BAB 2

LANDASAN TEORI

Bab ini membahas teori-teori dasar mengenai *Wireless Sensor Network* (WSN), seperti definisi dari WSN, jenis sensor pada WSN, struktur WSN, jenis komunikasi pada WSN, jenis-jenis aplikasi yang digunakan pada WSN, dan Preon32 sebagai node sensor yang akan dipakai.

2.1 Sensor

Subbab ini membahas definisi sensor dan jenis-jenis sensor yang telah dikembangkan.

2.1.1 Definisi Sensor [1]

Beberapa definisi sensor yang dikemukakan para ahli dijelaskan sebagai berikut:

1. Menurut Petruzela (2001)

   Alat untuk mendeteksi/mengukur sesuatu, yang digunakan untuk mengubah variasi mekanis, magnetis, panas, sinar, dan kimia menjadi tegangan dan arus listrik.

2. Menurut Fraden (2010)

   Perangkat yang menerima stimulus dan merespon dengan sinyal listrik.

3. Menurut Jones (2010)

   Perangkat yang merespon input fisik dengan keluaran yang dapat direkam dan berfungsi secara fungsional yang biasanya berupa listrik atau optik.

4. Menurut Chen, et al. (2012)

   Sebuah sensor umumnya mengacu pada perangkat yang mengubah pengukuran fisik menjadi sinyal yang dibaca oleh pengamat atau dengan instrumen.

2.1.2 Jenis-jenis Sensor [2]

Sensor memiliki parameter deteksi yang berbeda-beda. Sebuah sensor dapat memiliki banyak parameter deteksi. Tabel 2.1 menunjukkan jenis-jenis sensor yang telah dikembangkan hingga saat ini:

Tabel 2.1: Tabel jenis sensor

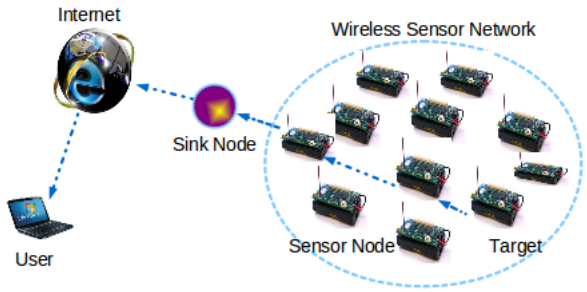
Jenis	Contoh Sensor
Suhu	Termistor, Termokopel
Tekanan	Pengukur tekanan, Barometer, Pengukur ionisasi
Optik	Fotodioda, Sensor inframerah, sensor CDC
Akustik	Resonator piezoelektrik, Mirkrofon
Mekanik	Pengukuran regangan, Sensor Taktil, Kapasitif diafragma
Getaran	Accelerometers, Gyroscopes, Sensor foto
Arus	Anemometer, Sensor aliran massa udara
Posisi	GPS, Sensor berbasis ultrasound, Sensor berbasis inframerah
Kimiawi	Sensor pH, Sensor elektrokimia, Sensor gas inframerah
Kelembaban	Sensor kapasitif dan resistif, Higrometer, Sensor kelembaban berbasis MEMS
Radiasi	Deteksi ionisasi, Penghitungan Geiger-Mueller

2.2 *Wireless Sensor Network* [2]

Subbab ini membahas mengenai definisi WSN, jenis-jenis node, komponen node sensor, protokol komunikasi, arsitektur WSN, topologi WSN, dan sistem operasi WSN.

2.2.1 Definisi *Wireless Sensor Network*

Wireless Sensor Network (WSN) adalah kumpulan node sensor yang disebar pada lingkungan, berinteraksi dengan node sensor lain, dan hasil data yang dikumpulkan akan diproses lebih lanjut oleh administrator berupa pekerja sipil, pemerintah, dan lain-lain. Data didapatkan dengan cara melakukan *sensing*(mendeteksi) pada lingkungan sekitar. Node sensor dan sensor adalah dua hal yang berbeda karena sensor termasuk ke dalam node sensor. Contoh dari WSN dapat dilihat pada Gambar 2.1[1].



Gambar 2.1: *Wireless Sensor Network.*

Aplikasi *Wireless Sensor Network* pada Berbagai Bidang

Wireless sensor network sudah mulai banyak digunakan untuk menyelesaikan permasalahan dalam kehidupan manusia. Beberapa bidang yang menggunakan *Wireless Sensor Network* adalah:

1. Aplikasi militer : *Wireless sensor network* digunakan sebagai pengendali, pengiriman pesan, komunikasi, dan pengamatan pada lingkungan peperangan.
2. Pengamatan lingkungan : Sensor node diletakkan pada beberapa bagian di mana sebuah fenomena ingin diamati.

[1]https://microcontrollerslab.com/wireless-sensor-networks-wsn-applications/

3. Transportasi : Melakukan pengamatan mengenai kondisi kemacetan suatu wilayah.
4. Aplikasi kesehatan : Beberapa aplikasi kesehatan berbasis sensor dapat membatu orang-orang yang memiliki kekurangan fisik, melakukan pemantauan terhadap pasien, dan mempermudah dokter untuk melakukan pengecekan terhadap pasien yang berada di rumah sakit.
5. Bidang pertanian : Penggunaan *Wireless Sensor Network* mempermudah petani untuk memantau keadaan lingkungan.
6. Pengamatan lingkungan : Banyak aplikasi *Wireless Sensor Network* yang digunakan untuk melakukan pengamatan mengenai kondisi bumi. Seperti mendeteksi aktivitas gunung berapi, laut, hutan, dan sebagainya.
7. Memantau kondisi bangunan : *Wireless Sensor Network* digunakan untuk melakukan deteksi pergerakan yang terjadi pada suatu bangunan, jembatan, terowongan, tanggul, dan sebagainya.

2.2.2 Jenis-jenis Node

Ada 3 jenis node yang terdapat pada WSN, yaitu:

- *Sensor Node*

  Sensor node adalah node yang berfungsi untuk membaca data lingkungan atau objek yang diteliti/diselidiki.

- *Router*

  Router adalah node yang berfungsi untuk meneruskan paket data dari sebuah node ke node lain. Node ini sangat berguna untuk komunikasi multi-hop.

- *Sink Node*

  Sink node adalah node yang berfungsi untuk mengumpulkan data dari *Sensor Node*, kemudian meneruskannya ke perangkat atau sistem lain, seperti ke *base-station*

2.2.3 Komponen Node Sensor [3]

Setiap node dalam WSN terdiri dari lima komponen dasar seperti pada gambar 2.2 [2]:

1. Perangkat Komunikasi

   Berfungsi untuk menerima/mengirim data ke node sensor lainnya atau *base-station.*

2. Mikrokontroler

   Berfungsi untuk melakukan fungsi perhitungan, mengontrol, dan memproses *device* yang terhubung dengan mikrokontroler.

3. Sensor

   Berfungsi untuk melakukan *sensing* pada lingkungan yang ingin diperiksa atau diteliti.

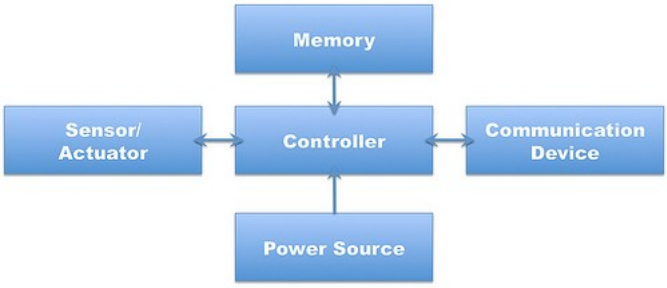
4. Memori

   Berfungsi sebagai tambahan memori bagi sistem *Wireless Sensor*, pada dasarnya sebuah unit mikrokontroler memiliki unit memori sendiri.

5. Power Supply

   Berfungsi sebagai sumber energi bagi node sensor agar bisa bekerja secara keseluruhan dengan memakai baterai sebagai sumber dayanya

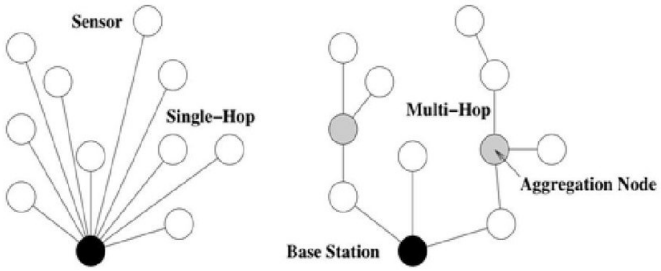
[2] https://botanmasure.wordpress.com/2015/08/19/pengertian-wireless-senssor-network/



Gambar 2.2: Komponen node sensor.

2.2.4 Protokol Komunikasi [4]

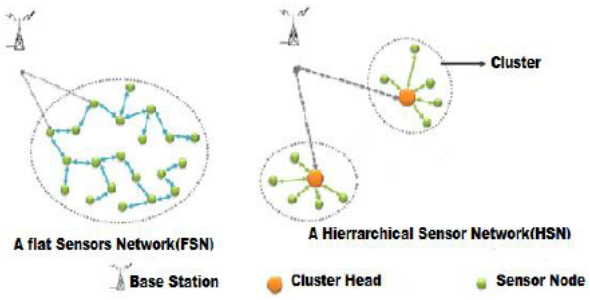
Terdapat 2 jenis protokol komunikasi, yaitu *Single-hop* dan *Multi-hop*. Perbedaan utama pada *single-hop* dan *multi-hop* adalah jumlah langkah sebuah paket butuhkan untuk mencapai tujuan akhir. Pada *single-hop*, ketika paket meninggalkan sumbernya atau node sensor asalnya hanya membutuhkan satu kali melangkah untuk mencapai tujuannya dalam hal ini *base-station*. *Single-hop* sangat cocok untuk komunikasi skala kecil karena jika jarak node dengan *base-station* terlalu jauh maka data yang diterima akan semakin lama dan bahkan bisa terjadi *loss*. Pada *multi-hop*, paket harus melalui node tetangganya (dua atau lebih) sebelum mencapai tujuan akhir (*base-station*). *Multi-hop* membutuhkan energi yang banyak/besar sehingga tidak baik digunakan untuk skala kecil, tetapi sangat berguna untuk skala besar dan bahkan energi yang terpakai lebih efisien. Contoh dari protokol komunikasi terlihat pada Gambar 2.3.



Gambar 2.3: *Single-hop* dan *multi-hop* pada *sensor network*.

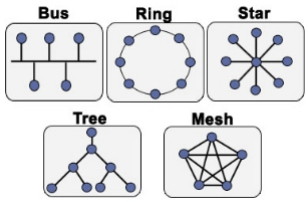
2.2.5 Arsitektur *Wireless Sensor Network* [5]

Terdapat 2 jenis arsitektur pada WSN, yaitu Flat dan Hirarkikal. Node-node pada arsitektur flat memiliki tugas atau peran yang sama untuk melakukan *sensing* dan mengirimkan data hingga tujuan akhirnya yang adalah *base-station*. Arsitektur Hirarkikal bertujuan untuk mencapai efisiensi energi, stabilitas, dan skalabilitas. Jaringan node disusun secara hierarki sehingga node sensor yang dibentuk menjadi *cluster head* dan node sensor lainnya akan menjadi *cluster member*. *Cluster head* berfungsi mengatur *cluster member* agar bekerja dengan baik.



Gambar 2.4: Arsitektur Flat dan Hirarkikal.

2.2.6 Topologi *Wireless Sensor Network* [1]



Gambar 2.5: Tipe-tipe topologi pada WSN.

Struktur jaringan pada kehidupan nyata dibuat dibuat dengan teori grafik matematika sehingga membentuk topologi-topologi yang berebeda. Berikut topologi-topologi yang dimaksud adalah:

1. Bus Topology

   Topologi ini menggunakan *medium* yang menjadi jalur untuk semua *devices* terhubung. Saat melakukan pengiriman data untuk ke sebuah *host*, pesan tersebut akan di *broadcast* ke seluruh jaringan yang artinya semua *devices* akan mendapatkan pesan tersebut. Hanya *host* yang seusai dengan tujuan pengirim akan menerima pesan tersebut. Keuntungan menggunakan topologi ini adalah mudah diimplementasikan dan dirawat pada jaringan kecil, jika jaringan sudah bertumbuh semakin besar maka topologi ini sudah tidak bisa dipakai karena mengakibatkan kerugian pada *bandwidth* yang akan dibagi pada semua *devices* yang terkoneksi dan akan terus dibagi semakin banyak *device* yang terkoneksi. Kelemahannya lainnya adalah isu keamanan pada pesan yang seharusnya dibaca oleh sebuah *host* dapat dibaca oleh *host* lainnya.

2. *Ring Topology*

   Pada topologi ini akan terbentuk sebuah lingkaran yang terdiri dari *device-device*, setiap *device* yang terhubung pasti memiliki 1 atau 2 tetangga di sebelahnya. Data dikirim melalui jaringan akan sesuai arah jarum jam atau sebaliknya. Dalam topologi ini tipe jaringan merupakan *self-managing* yang berarti tidak diperlukan *central controlling unit* untuk mengatur kegiatan mereka. Kerugian dari topologi ini adalah jika ada sebuah koneksi diantara 2 *hosts* mati/rusak maka akan mengganggu keseluruhan jaringan yang sudah terhubung dengan baik. Masalah lain yang terjadi jika sistem semakin besar(*host* bertambah) maka *host* harus dimasukkan dahulu dan mengganggu jaringan untuk beberapa waktu.

3. *Star Topology*

   Pada jaringan ini seluruh *hosts* akan terhubung pada *main device.Main device* dapat berupa pengontrol, router, atau switch. Dalam topologi ini jika memakai *central node* yang bersifat pasif seperti *hub*, setiap pesan akan dikirimkan ke seluruh *host* yang terhubung. Jika *central node* bersifat aktif, maka router atau switch dapat mengatur pesan sehingga *volume* pesan yang

lewat dapat diperkecil agar pesan menuju *host* yang sesuai dan meningkatkan keamanan karena pesan tidak sampai ke *host* yang bukan menjadi tujuan pesan. Dalam perbandingan dengan Ring Topology, jaringan ini tidak akan langsung mengalami gangguan untuk keseluruhan jaringan jika ada 1 *host* yang bermasalah, tetapi kelemahannya topologi ini bergantung pada *central node* karena jika bagian ini terjadi gangguan atau mati maka seluruh komunikasi akan berhenti seketika.

4. *Tree Topology*

   Topologi ini menggabungkan *bus* dan *star topology*. Jaringan *tree* ini dibangun dengan arsitektur hirarkikal. Pada *First level* akan ditetapkan sebuah *root node*. *Host* yang terhubung dengan *root node* akan membentuk *second level*. Setiap node akan terhubung dengan 1 node yang berasal dari level yang lebih tinggi, tetapi dapat terhubung dengan beberapa node yang berasal dari level lebih rendah. Keuntungan dari topologi ini adalah mudah di*maintenance* dan mudah menemukan kesalahan, jika ada masalah pada sebuah node tidak akan mempengaruhi keseluruhan jaringan. Kelemahannya adalah banyak membutuhkan *maintenance*.

5. *Mesh Topology* Bentuk topologi ini menyatukan semua topologi yang ada. Topologi tipe ini menggambarkan jaringan internet yang luas dengan ribuan *subnetwork* yang memiliki konfigurasi yang berbeda-beda. Keuntungan dari topologi ini dapat mengurus lalu lintas jaringan yang begitu banyak dan bahkan setiap node baru yang ditambahkan dapat dimasukkan tanpa mengganggu jaringan yang lain. Kelemahannya adalah karena begitu besar jaringannya sehingga membuat *maintenance* sangat sulit untuk mengelolanya.

2.2.7 Sistem Operasi *Wireless Sensor Network* [6]

Sistem operasi dalam *Wireless Sensor Network* terletak pada perangkat keras dan perangkat lunak. Tujuan dari sistem operasi sendiri adalah agar aplikasi dapat berinteraksi dengan perangkat keras.

OS/ Feature	Architecture	Programming model	Scheduling	Memory Management and Protection	Communication Protocol Support	Resource Sharing	Support for Real-time Applications
TinyOS	Monolithic	Primarily event Driven, support for TOS threads has been added	FIFO	Static Memory Management with memory protection	Active Message	Virtualization and Completion Events	No
Contiki	Modular	Protothreads and events	Events are fired as they occur. Interrupts execute w.r.t. priority	Dynamic memory management and linking. No process address space protection.	*u*IP and Rime	Serialized Access	No
MANTIS	Layered	Threads	Five priority classes and further priorities in each priority class.	Dynamic memory management supported but use is discouraged, no memory protection.	At Kernel Level COMM layer. Networking Layer is at user level. Application is free to use custom routing protocols.	Through Semaphores.	To some extent at process scheduling level (Implementation of priority scheduling within different processes types)
Nano-RK	Monolithic	Threads	Rate Monotonic and rate harmonized scheduling	Static Memory Management and No memory protection	Socket like abstraction for networking	Serialized access through mutexes and semaphores. Provide an implementation of Priority Ceiling Algorithm for priority inversion.	Yes
LiteOS	Modular	Threads and Events	Priority based Round Robin Scheduling	Dynamic memory management and it provides memory protection to processes.	File based communication	Through synchronization primitives	No

Gambar 2.6: Tabel Sistem Operasi pada *Wireless Sensor Network*

Pada Gambar 2.6, dapat dilihat jenis-jenis sistem operasi apa saja yang pada umumnya digunakan pada *Wireless Sensor Network* beserta dengan fitur-fitur yang disediakan oleh sistem operasi tersebut.

2.3 Monitoring Tool pada *Wireless Sensor Network* [7]

2.3.1 Monitoring Tool

WSN membutuhkan cara manajemen sensor-sensornya mulai dari memeriksa sisa energi baterai, jarak antar sensor, dsb sehingga membutuhkan tempat untuk memonitor semua hal tersebut yang disebut **Monitoring Tool**. *Monitoring Tool* memberikan fitur kepada pengguna untuk melakukan pengontrolan dasar dan diimplementasikan pada daerah penting yang sangat membutuhkan pengecekan performansi pada jaringan.

2.3.2 Kategori pada Monitoring Tool

Berdasarkan dari cara pendekatan untuk memonitor dan mengontrol pada WSN, *Monitoring Tool* dibagi menjadi 4 kategori, yaitu:

1. Passive Monitoring

   Sistem mengumpulkan informasi dan data dari jaringan yang nantinya akan di analisis lebih lanjut agar dapat mendapat kesimpulan apa yang harus dilakukan pada atau akan terjadi *event* apa pada masa yang akan datang.

2. Fault Detection Monitoring

   Sistem mengumpulkan informasi dari keadaan jaringan untuk mengidentifikasi di mana kesalahan terjadi.

3. Reactive Monitoring

   Sistem mengumpulkan informasi keadaan jaringan untuk diidentifikasi apakah terjadi suatu *event* pada lingkungan yang diperiksa dan secara adaptif melakukan konfigurasi ulang pada jaringan.

4. Proactive Monitoring

   Sistem secara aktif mengumpulkan dan menganalisis keadaan jaringan untuk mendeteksi *event* yang sudah berlalu dan memprediksi *event* yang akan datang agar dapat menjaga performansi pada jaringan.

2.3.3 Contoh Monitoring Tool yang Tersedia

Terdapat lima *Monitoring Tool* yang sudah ada sampai saat ini, yaitu:

1. Mote-view [3]

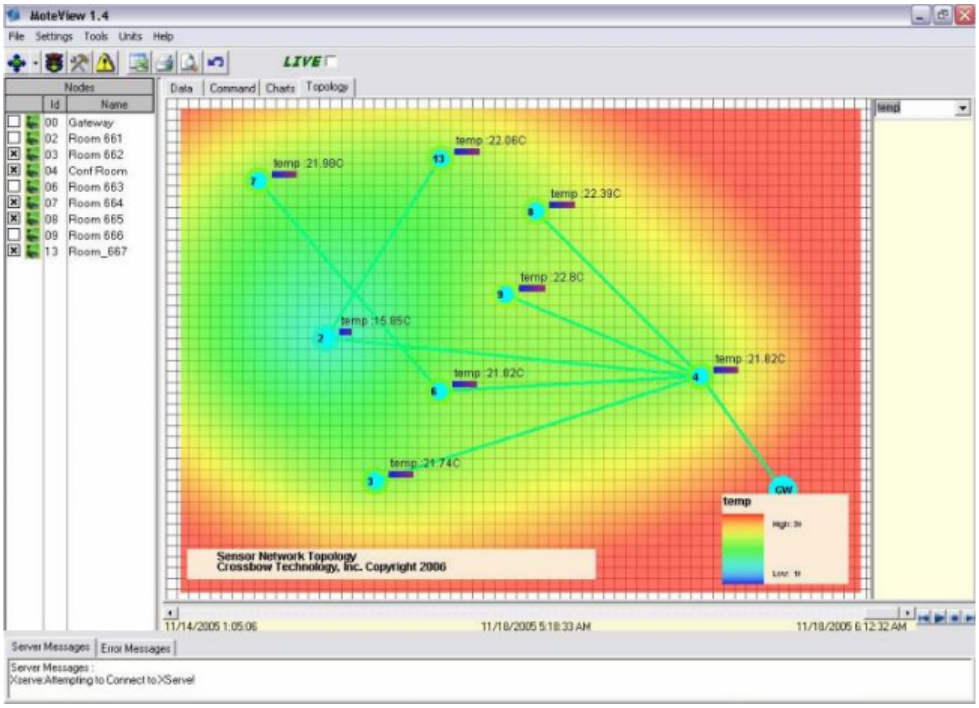
   Mote-view adalah framework perangkat lunak yang digunakan untuk visualisasi dan memonitor WSN. Menurut desainer Mote-view, WSN memiliki 3 tier, yaitu *Mote tier*, *Server tier*, dan *Client tier*. *Mote tier* menjalankan *embedded firmware* (TinyOS) dan driver hardware milik sensor, *server tier* menyediakan *logger* dan basisdata agar dapat mengambil data dari node sensor. Mote-view berada di *client tier* yang menyediakan GUI untuk visualisasi data sensor.

   Fungsi-fungsi yang ada pada Mote-view, antara lain:

   - Memvisualkan topologi, statistik jaringan dan sensor yang terkoneksi melakukan *login* atau mengirim data.

[3] https://eg.uc.pt/bitstream/10316/35740/1/Wireless%20Sensor%20Networks%20Monitoring%20Tool.pdf

- Warna node akan berubah ketika tidak mendapat dan mengirim data apapun pada waktu tertentu.
- Menampilkan *sensor-reading* secara *live*.
- Warna dapat diatur untuk menentukan suatu variabel misalkan suatu data sukses terkirim maka akan berubah warna.



Gambar 2.7: Tampilan utama Mote-view.

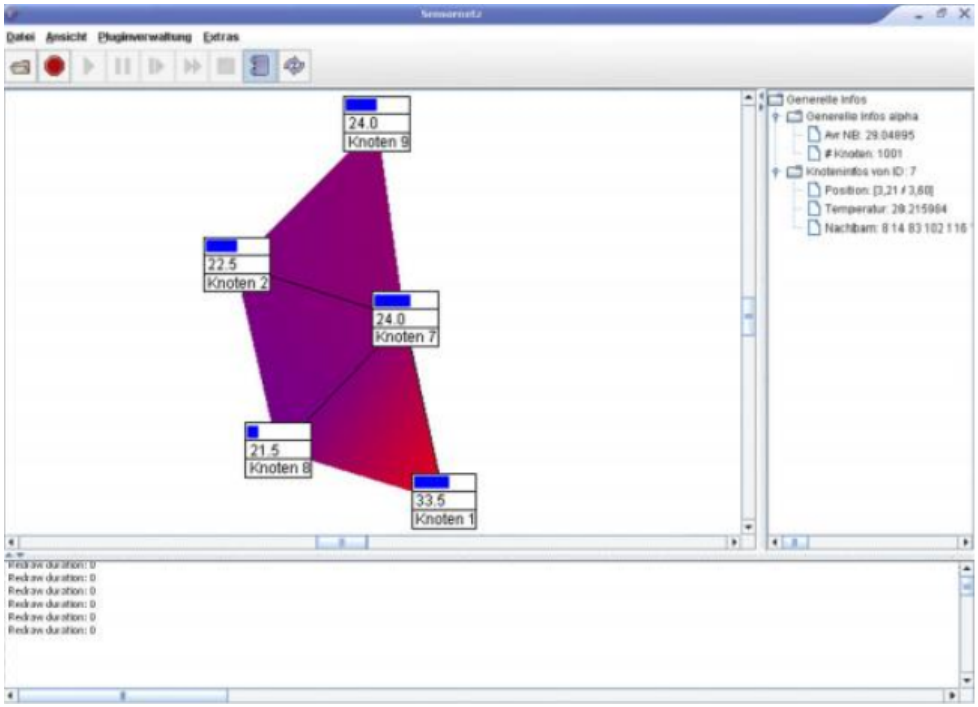
2. Spyglass [4]

   Spglass adalah framework modular untuk memonitor WSN yang menggunakan beberapa *plug-in* untuk menampilkan data seperti posisi node, sensor yang mengumpulkan data dan topologi jaringan. *Plug-in* tersebut dapat dikembangkan untuk mendukung tambahan fungsi lainnya.

   Fungsi-fungsi yang ada pada Spyglass, antara lain:

   - Menampilkan temperatur, relasi antar node, jaringan, dan dapat merekam aktivitas selama waktu yang ditentukan yang nantinya akan diputar kembali dalam kecepatan yang berbeda-beda.
   - Menggunakan *plug-in* yang ditaruh pada layer tersendiri melalui *drawing primitive* di Spyglass dapat menghindari konflik antar *plug-in* dan dapat dikembangkan sesuai keperluan.
   - Saat ini Spyglass dapat memakai beberapa *plug-in* yang dapat menampilkan node dan jaringan: *temperature map plug-in, display node plug-in, the battery plug-in, topology plug-in.*

[4] https://eg.uc.pt/bitstream/10316/35740/1/Wireless%20Sensor%20Networks%20Monitoring%20Tool.pdf



Gambar 2.8: Tampilan UI Spyglass.

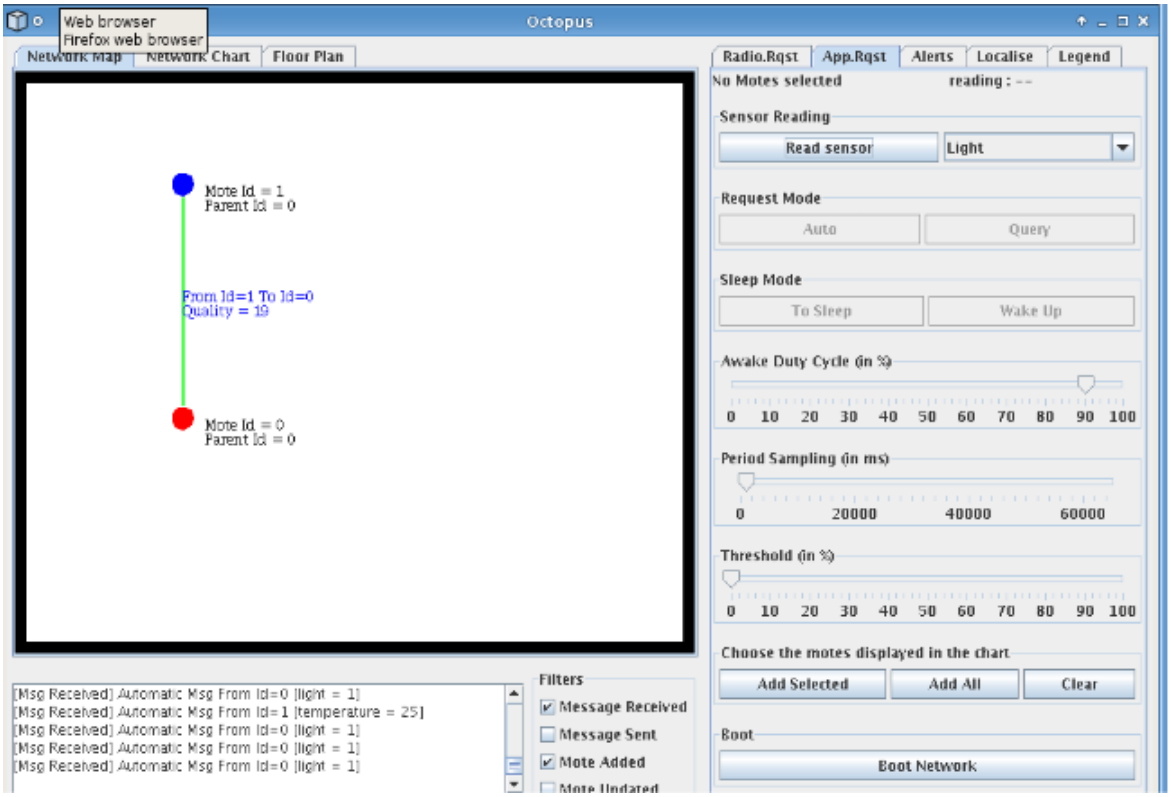
3. Octopus [5] [11]

   Octopus adalah sebuah Monitoring Tool WSN yang bebas memilih *routing protocol* untuk digunakan pada WSN. Octopus menyediakan informasi secara *live* tentang topologi jaringan dan data sensor yang dibaca. Octopus juga memperbolehkan konfigurasi ulang pada jaringan dan aplikasinya dengan perintah pendek kepada node sensor melalui udara.

   Fungsi-fungsi yang ada pada Octopus, antara lain:

   - GUI untuk visualisasi topologi jaringan sensor secara *live*.
   - Mendukung penggunaan *time-driven*, *query-driven*, dan *event base driven*.
   - Memperbolehkan pengguna untuk memberi perintah terhadap setiap node secara bersamaan maupun satu per satu.
   - Membuka developer agar dapat mengatur atau menambah fitur pada GUI melalui modular API.
   - Menyimpan data yang sudah dikumpulkan pada file .scv untuk di analisis lebih lanjut.
   - Memberi peringatan ketika sensor membaca data melebihi kapasitas yang pengguna sudah tentukan.

[5] https://eg.uc.pt/bitstream/10316/35740/1/Wireless%20Sensor%20Networks%20Monitoring%20Tool.pdf



Gambar 2.9: Tampilan UI Octopus.

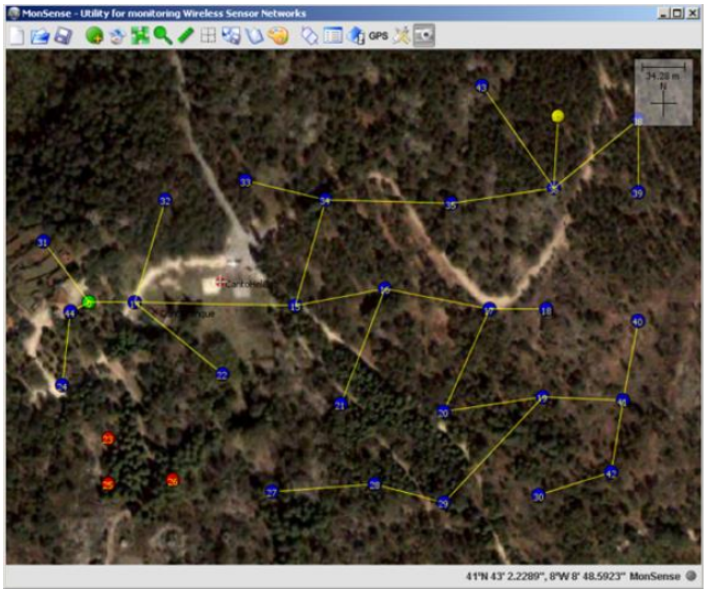
4. MonSense [6]

   MonSense adalah *modular WSN monitoring framework* untuk memonitor dan mengatur WSN. MonSense mendukung berbagai koneksi WSN dan dapat berinteraksi lebih dari satu *gateway* yang meneruskan informasi dari jaringan yang terhubung.

   Fungsi-fungsi yang ada pada MonSense, antara lain:

   - MonSense menampilkan peta dan menggunakan GPS untuk membantu penyebaran sensor agar berada di tempat yang sesuai dan sama persis.
   - koneksi atau jalur antar node dilambangkan dengan garis berwarna untuk parameter koneksi yang berbeda-beda seperti node sedang aktif, node terhubung dengan *base-station*, node sedang mengirim paket, dll.

[6] https://eg.uc.pt/bitstream/10316/35740/1/Wireless%20Sensor%20Networks%20Monitoring%20Tool.pdf



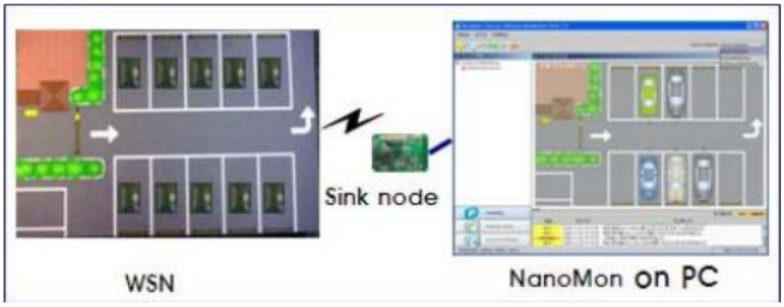
Gambar 2.10: Tampilan UI MonSense.

5. NanoMon [7]

   NanoMon adalah perangkat lunak fleksibel dan modular untuk memonitor WSN, ia(NanoMon) menawarkan arsitektur yang fleksibel dan mendukung berbagai macam permintaan pengguna untuk aplikasi jaringan sensor dengan cara yang adaptif.

   Fungsi-fungsi yang ada pada NanoMon, antara lain:

   - *Topology plug-in* untuk memvisualkan node sensor dan koneksi antara mereka.
   - *Sensor data plug-in* menampilkan nilai *sensing* dari node sensor, *data reception time*, dan juga mencari data sensor yang sesuai dengan kondisi input.
   - *Chart plug-in* memvisualkan data sensor menggunakan berbagai macam grafik.
   - *Sensor list plug-in* memvisualkan node sensor secara live dan informasi detil sensor seperti sisa baterai, letak penempatan sensor, tanda pengenalnya(ID) dan nilai *sensing* terakhir yang didapatkan.



Gambar 2.11: Tampilan testbed(kiri) dan UI NanoMon(kanan).

2.3.4 Karakteristik Monitoring Tool

Pada tabel 2.2 diperlihatkan perbandingan karakteristik dari masing-masing Monitoring Tool yang sudah dibahas pada subbab sebelumnya.

[7] https://eg.uc.pt/bitstream/10316/35740/1/Wireless%20Sensor%20Networks%20Monitoring%20Tool.pdf

Monitoring tools	**Data viusalization and controlling**	**Live visualization and data logging**	**Data retrieval method**	**Type of data displayed**	**Generate alerts**	**Source Code availability and extension**	**Supported Mote Platform**	**Operating system**
Mote-view	Data visualization and controlling	Live and data logging	Event and query driven	Nodes and network	yes	no	Mica series	TinyOS
Spyglass	Data visualization	Live	Time-driven	Nodes and network	no	no	Embedded Sensor Board ESB 430/2	C language
Octopus	Data visualization and controlling	Live and data logging	Time-driven and query-driven	Nodes and network	yes	yes and support extension	TelosB, Mica Family and Tyndall 25	TinyOS
MonSense	Data visualization and controlling	Live and data logging	Time-driven	network	yes	no	TelosB	TinyOS
NanoMon	Data visualization	Live and data logging	Time-driven and query-driven	Nodes and network	no	no	Nano-24 ETRI-SSN MicaZ	NanoQplus [10]

Tabel 2.2: Tabel perbandingan karakteristik Monitoring Tools.

Dari tabel perbandingan pada tabel 2.2 memiliki delapan karakteristik:

1. Data visualization and Controlling

   Fungsinya tergantung jika *tool* hanya digunakan untuk mengumpulkan data pada node yang ada di jaringan, atau *tool* juga dapat mengontrol jaringan. Pada kasus pertam, *tool* digunakan untuk mengumpulkan data saja dan *data flow* hanya 1 arah dari WSN menuju aplikasi *monitoring* yang berjalan di komputer. Pada kasus lain selain memvisualkan data, *tool* juga menyediakan mekanisme kepada pengguna untuk mengontrol node di jaringan.

2. Live visualization and Logging

   Terlepas dari visualisasi *event* yang terjadi pada *real time*, fungsi *logging* dibutuhkan untuk menyimpan *event* yang terjadi untuk nantinya dianalisis dan konsultasi *event* yang terjadi pada masa lalu.

3. Data Retrieval Method

   Fungsinya menyangkut cara data yang telah dikumpulkan ditransfer/dikirim ke *monitoring tool*. *Time-driven* mengirim data secara periodik menuju *monitoring tool*. *Query-driven* mengijinkan pengguna untuk melakukan *query* data atau informasi dari jaringan kapanpun mereka inginkan.

4. Type of Data displayed

   Karakteristik ini digunakan untuk membedakan *monitoring tools* dari informasi yang disediakan oleh *tool*. Kebanyakan *monitoring tools* menyediakan data untuk node dan jaringan kecuali MonSense yang hanya berfokus pada jaringan saja.

5. Alert

   Karakteristik ini menunjukkan kemampuan *monitoring tool* untuk memberi notifikasi kepada pengguna ketika ada *event* dari lingkungan yang diawasi terjadi.

6. Source Code Availability and Support of Extension

   Walaupun kebanyakan *tools* mengklaim bahwa *support extensions*, menambahkan fungsi lagi ke *tool*, tidak semuanya mempunyai *source code* yang tersedia.

7. Supported Mote Platform

   Setiap *tool* memiliki beberapa mote(node) *platforms* yang tersedia. Semakin banyak *platform* yang bisa digunakan semakin baik.

8. Operating System

   Aplikasi yang bekerja pada node diprogram pada *sensor operating system*. Beberapa sistem operasi menyediakan *library* untuk digunakan oleh program lain untuk berkomunikasi dengan aplikasi yang berjalan di node sensor.

2.3.5 Monitoring Tool Requirements

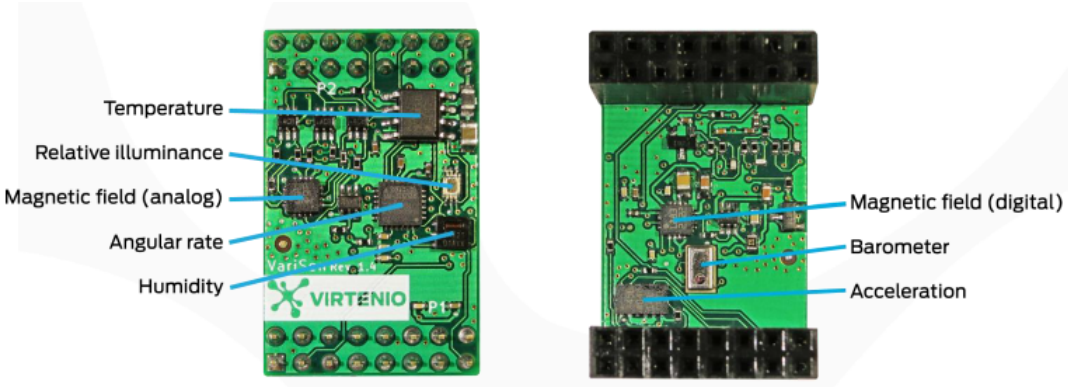
WSN banyak digunakan pada berbagai macam kondisi lingkungan seperti memonitor kesehatan pasien di rumah sakit, bidang militer dan industri tanaman, dsb. Tempat berbeda membutuhkan WSN yang memiliki kebutuhan yang berbeda juga tergantung daerah WSN tersebut digunakan. Pada dasarnya secara umum *monitoring tool* harusnya memiliki fungsi sebagai berikut:

- Menampilkan status node di jaringan.
- Menampilkan data yang sudah dikumpulkan oleh node.

- Menampilkan *logical network topology*.
- Mengijinkan pengguna untuk melakukan *basic control* node di jaringan.
- Memberi *alert* berdasarkan nilai yang diatur oleh pengguna.
- Dapat mengakses riwayat apa saja pada setiap node.

2.4 Preon32 [8]

Preon32 adalah salah satu jenis node sensor yang menggunakan PreonVM sebagai sistem operasinya. Preon32 versi umum memiliki 5 jenis sensor pada sebuah *board*. Sensor yang ada pada Preon32 ini antara lain sensor suhu (*temperature sensor*), sensor cahaya (*light intensity sensor*), sensor kelembaban udara (*relative humidity sensor*), sensor tekanan udara (*air pressure sensor*), dan sensor getaran (*acceleration sensor*). Pada versi tambahan Preon32 dapat dilengkapi dengan sensor untuk mendeteksi medan magnet, dan *gyroscope*. Semua jenis sensor tersebut dapat diatur melalui PreonVM dan pemrograman dapat dilakukan dengan Bahasa Pemrograman Java. Gambar 2.12 adalah *board* Preon32 beserta letak sensor-sensor tersebut.



Gambar 2.12: Preon32 Board

Karakteristik dari sensor *Preon32* ini adalah :

- 32-bit *microcontroller*
- Ram dengan kapasitas 64 Kbyte
- *radio tranceiver* dengan standar IEEE 802.15.4

2.4.1 Preon32 Shuttle [9]

Gambar 2.13: Sensor Preon 32 Shuttle

Dapat dilihat pada gambar 2.13 yang merupakan bentuk dari sensor *Preon32 Shuttle*. Sensor *Preon32 Shuttle* merupakan *Preon32* yang dilengkapi dengan beberapa fitur tambahan. untuk bentuk sensor *Preon32 Shuttle* dan sensor *Preon32* tidak terlalu berbeda, sensor *Preon32 Shuttle* merupakan sensor *Preon32* yang telah diperbaharui atau sudah ditambahkan modul-modul tambahan. Sensor Preon32 Shuttle ini memiliki energi yang terbatas, sehingga operasi yang dilakukan atau dimasukkan ke dalam sensor harus efisien agar energi yang dimiliki oleh sensor tidak mudah habis.

Perangkat keras yang akan digunakan untuk menjalankan perangkat lunak yang akan dibuat adalah sensor *Preon32 Shuttle*.Beberapa modul tambahan yang dimiliki oleh sensor *Preon32 Shuttle* adalah :

- Memiliki koneksi dengan USB.
- Dapat menerima sumber daya atau energi dari baterai.
- Memiliki tombol yang dapat dijadikan sebagai interaksi dengan pengguna.

2.4.2 PreonVM [10]

PreonVM adalah virtual machine (VM) yang dibuat oleh VIRTENIO untuk sistem komputer yang dirancang khusus (*embedded system*) dengan sumber daya yang terbatas. PreonVM dibuat sangat optimal dengan tidak dibutuhkannya sistem operasi tambahan dan berjalan langsung pada *microprocessor*. Dengan PreonVM ini *developer* dapat membuat aplikasi dengan mudah pada Bahasa Pemrograman Java yang mengumpulkan data dari sensor dan mengontrol aktuator. API pada PreonVM mendukung antarmuka radio sesuai dengan IEEE 802.15.4.

PreonVM memiliki karakteristik sebagai berikut:

- Aplikasi dibangun dengan Bahasa Pemrograman Java
- Mendukung semua tipe data pada Java seperti char, byte, int, long, float atau double
- *Garbage collection* dengan *memory defragmentation*
- Mendukung *exception handling*, *stack* dan array multidimensi
- Terdapat *system properties* untuk mengatur aplikasi
- Tidak membutuhkan sistem operasi tambahan
- Mendukung *thread* termasuk synchronized, Object.wait, Object.notify, Object.notifyAll, Thread.sleep, atau Thread.interrupt

BAB 3

ANALISIS

Pada bab ini dijelaskan mengenai deskripsi perangkat lunak, analisis kebutuhan perangkat lunak, analisis cara kerja sistem, analisis cara *base-station* menerima pesan dari banyak node sensor, dan analisis perbandingan perangkat lunak yang dibangun dengan yang sudah ada.

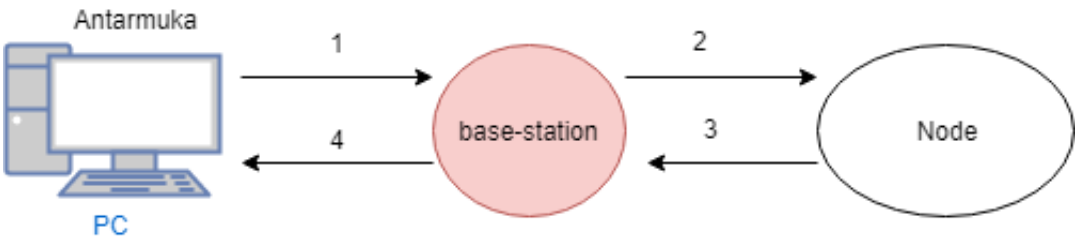
3.1 Deskripsi Perangkat Lunak

Perangkat lunak yang dibangun pada penelitian ini bertujuan untuk memvisualkan node-node sensor yang dipakai untuk memonitor aktivitas di suatu lingkungan dan terhubung ke sesama node. Perangkat lunak ini dapat menampilkan node-node sensor dengan 2 cara, yaitu node sensor akan mengirimkan data diri node sensor tersebut ke *base-station* lalu data yang diterima akan disimpan dan node akan ditampilkan pada perangkat lunak dengan warna yang sudah ditentukan untuk node sensor atau bisa juga dengan *base-station* mengirimkan pesan kemanapun dan jika sensor menerima pesan dan mengembalikan pesan beserta data node sensor tersebut ke *base-station* maka node akan ditampilkan pada perangkat lunak. Data diri node sensor berupa nama dan hasil *sensing*. Perangkat lunak juga menyediakan sebuah *log* yang berisi waktu, nama, dan data hasil *sensing* dari suatu node sensor.

Akan dibangun 3 buah perangkat lunak yang akan digunakan untuk melakukan *sensing* dan ditampilkan ke pengguna. Perangkat lunak pertama merupakan perangkat lunak yang akan diunggah pada *base-station*, perangkat lunak kedua akan diunggah pada node sensor, dan perangkat lunak ketiga akan digunakan sebagai media masukan dan keluaran yang dapat diterima oleh pengguna dari *base-station* berupa antarmuka pada PC. Ketiga perangkat lunak tersebut akan saling berkomunikasi agar dapat bekerja dengan baik. Perangkat lunak pada pc hanya akan memberi masukan dan menerima keluaran dari *base-station*, komunkasi terjadi dengan menggunakan media kabel. Perangkat lunak yang akan diunggah pada node sensor dapat berkomunikasi dengan *base-station* dengan menggunakan sinyal *wireless*.

Dalam pembuatan perangkat lunak ini ada beberapa faktor yang harus diperhatikan. Faktor-faktor tersebut antara lain topologi jaringan dan algoritma yang akan digunakan dalam pembuatan perangkat lunak. Untuk memperjelas gambaran perangkat lunak yang akan dibangun, dapat dilihat pada gambar 3.1 yang menunjukkan bagaimana alur komunikasi yang terjadi pada perangkat lunak yang dibangun. PC tidak dapat secara langsung berhubungan dengan sensor node begitu pula dengan node sensor tidak dapat berhubungan langsung dengan PC. Segala komunikasi yang terjadi antara PC dengan node sensor harus melalui *base-station*. Berikut penjelasan singkat keempat nomor yang terdapat gambar 3.1:

1. Perangkat lunak pada PC akan mengirim perintah pada *base-station* untuk meminta node sensor melakukan *sensing*.
2. *Base-station* mengirim pesan ke node sensor untuk melakukan *sensing*.
3. Node sensor mengirim hasil *sensing* ke *base-station*.
4. *Base-station* mengembalikan hasil *sensing* ke antarmuka untuk ditampilkan kepada pengguna.



Gambar 3.1: Arsitektur Perangkat Lunak

3.2 Analisis Perancangan Perangkat Lunak

3.2.1 Analisis Kebutuhan Perangkat Lunak

Berdasarkan latar belakang, rumusan masalah dan tujuan dapat didefinisikan bahwa kebutuhan aplikasi pemantauan WSN mencakup:

1. Identitas sensor
   Aplikasi mampu menampilkan nama disetiap bawah gambar node sensor dan pada tabel hasil *sensing* terakhir.

2. Gambar node
   Setiap sensor dan *base-station* akan diberi gambar agar saat ditampilkan pada perangkat lunak akan lebih jelas dan dapat dibedakan sensor dengan *base-station*.

3. Pemeriksaan keadaan sensor
   Aplikasi mampu melakukan pemanggilan kepada sensor, jika sensor tidak memberi respon maka akan dianggap mati pada perangkat lunak dan jika sensor memberi respon maka tetap akan di tampilkan pada perangkat lunak.

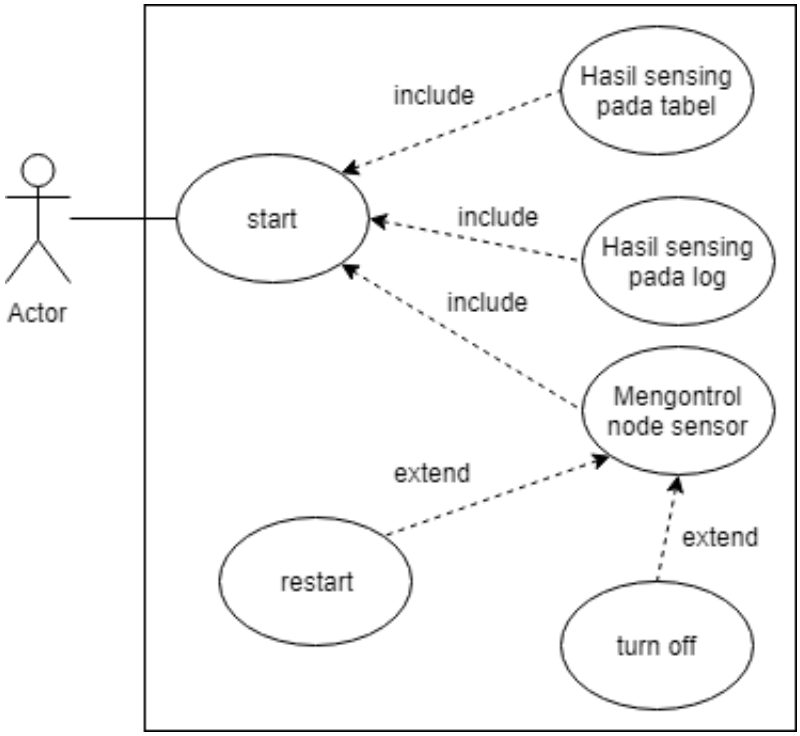
4. *Log*
   Aplikasi mampu menampilkan hasil data *sensing* untuk setiap data yang masuk lalu menampilkannya pada *log* dan juga menyimpannya pada *file text* agar dapat diperiksa bagi pengguna.

5. *Start, restart, dan turn off*
   Aplikasi memberikan fitur *start* untuk memulai perangkat lunak bekerja, fitur *restart* dan *turn off* untuk mematikan atau menyalakan kembali node sensor yang dipilih.

6. Tabel hasil *sensing* terakhir
   Aplikasi mampu menampilkan tabel yang memiliki nama-nama node sensor, keadaan mati atau nyala, dan hasil *sensing*.

3.2.2 Use Case Diagram

Diagram use case menggambarkan interaksi antara pengguna dengan perangkat lunak yang dibangun. Penggambaran ini diharapkan berguna untuk membantu memahami kebutuhan fungsional dari perangkat lunak.



Gambar 3.2: *Use case diagram.*

3.2.3 Use Case Skenario

Setiap fungsi pada diagram *use case* dilengkapi dengan skenario untuk memodelkan interaksi antara pengguna dan sistem. Berikut skenario setiap fungsi.

Tabel 3.1: Tabel skenario Perangkat lunak pertama kali dijalankan.

Nama	Perangkat lunak di-*start*
Deskripsi	Pengguna menekan tombol *start* dan perangkat lunak mulai bekerja.
Aktor	Pengguna
Pre-kondisi	Aplikasi sudah dijalankan.
Alur Skenario Utama	1. Sistem memuat aplikasi. 2. Pengguna meng-*click* tombol *start*. 3. Sistem bekerja. 4. Sistem mengirim pesan ke node sensor untuk melakukan *sensing*. 5. Node sensor mengembalikan nilai hasil *sensing*. 6. Sistem menampilkan nilai hasil *sensing*.

Tabel 3.2: Tabel skenario node sensor di-*restart*.

Nama	Node sensor di-*restart*
Deskripsi	Pengguna memilih node sensor yang ingin di-*restart* dan menekan tombol *restart*.
Aktor	Pengguna
Pre-kondisi	Tombol *start* sudah ditekan dan sistem sudah bekerja
Alur Skenario Utama	1. Pengguna memilih node sensor yang diinginkan dan menekan tombol *restart*. 2. Sistem bekerja. 3. Sistem mengirim pesan ke node sensor untuk melakukan *restart*. 4. Node sensor mengembalikan pesan sedang *restart* dan melakukan *restart*. 5. Sistem menampilkan pesan node tersebut sedang *restart*.

Tabel 3.3: Tabel skenario Node sensor di-*turn off*.

Nama	Node sensor di-*turn off*
Deskripsi	Pengguna memilih node sensor yang ingin di-*turn off* dan menekan tombol *turn off*.
Aktor	Pengguna
Pre-kondisi	Tombol *start* sudah ditekan dan sistem sudah bekerja
Alur Skenario Utama	1. Pengguna memilih node sensor yang diinginkan dan menekan tombol *turn off*. 2. Sistem bekerja. 3. Sistem mengirim pesan ke node sensor untuk melakukan *turn off*. 4. Node sensor mengembalikan pesan sedang *turn off* dan melakukan *turn off*. 5. Sistem menampilkan pesan node tersebut sedang *turn off*.

Tabel 3.4: Tabel skenario Melihat hasil *sensing* pada tabel.

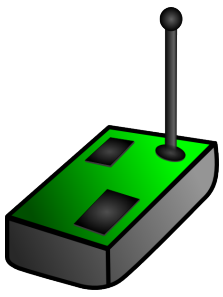
Nama	Melihat hasil *sensing* pada tabel
Deskripsi	Pengguna melihat hasil *sensing* terakhir setiap node sensor yang menyala pada tabel.
Aktor	Pengguna
Pre-kondisi	Sistem mengirim pesan *sensing* ke node sensor dan melakukan *sensing*.
Alur Skenario Utama	1. Sistem bekerja. 2. Sistem mengambil hasil *sensing* dan menampilkannya di tabel sesuai dengan baris node sensornya. 3. Pengguna melihat nilai terakhir setiap node sensor yang menyala pada tabel.

Tabel 3.5: Tabel skenario Melihat hasil *sensing* pada *log*.

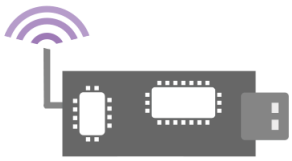
Nama	Melihat hasil *sensing* pada *log*
Deskripsi	Pengguna melihat hasil *sensing* pada log yang selalu menampilkan data baru yang masuk beserta waktunya.
Aktor	Pengguna
Pre-kondisi	Sistem mengirim pesan *sensing* ke node sensor dan melakukan *sensing*.
Alur Skenario Utama	1. Sistem bekerja. 2. Sistem langsung menampilkan hasil *sensing* pada *log* ketika ada data baru yang masuk. 3. Pengguna melihat nilai yang masuk di *log* beserta nama dan waktunya.

3.2.4 Analisis Keadaan Node sensor

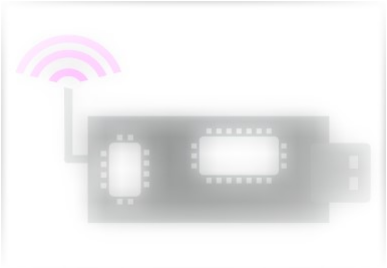
Pada perangkat lunak ini ada 3 jenis keadaan node sensor yang ditampilkan pada antarmuka, yaitu node sensor yang menyala (3.3), node sensor yang mati(3.4), dan node sensor yang tidak pernah ada/ tidak pernah terhubung(3.5).



Gambar 3.3: Visual Node Sensor ketika Menyala.



Gambar 3.4: Visual Node Sensor ketika Mati.

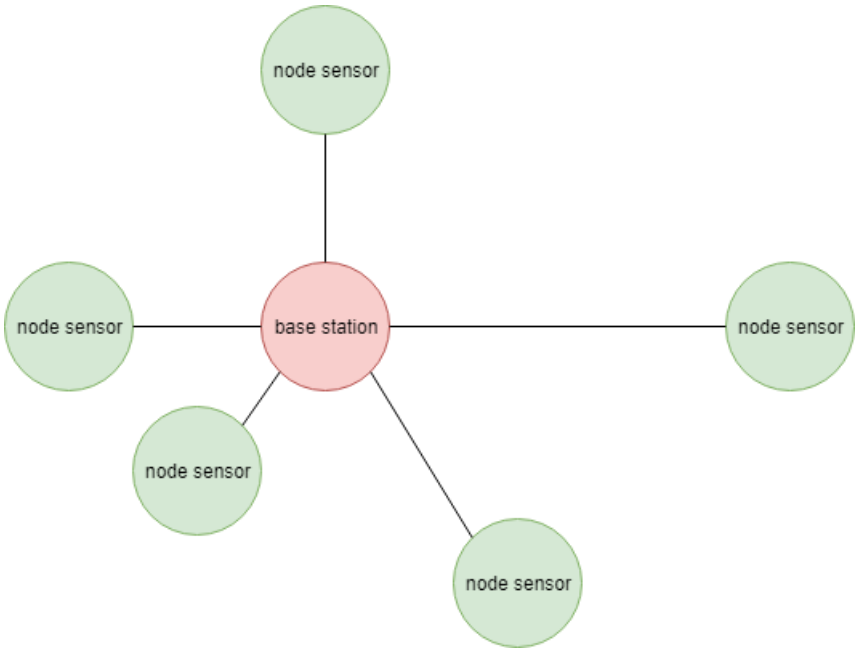


Gambar 3.5: Visual Node Sensor ketika tidak pernah ada atau terhubung.

Node sensor yang belum terhubung ketika perangkat lunak bekerja akan digambarkan seperti gambar 3.5. Sementara seperti gambar 3.4 untuk node sensor yang mati, tetapi sudah pernah terhubung dan melakukan *sensing*. Node tersebut mati mungkin karena di-*turn off*, *restart*, atau faktor lainnya sehingga node tersebut tidak merespon ke *base-station* dan diperkirakan bahwa node tersebut mati.

3.2.5 Analisis Topologi yang digunakan

Topologi yang digunakan adalah topologi Star dengan menggunakan arsitektur flat. Pada arsitektur flat yang memakai protokol komunikasi *single-hop* tidak terdapat hierarki dan *cluster head*. Berikut contoh topologi yang digunakan pada gambar 3.6

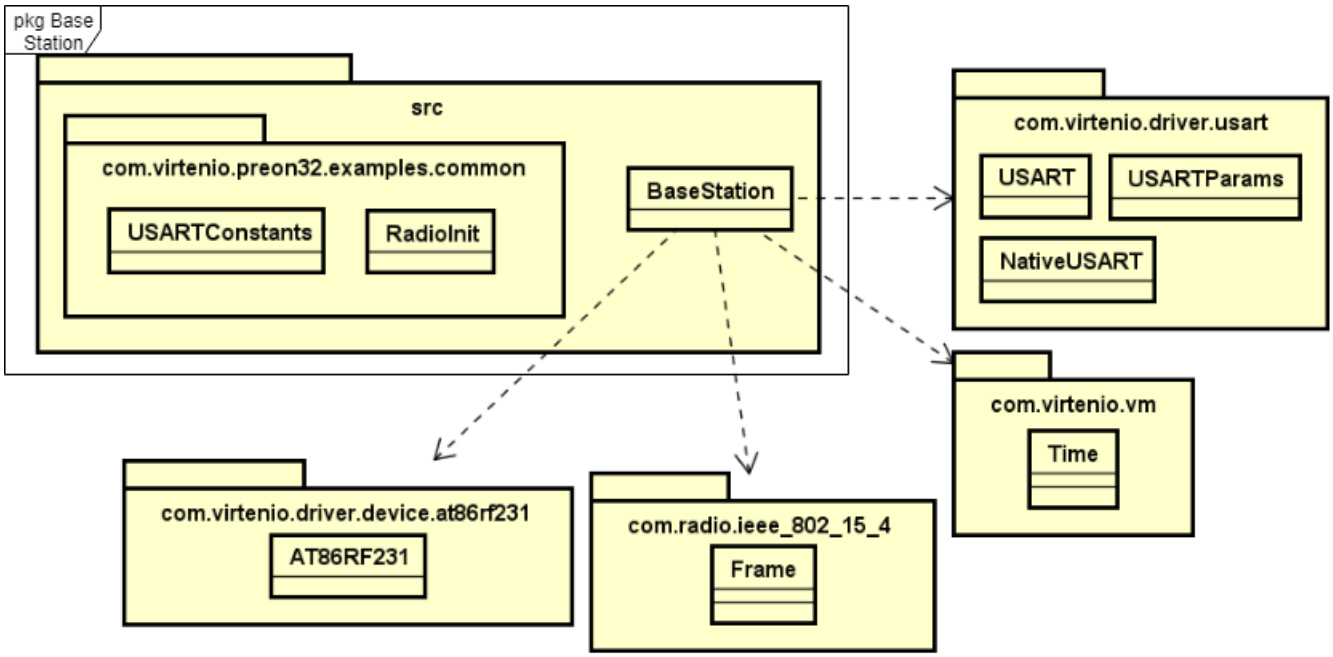


Gambar 3.6: Contoh Topologi yang digunakan.

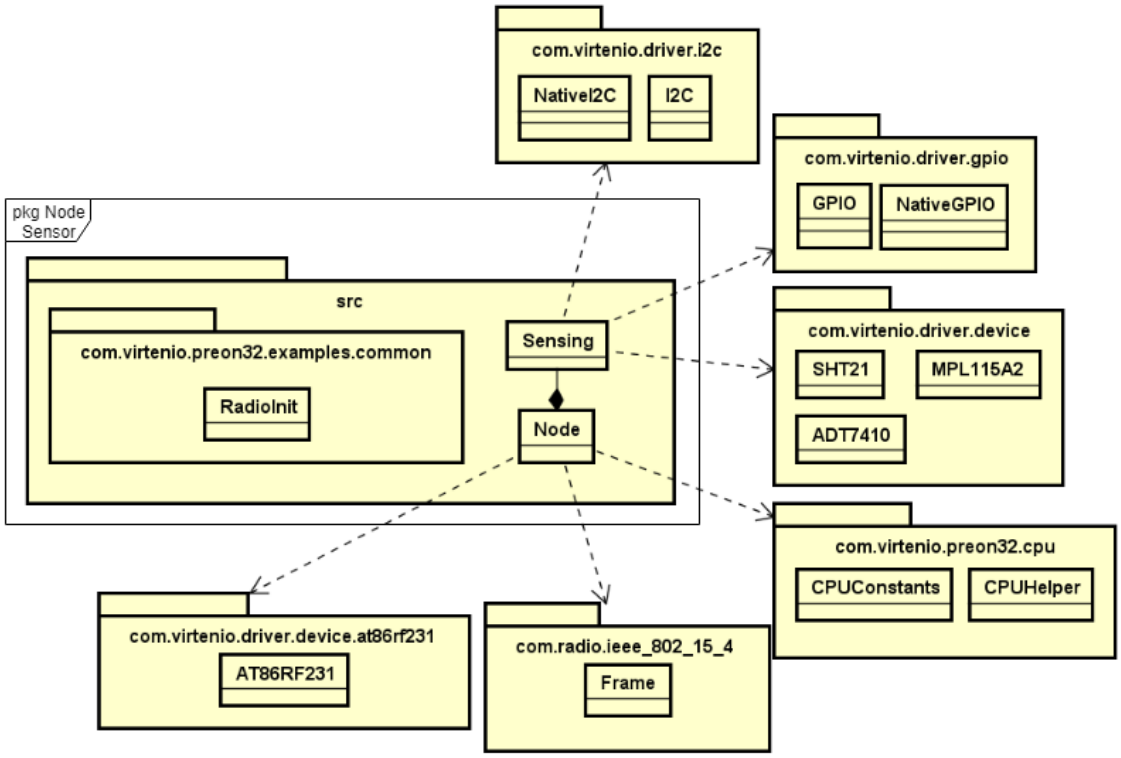
Pada gambar 3.6, Setiap node sensor (ditandai dengan warna hijau) yang telah melakukan *sensing* akan mengirimkan data langsung menuju *base-station* (ditandai dengan warna merah). Tujuan penggunaan komunikasi dan topologi tersebut adalah mudahnya mengkonfigurasi hubungan antara *base-station* dengan node sensor dibandingkan dengan topologi yang lain dan masih dapat menampilkannya pada perangkat lunak sehingga dapat dipantau.

3.2.6 Kelas Diagram Sederhana

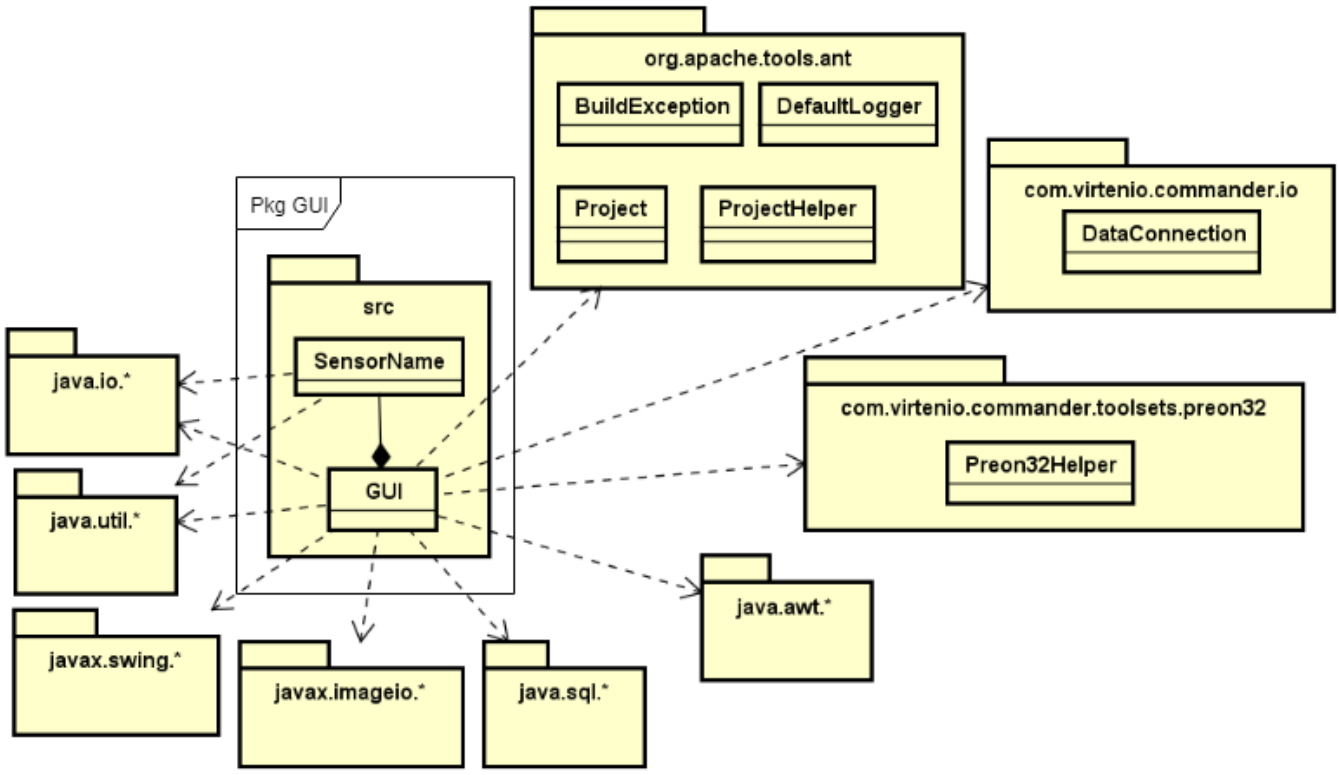
Pada subbab ini dibuat suatu diagram kelas sederhana untuk menjelaskan kelas-kelas yang dibutuhkan dalam membangun perangkat lunak pemantauan WSN. Aplikasi dibangun dengan Eclipse IDE dan menggunakan menggunakan Sandbox yang sudah disediakan oleh perusahaan Virtenio.



Gambar 3.7: Diagram Kelas Sederhana untuk *base-station*.



Gambar 3.8: Diagram Kelas Sederhana untuk node sensor.



Gambar 3.9: Diagram Kelas Sederhana untuk GUI.

Berikut adalah penjelasan dari kelas-kelas pada diagram.

- Kelas USARTConstant
  Kelas (disediakan oleh perusahaan virtenio) yang digunakan untuk membuat koneksi antara *base station* dengan program pada komputer pengguna.

- Kelas RadioInit
  Kelas ini digunakan untuk melakukan inisiasi koneksi radio antar node sensor .

- Kelas BaseStation
  Kelas ini digunakan oleh *base-station* untuk mengirim pesan untuk *sensing* kepada node sensor dan menerima datanya lalu dikirim ke kelas GUI.

- Kelas Node
  Kelas ini digunakan oleh node sensor untuk mengirim pesan atau hasil *sensing* kepada *base-station*.

- Kelas Sensing
  kelas ini melakukan *sense* yang mengeluarkan hasil seperti suhu, kelembaban, dan tekanan untuk digunakan oleh kelas Node.

- Kelas GUI
  Kelas ini sebagai *interface* untuk pengguna gunakan.

- Kelas SensorName
  Kelas ini membaca *file* nama-nama sensor yang akan dipakai kelas GUI sebagai penentu jumlah dan penulisan nama-namanya.

3.2.7 Format Pesan Pengiriman

Pengiriman pesan dari *base-station* ke node sensor dan sebaliknya pasti memiliki format pesan agar dapat diproses nantinya. Pengiriman pesan dari *base-station* ke node sensor ada 2 dari 3 jenis pesan dengan masing-masing formatnya sendiri, yaitu:

1. ***Sense***
   Pesan ini hanya berupa 1 kata, yaitu *sense* sehingga tidak perlu format khusus.

2. ***Turn off***
   Pesan untuk melakukan *turn off* memiliki format nama node sensor yang ingin dimatikan dan pesan aksi yang harus dilakukan.

3. ***Restart***
   Pesan untuk melakukan *restart* memiliki format nama node sensor yang ingin dimatikan dan pesan aksi yang harus dilakukan.

Pada pengiriman pesan dari node sensor ke *base-station* memiliki 2 format yang pertama untuk mengembalikan hasil *sensing* dan yang kedua adalah mengembalikan pesan bahwa node sedang *restart* atau *turn off*.

Pengiriman pesan dari *base-station* ke perangkat lunak hanya sebuah format, yaitu waktu terima pesan, nama node sensor pengirim pesan, dan isi pesan yang dikirim oleh node sensor.

3.3 Analisis Cara Kerja Sistem

Perangkat lunak yang dibangun akan memakai aristektur *flat* dan protokol komunikasi *single-hop* sehingga dapat dibuat algoritma yang dapat menggapai tujuan dari skripsi ini.

Alur kerja sistem pertama yang didesain dimulai dari perangkat lunak akan memberi kode perintah kepada *base-station* untuk *sensing* lalu *base-station* mengubah perintah tersebut menjadi

pesan yang akan dikirimkan kepada node sensor secara *broadcast*(mengirim ke semua node sensor yang dapat dijangkau). Node sensor akan menerima pesan tersebut dan melakukan *sensing* lalu node sensor akan mengembalikan pesan atau hasil *sensing* ke *base-station*. Hasil yang dikirim node sensor akan diterima oleh *base-station* dan diproses untuk dikirimkan ke perangkat lunak. pada perangkat lunak akan menampilkan node-node yang menyala beserta hasilnya sesuai pesan yang diberikan oleh *base-station*.

Alur kerja sistem kedua yang didesain dimulai dari pengguna memilih nama node sensor yang ingin dimatikan, setelah itu perangkat lunak akan memberi kode perintah kepada *base-station* untuk *turn off* lalu *base-station* mengubah perintah tersebut menjadi pesan yang akan dikirimkan kepada node sensor secara *broadcast*(mengirim ke semua node sensor yang dapat dijangkau). Node sensor akan menerima pesan yang dikirimkan dan melakukan pengecekan terlebih dahulu. Jika nama node yang terdapat pada pesan yang diterima sesuai dengan node tersebut maka node sensor akan mengembalikan pesan bahwa node akan mati dan setelah itu node tersebut akan berhenti bekerja/ *turn off*.

Alur kerja sistem ketiga yang didesain dimulai dari pengguna memilih nama node sensor yang ingin di-*restart*, setelah itu perangkat lunak akan memberi kode perintah kepada *base-station* untuk *restart* lalu *base-station* mengubah perintah tersebut menjadi pesan yang akan dikirimkan kepada node sensor secara *broadcast*(mengirim ke semua node sensor yang dapat dijangkau). Node sensor akan menerima pesan yang dikirimkan dan melakukan pengecekan terlebih dahulu. Jika nama node yang terdapat pada pesan yang diterima sesuai dengan node tersebut maka node sensor akan mengembalikan pesan bahwa node akan *restart* dan setelah itu node tersebut akan berhenti bekerja untuk beberapa saat kemudian menyala kembali.

Pada alur kerja sistem kedua dan ketiga jika nama node yang terdapat pada pesan (*restart* atau *turn off*) tidak sesuai dengan nama node tersebut maka node sensor yang menerima pesan akan tetap melakukan *sensing*.

3.4 Analisis Cara Base-station Menerima Pesan dari Banyak Node Sensor

Base-station mengirim pesan secara *broadcast* ke seluruh node sensor yang terhubung dan setiap node sensor memiliki waktu yang sama untuk melakukan *sensing* lalu mengembalikan hasilnya ke *base-station*. Dari hal tersebut menimbulkan pertanyaan bagaimana *base-station* dapat mengatur pesan yang datang secara bersamaan sementara *base-station* menerima pesan satu per satu dan jeda untuk melakukan *broadcast* lagi?

Langkah yang digunakan untuk mengatasi hal tersebut adalah node sensor mengirim ulang hasil *sensing* hingga diterima oleh *base-station*. Dengan cara tersebut akan membuat seluruh hasil *sensing* dapat diterima oleh *base-station*. Lalu untuk jeda *broadcast* dilakukan sesuai dengan jumlah maksimal antarmuka perangkat lunak dapat menampilkan node sensor. Misalkan setiap node membutuhkan waktu 1 detik untuk mengirim kembali hasil *sensing* dan terdapat 10 node sensor maka membutuhkan 10 detik untuk semua node sensor berhasil diterima oleh *base-station* sehingga jedanya adalah 10 detik sebelum *broadcast* lagi.

3.5 Analisis Perbandingan Perangkat Lunak yang dibangun dengan yang Sudah Ada

Setiap perangkat lunak yang sudah dibangun (Mote-View, Spyglass, Octopus, MonSense, dan Nanomon) memiliki kesamaan dan keunikannya sendiri. Berikut beberapa keunikan masing-masing setiap perangkat lunak yang sudah ada:

1. Mote-view
   - Warna node berubah ketika tidak mendapat atau menerima pesan apapun pada waktu tertentu.
   - warna node dapat diatur oleh pengguna untuk menentukan peran/keadaan setiap node.
2. Spyglass
   - perangkat lunak yang dapat menggunakan *plug-in* tambahan(*temperature map plug-in, battery plug-in*, dll) dan dapat dikembangkan sesuai keperluan pengguna
3. Octopus
   - Pengguna dapat membuka bagian developer untuk mengatur atau menambah fitur melalui API
   - satu-satunya yang mendukung tiga *data retrieval method*(*Time-driven, Query-driven,* dan *Event-driven*)
4. MonSense
   - MonSense menampilkan peta dan menggunakan GPS untuk membantu penyebaran sensor agar berada di tempat yang sesuai dan sama persis.
5. Nanomon
   - Perangkat lunak yang fleksibel karena dapat dipasang berbagai macam *plug-in* dari setiap kategori *plug-in*(*Topology plug-in, Sensor Data plug-in, Chart plug-in,* dan *Sensor list plug-in*)

Selain keunikannya, perangkat-perangkat lunak tersebut memiliki kesamaan penting yang membuat mereka menjadi sebuah alat pemantauan WSN. Hal-hal yang sama, yaitu:

1. Perangkat lunak menampilkan node-node sensor yang menyala dan mati.
2. Perangkat lunak menampilkan identitas node sensor.
3. Perangkat lunak menampilkan hasil *sensing*.
4. Hasil *sensing* disimpan pada sebuah *file*.
5. Perangkat lunak menampilkan sisa energi dari baterai setiap node sensor.
6. Perangkat lunak memiliki fitur untuk pengontrolan.
7. Perangkat lunak menampilkan letak node sensor sesuai di dunia nyata dan terhubung ke mana.

Dengan begitu perangkat lunak yang akan dibangun pada penelitian ini akan mengikuti aspek dari kesamaan setiap perangkat lunak yang sudah ada, tetapi lebih sederhana/kurang kompleks. Hal-hal yang disederhanakan untuk perangkat lunak yang dibangun adalah:

1. Perangkat lunak memiliki fitur pengontrolan yang sederhana(tidak seperti Octopus).
2. Perangkat lunak menampilkan node sensor tanpa menampilkan sisa energi dari baterai.
3. Perangkat lunak menampilkan posisi node sensor yang diurutkan letaknya berdasarkan nomor daripada letak aslinya.

BAB 4

PERANCANGAN

Pada bab ini dijelaskan mengenai perancangan aplikasi yang dibangun meliputi perancangan antarmuka, perancangan diagram sequence, dan perancangan kelas Aplikasi Pemantauan WSN.

4.1 Perancangan Masukan dan Keluaran

Perangkat lunak akan dibangun menggunakan bahasa Java dengan masukan berupa *file text* yang menyimpan nama-nama node sensor dengan batas masing-masing nama node sensor terdiri dari 1 kata yang memiliki panjang 4 karakter heksadesimal dan keluaran berupa *file text* yang memiliki nilai *log* dari hasil jalannya perangkat lunak.

4.2 Perancangan Format Pesan

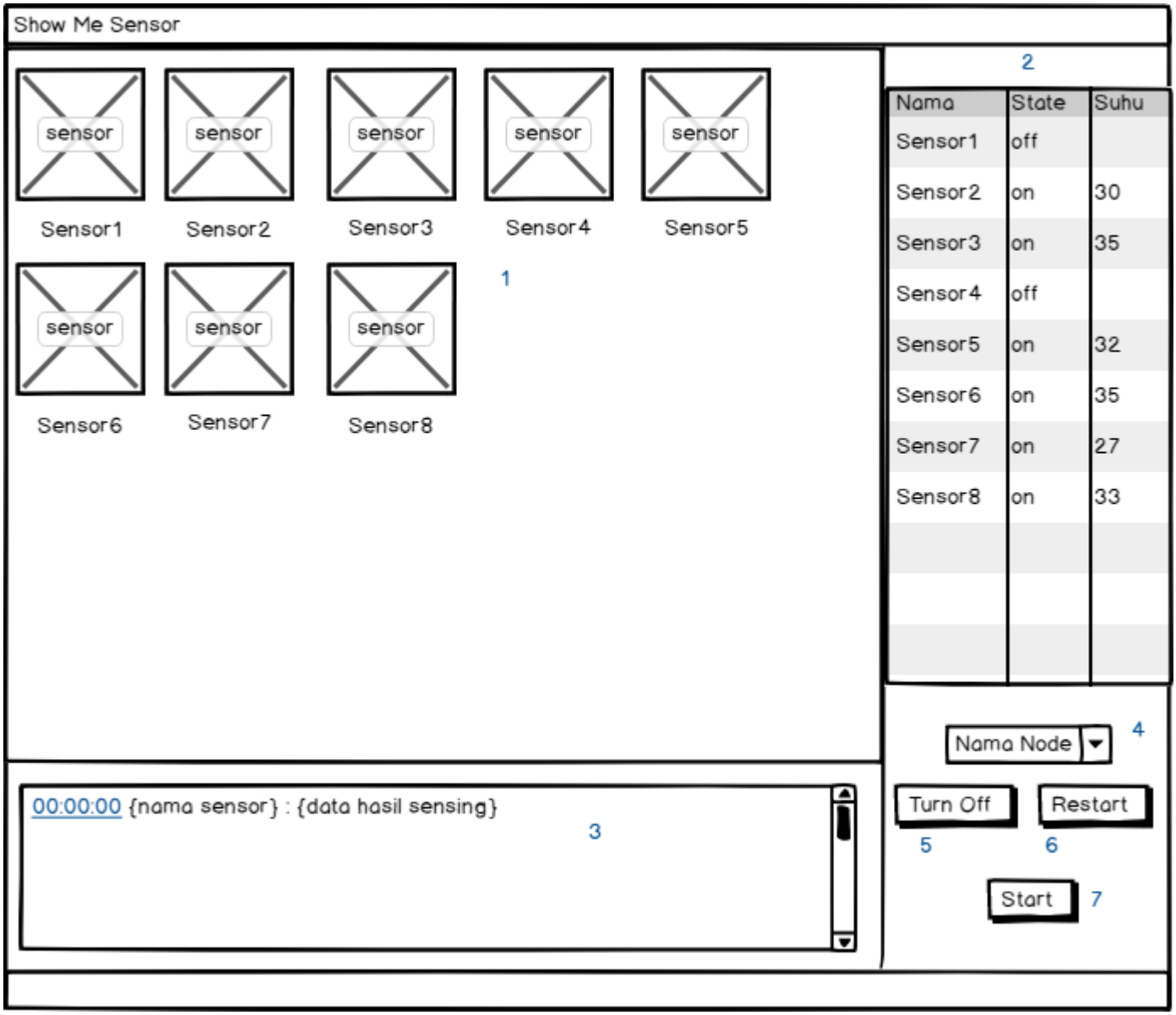
Berdasarkan hasil analisis format pesan pada subbab 3.2.7, format pesan pengiriman dari *base-station* ke node sensor jika melakukan *turn off* maka formatnya adalah "nama node sensor#stop". Karakter "#" digunakan sebagai pembagi nama node dengan pesan yang akan dibaca, sedangkan untuk *restart* hanya merubah kata "stop" menjadi "restart".

Format pesan dari node sensor ke *base-station* jika melakukan *sensing* adalah "suhu #kelembaban#tekanan", sedangkan untuk *restart* dan *turn off* pesannya berupa "is restarting" atau "is stopping".

Format pesan dari *base-station* ke perangkat lunak adalah "nama#pesan dari node sensor#waktu".

4.3 Perancangan Antarmuka

Pada perangkat lunak ini, dibutuhkan antarmuka untuk memudahkan pengguna dalam menggunakan perangkat lunak. Rancangan antarmuka untuk perangkat ini dapat dilihat pada Gambar 4.1.



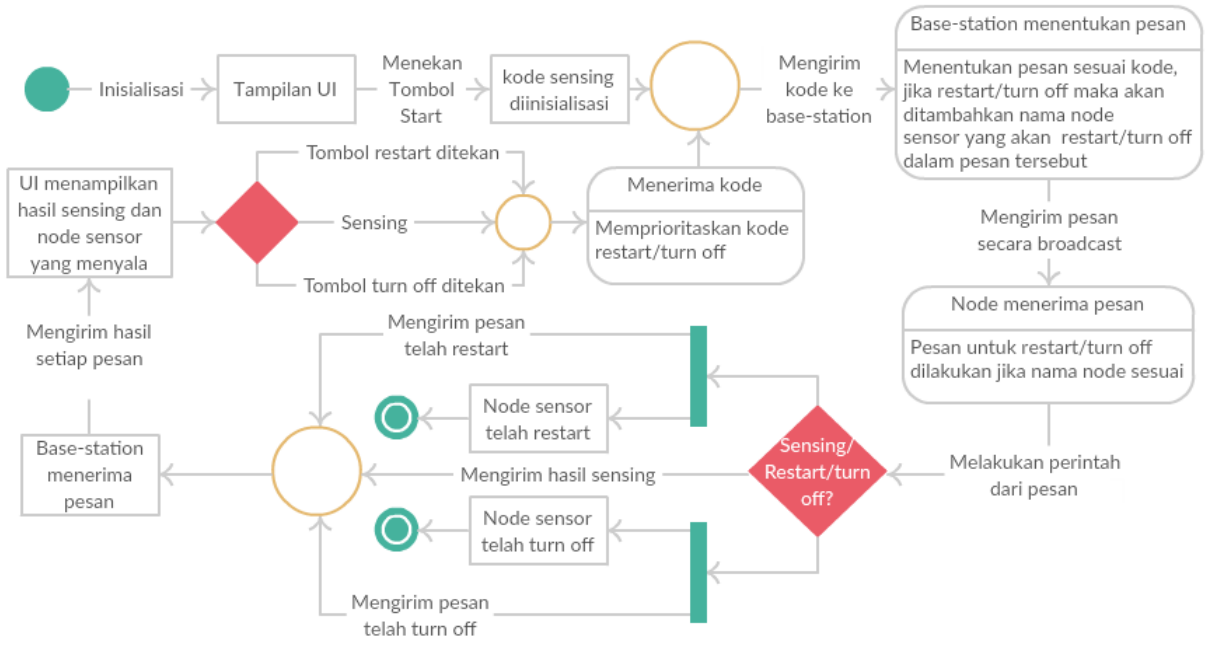
Gambar 4.1: Rancangan Antarmuka Perangkat Lunak.

No.	Tipe	Fungsi
1.	*ImageView*	Menampilkan node sensor yang menyala dan mati.
2.	*Table*	Tabel yang memperlihatkan nama sensor, keadaannya, dan data *sensing* tiap sensor
3.	*TextArea*	Tempat *log* ketika ada data baru masuk akan ditampilkan disana beserta waktunya
4.	*ComboBox*	Menyediakan nama-nama sensor yang ada dan digunakan untuk memilih node sensor yang ingin dimatikan atau *restart*
5.	*Button*	Memberi pesan kepada *base-station* bahwa sensor dengan nama pilihan sesuai *ComboBox* untuk dimatikan
6.	*Button*	Memberi pesan kepada *base-station* bahwa sensor dengan nama pilihan sesuai *ComboBox* untuk di-*restart*
7.	*Button*	Memberi pesan kepada *base-station* untuk mulai bekerja.

Tabel 4.1: Tabel fungsi elemen yang terdapat pada rancangan antarmuka

4.4 Perancangan Diagram State Pengembangan Pemantauan WSN

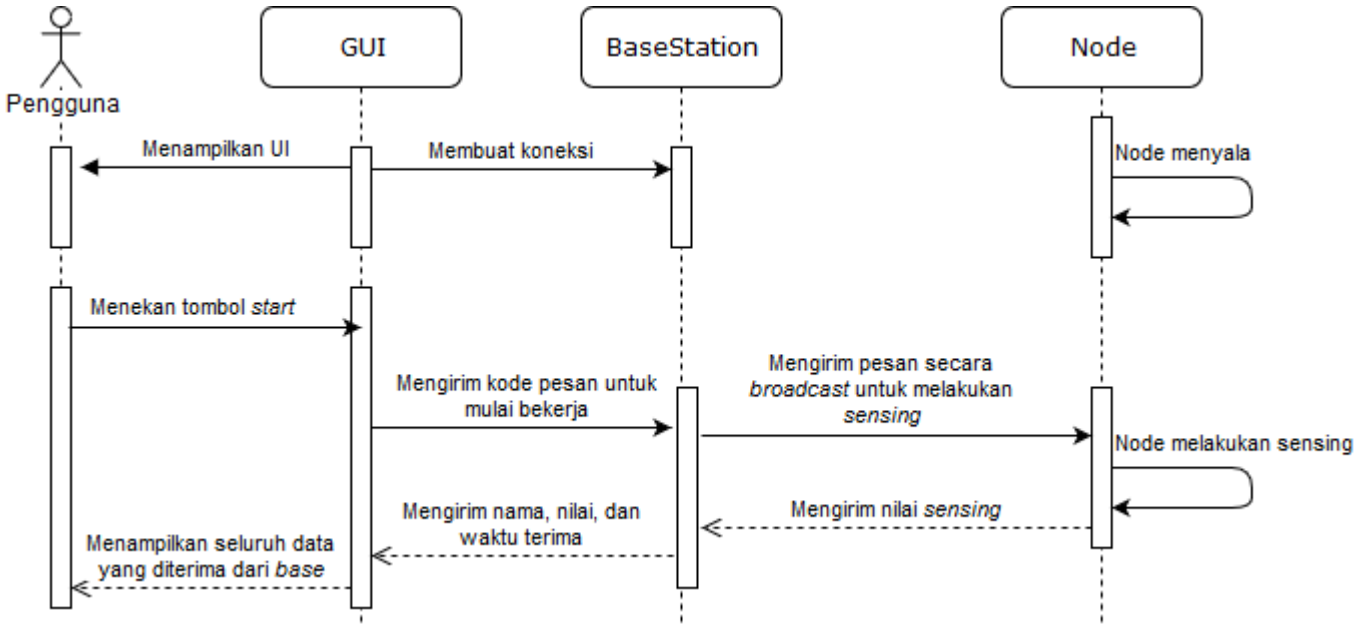
Diagram state yang dibuat berdasarkan langkah kerja perangkat lunak yang telah dibangun dan dapat dilihat pada gambar 4.2.



Gambar 4.2: Diagram State Perangkat Lunak.

4.5 Perancangan Diagram Sequence Pengembangan Pemantauan WSN

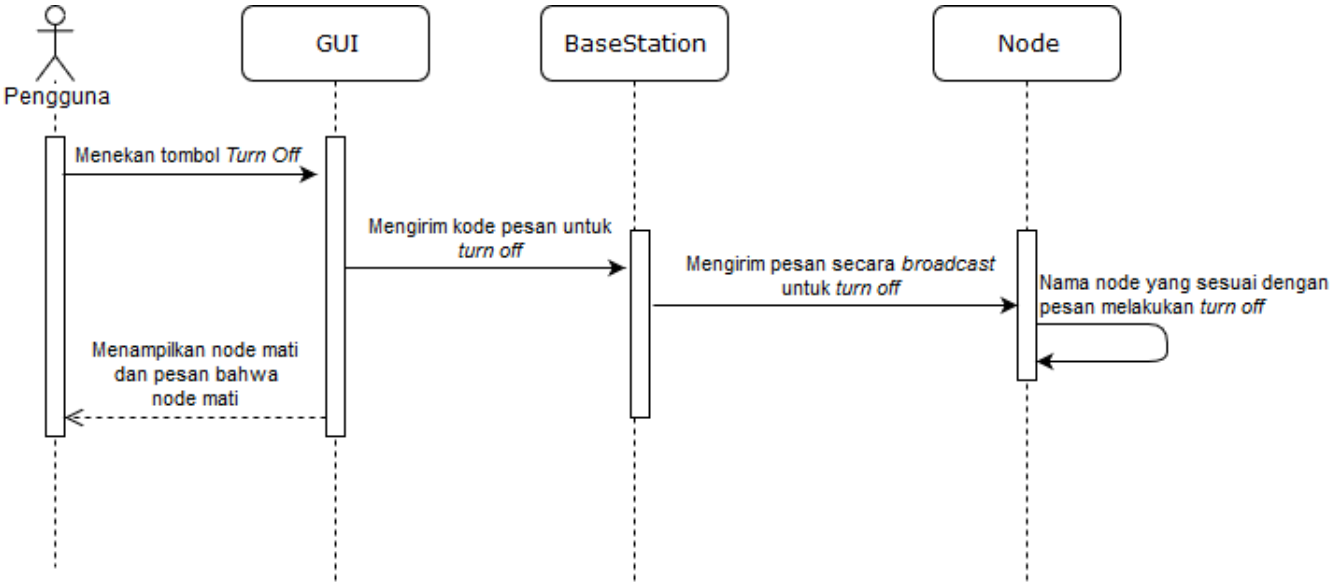
4.5.1 Diagram Sequence Fitur *Start*



Gambar 4.3: Diagram Sequence pengguna menggunakan fitur *start*.

Berdasarkan gambar 4.3, pengguna memulai dengan membuka perangkat lunak lalu perangkat lunak menampilkan UI yang sudah dibuat. *Base-station* menyala bersama dengan dijalankannya perangkat lunak. Pengguna kemudian meng-*click* tombol *start* lalu perangkat lunak akan mengirimkan perintah atau lebih tepat adalah kode kepada *base-station* untuk mulai bekerja. Setelah itu *base-station* akan mengirim pesan kepada node untuk melakukan *sensing* yang hasilnya akan dikembalikan kepada *base-station*. *Base-station* menerima hasil *sensing* dan kemudian mengirimnya beserta dengan nama node dan waktu penerimaannya kepada GUI yang kemudian ditampilkan kepada pengguna.

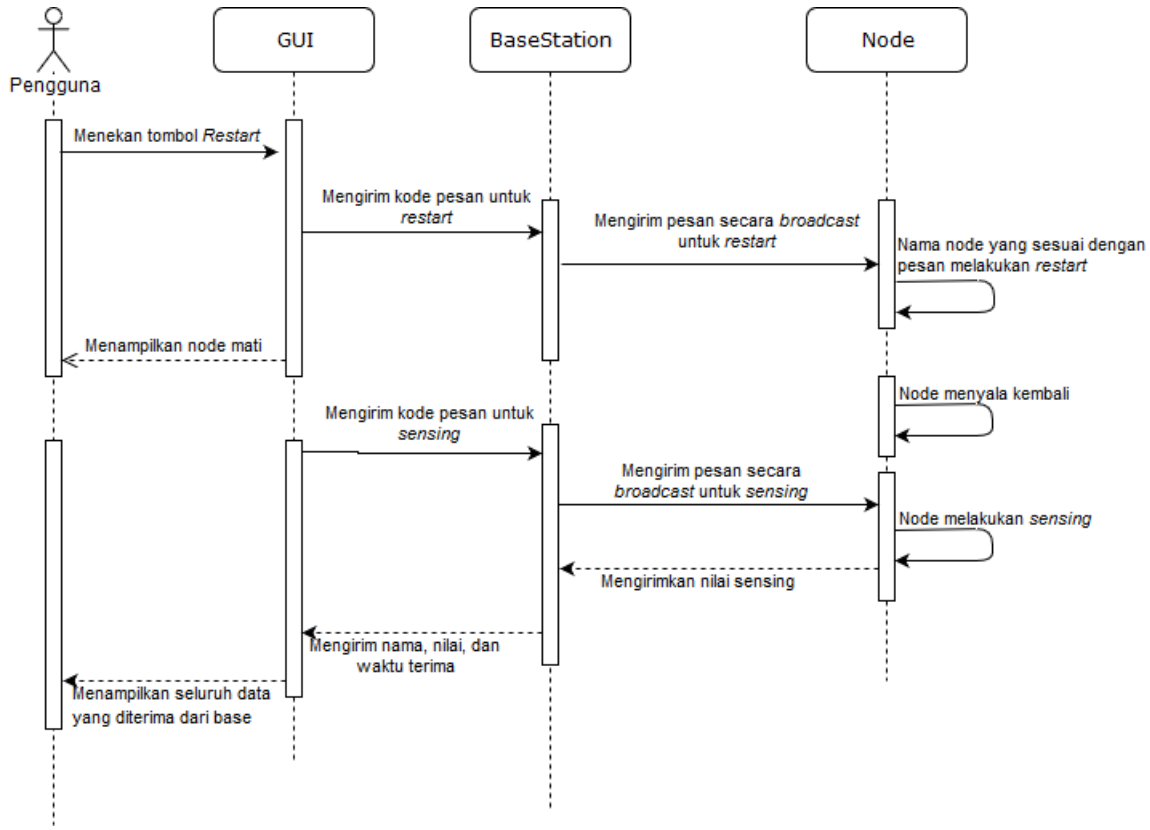
4.5.2 Diagram Sequence Fitur *Turn Off*



Gambar 4.4: Diagram Sequence pengguna menggunakan fitur *turn off*.

Sebelum menggunakan fitur ini perangkat lunak sudah berjalan dan pengguna sudah menggunakan fitur *start*. Berdasarkan gambar 4.4, pengguna meng-*click* tombol *turn off*. Perangkat lunak akan memberi perintah atau kode untuk melakukan *turn off* lalu *base-station* akan mengirim pesan kepada node sensor untuk melakukan *turn off*. Node akan mati dan *base-station* akan mengembalikan pesan kepada perangkat lunak bahwa node sensor telah mati dan perangkat lunak akan menampilkan node sensor yang tadinya menyala menjadi mati kepada pengguna.

4.5.3 Diagram Sequence Fitur *Restart*



Gambar 4.5: Diagram Sequence pengguna menggunakan fitur *restart*.

Sebelum menggunakan fitur ini perangkat lunak sudah berjalan dan pengguna sudah menggunakan fitur *start*. Berdasarkan gambar 4.5, pengguna meng-*click* tombol *restart*. Perangkat lunak akan memberi perintah atau kode pesan untuk melakukan *restart* lalu *base-station* akan mengirim pesan kepada node sensor untuk melakukan *restart*. Node akan mati dan menyala kembali lalu menunggu pesan dari *base-station*, setelah itu bekerja seperti biasa.

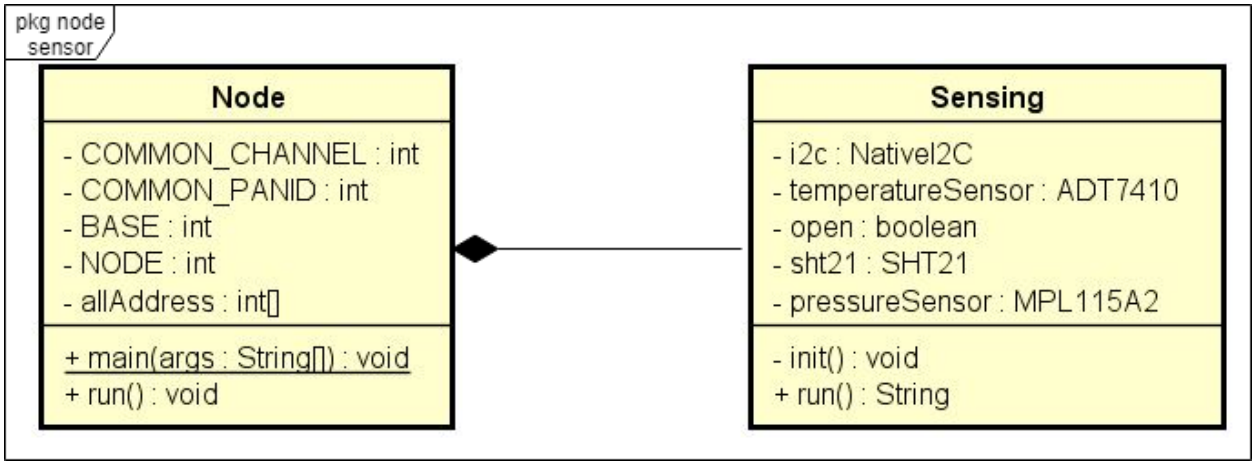
4.6 Diagram Kelas Detail Perangkat Lunak

Berdasarkan analisis kebutuhan perangkat lunak yang telah dilakukan pada Bab 3 tersebut, perangkat lunak yang akan dibangun akan memiliki struktur kelas diagram yang sama. Untuk penjelasan lebih lanjut masing-masing kelas ialah sebagai berikut:

1. Kelas Sensing

   Pada kelas ini akan dilakukan *sense* untuk mendapatkan suhu, kelembaban, dan tekanan. Berdasarkan gambar 4.6, penjelasan setiap atribut dan metode pada kelas ini sebagai berikut:

   - **Atribut**
     - NativeI2C i2c
       Atribut untuk melakukan inisialisasi *driver NativeI2C* untuk digunakan oleh sensor-sensor.
     - ADT7410 temperatureSensor
       Atribut ini digunakan sebagai *import* dari *driver* sensor untuk mengukur suhu yang ada pada node sensor Preon32.



Gambar 4.6: Diagram Kelas detil Sensing dan Node.

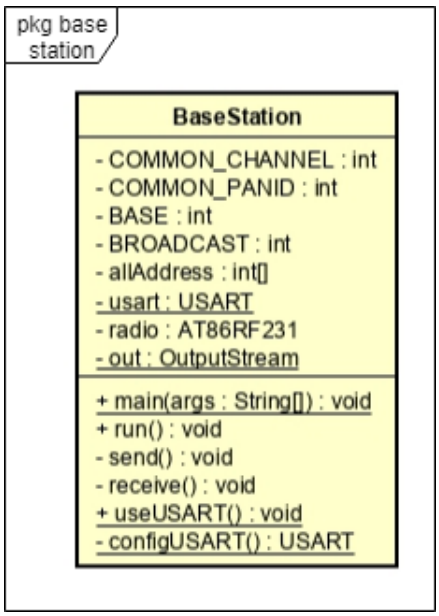
  - boolean open
    Atribut ini digunakan untuk menandakan apakah *driver* untuk *sensing* sudah menyala atau belum.
  - SHT21 sht21
    Atribut ini digunakan sebagai *import* dari *driver* sensor untuk mengukur kelembaban yang ada pada node sensor Preon32.
  - MPL115A2 pressureSensor
    Atribut ini digunakan sebagai *import* dari *driver* sensor untuk mengukur tekanan yang ada pada node sensor Preon32.
- **Metode**
  - void init()
    Metode ini digunakan untuk menyalakan *driver* yang diperlukan setiap sensor untuk melakukan *sensing*.
  - String run
    Metode ini digunakan untuk melakukan *sensing*. Metode ini mengembalikan *string* yang berisi nilai *sensing* dari setiap sensor yang digunakan.

2. Kelas Node

   Kelas ini akan diunggah ke dalam node sensor yang akan melakukan *sensing* dan mengirimkan hasilnya kepada *base-station*. Berdasarkan gambar 4.6, penjelasan setiap atribut dan metode pada kelas ini sebagai berikut:

   - **Atribut**
     - int COMMON_CHANNEL
       Atribut ini digunakan untuk menyimpan *Channel* dari satu jaringan node sensor.
     - int COMMON_PANID
       Atribut ini digunakan untuk menyimpan *PAN ID* dari satu jaringan node sensor.
     - int BASE
       Atribut ini digunakan untuk menyimpan alamat dari *base-station*.
     - int NODE
       Atribut ini digunakan untuk menyimpan alamat dari node itu sendiri.
     - final int[] allAddress
       Atribut ini digunakan untuk menyimpan semua alamat node yang ada dibuat atau sesuai dengan semua nama di *file text*.

- **Metode**
  - static void main(String[] args)
    Metode ini digunakan sebagai metode utama dari kelas ini dan memanggil metode runs().
  - void run()
    Metode ini digunakan untuk menerima pesan dari *base-station* berupa *sense, restart*, atau *turn off*. Jika pesan berisikan *sense* maka dilanjutkan dengan memanggil kelas Sensing dan mengirim hasilnya ke *base-station*. Jika *restart* atau *turn off* maka node sensor akan mati (*turn off)* dan menyala kembali (*restart*).



Gambar 4.7: Diagram Kelas detil BaseStation.

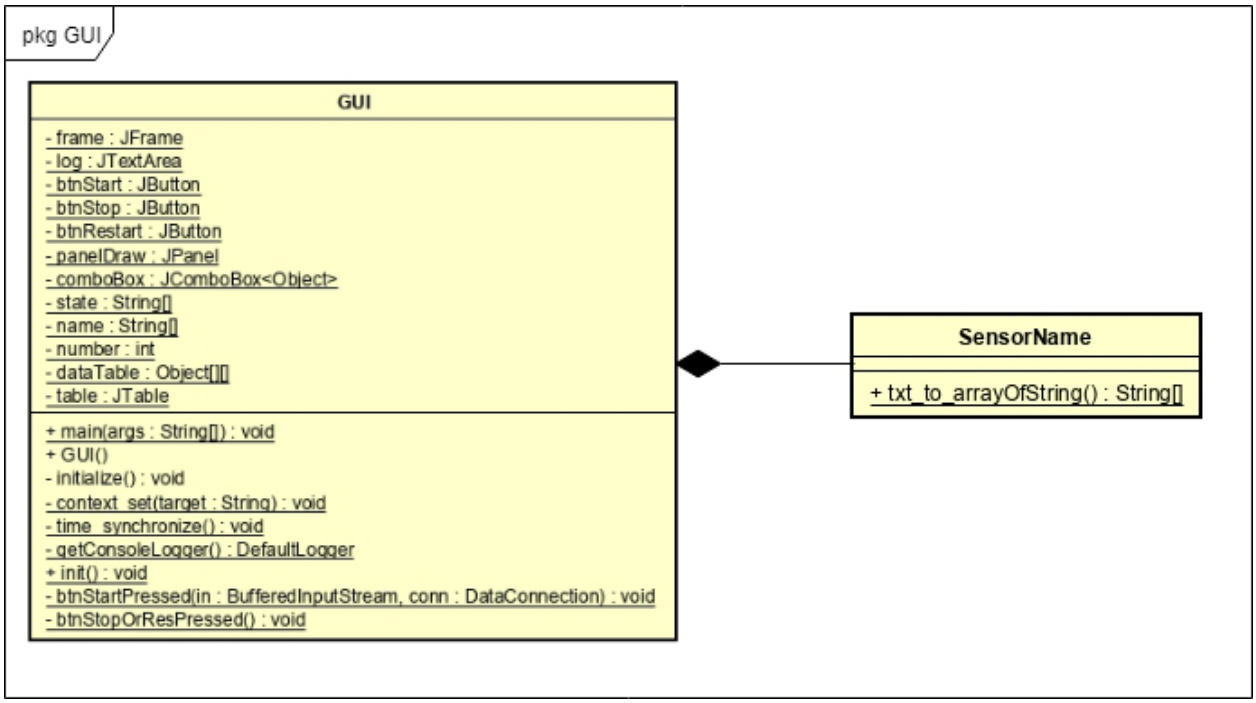
3. Kelas BaseStation

   Kelas ini akan diunggah ke dalam *base-station*. Kela ini menerima perintah dari kelas GUI untuk mengirim pesan kepada node sensor dan menerima pesan hasil *sensing* dari node sensor. Berdasarkan gambar 4.7, penjelasan setiap atribut dan metode pada kelas ini sebagai berikut:

   - **Atribut**
     - int COMMON_CHANNEL
       Atribut ini digunakan untuk menyimpan *Channel* dari satu jaringan node sensor.
     - int COMMON_PANID
       Atribut ini digunakan untuk menyimpan *PAN ID* dari satu jaringan node sensor.
     - int BASE
       Atribut ini digunakan untuk menyimpan alamat dari *base-station* itu sendiri.
     - int BROADCAST
       Atribut ini digunakan untuk menyimpan alamat *broadcast*(semua node sensor yg terjangkau).
     - final int allAddress
       Atribut ini digunakan untuk menyimpan semua alamat node yang ada dibuat atau sesuai dengan semua nama di *file text*.
     - static USART usart
       Atribut ini digunakan untuk inisialisasi USART yang menghubungkan *base station* dengan komputer yang terhubung langsung.

  - AT86TF231 radio
    Atribut ini digunakan sebagai *import* dari *driver* node sensor Preon32 untuk menggunakan frekuensi radio.
  - static OutputStream out
    Atribut ini digunakan untuk inisialisasi OutputStream yang menulis data dari *base-station* ke program komputer agar bisa dilihat oleh pengguna.

- **Metode**
  - static void main(String[] args)
    Metode ini digunakan sebagai metode utama dari kelas ini dan memanggil metode runs(), useUSART() dan melakukan inisialisasi atribut out.
  - void runs()
    Metode ini digunakan untuk inisialisasi radio lalu memanggil metode send() dan receive().
  - void send()
    Metode ini digunakan untuk mengirimkan pesan sesuai perintah dari kelas GUI menuju node mana saja (*broadcast*).
  - void receive()
    Metode ini digunakan untuk menerima data dari node sensor lalu diteruskan ke kelas GUI.
  - static void useUSART()
    Metode ini melakukan inisialisasi atribut usart dengan memanggil metode configUSART()
  - static USART configUSART()
    Metode ini digunakan untuk mengatur konfigurasi yang akan digunakan atribut usart.



Gambar 4.8: Diagram Kelas detil GUI dan SensorName.

4. Kelas SensorName

   Pada kelas ini terdapat sebuah metode yang yang membaca sebuah *file text* nama-nama sensor dan menjadikannya *array of string* yang akan digunakan pada kelas GUI

5. Kelas GUI

   Kelas ini sebagai antarmuka yang akan ditampilkan kepada pengguna dan menampilkan nilai-nilai sesuai dengan apa yang dikirimkan *base-station*. Berdasarkan gambar 4.8, penjelasan setiap atribut dan metode pada kelas ini sebagai berikut:

   - **Atribut**
     - static JTextArea log
       Atribut ini digunakan untuk menampilkan setiap ada data baru yang masuk dan sudah diproses sehingga mudah dibaca oleh pengguna.
     - static JButton btnStart
       Atribut ini digunakan untuk mengirim perintah kepada *base-station* untuk mulai bekerja dan menerima data dari *base-station* yang akan diproses untuk disimpan pada *file text log* dan tabel untuk ditampilkan.
     - static JButton btnStop
       Atribut ini akan mengganti perintah untuk mematikan node sensor sesuai atribut comboBox.
     - static JButton btnRestart
       Atribut ini akan mengganti perintah untuk *restart* node sensor sesuai atribut comboBox.
     - static JPanel panelDraw
       Atribut ini digunakan sebagai wadah untuk menampilkan gambar node sensor(menyala dan mati) beserta namanya.
     - static JComboBox<Object> comboBox
       Atribut ini menyimpan nama-nama sensor sesuai atribut name yang dapat menjadi pilihan node sensor yang ingin dimatikn atau *restart*.
     - static String[] state
       Atribut ini menyimpan keadaan node sensor(mati atau nyala) sesuai jumlah nama dari atribut name.
     - static String[] name
       Atribut ini menyimpan nama-nama sensor yang didapat dari kelas SensorName.
     - static int number
       Atribut ini digunakan sebagai kode perintah yang akan dikirimkan ke *base-station*.
     - static Object[][] dataTable
       Atribut ini digunakan sebagai isi tabel yang berupa nama, *state*, dan nilai *sensing*.
     - static JTable table
       Atribut ini digunakan sebagai wadah untuk menyimpan nilai terakhir dari sebuah node sensor.
   - **Metode**
     - static void main(String[] args)
       Metode ini digunakan untuk menampilkan antarmuka yang sudah diinisialisasi oleh metode GUI(), memanggil metode context_set(String target), memanggil metode time_synchronize(), dan init().
     - GUI()
       Metode ini digunakan untuk inisialisasi atribut name, atribut state, dan memanggil metode initialize().

- void initialize()
  Metode ini digunakan untuk membuat antarmuka sesuai dengan gambar 4.1.
- static void context_set(String target)
  Metode ini digunakan untuk memilih *context* yang akan digunakan misalnya *context* 1 adalah *base-station* sementara 2, 3 dan seterusnya adalah node sensor.
- static void time_synchronize()
  Metode ini digunakan untuk melakukan sinkronisasi waktu *base-station* sesuai dengan waktu pada komputer saat itu.
- static DefaultLogger getConsoleLogger()
  Metode ini digunakan untuk memunculkan tulisan pada *console*. Metode ini digunakan oleh metode void context_set(String target) dan time_synchronize().
- static void init()
  Metode ini digunakan untuk membangun koneksi antara program pada komputer dengan program pada *base station*. Metode ini juga memanggil metode btnStartPressed(BufferedInputStream in,DataConnection conn) dan btnStopOrResPressed()
- static void btnStartPressed(BufferedInputStream in,DataConnection conn)
  Metode ini memiliki parameter yang diperlukan ketika tombol *start* ditekan akan melakukan inisialisasi atribut number yang berisikan perintah.
  sensing,*restart*, atau*turn off* dan membutuhkan atribut lokal conn untuk mengantarkan atribut number kepada *base-station* lalu menunggu *base-station* mengembalikan data yang akan diterima melalui atribut lokal in.
- static void btnStopOrResPressed()
  Metode ini akan mengganti nilai atribut number yang akan digunakan sebagai kode *restart* atau *turn off* lalu setelah beberapa saat nilai atribut number kembali seperti semula ketika diinisialisasi pada metode btnStartPressed(BufferedInputStream in,DataConnection conn)

4.7 Perancangan Pseudocode Aplikasi Pengembangan Pemantauan WSN

Pada subbab ini dirancang *pseudocode* sesuai dengan analisis algoritma pada subbab 3.3. *Pseudocode* yang dibuat pada kelas BaseStation, Node, dan GUI terkait dengan pengiriman pesan, data hasil sensing, dan menampilkan node sensor yang menyala dan mati.

4.7.1 Node

Pada kelas Node terdapat algoritma pada metode *run* yang digunakan untuk menerima pesan dari *base-station* dan mengirim hasilnya kembali ke *base-station*. Jika pesan yang diterima berupa *turn off* maka node akan mengirimkan pesan untuk memberitahu bahwa dia akan mati setelah itu barulah node sensor tersebut mati. Jika pesan yang diterima berupa *restart* maka node akan memberikan pesan untuk memberitahu bahwa dia akan *restart* lalu akan mati dan tidak bisa menerima pesan apapun, setelah beberapa saat node tersebut akan menyala kembali dan bekerja seperti biasa.

Algorithm 1 Metode run()

```
1: function RUN
2:     NODE ← allAddress[2]
3:     final AT86RF231 radio ← RadioInit.intRadio();
4:     radio.setChannel(COMMON_CHANNEL)
5:     radio.setPANId(COMMON_PANID)
6:     radio.setShortAddress(NODE)
```

```
7:    Thread T ← thread baru do
8:    function RUN
9:        Sensing st ← membuat objek kelas sensing
10:       while true do
11:           Frame f ← null
12:           try
13:               f ← membuat frame baru
14:               radio akan menggunakan "AT86RF231.STATE_RX_AACK_ON" untuk meng-
      embalikan ACK ketika mendapatkan frame(paket)
15:               radio.setState(AT86RF231.STATE_RX_AACK_ON)
16:               radio.waitForFrame(f)
17:           catch Expected
18:           end try
19:           if f is not null then
20:               dg ← f.getPayload()
21:               str ← new String(dg, 0, dg.length)
22:               name ← ""
23:               order ← ""
24:               boolean done ← false
25:               while done is false do
26:                   try
27:                       if str != "sense" then
28:                           name ← str.substring(0,4)
29:                           order ← str.substring(5)
30:                           if name == Integer.toHexString(NODE) then
31:                               thread sleep 88 mili detik
32:                               message ← order
33:                           else
34:                               message ← st.run()
35:                           end if
36:                       else
37:                           message ← st.run()
38:                       end if
39:                       frame ← membuat frame baru
40:                       Melakukan setting pada frame (alamat asal dan tujuan)
41:                       radio akan menggunakan "AT86RF231.STATE_TX_ARET_ON" dengan
      harapan menerima ACK ketika mentransmit(mengirim) frame yang telah dibuat, jika tidak
      menerima ACK maka akan dilakukan pengiriman ulang
42:                           done ← true
43:                   catch exception
44:                   end try
45:               end while
46:               if name sama dengan Integer.toHexString(NODE) then
47:                   try
48:                       if order sama dengan "stop" then
49:                           CPUHelper.setPowerState(CPUConstants.V_POWER_STATE_OFF,
      Long.MAX_VALUE)
50:                       else if order sama dengan "restart" then
51:                           CPUHelper.setPowerState(CPUConstants.V_POWER_STATE_OFF,
      8000)
52:                       end if
53:                   catch exception
54:                   end try
```

```
55:                 end if
56:             end if
57:         end while
58:     end function
59:     end thread
60:     t.start()
61: end function
```

4.7.2 BaseStation

Pada kelas BaseStation terdapat algoritma pada metode *send* dan *receive*. Kedua metode bekerja secara bersamaan dengan menggunakan *Thread*. Metode *send* digunakan untuk mengirimkan pesan sesuai dengan perintah yang didapat dari perangkat lunak ke node sensor dengan cara *broadcast*. Metode *receive* digunakan untuk menerima pesan dari node sensor untuk diproses hasilnya dan dikirim ke perangkat lunak.

Algorithm 2 Metode send()

```
 1: function SEND
 2:     Thread T ← thread baru do
 3:     function RUN
 4:         msg ← "sense"
 5:         int choice
 6:         while true do
 7:             try
 8:                 choice ← BaseStation.usart.read()
 9:                 if choice = 50 then
10:                     msg ← "sense"
11:                 else if choice < 50 then
12:                     msg ← Integer.toHexString(allAddress[choice]) + "#" + "stop"
13:                 else if choice > 99 then
14:                     msg ← Integer.toHexString(allAddress[choice - 100]) + "#" + "restart"
15:                 end if
16:                 message ← msg
17:                 frame ← membuat frame baru
18:                 frame.setSrcAddr(BASE)
19:                 frame.setSrcPanId(COMMON_PANID)
20:                 frame.setDestAddr(BROADCAST)
21:                 frame.setDestPanId(COMMON_PANID)
22:                 radio.setState(AT86RF231.STATE_TX_ARET_ON)
23:                 frame.setPayload(message.getBytes())
24:             catch Expected
25:             end try
26:         end while
27:     end function
28:     end thread
29:     t.start()
30: end function
```

Algorithm 3 Metode receive()

```
1: function RECEIVE
2:     Thread T ← thread baru do
3:     function RUN
4:         String toPC, str, hex_addr
5:         long timeRaw
6:         byte[] dg
7:         while true do
8:             Frame f ← null
9:             try
10:                f ← membuat frame baru
11:                radio.setState(AT86RF231.STATE_RX_AACK_ON)
12:                radio.waitForFrame(f, 0)
13:                if f is not null then
14:                    timeRaw ← waktu sekarang
15:                    dg ← f.getPayload()
16:                    str ← new String(dg, 0, dg.length)
17:                    toPC ← hex_addr + "#" + str + "#" + timeRaw + "%"
18:                    Tulis toPC ke file
19:                end if
20:            catch Expected
21:            end try
22:        end while
23:    end function
24:    end thread
25:    t.start()
26: end function
```

4.7.3 GUI

Pada kelas GUI terdapat algoritma pada metode *btnStartPressed*. Metode diawali dengan pengguna men-*click* tombol *start* lalu perangkat lunak akan mengirimkan perintah ke *base-station* untuk melakukan *sensing* ke semua node sensor yang ada. Setelah *base-station* mengirim hasilnya, perangkat lunak akan membaca pesan tersebut dan memprosesnya agar mudah dimengerti oleh pengguna saat ditampilkan pada antarmuka.

Algorithm 4 Metode btnStartPressed(BufferedInputStream in,DataConnection conn)

```
1: function BTNSTARTPRESSED(BufferedInputStream in,DataConnection conn)
2:     btnStart.addActionListener(new ActionListener()
3:         function ACTIONPERFORMED(ActionEvent e)
4:             thread T ← thread baru do
5:                 function RUN
6:                     number ← 50
7:                     Set button start, restart dan stop enable
8:                     Matriks boolean isON dan matriks countOff sepanjang nama sensor yang
   tersedia
9:                     Semua nilai countOFF menjadi 8
10:                    do
11:                        Semua nilai isON menjadi false
```

```
12:                     try
13:                         Flush konensi
14:                         conn.write(number)
15:                         Buat matrik buffer bertipe Byte dengan ukuran 1 × 1024
16:                         Thread sleep 5000 mili detik
17:                         while in.available() > 0 do
18:                             s ← Menyimpan hasil read buffer
19:                             subNode ← Split s dengan "%"
20:                             for j=0 : subNode.length -1 do
21:                                 subStr ← split subNode[j] dengan "#"
22:                                 for i=0 : state.length do
23:                                     if subStr[0] = name[i] then
24:                                         if isOn[i] then
25:                                             break
26:                                         end if
27:                                         if subStr[1] != "restart" and subStr[1] != "stop" then
28:                                             isOn[i] ← true
29:                                             countOff[i] ← 1
30:                                             Simpan temperature, humidity dan barometer dari
    subStr[1], subStr[2], dan subStr[3]
31:                                             timeRaw ← subStr[4]
32:                                             clock ← Mengganti format waktu timeRaw
33:                                             state[i] ← "on"
34:                                             Menyiapkan BufferedWriter untuk mencatat ke file
    log
35:                                             Menyusun dataTable[i] dengan hasil sensing, yaitu
    temperatur, humidity dan barometer
36:                                             result ← menyusun data yang akan dicatat ke log
37:                                             Mencatat result ke file log
38:                                             break
39:                                         else
40:                                             isON[i] ← on
41:                                             timeRaw ← waktu sistem
42:                                             Format waktu sistem untuk disimpan ke log
43:                                             if subStr[1] = "restart" then
44:                                                 subStr[1] ← "restarting"
45:                                             else
46:                                                 subStr[1] "stopping"
47:                                             end if
48:                                             Menyimpan waktu dan subStr[1] ke file log
49:                                             Menyusun dataTable[i] untuk state mati
50:                                             countOff[i] ← 4
51:                                             break
52:                                         end if
53:                                     end if
54:                                 end for
55:                             end for
56:                         end while
```

```
57:                         for i=0 : isON.length do
58:                             if isON[i] = false then
59:                                 if countOFF[i] < 4 then
60:                                     countOFF[i] ← countOFF[i] + 1
61:                                 else if countOFF[i] >= 4 and countOFF[i] < 8 then
62:                                     state[i] ← "off
63:                                     Menyusun dataTable[i] untuk sensor mati
64:                                     counOFF[i] ← countOFF[i] + 1
65:                                 else
66:                                     state[i] ← "none"
67:                                     Menyusun dataTable[i] bahwa node hilang
68:                                 end if
69:                             end if
70:                         end for
71:                         Repaint antarmuka
72:                     catch Exception
73:                     end try
74:                 while true
75:             end function
76:         end thread
77:         t.start()
78:     end function
79: end function
```

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Bab ini terdiri atas implementasi, pengujian, dan masalah yang dihadapi. Pada bagian implementasi dijelaskan mengenai lingkungan implementasi dan hasil dari implementasi. Pada bagian pengujian berisi hasil dari pengujian. Pada bagian masalah yang dihadapi dijelaskan masalah-masalah yang dihadapi pada saat implementasi.

5.1 Implementasi

5.1.1 Lingkungan Implementasi

Spesifikasi perangkat keras yang digunakan untuk implementasi adalah Notebook Asus A455L dengan spesifikasi sebagai berikut:

Processor	Intel Core i3-4030U 1.9 Ghz
Memory	6144MB DDR3L
Storage	500GB HDD
VGA	NVDIA GeForce 840M + Intel HD Graphics Family
Operating System	Windows 10 64-bit

Berikut adalah spesifikasi perangkat lunak yang digunakan untuk implementasi:

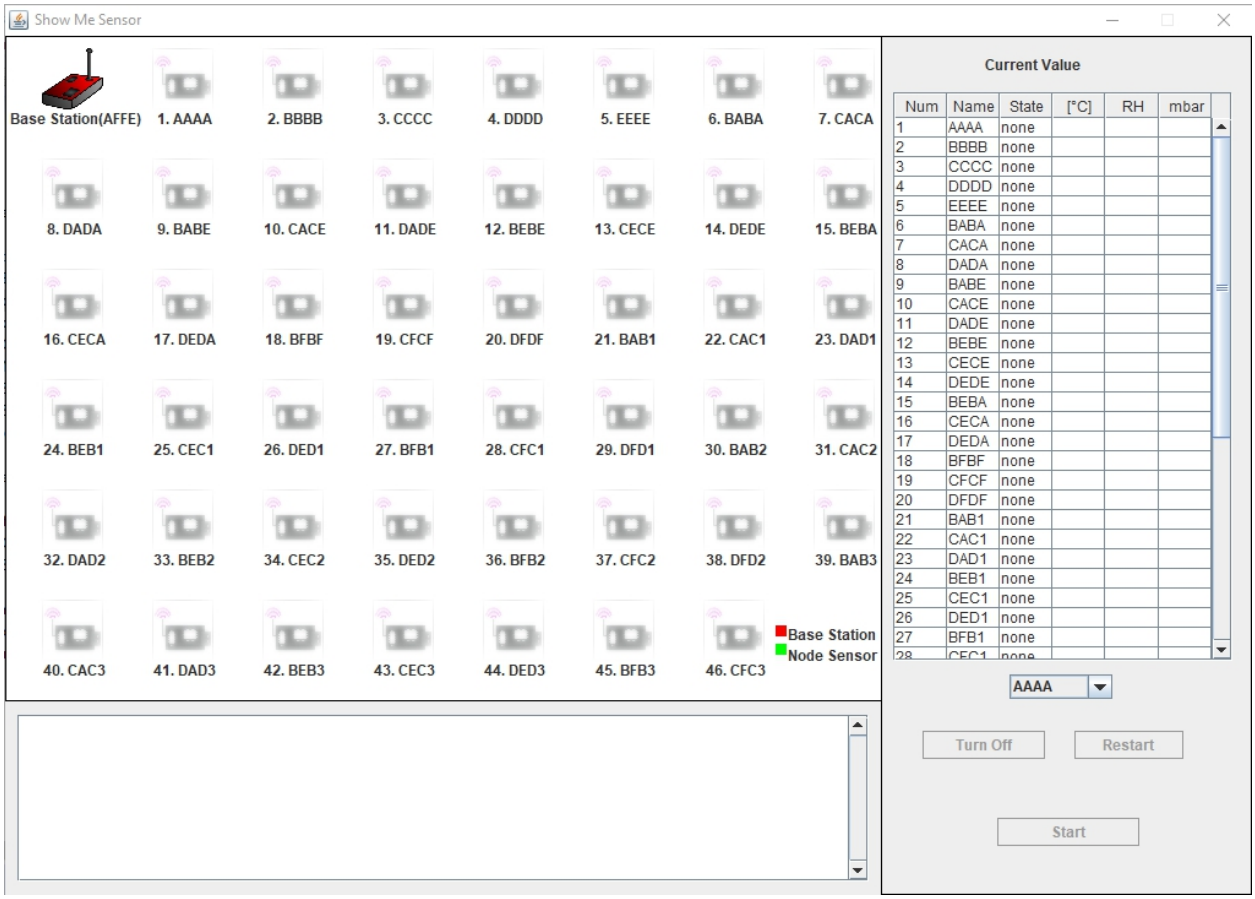
IDE	Eclipse IDE 2018-09 (4.9.0)
Bahasa Pemrograman	Java
Java Library	Java 1.8.0_201

Berikut adalah spesifikasi node sensor yang digunakan untuk implementasi:

Nama	Eclipse IDE 2018-09 (4.9.0)
Processor	Java
Operating System	PreonVM
Penyimpanan Sistem	64 kByte SRAM
Penyimpanan Data	8Mbit
Pita frekuensi	2400.0 - 2483.5 MHz
Jangkauan	250 meter (luar ruangan) dan 30 meter (dalam ruangan)
Sensor-sensor	Sensor suhu, sensor cahaya, sensor kelembaban, sensor tekanan udara, dan sensor getaran

5.1.2 Implementasi Antarmuka Perangkat Lunak

Gambar 5.1 adalah tampilan awal implementasi perangkat lunak dari perancangan yang telah dilakukan pada Bab 4. Pengguna menunggu beberapa saat sebelum bisa menggunakan perangkat lunak karena programnya sedang melakukan inisiasi.



Gambar 5.1: Antarmuka Perangkat Lunak

5.1.3 Implementasi Perangkat Lunak

Perangkat lunak yang telah dibuat adalah perangkat lunak yang sesuai dengan perancangan pada Bab 4. Kode perangkat lunak dilampirkan secara lengkap pada halaman Lampiran A.Perangkat lunak ini diimplementasi dengan menggunakan perangkat lunak Eclipse IDE 2018-09. Berikut penjelasan setiap algoritma yang sudah di desain pada masing-masing kelas yang diperlihatkan pada subbab 4.6, diantaranya:

1. Kelas Node
   Bedasarkan hasil perancangan yang telah dilakukan, kelas Node akan memiliki algoritma pada metode *run*. Metode ini bekerja dengan menunggu pesan masuk dan melaksanakan tugasnya sesuai pesan yang diminta dan mengembalikan hasilnya dalam bentuk pesan ke *base-station*.

Listing 5.1: Metode run pada kelas Node

```
1     public void run() throws Exception {
2         NODE = allAddress[2];
3         final AT86RF231 radio = RadioInit.initRadio();
4         radio.setChannel(COMMON_CHANNEL);
5         radio.setPANId(COMMON_PANID);
6         radio.setShortAddress(NODE);
7
8         Thread t = new Thread() {
9             public void run() {
10                Sensing st = new Sensing();
11                while (true) {
12                    Frame f = null;
13                    try {
14                        f = new Frame();
15                        radio.setState(AT86RF231.STATE_RX_AACK_ON);
16                        radio.waitForFrame(f);
17                    } catch (Exception e) {
```

```
18                  }
19                  if (f != null) {
20                      byte[] dg = f.getPayload();
21                      //mengubah pesan dari byte menjadi string
22                      String str = new String(dg, 0, dg.length);
23
24                      String name = "";
25                      String order = "";
26                      String message = "";
27                      boolean done = false;
28                      while (!done) {
29                          try {
30                              if (!str.equalsIgnoreCase("sense")) {
31                                  name = str.substring(0, 4);
32                                  order = str.substring(5);
33                                  if (name.equalsIgnoreCase(Integer.toHexString(NODE))) {
34                                      Thread.sleep(88);
35                                      message = order;
36                                  } else {
37                                      message = st.run();
38                                  }
39                              } else {
40                                  message = st.run();
41                              }
42                              // ////////////////////////////////////////////////////////////////////
43                              Frame frame = new Frame(Frame.TYPE_DATA | Frame.ACK_REQUEST | Frame.DST_ADDR_16
44                                      | Frame.INTRA_PAN | Frame.SRC_ADDR_16);
45                              frame.setSrcAddr(NODE);
46                              frame.setSrcPanId(COMMON_PANID);
47                              frame.setDestAddr(BASE);
48                              frame.setDestPanId(COMMON_PANID);
49                              radio.setState(AT86RF231.STATE_TX_ARET_ON);
50                              frame.setPayload(message.getBytes()); // penting
51                              radio.transmitFrame(frame);
52                              // ////////////////////////////////////////////////////////////////////
53                              done = true;
54                          } catch (Exception e) {
55                          }
56                      }
57                      if (name.equalsIgnoreCase(Integer.toHexString(NODE))) {
58                          try {
59                              if (order.equalsIgnoreCase("stop")) {
60                                  CPUHelper.setPowerState(CPUConstants.V_POWER_STATE_OFF, Long.MAX_VALUE);
61                              } else if (order.equalsIgnoreCase("restart")) {
62                                  CPUHelper.setPowerState(CPUConstants.V_POWER_STATE_OFF, 8000);
63                              }
64                          } catch (Exception e) {
65                          }
66                      }
67                  }
68              }
69          }
70      };
71      t.start();
72  }
```

Pada Listing 5.1, baris 12 sampai 18 memiliki fungsi untuk menunggu *frame* yang dikirimkan dari *base-station*(Listing 5.2) dengan jaringan radio dan mengembalikan ACK. Pada baris 30 sampai 41 memiliki fungsi untuk memeriksa pesan dari *frame* yang diterima untuk *sensing, restart*, atau *turn off*. Pada baris 43 sampai 51 memiliki fungsi untuk menyiapkan pesan yang dimasukkan ke dalam *frame* beserta alamat asal dan tujuan sesuai dengan perintah lalu mengirimnya sesuai alamat tujuannya(*base-station*, Listing 5.3) melalui jaringan radio. Jika perintah sensing maka pesan akan berupa hasil *sensing*, jika *restart* atau *turn off* maka pesan yang dikembalikan berupa pemberitahuan bahwa node telah *restart* atau *turn off*. Pada baris 57 sampai 63 memiliki fungsi untuk memeriksa apakah nama sesuai dengan pesan dari *base-station* untuk *restart* atau *turn off*.

2. Kelas BaseStation
   Bedasarkan hasil perancangan yang telah dilakukan, kelas BaseStation akan memiliki algoritma pada metode *send* dan *receive*. Metode *send* bekerja dengan menerima perintah dari kelas GUI yang menentukan pesan yang akan dikirim kepada node sensor dengan cara *broadcast*.

Listing 5.2: BaseStation.java

```
1   private void send() {
2       Thread t = new Thread() {
3           public void run() {
4               String msg = "sense";
5               int choice;
6               while (true) {
7                   try {
8                       choice = BaseStation.usart.read();
9                       if (choice == 50) {
10                          msg = "sense";
11                      } else if (choice < 50) {
```

```
12                          msg = Integer.toHexString(allAddress[choice]) + "#" + "stop";
13                      } else if (choice > 99) {
14                          msg = Integer.toHexString(allAddress[choice - 100]) + "#" + "restart";
15                      }
16                      String message = msg;
17                      // ////////////////////////////////////////////////////////////////////
18                      Frame frame = new Frame(Frame.TYPE_DATA | Frame.ACK_REQUEST | Frame.DST_ADDR_16
19                              | Frame.INTRA_PAN | Frame.SRC_ADDR_16);
20                      // mengisi source address dan destination address pada frame
21                      frame.setSrcAddr(BASE);
22                      frame.setSrcPanId(COMMON_PANID);
23                      frame.setDestAddr(BROADCAST);
24                      frame.setDestPanId(COMMON_PANID);
25                      // state radio untuk mengirim pesan
26                      radio.setState(AT86RF231.STATE_TX_ARET_ON);
27                      frame.setPayload(message.getBytes()); // penting untuk mengantar paket ke frame
28                      // radio mengantar frame menuju alamat broadcast
29                      radio.transmitFrame(frame);
30                      // ////////////////////////////////////////////////////////////////////
31                  } catch (Exception e) {
32                  }
33              }
34          }
35      };
36      t.start();
37  }
```

Pada Listing 5.2, baris 8 sampai 15 memiliki fungsi menerima perintah dari perangkat lunak yang akan diproses untuk menentukan pesan yang akan dikirimkan agar sesuai dengan perintah. Pada baris 16 sampai 29 memiliki fungsi untuk menyiapkan *frame* yang berisikan pesan, alamat asal, dan alamat tujuan sesuai dengan perintah lalu mengirimnya sesuai alamat tujuannya(node sensor, Listing 5.1) melalui jaringan radio.

Metode *receive* digunakan untuk menerima pesan dari node sensor untuk diproses hasilnya dan dikirim ke perangkat lunak.

Listing 5.3: Metode receive pada kelas BaseStation

```
1   private void receive() {
2       Thread t = new Thread() {
3           public void run() {
4               String toPC, str, hex_addr;
5               long timeRaw;
6               byte[] dg;
7               while (true) {
8                   Frame f = null;
9                   try {
10                      f = new Frame();
11                      // state radio menerima pesan
12                      radio.setState(AT86RF231.STATE_RX_AACK_ON);
13                      radio.waitForFrame(f, 0);
14                      if (f != null) {
15                          timeRaw = Time.currentTimeMillis();
16                          dg = f.getPayload();
17                          //mengubah pesan dari byte menjadi string
18                          str = new String(dg, 0, dg.length);
19                          //mengambil nilai alamat dan diubah menjadi bentuk string untuk dijadikan nama
20                          hex_addr = Integer.toHexString((int) f.getSrcAddr());
21
22                          toPC = hex_addr + "#" + str + "#" + timeRaw + "%";
23                          out.write(toPC.getBytes(), 0, toPC.length());
24                      }
25                  } catch (Exception e) {
26                  }
27              }
28          }
29      };
30      t.start();
31  }
```

Pada Listing 5.3, baris 8 sampai 13 memiliki fungsi untuk menunggu *frame* yang dikirimkan dari *node sensor*(Listing 5.1) dengan jaringan radio dan mengembalikan ACK. Pada baris 14 sampai 24, *frame* yang sudah diterima akan diproses untuk dikirimkan lagi ke perangkat lunak.

3. Kelas GUI
   Bedasarkan hasil perancangan yang telah dilakukan, kelas GUI akan memiliki algoritma pada metode btnStartPressed yang digunakan untuk mengirim perintah ke *base-station* dan menerima hasilnya untuk ditampilkan di antarmuka.

Listing 5.4: Metode btnStartPressed pada kelas GUI

```
1   private static void btnStartPressed(BufferedInputStream in,DataConnection conn) {
```

```
        btnStart.setEnabled(true);
        btnStart.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                Thread t = new Thread() {
                    public void run() {
                        number = 50;
                        btnStop.setEnabled(true);
                        btnRestart.setEnabled(true);
                        btnStart.setEnabled(false);
                        boolean[] isON = new boolean[state.length];
                        int[] countOFF = new int[state.length];
                        for (int i = 0; i < state.length; i++) {
                            countOFF[i] = 8;
                        }
                        do {
                            for (int i = 0; i < state.length; i++) {
                                isON[i] = false;
                            }
                            try {

                                conn.flush();
                                // number lebih kecil dari 50 artinya turn off
                                // number lebih besar dari 99 artinya restart
                                // number = 50 artinya sense
                                conn.write(number);
                                byte[] buffer = new byte[1024];
                                Thread.sleep(5000);
                                while (in.available() > 0) {
                                    in.read(buffer);
                                    String s = new String(buffer);
                                    conn.flush();

                                    String[] subNode = s.split("%");
                                    for (int j = 0; j < subNode.length - 1; j++) {
                                        String[] subStr = subNode[j].split("#");
                                        for (int i = 0; i < state.length; i++) {
                                            String result;
                                            Date timeRaw;
                                            String clock;
                                            if (subStr[0].equalsIgnoreCase(name[i])) {
                                                // untuk menangani duplikat pesan
                                                if (isON[i]) {
                                                    break;
                                                }
                                                if (!subStr[1].equalsIgnoreCase("restart")
                                                        && !subStr[1].equalsIgnoreCase("stop")) {
                                                    isON[i] = true;
                                                    countOFF[i] = 1;
                                                    // menyimpan nilai sensing dalam float yang awalnya berupa string
                                                    float temperature = Float.parseFloat(subStr[1]);
                                                    float humidity = Float.parseFloat(subStr[2]);
                                                    // satuan awal barometer adalah kpa dan diubah menjadi milibar
                                                    float barometer = Float.parseFloat(subStr[3]) / 100;

                                                    // mengatur waktu
                                                    timeRaw = new Date(Long.parseLong(subStr[4]));
                                                    String timeForFile = new SimpleDateFormat("dd-MMM-yyyy")
                                                            .format(timeRaw);
                                                    clock = new SimpleDateFormat("kk:mm:ss").format(timeRaw);

                                                    // menyimpan nilai log pada file
                                                    File file = new File("src/log/" + timeForFile + ".txt");
                                                    FileWriter fw = new FileWriter(file, true);
                                                    BufferedWriter bw = new BufferedWriter(fw);
                                                    PrintWriter save = new PrintWriter(bw);

                                                    state[i] = "on";
                                                    dataTable[i][2] = state[i];
                                                    dataTable[i][3] = String.format("%.0f", temperature);
                                                    dataTable[i][4] = String.format("%.2f", humidity);
                                                    dataTable[i][5] = String.format("%.4f", barometer);

                                                    result = "[␣" + clock + "␣]" + subStr[0].toUpperCase() + "␣:"
                                                            + String.format("%.0f", temperature) + "␣[ÂřC],␣"
                                                            + String.format("%.2f", humidity) + "␣RH,␣"
                                                            + String.format("%.4f", barometer) + "␣bar" + "\n";

                                                    log.append(result);
                                                    save.println(result);
                                                    save.close();
                                                    break;
                                                } else {
                                                    isON[i] = true;
                                                    timeRaw = new Date(Long.parseLong(subStr[2]));
                                                    clock = new SimpleDateFormat("kk:mm:ss").format(timeRaw);
                                                    if (subStr[1].equalsIgnoreCase("restart")) {
                                                        subStr[1] = "restarting";
                                                    } else {
                                                        subStr[1] = "stopping";
                                                    }
                                                    result = "[␣" + clock + "␣]" + subStr[0].toUpperCase() + "␣is␣"
                                                            + subStr[1] + "\n";
                                                    log.append(result);
                                                    state[i] = "off";
                                                    dataTable[i][2] = state[i];
                                                    countOFF[i] = 4;
                                                    break;
                                                }
                                            }
                                        }
```

```
101                                     }
102                                 }
103                             }
104                             for (int i = 0; i < isON.length; i++) {
105                                 if (!isON[i]) {
106                                     if (countOFF[i] < 4) {
107                                         countOFF[i]++;
108                                     } else if (countOFF[i] >= 4 && countOFF[i] < 8) {
109                                         // mati
110                                         System.out.println("mati" + countOFF[i] + i);
111                                         state[i] = "off";
112                                         dataTable[i][2] = state[i];
113                                         countOFF[i]++;
114                                     } else {
115                                         state[i] = "none";
116                                         dataTable[i][2] = state[i];
117                                     }
118                                 }
119                             }
120                             // melakukan update pada gambar dan tabel
121                             table.repaint();
122                             panelDraw.repaint();
123                         } catch (Exception ex) {
124                         }
125                     } while (true);
126                 }
127             };
128             t.start();
129         }
130     });
131 }
```

Pada Listing 5.4, Metode ini bekerja diawali dengan mengirim perintah kepada *base-station* untuk melakukan *sensing* (baris 22 sampai 28) lalu menunggu beberapa saat dan pada baris 29 sampai 34, perangkat lunak menerima pesan dari *base-station*. Pesan akan mulai diproses dan ditampilkan pada antarmuka dari baris 35 sampai 103.

5.2 Pengujian

Pada bagian ini, pengujian terhadap perangkat lunak yang dibangun dapat dibagi menjadi 2, yaitu pengujian fungsional dan pengujian eksperimental.

5.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan secara Black Box Testing, dimana cara pengujian dilakukan dengan hanya menjalankan atau mengeksekusi aplikasi kemudian diamati apakah hasil sesuai dengan proses yang diinginkan. Hasil pengujian fungsional dapat dilihat pada tabel 5.1.

Kesimpulan dari pengujian fungsional ini adalah perangkat lunak yang dibangun sudah sesuai dengan yang diinginkan dan bekerja dengan baik.

5.2.2 Pengujian Eksperimental

Pengujian eksperimental dilakukan di *Rooftop* Gedung 10 Universitas Katholik Parahyangan sebagai ruang terbuka dan di dalam ruang tertutup. Arsitektur WSN yang digunakan saat pengujian adalah flat dengan *single-hop*.Pengujian dilakukan dengan menggunakan 6 node sensor yang terdiri dari 1 node sensor sebagai *base station* dan 5 node sensor untuk melakukan *sensing*.

No	Langkah Pengujian	Hasil yang Diharapkan	Hasil Pengujian	Status
1	Menjalankan perangkat lunak	Perangkat lunak akan menampilkan antarmuka yang sudah dibuat dan jumlah node sensor sesuai dengan jumlah nama pada *file text*	Perangkat lunak menampilkan antarmuka dan jumlah node sensor yang sesuai dengan *file text*	OK
2	Menekan tombol *start*	Perangkat lunak akan menampilkan node sensor yang menyala beserta pesan-pesannya ditampilkan pada tabel dan *log*	Perangkat lunak menampilkan node sensor yang menyala beserta pesan-pesannya ditampilkan pada tabel dan *log*	OK
3	Memilih node sensor dan menekan tombol *restart*	Perangkat lunak akan menampilkan node sensor yang mati sesaat atau pada *log* akan diperlihatkan pesan bahwa node tersebut sedang *restart*	Perangkat lunak menampilkan node sensor yang mati sesaat atau pada *log* diperlihatkan pesan bahwa node tersebut sedang *restart*	OK
4	Memilih node sensor dan menekan tombol *turn off*	Perangkat lunak akan menampilkan node sensor yang mati atau pada *log* akan diperlihatkan pesan bahwa node tersebut telah berhenti	Perangkat lunak menampilkan node sensor yang mati atau pada *log* diperlihatkan pesan bahwa node tersebut telah berhenti	OK

Tabel 5.1: Tabel Pengujian Fungsional Perangkat Lunak

Tabel 5.2: Tabel Pengujian Eksperimen *Start* dengan 5 Node Sensor

No	Berhasil
1	Ya
2	Ya
3	Tidak
4	Ya
5	Ya
6	Tidak
7	Ya
8	Ya
9	Ya
10	Tidak

Tabel 5.3: Tabel Pengujian Eksperimen Menambah Node Baru pada Perangkat Lunak sedang Bekerja

No	Berhasil
1	Ya
2	Ya
3	Ya
4	Ya
5	Ya
6	Ya
7	Ya
8	Ya
9	Tidak
10	Tidak

Tabel 5.4: Tabel Pengujian Eksperimen *Restart*

No	Berhasil
1	Ya
2	Ya
3	Tidak
4	Tidak
5	Ya
6	Ya
7	Ya
8	Ya
9	Tidak
10	Tidak

Tabel 5.5: Tabel Pengujian Eksperimen *Turn Off*

No	Berhasil
1	Ya
2	Ya
3	Ya
4	Tidak
5	Ya
6	Ya
7	Ya
8	Ya
9	Tidak
10	Ya

Tabel 5.6: Tabel Pengujian Eksperimen Melepas dan Memasang Kembali Baterai

No	Berhasil
1	Ya
2	Ya
3	Ya
4	Ya
5	Ya
6	Ya
7	Ya
8	Ya
9	Ya
10	Ya

Pada tabel 5.2, pengujian difokuskan pada apakah ketika pengguna menekan tombol *start* maka perangkat lunak menampilkan node sensor yang menyala sesuai dengan jumlah node sensor yang pada kenyataanya memang menyala. Adanya kegagalan karena hanya 3 atau 2 dari 5 node sensor yang dapat dijangkau oleh pesan *base-station* sehingga terjadi *loss* pada node lainnya atau kondisi lingkungan yang menganggu jaringan *radio* node.

Pada tabel 5.2 terjadi 2 kali kegagalan karena pada percobaan 9 dan 10, node sensor yang baru akan ditambahkan berada di luar jangkauan *base-station*. Pada tabel 5.4 terjadi 4 kegagalan dimana ada 2 kali berturut-turut karena jaringan node sensor yang tidak *reliable* sehingga membuat pesan yang seharusnya dikirimkan untuk melakukan *restart* tidak diterima.

Pada Tabel 5.5 mengalami kejadian yang sama dengan tabel 5.4,yaitu jaringan node sensor yang tidak *reliable* sehingga membuat pesan yang seharusnya dikirimkan untuk melakukan *turn off* tidak diterima. Pada tabel 5.6, semua pengujian untuk memeriksa apakah node akan menyala lagi ketika baterai dicabut dan dipasang kembali berhasil seluruhnya.

Dapat dilihat dari eksperimen yang telah dilakukan terdapat percobaan yang berhasil dan gagal. Hal tersebut disebabkan oleh adanya faktor yang mempengaruhi kinerja dari node-node sensor terutama pesan atau paket yang *loss*. Pesan yang *loss* dikarenakan oleh kondisi lingkungan yang berbeda yang menyebabkan terganggunya jaringan pengiriman data atau paket. Keseluruhan hasil dari pengujian ini memperlihatkan bahwa perangkat lunak sudah bekerja dengan baik dalam hal pemantauan.

5.3 Masalah yang Dihadapi pada Saat Implementasi

Berikut adalah beberapa masalah yang dihadapi pada saat implementasi:

1. Keterbatasan jumlah alat yang digunakan. Karena node sensor yang digunakan terbatas jadi harus digunakan bersama dengan mahasiswa lain yang menggunakan node sensor juga untuk skripsi mereka.

2. Node sensor dengan *base-station* terjadi *loss*. Hal ini terjadi sesekali dan tidak menentu.

3. Kode program tiap node sensor dan *base-station* perlu diunggah satu per satu sehingga menghabiskan waktu dan perlu teliti karena setiap node juga diberi alamat yang berbeda-beda saat diunggah.

4. *Driver* yang ada pada Preon32 tidak selalu bekerja karena node sensor dari Preon32 tidak terpasang perangkat keras yang memiliki *driver*-nya.