BAB 2
TINJAUAN PUSTAKA

2.1 Tempat Penelitian

Balai Perbenihan Tanaman Hutan (BPTH) Jawa Barat dibentuk sejalan dengan manat undang-undang 23 tahun 2014 tenang pemerintahan daerah, melalui Preaturan Gubernur Jawa Barat Nomor 91 tahun 2016 merupakan salah satu Unit Pelaksana Teknis Daerah yang mempunyai tugas pokok menyelenggarakan sebagian tugas teknis operasional di bidang perbenihan tanaman hutan, meliputi pengelolaan dan pengembangan sumber benih, sertifikasi dan peredaran benih, serta mengendalikan pelaksanaan tugas pokok dan fungsinya yaitu :

1. Penyelenggaraan penyusunan badan kebijakan teknis di bidang perbenihan tanaman hutan.
2. Penyelenggaraan Perbenihan Tanaman Hutan.
3. Penyelengaraan evaluasi dan pelaporan balai dan penyelenggaraan fungsi lain sesuai dengan tugas pokok dan fungsinya.

Beban yang cukup besar bagi Balai Perbenihan Tnaman Hutan (BPTH) Provinsi Jawa Barat adalah persyaratan keberhasilan pembangunan hutan tanaman di masa mendatang adalah penyediaan benih yang bermutu tinggi, yaitu unggul mutu genetiknya dan mampu beradaptasi dengan kondisi lingkungan tempat tumbuhnya. Benih demikian akan dapat meningkatkan kualitas tegakan, produksi kayu, daya tahan terhadap hama dan penyakit serta memperpendek daur. Dengan pengertian lain benih bermutu merupakan input yang efektif untuk memperoleh keuntungan dari usaha di bidang pembangunan hutan tanaman.

Dengan terbentuknya Balai Perbenihan Tanaman Hutan Provinsi Jawa Barat kedepan mudah mudahan dapat menjawab tantangan dalam upaya rehabilitasi hutan dan lahan melalui produk benih dan bibit yang berkualitas. Tantangan yang cukup besar bahwa Propinsi Jawa Barat terdiri dari daerah pegunungan yang terbentang dari wilayah barat sampai dengan wilayah timur berupa Daerah Aliran Sungai (DAS) yang tingkat erosi/bahaya erosi, dan

sedimentasinya tinggi. Kondisi ini merupakan salah satu luasnya lahan kritis di daerah Hulu DAS dan tingginya fluktuasi debit air diantara musim kemarau dan musim penghujan [20].

2.1.1 Maksud dan Tujuan

Program dan Gambaran tentang Balai Perbenihan Tanaman Hutan di bidang pembenihan dan pembibitan disusun dengan maksud untuk memberikan arah sekaligus menjadi acuan bagi aparatur, masyarakat dan dunia usaha dalam mewujudkan cita-cita dan tujuan pembangunan daerah khususnya di bidang Pembenihan dan Pembibitan komoditi kehutanan [20]. Adapun tujuannya adalah :

1. Menetapkan cara mencapai tujuan melalui program dan kegian berdasarkan alokasi dana pugu indikatif.
2. Mencari beberapa peluang untuk mencapai program dan kegiatan dimaksud melalui sumber-sumber pendanaan baik melalui APBN, APBD maupun sumber lain yang sifatnya tidak mengikat.
3. Menjalin kerjasama dan kemitraan yang sifatnya tidak mengikat melalui pengembangan sarana dan prasarana yang menudung proram pembibitan dan pembenihan pada Kawasan Hutan Daerah Kiarapayung (HDK) Kabupaten Sumedang Provinsi Jawa Barat.

2.1.2 Struktur Organisasi

Dalam menjalankan tugas yang telah dibebankan, Balai Perbenihan Tanaman Hutan Provinsi Jawa Barat memiliki struktur organisasi sebagai berikut. Pimpinan adalah Kepala Balai 1 (satu) dibantu oleh Kepala Sub Bagian Tata Usaha 1 (satu), Kepala Seksi Sertifikasi dan Peredaran Benih dan Bibit dan Seksi Sumber Benih ditambah beberapa unsur pengelola dan kelompok jabatan fungsional [20]. Secara structural kondisi pegawai yang ada di Balai Perbenihan dan Pembibitan Jawa Barat saat ini pada Gambar 2.1.

Gambar 2.1 Stuktur Organisasi **[20]**

2.1.3 Visi dan Misi

Dalam mewujudkan visi yang telah ditetapkan dan mengacu kepada strategi RPJMD Provinsi Jawa Barat Jawa Barat Maju dan Sejahtera Untuk Semua, maka Visi Dinas Kehutanan Tahun 2013-2018 adalah : "*HUTAN LESTARI BAGI KESEJAHTERAAN MASYARAKAT*".

Misi secara eksplisit menyatakan apa yang harus dicapai oleh suatu organisasi dan kegiatan spesifik apa yang harus dilaksanakan dalam pencapaian tersebut [20]. Adapun secara gamblang misi dinas yaitu :

1. Mengingkatkan Kemantapan Kawasan Hutan dan Keberlangsungan Fungsi Kawasan Lindung

   Tujuan :

   a. Meningkatkan Kualitas Kawasan Lindung.
   b. Menurunkan Luas Lahan Kritis.
   c. Meningkatkan Kualitas Konservasi Keaneka ragaman Hayati.

   Sasaran :

   a. Terwujudnya Fungsi Kawasan Lindung 45%.
   b. Terlaksananya Rehabilitasi Lahan Kritis pada Daerah Aliran Sungai (DAS) Prioritas.
   c. Meningkatnya upaya perlindungan keaneka ragaman hayati.
   d. Terlaksananya Rehabilitasi Hutan Mangrove dan Hutan Pantai.
2. Optimalisasi Pemanfaatan Hasil Hutan Berbasis Pemberdayaan Masyarakat

   Tujuan :

a. Mengoptimalkan Produksi danPemanfaatan Hasil Hutan.
b. Mendorong Peningkatan Perekonomian Masyarakat.

Sasaran :

a. Meningkatnya Pemanfaatan Hasil Hutan.
b. Meningkatnya Pemanfaatan Jasa Lingkungan Wisata.
c. Meningkatnya Peran Serta Masyarakat Sekitar Hutan dan di Kawasan Lindung.
d. Menciptakan Wirausahawan Baru.

3. Meningkatnya Pelayanan Publik dan Aparatur

Tujuan :

a. Meningkatkan Layanan Dasar Kepada Masyarakat dan Instansi Lain.
b. Meningkatnya Layanan Dasar Dalam Menunjang Kinerja Aparatur Dinas Kehutanan.

Sasaran :

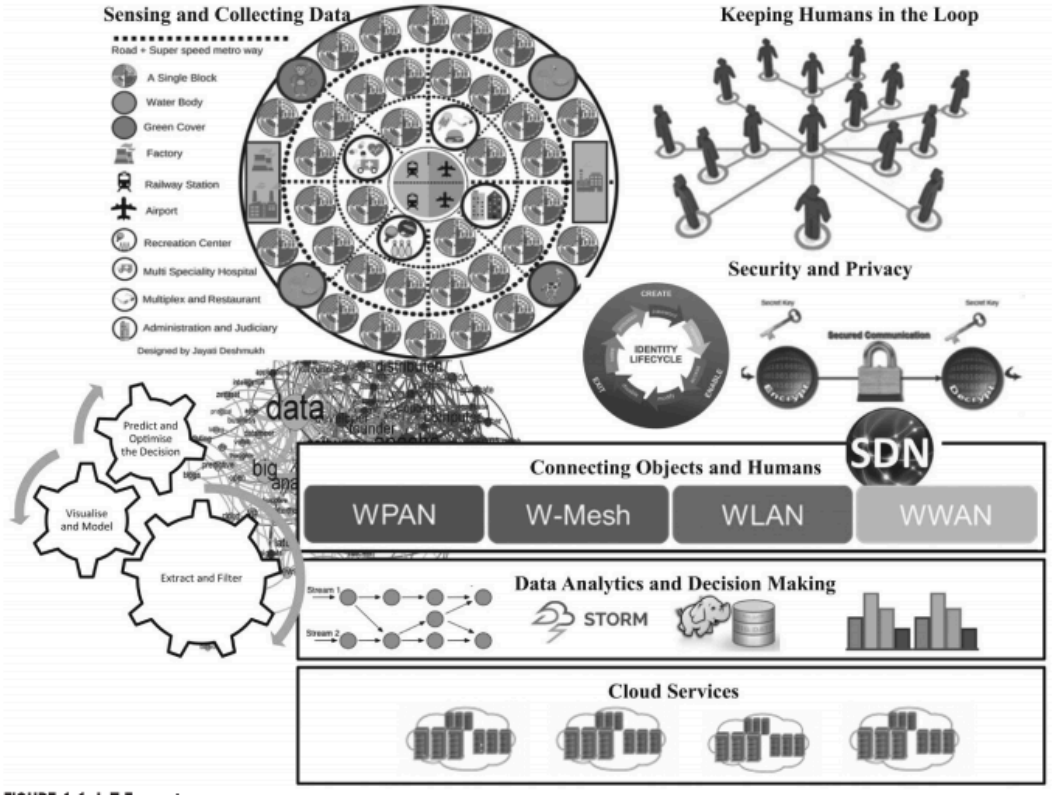
a. Terpenuhinya Informasi Kehutanan yang Berkualitas.
b. Terpenuhinya Sarana Prasarana dan Kebutuhan Administratif Aparatur.

2.2 *Internet Of Things* (IOT)

Setelah empat dekade sejak munculnya Internet oleh ARPANET, istilah internet mengacu pada kategori besar aplikasi dan protokol yang dibangun di atas jaringan komputer yang canggih dan saling berhubungan, melayani miliaran pengguna di seluruh dunia dalam 24/7. Fokusnya telah bergeser ke arah integrasi manusia dan perangkat tanpa adanya batasan untuk meliput ranah fisik dengan lingkungan manusia, dan terciptalah *Internet Of Things* (IOT).

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. Konteksnya mampu berkomunikasi dengan entitas lain, membuatnya dapat diakses kapan saja, dimana

saja. Secara garis besar dengan *Internet Of Things* (IOT) objek harus dapat diakses tanpa batasan waktu atau tempat. Pada Gambar 2.2 menunjukan *ecosystem Internet Of Things* (IOT).



Gambar 2.2 Ecosystem Internet Of Things (IOT) **[23]**

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 [23]. Pada penelitian ini konsep *Internet Of Things* (IOT) digunakan sebagai konsep utama manusia dengan lingkungan untuk melakukan

monitoring ruangan pengujian benih tanaman hutan dengan menggunakan microcontroller *raspberry pi,* sensor yang digunakan adalah sensor DHT 22 untuk mendapatkan keadaan suhu dan kelembaban ruangan, sensor Soil Moisture untuk mendapatkan keadaan kelembaban tanah dan modul relay untuk melakukan pengontrolan *absorbing fan* dan *water jet pump*.

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* [23]. 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 applikasi.

   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 masi 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 [23].

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. [23].

2.2.2 Monitoring and Actuating

Pemantauan perangkat melalui API dapat membantu menjadi lebih mudah. API dapat melaporkan penggunaa daya, kinerja peralatan dan status sensor sehingga alat dapat melakukan tindakan setelah mengirim perintah yang telah di tentukan. *Real-time application* dapat memanfaatkan fitur API untuk melaporkan

status sistem saat ini, seadngkan manajer dan pengembang memiliki opsi untuk secara bebas memanggil API tanpa perlu mengakses perangkat secara fisik atau *remote devices*, dapat membatnu mengidentifikasi cata produski atau kinerja melalui penerapan deteksi *anomaly* pada data yang dikumpulkan dengan demikian dapat meingkatkanya produktivitas [23].

2.3 Smart Energy

Konsep sistem *smart energy* diperkenalkan untuk mengidentifikasi potensi sinergi antar sub-sektor. Sebagai lawan, misalnya, smart grid dan konsep serupa, yang mengambil fokus utama pada sub-sektor yang bersangkutan, sistem *smart energy* memasukkan seluruh sistem energi dalam pendekatannya untuk mengidentifikasi desain infrastruktur energi yang sesuai dan strategi operasi. Hipotesisnya adalah bahwa solusi yang paling efektif dan paling murah ditemukan ketika masing-masing sub-sektor digabungkan dengan sektor-sektor lainnya.

Salah satu poin utamanya adalah bahwa analisis masing-masing teknologi dan sektor bersifat kontekstual dan, untuk melakukan analisis yang tepat, kita harus sistem energi keseluruhan di mana infrastruktur harus beroperasi. Poin utama lainnya adalah bahwa berbagai sub-sektor memengaruhi yang lain dan kita harus mempertimbangkan pengaruh tersebut jika solusi terbaik harus diidentifikasi [13]. Pada penelitian ini konsep *smart energy* (*solar energy*) digunakan untuk memanfaatkan cahaya matahari di ubah menajdi tenaga listrik dan di simpan didalam baterai sehingga menjadi sumber daya listrik utama ke microcontroller dan juga beberapa alat lainya.

2.3.1 Pembangit Listrik Tenaga Surya (PLTS)

Pembangkit Listrik Tenaga Surya adalah sebuah daya alternatif dari daya konvensional (PLN), PLTS yang bersumber dari sinar matahari ini tidak memiliki keterbatasan di Negara Indonesia, karena termasuk Negara Tropis sehingga hampir satu tahun penuh matahari akan terus menyinari daratan ini [3].

2.3.2 *Solar Panel*

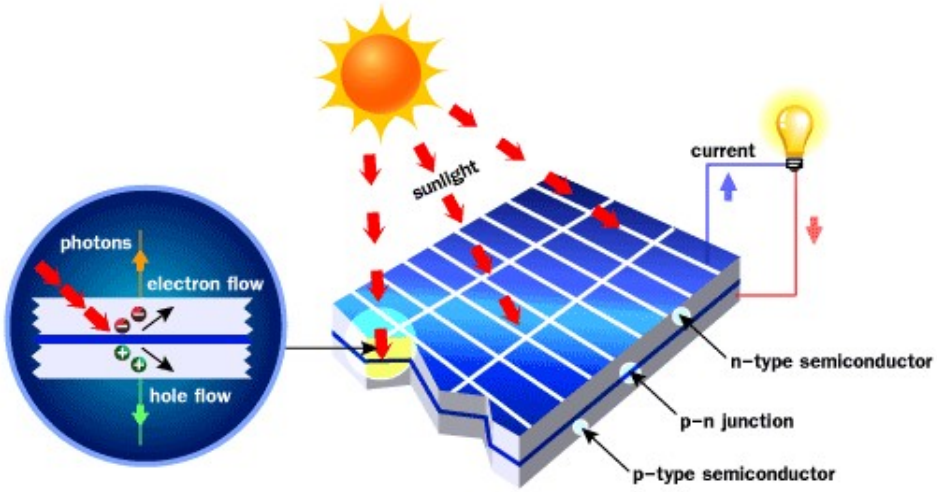
Solar panel adalah alat yang terdiri dari sel surya yang mengubah energi cahaya menajdi energi listrik arus searah (DC) dengan sel-sel fotovaltaik.. *Solar panel* memiliki umur lebih dari 20 tahun dan dalam kurun waktu tersebut *solar panel* akan mengalami penurunan efisiensi yang signifikan. Gambar 2.3 ini merupakan contoh panel surya.

***Gambar 2.3 Solar Panel* [8]**

Posisi ideal *solar panel* adalah menghadap langsung ke sinar matahari agar solar panel langsung menerima cahaya matahari secara maksimum. *Solar panel* modern memiliki perlindungan *overheating* yang baik dalam bentuk semen konduktif termal. Perlindungan *overheating* penting dikarenakan panel surya mengkonversi kurang dari 20% dari energi surya yang ada menjadi listrik, sementara sisanya akan terbuang sebagai panas, dan tanpa perlindungan yang memadai kejadian *overheating* dapat menurunkan efisiensi panel surya secara seignifikan [8].

2.3.3 Prinsip Kerja *Solar Panel*

Solar panel bekerja dengan menggunakan prinsip p-n *junction*, adapun semikonduktor yang biasanya digunakan adalah silikon. Namun silikon murni tidak memiliki elektron bebas sehingga konduktivitasnya buruk, untuk mengubah sifat kelistrikan silikon, maka perlu didoping dengan *atom* yang mimliki elektrovalensi 3 seperti *Boron,* seperti pada Gambar 2.4.



***Gambar 2.4 Prinsip Kerja Solar Panel* [8]**

Pada silikon tipe ini, terjadi kekurangan elektron karena *boron* merupakan unsur yang memliki valensi 3 sehingga muncul *hole* yang bersifat sebagai *akseptor*. Sedangkan untuk memperoleh silikon tipe-n maka perlu didoping dengan atom yang memiliki elektrovalensi 5, contohnya *Fosfor*. *Atom fosfor* merupakan unsur yang memiliki valensi 5 sehingga ketika mendoping akan muncul kelebihan electron yang bersifat donor.

Ketika kedua jenis semikonduktor ini melakukan kontak, maka kelebihan electron di semikonduktor tipe-n akan bergerak menuju *hole* pada semikonduktor tipe-p. Akibatnya akan terbentuk kutub positif pada tipe-p dan terbentuk kutub negatif pada tipe-n. Kemudian karena elektron dan *hole* berpasangan maka akan terbentuk medan listrik. Ketika cahaya matahari menyinari *solar panel*, maka tiap satu *foton* (partikel matahari) akan menghasilkan 1 elektron dan 1 *hole* yang jiga masuk jangkauan medan listrik, elektron akan bergerak ke semikonduktor tipe-n

dan *hole* akan bergerak ke semikonduktor tipe-p. Apabila dberi jalur arus eksternal, maka elektron akan bergerak ke tipe-p dan bersatu dengan *hole*. Aliran elektron yang bergerak akan menghasilkan arus, dan medan listrik akan menghailskan tegangan. Maka akan diperoleh daya listrik yang dibutuhkan [8].

2.3.4 Daya Listrik Panel Surya

Sebuah panel surya memiliki nilai WP masing-masing, WP adalah singkatan dari *Watt-Peak*, mengGambarkan besarnya daya listrik tertinggi yang dapat dihasilkan oleh suatu *solar panel*. *Solar panel* 50 WP menandakan bahwa daya listrik maksimal yang dapat dihasilkan oleh panel surya tersebut sebesar 50 watt setiap jam pada saat matahari terik, beberapa faktor yang dapat memepengaruhi kinerja panel surya seperti cuaca mendung akan berakibat pada minimnya cahaya matahari [8].

Rumus daya listrik yang dihasilkan pada panel surya dalam satu hari adalah:

$$P = WP \times n \qquad (2.1)$$

Keterangan :

P : Daya yang dihasilkan dalam satu hari (*Watt Hour*).

WP : Besar daya listrik tertinggi yang dihasilkan *solar panel* (*Watt Peak*).

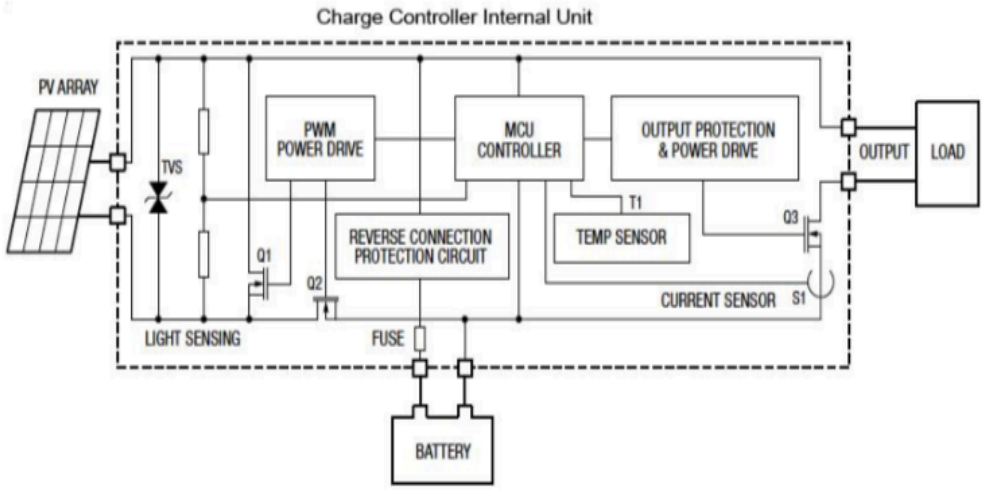
n : Lama terik. matahari menyinari *solar panel* dalam satu hari.

2.3.5 *Solar Charge Controller*

Solar charge controller merupakan alat yang penting dalam pengisian *solar panel* ke baterai. *Controller* ini digunakan untuk mengatur arus searah dari panel surya ke baterai dan arus yang diambil dari baterai ke beban. *Controller* mampu mengatasi *overcharging* (kelebihan pengisian karena baterai sudah terisi penuh) dan kelebihan tegangan dari panel surya. Kelebihan tegangan dan pengisian akan mengurangi umur baterai. Gambar 2.5 ini merupakan contoh *solar charge controller*.

***Gambar 2.5 Solar Charge Controller* [9]**

Controller yang baik memiliki kemampuan mendeteksi kapasitas baterai pada saat pengisian arus. Bila baterai telah terisi penuh maka secara otomatis pengisian arus dari *solar panel* berhenti. *Controller* terdiri dari 1 *input* yang terhubung dengan *solar panel*, 1 *output* yang terhubung dengan baterai dan 1 *output* yang terhubung dengan beban. Arus listrik yang berasal dari baterai tidak mungkin berbalik arah masuk ke *solar panel* karena pada *controller* terdapat *diode protection* yang hanya melewatkan arus listrik dari *solar panel* ke baterai [8]. Gambar 2.6 ini merupakan blok diagaram *solar charge controller*.



***Gambar 2.6 Block Diagram Solar Charge Controller* [8]**

Penjelasan singkat mengenai blok diagram *solar charge controller* :

a. *MCU Controller*

MCU controller berfungsi sebagai otak dari *solar charge controller*, sebuah chip yang berfungsi sebagai pengontrol rangkaian elektronik.

b. PWM Power Drive

Solar charge controller menggunakan lebar *pulse* dari *on* dan *off* elektrikal sehingga menciptakan seakan-akan *sine wave electrical form*. Lamanya arus *pulse* yang sedang diisi ulang secara perlahan-lahan berkurang sebagaimana tegangan baterai meningkat mengurangi rata-rata arus yang masuk ke dalam baterai.

c. Reverse Connection Protection Circuit

Fungsi dari *reverse connection protection circuit* yaitu untuk memproteksi arus yang masuk ke baterai. Apabila baterai telah terisi penuh maka secara otomatis pengisian daya dari *solar panel* berhenti.

d. Output Protection & Power Driver

Sama seperti *reverse connection protection circuit* bagian ini berfungsi untuk memproteksi output yang menuju beban. Apabila arus masuk ke beban maka fungsi pengisian pada baterai akan terhenti dan sebaliknya apabila baterai mulai mengisi, *output* tidak dapat mengalir ke beban. Bagian ini juga berfungsi sebagai pemutus apabila daya yang digunakan ke beban melewati batas minimal dari baterai sehingga baterai akan lebih tahan lama.

e. Sensor Temperatur

Sensor temperature berfungsi untuk mendeteksi temperature pada *solar charge controller*. Kelebihan daya yang masuk akan diubah menjadi panas oleh karena itu dibutuhkan monitoring suhu pada *controller*.

f. Sensor Arus

Sensor arus berfungsi untuk memonitoring arus yang mengalir pada beban.

Solar Charge Controller memiliki dua mode kerja, yaitu *charging mode* atau mode pada saat panel surya melakukan pengisian arus dan tegangan pada baterai dan *operation mode* atau mode pada saat penggunaan baterai oleh beban. Pada saat *charging mode*, umumnya terdapat tiga keadaan selama pengisian yaitu :

a. Fase *Bulk*

Baterai di-*charge* dengan tegangan *bulk* yaitu antara 14.4V-14.6V dan arus maksimum dari *solar panel*. Ketika kapasitas tegangan baterai sudah sama dengan tegangan *bulk*, maka fase *absorption* dimulai.

b. Fase *Absorption*

Pada fase inim tegangan baterai akan dijaga agar tetapi sesuai dengan tegangan *bulk*. Arus yang dialirkan akan menurun sehingga mencapai kapasitas dari baterai.

c. Fase *Float*

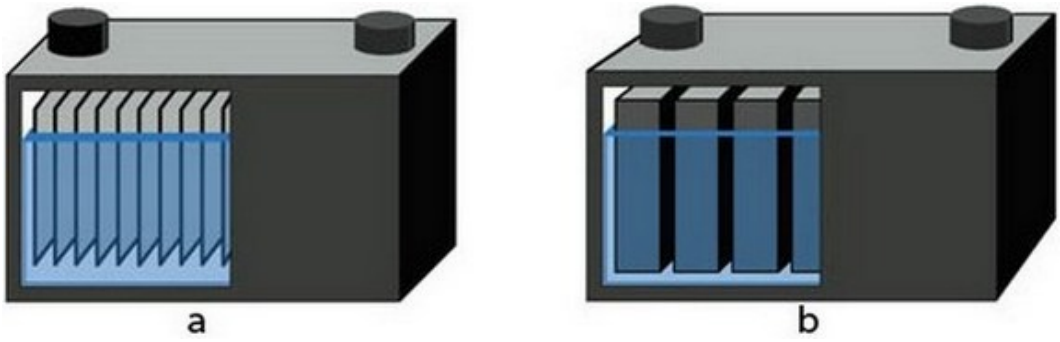
Tegangan baterai akan dijaga pada tegangan *float* yaitu antara 13.4V-13.7V. pada fase ini beban sudah dapat menggunakan arus dari baterai.

Pada saat *Operation Mode*, baterai sudah dapat digunakan oleh beban. Jika terjadi *over load*, maka baterai akan dilepas dari beban oleh *controller*.

2.3.6 Baterai / AKI

Aki adalah media penyimpan muatan listrik. Secara garis besar aki dibedakan berdasarkan aplikasi dan konstruksi. Berdasarkan aplikasi maka aki dibedakan untuk engine starter (otomotif) dan deep cycle. Aki otomotif umumnya dibuat dengan pelat timbal yang tipis namun banyak sehingga luas permukaannya lebih besar. Dengan demikian aki ini bisa menyuplai arus listrik yang besar pada saat awal untuk menghidupkan mesin. Aki *deep cycle* biasanya digunakan untuk sistem *fotovoltaik* (*solar sell*) dan *back up power*, dimana aki mampu mengalami

discharge hingga muatan listriknya tinggal sedikit [8]. Gambar 2.7 ini perbedaan Jenis Aki starter dengan *deep cycle*.



***Gambar 2.7 (a) Aki Starter dan (b) Aki Deep Cycle* [8]**

Jenis aki starter atau otomotif sebaiknya tidak mengalami discharge hingga melampaui 50% kapasitas muatan lsitriknya untuk menjaga keawetan aki. Apabila muatan aki basah sampai di bawah 50% dan dibiarkan dalam waktu lama (berhari-hari tidak di-charge kembali), maka kapasitas muat aki tersebut akan semakin berkurang sehingga menjadi tidak awet. Berkurangnya kapasitas muat aki tersebut karena proses pembentukan kristal sulfat yang menempel pada pelat ketika muatan aki tidak penuh (di bawah 50%).

2.4 Benih Tanaman Hutan

Benih adalah biji yang dipersiapkan untuk tanaman, telah melalui proses seleksi sehingga diharapkan dapat mencapai proses tumbuh yang besar. Benih tanaman hutan adalah segala sesuatu yang berkaitan dengan pembangunan sumberdaya genetic, permuliaan tanaman hutan, pengadaan dan pengedaran benih dan bibit, serta sertifikasi, dan bahan tanaman yang berupa bahan genetatif (biji) atau bahan vegetative yang digunakan untuk mengembangbiakan tanaman hutan [5]. Pada penelitian ini benih digunakan sebagai parameter utama dalam pengujian di ruangan pengujian benih tanaman hutan.

2.5 *Unified Modeling 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 elum dibuat sama sekali. Sebagai bahasa pemograman, UML dapat menterjemahkan diagram yang ada di UML menjadi code program yang siap untuk di jalankan [14].

Struktur sebuah sistem dideskripsikan dalam 5 view diman asalah 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 erbeda akan menekankan pada aspek yang berbeda dari siste 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 mengantdung semua *view* yang lain yang mendeskripsikan aspek-aspek tertentu dari rancangan sistem. Itulah sebabnya *use case view* menajdi pusat peran dan sering dikatakan yang mengdrive 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. Yang termasuk dalam *physical* view ini adalah *deployment diagram* dan *timing diagram*.
5. *Process View*, berhubungan erat dengan aspek dinamis dari sistem, proses yang terjadi di sistem dan bagaimana komunikasi yang terjadi di sistem serta tingkah laku sistem saat dijalankan. *Process view* menjelaskan apa itu *concurrency*, *distribusi integrasi*, *kinerja* dan lain-lain. Yang termasuk dalam process *view* adalah *activiy diagram*, *communication diagram*, *sequence diagram* dan *interaction overview diagram*.

Mesikpun UML sudah cukup banyak menyediakan diagram yang bisa membantu mendefinisikan sebuah aplikasi, tidak berarti bahwa semua diagram tersebut akan bisa menjawab persoalan yang ada. Dalam banyak kasus, diagram lain selain UML sangat banyak membantu. Pada penelitian ini konsep UML digunakan untuk mengGambarkan bagaimana sistem bekerja, hubungan antar class, memberi Gambaran kepada user sistem yang akan di gunakan dan memberikan penjelasan detail pemanggilan fungsi-fungsi atau method dalam class [14].

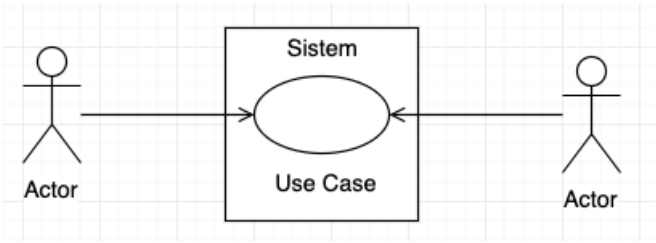
2.5.1 Diagram UML

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 2.8 menunjukan *Use Case Diagram* dalam UML [14].



***Gambar 2.8 Use Case Diagram* [14]**

Berikut ini adalah bagian dari sebuah use case diagram :

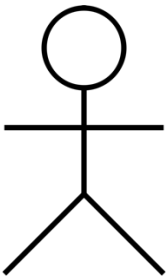
a. Use Case

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 aplikai, bukan 'bagaimana' software aplikasi mengerjakannya. Setiap *user case* harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaksinya dengan *actor*. Nama *use case* boleh terdiri dari beberapa kata dan tidak boleh ada dua *use case* yang memiliki nama yang sama. Gambar 2.9 menunjukan bentuk *Use Case* dalam UML.

***Gambar 2.9 Use Case* [14]**

b. *Actors*

Actors adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncukl dalam beberapa peran. Bahwa actor berinteraksi dengan *use case*, tetapi tidak memiliki control atas *use case*. Gambar 2.10 menunjukan bentuk *actor* dalam UML.



***Gambar 2.10 Actor* [14]**

c. *Relationship*

Relationship adalah hubungan antara *use cases* dengan *actors* [14]. *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*

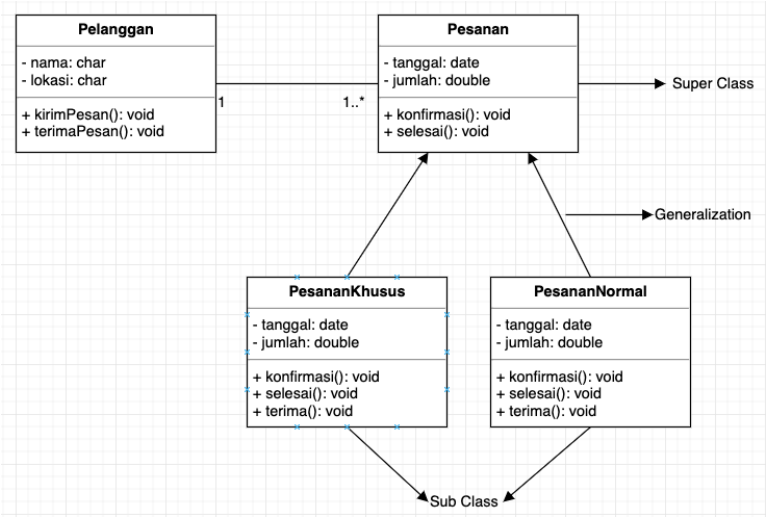
Hubungan *inheritance* (pewarisan) yang melibatkan *actor* yang satu (*the child*) dengan *actor* lainnya (*the parent*). Generalisasi tipe ini menggunakan garis lurus dengan mata panah tertutup di ujungnya.

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,o 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 [14]. Gambar 2.11 menunjukan *Class Diagram* dalam UML.



***Gambar 2.11 Class Diagram* [14]**

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

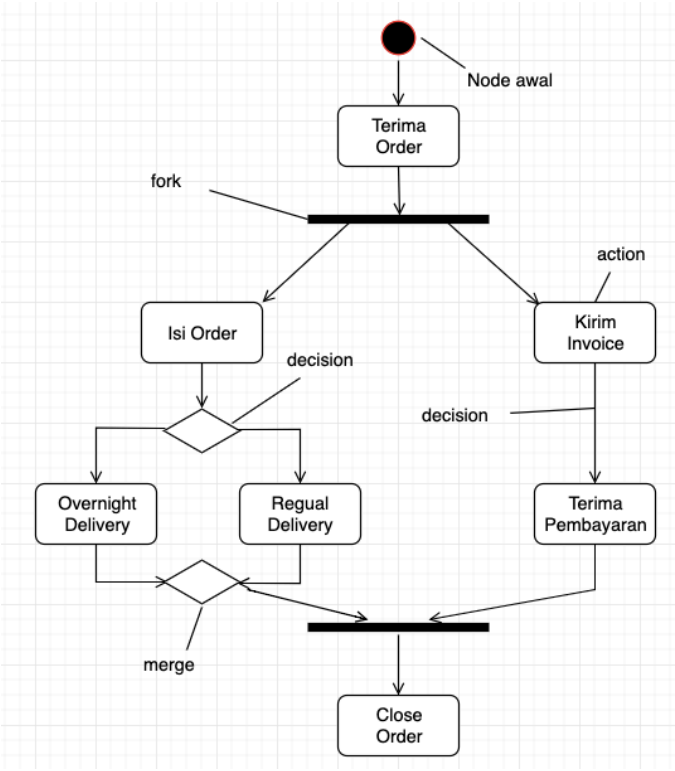
c. *Constraint*

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 sctivity 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. Langa selanjutnya perlu penjabaran tata letak dari keseluruhan aliran agara bisa ditransofrmasikan ke activity diagram [14]. Gambar 2.12 menunjukan *Activity Diagram* dalam UML.



***Gambar 2.12 Activity Diagram* [14]**

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.

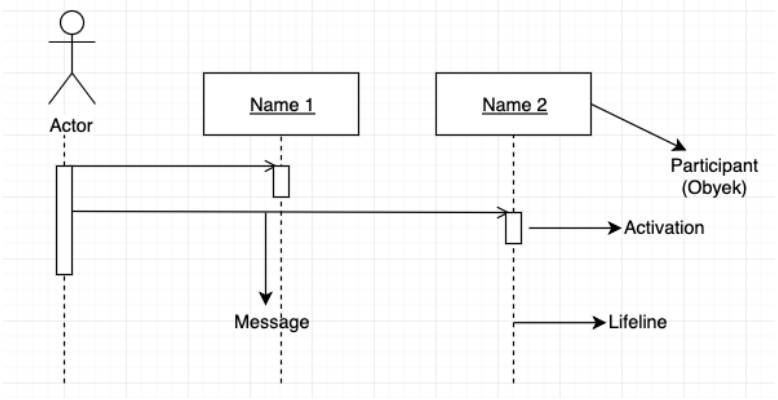
g. *Final state*

   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 ururnsannya. 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 [14]. Gambar 2.13 menunjukan *Sequence Diagram* dalam UML.



***Gambar 2.13 Sequence Diagram* [14]**

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.

2.6 *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 [14]. 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 ada 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.

2.7 *Entity Relationship Diagram* (ERD)

Entity Relationship Diagram (ERD) adalah model teknik pendekatan yang menyatakan atau mengGambarkan hubungan suatu model. Didalam hubungan ini tersebut dinyatakan yang utama dari ERD adalah menunjukan objek data (*Entity*)

dan hubungan (*Relationship*), yang ada pada Entity berikutnya. Proses memungkinkan analis menghasilkan struktur basis data dapat disimpan dan diambil secara efisien [15]. Pada penelitian ini konsep ERD digunakan untuk merancang database yang akan di gunakan dalam sistem. Simbol-simbol dalam *Entity Relationship Diagram* (ERD) adalah sebagai berikut:

1. Entitas, suatu yang nyata atau bastrak yang mempunyai karakteristik dimana kita akan menyimpan data.
2. Atribut, ciri umum semua atau sebagian besar instansi pada entitas tertentu.
3. Relasi, hubungan alamiah yang terjadi antara satu atau lebih entitas.
4. Link, garis penghubung atribut dengan kumpulan entitas dan kumpulan entitas dengan relasi.

Hubungan kardinalitas relasi dalam *Entity Relationship Diagram* (ERD) adalah sebagai berikut:

1. Satu ke satu (One to One)

   Setiap elemen dari Entitas A berhubungan paling banyak dengan elemen pada Entitas B. Demikian juga sebaliknya setiap elemen B berhubungan paling banyak satu elemen pada Entitas A.
2. Satu ke banyak (One to Many)

   Setiap elemen dari Entitas A berhubungan dengan maksimal banyak elemen pada Entitas B. Dan sebaliknya setiap elemen dari Entitas B berhubungan dengan paling banyak satu elemen di Entitas A.
3. Banyak ke satu (Many to One)

   Setiap elemen dari Entitas A berhubungan paling banyak dengan satu elemen pada Entitas B. Dan sebaliknya setiap elemen dari Entitas B behubungan dengan maksimal banyak elemen di entitas A.
4. Banyak ke banyak (Many to Many)

   Setiap elemen dari Entitas A berhubungan maksimal banyak elemen pada Entitas B demikian sebaliknya.

2.8 *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 [18]. Pada penelitian ini MySQL digunakan sebagai platform penyimpanan data dan implementasi model rancangan database yang sudah di buat. Gambar 2.14 menunjukan logo *MySQL*.

***Gambar 2.14 Logo MySQL* [18]**

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.

2.9 *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 [16]. 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 – servicesnya. Pada penelitian ini bahasa pemograman python digunakan sebagai

komponen utama untuk pembanguan website dan konfigurasi *raspberry pi*. Gambar 2.15 menunjukan logo *python*.

***Gambar 2.15 Logo Python* [16]**

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.

2.10 Flask Web Microframework

Flask adalah kerangka kerja kecil oleh sebagian besar standar, cukup kecil untuk disebut "*microframework*." Ini cukup kecil sehingga setelah terbiasa dengannya, mungkin akan menjadi mampu membaca dan memahami semua kode sumbernya.

Flask sudah dikenal sebagai *framework* yang dapat diperpanjang dari bawah ke atas, ini memberikan inti yang solid dengan layanan dasar, sementara ekstensi menyediakan sisanya. Karena user dapat memilih paket ekstensi yang diinginkan, user berakhir dengan *stack* ramping yang tidak *bloat* dan melakukan apa yang dibutuhkan.

Flask memiliki dua dependensi utama. *routing*, *debugging*, dan *Web Server Gateway Interface* (WSGI) berasal dari *Werkzeug*, sementara dukungan template disediakan oleh *Jinja2*. *Werkzeug* dan *Jinja2* dikembangkan oleh pengembang inti Flask [17]. Pada penelitian ini microframework flask digunakan untuk pembuatan website. Gambar 2.16 menunjukan logo *flask microframework*.

***Gambar 2.16 Logo Flask* [17]**

2.11 *Codeigniter*

Codeigniter adalah sebuah framework php yang bersifat open source dan menggunakan metode MVC (*Model, View, Controller*). *Codeigniter* bersifat *free*, framework codeigniter dibuat dengan tujuan sama seperti framework lainnya yaitu

untuk memudahkan developer atau programmer dalam membangun sebuah aplikasi berbasis web tanpa harus membuat nya dari awal. MVC adalah teknik atau konsep yang memisahkan komponen utama menjadi tiga komponen yaitu model, view dan controller.

Gambar 2.17 Logo Codeigniter

2.12 *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 (Java, 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 di bawah 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 [19]. Pada penelitian ini JSON digunakan sebagai API untuk mengirim data sensor dari *raspberry pi* ke website.

2.13 *PyCharm*

PyCharm adalah *integrated development environment* (IDE) yang digunakan dalam pemrograman komputer, khusus untuk bahasa *Python*. Ini dikembangkan oleh perusahaan *JetBrains Ceko*. Ini menyediakan analisis kode, debugger grafis, unit tester terintegrasi, VCSes, dan dukungan pengembangan web dengan *flask* [21]. Pada penelitian ini pycharm digunakan sebagai IDE pengkodean website dan *raspberry pi*. Gambar 2.17 menunjukan logo *PyCharm*.

***Gambar 2.18 Logo PyCharm* [21]**

Fitur-fitur yang tersedia dalam *pycharm* adalah sebagai berikut :

1. *Coding assistance*, *analysis*, *code completition*, *error highlighting*, *linter integration*, dan *quick fixes*.
2. Navigasi proyek dan kode: tampilan proyek khusus, tampilan struktur file, dan lompatan cepat antara file, kelas, metode, dan penggunaan.
3. *Python refactoring*: termasuk mengganti nama, mengekstrak metode, memperkenalkan variabel, memperkenalkan konstan, *pull up*, *push down* dan lainnya.
4. Dukungan untuk *web framework*: *Django*, *web2py*, dan *Flask*.
5. *Integrated Python debugger*.
6. *Version control integration* : *Mercurial*, *Git*, *Subversion*, *Perforce* dan *VCS* dengan *changelists* dan *merge*.

2.14 *Fritzing*

Fritzing merupakan salah satu dari sekian banyak aplikasi untuk mendesain rangkaian elektronika. Sama halnya seperti aplikasi lain (*Eagle*, *Protel*, *Pad2pad*, dll) *Fritzing* juga memiliki *Board Designer* (untuk membuat jalur PCB). Pada penelitian ini fritzing digunakan untuk mendesain komponen *raspberry pi*, sensor-sensor dan modul yang digunakan [22]. Gambar 2.18 menunjukan logo *fritzing*.

***Gambar 2.19 Logo Fritzing* [22]**

Fritzing sedikit berbeda dengan aplikasi-aplikasi lainnya, kelebihan yang ada pada *fritzing* di antaranya :

1. Open Source, Sudah menyediakan banyak library, bahkan untuk mikrokontroller buatan seperti *Arduino*, *Raspberry Pi*, *basic stamp*, *pic.exe*, dan lain-lain. Serta librarynya juga banyak tersedia di internet dan bisa di download secara gratis.
2. Gambar rangkaian bisa di-*Share* di internet melalui aplikasi ini, jadi Gambar rangkaiannya bersifat *Open Hardware*.
3. Dapat mengGambar rangkaian di *BreadBoard*. fitur seperti ini sangat membantu bagi pemula yang ingin belajar rangkaian elektronika, karena Gambar rangkaiannya sangat mudah dimengerti, bahkan bisa ditiru langsung ke BreadBoard.

2.15 *Access Point*

Access Point adalah sebuah perangkat jaringan yang berisi sebuah transceiver dan antena untuk mentransmisikan dan menerima sinyal ke dan dari clients remote. Dengan *access points* (AP) clients *wireless* bisa dengan cepat dan mudah untuk terhubung kepada jaringan LAN kabel secara *wireless*. Dengan kata lain *access*

point merupakan sebuah alat yang digunakan untuk menghubungkan alat-alat dalam suatu jaringan, dari dan ke jaringan wireless [8]. Pada penelitian ini konsep access point digunakan untuk menghubungkan alat *raspberry pi* di ruangan pengujian benih tanaman hutan dengan ruangan atau kantor penguji melalui *Wi-Fi*.

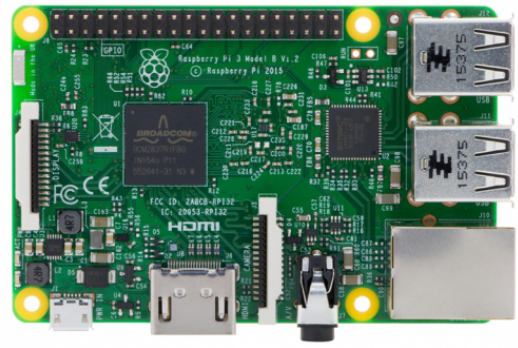
Secara garis besar, access Point berfungsi sebagai pengatur lalu lintas data, sehingga memungkinkan banyak client dapat saling terhubung melalui jaringan (network). Sedangkan jika diperinci lebih jelas lagi fungsi access point adalah sebagai berikut :

1. Mengatur supaya *access point* dapat berfungsi sebagai *DHCP server*.
2. Menjalankan fitur *Wired Equivalent Privacy* (WEP) dan *Wi-Fi Protected Access*(WPA) untuk keamanan jaringan.
3. Mengatur akses berdasarkan *MAC address device* pengakses.

Sebagai *Hub/Switch* yang bertindak untuk menghubungkan jaringan lokal dengan jaringan *wireless/nirkabel*.

2.16 Raspberry Pi 3 Model B

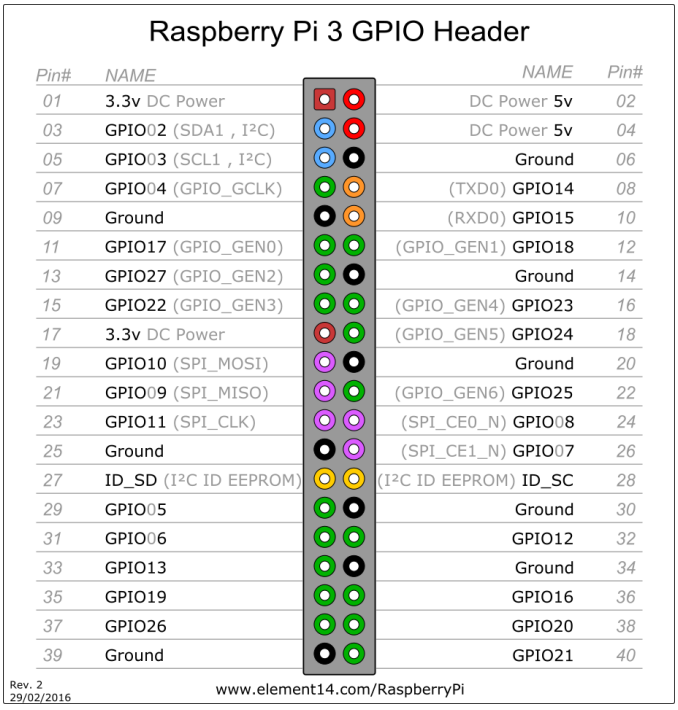
Raspberry Pi 3 Model B merupakan generasi ketiga dari keluarga *raspberry pi*. *raspberry pi* 3 memiliki *RAM* 1GB dan grafis *Broadcom VideoCore* IV pada frekuensi *clock* yang lebih tinggi dari sebelumnya yang berjalan pada 250MHz. *Raspberry Pi* 3 juga memiliki 4 *USB port*, 40 *pin GPIO*, *Full HDMI 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* [10]. Pada penelitian ini raspberry pi digunakan sebagai mini pc untuk server website dan juga pengontrolan sensor dan modul relay di ruangan pengujian benih tanaman hutan. Gambar 2.19 adalah bentuk *raspberry pi* 3 model b.



***Gambar 2.20 Raspberry Pi 3 Model B* [10]**

2.16.1 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 2.22.



***Gambar 2.21 Pin GPIO Raspberry Pi 3 Model B* [10]**

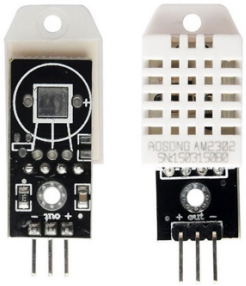
2.17 Sensor

Sensor adalah suatu perangkat yang mendeteksi perubahan energi yang berada di alam seperti energi listrik, energi fisika, energi kimia, energi biologi, energi mekanik dan sebagainya. Di dalam sebuah sensor terdapat transduser yang berfungsi untuk mengubah besaran mekanis, magnetis, panas, sinar, dan kimia menjadi besaran listrik berupa tegangan, resistansi dan arus listrik [6].

2.17.1 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 [11].

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. Pada penelitian ini sensor DHT 22 digunakan untuk mendapatkan suhu dan kelembaban ruangan pengujian benih tanaman hutan. Gambar 2.21 merupakan bentuk sensor DHT 22.



Gambar 2.22 Sensor DHT 22 **[11]**

2.17.2 Sensor *Soil Moisture*

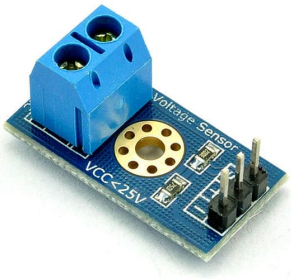
Sensor *Soil moisture* adalah sensor kelembaban tanah yang bekerja dengan prinsip membaca jumlah kadar air dalam tanah di sekitarnya. Sensor ini merupakan sensor ideal untuk memantau kadar air tanah untuk tanaman. Sensor ini menggunakan dua konduktor untuk melewatkan arus melalui tanah, kemudian membaca nilai resistansi untuk mendapatkan tingkat kelembaban [12]. Pada penelitian ini sensor soil moisture digunakan untuk mendapatkan status keadaan tanah kering atau tidak di ruangan pengujian benih tanaman hutan. Gambar 2.22 merupakan bentuk sensor *Soil Moisture*.

***Gambar 2.23 Sensor Soil Moisture* [12]**

Lebih banyak air dalam tanah akan membuat tanah lebih mudah menghantarkan listrik (nilai resistansi lebih besar), sedangkan tanah kering akan mempersulit untuk menghantarkan listrik (nilai resistansi kurang). Sensor soil moisture dalam penerapannya membutuhkan daya sebesar 3.3 v atau 5 V dengan keluaran tegangan sebesar 0 – 4.2 V.

2.17.3 Sensor Tegangan

Prinsip kerja modul sensor tegangan yaitu didasarkan pada prinsi penekanan resistansi dan dapat membuat tegangan input berkurang hingga 5 kali dari tegangan asli. Sehingga sensor hanya mampu membaca teganga maksimal 25 V bila menggunakan vcc 5V dan maksimal 16.5V jiga menggunakan tegangan vcc 3.3V [25]. Gambar 2.23 menunjukan bentuk sensor tegangan.

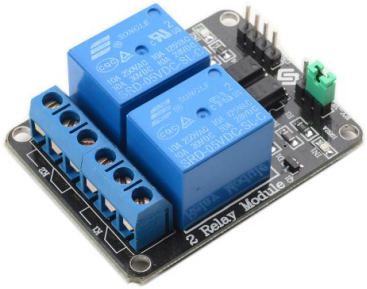


Gambar 2.24 Sensor Tegangan [25]

Pada dasarnya pembacaan sensor hanya dirubah dalam bentuk bilangan 0 sampai 256 karena menggunakan ADC 8 bit. Resolusi simulsi modul 5V / 256 = 0,01953V, sehingga untuk vcc 5V dapat dirumuskan menggunakan persamaan Volt = (Vout * 5) / 256.

2.18 Relay

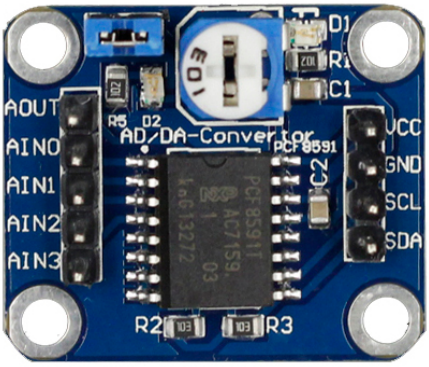
Relay adalah saklar yang dioperasikan secara elektrik. *Relay* dipakai ketika sinyal berdaya rendah digunakan untuk mengontrol sebuah rangkaian (isolasi elektrik penuh terjadi antara rangkaian pengontrol dan rangkaian yang dikontrol) atau ketika beberapa sirkuit harus dikontrol oleh satu sinyal. Relay 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 [8]. Pada penelitian ini modul relay digunakan untuk mengontrol absorbing fan berfungsi mengeluarkan udara panas di dalam ruangan dan menyalakan water jet pump ketika keadaan tanah sudah kering. Gambar 2.25 merupakan bentuk relay.



Gambar 2.25 Relay **[8]**

2.19 *Analog to Digital Converter* PCF8591

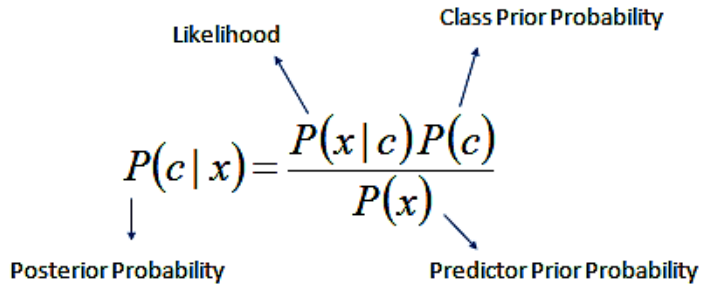
PCF8591 adalah perangkat akusisi data CMOS 8-bit berdaya rendah satu catu daya tunggal dengan empat input analog, satu output analog dan antarmuka bus serial I2C. Tiga pin alamat A0, A1 dan A2 digunakan untuk memprogram alamat perangkat keras, memungkinan penggunakan sehingga delapan perangkat yang terhubung ke bus-I2C tanpa perangkat keras tambahan. Alamat, control dan data ked an dari perangkat ditransfer secara serial melalui dua jaur dua arah bus I2C. Fungsi perangkat ini meliputi multiplexing input analog, fungsi on-chip track and hold, konversi analog-ke-digital 8-bit dan konversi digital-ke-analog 8-bit. Tingkat konversi maksimum diberikan oleh kecepatan maksimum bus-I2C [26]. Gambar 2.25 menunjukan bentuk sensor adc pcf8591.



Gambar 2.26 ADC PCF8591 ***[26]***

2.20 *Naïve Bayes*

Klasifikasi *Naïve Bayes* sangat sederhana yang mengasumsikan bahwa atribut klasifikasi independent dan tidak memilki korelasi di antara mereka. Banyak penliti telah menemukan bahwa asumsi independensi ini tidak berfungsi dalam semua kasus yang diusulkan metode alternative lain untuk mengingkatkan kinerja. Teknik *naïve bayes* didasarkan pada probabilitas bersyarat dan kemungkinan kejadian maksimum [27]. Rumus *naïve bayes* dapat dilihat pada Gambar 2.26.



$$P(c \mid \mathrm{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

Gambar 2.27 Naïve Bayes *[27]*

2.21 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.

2.21.1 *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 [24]. 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* [24]. 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 usuability 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.