BAB 2
LANDASAN TEORI

2.1 Tempat Penelitian

PT. LSKK (Langgeng Sejahtera Kreasi Komputasi) berdiri sejak tahun 2008 merupakan perusahaan di bidang Integrasi teknologi, Perusahaan ini didirikan sebagai wahana untuk turut berkontribusi dalam pembangunan nasional, khususnya di bidang ke-elektroteknik-an dan bidang teknik pada umumnya, Berbagai produk telah dihasilkan sejak tahun 2008. Diantaranya produk-produk hasil kerjasama dengan Tentara Nasional Indonesia seperti Rompi Taktis, Anoa Simulator, *Miss Distance Indicator*, How Can Pams, Posko Dahanud *Mobile*, *Hostile Artillery Locator*.

2.2 *Internet Of Things*

Internet of things, atau dikenal juga dengan singkatan IoT, merupakan sebuah konsep yang bertujuan untuk memperluas manfaat dari konektivitas internet yang tersambung secara terus-menerus. Adapun kemampuan seperti berbagi *data*, *remote control*, dan sebagainya, termasuk juga pada benda di dunia nyata. Contohnya bahan pangan, elektronik, koleksi, peralatan apa saja, termasuk benda hidup yang semuanya tersambung ke jaringan lokal dan global melalui sensor yang tertanam dan selalu aktif.

Pada dasarnya, *Internet of thingss* mengacu pada benda yang dapat diidentifikasikan secara unik sebagai representasi virtual dalam struktur berbasis Internet. Istilah *internet of thingss* awalnya disarankan oleh Kevin Ashton pada tahun 1999 dan mulai terkenal melalui Auto-ID Center di MIT.

Cara kerja *internet of things* yaitu dengan memanfaatkan sebuah argumentasi pemrograman yang dimana tiap-tiap perintah argumennya itu menghasilkan sebuah interaksi antara sesama mesin yang terhubung secara otomatis tanpa campur tangan manusia dan dalam jarak berapa pun. Internet yang menjadi penghubung di antara kedua interaksi mesin tersebut, sementara manusia

hanya bertugas sebagai pengatur dan pengawas bekerjanya alat tersebut secara langsung.

2.2.1 Arsitektur *Internet Of Things*

Blok arsitektur *Internet Of Things*(IOT) adalah perangkat sensorik, *remote service invocation* dan komunikasi jaringan [9]. Arsitektur sistem *holistic* untuk *internet Of things* perlu menjamin secara sempurna pengoperasian komponennya dan menyatukan *hardware* dan *software* secara bersamaan, untuk mencapai hal ini, pertimbangan yang cermat diperlukan dalam merancang *failure recovery* dan *scalability*. Model arsitektur *internet Of things* sebagai berikut:

1. SOA-Based *Architecture Service-Oriented Architecture* (SOA) penting untuk penyedia layanan dan pengguna, memastikan *interoperability* di antara perangkat heterogen. Terdapat 4 lapisan fungsi yang berbeda sebagai berikut:
    a. *Sensing Layer*, integrasi dengan objek *hardware* yang tersedia untuk mendapatkan status dari lingkungan.
    b. *Network Layer*, infrastruktur untuk mendukung koneksi *wireless* atau koneksi kabel.
    c. *Service Layer*, untuk membuat dan mengelola layanan yang diperlukan oleh pengguna atau aplikasi.
    d. *Interfaces Layer*, terdiri dari metode interaksi dengan pengguna atau aplikasi.

    Secara umum, dalam arsitektur SOA sistem yang kompleks dibagi menjadi beberapa subsistem yang modular sehingga memberikan cara mudah untuk mempertahankan keseluruhan sistem dengan merawat komponen individualnya, dengan ini dapat memastikan bahwa jika terjadi kegagalan komponen, sisa sistem masih dapat beroperasi secara normal, ini adalah nilai yang sangat besar untuk desain yang efektik dari 18 arsitektur aplikasi *Internet* Of *Things* (IOT) dimana *reability* adalah parameter yang paling signifikan. SOA telah digunakan

secara intesif di WSN, karena tingkat abstraksi yang sesuai dan keunggulan yang berkaitan dengan desain modularnya [9].

2. API-Oriented *Architecture* pendekatan konvensional untuk mengembangkan solusi berorientasi layanan *Remote Method Invocation* (RMI) sebagai sarana untuk menggambarkan, menemukan dan memanggil layanan, karena *overhead* dan *complexity* yang dipaksakan oleh teknik ini, API web dan *Representational State Transfer* (REST) sebagai metode yang memiliki solusi alternatif yang menjanjikan. Sumber daya yang dibutuhkan berkisar dari *bandwidth* jaringan ke kapasitas komputasi dan penyimanan data, dipicu oleh permintaan konversi data permintaan respons yang terjadi secara teratur selama *service calls*. Format pertukaran data ringan seperti *JavaScript Object Notation* (JSON) dapat mengurangi *overhead*, terutama untuk *smart devices* dan sensor dengan sumber daya terbatas, dengan mengganti file XML berukuran besar, ini membantu dalam menggunakan *communication channel* dan pemprosesan di perangkat lebih efisien. Membangun API untuk aplikasi *Internet Of* Things (IOT) membantu penyedia layanan menarik lebih banyak pelanggan sambil berfokus pada fungsionalitas produk dari pada presentasi. Selain itu, lebih mudah untuk mengaktifkan *multitenancy* dengan fitur keamanan API seperti *Auth*, API yang memang mampu meningkatkan eksposisi dan komersialisasi layanan. Dengan API menyediakan pemantauan layanan dan alat bisa lebih efisien dari pada berorientasi dengan layanan sebelumnya [9].

2.3 Pemantauan dan Aktuasi

Pemantauan adalah prosedur penilaian yang secara deskriptif dimaksudkan untuk mengidentifikasi dan/atau mengukur pengaruh dari kegiatan yang sedang berjalan tanpa mempertanyakan hubungan kausalitas. (Wollman, 2003:6) PP No.39/2006 tentang Tatacara Pengendalian dan Evaluasi Pelaksanaan Rencana Pembangunan: Pemantauan adalah kegiatan mengamati perkembangan pelaksanaan rencana pembangunan, mengidentifikasi serta mengantisipasi

permasalahan yang timbul dan/atau akan timbul untuk dapat diambil tindakan sedini mungkin.

2.4 *Smart home*

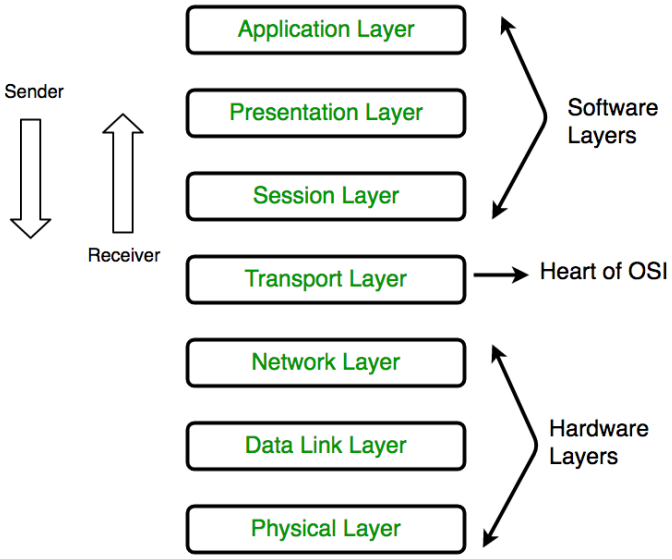
Smart home atau rumah pintar merupakan bagian dari konsep *internet of things*, dimana semua benda atau perabotan sehari – hari yang akrab dengan kehidupan masyarakat di "pintarkan" karena integrasi teknologi dalam bentuk cip yang serba bisa. Dalam hal *Smart home*, *Internet Of things* telah hadir dalam rupa barang – barang yang biasa ditemui di rumah orang kebanyakan [10]. *Smart home* hadir untuk memudahkan para penghuni rumah dalam mengatur segala hal yang berhubungan dengan kenyamanan diri sebagai penghuni rumah, mulai dari soal keamanan hingga soal akses perabotan yang dibuat lebih interaktif dan bisa dikendalikan melalui satu alat saja, yaitu aplikasi pada *smartphone* atau perangkat lainnya.

Gambar 2.1 Ilustrasi *home automation*

2.5 *Open System Interconnection Layer*

OSI adalah akronim dari *Open System Interconnection* yang merupakan referensi abstrak berupa kerangka konseptual untuk mendefinisikan sebuah standar jaringan komputer. Tujuannya adalah untuk menjadi rujukan para vendor dan pengembang dalam mengembangkan sebuah jaringan dalam sebuah produk ataupun *software*. Sehingga tiap-tiap produk bisa saling terkoneksi meski dikembangkan oleh pengembang yang berbeda [11].

Model OSI akan memberikan deskripsi abstrak yang menjelaskan desain lapisan-lapisan komunikasi dan protokol jaringan. OSI inilah yang kemudian membentuk standar umum jaringan untuk membuat gambar antar vendor yang berbeda. Model OSI ini digambarkan dalam tujuh *layer*.



Gambar 2.2 OSI *Layer* [11]

Model OSI hanya merupakan model ideal yang digunakan sebagai acuan untuk memudahkan mempelajari bagaimana protokol – protokol jaringan berfungsi dan berinteraksi. Secara umum, fungsi dari masing-masing *layer* dapat dilihat pada Tabel 2.1.

Tabel 2.1 OSI *Layer*

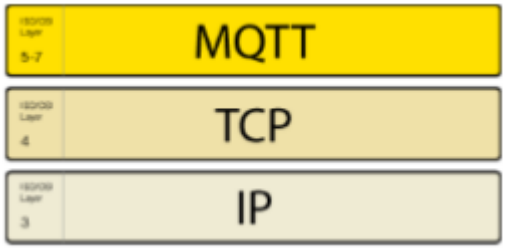
Layer	Keterangan
7 (*Application*)	Berfungsi sebagai antarmuka (penghubung) aplikasi dengan fungsional jaringan, mengatur bagaimana aplikasi dapat mengakses jaringan dan kemudian membuat pesan-pesan. Pada *layer* inilah sesungguhnya *user* "berinteraksi dengan jaringan". Contoh protokol yang berada pada lapisan ini: FTP, telnet, SMTP, HTTP, POP3, NFS.
6 (*Presentation*)	Berfungsi untuk mentranslasikan data yang hendak di transmisikan oleh aplikasi ke dalam format yang dapat

	ditransmisikan melalui jaringan. protokol yang berada pada *level* ini adalah sejenis *redirector software*, seperti *network shell* (semacam *Virtual Network Computing* (VNC)atau *Remote Desktop Protocol* (RDP)). Kompresi data dan enskripsi juga ditangani *layer* ini .
5 (*session*)	Berfungsi untuk mendefinisikan bagaimana koneksi dimulai, dipelihara, dan diakhiri.
4 (*Transport*)	Berfungsi untuk memecah data menjadi paket-paket data serta memberikan nomor urut setiap paket data, sehingga dapat disusun kembali setelah diterima. Paket yang diterima dengan sukses akan diberi tanda (*Acknowledgement*). Sedangkan paket yang rusak atau hilang ditengah jalan akan dikirim ulang.
3 (*Network*)	Berfungsi untuk mendefinisikan alamat-alamat IP dan melakukan *routing* melalui *internetworking* dengan menggunakan *router* dan *switch layer* 3. Pada *layer* ini juga dilakukan proses deteksi *error* dan transmisi ulang paket-paket yang *error*. Contoh protokol yang digunakan seperti: IP, IPX.
2 (*Data Link*)	Berfungsi untuk menentukan bagaimana bit-bit data dikelompokkan menjadi format yang disebut *frame*. Pada *level* ini terjadi *error correction*, *flow control* , pengalamatan perangkat keras (MAC *Address*). Menurut spesifikasi IEEE 802, *layer* ini dikelompokkan menjadi dua yaitu *Logical Link Control* (LLC) dan *Media Access Control* (MAC).
1 (*Physical*)	Berfungsi untuk mendefinisikan media transmisi jaringan, sinkronisasi bit, arsitektur jaringan, dan topologi jaringan. Selain itu, level ini juga mendefinisikan bagaimana *Network Interface Card* (NIC) berinteraksi dengan media *wire* atau *wireless*.

2.6 *Meesage Queuing Telemetry Transport*

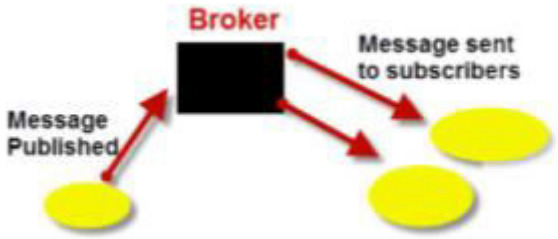
Message Queuing Telemetry Transport (MQTT) adalah protokol yang bersifat *clientserver* dan *publish - subscribe*. Pada OSI *layer*, protokol MQTT

berada pada *layer* aplikasi dan berjalan pada TCP/IP dilihat pada Gambar 2.3 [12].



Gambar 2.3 *Layer* Utama Protokol Pengiriman Data

MQTT sebagaimana yang terlihat pada gambar bukan hanya merupakan protokol aplikasi namun juga menggunakan *layer* 5 sampai 7 pada OSI *layer* (*Application*, *Presentation*, dan *Session*). Konsep *publish* dan *subscribe* dari protokol MQTT dapat dilihat pada Gambar 2.4 dibawah :



Gambar 2.4 Konsep *Publish* dan *Subcribe* [1]

Message Queue Telemetry Transport (MQTT) adalah sebuah protokol komunikasi data *machine to machine* (M2M) yang berada pada *layer* aplikasi, MQTT bersifat *lightweight message* artinya MQTT berkomunikasi dengan mengirimkan data pesan yang memiliki *header* berukuran kecil yaitu hanya sebesar 2bytes untuk setiap jenis data, sehingga dapat bekerja di dalam lingkungan yang terbatas sumber dayanya seperti kecilnya *bandwidth* dan terbatasnya sumber daya listrik, selain itu protokol ini juga menjamin terkirimnya semua pesan walaupun koneksi terputus sementara, protokol MQTT menggunakan metode *publish*/*subscribe* untuk metode komunikasinya.

Publish/subscribe sendiri adalah sebuah pola pertukaran pesan di dalam komunikasi jaringan dimana pengirim data disebut *publisher* dan penerima data disebut dengan *subscriber*, metode *publish/subscribe* memiliki beberapa kelebihan salah satunya yaitu *loose coupling* atau *decouple* dimana berarti antara *publisher* dan *subscriber* tidak saling mengetahui keberadaannya, terdapat 3 buah *decoupling* yaitu *time decoupling*, *space decoupling* dan *synchronization decoupling*, *time decoupling* adalah sebuah kondisi dimana *publisher* dan *subscriber* tidak harus saling aktif pada waktu yang sama, *space decoupling* adalah dimana *publisher* dan *subscriber* aktif di waktu yang sama akan tetapi antara *publisher* dan *subscriber* tidak saling mengetahui keberadaan dan identitas satu sama lain, dan yang terakhir adalah *synchronization decoupling* kondisi dimana pengaturan event baik itu penerimaan atau pengiriman pesan di sebuah *node* hingga tidak saling mengganggu satu sama lain [2].

Pengiriman data pada MQTT didasari oleh topik, MQTT topik memiliki tipe data *string* dan untuk perbedaan hirarki atau level dari topik digunakan tanda baca "/" (*Solace*, n.d.).MQTT memiliki 3 level *quality of service* (QOS) dalam pengiriman pesannya yaitu 0,1,2 [2].

Dalam MQTT dikenal istilah topik yaitu berupa UTF-8 *string* yang perannya hampir sama seperti topik pada chat hanya lebih sederhana dan berfungsi sebagai *filter* untuk *broker* dalam mengirimkan pesan ke tiap klien yang terhubung atau dengan kata lain topik adalah kanal bagi klien untuk *subscribe*.

2.6.1 *Quality Of Service* (QOS)

Quality of service adalah teknik untuk mengelola *bandwidth*, *delay, throughput* dan *packet loss* untuk aliran dalam jaringan [13], tujuan dari mekanisme QoS adalah mempengaruhi satu diantara empat parameter dasar QoS yang telah ditentukan. QoS didesain untuk membantu *end user* (*client*) menjadi lebih produktif dengan memastikan bahwa *user* mendapatkan performansi yang handal dari aplikasi - aplikasi berbasis jaringan. QoS mengacu pada kemampuan jaringan untuk menyediakan layanan yang lebih baik pada jaringan yang berbeda-beda, paramater QoS yang digunakan adalah :

1. *Throuhgput*

Throughput merupakan kemampuan sebenarnya dari suatu jaringain dalam melalukan pengiriman data. Berbeda dengan *bandwidth*, karena *bandwidth* bersifat tetap, sedangkan *throughput* sifatnya dinamis tergantung *traffic* yang sedang terjadi, tabel kategori throughput dapat dilihat pada Tabel 2.2.

Tabel 2.2 Kategori Throughput [14]

Kategori *Throughput*	*Throughput*
Buruk	0 - 388 Kbps
Cukup	338 - 700 Kbps
Baik	700 - 1.200 Kbps
Sangat Baik	> 1.200 Kbps

2. *Packet loss*

Packet loss adalah sebuah kegagalan transmisi paket IP dalam mencapai tujuannya. Setiap paket yang dikirimkan melalui sebuah protokol pengiriman data memiliki risiko dalam mencapai tujuannya dapat disebabkan oleh beberapa kemungkinan diantaranya adalah :

a. Terjadinya *Overload* trafik didalam jaringan.
b. *Error* yang terjadi pada media pengiriman fisik.
c. kegagalan yang terjadi pada sisi penerima antara lain disebabkan karena *overflow* yang terjadi pada *buffer*.

Secara umum terdapat empat kategori performansi jaringan yaitu kategori baik, cukup, buruk dan jelek, kategori berdasarkan nilai *packet loss* dapat dilihat pada Tabel 2.3 sebagai berikut:

Tabel 2.3 Kategori Packet Loss [14]

Kategori *Packet Loss*	*Packet Loss*
Baik	0%
Cukup	3%
Buruk	15%
Jelek	25%

3. *Delay*

Delay adalah waktu tunda suatu paket yang diakibatkan oleh proses transmisi dari satu titik ke titik lain yang menjadi tujuannya. Secara umum

terdapat empat kategori *delay* berdasarkan nilai *delay* yaitu seperti tampak pada Tabel 2.4 berikut:

Tabel 2.4 Kategori Delay [14]

Kategori Delay	Besar *Delay* (milidetik)
Sangat Bagus	< 150 ms
Bagus	150 s/d 300 ms
Sedang	300 s/d 450 ms
Jelek	> 450 ms

Pengiriman pesan yang dikirim memiliki satu dari tiga *level Quality of Service* (QoS). Level – level ini memberikan garansi akan konsistensi dari pengiriman pesan, Setiap *quality of service* memiliki kekurangan dan kelebihannya masing – masing, yang mana dalam pemilih *quality of service* harus dipertimbangkan kembali pengiriman pesan yang diperlukan, dibutuhkan *quality of service* yang memadai, sehingga dapat meningkat efisiensi dalam pengiriman data yang dapat dilihat pada Tabel 2.5 *Quality of Service Level* [2].

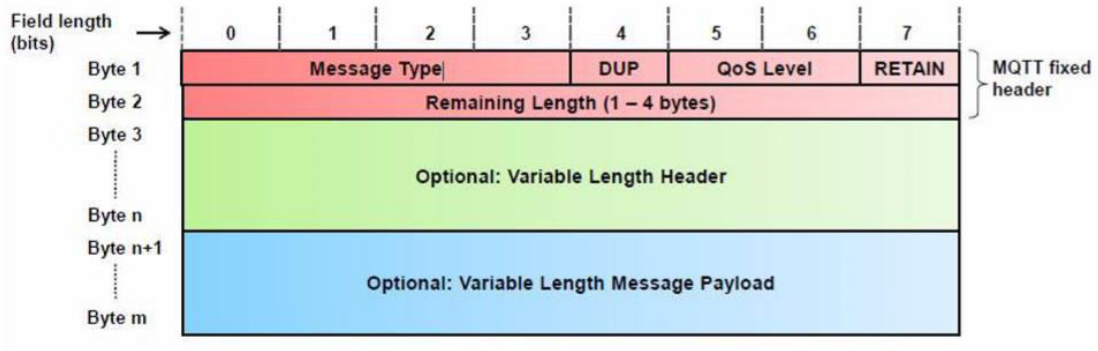
Tabel 2.5 *Quality of Service Level*

Level	Tipe	Deskripsi
0	*Fire and Forget*	Pesan dikirim hanya sekali. Pesan yang terkirim tergantung dari reliabilitas *stack* TCP tergantung keberadaan jaringan dan tidak ada usaha untuk mentransmisikan pesan kembali. QoS ini adalah yang digunakan dalam sistem *home automation* dalam pengiriman data status sensor pada penelitian ini.
1	*At least once*	Pesan dikirimkan setidaknya satu kali. Jadi klien setidaknya akan menerima pesan sekali. Jika *subscriber* tidak mengakui (*acknowledge*) maka *broker* akan mengirimkan pesan sampai *publisher* menerima status pengakuan pesan dari klien.
2	*Exactly once*	Pesan dikirimkan setidaknya satu kali. Jika *subscriber* tidak mengakui (*acknowledge*) maka *broker* akan

		mengirimkan pesan sampai *publisher* menerima status pengakuan pesan dari klien.

2.6.2 Struktur Paket MQTT

MQTT memiliki *header* tetap (*fixed header*) sebesar 2 *Bytes* diikuti oleh *variable length header* dan *variable length message payload* yang besarannya tergantung dari panjang dari pesan yang dikirim [1], struktur paket tersebut yang membuat struktur paket MQTT *lightweight* yang mana struktur ringan tersebut tepat untuk sistem *home automation* dengan *bandwidth* yang rendah atau terbatas. Struktur paket MQTT dapat dilihat pada Gambar 2.5 Struktur Paket MQTT.



Gambar 2.5 Struktur Paket MQTT

2.7 *Delay*

Delay merupakan penundaan waktu suatu paket yang diakibatkan oleh proses transmisi dari satu titik ke titik yang lain yang menjadi tujuannya. *Delay* merupakan penundaan waktu paket tiba ke dalam sistem komputer *client* atau *server* sampai selesai ditransmisikan. Untuk menghitung *delay* transmisi dapat dicari dengan menggunakan persamaan 2.1 sebagai berikut [1]:

$$Delay(\text{second}) = \frac{waktu\ paket\ yang\ dikirim}{jumlah\ paket} \quad (2.1)$$

Dengan penjelasan sebagai berikut:

$Delay(second)$ = *delay* transmisi yang dikirim oleh *client* menuju *server*

Waktu paket yang dikirim = selisih waktu pengiriman paket pertama dan terakhir
Jumlah paket = jumlah paket yang tersaring

2.8 *Packet loss*

Packet loss adalah kegagalan pengiriman paket pada alamat tujuannya sehingga menyebabkan beberapa atau seluruh paket dalam waktu pengiriman hilang. Untuk menghitung *packet loss* dapat menggunakan persamaan 2.2 sebagai berikut [1]:

$$Packet\ loss = \frac{Paket\ total\ tertangkap - paket\ terkirim}{Paket\ total\ tertangkap} \quad (2.2)$$

Dengan penjelasan sebagai berikut:
$Packet\ Loss$ = paket hilang
$Paket\ terkirim$ = paket yang dikirim oleh *client* ke *server*
$Paket\ total\ tertangkap$ = paket yang ditangkap oleh perangkat lunak Wireshark

2.9 *Throughput*

Throughput merupakan suatu kinerja jaringan yang terukur. *Throughput* juga diartikan sebagai kemampuan sebenarnya suatu jaringan dalam melakukan pengiriman data per satuan waktu [1]. Untuk menghitung *throughput* terhadap alamat *website* dapat menggunakan persamaan 2.3 sebagai berikut:

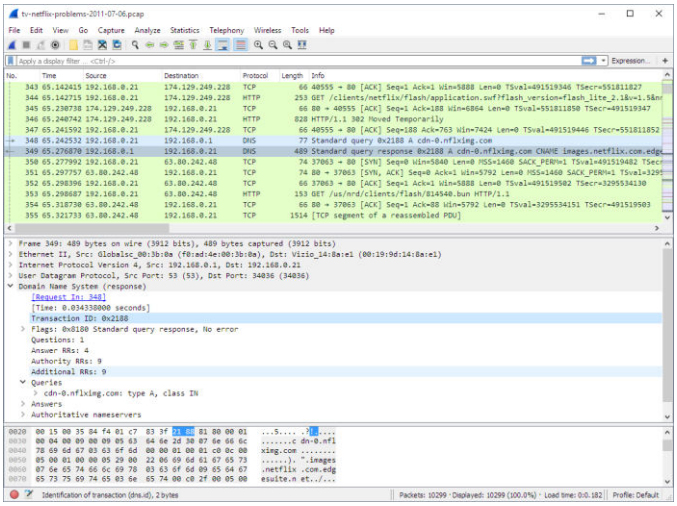
$$throughput = \frac{Jumlah\ data\ yang\ dikirim}{waktu\ pengiriman\ data} \quad (2.3)$$

dengan:
$Throughput$ = *Bytes* / *sec*
$Jumlah\ data\ yang\ dikirim$ = *Bytes* rata-rata / *sec*
$Waktu\ pengiriman\ data$ = waktu antara paket terakhir dan pertama

2.10 Wireshark

Wireshark *Network Protocol Analyzer* adalah sebuah aplikasi perangkat lunak yang digunakan untuk dapat melihat dan mencoba menangkap paket-paket

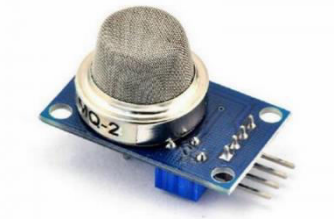
jaringan dan berusaha untuk menampilkan semua informasi di paket tersebut sedetail mungkin [15]. *Open source* dari wireshark menggunakan *Graphical User Interface* (GUI) seperti diperlihatkan pada Gambar 2.6 Tampilan UI Wireshark



Gambar 2.6 Tampilan UI Wireshark

2.11 Sensor Gas MQ-2

Sensor MQ-2 merupakan sensor yang dapat mendeteksi beberapa jenis gas yang mudah terbakar seperti butana, metana, LPG, propana, alkohol, hidrogen dan dapat mendeteksi PPM asap karbon [16]. Karakteristik hambatan sensor MQ-2 terhadap perubahan berbagai kadar gas di udara dapat dilihat pada Gambar 2.7 Sensor MQ-2.



Gambar 2.7 Sensor MQ-2

2.12 *Flame* Sensor

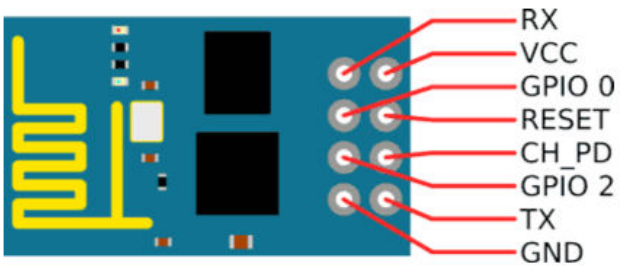
Modul sensor api yang terdiri dari sensor api (IR *receiver*), *resistor*, kapasitor, *potentiometer* dan komparator LM393 dalam sebuah modul yang terintegrasi, modul ini dapat mendeteksi cahaya *infrared* dengan gelombong yang memiliki rentang 700nm hingga 1000nm, Cara kerja sensor ini yaitu dengan mengidentifikasi atau mendeteksi nyala api dengan menggunakan metode optik. Pada sensor ini menggunakan tranduser yang berupa *infrared* (IR) sebagai *sensing* sensor [17]. Tranduser ini digunakan untuk mendeteksi akan penyerapan cahaya pada panjang gelombang tertentu, sensor api dapat dilihat pada Gambar 2.8.

Gambar 2.8 LM393 Sensor Api [18]

Yang dimana memungkinkan alat ini untuk membedakan antara spektrum cahaya pada api dengan spektrum cahaya lainnya seperti spektrum cahaya lampu.

2.13 Modul ESP 8266

Modul ESP8266 adalah sebuah komponen cip terintegrasi yang didesain untuk keperluan dunia masa kini yang serba tersambung. Cip ini menawarkan solusi *networking* Wi-Fi yang lengkap dan menyatu, yang dapat digunakan sebagai penyedia aplikasi atau untuk memisahkan semua fungsi *networking* Wi-Fi ke pemproses aplikasi lainnya. ESP8266 memiliki kemampuan *on-board processing* dan *storage* yang memungkinkan cip tersebut untuk diintegrasikan dengan sensor-sensor atau dengan aplikasi alat tertentu melalui pin *input output* hanya dengan pemrograman singkat [19]. Modul komunikasi WiFi dengan IC SoC ESP8266EX Serial-to-WiFi *Communication Module* ini merupakan modul WiFi dengan harga ekonomis.



Gambar 2.9 Informasi GPIO ESP8266

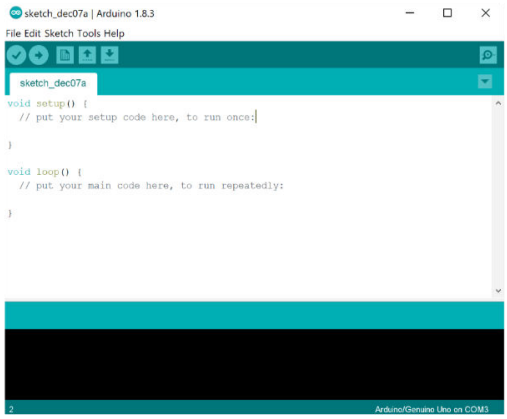
Modul WiFi ini bekerja dengan catu daya 3,3 *volt*. Salah satu kelebihan modul ini adalah kekuatan transmisinya yang dapat mencapai 100 meter, dengan begitu modul ini memerlukan koneksi arus yang cukup besar (rata-rata 80 mA, mencapai 215 mA pada CCK 1 MBps, moda transmisi 802.11b dengan daya pancar +19,5 dBm belum termasuk 100 mA untuk sirkuit pengatur tegangan *internal*).Perhatian bagi pengguna Arduino: sumber daya dari pin 3v3 Arduino tidak dirancang untuk memasok arus dalam jumlah besar, gunakan catu daya terpisah. dapat menggunakan DC *Buck Converter* semacam AMS1117-3.3 untuk mengkonversi tegangan dari catu daya 5 v*olt*. Untuk berkomunikasi dengan MCU 5 *volt*, gunakan level *converter* 5V ⇔ 3v3.Untuk komunikasi, model ini menggunakan koneksi 115200,8,N,1 (115.200 bps, 8 data-*bit*, *no parity*, 1 *stop* bit).

2.14 *Sofware* Arduino IDE

Sehubungan dengan pembahasan untuk saat ini *software* arduino yang akan digunakan adalah *driver*. IDE arduino adalah *software* yang sangat canggih ditulis dengan menggunakan *java* [20]. IDE Arduino terdiri dari :

1. *Editor* program, sebuah *window* yang memungkinkan pengguna menulis dan mengubah program dalam bahasa *processing*.
2. *Compiler*, sebuah modul yang mengubah kode program (bahasa *processing* menjadi kode *biner*. Bagaimanapun sebuah mikrokontroler tidak akan bisa memahami bahasa *processing*. Yang bisa dipahami oleh mikrokontroler adalah kode *biner*. Itulah sebabnya *compiler* diperlukan dalam hal ini.

3. *Uploader*, sebuah modul yang memuat kode *biner* dari komputer ke dalam *memory* dalam papan arduino.



Gambar 2.10 Tampilan Arduino IDE

2.15 *Solid State Relay*

Solid State Relays adalah sebuah komponen semikonduktor yang bekerja layaknya sebuah *relay* elektro mekanis dan untuk otomasi serta mampu mengendalikan beban listrik tanpa penggunaan komponen mekanis seperti halnya pada *relay* mekanisme, Tidak seperti jenis *relay* mekanis yang menggunakan *coil*, medan magnet, serta pegas dan terminal kontak untuk mengalirkan listrik, SSR tidak lagi menggunakan komponen bergerak seperti itu [21]. SSR telah menggunakan arus dan semikonduktor *solid state* untuk menangani *input* terhadap *output* serta memerankan fungsinya sebagai saklar.

Seperti halnya dengan *relay* mekanik, SSR memiliki fungsi pemisahan *input output* seperti halnya sebuah saklar dengan hambatan yang sangat tinggi pada kondisi *open*. Sedangkan saat terhubung, SSR mampu mengalirkan arus dengan arus yang sangat besar dengan hambatan yang sangat kecil, perangkat keras *solid state relay* dapat dilihat pada Gambar 2.11 *Solid State Relay*.

Gambar 2.11 ***Solid State Relay***

2.16 MySql

MySQL adalah sebuah perangkat lunak sistem manajemen basis data SQL (bahasa Inggris: *database management system*) atau DBMS yang *multithread*, *multi-user*, dengan sekitar 6 juta instalasi di seluruh dunia. MySQL AB membuat MySQL tersedia sebagai perangkat lunak gratis di bawah lisensi GNU *General Public License* (GPL), tetapi mereka juga menjual dibawah lisensi komersial untuk kasus-kasus dimana penggunaannya tidak cocok dengan penggunaan GPL [22]. Pada penelitian ini, MySQL digunakan sebagai perangkat lunak untuk penyimpanan data.

Gambar 2.12 Logo Mysql

2.17 *Unified Modeling Language*

Unified Modeling Language (UML) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi objek. Hal ini disebabkan karena UML menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atau visi mekanisme yang efektif untuk berbagi dan mengkomunikasikan rancagan mereka dengan yang lain. *Unified Modeling Language* (UML) merupakan kesatuan dari bahasa pemodelan yang dikembangkan Booch, *Object Modeling Technique* (OMT) dan *Object Orented Software Engineering* (OOSE) Metode ini menjadikan proses analisis dan desain kedalam empat tahapan iterative, yaitu identifikasi kelas – kelas, dan objek – objek, identifikasi semantik dari hubungan objek dan kelas tersebut, perincian *interface* dan implementasi. Keunggulan metode Booch adalah pada detail dan kayanya dengan notasi dan elemen, pemodelan OMT yang dikembangkan oleh Rumbaugh didasarkan pada analisis terstruktur dan pemodelan *entity-relationship*. Tahapan utama dalam metodologi ini adalah nalisis, design sistem, design objek dan implementasi. Keunggulan metode ini adalah dalam penotasian yang mendukung konsep OOSE.

Metode OOSE dari Jacobson lebih memberi menekanan pada *use case*. OOSE memiliki tiga tahapan yaitu membuat model *requirement* dan analisis, desain dan implementasi, dan model pengujian. Keunggulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhana namun mencakup seluruh tahapan dalam rekayasa perangkat lunak. Dengan UML, metode Booch, OMT dan OOSE digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baru yang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam daripada metode lainnya.

Sebagai sebuah notasi grafis yang *relative* sudah dibakukan (*open standard*) dan dikontrol oleh OMG (*Object Management Group*) mungkin lebih dikenal sebagai badan yang berhasil membakukan CORBA (*Common Object Request Broker Architecture*), UML menawarkan banyak keistimewaan. UML tidak hanya dominan dalam penotasian di lingkunan OO tetapi juga popular di luar lingkungan

OO. Paling tidak ada tiga karakter penting yang melekat di UML yaitu sketsa, cetak biru dan bahasa pemograman. Sebagai sebuah sketsa, UML bisa berfungsi sebagai jembatan dalam mengkomunikasikan beberapa aspek dari sistem. UML pula digunakan sebagai landasan utama dalam pembuatan aplikasi yang memiliki anggota sehingga dalam proses pembuatan memiliki pandangan yang dalam pembangunan perangkat lunak tersebut, Dengan demikian semua anggota tim akan mempunyai gambaran yang sama tentang suatu sistem. UML bisa juga berfungsi sebagai sebuah cetak biru karena sangat lengkap dan detil. Dengan cetak biru ini maka akan bisa diketahui informasi detail tentang koding program (*forward engineering*) atau bahkan membaca program dan menginterpretasikannya kembali kedalam diagram (*reverse engineering*). *Reverse engineering* sangat berguna pada situasi dimana code program yang tidak terdokumentasi asli hilang atau bahkan belum dibuat sama sekali. Sebagai bahasa pemograman, UML dapat menterjemahkan diagram yang ada di UML menjadi *code* program yang siap untuk di jalankan.

Struktur sebuah sistem dideskripsikan dalam 5 *view* dimana salah satu diantaranya *scenario*, *scenario* ini memegang peran khusus untuk mengintegrasikan *content* ke *view* yang lain. Kelima *view* tersebut berhubungan dengan diagram yang dideskripsikan di UML. Setiap *view* berhubungan dengan perspektif tertentu di mana sistem akan diuji. *View* yang berbeda akan menekankan pada aspek yang berbeda dari sistem yang mewakili ketertarikan sekelompok *stakeholder* tertentu. Penjelasan lengkap tentang sistem bisa dibentuk dengan menggambungkan informasi-informasi yang ada pada kelima *view* tersebut sebagai berikut :

1. *Scenario*, menggambarkan interaksi diantara objek dan diantara proses. *Scenario* ini digunakan untuk dientifikasi elemen arsitektur, ilustrasi dan validasi disain arsitektur serta sebagai titik awal untuk pengujian prototype arsitektur. *Scenario* ini bisa juga disebut dengan *use case view*. *Use case view* ini mendefinisikan kebutuhan sistem karena mengandung semua *view* yang lain yang mendeskripsikan aspek – aspek tertentu dari rancangan

sistem. Itulah sebabnya *use case view* menjadi pusat peran dan sering dikatakan yang menjalakan proses pengembangan perangkat lunak.

2. *Development View*, menjelaskan sebuah sistem dari perspektif programmer dan terkonsentrasikan ke manajemen perangkat lunak. *View* ini dikenal juga sebagai *implementation view*. Diagram UML yang termasuk dalam development *view* di antaranya adalah *component diagram* dan *package diagram*.
3. *Logical View*, terkait dengan fungsionalitas sistem yang dipersiapkan untuk pengguna akhir. *Logical view* mendeskripsikan struktur logika yang mendukung fungsi-fungsi yang dibutuhkan di *use case*. Desain *view* ini berisi *object diagram*, *class diagram*, *state machine diagram* dan *composite structure diagram*.
4. *Physical View*, menggambarkan sistem dari perspektif *sistem engineer*. Fokus dari *physical view* adalah *topologi* sistem perangkat lunak. *View* ini dikenal juga sebagai *deployment view*. Yang termasuk dalam *physical* view ini adalah *deployment diagram* dan *timing diagram*.
5. *Process View*, berhubungan erat dengan aspek dinamis dari sistem, proses yang terjadi di sistem dan bagaimana komunikasi yang terjadi di sistem serta tingkah laku sistem saat dijalankan. *Process view* menjelaskan apa itu *concurrency*, *distribusi integrasi*, *kinerja* dan lain-lain. Yang termasuk dalam process *view* adalah *activiy diagram*, *communication diagram*, *sequence diagram* dan *interaction overview diagram*.

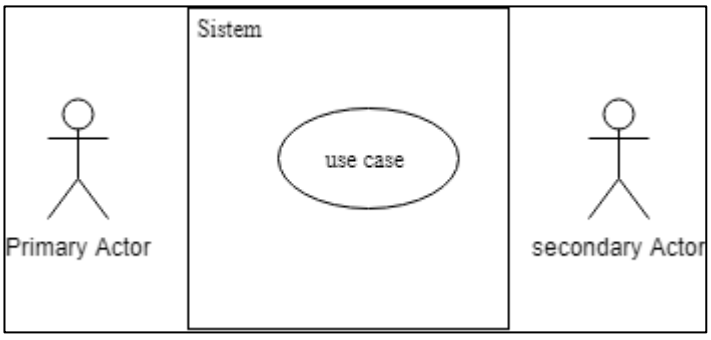
Mesikpun UML sudah cukup banyak menyediakan diagram yang bisa membantu mendefinisikan sebuah aplikasi, tidak berarti bahwa semua diagram tersebut akan bisa menjawab persoalan yang ada. Dalam banyak kasus, diagram lain selain UML sangat banyak membantu. Pada penelitian ini konsep UML digunakan untuk menggambarkan bagaimana sistem bekerja, hubungan antar *class*, memberi gambaran kepada *user* sistem yang akan di gunakan dan memberikan penjelasan detail pemanggilan fungsi-fungsi atau *method* dalam *class*.

2.17.1 Diagram *Unified Modeling Language*

UML menyediakan 4 macam diagram untuk memodelkan aplikasi perangkat lunak berorientasi objek . yaitu:

1. ***Use Case Diagram***

   Use case diagram adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara mendeskripsikan tipikal interaksi antara user (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan antara pengguna dan sistem disebut *scenario*. Setiap *scenario* mendeskripsikan urutan kejadian [23]. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatan *use case* adalah serangkaian skenario yang digabungkan Bersama – sama oleh tujuan umum pengguna. Diagram use case menunjukan 3 aspek dari sistem yaitu : *actor*, *use case* dan *sistem* atau *sub sistem boundary*. *Actor* mewakili peran orang, sistem yang lain atau alat ketika berkomunikasi dengan *use case*. Gambar 2.13 Konsep *Use Case Diagram* menunjukan *Use Case Diagram* dalam UML.



Gambar 2.13 Konsep *Use Case Diagram* [23]

Berikut ini adalah bagian dari sebuah *use case* diagram :

a. Use Case

Use case adalah abstraksi dari interaksi antarasistem dengan *actor*. Oleh karena itu sangat penting untuk memilih abstraksi yang cocok. *Use case* dibuat berdasarkan keperluan *actor*. *Use case* harus merupakan 'apa' yang dikerjakan *software* aplikasi, bukan 'bagaimana' *software* aplikasi mengerjakannya. Setiap *user case* harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaksinya dengan *actor*. Nama *use case* boleh terdiri dari beberapa kata dan tidak boleh ada dua *use case* yang memiliki nama yang sama.

b. *Actors*

Actors adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Bahwa *actor* berinteraksi dengan *use case*, tetapi tidak memiliki kendali atas *use case*.

c. *Relationship*

Relationship adalah hubungan antara *use cases* dengan *actors*. *Relationship* dalam *use case Diagram* meliputi:

a. Asosiasi antara *actor* dan *use case*

Hubungan antara *actor* dan *use case* yang terjadi karena adanya interaksi antara kedua belah pihak. Asosiasi tipe ini menggunakan garis lurus dari *actor* menuju *use case* baik dengan menggunakan mata panah terbuka ataupun tidak.

b. Asosiasi antara 2 *use case*

Hubungan antara *use case* yang satu dan *use case* lainnya yang terjadi karena adanya interaksi antara kedua belah pihak. Asosiasi tipe ini menggunakan garis putus-putus/garis lurus dengan mata panah terbuka di ujungnya.

c. Generalisasi antara 2 *actor*

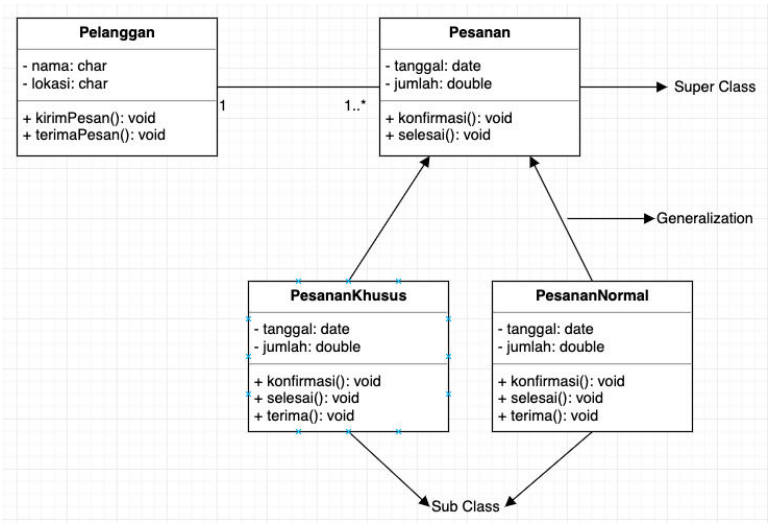
Hubungan *inheritance* (pewarisan) yang melibatkan *actor* yang satu (*the child*) dengan *actor* lainnya (*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.

d. Generalisasi antara 2 *use case*.

Hubungan *inheritance* (pewarisan) yang melibatkan *use case* yang satu (*the child*) dengan *use case* lainnya(*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.

2. *Class Diagram*

Class diagram adalah diagram statis. Ini mewakili pandangan statis dari suatu aplikasi. *Class Diagram* tidak hanya digunakan untuk memvisualisasikan, menggambarkan, dan mendokumentasikan berbagai aspek sistem tetapi juga membangun kode eksekusi dari aplikasi perangkat lunak. *Class diagram* menggambarkan *atribut*, *operation* dan juga *constraint* yang terjadi pada sistem. *Class diagram* banyak digunakan dalam pemodelan sistem *Object Oriented* karena mereka adalah satu-satunya diagram UML, yang dapat dipetakan langsung dengan bahasa berorientasi objek. *Class* diagram menunjukan koleksi *Class*, antarmuka, asosiasi, kolaborasi, dan *constraint*. Dikenal juga sebagai diagram struktural. Gambar 2.14 *Class Diagram*menunjukan *Class diagram* dalam UML.



Gambar 2.14 *Class Diagram*

Class diagram mempunyai 3 relasi dalam penggunaanya, yaitu :

a. *Assosiation*

Assosiation adalah sebuah hubungan yang menunjukkan adanya interaksi antar *class*. Hubungan ini dapat ditunjukkan dengan garis dengan mata panah terbuka di ujungnya yang mengindikasikan adanya aliran pesan dalam satu arah antara *class* dan *actor*.

b. *Generalization*

Generalization adalah sebuah hubungan antar *class* yang bersifat dari khusus ke umum pada umumnya digunakan untuk penggunaan *actor* dan *class.*

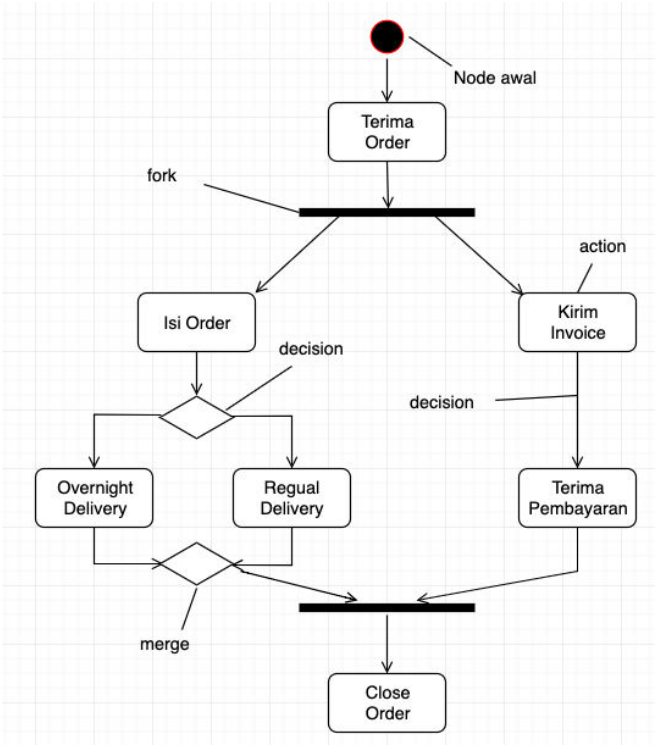
c. *Constraint*

Constraint adalah sebuah hubungan yang digunakan dalam sistem untuk memberi batasan pada sistem sehingga didapat aspek yang tidak fungsional.

3. *Activity Diagram*

Activity diagram adalah bagian penting dari UML yang menggambarkan aspek dinamis dari sistem. Logika *procedural*, proses bisnis dan aliran kerja suatu bisnis bisa dengan mudah dideskripsikan

dalam *activity diagram*. *Activity diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaanya dengan *flowchart* adalah *activity diagram* bisa mendukung perilaku pararel sedangkan *flowchart* tidak bisa. Tujuan dari *activity* diagram adalah untuk menangkap tingkah laku dinamis dari sistem dengan cara menunjukan aliran pesan dari satu aktifitas ke aktifitas lainya. Sebelum menggambarkan sebuah *activity* diagram, perlu adanya pemahaman yang jelas tentang elemen yang akan digunakan di activity diagram. Elemen utama dalam *activity* diagram adalah aktivitas itu sendiri. Aktivitas adalah fungsi yang dilakukan oleh sistem setelah akitivitas terindentifikasi,. Gambar 2.15 *Activity Diagram* menunjukan *Activity Diagram* dalam UML.



Gambar 2.15 ***Activity Diagram***

Berikut ini merupakan komponen dalam *activity diagram*, yaitu :

a. *Activity node*

Activity node menggambarkan bentuk notasi dari beberapa proses yang beroperasi dalam kontrol dan nilai data.

b. *Activity edge*

Activity edge menggambarkan bentuk *edge* yang menghubungkan aliran aksi secara langsung ,dimana menghubungkan *input* dan *output* dari aksi tersebut.

c. *Initial state*

Bentuk lingkaran berisi penuh melambangkan awal dari suatu proses.

d. *Decision*

Bentuk wajib dengan suatu *flow* yang masuk beserta dua atau lebih *activity node* yang keluar. *Activity node* yang keluar ditandai untuk mengindikasikan beberapa kondisi.

e. *Fork*

Satu bar hitam dengan satu *activity node* yang masuk beserta dua atau lebih *activity node* yang keluar.

f. *Join*

Satu bar hitam dengan dua atau lebih *activity node* yang masuk beserta satu *activity node* yang keluar, tercatat pada akhir dari proses secara bersamaan. Semua *actions* yang menuju *join* harus lengkap sebelum proses dapat berlanjut.

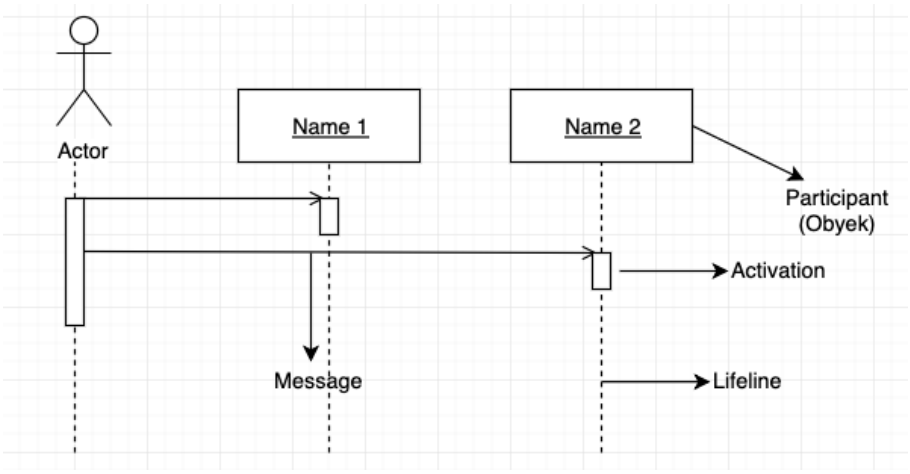
g. *Final state*

Bentuk lingkaran berisi penuh yang berada di dalam lingkaran kosong, menunjukkan akhir dari suatu proses.

4. *Sequence Diagram*

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Diagram ini menunjukan sejumlah contoh obyek dan *message* (pesan) yang diletakan diantara obyek-obyek ini di dalam *use case*. Komponen utama *sequence diagram* terdiri atas objek yang ditulisan dengan kotak segiempat bernama. *Message,* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukan dengan *progress vertical*. Objek diletakan di detak bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan untuk

menyederhanakan diagram, istilah objek dikenal juga dengan *participant*, setiap *participant* terhubung dengan garis titik-titik yang disebut *lifeline*. Sepanjang *lifeline* ada kotak yang disebut *activation*, *activation* mewakili sebuah eksekusi operasi dari *participant*. Panjang kotak ini berbanding lurus dengan durasi *activation*. Sebuah *message* bisa jadi *simple*, *synchronous* atau *asynchronous*. *Message* yang *simple* adalah sebuah perpindahan (*transfer*) *control* dari satu *participant* ke *participant* yang lainnya. Jika sebuah *participant* mengirimkan sebuah *message synchronous*, maka jawaban atas *message* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *message asynchronous* yang dikirimkan, maka jawaban atas *message* tersebut tidak perlu ditunggu. *Time* adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas ke bawah. *Message* yang lebih dekat dari atas akan dijadikan terlebih dahulu dibanding *message* yang lebih dekat ke bawah. Gambar 2.16 *Sequence Diagram* menunjukan *sequence diagram* dalam UML.



Gambar 2.16 *Sequence Diagram*

Berikut ini merupakan komponen dalam *sequence diagram* :

a. *Activations*

Activations menjelaskan tentang eksekusi dari fungsi yang dimiliki oleh suatu objek.

b. *Actor*

Actor menjelaskan tentang peran yang melakukan serangkaian aksi dalam suatu proses.

c. *Collaboration boundary*

Collaboration boundary menjelaskan tentang tempat untuk lingkungan percobaan dan digunakan untuk memonitor objek.

d. *Parallel vertical lines*

Parallel vertical lines menjelaskan tentang suatu garis proses yang menunjuk pada suatu *state*.

e. *Processes*

Processes menjelaskan tentang tindakan/aksi yang dilakukan oleh aktor dalam suatu waktu.

f. *Window*

Window menjelaskan tentang halaman yang sedang ditampilkan dalam suatu proses.

g. *Loop*

Loop menjelaskan tentang model logika yang berpotensi untuk diulang beberapa kali.