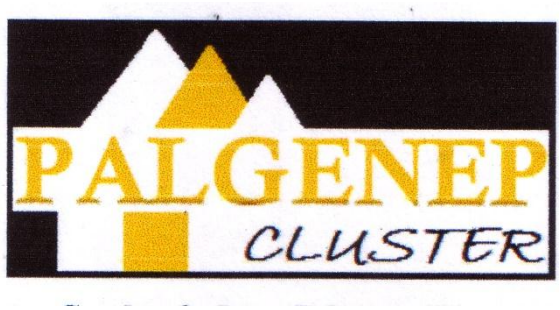
BAB II
LANDASAN TEORI

1.1. Sejarah Palgenep Cluster

Palgenep *Cluster* selesai dibangun pada tanggal 3 Februari 2015 dengan jumlah hunian 14 rumah dengan tipe 40/70 dengan konsep rumah modern minimalis, awal mulanya tanah yang dibangun Palgenep *Cluster* ini merupakan sebuah tanah milik warga setempat, karena dirasa tempat tersebut cukup strategis seperti dekat dengan pusat olah raga, pusat niaga, sarana kesehatan, sarana pendidikan dan akses ketempat tersebut pun banyak jalur alternatifnya sehingga oleh pihak Palgenep *Cluster* tanah tersebut dibeli kemudian dibangunlah sebuah hunian *cluster* yang menghasilkan hunian yang nyaman dan modern minimalis.

1.1.1. Logo Palgenep Cluster

Gambar 2.1 adalah logo dari Palgenep *Cluster* yang sering digunakan



Gambar 2.1 Logo Palgenep *Cluster*

1.1.2. Visi Dan Misi Palgenep Cluster

Adapun visi dan misi dari Palgenep *Cluster* adalah sebagai berikut:

1.1.2.1.Visi

Membangun hunian yang nyaman dan aman serta berkomitmen penuh memberikan nilai yang optimal bagi seluruh pelanggan, rekan usaha dan masyarakat

1.1.2.2.Misi

Adapun misi dari Palgenep Cluster adalah:

a. Membangun hunian yang bernilai tambah bagi kepuasan pelanggan.

b. Memenuhi kebutuhan masyarakat akan hunian dan area komersial berkualitas.
c. Memberikan harga yang berkompetitif dengan kualitas hunian yang baik.
d. Terdepan dalam disain arsitektur.
e. Kesempurnaan pelayanan dan kepuasan total pelanggan
f. Memberikan rasa aman kepada pelanggan.
g. Meningkatkan kosntribusi positif kepada masyarakat dan lingkungan.

2.2. Landasan Teori

Landasan teori merupakan penjelasan berbagai konsep dasar dan teori – teori yang berkaitan dalam pembangunan sistem *monitoring* dan kontrol alat elektronik menggunakan *internet of things*. Beberapa teori terkait dengan pembangunan perangkat ini adalah konsep yang digunakan perangkat lunak. Perangkat keras dan bahasa peograman yang dibutuhkan dalam pembangunan sistem *monitoring* dan kontrol alat elektronik menggunakan *internet of things*.

2.2.1. Jaringan Komputer

Menurut definisi, jaringan komputer merupakan sejumlah komputer yang dapat saling berkomunikasi. Dalam komunikasi ini dapat terjadi perpindahan data ataupun berbagi sumber daya. Dalam skala luas, internet juga merupakan jaringan komputer. Jadi, suatu jaringan komputer tidak hanya terjadi pada sejumlah komputer yang terdapat pada suatu ruangan ataupun gedung atau perusahaan. Pada dasarnya tekonologi jaringan komputer itu sendiri merupakan perpaduan antara teknologi dan juga teknologi komunikasi [**5**].

2.2.2. Internet of Things

Internet of Things (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 termasuk juga pada benda dunia nyata. Bahan pangan, elektronik, peralatan apa saja, koleksi,

termasuk benda hidup, yang semuanya tersambung ke jaringan lokal dan global melalui sensor tertanam dan selalu "*on*" [**2**].

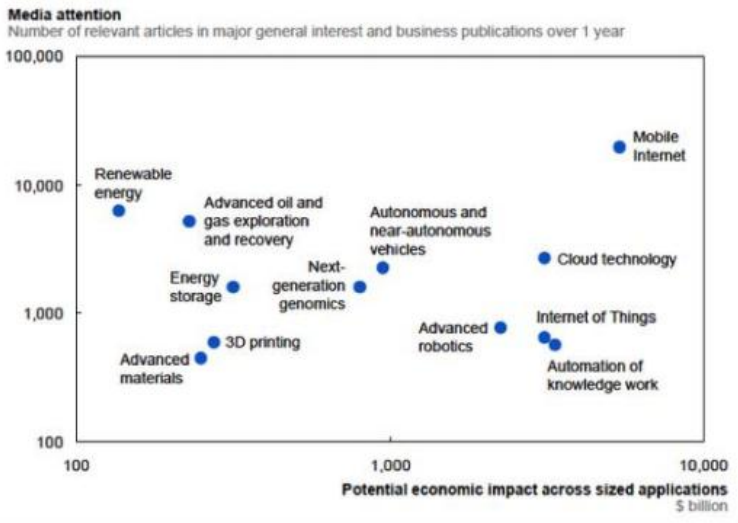
Interaksi antara manusia dengan manusia sudah sangat biasa sejak zaman dahulu kala. Interaksi antara manusia dengan mesin sudah biasa pula, sejak adanya penemuan teknologi seperti komputer atau gadget *devices* lainnya. Namun, Menurut analisa McKinsey Global Institute, *internet of things* adalah sebuah teknologi yang memungkinkan kita untuk menghubungkan mesin, peralatan, dan benda fisik lainnya dengan sensor jaringan dan aktuator untuk memperoleh data dan mengelola kinerjanya sendiri, sehingga memungkinkan mesin untuk berkolaborasi dan bahkan bertindak berdasarkan informasi baru yang diperoleh secara independen. Sebuah publikasi mengenai *Internet of things in 2020* menjelaskn bahwa *internet of things* adalah suatu keadaan ketika benda memiliki identitas, bisa beroperasi secara intelijen, dan bisa berkomunikasi dengan sosial, lingkungan, dan pengguna.

Dapat kita simpulkan bahwa *internet of things* membuat suatu koneksi antara mesin dengan mesin, sehingga mesin-mesin tersebut dapat berinteraksi dan bekerja secara independen sesuai dengan data yang diperoleh dan diolahnya secara mandiri. Tujuannya adalah untuk membuat manusia berinteraksi dengan benda dengan lebih mudah, bahkan supaya benda juga bisa berkomunikasi dengan benda lainnya.

Teknologi *internet of things* sangat luar biasa. Jika sudah direalisasikan, teknologi ini tentu akan sangat memudahkan pekerjaan manusia. Manusia tidak akan perlu lagi mengatur mesin saat menggunakannya, tetapi mesin tersebut akan dapat mengatur dirinya sendiri dan berinteraksi dengan mesin lain yang dapat berkolaborasi dengannya. Hal ini membuat mesin-mesin tersebut dapat bekerja sendiri dan manusia dapat menikmati hasil kerja mesin-mesin tersebut tanpa harus repot-repot mengatur mereka.

Cara kerja dari *internet of things* cukup mudah. Setiap benda harus memiliki sebuah IP Address. IP Address adalah sebuah identitas dalam jaringan yang membuat benda tersebut bisa diperintahkan dari benda lain dalam jaringan yang sama. Selanjutnya, IP address dalam benda-benda tersebut akan dikoneksikan ke jaringan

internet. Saat ini, koneksi internet sudah sangat mudah kita dapatkan. Dengan demikian, kita dapat memantau benda tersebut bahkan memberi perintah kepada benda tersebut. Sebagai contoh, jika ada speaker yang memiliki IP address dan terkoneksi internet di Amerika Serikat, kita dapat memerintahkan speaker tersebut untuk menyalakan musik walaupun kita berada di Indonesia. Yang kita perlukan hanyalah koneksi internet. Lalu apa hubungannya dengan *internet of things*, Setelah sebuah benda memiliki IP address dan terkoneksi dengan internet, pada benda tersebut juga dipasang sebuah sensor. Sensor pada benda memungkinkan benda tersebut memperoleh informasi yang dibutuhkan. Setelah memperoleh informasi, benda tersebut dapat mengolah informasi itu sendiri, bahkan berkomunikasi dengan benda-benda lain yang memiliki IP address dan terkoneksi dengan internet juga. Akan terjadi pertukaran informasi dalam komunikasi antara benda-benda tersebut. Setelah pengolahan informasi selesai, benda tersebut dapat bekerja dengan sendirinya, atau bahkan memerintahkan benda lain juga untuk ikut bekerja. Jadi, dalam *internet of things* manusia akan bertindak sebagai raja dan akan dilayani oleh benda-benda disekitarnya.



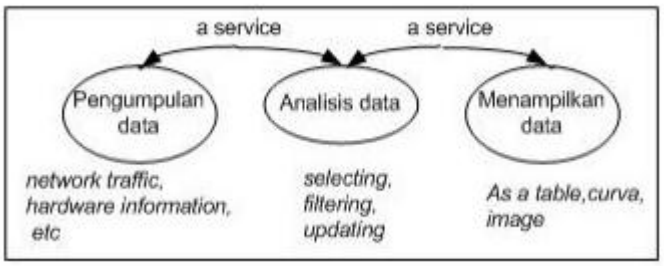
Gambar 2.2 Hubungan Potensi Ekonomi

Menurut McKinsey Global Institute, dalam sebuah artikel yang ditullis oleh Michael Hendrix, *internet of things* akan menjadi salah satu teknologi yang akan gencar di masa depan. Institusi ini memprediksi bahwa peluang pengaruh ekonomi dari *internet of things* akan mencapai 2.7 sampai 6.2 triliun Dolar Amerika Serikat. Pengaruh ekonomi dari industri *internet of things* menempati posisi ketiga terbesar setelah *mobile internet* dan *automation of knowledge work*. Hal ini membuktikan bahwa suatu hari nanti teknologi *internet of things* benar-benar akan berkembang dan menjadi tren di dunia [**6**].

2.2.3. Sistem *Monitoring*

Sistem *monitoring* merupakan suatu proses untuk mengumpulkan data dari berbagai sumber daya. Biasanya data yang dikumpulkan merupakan data yang real time. Secara garis besar tahapan dalam sebuah sistem *monitoring* terbagi ke dalam tiga proses besar seperti yang terlihat pada Gambar 2.3, yaitu:

1. proses di dalam pengumpulan data *monitoring.*
2. proses di dalam analisis data *monitoring.*
3. proses di dalam menampilkan data hasil *montoring*



Gambar 2.3 Proses Sistem *Monitoring*

Aksi yang terjadi di antara proses-proses dalam sebuah sistem *monitoring* adalah berbentuk *service*, yaitu suatu proses yang terus-menerus berjalan pada interval waktu tertentu. Proses-proses yang terjadi pada suatu sistem *monitoring* dimulai dari pengumpulan data seperti data dari *network trafic*, *hardware information*, dan lain-lain yang kemudian data tersebut dianalisis pada proses analisis data dan pada akhirnya data tersebut akan ditampilkan. Pada beberapa aplikasi sistem *monitoring*, akses benar-benar dibatasi dari local host terminal saja. Pertanyaannya

apakah bisa dilakukan *monitoring* dari jarak jauh, dimana semua data yang dikumpulkan dari terminal komputer yang 8 berada di lokasi berbeda dengan instrumennya misalnya dengan menggunakan jaringan LAN (Local Area Network) atau bahkan internet.

2.2.4. Sistem Kontrol

Sistem kendali / kontrol telah memegang peranan yang sangat penting dalam perkembangan ilmu dan teknologi. Peranan sistem kendali meliputi semua bidang kehidupan. Dalam peralatan, misalnya proses pada industry pesawat terbang, peluru kendali, pesawat ruang angkasa, dan lain-lain. Sedangkan dalam bidang non teknis meliputi bidang biologi, ekonomi, social, kedokteran, dan lainlain. Sistem kendali yang semakin berkembang dapat meningkatkan kinerja sistem, kualitas produksi, dan menekan biaya produksi. Sistem kendali dapat dikatakan sebagai hubungan antara komponen yang membentuk sebuah konfigurasi sistem, yang akan menghasilkan tanggapan sistem yang diharapkan Jadi harus ada yang dikendalikan, yang merupakan suatu sistem fisis, yang biasa disebut dengan kendalian.

2.2.5. Mikrokontroler

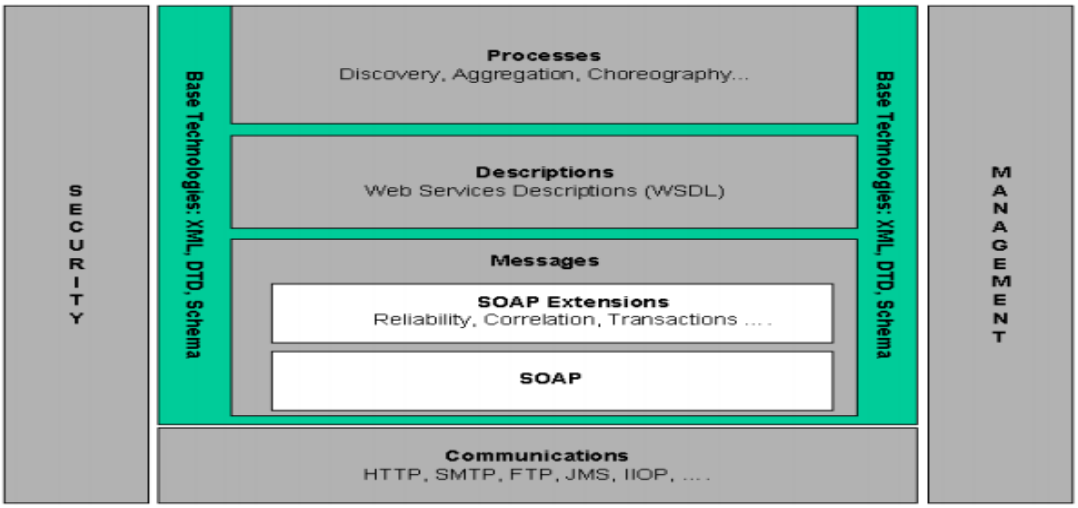
Mikrokontroler adalah sebuah sistem komputer fungsional dalam sebuah chip. Di dalamnya terkandung sebuah prosesor, memori (sejumlah kecil RAM, memori program, atau keduanya), dan perlengkapan input-output. Seperti umumnya komputer, mikrokontroler adalah alat yang mengerjakan instruksi-instruiksi yang diberikan kepadanya. Artinya, bagian terpenting dan utama dari suatu sistem terkoumputerisasi adalah program itu sendiri yang dibuat oleh seorang programmer [**7**].

2.2.6. Web Service

Web service adalah sebuah entitas komputasi yang dapat diakses melalui jaringan internet maupun intranet dengan standar protokol tertentu dalam platform dan antarmuka bahasa pemograman yang independen. Tujuan pengembangannya adalah untuk menjembatani komunikasi antar program, sehingga aplikasi yang satu

dan aplikasi yang lain yang terdapat pada suatu jaringan yang sama atau pada jaringan berbeda dapat saling berkomunikasi asalkan menggunakan standart protokol yang ditetapkan oleh *web service*. Hal ini bisa terjadi, karena standart protokol itu tidaklah terikat pada suatu platform atau bahasa pemograman.

Protokol itu sendiri dibangun oleh *Extensible Markup Language* (XML) yang memang kenyataannya telah didukung oleh banyak paltform, bahasa pemograman, dan oleh *developer* di seluruh dunia. Arsitektur *web service* dibangun oleh beberapa layer dan teknologi yang saling berhubungan. Banyak cara untuk memvisualisaskan *service*, sama banyaknya dengan cara untuk membangun dan menggunakan *web service* gambar berikut hanyalah salah satu cara untuk menggambarkan arsitekturnya.



Gambar 2.4 Arsitektur Web Service

Web service disusun oleh tiga komponen standart, yaitu :

1. *Simple Object Access Protocol* (SOAP), yaitu protokol yang bertanggungjawab dalam pertukuran informasi dalam lingkungan jaringan terdistribusi.

2. *Web service definition Language* (WDSL), dokumen standart yang dituliskan pada format XML, dan mendefinikasan kehadiran *web service* dalam suatu jaringan.
3. Universal description, Discovery and Inttegration(UDDI), yaitu suatu lokasi direktori yang berisikan *service* (layanan) dan bersifat bebas *platform* (*platform independent*), dituliskan berbasis XML dan dapat diakses oleh entitas yang berada di dalam dan luar jaringan.

   Adanya standart tersebut membuat *web service* mudah di akses melalui berbagai antarmuka dan juga memberi peluang dimungkinkannya berbagai sistem yang dibangun pada platform yang berbeda dan bahasa yang berbeda untuk berkolaborasi dalam suatu pekerjaan.

2.2.7. Android

Android adalah sistem operasi yang berbasis Linux untuk telepon seluler seperti telepon pintar (*Smartphone*) dan komputer tablet. Android menyediakan *platform* terbuka bagi para pengembang untuk menciptakan aplikasi mereka sendiri untuk digunakan oleh bermacam peranti begerak. Awalnya, Google Inc. membeli Android Inc., pendatang baru yang membuat peranti lunak untuk ponsel. Kemudian untuk mengembangkan Android, dibentuklah Open Handset Alliance, konsorsium dari 34 perusahaan piranti keras, piranti lunak dan telekomunikasi, termasuk Google, HTC, Intel, Motorola, Qualcomm, T-Mobile dan Nvidia.

Bebrapa pengertian lain dari Android, yaitu :

1. Merupakan platform terbuka (*Open source*) bagi para programmer untuk membuat atau mengembangkan aplikasi.
2. Merupakan sistem operasi yang dibeli Google Inc dari Android Inc.
3. Bukan bahasa pemrograman, akan tetapi hanya menyediakan lingkungan hidup atau runtime environment yang disebut DVM (*Dalvik Virtual Machine*) yang telah dioptimasi untuk *device* dengan sistem memori yang kecil.

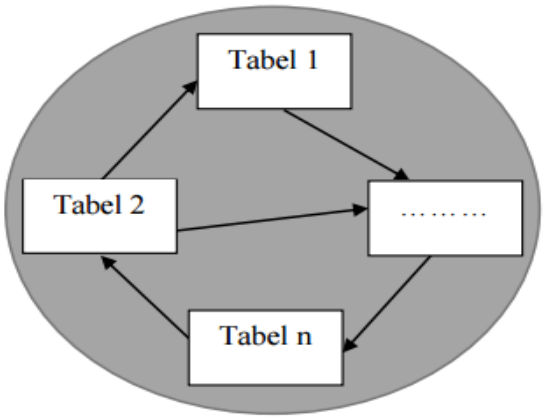
2.2.7.1.Sejarah Android

Pada Juli 2000, Google bekerjasama dengan Android Inc., perusahaan yang berada di Palo Alto, California Amerika Serikat. Para pendiri Android Inc. bekerja pada Google, diantaranya Andy Rubin, Rich Miner, Nick Sears, dan Chris White. Saat itu banyak yang menganggap fungsi Android Inc. Hanyalah sebagai perangkat lunak pada telepon seluler. Sejak saat itu muncul rumor bahwa Google hendak memasuki pasar telepon seluler. Di perusahaan Google, tim yang dipimpin Rubin bertugas mengembangkan program perangkat seluler yang didukung oleh kernel Linux. Hal ini menunjukkan indikasi bahwa Google sedang bersiap menghadapi persaingan dalam pasar telepon seluler. Versi android terbaru yaitu versi 4.0. (Ice Cream Sandwitch). Android juga sudah bergabung dengan beberapa smart mobile seperti LG, Samsung, Sony Ericsson, dan lainnya. Sekitar September 2007 sebuah studi melaporkan bahwa Google mengajukan hak paten aplikasi telepon seluler (akhirnya Google mengenalkan Nexus One, salah satu jenis telepon pintar GSM yang menggunakan Android pada sistem operasinya. Telepon seluler ini diproduksi oleh HTC Corporation dan tersedia di pasaran pada 5 Januari 2010). 13 Pada 9 Desember 2008, diumumkan anggota baru yang bergabung dalam program kerja Android ARM Holdings, Atheros Communications, diproduksi oleh Asustek Computer Inc, Garmin Ltd, Softbank, Sony Ericsson, Toshiba Corp, dan Vodafone Group Plc. Seiring pembentukan Open Handset Alliance, OHA mengumumkan produk perdana mereka, Android, perangkat bergerak (*Mobile*) yang merupakan modifikasi kernel Linux 2.6. Sejak Android dirilis telah dilakukan berbagai pembaruan berupa perbaikan bug dan penambahan fitur baru.

2.2.8. Basis Data

Sistem basis data adalah sistem terkomputerisasi yang tujuan utamanya adalah memelihara data yang sudah diolah atau informasi dan membuat informasi tersedia saat dibutuhkan. Pada intinya basis data adalah media untuk menyimpan data agar dapat diakses dengan mudah dan cepat. Pada buku ini menggunakan basis data

relasional yang diimplementasikan dengan table-tabel yang saling memiliki relasi seperti pada gambar berikut.



Gambar 2.5 Gambaran Basis Data

Sistem informasi tidak dapat dipisahkan dengan kebutuhan akan basis data apapun bentuknya, entah berupa file teks ataupun *Database Management System* (DBMS). Kebutuhan basis data dalam sistem informasi meliputi:

1. Memasukan, menyimpan dan mengambil data.
2. Membuat laporan berdasarkan data yang telah disimpan.

Tujuan dari buatnya table-tabel disini adalah untuk menyimpan data ke dalam tabel-tabel agar mudah diakses. Oleh karena itu, untuk merancang tabel-tabel yang akan dibuat maka dibutuhkan pola piker penyimpanan data nantinya jika dalam bentuk baris-baris data (*record*) dimana setiap baris terdiri dari beberapa kolom.

2.2.9. Pemograman Berorintasi Objek

Metodologi berorientasi objek adalah suatu strategi pembangunan peralngkan lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek

merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientas objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek.

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekata terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini beragam dengan platform yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut. Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

1. Meningkatkan produktivitas Karena kelas dan objek yang ditemukan dalam suatu masalah masih dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).
2. Kecepatan pengembangan Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengodean.
3. Kemudahan pemeliharaan Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dan pola-pola yang mungkin sering berubah-ubah.
4. Adanya konsistensi Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.
5. Meningkatkan kualitas perangkat lunak

Karena pendekatan pengembangan lebih dekat dengan dunia nyata dan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

2.2.9.1.Konsep Dasar Berorientasi Objek

Pendekatan berorientasi objek merupakan suatu teknik atau cara pendekatan dalam melihat permasalahan dan sistem (sistem perangkat lunak, sistem informasi, atau sistem lainnya). Pendekatan berorientasi objek akan memandang sistem yang akan dikembangkan sebagai suatu kumpulan objek yang berkorespondensi dengan objek-objek dunia nyata. Ada banyak cara untuk mengabstraksikan dan memodelkan objek-objek tersebut, mulai dan abstraksi objek, kelas, hubungan antar kelas sampai abstraksi sistem. Saat mengabstraksikan dan memodelkan objek, data dan proses-proses yang dipunyai oleh objek akan dienkapsulasi (dihubungkus) menjadi suatu kesatuan. Dalam rekayasa perangkat lunak, konsep pendekatan berorientasi objek dapat diterapkan pada tahap analisis, perancanggan, pemrograman, dan pengujian perangkat lunak. Ada berbagai teknik yang dapat digunakan pada masing-masing tahap tersebut, dengan aturan dan alat bantu pemodelan tertentu. Sistem berorientasi objek merupakan sebuah sistem yang dibangun dengan berdasarkan metode berorientasi objek adalah sebuah sistem yang komponennya dibungkus (dienkapsulasi) menjadi kelompok daata dan fungsi. Setiap komponen dalam sistem tersebut dapat mewarisi atribut dan sifat dan komponen lainnya, dan dapat berinteraksi satu sama lain. Berikut ini adalah beberapa konsep dasar yang harus dipahami tentang metodologi berorientasi objek:

1. Kelas (*Class*)

   Kelas adalah sekumpulan objek-objek dengan karakteristik yang sama. Kelas merupakan definisi statis dan himpunan objek yang sama yang mungkin lahir atau diciptakan dan kelas tersebut. Sebuah kelas akan mempunyai sifat (atribut), kelakuan (metode/operasi), hubungan (*relationship*) dan arti. Suatu kelas dapat diturunkan dan kelas yang lain, dimana atribut dan kelas semula dapat diwariskan ke kelas yang baru. Secara tekniks kelas adalah sebuah struktur dalam pembuatan perangkat lunak. Kelas merupakan bentuk struktur pada kode program yang menggunakan metodologi berorientasi objek.

2. Objek (*object*)

   Objek adalah abstraksi dari sesuatu yang mewakili dunia nyata benda, manusia, satua organisasi, tempat, kejadian, struktur, status, atau hal-hal lain yang bersifat abstra. Objek merupakan suatu entitas yang mampu menyimpan informasi (status) dan mempunyai operasi (kelakuan) yang dapat diterapkan atau dapat berpengaruh pada status objeknya. Objek mempunyai siklus hidup yaitu diciptakan, dimanipulasi, dan dihancurkan. Secara teknis, sebuah kelas saat program dieksekusi makan akan dibuat sebuah objek. Objek dilihat darisegi teknis adalah elemen pada saat runtime yang akan diciptakan, dimanipulasi, dan dihancurkan saat eksekusi sehinga sebuah objek hanya ada saat sebuah program dieksekusi. Jika masih dalam bentuk kode, disebut sebagai kelas jadi pada saat runtime (saat sebuah program dieksekusi), yang kita punya adalah objek, di dalam teks program yang kita lihat hanyalah kelas.

3. Metode (*method*)

   Operasi atau metode pada sebuah kelas hampir sama dengan fungsi atau prosedur pada metodologi struktural. Sebuah kelas boleh memiliki lebih dari satu metode atau operasi. Metode atau operasi yang berfungsi untuk memanipulasi objek itu sendiri. Operasi atau metode merupakan fungsi atau transformasi yang dapat dilakukan terhadap objek atau dilakukan oleh objek. Metode atau operasi dapat berasal dari *event*, aktifitas atau aksi keadaan, fungsi, atau kelakuan dunnia nyata. Contoh metode atau operasi misalnya *Read, Write*, *Move, Copy*, dan sebagainya.

4. Atribut (*attribute*)

   Atribut dari sebuah kelas adalah variabel global yang dimiliki sebuah kelas. Atribut dapat beruoa nilai atau elemen-elemen data yang dimiliki oleh objek dalam kelas objek. Atribut dipunyai secara individual oleh sebuah objek, misalnya berat, jenis, nama, dan sebagainya.

5. Abstraksi (*abstraction*)

   Prinsip untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana dengan mengabaikan aspek-aspek lain yang tidak sesuai dengan permasalahan.

6. Enkapsulasi (*encapsulation*)

   Pembungkusa atribut data dan layanan (operasi-operasi) yang dipunyai objek untuk menyembunyikan implementasi dan objek sehingga objek lain tidak mengetahui cara kerja-nya.

7. Pewarisan (*inheritance*)

   Mekanisme yang memungkinkan satu objek mewarisi sebagian atau seluruh definisi dan objek lain sebagai bagian dan dirinya.

8. Antarmuka (*interface*)

   Antarmuka sangat mirip dengan kelas, tapi tanpa atribut kelas dan memiliki metode yang dideklarasikan tanpa isi. Deklarasi metode pada sebuah interface dapat diimplementasikan oleh kelas lain.

9. *Reusability*

   Pemanfaatan kembali objek yang sudah didefinisikan untuk suatu permasalahan pada permasalahan lainnya yang melibatkan objek tersebut.

10. Generalisasi dan Spesialisasi

    Menunjukkan hubungan antara kelas dan objek yang umum dengan kelas dan objek yang khusus. Misalnya kelas yang lebih umum (generalisasi) adalah kendaraan darat dan kelas khususnya (spesialisasi) adalah mobil, motor, dan kereta.

11. Komunikasi Antarobjek

    Komunikasi antarobjek dilakukan lewat pesan (*message*) yang dikirim dari satu objek ke objek lainnya.

12. Polimorfisme (*polymorphism*)

Kemampuan suatu objek digunakan di banyak tujuan yang berbeda dengan nama yang sehingga menghemat baris program.

13. *Package*

    Package adalah sebuah kontainer atau kemasan yang dapat digunakan untuk mengelompokkan kelas-kelas sehingga memungkinkan beberapa kelas yang bernama sama disimpan dalam package yang berbeda [**8**].

2.2.9.2.Pengenalan UML

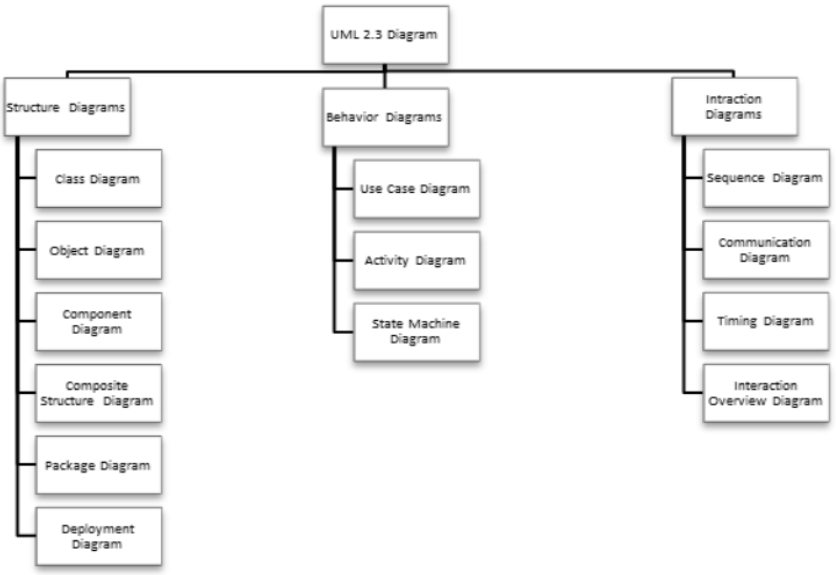
UML adalah sebuah bahasa yang telah menjadi standar dalam industri untuk menvisualisasi, menspesifikasi, merancang dan mendokumentasi sistem piranti lunak. Unified Modelling Language (UML) menawarkan sebuah standar untuk merancang model sebuah sistem. Tujuan UML adalah :

1. Memberikan model yang siap pakai, bahasa pemodelan visual yang ekspresif untuk mengembangkan dan saling menukar model dengan mudah dan dimengerti secara umum.
2. Memberikan bahasa pemodelan yang bebas dari berbagai bahasa pemrograman dan proses rekayasa.
3. Menyatukan praktik-praktik terbaik yang terdapat dalam pemodelan. UML menyediakan beberapa notasi dan artifact standar yang bisa digunakan sebagai alat komunikasi bagi para pelaku dalam proses analisis dan desain. Arti fact didalam UML didefinisikan sebagai informasi dalam bentuk yang digunakan atau dihasilkan dalam proses pengembangan perangkat. Contohnya adalah source code yang dihasilkan oleh proses pemrograman [**9**].

.

2.2.9.3.Diagram UML

Pada UML 2.2.9.1 terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada.



Gambar 2.6 Diagram UML

Berikut ini penjelasan singkat dari pembagian kategori tersebut :

1. Structure diagrams yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.
2. Behavior diagrams yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkaian perubahan yang terjadi pada sebuah sistem.
3. Interaction diagrams yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antar subsistem pada suatu sistem [**8**].

2.2.9.4.*Use Case* Diagram

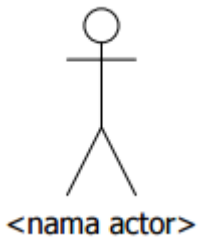
Use Case atau diagram *Use Case* merupakan pemodelan untuk kelakuan (*behavio*r) sistem informasi yang akan dibuat. *Use Case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *Use Case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi - fungsi itu.

Syarat penamaan pada *Use Case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *Use Case* yaitu pendefinisian apa yang disebut aktor dan *Use Case*.

2.2.9.4.1. *Actor*

Actor adalah sesuatu (entitas) yang berhubungan dengan sistem dan berpartisipasi dalam *use case*. *Actor* menggambarkan orang, sistem atau entitas *Eksternal* yang secara khusus membangkitkan sistem dengan input atau masukan kejadian-kejadian, atau menerima sesuatu dari sistem. *Actor* dilukiskan dengan peran yang mereka mainkan dalam *use case*, seperti *Staff*, Kurir dan lain-lain.

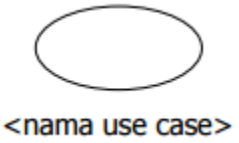
Dalam *use case* diagram terdapat satu aktor pemulai atau *initiator actor* yang membangkitkan rangsangan awal terhadap sistem, dan mungkin sejumlah aktor lain yang berpartisipasi atau participating *actor*. Akan sangat berguna untuk mengetahui siapa aktor pemulai tersebut [**8**].



Gambar 2.7 Bentuk *Actor* dalam UML

2.2.9.4.2. *Use Case*

Use case yang dibuat berdasarkan keperluan aktor merupakan gambaran dari "apa" yang dikerjakan oleh sistem, bukan "bagaimana" sistem mengerjakannya. *Use case* diberi nama yang menyatakan apa hal yang dicapai dari interaksinya dengan aktor.

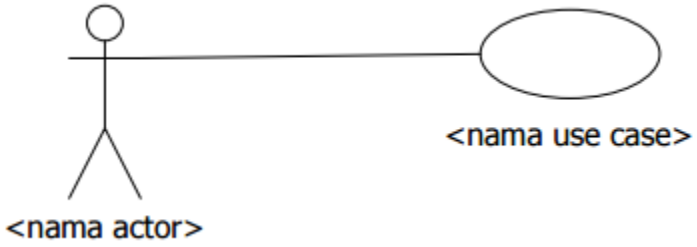


Gambar 2.8 Bentuk Bentuk *Use Csae*

2.2.9.4.3. *Relationship*

Relasi (*relationship*) digambarkan sebagai bentuk garis antara dua simbol dalam *use case* diagram. Relasi antara *actor* dan *use case* disebut juga dengan asosiasi (*association*). Asosiasi ini digunakan untuk menggambarkan bagaimana hubungan antara keduanya.

Relasi-relasi yang terjadi pada *use case* diagram bisa antara *actor* dengan *use case* atau *use case* dengan *use case*.



Gambar 2.9 Bentuk *Relationship* dalam UML

Relasi antara *use case* dengan *use case* :

1. *Include*, pemanggilan *use case* oleh *use case* lain atau untuk menggambarkan suatu *use case* termasuk di dalam *use case* lain (diharuskan). Contohnya adalah

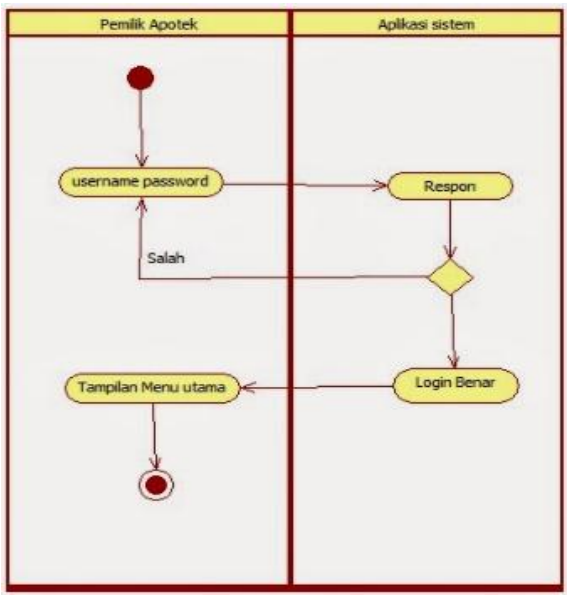
pemanggilan sebuah fungsi program. Digambarkan dengan garis lurus berpanah dengan tulisan <<*include*>>.

2. *Extend*, digunakan ketika hendak menggambarkan variasi pada kondisi perilaku normal dan menggunakan lebih banyak kontrol *form* dan mendeklarasikan *ekstension* pada use case utama. Atau dengan kata lain adalah perluasan dari *use case* lain jika syarat atau kondisi terpenuhi. Digambarkan dengan garis berpanah dengan tulisan <<*extend*>>.
3. *Generalization/Inheritance*, dibuat ketika ada sebuah kejadian yang lain sendiri atau perlakuan khusus dan merupakan pola berhubungan baseparent *use case*. Digambarkan dengan garis berpanah tertutup dari *base use case* ke parent *use case*.

2.2.9.5.*Activity* Diagram

Diagram aktivitas atau *activity* diagram menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut :

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan
2. Urutan atau pengelompokan tampilan dari sistem/user *interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya



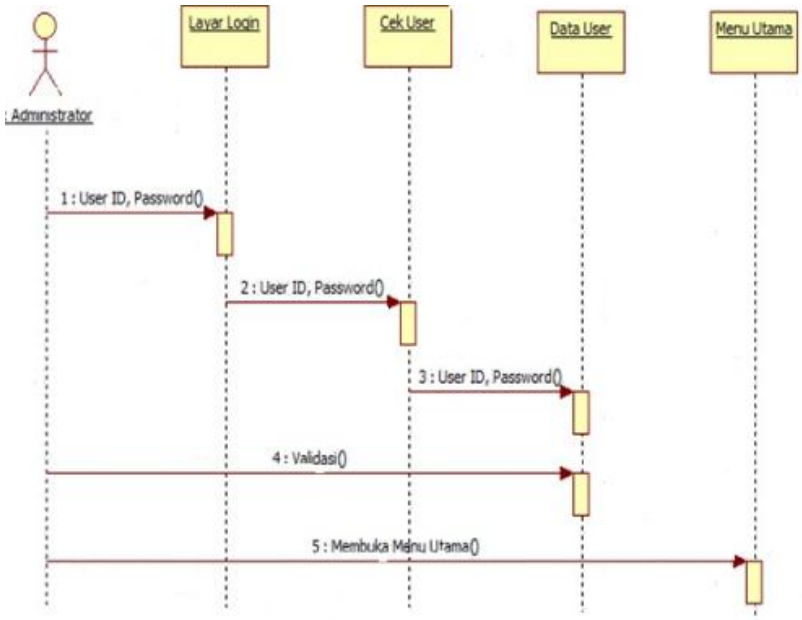
Gambar 2.10 Contoh *Activity* Diagram

2.2.9.6.*Sequence* Diagram

Diagram *sequence* menggambarkan kelakuan objek pada use case dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sequence maka harus diketahui objek-objek yang terlibat dalam sebuah use case beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambar adalah sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua use case yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

Penomoran pesan berdasarkan urutan interaksi pesan. Penggambaran letak pesan harus berurutan, pesan yang lebih atas dari lainnya adalah pesan yang berjalan terlebih dahulu.



Gambar 2.11 Contoh *Sequence* Diagram

2.2.9.7.*Class* Diagram

Diagram kelas atau *class* diagram menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Kelas-kelas yang ada pada struktur sistem harus dapat melakukan fungsifungsi sesuai dengan kebutuhan sistem. Susunan struktur kelas yang baik pada diagram kelas ebaiknya memiliki jenis-jenis kelas berikut :

1. Kelas Main Kelas yang memiliki fungsi awal dieksekusi ketika sistem dijalankan.
2. Kelas yang menangani tampilan sistem Kelas yang mendefinisikan dan mengatur tampilan ke pemakai.
3. Kelas yang diambil dari pendefinisian usecase Kelas yang menangani fungsi-fungsi yang harus ada diambil dari pendefinisian use case.

4. Kelas yang diambil dari pendefinisian data Kelas yang digunakan untuk memegang atau membungkus data menjadi sebuah kesatuan yang diambil maupun akan disimpan ke basis data. Jenis-jenis kelas diatas juga dapat digabungkan satu sama lain sesuai dengan pertimbangan yang dianggap baik asalkan fungsi-fungsi yang sebaiknya ada pada struktur kelas tetap ada. Susunan kelas juga dapat ditambahkan kelas utilitas seperti koneksi ke basis data, membaca file teks, dan lain sebagainya sesuai kebutuhan.

Dalam mendefinisikan metode yang ada di dalam kelas perlu memperhatikan apa yang disebut dengan cohesion dan coupling. Cohesion adalah ukuran seberapa dekat keterkaitan instruksi di dalam sebuah metode terkait satu sama lain sedangkan coupling adalah ukuran seberapa dekat keterkaitan instruksi antara metode yang satu dengan metode yang lain dalam sebuah kelas. Sebagai aturan secara umum maka sebuah metode yang dibuat harus memiliki kadar choesion yang kuat dan kadar coupling yang lemah.

Dalam *class* diagram terdapat beberapa relasi (hubungan antar *class*) yaitu:

1. *Generalization* dan *Inheritence*

   Diperlukan untuk memperlihatkan hubungan pewarisan (*inheritance*) antar unsur dalam diagram kelas. Pewarisan memungkinkan suatu kelas mewarisi semua atribut, operasi, relasi, dari kelas yang berada dalam hirarki pewarisannya.

2. *Associations*

   Hubungan statis antar class. Umumnya menggambarkan *class* yang memiliki atribut berupa *class* lain, atau class yang harus mengetahui ekstensi *class* lain. Dalam notasi UML kita mengenal asosiasi 2 arah (*bidirectional*) dan 1 arah (*undirectional*).

3. *Aggregation*

   Hubungan antar-*class* dimana *class* yang satu (*part class*) adalah bagian dari class lainnya (*whole class*).

4. *Composition*

Aggregation dengan ikatan yang lebih kuat. Di dalam composite aggregation, siklus hidup part class sangat bergantung pada whole class sehingga bila objek instance dari *whole class* dihapus maka objek instance dari part class juga akan terhapus.

5. *Depedency*

   Hubungan antar-*class* dimana sebuah *class* memiliki ketergantungan pada *class* lainnya tetapi tidak sebaliknya.

6. *Realization*

   Hubungan antar-*class* dimana sebuah *class* memiliki keharusan untuk mengikuti aturan yang ditetapkan *class* lainnya. Biasanya *realization* digunakan untk menspesifikasikan hubungan antara sebuah *interface* dengan *class* yang mengimplementasikan *interface* tersebut [**10**].

2.2.10. Cara Mengecek Daya

Apa kepanjangan dan fungsi dari MCB, kepanjangannya yaitu Miniature Circuit Breaker, fungsinya Mengamankan, membatasi serta memutuskan listrik di dalam rumah bila kelebihan beban, terjadi korsleting dan lain2. Listrik dari PLN yang masuk ke MCB memiliki daya yg besar dan ketika masuk ke dalam MCB akan di batasi sesuai dengan ukuran daya MCB tersebut

Bagaimana cara mengecek daya / watt listrik yang terpasang di rumah kita, caranya adalah lihat saklar listrik dari PLN yg terpasang di luar rumah kita



Rumus menghitung Watt / daya listrik rumah adalah Ampere x Volt jadi perhitungannya 6A X 220V = 1320 VA. Jadi daya listrik yg terpasang di rumah adalah 1300 (pembulatan dalam menyebut biar mudah). jika di rumah anda CL10 = 10 Ampere, jika CL4 = 4 Ampere, maka tinggal di kalikan 220 / 230V hasilnya adalah daya listrik yang terpasang.

Angka "6000" menunjukkan *rated breaking capacity* MCB, yaitu kemampuan kerja MCB masih baik sampai arus maksimal 6000A, yang biasanya terjadi saat hubung singkat arus listrik. Dimana diatas angka ini MCB akan berpotensi rusak. Dan angka "3" adalah I^2t*classification*, yaitu karakteristik energi maksimum dari arus listrik yang dapat melalui MCB [**11**].

2.2.11. Perangkat Keras

Perangkat keras adalah suatu perangkat yang dapat membantu mempermudah dalam membuat alat. Perangkat keras berupa perangkat berbentuk fisik. Perangkat keras yang digunakan antara lain adalah:

2.2.11.1. Mr Q7

Mikrokontroler Mr.Q7 merupan papan mikrokontroler Arduino Pro Mikro ATMEGA32u4 yang dibangun dengan modul ESP 8266 ESP-07 WiFi Wireless yang sudah support dengan wireless dan dirangkai dengan berbagai komponen lain sehingga memiliki fungsi yang hampir sama dengan arduino Uno.



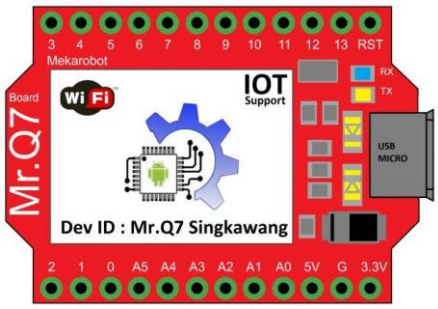
Gambar 2.12 Mikrokontroler MR Q7

Spesifikasi dari MR.Q7 antara lain sebagai berikut:

Tabel 2.1 Spesifikasi MR.Q7

Nama	Spesifikasi
Mikrokontroler	ATMEGA328
Operating Voltage	5V
Input Voltage	(recommended) 7-12V
Input Voltage	(limits) 6-20V
Digital I/O Pins	23
PWM	Channels 10
Analog Input	Channels 13
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB
SRAM	2.5 KB
EEPROM	1 KB
Clock Speed	16 MHz

Adapun arsitektur pin yang terdapat di MR Q7 bisa dilihat sebagai berikut:



Gambar 2.13 Skema Mikrokontroler MR Q7

2.2.10.2. Modul Relay

Prinsip Kerja Relay - Relay adalah Saklar (*Switch*) yang dioperasikan secara listrik dan merupakan komponen *Electromechanical* (Elektromekanikal) yang terdiri

dari 2 bagian utama yakni Elektromagnet (*Coil*) dan Mekanikal (seperangkat Kontak Saklar/*Switch*). Relay menggunakan Prinsip Elektromagnetik untuk menggerakkan Kontak Saklar sehingga dengan arus listrik yang kecil (*low power*) dapat menghantarkan listrik yang bertegangan lebih tinggi. Sebagai contoh, dengan Relay yang menggunakan Elektromagnet 5V dan 50 mA mampu menggerakan Armature Relay (yang berfungsi sebagai saklarnya) untuk menghantarkan listrik 220V 2A.

Features:

1. 4-Channel Relay breakout.
2. Power supply range from 5V~7.5V9 Onboard Photocoupler isolation.
3. Equiped with high-current relay, AC250V 10A ; DC30V 10A.
4. Relay Output Indicator LED.
5. TTL logic interface can be directly connected to microcontroller

2.2.10.3. Terminal Blok

Terminal Block adalah Suatu tempat berhentinya arus listrik sementara,yang akan dihubungkan ke komponen yang lain/Komponen Outgoing.Dalam Pembuatan panel listrik, TERMINAL BLOCK termasuk salah satu komponen utama. Sebab memiliki manfaat yang besar Didalam terminal ada *incoming* dan *Outgoing* yang fungsinya :

Incoming Adalah Konektor Arus Masuk dan *Outgoing* adalah Konektor Arus Keluar.

2.2.10.4. Power Supply

dalah sebagai alat atau perangkat keras yang mampu menyuplai tenaga atau tegangan listrik secara langsung dari sumber tegangan listrik ke tegangan listrik yang lainnya. *Power supply* biasanya digunakan untuk komputer sebagai penghantar tegangan listrik secara langsung kepada komponen-komponen atau perangkat keras lainnya yang ada di komputer tersebut, seperti hardisk, kipas, motherboard dan lain sebagainya. *Power supply* memiliki *input* dari tegangan yang berarus *alternating* current (AC) dan mengubahnya menjadi arus *direct current* (DC) lalu

menyalurkannya ke berbagai perangkat keras yang ada dikomputer kita. Karena memang arus *direct current* (DC)-lah yang dibutuhkan untuk perangkat keras agar dapat beroperasi, *direct current* biasa disebut juga sebagai arus yang searah sedangkan *alternating current* merupakan arus yang berlawanan.

2.2.11. Perangkat Lunak

Adapun beberapa perangkat lunak pendukung dalam penyusunan penelitian pembangunan sistem *monitoring* dan kontrol alat elektronik menggunakan *internet of things* ini adalah

2.2.11.1. Arduino Ide

Arduino IDE adalah *software* yang sangat canggih ditulis dengan menggunakan Java. IDE Arduino terdiri dari:

1. Editor program, sebuah window yang memungkinkan pengguna menulis dan mengeditprogram 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 didalam papan Arduino.

Sebuah kode program Arduino umumnya disebut dengan istilah *sketch.* Kata "*sketch*" digunakan secara bergantian dengan "kode program" dimana keduanya memiliki arti yang sama.



Gambar 2.14 Tampilan Pertama Arduino Ide

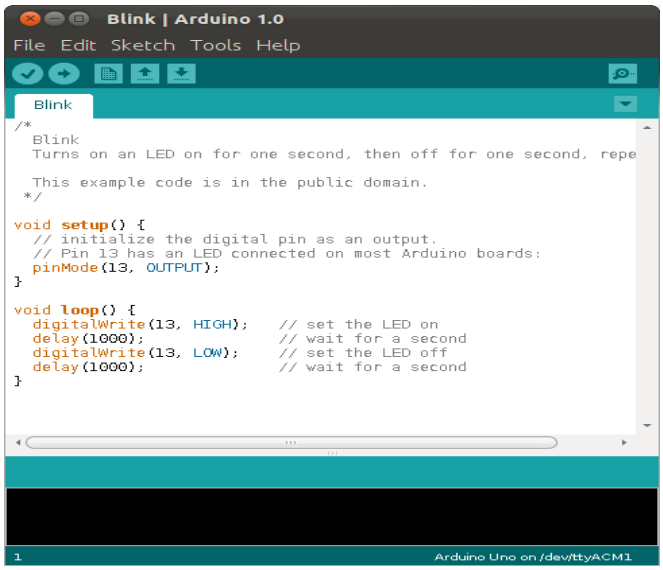
2.2.11.1.1. Bahasa Pemograman Arduino Berbasis Bahasa C

Seperti yang telah dijelaskan diatas program Arduino sendiri menggunakan bahasa C. walaupun banyak sekali terdapat bahasa pemrograman tingkat tinggi (*high level language*) seperti pascal, basic, cobol, dan lainnya. Walaupun demikian, sebagian besar dari paraprogramer profesional masih tetap memilih bahasa C sebagai bahasa yang lebih unggul, berikut alasan-alasannya:

1. Bahasa C merupakan bahasa yang *powerful* dan *fleksibel* yang telah terbukti dapat menyelesaikan program-program besar seperti pembuatan sistem operasi, pengolah gambar (seperti pembuatan game) dan juga pembuatan kompilator bahasa pemrograman baru.
2. Bahasa C merupakan bahasa yang *portabel* sehingga dapat dijalankan di beberapa sistem operasi yang berbeda. Sebagai contoh program yang kita tulis dalam sistem operasi windows dapat kita kompilasi didalam sistem operasi linux dengan sedikit ataupun tanpa perubahan sama sekali.
3. Bahasa C merupakan bahasa yang sangat populer dan banyak digunakan oleh programer berpengalaman sehingga kemungkinan besar *library* pemrograman

telah banyak disediakan oelh pihak luar / lain dan dapat diperoleh dengan mudah.

4. Bahasa C merupakan bahasa yang bersifat modular, yaitu tersusun atas rutin-rutin tertentu yang dinamakan dengan fungsi *(function*) dan fungsi-fungsi tersebut dapat digunakan kembali untuk pembuatan program-program lainnya tanpa harus menulis ulang implementasinya.
5. Bahasa C merupakan bahasa tingkat menengah (*middle level language*) sehingga mudah untuk melakukan interface (pembuatan program antar muka) ke perangkat keras.
6. Struktur penulisan program dalam bahasa C harus memiliki fungsi utama, yang bernama main(). Fungsi inilah yang akan dipanggil pertama kali pada saat proses eksekusi program. Artinya apabila kita mempunyai fungsi lain selain fungsi utama, maka fungsi lain tersebut baru akan dipanggil pada saat digunakan.



Gambar 2.15 Tampilan Arduino Ide

Oleh karena itu bahasa C merupakan bahasa prosedural yang menerapakan konsep runtutan (program dieksekusi per baris dari atas ke bawah secara berurutan),

maka apabila kita menuliskan fungsi-fungsi lain tersebut dibawah fungsi utama, maka kita harus menuliskan bagian prototipe (*prototype*), hal ini dimaksudkan untuk mengenalkan terlebih dahulu kepada kompiler daftar fungsi yang akan digunakan di dalam program. Namun apabila kita menuliskan fungsi-fungsi lain tersebut diatas atau sebelum fungsi utama, maka kita tidak perlu lagi untuk menuliskan bagian prototipe diatas.

Selain itu juga dalam bahasa C kita akan mengenal *file header,* biasa ditulis dengan ekstensi h(*.h), adalah file bantuan yang yang digunakan untuk menyimpan daftar-daftar fungsi yang akan digunakan dalam program. Bagi anda yang sebelumnya pernah mempelajari bahasa pascal, *file header* ini serupa dengan unit. Dalam bahasa C, file header standar yang untuk proses *input/output* adalah <*stdio.h*>.

Perlu sekali untuk diperhatikan bahwa apabila kita menggunakan *file header* yang telah disediakan oleh kompilator, maka kita harus menuliskannya didalam tanda‘<’ dan ‘>’ (misalnya <*stdio.h*>). Namun apabila menggunakan *file header* yang kita buat sendiri, maka file tersebut ditulis diantara tanda “ dan ” (misalnya “coba*header.h*”). perbedaan antara keduanya terletakpada saat pencerian file tersebut. Apabila kita menggunakan tanda <>, maka file tersebut dianggap berada pada direktori deafault yang telah ditentukan oleh kompilator. Sedangkan apabila kita menggunakan tanda “”, maka *file header* dapat kita dapat tentukan sendiri lokasinya.

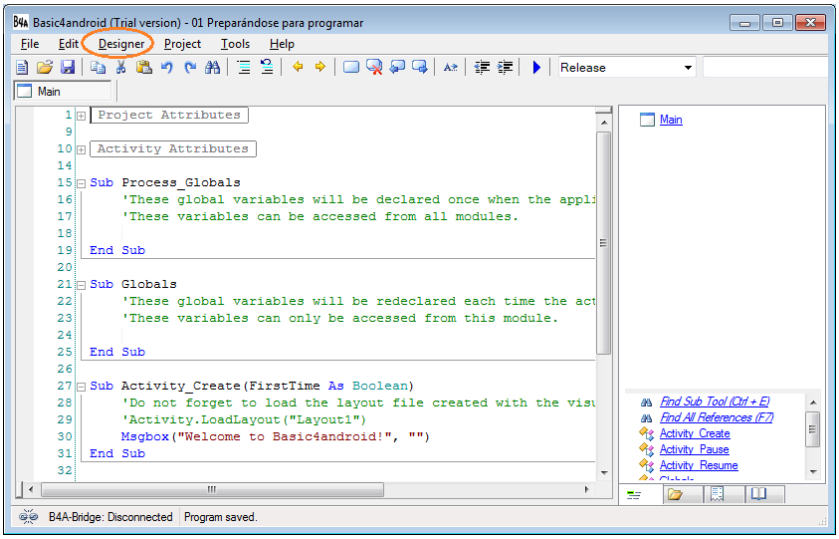
File header yang akan kita gunakan harus kita daftarkan dengan menggunakan directive *#include.* Directive *#include* ini berfungsi untuk memberi tahu kepada kompilator bahwa program yang kita buat akan menggunakan file-file yang didaftarkan. Berikut ini contoh penggunaan directive

1. *#include.*
2. *#include<stdio.h>*
3. *#include<stdlib.h>*
4. *#include”myheader.h”*

Setiap kita akan menggunakan fungsi tertentu yang disimpan dalam sebuah *file header*, maka kita juga harus mendaftarkan *file header*nya dengan menggunakan directive*#include*. Sebagai contoh, kita akan menggunakan fungsi *getch()* dalam program, maka kita harus mendaftarkan *file header*<*conio.h*>.

2.2.11.2. Android4Basic

Basic4Android adalah development tool sederhana yang powerful untuk membangun aplikasi Android. Bahasa Basic4Android mirip dengan bahasa Visual Basic dengan tambahan dukungan untuk objek. Aplikasi Android (APK) yang dicompile oleh Basic4Android adalah aplikasi Android native/asli dan tidak ada extra runtime seperti di Visual Basic yang ketergantungan file msvbvm60.dll, yang pasti aplikasi yang dicompile oleh Basic4Android adalah No Dependencies (tidak ketergantungan file oleh lain). IDE Basic4Android hanya fokus pada development Android.



Gambar 2.16 Tampilan Basic 4 Android

Basic4Android termasuk designer GUI untuk aplikasi Android yang powerful dengan dukungan Built-in untuk multiple screens dan orientations, serta tidak dibutuhkan lagi penulisan XML yang rumit.

Basic4Android memiliki kekayaan dalam satuan libraries (perpustakaan) yang membuatnya menjadi lebih mudah untuk mengembangkan macam-macam aplikasi Android yang advanced. Library-nya adalah:

1 SQL databases
2 GPS
3 Serial ports (Bluetooth)
4 Camera
5 XML parsing
6 Web services (HTTP)
7 Services (background tasks)
8 JSON
9 Animations
10 Network (TCP and UDP)
11 Text To Speech (TTS)
12 Voice Recognition
13 WebView
14 AdMob (ads)
15 Charts
16 OpenGL
17 Graphics

Basic4Android membutuhkan tiga komponen tambahan yaitu NET FrameWork, Java JDK, Android SDK