BAB 2
TINJAUAN PUSTAKA

2.1 Tempat Penelitian

Rumah Sakit Umum Daerah (RSUD) Cicalengka berlokasi di Jl. H. Darham No. 35 Kp. Cikopo Ds. Tenjolaya Kecamatan Cicalengka.

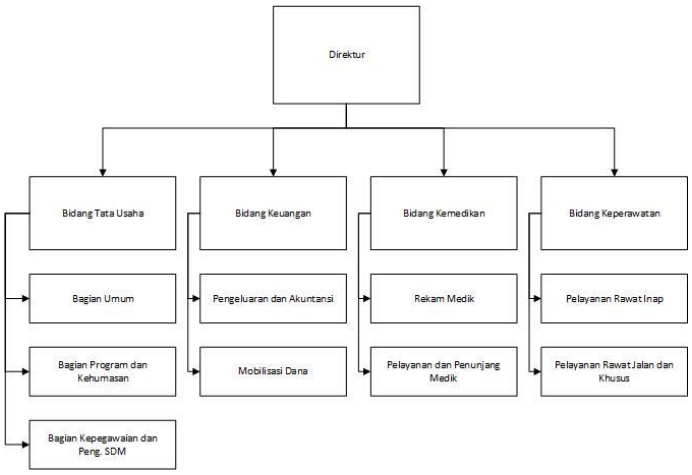
2.1.1 Visi dan Misi

Visi RSUD Cicalengka adalah "Memantapkan pelayanan RSUD Cicalengka yang Maju, Mandiri dan Berdaya saing, melalui tata kelola yang baik berlandaskan Religius, Kultural dan Berwawasan Lingkungan."

Misi RSUD Cicalengka adalah:

1. Meningkatkan mutu pelayanan rumah sakit yang maju dan berdaya saing
2. Meningkatkan kualitas Sumber daya Rumah Sakit
3. Meningkatkan sistem tata kelola Rumah Sakit yang baik dan mandiri.

2.1.2 Struktur Organisasi



Gambar 2.1 Struktur Organisasi RSUD Cicalengka

2.2 Landasan Teori

Landasan teori berfungsi untuk mendukung dalam perancangan sebuah penelitian baik secara teori maupun hal-hal lain yang berkaitan tentang penelitian

ini. Di dalam penelitian ini, akan membahas berbagai teori yang berkaitan dengan skripsi ini sebagai bentuk landasan dalam pembuatan penelitian ini.

2.2.1 Inkubator Bayi

Inkubator bayi menurut (N.S. Salahudin, 2013) adalah suatu peralatan kesehatan yang berupa kotak yang dirancang untuk mempertahankan suhu internal yang konstan, sehingga dapat membantu bayi yang lahir prematur untuk bertahan hidup. Bayi yang baru lahir secara prematur sangatlah rentan terhadap serangan bakteri yang dapat mengganggu kehidupan si bayi, oleh karena itu bayi yang lahir prematur harus ditempatkan pada inkubator atau kotak penghangat untuk bayi. Dalam inkubator terdapat pemanas untuk menghangatkan bayi yang ada di dalamnya. Suhu inkubator perlu dijaga kehangatannya sekitar 35°C – 36°C karena bayi memiliki jaringan lemak yang lebih sedikit sehingga berisiko terkena hipotermia atau suhu tubuh rendah. Kelembaban inkubator juga perlu dijaga, karena pernafasan bayi akan optimal dengan level kelembaban 50 % RH – 60 % RH. [1]

Inkubator bayi dikelompokkan menjadi dua jenis yaitu inkubator sederhana dan inkubator digital.

1. Inkubator Sederhana

Inkubator sederhana adalah inkubator yang banyak digunakan oleh instansi kesehatan kelas menengah ke bawah. Jenis ini biasanya berupa kotak (*box* bayi) yang dilengkapi dengan alat pengukur suhu ruang. Hal ini kurang efektif karena tidak ada pengatur suhu ruang inkubator, sehingga panas ruang inkubator tidak dapat disesuaikan dengan kebutuhan bayi.

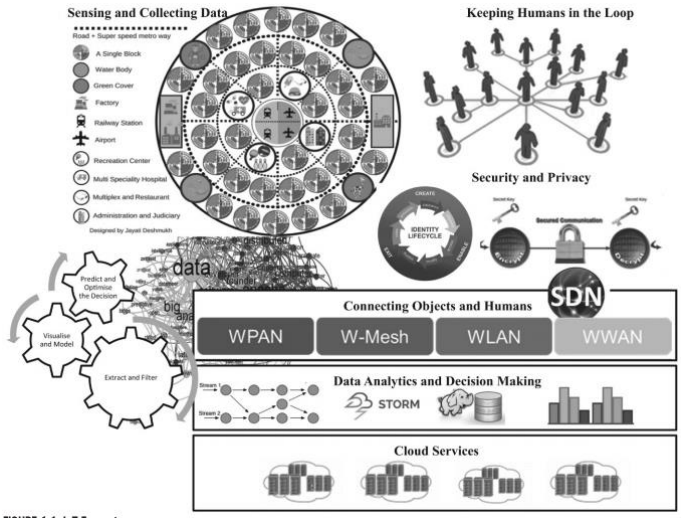
2. Inkubator digital

Inkubator bayi digital merupakan pengembangan dari inkubator bayi sederhana. Jenis ini ditambahkan fungsi yang berkaitan dengan pengaturan suhu ruang inkubator. Secara umum inkubator bayi yang akan dirancang terdiri dari dua bagian yaitu pemanas dan tempat penghangat bayi.

2.2.2 *Internet of Things*

Internet of Things (IOT) terdiri dari 2 pilar utama yaitu "Internet" dan "*Things*", jadi setiap obyek yang mampu terhubung ke internet akan masuk ke dalam kategori "*Things*" seperti mencakup seperangkat entitas yang lebih umum seperti *smartphone*, *sensors*, manusia dan objek lainnya.

Internet of Things merupakan suatu skenario objek yang dapat melakukan pengiriman data atau informasi melalui jaringan tanpa ada campur tangan manusia. Teknologi IoT telah berkembang dari keonvergensi *micro-electromechanical* system (MEMS), dan internet pada jaringan nirkabel. IoT sangat erat hubungannya dengan komunikasi mesin dengan mesin atau (M2M) tanpa ada campur tangan manusia ataupun komputer yang lebih dikenal dengan istilah cerdas (*smart*) [6]. Berikut ini merupakan gambar *ecosystem Internet of Things* (IoT).



Gambar 2.2 *Ecosystem Internet of Things* (IoT) [6]

Internet of Things (IOT) juga diidentifikasi sebagai enabler untuk interaksi mesin-ke-mesin, manusia-ke-mesin, dan manusia dengan lingkungan. Dengan meningkatnya jumlah smart device dan adopsi protocol baru seperti IPv6, *Internet of Things* (IOT) diperkirakan akan bergeser kearah perpaduan *smart networking* dan *autonomous networks* dari objek-objek yang dilengkapi internet. *Internet of Things* (IOT) menawarkan banyak keuntungan untuk berbagai aplikasi, termasuk *emergency management*, *smart farming*, perawatan kesehatan dan lain-lain. Peran

penting lain dari *Internet of Things* (IOT) adalah membangun sistem kolaboratif yang mampu merespons secara efektif terhadap peristiwa yang ditangkap melalui sensor dengan efektif [6].

2.2.2.1 *Internet of Things (IOT) Architectures*

Block arsitektur *Internet of Things* (IOT) adalah perangkat sensorik, *remote service invocation* dan komunikasi jaringan. Arsitektur sistem *holistic* untuk *Internet of Things* (IOT) 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* [6]. Model arsitektur *Internet of Things* (IOT) sebagai berikut:

1. ***SOA-Based Architecture***

   Service-Oriented Architecture (SOA) mungkin penting untuk penyedia layanan dan pengguna, memastikan *interoperability* di antara perangkat heterogen. Terdapat 4 lapisan fungsi yang berbeda sebagai berikut:

   a. *Sensing Layer*, integerasi 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 applikasi.
   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 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 [6].

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 tenik ini, *API* web dan *Representational State Transfer* (REST) sebagai metode yang memiliki solusi alternative yang menjanjikan. Sumber daya yang dibutuhkan berkisar dari bandwidth jarigan 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 applikasi *Internet of Things* (IOT) membantu penyedia layanan menarik lebih banyak pelanggan sambal berfokus pada fungsionaltias produk dari pada presentasi. Selain itu, lebih mudah untuk mengaktifkan *multitenancy* dengan fitur keamanan API sepeti *OAuth*, 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 [6].

2.2.3 Wawancara

Wawancara adalah suatu cara pengumpulan data yang digunakan untuk memperoleh informasi langsung dari sumbernya. Wawancara digunakan bila kita ingin mengetahui hal-hal dari responden yang jumlahnya sedikit, secara mendalam. Pada umumnya wawancara dibagi menjadi dua, yaitu wawancara berstruktur dan wawancara tidak berstruktur. Dalam wawancara berstruktur semua pertanyaan telah dirumuskan sebelumnya dengan cermat, biasanya secara tertulis.

Di sisi lain, wawancara tak berstruktur lebih bersifat informal. Pertanyaan tentang pandangan, sikap, keyakinan subjek atau keterangan lainnya dapat diajukan secara bebas kepada subjek [2]. Pada penelitian ini jenis wawancara yang digunakan adalah wawancara tidak berstruktur karena wawancara yang dilakukan pertanyaan yang diajukan secara bebas.

2.2.4 Observasi

Pengamatan atau observasi adalah suatu teknik atau cara untuk mengumpulkan data dengan jalan mengamati kegiatan yang sedang berlangsung. Pengamatan dapat dilakukan dengan partisiasi ataupun nonpartisipasi. Dalam pengamatan partisipatori (*participatory observation*) pengamat ikut serta dalam kegiatan yang sedang berlangsung. Sementara dalam pengamatan nonpartisipatori (*nonparticipatory observation*) pengamat tidak ikut serta dalam kegiatan. Pengamat hanya berperan mengamati kegiatan [2]. Pada penelitian ini jenis pengamatan yang digunakan adalah pengamatan nonpartisipatori karena penulis tidak ikut serta dalam kegiatan dan hanya berperan mengamati saja.

2.2.5 Unified Modelling Language (UML)

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 permodelan visual yang memungkinkan bagi pengembang sistem untuk membuat cetak biru atau svisi mekanisme yang efektif untuk berbagi dan mengkomunikasikan rancagan mereka dengan yang lain. *Unified Modeling Language* (UML) merupakan kesatuan dari Bahasa permodelan yang dikembangkan booch, *Object Modeling Technique* (OMT) dan *Object Orented Software Engineering* (OOSE) Metode ini menjadikan proses analisis dan *design* kedalam empat tahapan iterative, yaitu identifikasi kelas-kelas, dan objek-objek, identifikasi semantic dari hubungan objek dan kelas tersebut, perincian *interface* dan implementasi. Keunggulan metode Booch adalah pada detail dan kayanya dengan notasi dan elemen, permodelan 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 OO.

Metode OOSE dari Jacobson lebih memberi menekanan pada use case. OOSE memiliki tiga tahapan yaitu membuat model requirement dan analisis, *design* 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. 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 coding 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. [7].

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* menajdi pusat peran dan sering dikatakan yang meng-*drive* 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*. *Design 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*. 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*.

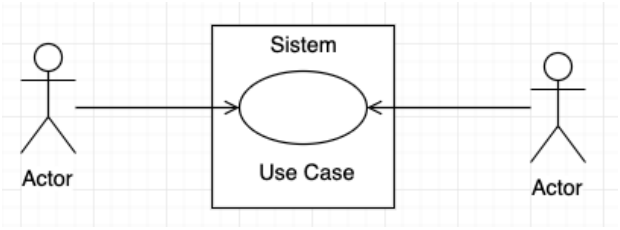
Mesikpun UML sudah cukup banyak menyediakan diagram yang bisa membantu mendefinisikan sebuah aplikasi, tidak berarti bahwa semua diagram tersebut akan bisa menjawab persoalan yang ada. Dalam banyak kasus, diagram lain selain UML sangat banyak membantu. Pada penelitian ini konsep UML digunakan untuk menggambarkan bagaimana sistem bekerja, hubungan antar *class*, memberi gambaran kepada pengguna sistem yang akan di gunakan dan memberikan penjelasan detail pemanggilan fungsi-fungsi atau method dalam *class* [7].

2.2.5.1 Diagram UML

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. Setiap urutan diinisialisasi oleh orang, *system* yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatan *use case* adalah serangkaian *scenario* 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 2.3 menunjukan *Use Case Diagram* dalam UML [7].

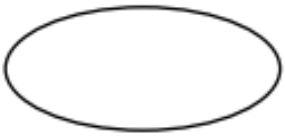


Gambar 2.3 *Use Case Diagram* [7]

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

a. Use Case

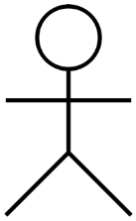
Use case adalah abstraksi dari interaksi antarasistem dengan *actor*. Oleh karena itu sangat penting untuk memilih abstraksi yang cocok. *Use case* dibuat berdasarkan keperluan *actor*. *Use case* harus merupakan ‘apa’ yang dikerjakan *software* aplikasi, bukan ‘bagaimana’ *software* aplikasi mengerjakannya. Setiap *use 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. Gambar 2.4 menunjukan bentuk *Use Case* dalam UML.



Gambar 2.4 *Use Case* [7]

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 *control* atas *use case*. Gambar 2.5 menunjukan bentuk *actor* dalam UML.



Gambar 2.5 *Actor* [7]

c. *Relationship*

Relationship adalah hubungan antara *use cases* dengan *actors* [7]. *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.

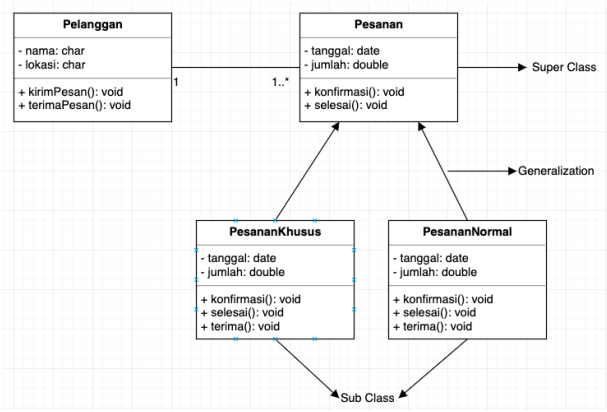
d. Generalisasi antara 2 *use case*.

   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 OO 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 structural* [7]. Gambar 2.6 menunjukan *Class Diagram* dalam UML.



Gambar 2.6 *Class Diagram* [7]

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.

b. *Generalization*

Generalization adalah sebuah hubungan antar *class* yang bersifat dari khusus ke umum

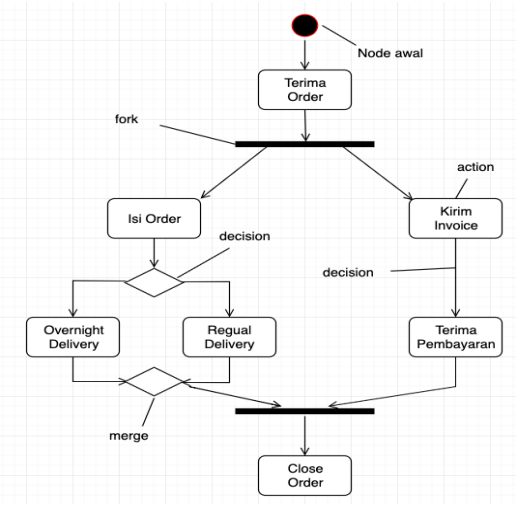
c. *Constraint*

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 mendukug 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. Secara umum *activity diagram* digunakan untuk menggambarkan diagram alir yang terdiri dari banyak aktifitas dalam sistem dengan beberapa fungsi tambahan seperti percabangan, aliran pararel, *swim lane*, dan sebagainya. Sebelum menggambarkan sebuah *activity diagram*, perlu adanya pemahaman yang jelas tentang elemen yang akan digunakan di *activity diagram*. Elemen tuama dalam *activity diagram* adalah aktifitas itu sendiri. Aktifitas adalah fungsi yang dilakukan oleh sistem setelah akitifitas terindentifikasi, selanjutnya yang perlu diketahui adalah bagaimana semua elemen tersebut berasosiasi dengan *constraint* dan kondisi. Langkah selanjutnya perlu penjabaran tata letak dari keseluruhan aliran agara bisa ditransofrmasikan ke *activity diagram* [7]. Gambar 2.7 menunjukan *Activity Diagram* dalam UML.



Gambar 2.7 *Activity Diagram* [7]

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.

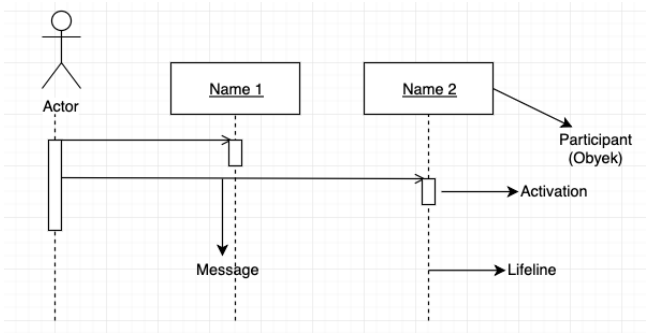
g. *Final state*

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 guna 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 ururssannya. Namun jika message 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 [7]. Gambar 2.8 menunjukan *Sequence Diagram* dalam UML.



Gambar 2.8 *Sequence Diagram* [7]

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.

2.2.6 Object Oriented (OO)

Object oriented adalah sebuah obyek memiliki keadaan sesaat (*state*) dan perilaku (*behaviour*). State sebuah obyek adalah kondisi obyek tersebut yang dinyatakan dalam *attribute*/*properties*. *State* sebuah obyek adalah kondisi obyek tersebut yang dinyatakan dalam *attribute*/*properties*. Sedangkan perilaku suatu obyek mendefinisikan bagaimana sebuah obyek bertindak/beraksi dan memberikan reaksi. Perilaku sebuah objek dinyatakan dalam *operation*. Menurut Schmuller, *attribute* dan *operation* bila disatukan akan memberikan fitur/*features*. Himpunan objek-objek yang sejenis disebut class. Objek adalah contoh *instance* dari sebuah *class*. Ada dua macam aplikasi yang tidak cocok dikembangkan dengan metode OO. Yang pertama adalah aplikasi yang sangat berorientasi ke *database*. Aplikasi yang sangat berorientasi ke penyimpanan dan pemanggilan data sagnat tidak cocok dikembangkan dengan OO karena akan banyak kehilangan manfaat dari penggunaan RDBMS (*Relational Database Management System*) untuk penyimpanan data. Akan tetapi RDBMS juga mempunyai keterbatasan dalam penyimpanan dan pemanggilan struktur data yang kompleks seperti multimedia, data spasial dan lain-lain. Bentuk aplikasi lain yang kurang cocok dengan pendekatan OO adalah aplikasi yang membutuhkan banyak algoritma. Beberapa applikasi yang melibatkan perhitungan yang besar dan

kompleks [7]. Pada penelitian ini konsep OO digunakan untuk konsep pengkodean pada sistem yang akan dibangun. Berikut ini adalah konsep-konsep dalam pemrograman berorientasi objek:

1. ***Class***

   Kelas (*Class*) merupakan penggambaran satu set objek yang memiliki atribut yang sama. Kelas mirip dengan tipe data ada pemrograman non objek, akan tetapi lebih komprehensif karena terdapat struktur sekaligus karakteristiknya. Kelas baru dapat dibentuk lebih spesifik dari kelas pada umumnya. Kelas merupakan jantung dalam pemrograman berorientasi objek.

2. ***Object***

   Objek merupakan teknik dalam menyelesaikan masalah yang kerap muncul dalam pengembangan perangkat lunak. Teknik ini merupakan teknik yang efektif dalam menemukan cara yang tepat dalam membangun sistem dan menjadi metode yang paling banyak dipakai oleh para pengembang perangkat lunak. Orientasi objek merupakan teknik pemodelan sistem riil yang berbasis objek.

3. ***Abstraction***

   Kemampuan sebuah program untuk melewati aspek informasi yang diolah adalah kemampuan untuk fokus pada inti permasalahan. Setiap objek dalam sistem melayani berbagai model dari pelaku abstrak yang dapat melakukan kerja, laporan dan perubahan serta berkomunikasi dengan objek lain dalam sistem, tanpa harus menampakkan kelebihan diterapkan.

4. ***Encapsulation***

   Encapsulation adalah proses memastikan pengguna sebuah objek tidak dapat menggantikan keadaan dari sebuah objek dengan cara yang tidak sesuai prosedur. Artinya, hanya metode yang terdapat dalam objek tersebut yang diberi izin untuk mengakses keadaan yang diinginkan. Setiap objek mengakses *interface* yang menyabutkan bagaimana objek lainnya dapat berintegrasi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

5. ***Polimorfism***

   Polimorfism merupakan suatu fungsionalitas yang diimplikasikan dengan berbagai cara yang berbeda. Pada program berorientasi objek, pembuat program dapat memiliki berbagai implementasi untuk sebagian fungsi tertentu.

6. ***Inheritance***

   Seperti yang sudah diuraikan di atas, objek adalah contoh / *instance* dari sebuah *class*. Hal ini mempunyai konsekuensi yang penting yaitu sebagai *instance* sebuah *class*, sebuah objek mempuyai semua karakteristik dari classnya. Inilah yang disebut dengan *Inheritance* (pewarisan sifat). Dengan demikian apapun *attribute* dan *operation* dari *class* akan dimiliki pula oleh semua obyek yang *disinherit*/diturunkan dari class tersebut. Sifat ini tidak hanya berlaku untuk objek terhadap *class*, akan tetapi juga berlaku untuk *class* terhadap *class* lainnya.

2.2.7 Python

Python diciptakan oleh Guido van Rossum di Belanda pada tahun 1990 dan namanya diambil dari acara televisi kesukaan Guido *Monty Python's Flying Circus*. Van Rossum mengembangkan *Python* sebagai hobi, kemudian *Python* menjadi bahasa pemrograman yang dipakai secara luas dalam industri dan pendidikan karena sederhana, ringkas, sintaks intuitif dan memiliki pustaka yang luas. Python saat ini dikembangkan dan dikelola oleh tim relawan yang besar dan tersedia secara gratis dari Python *Software Foundation*. Python termasuk dari jajaran bahasa pemograman tingkat tinggi seperti bahasa pemograman C, C++, Java, Perl dan Pascal, dikenal juga bahasa pemograman tingkat rendah, yang dikenal sebagai bahasa mesin yaitu bahasa pemograman assembly, kenyataanya komputer hanya dapat mengeksekusi bhasa tingkat rendah, jadi bahasa tingkat tingkat tinggi harus melewati beberapa proses untuk diubah ke bahasa pemograman tingkat rendah, hal tersebut merupakan kelemahan yang tidak berarti bahasa pemograman tingkat tinggi [8]. Tetapi kekurangna tersebut tidak sebanding dengan kelebihanya. Pertama, lebih mudah memprogram sebuah

aplikasi dengan bahasa tingkat tinggi. Lebih cepat, lebih mudah dimengerti menulis program komputer dengan bahasa tingkat tinggi, dan juga kesalahan dalam penulisan program cenderung tidak mengalami kesalahan yang berarti. Kedua bahasa pemograman tingkat tinggi lebih portabel dalam arti bisa digunakan untuk menulis di berbagai jenis arsitektur komputer yang berlainan dengan sedikit modifikasi ataupun tidak memerlukan modifikasi sama sekali. Bahasa pemograman tingkat rendah hanya dapat berjalan di satu jenis arsitektur komputer dan harus ditulis ulang untuk menjalankannya di lain mesin, hal ini diakrenakan karena perbedan urutan register dan *services* – *services*nya. Pada penelitian ini bahasa pemograman python digunakan sebagai komponen utama untuk pembanguan website dan konfigurasi *raspberry pi*. Gambar 2.9 menunjukan logo python.

Gambar 2.9 Logo Python [8]

Python digunakan di berbagai bidang pengembangan. Berikut beberapa implementasi bahasa python yang paling popular:

1. Website

   Bahasa pemograman python dapat digunakan sebagai *server-side* yang diintegrasikan dengan berbagai internet *protocol* misalnya JSON, *Email Processing*, FTP, dan IMAP. Selain itu python juga mempunyai library pendukung untuk pengembangan website seperti Django, Flask, Pyramid dan Bottle.

2. Penelitian

   Python dapat digunakan juga untuk melakukan riset ilmiah untuk mempermudah perhitungan numerik. Misalnya penerapan algoritma KNN, Naïve Bayes dan lain-lain. Beberapa *library* yang sering digunakan untuk riset seperti Pandas, Numpy, Mathplotlib dan lain-lain.

3. Media Pembelajaran

Python dapat digunakan sebagai media pembelajaran di universitas. Python sangat mudah dan hemat untuk dipelajari sebagai *Object Oriented Programming* dibandingkan bahasa lainnya seperti MATLAB, C++, dan C#.

4. *Graphical User Interface* (GUI)

   Python dapat digunakan untuk membangun interface sebuah aplikasi. Tersedia *library* untuk membuat GUI menggunakan python, misalnya Qt, win32extension, dan GTK+.

5. *Internet of Things* (IOT)

   Bahasa pemrograman Python mendukung ekosistem *Internet of Things* (IOT) dengan sangat baik. *Internet of Things* (IOT) merupakan sebuah teknologi yang menghubungkan benda-benda di sekitar kita ke dalam sebuah jaring-jaring yang saling terhubung.

2.2.8 Java Script Object Notation (JSON)

Format data *JavaScript Object Notation* (JSON) memungkinkan aplikasi untuk berkomunikasi melalui jaringan, biasanya melalui RESTful APIs. JSON adalah teknologi-agnostik, *nonproprietary*, dan *portabel*. Semua bahasa modern (mis., Jawa, JavaScript, Ruby, C #, PHP, Python, dan Groovy) dan *platform* memberikan dukungan yang sangat baik untuk memproduksi (membuat serial) dan mengonsumsi data JSON (*deserializing*). JSON sederhana: terdiri dari konstruksi ramah-pengembang seperti *Objects*, *Array*, dan pasangan nama / nilai. JSON tidak terbatas pada *Representational State Transfer* (REST).

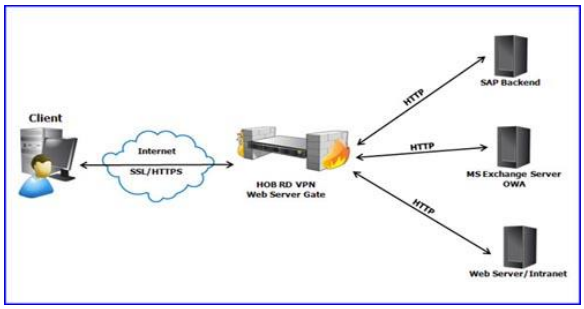
RESTful Web *Services* sebagai tidak standar, tetapi (seperti HTTP) JSON sebenarnya merupakan standar. Baik *Internet Engineering Task Force* (IETF) dan *Ecma International* (sebelumnya *European Computer Manufacturers Association*, atau ECMA) telah mengakui JSON sebagai standar. Douglas Crockford awalnya menciptakan JSON pada tahun 2001, dan awalnya membakukannya pada tahun 2006 dibawah RFC 4627 melalui IETF; lihat spesifikasi JSON. Pada musim gugur 2013, Ecma International juga menstandarisasi JSON di bawah ECMA 404; lihat spesifikasi JSON mereka. Dengan pengakuan Ecma, JSON sekarang dianggap

sebagai standar pemrosesan data internasional formal [9]. Pada penelitian ini JSON digunakan sebagai API untuk mengirim data sensor dari raspberry pi ke website.

2.2.9 Web *Server*

Web *server* adalah sebuah *software* yang memberikan layanan berbasis data dan berfungsi menerima permintaan dari HTTP atau HTTPS pada klien Web *server* menunggu permintaan dari *client* yang menggunakan browser seperti Netscape Navigator, Internet Explorer, Mozilla, dan program browser lainnya. Jika ada permintaan dari browser, maka web server akan memproses permintaan itu kemudian memberikan hasil prosesnya berupa data yang diinginkan kembali ke *browser*. Data ini mempunyai format yang standar, disebut dengan format SGML (*standard general markup language*). Data yang berupa format ini kemudian akan ditampilkan oleh *browser* sesuai dengan kemampuan *browser* tersebut.

Web *server* berfungsi untuk mentransfer berkas melalui protokol komunikasi yang telah ditentukan atas permintaan pengguna. Berkas yang ditransfer dapat berupa teks, gambar, video, dan lainnya yang merupakan elemen sebuah halaman web. Web *server* ini saat ini berfungsi pula untuk menjalankan program-program yang memang dirancang untuk berjalan di web *server*. Bahasa-bahasa tersebut ialah seperti PHP atau ASP. Sehingga web server dapat juga dapat melakukan pengolahan data yang diberikan oleh pengguna. Fitur ini biasa disebut *server site scripting*. Gambar 2.10 menunjukan alur web server.



Gambar 2.10 Web *Server*

Beberapa Jenis Web *Server* di antanya adalah:

1. Apache Web *Server* / The HTTP Web *Server*
2. Apache Tomcat
3. Microsoft windows Server 2008 IIS (*Internet Information Services*)
4. Lighttpd
5. Zeus Web *Server*
6. Sun Java System Web *Server*

2.2.10 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 [10]. Pada penelitian ini MySQL digunakan sebagai platform penyimpanan data dan implementasi model rancangan database yang sudah di buat. Gambar 2.11 menunjukan logo *MySQL*.

Gambar 2.11 Logo MySQL [10]

MySQL memiliki banyak fitur. fitur dimiliki MySQL sebagai berikut :

1. *Relational Database System,* Seperti halnya *software database* lain yang ada di pasaran, MySQL termasuk RDBMS.
2. *Arsitektur Client-Server, MySQL* memiliki arsitektur client-server dimana *server database MySQL* terinstal di *server*. Client *MySQL* dapat berada di komputer yang sama dengan *server*, dan dapat juga di komputer lain yang berkomunikasi dengan server melalui jaringan bahkan *internet.*
3. *Mengenal perintah SQL standar, SQL (Structured Query Language)* merupakan suatu bahasa standar yang berlaku di hampir semua *software database. MySQL* mendukung *SQL* versi *SQL:2003*.

4. *Performace Tuning*, *MySQL* mempunyai kecepatan yang cukup baik dalam menangani *query-query* sederhana, serta mampu memproses lebih banyak *SQL* per satuan waktu.
5. *Sub Select*.
6. *Views*.
7. *Stored Prosedured* (SP).
8. *Triggers.*
9. *Replication*.
10. *Foreign Key*.
11. Security yang baik.

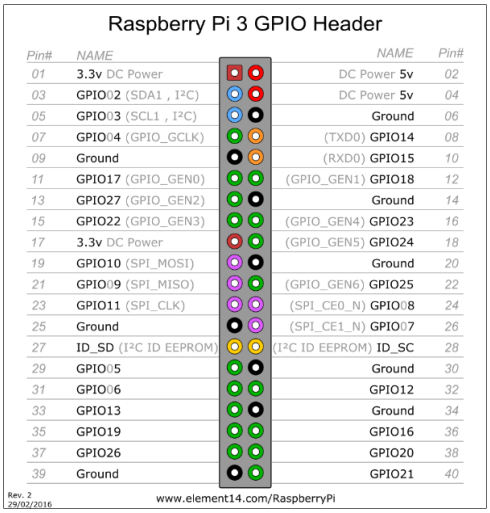
2.2.11 Raspberry Pi 3 Model B

Raspberry Pi 3 Model B merupakan generasi ketiga dari keluarga *raspberry pi*. *raspberry pi* 3 memiliki *RAM* 1GB dan grafis *Broadcom VideoCore* IV pada frekuensi *clock* yang lebih tinggi dari sebelumnya yang berjalan pada 250MHz. *Raspberry Pi* 3 juga memiliki 4 *USB port*, 40 *pin GPIO, Full HDMI port*, *Port Ethernet*, *Combined 3.5mm audio jack and composite video*, *Camera interface* (CSI), *Display interface* (DSI), slot kartu *Micro SD* (Sistem tekan-tarik, berbeda dari yang sebelum nya ditekan-tekan), dan V*ideoCore* IV *3D graphics core* [11]. Pada penelitian ini raspberry pi digunakan sebagai mini pc untuk *server* website dan juga pengontrolan sensor dan modul relay di inkubator bayi. Gambar 2.12 adalah bentuk *raspberry pi* 3 model B.

Gambar 2.12 Raspberry Pi 3 Model B

2.2.11.1 GPIO Raspberry Pi 3 Model B

Raspberry pi 3 model B terdapat 40 buah pin. Pin pada raspberry pi 3 model B terdiri dari beberapa bagian yaitu bagian VCC, GND, dan GPIO (*General Purpose Input/Output*) terdapat 3 pin VCC dan 8 pin GND [10]. Pin GPIO mulai dari GPIO 2 hingga GPIO 27, pada pin GPIO terdapat fungsi lain yang dapat dilihat pada Gambar 2.13.



Gambar 2.13 Pin GPIO Raspberry Pi 3 Model B [11]

2.2.12 Sensor DHT 22

DHT 22 adalah sensor suhu dan kelembaban, memiliki *output* sinyal digital yang dikalibrasi dengan sensor suhu dan kelembaban yang kompleks. Teknologi ini memastikan keandalan tinggi dan sangat baik stabilnya dalam jangka panjang. Mikrokontroler terhubung pada kinerja tinggi sebesar 8 bit. Sensor ini termasuk elemen resistif dan perangkat pengukur suhu *NTC (Negative Temperature Coefficient)*. Memiliki kualitas yang sangat baik, respot cepat, kemampuan anti-gangguan dan keuntungan biaya tinggi kinerja [12].

Setiap sensor DHT 22 memiliki fitur kalibrasi sangat akurat dari kelembaban ruang kalibrasi. Koefisien kalibrasi yang disimpan dalam memori program OTP, sensor internal mendeteksi sinyal dalam proses, sistem antarmuka tunggal-kabel serial *integrase* untuk menjadi cepat dan mudah. Ukuran yang

kecil, daya rendah, sinyal transmisi jarak hingga 20 meter. Gambar 2.14 menunjukan bentuk sensor DHT 22.

Gambar 2.14 Sensor DHT 22 [12]

2.2.13 Web *Camera*

Web *camera* (webcam) adalah sebutan bagi kamera *real time* yang gambarnya bisa dilihat melalui website, program pengolah pesan cepat, atau aplikasi pemanggilan video. Webcam merupakan sebuah kamera video digital kecil yang dihubungkan ke komputer melalui *port* USB, *port* COM atau dengan jaringan Ethernet atau Wi-Fi. Contoh webcam dapat dilihat pada gambar 2.15 berikut:

Gambar 2.15 Web *Camera*

Fungsi dari webcam telah kita ketahui yaitu untuk memudahkan kita dalam mengolah pesan cepat seperti *chat* melaui video atau bertatap muka melalui video secara langsung. Webcam juga berfungsi sebagai alat untuk men-transfer sebuah media secara langsung.

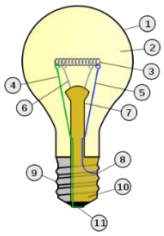
2.2.14 *Heater*

Heater (elemen pemanas) adalah alat listrik yang berfungsi untuk memanaskan suatu objek. Lampu listrik adalah suatu perangkat yang dapat menghasilkan cahaya saat dialiri arus listrik. Arus listrik dapat berasal dari tenaga listrik yang dihasilkan oleh pembangkit listrik terpusat seperti Perusahaan Listrik Negara (PLN) ataupun tenaga listrik yang dihasilkan oleh baterai dan aki.

Selain mengubah energi listrik menjadi cahaya, lampu listrik juga mengeluarkan panas. Dari energi panas tersebut dapat dimanfaatkan sebagai sumber pengering. Panas yang dikeluarkan dari lampu sama dengan panas matahari yang dimanfaatkan manusia saat mengeringkan pakaian yaitu sekitar 33℃ - 40℃. Jenis-jenis lampu listrik adalah:

a. Lampu pijar (*Incandescent Lamp*)
b. Lampu lucutan gas (*Gas-discharge Lamp*)
c. Lampu LED (*Light Emmiting Diode*)

Lampu pijar (*Incandescent Lamp*) adalah suatu jenis lampu yang menghasilkan cahaya dengan cara memanaskna kawat filamen di dalam bola kaca. Kaca yang menyelubungi filamen panas tersebut menghalangi udara masuk untuk berhubungan dengannya sehingga filamen tidak akan langsung rusak akibat teroksidasi. Gambar 2.16 menunjukan bagian-bagian lampu pijar.



Gambar 2.16 Lampu Pijar

Bagian-bagian lampu pijar:

1. Bola lampu
2. Gas bertekanan rendah (argin, neon, nitrogen)
3. Filamen wolfram
4. Kawat penghubung ke kaki tengah
5. Kawat penghubung ke ulir
6. Kawat penyangga
7. Kaca penyangga
8. Kontak listrik di ulir
9. Sekrup ulir
10. Isolator
11. Kontak listrik di kaki tengah

2.2.15 *Fan* DC/Motor DC

Fan DC atau motor DC berfungsi untuk mengubah tenaga listrik menjadi tenaga mekanis dimana gerak tersebut berupa putaran dari motor. Motor DC mempunyai dua terminal elektrik. Dengan memberikan beda tegangan pada kedua terminal tersebut maka motor akan dapat berputar pada satu arah dan apabila polaritas dari tegangan tersebut dibalik, maka arah putaran motor akan terbalik pula. Hal ini berlaku pada motor DC dan tidak berlaku pada motor AC, polaritas dari tegangan yang diberikan pada dua terminal menentukan arah putaran motor sedangkan beda tegangan yang diberikan menentukan kecepatan motor tersebut. Gambar 2.17 menunjukan bentuk fan DC.

Gambar 2.17 Fan DC/Motor DC

2.2.16 Relay

Relay adalah saklar yang dioperasikan secara elektrik. Relay dipakai ketika sinyal berdaya rendah digunakan untuk mengontrol sebuah rangkaian (isolasi elektrik penuh terjadi antara rangkaian pengontrol dan rangkaian yang dikontrol) atau ketika beberapa sirkuit harus dikontrol oleh satu sinyal. Relay pada mulanya digunakan pada sirkuit telegram jarak jauh, mengulangi sinyal yang datang dari suatu sirkuit dan mentransmisikan kembali sinyal tersebut ke sirkuit yang lain. Relay digunakan secara luas dalam *switching* telepon dan juga pada komputer mula-mula untuk melakukan operasi logis [13].

Pada penelitian ini modul relay digunakan untuk memasukkan udara segar dan mengeluarkan udara panas di dalam inkubator dan menyalakan lampu. Gambar 2.18 merupakan bentuk relay.

Gambar 2.18 Relay [13]

2.2.17 Metode Pengujian

Pengujian perangkat lunak merupakan proses eksekusi program atau perangkat lunak dengan tujuan mencari kesalahan atau kelemahan dari program tersebut. Proses tersebut dilakukan dengan mengevaluasi atribut dan kemampuan program. Suatu program yang diuji akan dievaluasi apakah keluaran atau output yang dihasilkan telah sesuai dengan yang diinginkan atau tidak. Ada berbagai macam metode pengujian, teknik *black box* dan teknik *white box* merupakan metode pengujian yang telah dikenal dan banyak digunakan oleh pengembang perangkat lunak.

2.2.17.1 Black Box Testing

Black box testing merupakan metode pengujian dengan pendekatan yang mengasumsikan sebuah sistem perangkat lunak atau program sebagai sebuah kotak hitam (*black box*). Pendekatan ini hanya mengevaluasi program dari *output* atau hasil akhir yang dikeluarkan oleh program tersebut. Struktur program dan kode-kode yang ada di dalamnya tidak termasuk dalam pengujian ini. Keuntungan dari metode pengujian ini adalah mudah dan sederhana. Namun, pengujian dengan metode ini tidak dapat mendeteksi kekurangefektifan pengkodean dalam suatu program. Metode pengujian *black box* merupakan metode pengujian dengan pendekatan yang mengasumsikan sebuah sistem perangkat lunak atau program sebagai sebuah kotak hitam (*black box*). Pendekatan ini hanya mengevaluasi program dari output atau hasil akhir yang dikeluarkan oleh program tersebut. Struktur program dan kode-kode yang ada di dalamnya tidak termasuk dalam pengujian ini. Keuntungan dari metode pengujian ini adalah mudah dan

sederhana. Namun, pengujian dengan metode ini tidak dapat mendeteksi kekurangefektifan pengkodean dalam suatu program [14].

Ciri-ciri *black box testing* adalah sebagai berikut:

1. *Black box testing* berfokus pada kebutuhan fungsional pada *software*, berdasarkan pada spesifikasi kebutuhan dari *software*.
2. Merupakan pendekatan pelengkap dalam mencangkup error dengan kelas yang berbeda dari metode *white box testing*.
3. Melakukan pengujian tanpa pengetahuan detil struktur internal dari sistem atau komonen yang dites. Juga disebut sebagai *behavioural testing*, *specification-based testing*, *input*/*ouput testing* atau *functional testing*.
4. Terdapat jenis test yang dapat dipilih berdasarkan pada tipe *testing* yang digunakan.
5. Kategori *error* yang akan diketahuai melalui *black box testing* seperti fungsi yang hilang atau tidak benar, *error* dari antar-muka, *error* dari struktur data atau akses *eksternal database*, *error* dari kinerja dan *error* dari inisialisasi.

Equivalence class partitioning adalah sebuah metode *black box* terarah yang meningkatkan efisiensi dari pengujian dan meningkatkan *coverage* dari *error* yang potensial. Sebuah *equivalence class* adalah sebuah kumpulan dari nilai *variable input* yang memprduksi *output* yang sama. Selanjutnya *output correctness test* merupakan pengujian yang memakan sumber daya paling besar dari pengujian. Pada kasus yang sering terjadi dimana hanya *output correctness test* yang dilakukan, maka sumber daya pengujian akan digunakan semua. Implementasi dari kelas-kelas pengujian lain tergantung dari sifat produk *software* dan pengguna selanjutnya dan juga prosedur dan keputusan pengembang. *Output correctness test* mengaplikasikan konsep dari *test case* [14].

Pemilihan *test case* yang baik dapat dicapai dengan efesiensi dari penggunaan *equivalence class partitioning*. Jenis-jenis *test case* sebagai berikut:

1. *Availability test*

   Availability didefinisikan sebagai waktu reaksi yaitu waktu yang dibutuhkan untuk mendapatkan informasi yang diminta atau waktu yang

dibutuhkan oleh firmware yang diinstal pada perlengkapan komputer untuk bereaksi. *Availability* adalah yang paling penting dalam aplikasi online sistem informasi yang sering digunakan. Kegagalan *firmware software* untuk memenuhi persyaratan ketersediaan dapat membuat perlengkapan tersebut tidak berguna.

2. *Realiability test*

   Realiability berkaitan dengan fitur yang dapat diterjemahkan sebagai kegiatan yang terjadi sepanjang waktu seperti waktu rata-rata antara kegagalan (misalnya 500 jam), waktu rata-rata untuk *recovery* setelah kegagalan sistem (misalnya 15 menit) atau *average downtime* per bulan (misalnya 30 menit per bulan). Persyaratan realiabilitas memiliki efek selama *regular full-capacity* operasi sistem. Harus diperhatikan bahwa penambahan faktor *software realiability* juga berkaitan dengan perangkat, sistem operasi, dan efek dari sistem komunikasi data.

3. *Stress test*

   Stress test terdiri dari 2 tipe pengujian yaitu *load test* dan *durability test.* Suatu hal yang mungkin untuk melakuakn pengujian-pengujian tersebut setelah penyelesaian sistem software. Durability test dapat dilakukan hanya setelah firmware atau sistem informasi *software* diinstall dan siap untuk diuji. Pada *load test* berkaitan dengan *functional performance system* dibawah beban maksimal operasional, yaitu maksimal transaksi per menit, hits per menit ke tempat internet dan sebagainya. *Load test*, yang biasanya dilakukan untuk beban yang lebih tinggi dari yang diindikasikan spesifikasi persyaratan merupakan hal yang penting untuk sistem *software* yang rencananya akan dilayani secara simultan oleh sejumlah pengguna. Pada sebagian besar kerja sistem software, beban maksimal menggambarkan gabungan beberapa tipe transaksi. Selanjutnya *durability test* dilakukan pada kondisi operasi fisik yang ekstrem seperti temperatur yang tinggi, kelembaban, mengendara dengan kecepatan tinggi, sebagai detail persyaratan spesifikasi durabilitas. Jadi, dibutuhkan untuk *real-time firmware* yang diintegrasikan ke dalam sistem seperti sistem senjata,

kendaraan transport jarak jauh, *Internet of Things* (IOT) dan keperluan meterologi. Isu ketahan pada *firmware* terdiri dari respon *firmware* terhadap efek cuara seperti temperatur panas atau dingin yang ekstrem, debu, kegagalan operasi ekstrem karena kegagalan listrik secara tiba-tiba, loncatan arus listrik dan putusnya komunikasi secara tiba-tiba.

4. *Training usability test*

   Ketika sejumlah besar pengguna terlibat dalam sistem operasi, *training usability requirement* ditambahkan dalam agenda pengujian. Lingkup dari *training usability test* ditentukan oleh sumber yang dibutuhkan untuk melatih pekerja baru untuk memperoleh level perkenalan dengan sistem yang ditentukan atau untuk mencapai tingkat produksi tertentu. Detail dari pengujian ini, sama halnya dengan yang lain, didasarkan pada karakteristik pekerja. Hasil dari pengujian ini harus menginspirasi rencana dari kursus pelatihan dan *follow-up* serta memperbaiki sistem operasi *software*.

5. *Operational usability test*

   Fokus dari pengujian ini adalah produktifitas operator, yang aspeknya terhadap sistem yang mempengaruhi performance dicapai oleh operator sistem. *Operational usability test* dapat dijalankan secara manual.

Revision class partitioning adalah sebuah metode *black box* lainnya yang merupakan faktor dasar yang menentukan keberhasilan paket suatu *software*, pelayanan jangka panjang, dan keberhasilan penjualan ke sejumlah besar populasi pengguna [24]. Berkaitan dengan hal tersebut terdapat tiga kelas pengujian revisi sebagai berikut:

1. *Reusability test*

   Reusability menentukan bagian mana dari suatu program (modul, integrasi, dbs) yang akan dikembangkan untuk digunakan kembali pada projet pengembangan *software* lainnya, baik yang telah direncanakan maupun yang belum. Bagian ini harus dikembangkan, disusun, dan didokumentasikan menurut prosedur perpustakaan *software* yang digunakan ulang.

2. *software interoperability test*

software interoperability berkaitan dengan kemamputan *software* dalam memenuhi perlengkapan dan paket *software* lainnya agar memungkinkan untuk mengoperasikannya bersama dalam satu sistem komputer kompleks.

3. *Equipment interoperability test*

   Equipment interoperability berkaitan dengan perlengkapan *firmware* dalam menghadapi untuk perlengkapan lain dan atau paket *software*, dimana persyaratan mencantumkan *specified interfaces*, termasuk dengan *interfacing standard*. Pengujian yang relevan harus menguji implementasi dari *interoperability reqirements* dalam sistem.

Keuntungan dan kekurangan dari *black box testing*, beberapa keuntungan dari *black box testing*, diantaranya sebagai berikut:

1. *Black box* testing memungkinkan kita untuk memiliki sebagian besar tingkat pengujian, yang sebagian besarnya dapat diimplementasikan.
2. Untuk tingkat pengujian yang dapat dilakukan baik dengan *white box testing* maupun *black box testing*, *black box testing* memerlukan lebih sedikit sumber dibandingkan dengan yang dibutuhkan oleh *white box testing* pada pake *software* yang sama.

Sedangkan kekurangan dari *black box testing* adalah sebagai berikut:

1. Adanya kemungkinan untuk terjadinya beberapa kesalahan yang tidak disengaja secara bersama-sama akan menimbulkan respon pada pengujian ini dan mencegah deteksi kesalahan (*error*). Dengan kata lain, *black box test* tidak siap untuk mengidentifikasi kesalahan-kesalahan yang berlawanan satu sama lain sehingga menghasilkan output yang benar.
2. Tidak adanya kontrol terhadap *line coverage*. Pada kasus dimana *black box test* diharapkan dapat meningkatkan *line converage*, tidak ada cara yang mudah untuk menspesifikasikan parameter-parameter pengujian yang dibutuhkan untuk meningkatkan *coverage*. Akibatnya, *black box test* dapat melakukan bagian penting dari baris kode, yang tidak ditangani oleh set pengujian.
3. Ketidakmungkinan untuk menguji kualitas pembuatan kode dan pendekatannya dengan standar pembuatan kode.