang berbeda.

Pada cara kerja JavaScript di Ionic tidak jauh berbeda dengan kinerjanya pada *website*. Mengakses elemen pada HTML dan membuat aksi jika elemen-elemen HTML tersebut mengalami perubahan, misalnya berubahnya warna halaman *website* begitu sebuah tombol di klik. JavaScript membuat sebuah halaman *website* menjadi lebih dinamis dan interaktif. JavaScript yang digunakan pada sebuah aplikasi bergerak yang dibangun menggunakan *Hybrid Application* adalah kerangka kerja Angular.

Angular adalah proyek *open-source* dari Google yang telah sangat populer di kalangan para pengembang aplikasi *website*. Angular membantu para pengembang dalam menuliskan baris-baris kode program secara cepat dan memiliki struktur yang baik. Angular dimulai pada 2009 oleh Meisko Hervery dan Adam Abrons. Akhir nya Hervey bergabung dengan Google dan membawa Angular dengannya. Saat ini, proyek ini sangat populer di kalangan pengembang dan dibawah lisensi MIT (Aditya,2015).

Kerangka kerja angular berbeda dengan beberapa kerangka kerja lainnya seperti PHP dengan konsep *Model View Controller* (MVC). Angular dibangun dengan menggunakan konsep *Model View Whatever* (MVW), pengertian dari *Whatever* pada angular adalah *ViewController*. Proses dari konsep kerangka kerja Angular adalah ketika *Model* menjalankan algoritma lalu terdapat konstruktor pada *Controller* yang mendefinisikan *Model*, setelah itu *Controller* memanggil *View*. Pada penggunaan kerangka kerja Angular, *Controller* tidak perlu memanggil *View*, karena *View* secara otomatis melekat erat dengan *Controller* dengan adanya *Scope* pada Angular yang bisa disebut dengan data *binding*.

Angular bekerja pada sisi *Back End* pada Ionic, untuk mengatur HTML *templating* yang telah dipisahkan oleh *state* Angular. Pada pembangunan menggunakan Ionic 1.7, masih menggunakan Ionic 1 yang berjalan menggunakan Angular 1. Pada kerangka kerja Angular 1 masih menggunakan Ecma Script 5 (ES5). ES adalah bahasa script yang distandarisasikan oleh Ecma Internasional, lalu 5 adalah versi release. Dalam perkembangan ES sudah mencapai ES6 yang digunakan oleh Angular2 dan Ionic 2 pada pembangunan aplikasi bergerak. Penggunaan kerangka kerja Angular dapat dilihat pada Gambar 2.9.

```
var App = angular.module('app', []);

App.controller('PhoneListController', function PhoneListController($scope) {
  $scope.phones = [
    {
      name: 'Nexus S',
      snippet: 'Fast just got faster with Nexus S.'
    }, {
      name: 'Motorola XOOM™ with Wi-Fi',
      snippet: 'The Next, Next Generation tablet.'
    }, {
      name: 'MOTOROLA XOOM™',
      snippet: 'The Next, Next Generation tablet.'
    }
  ];
});
```

Gambar 2.13 *Model dan Controller* pada Angular

(Sumber : Aditya,2015)

Pada Gambar 2.13 berisikan kode program Angular yang merupakan *Model* dan *Controller*. Mendefisikan model dengan nama app dan ditaruh dalam *variable* App, penarukan *Model* pada *variable* memudahkan untuk mendefisikan *Controller*. Pada contoh diatas PhoneListController adalah nama *Controller* yang diberikan pada penulisan kerangka kerja Angular ini, diberikan parameter *Scope* karena *Scope* yang akan menghubungkan *Controller* dengan *View*.

Setelah pembuatan *Model* dan *Controller* sudah selesai. Selanjutnya mendefinisikan nama *Model* pada file HTML dengan menggunakan perintah *ng-app* dengan nama app, dan pendefinisian *Controller* dengan menggunakan *ng-controller* dengan nama *PhoneListController*. Kedua hal ini dilakukan agar

kerangka HTML dapat diubah berdasarkan *Model* dan *Controller* Angular dan dapat mengambil data yang telah dibuat pada *controller* tersebut. Penerapan kedua hal tersebut dapat dilihat pada Gambar 2.14.

```
<html ng-app="app">
  <head>
    ...
    <script src="bower_components/angular/angular.js"></script>
    <script src="app.js"></script>
  </head>
  <body ng-controller="PhoneListController">
    <ul>
      <li ng-repeat="phone in phones">
        <span>{{ phone.name }}</span>
        <p>{{ phone.snippet }}</p>
      </li>
    </ul>
  </body>
</html>
```

Gambar 2.14 *View* pada Angular

(Sumber : Aditya,2015)

Penjelasan pada Gambar 2.14 mengenai pendefinisian *Model* dan *Controller* Angular. Atribut *ng-app* pada tag HTML untuk pendefinisian aplikasi atau Model, didalam *Model* Angular bisa bersisi lebih dari satu *Controller*. Pada setiap tag-tag HTML dapat diisikan atribut *ng-controller* yang berfungsi untuk menjalankan beberapa aktifitas yang sudah dituliskan pada kode didalam *Controller*. Pada atribut HTML untuk menjalankan fungsi Angular terdapat beberapa atribut didalamnya. Contoh atribut yang berjalan untuk Angular dapat dilihat pada Tabel 2.4.

Tabel 2.4 Contoh atribut HTML untuk Angular (Sumber : Aditya,2015)

Atribut	Keterangan
ng-app	Pendefinisian Model Angular
ng-controller	Pendefinisian Controller Angular
{ }	Peletakan objek binding

ng-model filter	Untuk memfilter data binding
ng-repeat	Perulangan pada Angular
ng-click	Untuk memanggil elemen ketika di klik
ng-class	Untuk mengatur kelas pada CSS

2.9.4 Cordova

Cordova merupakan *Application Layer* yang mengatur komunikasi antara jendela *Browser* dengan *native API*. Cordova menyediakan banyak fitur dan *plugin* yang dapat digunakan oleh para pengembang dalam mengembangkan aplikasinya. Cordova berasal dari Phonegap, Adobe telah mengakuisisi Phonegap dan menggabungkannya ke dalam Cordova. Pemilik *layer* Cordova saat ini adalah *Apache Software Foundation*. Phonegap adalah bagian dari Cordova, Phonegap adalah bagian inti dari Cordova yang mendukung beberapa fitur berbayar dari Adobe. Cordova adalah proyek *Open Source Apache* yang memiliki komunitas yang cukup besar. Adobe merupakan pengembang utama dari *framework* ini, Cordova berada di bawah lisensi Apache 2.0 (Aditya,2015).

Cordova bekerja pada Ionic ketika komponen Ionic ingin dijadikan APK *debug* atau APK *release*. Pembuatan APK Cordova berada diantara komponen Ionic dan *native API* berupa SDK dan JDK. Pembuatan dilakukan pada terminal atau CMD, sebelum menjalankan semua itu perintah *ionic serve* memanggil *library* Node JS. Karena didalam Ionic, Cordova dan Angular adalah *package* milik Node JS. Pembentukan APK bisa dengan perintah *ionic build android* yang akan berada pada folder *root/platform/android/build/output/apk/android-debug.apk*. Perintah *ionic run android* untuk memasang aplikasi langsung ke ADV atau *smartphone*.

Apabila pembuatan aplikasi sudah selesai, Cordova dapat membuatkan APK *release* yang siap untuk ditaruh ke dalam *Store*. Pembuatan APK *release* membutuhkan *tool* tambahan yaitu Zipalign milik Android SDK. Setelah APK *release* sudah jadi atau APK *debug* sudah jadi, aplikasi bisa dijalankan pada *smartphone* dengan sistem operasi Android.

2.10 Sistem Operasi Android

Android adalah sistem operasi berbasis linux yang dirancang untuk perangkat bergerak dengan layar sentuh seperti *smartphone* dan komputer tablet. Pada awalnya Android dikembangkan oleh Android, Inc., dan dibeli oleh Google pada tahun 2005. Android bersifat *open-source* (sumber terbuka), yaitu lisensi perizinan yang memungkinkan perangkat lunak tersebut dimodifikasi secara bebas.

Arsitektur Android terdiri dari berbagai lapisan dan setiap lapisan terdiri dari beberapa program yang memiliki fungsi berbeda. Berikut ini beberapa lapisan arsitektur, yaitu (Anthony,2016):

1. *Linux Kernel*

Android dibangun di atas kernel Linux 2.6. Namun secara keseluruhan android bukanlah linux, karena dalam android tidak terdapat paket standar yang dimiliki oleh linux lainnya. *Linux Kernel* merupakan layer tempat keberadaan inti dari operating system android. Layer ini berisi file-file system yang mengatur system processing, memory, resource, drivers, dan sistem android lainnya. Inilah yang membuat file sistem pada Android mirip dengan file sistem pada sistem operasi berbasis Linux.

2. *Libraries*

Libraries atau Pustaka bukanlah suatu aplikasi yang dapat berjalan sendiri dan hanya dapat digunakan oleh program yang berada di level atasnya. Android menggunakan beberapa paket pustaka yang terdapat pada C/C++ dengan standar *Berkeley Software Distribution* (BSD). Beberapa pustaka (*library*) diantaranya :

1. *Surface Manager*, untuk mengatur / mengelola hak akses ke subsistem *layer*, lapisan komposit 2D dan grafis 3D dari dari berbagai aplikasi.
2. *Media Library*, untuk memutar dan merekam berbagai macam format audio dan video dengan PacketVideo berdasarkan OpenCore.
3. *Graphic Library* termasuk didalamnya mesin grafis 2D (SGL) dan OpenGL, untuk tampilan 2D dan 3D.

4. *SQLite*, untuk mengatur relasi *database* yang digunakan pada aplikasi.
5. *SSL* dan *WebKit* untuk *Browser* dan keamanan internet.
6. *LibWebCore* untuk *Android browser* dengan mesin *website* modern yang powerfull.
7. *FreeType* untuk *bitmap* dan *vector font rendering*
8. *System C library*, digunakan untuk variasi dari implementasi *BSD* berdasarkan pelaksanaan *system* standar *C library* (*libc*).

3. *Android Runtime*

Pada *Android* tertanam paket pustaka inti yang menyediakan sebagian besar fungsi *Android*. Inilah yang membedakan *Android* dibandingkan dengan sistem operasi lain yang juga mengimplementasikan *Linux*. *Android Runtime* merupakan mesin virtual yang membuat aplikasi *Android* menjadi lebih tangguh dengan paket pustaka yang telah ada. Dalam *Android Runtime* terdapat 2 bagian utama, diantaranya :

1. Pustaka Inti (*core*) *library*, *android* dikembangkan melalui bahasa pemrograman *Java*, tapi *Android Runtime* bukanlah mesin virtual *Java*. Pustaka inti *Android* menyediakan hampir semua fungsi yang terdapat pada pustaka *Java* serta beberapa pustaka khusus *Android*. *Core library* berfungsi untuk menerjemahkan bahasa *Java* atau *C*, karena *Android* asal mula terbangun menggunakan Bahasa *C* dan dikembangkan menjadi *Java*.
2. Mesin Virtual *Dalvik*, *Dalvik* merupakan sebuah mesin virtual yang dikembangkan oleh Dan Bornstein yang terinspirasi dari nama sebuah perkampungan yang berada di *Iceland*. *Dalvik* hanyalah interpreter mesin virtual yang mengeksekusi file dalam format *Dalvik Executable* (*.dex). Dengan format ini *Dalvik* akan mengoptimalkan efisiensi penyimpanan dan pengalamatan memori pada file yang dieksekusi. *Dalvik* berjalan di atas kernel *Linux 2.6*, dengan fungsi dasar seperti *threading* dan manajemen memori yang terbatas. [Nicolas Gramlich, Andbook, anddev.org]. Mesin Virtual *Dalvik* merupakan sebuah mesin virtual

berbasis register yang dioptimalkan untuk menjalankan fungsi-fungsi pada Android secara efisien.

4. *Application Framework*

Kerangka aplikasi menyediakan kelas-kelas yang dapat digunakan untuk mengembangkan aplikasi Android. Selain itu, juga menyediakan abstraksi generik untuk mengakses perangkat, serta mengatur tampilan *user interface* dan sumber daya aplikasi. Beberapa program penting pada *Application Framework* antara lain :

1. *View*, Melihat tampilan menu secara list, grid, text boxes
2. *Activity Manager*, berfungsi untuk mengontrol siklus hidup aplikasi dan menjaga keadaan "*Backstack*" untuk navigasi penggunaan.
3. *Content Providers*, berfungsi untuk merangkum data yang memungkinkan digunakan oleh aplikasi lainnya, seperti daftar nama.
4. *Resource Manager*, untuk mengatur sumber daya yang ada dalam program. Serta menyediakan akses sumber daya diluar kode program, seperti karakter, grafik, dan file layout.
5. *Location Manager*, berfungsi untuk memberikan informasi detail mengenai lokasi perangkat Android berada.
6. *Notification Manager*, mencakup berbagai macam peringatan seperti, pesan masuk, janji, dan lainnya yang akan ditampilkan pada status bar.

5. *Application Layer*

Puncak dari diagram arsitektur Android adalah lapisan aplikasi dan *widget*. Lapisan aplikasi merupakan lapisan yang paling tampak pada pengguna ketika menjalankan program. Pengguna hanya akan melihat program ketika digunakan tanpa mengetahui proses yang terjadi dibalik lapisan aplikasi. Lapisan ini berjalan dalam *Android Runtime* dengan menggunakan kelas dan *service* yang tersedia pada *framework* aplikasi. Lapisan aplikasi Android sangat berbeda dibandingkan dengan sistem operasi lainnya. Pada Android

semua aplikasi, baik aplikasi inti maupun aplikasi pihak ketiga berjalan diatas lapisan aplikasi dengan menggunakan pustaka API yang sama.

Pada pembangunan aplikasi bergerak berbasis Android terdapat beberapa tools yang digunakan untuk pembangunan aplikasi. *Tool* ini menjadi inti dalam pembangunan, terdiri dari :

1. *Android Software Development Kit*

Android Software Development Kit (SDK) adalah *tools Application Programming Interface* (API) yang diperlukan untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java. Android SDK terdiri dari *debugger, libraries, handset emulator*, dokumentasi, contoh kode dan *tutorial*. Android SDK berfungsi sebagai alat bantu dan API untuk mulai mengembangkan aplikasi pada *platform* Android menggunakan bahasa pemrograman Java.

2. *Java Development Kit*

Java Development Kit (JDK) merupakan perangkat lunak yang digunakan untuk melakukan proses kompilasi dari Java code ke *bytecode* yang dapat dimengerti dan dapat dijalankan oleh *Java Runtime Environment* (JRE). JDK berisi sekumpulan *command line tool* untuk menciptakan program Java. JDK wajib terinstal pada komputer yang akan melakukan proses pembuatan aplikasi berbasis Java, namun tidak wajib terinstal di perangkat yang akan menjalankan aplikasi yang dibangun dengan Java.

3. *Android Virtual Device*

Android Virtual Devices (AVD) merupakan emulator perangkat Android pada komputer. AVD dapat digunakan untuk menjalankan aplikasi yang sudah dibuat kedalam berbagai jenis perangkat sesuai model yang di pilih. Dengan menggunakan *AVD programmer* dapat mengembangkan dan mencoba aplikasi Android tanpa harus menggunakan perangkat Android sebenarnya.

2.11 Application Programming Interface

Application Programming Interface (API) adalah sebuah bahasa dan format pesan yang digunakan oleh program aplikasi untuk berkomunikasi dengan sistem operasi atau program pengendalian lainnya seperti *System Manajemen Database* (DBMS) atau komunikasi protokol. API diimplementasikan dengan menulis fungsi panggilan atau sintaks dalam program, yang menyediakan sarana yang diperlukan untuk meminta layanan program. Pada dasarnya, program API mendefinisikan cara yang tepat bagi *developer* untuk meminta layanan dari program itu. Sebuah API dapat digunakan untuk mengizinkan aplikasi lain atau pihak ketiga dan para pengembang untuk memanfaatkan fasilitas-fasilitas pada API tersebut. Beberapa API yang banyak digunakan oleh pengembang diantaranya adalah Google Map API, Twitter API, dan Facebook API (Anonim,2016).

2.12 JavaScript Object Notation

JavaScript Object Notation (JSON) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman JavaScript, Standar ECMA-262 Edisi ke-3 - Desember 1999. Walaupun JSON didasarkan pada subset bahasa pemrograman JavaScript, JSON merupakan format teks yang tidak bergantung pada suatu bahasa. Kode untuk pengolahan dan pembuatan data JSON telah tersedia untuk banyak jenis bahasa pemrograman. Sehingga memudahkan para pengembang untuk melakukan pertukaran data pada aplikasinya (Anonim,2016).

JSON menggunakan bentuk atau struktur sebagai berikut :

1. Objek adalah sepasang nama atau 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).

2. Array atau Larik adalah kumpulan nilai yang terurutkan. Larik dimulai dengan [(kurung kotak buka) dan diakhiri dengan] (kurung kotak tutup). Setiap nilai dipisahkan oleh , (koma).
3. 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.
4. 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.
5. Angka adalah sangat mirip dengan angka di C atau Java, kecuali format oktal dan heksadesimal tidak digunakan.

2.13 Unified Modeling Language

Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang di berbagai negara dapat mengerti pemodelan perangkat lunak. Pada perkembangan teknik pemrograman berorientasi objek, munculah sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML) (Shalahuddin,2013).

UML adalah metode pemodelan secara visual sebagai sarana untuk merancang dan atau membuat *software* berorientasi objek. Karena UML adalah salah satu *tool* atau model untuk merancang pengembangan *software* yang berbasis *object oriented*. UML sendiri juga memberikan standar penulisan sebuah sistem *blue print*, yang meliputi konsep bisnis proses, penulisan kelas-kelas dalam bahasa program yang spesifik, skema *database*, dan komponen-komponen yang diperlukan dalam sistem *software*. Sebuah bahasa model adalah sebuah bahasa yang mempunyai *vocabulary* dan konsep tatanan atau aturan penulisan serta secara fisik mempresentasikan dari sebuah sistem.

UML adalah sebuah bahasa standar untuk pengembangan sebuah *software* yang dapat menyampaikan bagaimana membuat dan membentuk model-model, tetapi tidak menyampaikan apa dan kapan model yang seharusnya dibuat yang merupakan salah satu proses implementasi pengembangan *software*. UML tidak hanya merupakan sebuah bahasa pemrograman visual saja, namun juga dapat secara langsung dihubungkan ke berbagai bahasa pemrograman, seperti Java, C++, Visual Basic, atau bahkan dihubungkan secara langsung ke dalam sebuah *object-oriented database*. Begitu juga mengenai pendokumentasian dapat dilakukan seperti; *requirements*, arsitektur, *design*, *source code*, *project plan*, *tests*, dan *prototypes*.

Dalam Perancangan aplikasi yang akan dibangun pada penulisan ini, pembangunan akan dimodelkan pada diagram UML yaitu *Use Case Diagram*, *Class Diagram* dan *Activity Diagram* pada perancangan Alur Program aplikasi :

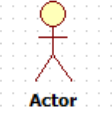
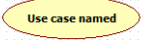

1. Diagram *Use Case*

Diagram *Use Case* menggambarkan apa saja aktivitas yang dilakukan oleh suatu sistem dari sudut pandang pengamatan luar yang menjadi persoalan adalah apa yang dilakukan bukan bagaimana melakukannya. Kegunaan dari diagram *use case* yaitu:

- a. Menjelaskan fasilitas yang ada (*requirements*). *Use Case* baru selalu menghasilkan fasilitas baru ketika sistem dianalisa, dan *design* menjadi lebih jelas.
- b. Komunikasi dengan klien. Pada *use case* penggunaan notasi dan simbol membuat pengembangan menjadi lebih mudah berkomunikasi dengan klien-kliennya.
- c. Membuat pengujian dari kasus-kasus secara umum. Kumpulan dari kejadian untuk *use case* dapat dilakukan dengan pengujian kasus layak untuk kejadian tersebut.

Pada diagram *use case* terdapat beberapa simbol yang digunakan. Simbol tersebut yaitu *actor*, *use case*, dan *association* yang dapat dilihat pada Tabel 2.5











Tabel 2.5 Simbol pada Diagram *Use Case*

SIMBOL	NAMA	KETERANGAN
	Actor	<i>Actor</i> adalah pengguna sistem. <i>Actor</i> tidak terbatas hanya manusia saja, jika sebuah sistem berkomunikasi dengan aplikasi lain dan membutuhkan input atau memberikan output, maka aplikasi tersebut juga bisa dianggap sebagai <i>actor</i>
	Use Case	<i>Use Case</i> digambarkan sebagai lingkaran elips dengan nama use case dituliskan didalam <i>eclips</i> tersebut.
	Association	Asosiasi digunakan untuk menghubungkan actor dengan <i>use case</i> . Asosiasi digambarkan dengan sebuah garis yang menghubungkan antara <i>Actor</i> dengan <i>Use Case</i>

2. Diagram *Activity*

Pada dasarnya diagram *Activity* sering digunakan oleh *flowchart*. Diagram ini berhubungan dengan diagram *statechart*. Diagram *activity* berfokus pada aktifitas-aktifitas yang terjadi yang terkait dalam suatu proses tunggal. Diagram ini menunjukkan bagaimana aktifitas-aktifitas tersebut bergantung satu sama lain. Pada diagram *activity* terdapat beberapa simbol yang digunakan yaitu *InitialState*, *FinalState*, *ActionState*, *Decision*, *Synchronization*, *Rake*, *Signal Accep State*, *Signal Send State*, *Flow Final*, dan *Time* untuk lebih jelasnya dapat dilihat pada tabel 2.6.

Tabel 2.6 Simbol pada Diagram *Activity*

SIMBOL	NAMA	KETERANGAN
	InitialState	Titik Awal
	FinalState	Titik Akhir
	ActionState	Activity
	Decision	Pilihan Untuk mengambil Keputusan
	Synchronization	Fork; untuk menggabungkan dua kegiatan paralel menjadi satu.
	Rake	Menunjukkan adanya dekomposisi
	Signal Accept State	Tanda penerimaan
	Signal Send State	Tanda pengiriman
	Flow Final	Aliran akhir (Flow Final)
	Time	Tanda Waktu

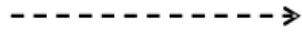
3. Diagram *Class*

Ada 4 macam hubungan dalam penggunaan UML :

1. *Dependency*

Dependency merupakan relasi yang menunjukkan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain. Elemen yang ada di bagian tanda panah adalah elemen yang tergantung pada elemen yang ada dibagian tanpa tanda panah. Terdapat 2 *stereotype* dari *dependency*, yaitu *include* dan *extend*. *Include* menunjukkan bahwa suatu bagian dari elemen (yang ada digaris tanpa panah) memicu eksekusi bagian dari elemen lain (yang ada di

garis dengan panah). *Extend* menunjukkan bahwa suatu bagian dari elemen di garis tanpa panah bisa disisipkan kedalam elemen yang ada di garis dengan panah seperti pada Gambar 2.15.



Gambar 2.15 *Dependency*

2. *Association*

Association menggambarkan navigasi antar kelas (*navigation*), berapa banyak obyek lain yang bisa berhubungan dengan satu obyek (*multiplicity* antar kelas) dan apakah suatu kelas menjadi bagian dari kelas lainnya (*aggregation*). *Navigation* dilambangkan dengan penambahan tanda panah di akhir garis. *Bidirectional navigation* menunjukkan bahwa dengan mengetahui salah satu class bisa didapatkan informasi dari kelas lainnya. Sementara *Unidirectional navigation* hanya dengan mengetahui kelas diujung garis *association* tanpa panah kita bisa mendapatkan informasi dari kelas di ujung dengan panah, tetapi tidak sebaliknya. *Aggregation* mengacu pada hubungan “has-a”, yaitu bahwa suatu class memiliki kelas lain, misalnya rumah memiliki kelas yaitu kamar. Umumnya *association* digambarkan dengan sebuah garis yang dilengkapi dengan sebuah label, nama, dan status hubungannya. Gambar dari *Asocation* dapat dilihat pada Gambar 2.16.



Gambar 2.16 *Association*

3. *Generalization*

Generalization menunjukkan hubungan antara elemen yang lebih umum ke elemen yang lebih spesifik. Dengan *generalization*, kelas yang lebih spesifik (*subclass*) akan menurunkan atribut dan operasi dari kelas yang lebih umum (*superclass*) atau “*subclass is superclass*”. Dengan menggunakan notasi *generalization* konsep *inheritance* dari prinsip hirarki dapat dimodelkan. Digambarkan dengan garis panah seperti Gambar 2.17.

Gambar 2.17 *Generalization*

4. *Realization*

Realization menunjukkan hubungan bahwa elemen yang ada di bagian tanpa panah akan merealisasikan apa yang dinyatakan oleh elemen yang ada di bagian dengan panah. Misalnya kelas merealisasikan *package*, *component* merealisasikan kelas atau *interface*. Gambar *realization* dapat dilihat pada Gambar 2.18.

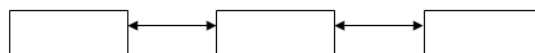
Gambar 2.18 *Realization*

2.14 Struktur Navigasi

Struktur navigasi adalah struktur atau alur dari suatu program yang merupakan rancangan hubungan/rantai kerja dari beberapa area yang berbeda dan dapat membantu mengorganisasikan seluruh elemen pembuatan aplikasi. Menentukan struktur navigasi merupakan hal yang sebaiknya dilakukan sebelum membuat suatu aplikasi. Ada empat macam bentuk dasar dari struktur navigasi yang biasa digunakan yaitu :

1. Struktur Navigasi Linear

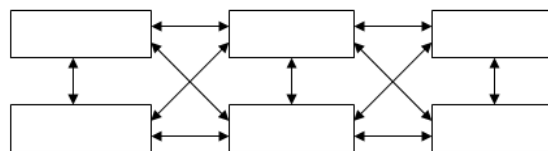
Struktur navigasi linear mempunyai satu rangkaian cerita yang berurut, yang menampilkan satu demi satu tampilan layar secara berurut menurut urutannya. Tampilan yang dapat ditampilkan pada struktur jenis ini adalah satu halaman sebelumnya atau satu halaman sesudahnya, tidak dapat dua halaman sebelumnya atau dua halaman sesudahnya. Gambar dari struktur navigasi linear dapat dilihat pada Gambar 2.19.



Gambar 2.19 Struktur Navigasi Linear

2. Struktur Navigasi Non-linear

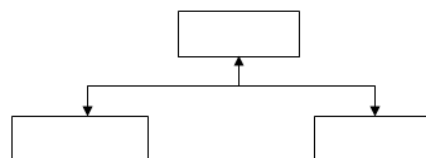
Struktur navigasi non-linear atau struktur tidak berurut merupakan pengembangan dari struktur navigasi linier. Pada struktur ini diperkenankan membuat navigasi bercabang. Percabangan yang dibuat pada struktur nonlinier ini berbeda dengan percabangan pada struktur hirarki, karena pada percabangan non-linear ini walaupun terdapat percabangan. Struktur navigasi non-linear dapat dilihat pada Gambar 2.20.



Gambar 2.20 Struktur Navigasi Non-linear

3. Struktur Navigasi Hirarki

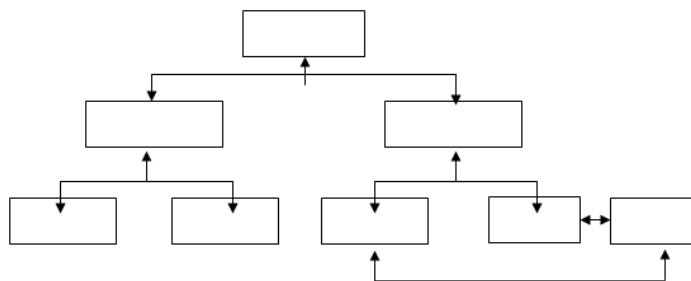
Struktur navigasi hirarki biasa disebut struktur bercabang, merupakan suatu struktur yang mengandalkan percabangan untuk menampilkan data berdasarkan kriteria tertentu. Tampilan pada menu pertama akan disebut sebagai Master Page (halaman utama pertama), halaman utama ini mempunyai halaman percabangan yang disebut Slave Page (halaman pendukung). Jika salah satu halaman pendukung dipilih atau diaktifkan, maka tampilan tersebut akan bernama Master Page (halaman utama kedua), dan seterusnya. Pada struktur navigasi ini tidak diperkenankan adanya tampilan secara linier. Struktur navigasi hirarki dapat dilihat pada Gambar 2.21.



Gambar 2.21 Struktur Navigasi Hirarki

4. Struktur Navigasi Campuran

Struktur navigasi campuran merupakan gabungan dari ketiga struktur sebelumnya yaitu linier, non-linier dan hirarki. Struktur navigasi ini juga biasa disebut dengan struktur navigasi bebas. Struktur navigasi ini banyak digunakan dalam pembuatan aplikasi karena struktur ini dapat digunakan dalam pembuatan aplikasi sehingga dapat memberikan ke-interaksian yang lebih tinggi. Struktur navigasi campuran dapat dilihat pada Gambar 2.22.



Gambar 2.22 Struktur Navigasi Campuran