

**SISTEM *MONITORING PARAMETER FISIK AIR KOLAM IKAN*
MENGGUNAKAN JARINGAN SENSOR NIRKABEL BERBASIS
PROTOKOL LORA**

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Delarta Tok Adin
NIM: 155150200111206



PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2019

PENGESAHAN

Sistem *Monitoring* Parameter Fisik Air Kolam Ikan Menggunakan Jaringan Sensor Nirkabel Berbasis Protokol LoRa

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Delarta Tok Adin
NIM: 155150200111206

Skripsi ini telah diuji dan dinyatakan lulus pada
4 Juli 2019

Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I

Adhitya Bhawiyuga, S.Kom., M.Sc.
NIP: 19890720 201803 1 002

Dosen Pembimbing II

Widhi Yahya., S.Kom., M.T.
NIK: 201607 891121 1 001

Mengetahui

Ketua Jurusan Teknik Informatika



PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disisipi dalam naskah ini dan disebutkan dalam daftar referensi.

Apabila ternyata di dalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 8 Juli 2019



Delarta Tok Adin

NIM: 155150200111206

PRAKATA

Puji syukur penulis panjatkan atas kehadiran Allah SWT yang telah memberikan rahmat dan hidayat-Nya sehingga skripsi dengan judul “Sistem Monitoring Parameter Fisik Air Kolam Ikan Menggunakan Jaringan Sensor Nirkabel Berbasis Protokol LoRa” dapat terselesaikan.

Penulisan skripsi tidak hanya serta merta usaha penulis sendiri, tetapi juga dukungan dari berbagai pihak yang telah membantu kelancaran penulisan skripsi. Oleh karena itu, penulis ingin menyampaikan rasa terimakasih dan hormat kepada:

1. Bapak Adhitya Bhawiyuga, S.Kom., M.Sc dan bapak Widhi Yahya., S.Kom., M.T selaku dosen pembimbing skripsi yang telah dengan sabar dan tekun dalam membimbing jalannya proses skripsi.
2. Bapak dan ibu sekeluarga yang telah membesar dan memberi amanah kepada penulis untuk menyelesaikan skripsi dan memastikan semua kebutuhan penulis terpenuhi selama proses skripsi.
3. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D selaku Ketua Jurusan Teknik Informatika.
4. Anggota “Perpus Team” yaitu: Septian Dwi Cahyo, Hilmy Khairi Idris, Fera Fanesya, Prita Nur Rizky Faridiani, Salma Hanifah, Katherine Ivana Ruslim dan Meidita Famirah yang telah memberikan dukungan moral dan mental kepada penulis.
5. Desy Diandra Bestari, Rafiqah Majidah, Tria Melia S, dan terutama Fitri Febriyani yang telah membantu pengujian yang dilakukan dalam penelitian ini.
6. Teman teman KBJ yang telah memberikan motivasi untuk menyelesaikan skripsi.
7. Seluruh dosen dan civitas Fakultas Ilmu Komputer yang telah membagi ilmunya kepada penulis.

Penulis menyadari dalam penyusunan skripsi ini masih terdapat ketidak sempurnaan maka dari itu penulis mengharapkan kritik dan saran yang membangun demi perbaikan skripsi ini. Semoga dengan skripsi ini dapat memberikan bagi penulis, pihak yang terlibat dan semua yang membacanya.

Malang, 8 Juli 2019

Penulis

delartatok@student.ub.ac.id

ABSTRAK

Delarta Tok Adin, Sistem Monitoring Parameter Fisik Air Kolam Ikan Menggunakan Jaringan Sensor Nirkabel Berbasis Protokol LoRa.

Pembimbing: Adhitya Bhawiyuga, S.Kom., M.Sc. dan Widhi Yahya, S.Kom., M.T.

Wireless Sensor Network (WSN) adalah jaringan nirkabel yang terdiri dari perangkat otonom yang terdistribusi menggunakan sensor untuk memantau kondisi lingkungan. Pemantauan kondisi perairan dengan menggunakan teknologi WSN sebelumnya telah dilakukan menggunakan protokol ZigBee dan berhasil memantau parameter suhu, oksigen terlarut dan tingkat keasaman. ZigBee memiliki kekurangan yaitu pada jangkauan yang relatif pendek yang hanya menjangkau kurang dari 150 meter, sehingga tidak cocok untuk pemantauan area yang luas. Salah satu protokol dengan jangkauan yang luas adalah LoRa. LoRa adalah salah satu protokol *Low Power Wide Area Wireless Network (LPWAN)* untuk aplikasi *Internet of Things (IoT)*. Penelitian ini berfokus pada mekanisme aliran data dari *sensor node* hingga menuju komponen visualisasi data. Arsitektur pada penelitian ini terdiri dari *sensor node*, *gateway*, *cloud server* dan komponen visualisasi data. Perangkat sensor yang digunakan yaitu sensor suhu, keasaman, kekeruhan, dan oksigen terlarut. Data yang diakuisisi dari *sensor node* dijalankan mekanisme *packing* dalam bentuk *struct*. Data *struct* kemudian dikirimkan menggunakan LoRa menuju *gateway*. Di dalam *gateway* dijalankan mekanisme untuk mengkonversi tipe data *struct* menjadi JSON. Data yang telah dikonversi menjadi JSON berikutnya akan dikirim oleh *gateway* menuju *cloud server* menggunakan protokol HTTP dan menampilkan hasil dalam bentuk *web application*. Pengujian fungsionalitas dilakukan agar sistem berjalan sebagaimana mestinya. Pengujian kinerja dilakukan dengan parameter *successful rate* dengan hasil yang didapatkan kinerja paling baik pada jarak 400 meter pada interval waktu 60 detik dengan successful rate 95%. Berdasarkan hasil tersebut penelitian ini dapat menjadi solusi atas masalah pemantauan kondisi kualitas perairan khususnya dalam hal jangkauan pemantauan yang luas.

Kata kunci: Wireless Sensor Network, LoRa, Zigbee, JSON, Struct

ABSTRACT

Delarta Tok Adin, Pond Water Monitoring System Using Wireless Sensor Network Based On LoRa Protocol

Supervisors: Adhitya Bhawiyuga, S.Kom., M.Sc. and Widhi Yahya, S.Kom., M.T.

Wireless Sensor Network (WSN) is a wireless network consist of individual devices distributed throughout the monitored area. Water monitoring system has been done using ZigBee protocol monitored temperature, dissolved oxygen, and acidity. ZigBee has disadvantages in range aspect so it's not suitable for monitoring in wide area. One of the many protocol that has wide network range is LoRa. LoRa is Low Power Wide Area Wireless Network (LPWAN) for IoT applications. This Research focused on the mechanism of data flow from sensor node to data visualization component. Architecture in this research consist of sensor node, gateway, cloud server and data visualization component. Sensor used are temperature sensor, acidity sensor, turbidity sensor and dissolved oxygen sensor. Data that has been acquired from sensor node then converted into struct. Struct data then sent to gateway using LoRa. Gateway then do the conversion from struct to JSON. Data that has been converted to JSON then will be sent to cloud server to be stored in the database using HTTP protocol. Functionality testing is applied to ensure the system work as it should be. Performance testing is done with successful rate parameter with range and time interval as the variables. Based on that testing this research can be solution to the water monitoring problem especially at the long range monitoring area.

Keywords: Wireless Sensor Network, LoRa, Zigbee, JSON, Struct

DAFTAR ISI

PENGESAHANii
PERNYATAAN ORISINALITAS	iii
PRAKATA	iv
ABSTRAK.....	v
<i>ABSTRACT.</i>	vi
DAFTAR ISI.....	vii
DAFTAR TABEL.....	x
DAFTAR GAMBAR	xii
BAB 1 PENDAHULUAN	1
1.1 Latar belakang	1
1.2 Rumusan masalah.....	2
1.3 Tujuan	2
1.4 Manfaat.....	3
1.5 Batasan masalah.....	3
1.6 Sistematika pembahasan.....	3
BAB 2 LANDASAN KEPUSTAKAAN	5
2.1 Kajian Pustaka.....	5
2.2 Dasar Teori	6
2.2.1 Akuakultur.....	6
2.2.2 Sensor	6
2.2.3 Jaringan Sensor Nirkabel	10
2.2.4 Arduino	12
2.2.5 Raspberry Pi	13
2.2.6 LoRa	14
2.2.7 Structure (struct).....	16
2.2.8 JSON.....	17
BAB 3 METODOLOGI.....	18
3.1 Studi Kepustakaan	18
3.2 Analisis Kebutuhan	19

3.3 Perancangan Sistem.....	19
3.4 Implementasi Sistem	21
3.5 Pengujian Sistem	22
3.6 Pengambilan Kesimpulan dan Saran	22
BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN SISTEM	23
4.1 Analisis Kebutuhan Sistem	23
4.1.1 Kebutuhan Fungsional Sistem.....	23
4.1.2 Kebutuhan Data	23
4.1.3 Kebutuhan Non Fungsional.....	24
4.2 Perancangan Sistem.....	25
4.2.1 Gambaran Umum Sistem.....	25
4.2.2 Perancangan Alur Sistem.....	26
4.2.3 Perancangan <i>Sensor node</i>	27
4.2.4 Perancangan <i>Gateway</i>	32
4.2.5 Perancangan Komponen Visualisasi Data	35
4.2.6 Perancangan Data Sensor	36
4.2.7 Perancangan Pengujian	38
BAB 5 implementasi sistem	41
5.1 Implementasi Sistem	41
5.1.1 Implementasi <i>Sensor node</i>	41
5.1.2 Implementasi <i>Gateway</i>	46
5.1.3 Implementasi Komponen Visualisasi Data	52
BAB 6 PENGUJIAN SISTEM	54
6.1 Pengujian Fungsional	54
6.1.1 Pengujian Kode KF-01.....	54
6.1.2 Pengujian Kode KF-02.....	55
6.1.3 Pengujian Kode KF-03.....	56
6.1.4 Pengujian Kode KF-04.....	57
6.1.5 Pengujian Kode KF-05.....	58
6.1.6 Pengujian Kode KF-06.....	59
6.1.7 Pengujian Kode KF-07	62
6.1.8 Pengujian Kode KF-08.....	63

6.2 Pengujian Kinerja	66
6.2.1 Pengujian Jarak 50 meter	66
6.2.2 Pengujian Jarak 100 meter	67
6.2.3 Pengujian Jarak 200 meter	68
6.2.4 Pengujian Jarak 400 meter	68
BAB 7 Kesimpulan dan saran	70
7.1 Kesimpulan	70
7.2 Saran	71
Daftar Pustaka	72
LAMPIRAN A	74
A.1 Kode Program	74

DAFTAR TABEL

Tabel 2.1 Komponen sensor oksigen terlarut.....	7
Tabel 2.2 Spesifikasi <i>analog ph meter</i>	8
Tabel 2.3 Spesifikasi sensor <i>analog turbidity</i>	9
Tabel 2.4 Spesifikasi sensor <i>digital waterproof temperature</i>	10
Tabel 2.5 Spesifikasi Arduino Nano.....	13
Tabel 2.6 Spesifikasi Raspberry Pi 3 Model B+	14
Tabel 2.7 Spesifikasi Modul RFM95/96/97/98(W).....	16
Tabel 2.8 Format struct	17
Tabel 2.9 Format JSON	17
Tabel 4.1 Kebutuhan Fungsional Sistem.....	23
Tabel 4.2 Kebutuhan data.....	24
Tabel 4.3 Kebutuhan perangkat keras.....	24
Tabel 4.4 Kebutuhan perangkat lunak	25
Tabel 4.5 <i>Pinout</i> Modul HopeRF-RFM95 dengan Arduino Nano	28
Tabel 4.6 <i>Pinout</i> modul sensor keasaman dengan Arduino Nano.....	29
Tabel 4.7 <i>Pinout</i> modul sensor oksigen terlarut dengan Arduino Nano.....	30
Tabel 4.8 <i>Pinout</i> modul sensor kekeruhan dengan Arduino Nano	30
Tabel 4.9 <i>Pinout</i> sensor suhu (<i>temperature</i>) dengan Arduino Nano	31
Tabel 4.10 <i>Pinout</i> modul LoRa HopeRF-RFM95 dengan Raspberry Pi	33
Tabel 4.11 Perancangan data <i>struct</i>	37
Tabel 4.12 Perancangan data JSON.....	37
Tabel 4.13 Skenario Pengujian Fungsional	38
Tabel 5.1 <i>Pseudocode Sensor node</i>	42
Tabel 5.2 Implementasi data <i>struct</i>	43
Tabel 5.3 Implementasi sensor oksigen terlarut.....	43
Tabel 5.4 Implementasi sensor keasaman	44
Tabel 5.5 Implementasi sensor kekeruhan.....	44
Tabel 5.6 Implementasi sensor suhu.....	45
Tabel 5.7 Implementasi modul LoRa	46
Tabel 5.8 <i>Pseudocode Gateway</i>	48

Tabel 5.9 Implementasi format JSON.....	49
Tabel 5.10 Implementasi modul LoRa pada <i>gateway</i>	49
Tabel 5.11 Implementasi unpacking <i>data sensor</i>	50
Tabel 5.12 Implementasi dan pengiriman data melalui protokol HTTP	51
Tabel 5.13 Implementasi kode program <i>request REST API</i>	52
Tabel 6.1 Deskripsi Pengujian Kode KF-01.....	54
Tabel 6.2 Deskripsi Pengujian Kode KF-02.....	55
Tabel 6.3 Deskripsi Pengujian Kode KF-03.....	56
Tabel 6.4 Deskripsi Pengujian Kode KF-04.....	57
Tabel 6.5 Deskripsi Pengujian Kode KF-05.....	58
Tabel 6.6 Deskripsi Pengujian Kode KF-06.....	59
Tabel 6.7 Deskripsi Pengujian Kode KF-07.....	62
Tabel 6.8 Deskripsi Pengujian KF-08	63

DAFTAR GAMBAR

Gambar 2.1 Sensor <i>analog dissolved oxygen</i>	7
Gambar 2.2 Sensor <i>analog ph meter</i>	8
Gambar 2.3 Sensor <i>analog turbidity</i>	9
Gambar 2.4 Sensor <i>digital waterproof temperature</i>	10
Gambar 2.5 Arsitektur WSN.....	11
Gambar 2.6 <i>WSN Protocol Stack</i>	11
Gambar 2.7 Tampilan Arduino IDE.....	12
Gambar 2.8 Arduino Nano.....	12
Gambar 2.9 Raspberry Pi 3 Model B+	13
Gambar 2.10 Arsitektur LoRa.....	15
Gambar 2.11 Modul RFM95/96/97/98(W).....	16
Gambar 3.1 Diagram Alur Metodologi Penelitian.....	18
Gambar 4.1 Rancangan Arsitektur Sistem.....	26
Gambar 4.2 <i>Flowchart</i> Perancangan Alur Sistem	27
Gambar 4.3 Perancangan <i>Sensor node</i>	28
Gambar 4.4 <i>Flowchart</i> alur kerja <i>sensor node</i>	32
Gambar 4.5 Arsitektur <i>Gateway</i>	33
Gambar 4.6 <i>Flowchart</i> Alur kerja <i>Gateway</i>	34
Gambar 4.7 Rancangan aplikasi	35
Gambar 4.8 alur kerja komponen visualisasi data	36
Gambar 5.1 Implementasi <i>Sensor node</i>	41
Gambar 5.2 Implementasi Perangkat Keras <i>Gateway</i>	47
Gambar 5.3 Implementasi <i>Web Application</i>	53
Gambar 6.1 Hasil Pengujian Sensor <i>Dissolved Oxygen</i>	55
Gambar 6.2 Hasil Pengujian Sensor <i>Turbidity</i>	56
Gambar 6.3 Hasil Pengujian Sensor <i>Ph Meter</i>	57
Gambar 6.4 Hasil Pengujian Sensor <i>Suhu</i>	58
Gambar 6.5 Hasil Pengujian Pengiriman Data dari <i>Sensor node</i>	59
Gambar 6.6 Hasil Pengujian Penerimaan Data pada <i>Gateway</i> dan Pengiriman Data ke Cloud	60

Gambar 6.7 Hasil Data Sensor <i>Dissolved Oxygen</i> diterima pada Cloud	60
Gambar 6.8 Hasil Data Sensor <i>Ph Meter</i> diterima pada Cloud.....	61
Gambar 6.9 Hasil Data Sensor <i>Turbidity</i> diterima pada Cloud	61
Gambar 6.10 Hasil Data Sensor Suhu diterima pada Cloud	61
Gambar 6.11 Hasil Pengujian <i>Request API Data Web Application</i>	63
Gambar 6.12 Data sensor oksigen terlarut pada <i>cloud server</i>	64
Gambar 6.13 Data sensor keasaman pada <i>cloud server</i>	65
Gambar 6.14 Data sensor kekeruhan pada <i>cloud server</i>	65
Gambar 6.15 Data sensor suhu pada <i>cloud server</i>	66
Gambar 6.16 Grafik hasil pengujian jarak 50 meter	67
Gambar 6.17 Grafik hasil pengujian jarak 100 meter	68
Gambar 6.18 Grafik hasil pengujian jarak 200 meter	68
Gambar 6.19 Grafik hasil pengujian jarak 400 meter	69

BAB 1 PENDAHULUAN

1.1 Latar belakang

Budidaya perikanan darat merupakan usaha pemeliharaan dan penangkapan ikan di perairan darat (Yani, 2007). Salah satu faktor yang memiliki peranan penting dalam usaha perikanan adalah faktor kondisi air yang baik. Faktor yang menentukan kondisi air meliputi kecerahan, suhu, tingkat keasaman dan oksigen terlarut (Ira, 2014). Saat ini prosedur pemantauan kondisi perairan masih belum efisien karena petani ikan masih menggunakan pemantauan secara langsung. Hal tersebut dilakukan berdasarkan pengalaman petani, sehingga membutuhkan banyak waktu untuk dapat melakukan pemantauan (Encinas, et al., 2017). Pemantauan secara langsung juga tidak efektif karena kondisi air yang berubah dengan cepat baik dari segi fisika maupun kimia (Bengraine & Marhaba, 2003). Hal itu berakibat air tidak memiliki akurasi pemantauan yang baik karena pemantauan tidak *realtime*. Agar kondisi perairan terpantau dengan baik dan memiliki akurasi tinggi maka dibutuhkan suatu sistem untuk memantau kondisi perairan secara *realtime*. Teknologi WSN dapat membantu terwujudnya pemantauan kondisi perairan secara *realtime*.

Wireless Sensor Network (WSN) adalah jaringan nirkabel yang terdiri dari perangkat otonom yang terdistribusi menggunakan sensor untuk memantau kondisi lingkungan (NI, 2016). WSN terdiri dari komponen – komponen berukuran kecil yang hemat daya dan sensor yang cerdas. Di dalam WSN terdapat *source* dan *sink* yang saling berkomunikasi. *Source* adalah entitas yang mengirimkan data (contoh: *sensor node*, *actuator node*) Sedangkan *sink* adalah entitas yang menerima dan mengumpulkan data (contoh: *gateway*) (Karl & Willig, 2005). WSN bekerja dengan cara *source* di pasang di beberapa lingkungan yang akan di teliti, lalu *source* node akan mengirimkan data hasil *sensing* menuju *sink* sehingga terwujud sebuah jaringan.

Pemantauan kondisi perairan dengan menggunakan teknologi WSN sebelumnya telah dilakukan oleh (Sung, et al., 2014) menggunakan protokol ZigBee dan berhasil memantau parameter suhu, Oksigen terlarut dan tingkat keasaman. Protokol ZigBee memiliki kelebihan antara lain keandalan yang tinggi, konsumsi daya yang rendah dan transmisi data yang cepat (Tomar, 2011). Selain kelebihan yang dimiliki, ZigBee juga memiliki kekurangan yaitu pada jangkauan yang relatif pendek (10 – 150 meter) (Digi International, 2007). Jangkauan yang pendek menjadi masalah jika area yang dipantau lebih dari 150 meter. Maka dari itu dibutuhkan protokol yang memiliki jangkauan lebih luas. Salah satu protokol dengan jangkauan yang luas yaitu LoRa (Lora Alliance, 2015).

LoRa adalah salah satu protokol *Low Power Wide Area Wireless Network (LPWAN)* untuk aplikasi *Internet of Things (IoT)*. Kelebihan dari LoRa antara lain adalah jangkauan yang luas (5km – 15km) dan konsumsi daya yang lebih rendah dari ZigBee (Pule, et al., 2017). LoRa efisien digunakan ketika *sensor node*

mempunyai daya yang terbatas dan *sensor node* tidak membutuhkan pengiriman data yang terlalu besar (Lora Alliance, 2015).

Tantangan dalam menggunakan LoRa adalah *datarates* yang rendah yaitu sebesar 50kbps (Lora Alliance, 2015), sedangkan dalam pemantauan kondisi air dibutuhkan data *sensing* berupa kadar oksigen, keasaman, kekeruhan dan suhu. Untuk mengatasi hal tersebut maka diperlukan suatu mekanisme *packing* data yang memiliki ukuran data kecil untuk pengiriman data sensor. Salah satu tipe data dengan karakteristik ukuran data kecil dan dapat menampung banyak tipe data adalah *struct*. *Struct* memungkinkan pengelompokan dari beberapa tipe data ke dalam satu tipe data sehingga cocok untuk pemantauan dengan banyak perangkat sensor.

Berdasarkan dari permasalahan yang disampaikan, penulis mengusulkan penelitian dengan judul “Sistem Monitoring Parameter Fisik Air Kolam Ikan Menggunakan Jaringan Sensor Nirkabel Berbasis Protokol LoRa”. Arsitektur pada penelitian ini terdiri dari *Sensor node*, *Gateway*, dan *Application*. *Sensor Node* adalah *node* yang digunakan untuk mendapatkan data dari sensor. Perangkat sensor yang digunakan yaitu sensor suhu, keasaman, kekeruhan, dan oksigen terlarut. Data yang diakuisisi dari *sensor node* lalu dijalankan mekanisme *packing* dalam bentuk *struct*. *Struct* digunakan karena dapat menggabungkan tipe data yang berbeda dan memiliki ukuran yang kecil. Data *struct* kemudian dikirimkan menggunakan LoRa menuju *gateway*. Di dalam *gateway* dijalankan mekanisme untuk mengkonversi tipe data *struct* menjadi JSON karena *cloud server* hanya menerima data berupa JSON. Data yang telah dikonversi menjadi JSON berikutnya akan dikirim oleh *gateway* menuju *cloud server* menggunakan protokol HTTP untuk disimpan dalam *database* dan menampilkan hasil dalam bentuk *web application*. Penelitian ini diharapkan menjadi solusi atas masalah pemantauan kondisi kualitas perairan khusunya dalam hal jangkauan pemantauan yang luas.

1.2 Rumusan masalah

1. Bagaimana arsitektur sistem monitoring paramater fisik air kolam ikan berbasis protokol LoRa?
2. Bagaimana representasi data yang digunakan dalam pengiriman data dari *sensor node* menuju *gateway*?
3. Bagaimana mekanisme pengiriman data dari *sensor node* menuju *gateway*?
4. Bagaimana mekanisme pengiriman data dari *gateway* menuju *cloud server*?
5. Bagaimana kinerja protokol LoRa dalam mengirimkan data *sensor node* menuju *gateway*?

1.3 Tujuan

Berdasarkan dari rumusan masalah diatas, tujuan penelitian yang didapatkan adalah sebagai berikut:

1. Mengetahui arsitektur yang cocok untuk digunakan pada sistem yang dibuat.
2. Mengetahui representasi data yang cocok untuk sistem yang dibuat.
3. Mengetahui mekanisme pengiriman data dari *sensor node* menuju *gateway*.
4. Mengetahui mekanisme pengiriman data dari *gateway* menuju *cloud server*.
5. Mengetahui performa protokol LoRa dalam mengirimkan data *sensor node* pada sistem monitoring parameter fisik air kolam ikan.

1.4 Manfaat

Manfaat dari penelitian ini adalah dihasilkannya sistem monitoring parameter fisik air kolam ikan yang nantinya akan digunakan petani ikan untuk mempermudah memonitor parameter fisik air kolam ikan jarak jauh.

1.5 Batasan masalah

1. Menggunakan protokol LoRa untuk transmisi data dari *sensor node* ke *gateway*.
2. Menggunakan sensor suhu, kelembaban, keasaman dan oksigen terlarut.
3. Pengujian yang dilakukan adalah pengujian fungsional dan pengujian kinerja.

1.6 Sistematika pembahasan

Sistematika pembahasan dalam penelitian ini terdiri atas tujuh bab dengan uraian sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini berisi tentang latar belakang masalah yang ada, rumusan masalah dari topik penelitian, batasan masalah, tujuan, manfaat, serta sistematika pembahasan.

BAB 2 LANDASAN KEPUSTAKAAN

Bab ini menguraikan kajian pustaka dari hasil studi literatur tentang penelitian terdahulu yang terkait topik dan teori-teori yang digunakan dalam pengembangan penelitian penerapan sistem monitoring kondisi air.

BAB 3 METODOLOGI

Bab ini berisi penjelasan langkah-langkah metode yang digunakan dalam menganalisis, merancang dan mengimplementasikan protokol LoRa dalam sistem monitoring parameter fisik air kolam.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN SISTEM

Pada bab ini berisikan analisa kebutuhan dan proses perancangan sistem di dalam mengimplementasi protokol LoRa untuk sistem monitoring parameter fisik kolam ikan.

BAB 5 IMPLEMENTASI SISTEM

Pada bab ini membahas implementasi sistem dalam pengiriman data sensor pada kolam ikan berbasis protokol LoRa dan mengirimkan data sensor menggunakan modul RFM9x hingga mendapat hasil pengiriman data sensor.

BAB 6 PENGUJIAN SISTEM

Bab ini membahas mengenai pengujian dan analisis terhadap hasil yang didapat dengan tujuan untuk mendapatkan suatu kesimpulan dari masing – masing pengujian.

BAB 7 PENUTUP

Pada bab ini berisikan kesimpulan yang telah diperoleh dari pembuatan maupun pengujian sistem monitoring parameter fisik air kolam ikan dan dapat digunakan sebagai saran untuk perkembangan sistem selanjutnya.

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab landasan kepustakaan ini berisi tentang kajian pustaka dan dasar teori yang berkaitan dengan penelitian yang sedang dilakukan. Kajian pustaka adalah penelitian sebelumnya yang berkaitan dengan penelitian ini, dalam kajian pustaka diberikan informasi mengenai penelitian sebelumnya yang terkait dengan topik penelitian yang diangkat.

2.1 Kajian Pustaka

Dalam penelitian ini didapatkan beberapa penelitian-penelitian sebelumnya yang terkait dengan permasalahan yang ada. Untuk lebih jelasnya, kajian pustaka dapat dilihat pada Tabel 2.1.

Tabel 2.1 Kajian Pustaka

No	Judul	Metode	Hasil
1.	<i>Remote fish aquaculture monitoring system based on wireless transmission technology</i> (Sung, et al., 2014)	Zigbee Wireless Transmission	Sistem memiliki fungsi kontrol untuk memonitor status tambak ikan.
2.	<i>Design and deployment of wireless sensor networks for aquaculture monitoring and control based on virtual instruments</i> (Simbeye, et al., 2014)	ZigBee Wireless Communication Modules	Penelitian ini berhasil mendesain dan Jaringan sensor nirkabel untuk monitor akuakultur telah di desain dan diimplementasikan. Data yang terkumpul memberikan hasil analisis dari operasi sistem yang berhasil.
3.	<i>Design and implementation of a distributed IoT system for the monitoring of water quality in Aquaculture</i> (Encinas, et al., 2017)	ZigBee Wireless Transmission	Penelitian ini mengimplementasikan prototype dan bukti dari konsep tentang pemantauan jarak jauh yang menggunakan konsep IoT untuk memantau kualitas air dalam akuakultur. Sistem ini

		membutuhkan biaya yang murah, konsumsi daya yang rendah, skalabilitas, serba guna, terdistribusi, bergerak, dan akurat.
--	--	---

Tabel 2.1 menjelaskan tentang kajian pustaka yang digunakan pada penelitian ini. Tabel tersebut berisi judul, metode, dan hasil dari penelitian sebelumnya yang pernah dilakukan yang serupa dengan penelitian yang dilakukan.

2.2 Dasar Teori

2.2.1 Akuakultur

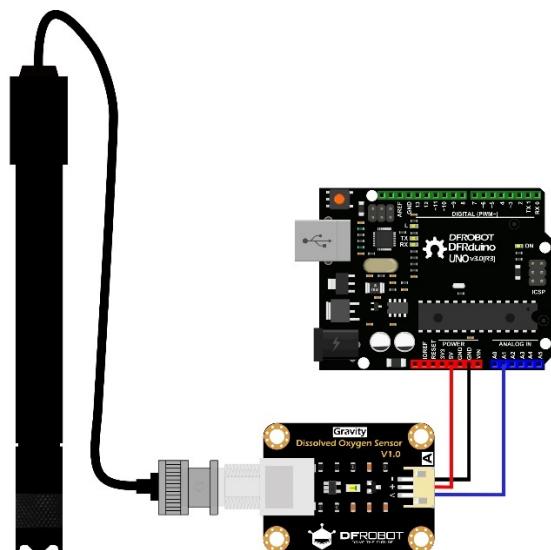
Akuakultur adalah pertanian di bidang organisme akuatik, yaitu ikan, moluska, krustasea, dan tumbuhan akuatik. Pertanian diartikan sebagai bentuk campur tangan dalam proses pemeliharaan untuk meningkatkan produksi, seperti pengaturan stok, pemberian makan, perlindungan dari predator, dan lain-lain. Pertanian disini juga diartikan sebagai kepemilikan individu atau perusahaan dari saham yang dibudidayakan (FAO, 1988).

2.2.2 Sensor

Sensor adalah perangkat yang mendekripsi dan merespon masukan dari lingkungan fisik. Masukan bisa berupa cahaya, panas, gerakan, kelembaban, tekanan dan semua fenomena alam yang lain. Hasil keluaran dari sensor secara umum adalah sinyal yang diubah ke tampilan yang dapat dipahami manusia. Sensor yang digunakan dalam penelitian ini adalah sensor oksigen terlarut, sensor keasaman, sensor kekeruhan, dan suhu.

2.2.2.1 Sensor oksigen terlarut

Oksigen terlarut adalah salah satu parameter penting dalam kualitas air. Rendahnya oksigen di dalam air akan menyebabkan sulitnya organisme dalam bernapas. Sensor oksigen yang digunakan dalam penelitian ini adalah *Gravity: Analog Dissolved Oxygen Sensor Kit* dari DFRobot. Sensor oksigen terlarut yang digunakan dapat dilihat pada Gambar 2.1.



Gambar 2.1 Sensor *analog dissolved oxygen*

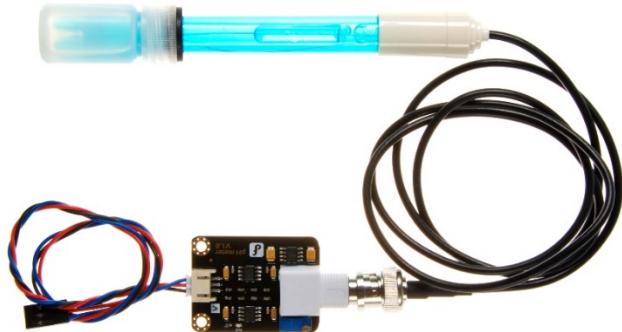
Gambar 2.1 menunjukkan rangkaian sensor oksigen terlarut. Dalam rangkaian tersebut terdapat komponen yaitu *sensor probe*, *signal converter board*, dan mikrokontroler. Spesifikasi perangkat sensor oksigen terlarut dapat dilihat pada Tabel 2.1.

Tabel 2.1 Komponen sensor oksigen terlarut

Dissolved Oxygen Probe	Probe bersifat <i>galvanic</i> , jadi tidak memerlukan waktu polarisasi
	Larutan dan tutup membran dapat diganti, mengurangi biaya <i>maintenance</i>
Signal Converter Board	3.3-5.5V <i>Power supply</i> , kompatibel dengan sebagian besar perangkat mikrokontroler arduino
	0-3.0V <i>analog output</i> , kompatibel dengan semua mikrokontroler dengan fungsi ADC

2.2.2.2 Sensor keasaman

Tingkat keasaman adalah salah satu parameter penting dalam kualitas air. Tingkat keasaman diukur dengan satuan pH. pH kurang dari 7 bersifat asam, sedangkan pH lebih dari 7 bersifat basa. Sensor keasaman yang digunakan dalam penelitian ini adalah *Gravity: Analog PH Meter Sensor Kit* dari DFRobot. Sensor oksigen terlarut yang digunakan dapat dilihat pada Gambar 2.1.



Gambar 2.2 Sensor *analog ph meter*

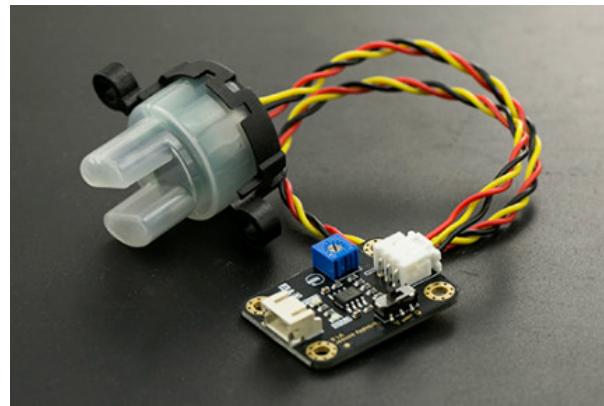
Gambar 2.2 menunjukkan rangkaian sensor keasaman *gravity: analog ph meter*. Dalam rangkaian tersebut terdapat komponen yaitu *sensor probe*, *signal converter board*, dan mikrokontroler. Spesifikasi perangkat sensor keasaman dapat dilihat pada Tabel 2.2.

Tabel 2.2 Spesifikasi *analog ph meter*

<i>Module Power: 5.00V</i>
<i>Module size: 43 x 32mm</i>
<i>Measuring Range: 0 – 14 pH</i>
<i>Measuring Temperature: 0 – 60 C</i>
<i>Accuracy: +- 0.1pH (25 C)</i>
<i>Response Time: <= 1min</i>
<i>pH Sensor with BNC Connector</i>
<i>Gain Adjustment Potentiometer</i>
<i>Power Indicator LED</i>

2.2.2.3 Sensor kekeruhan

Kekeruhan pada air mempengaruhi kualitas air, semakin keruh air, semakin sedikit cahaya yang masuk ke air. Intensitas cahaya yang masuk ke dalam air mempengaruhi kehidupan organisme air. Cahaya yang terlalu terang akan mengakibatkan stress pada ikan, sebaliknya cahaya yang kurang akan menyebabkan ikan sulit mencari makan. Sensor yang digunakan untuk mendeteksi tingkat kekeruhan air dalam penelitian ini adalah *Gravity: Analog Turbidity Sensor* dapat dilihat pada Gambar 2.3.



Gambar 2.3 Sensor *analog turbidity*

Gambar 2.3 menunjukkan perangkat sensor tingkat kekeruhan *gravity: analog turbidity*. Dalam rangkaian tersebut terdapat komponen yaitu *sensor probe* dan *signal converter board*. Spesifikasi perangkat sensor keasaman dapat dilihat pada Tabel 2.3.

Tabel 2.3 Spesifikasi sensor *analog turbidity*

<i>Operating Voltage: 5V DC</i>
<i>Operating Current: 40mA (MAX)</i>
<i>Response Time: <500ms</i>
<i>Insulation Resistance: 100M (Min)</i>
<i>Output Method: Analog</i>
<i>Analog output: 0-4.5V</i>
<i>Digital Output: High/Low level signal</i>
<i>Operating Temperature: 5 °C~90 °C</i>
<i>Storage Temperature: -10 °C~90 °C</i>
<i>Weight: 30g</i>
<i>Adapter Dimensions: 38mm*28mm*10mm/1.5inches *1.1inches*0.4inches</i>

2.2.2.4 Sensor suhu

Suhu adalah salah satu parameter penting dalam kualitas air yang baik. Suhu yang terlalu tinggi dapat mempengaruhi kehidupan ikan yaitu peningkatan makan ikan, peningkatan metabolisme ikan, dan kematian ikan. Suhu yang optimal untuk akuakultur berkisar antara 25 sampai 32 derajat celcius. Pada penelitian ini, sensor suhu yang digunakan adalah *DS18B20: Digital Waterproof Temperature Sensor*. Sensor suhu yang digunakan dapat dilihat pada Gambar 2.1.



Gambar 2.4 Sensor *digital waterproof temperature*

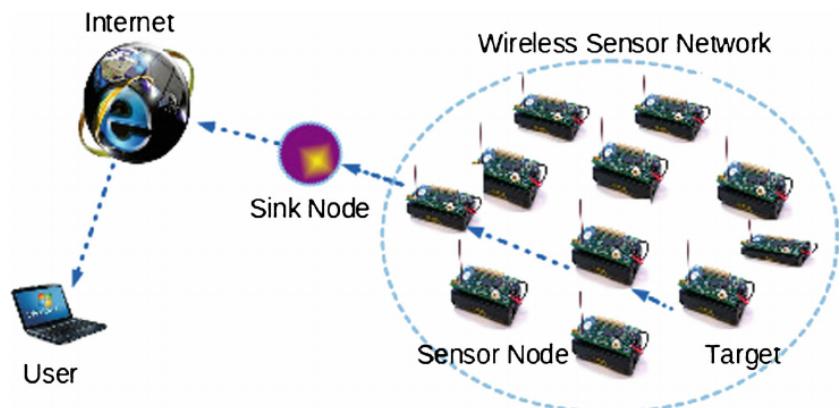
Gambar 2.4 menunjukkan perangkat sensor suhu DS18B20: *Digital Waterproof Temperature Sensor*. Spesifikasi perangkat sensor keasaman dapat dilihat pada Tabel 2.4.

Tabel 2.4 Spesifikasi sensor *digital waterproof temperature*

Ketahanan fleksibilitas, konduktivitas termal yang lebih baik.
Ukuran selubung baja tahan karat: 6 x 30mm
Rentang catu daya: 3.0V hingga 5.5V
Kisaran suhu pengoperasian: -55 ° C hingga + 125 ° C (-67 ° F hingga + 257 ° F)
Kisaran suhu penyimpanan: -55 ° C hingga + 125 ° C (-67 ° F hingga + 257 ° F)
Akurasi pada rentang -10 ° C hingga + 85 ° C: ± 0,5 ° C.

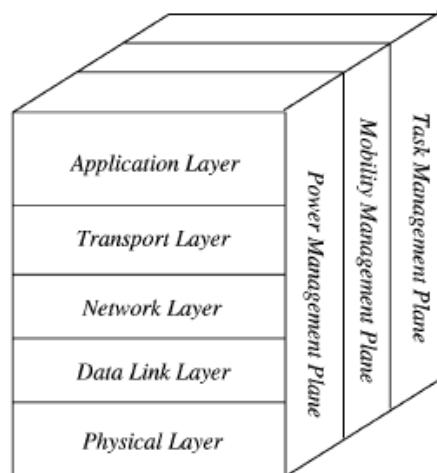
2.2.3 Jaringan Sensor Nirkabel

Jaringan Sensor Nirkabel atau *Wireless Sensor Network (WSN)* adalah suatu jaringan nirkabel yang terdiri dari beberapa *sensor node* yang saling berkomunikasi dan bekerja sama untuk mengumpulkan data-data dari lingkungan yang dipantau. Karena alasan tersebut suatu *node* diperlengkapi dengan peralatan sensor yang digunakan untuk mendeteksi lingkungan sekitar dan peralatan komunikasi yang digunakan untuk berkomunikasi dengan *sensor node* yang lain (Dargie & Poellabauer, 2010). Arsitektur umum WSN ditunjukkan pada Gambar 2.5.



Gambar 2.5 Arsitektur WSN

Gambar 2.5 menunjukkan arsitektur WSN pada umumnya yaitu terdiri dari beberapa *sensor node (source node)* yang saling terhubung dan *sink node* yang terhubung ke internet untuk kemudian diakses oleh pengguna. Cara kerja WSN secara umum adalah data *sensing* yang diperoleh dari beberapa *sensor node* dikumpulkan ke dalam *sink node*. Peran *sink node* dalam WSN adalah mengumpulkan data *sensing* dan terhubung dengan internet. Dalam penelitian ini yang berperan sebagai *sink node* adalah *gateway*. *Protocol Stack* untuk WSN ditunjukkan pada Gambar 2.6.



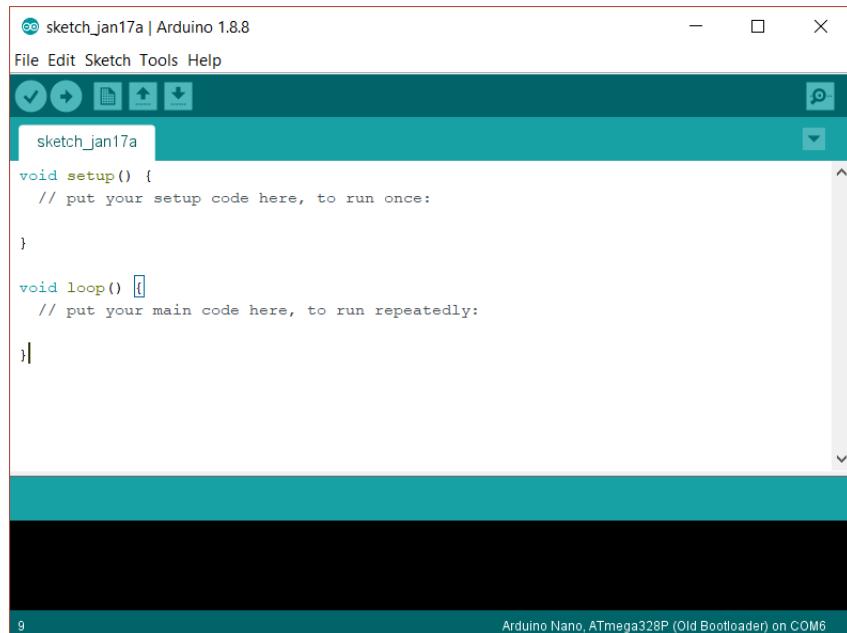
Gambar 2.6 WSN Protocol Stack

Sumber: Akyildiz (2002)

Dilihat dari Gambar 2.6 *protocol stack* pada WSN didasari oleh model OSI. Ada lima lapisan (*layer*) yaitu *application layer*, *transport layer*, *network layer*, *data link layer* dan *physical layer*. Di dalam *layer* utama ditambahkan *cross layer* yaitu *power management plane*, *mobility management plane* dan *task management plane*.

2.2.4 Arduino

Arduino adalah platform elektronik *open-source hardware* dan *software* yang mudah digunakan. Papan arduino dapat membaca *input* antara lain: lampu pada sensor, jari pada tombol, atau pesan dalam aplikasi *Telegram*, lalu menjadikannya *output* seperti: menyalakan mesin, menyalakan LED, mem-*publish* sesuatu secara online. Perintah-perintah dalam arduino ditulis dalam bahasa pemrograman Arduino dan menggunakan *software* Arduino IDE. Tampilan arduino IDE dapat dilihat pada Gambar 2.7.



The screenshot shows the Arduino IDE interface. The title bar reads "sketch_jan17a | Arduino 1.8.8". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for file operations like Open, Save, and Print. The main workspace displays the following code:

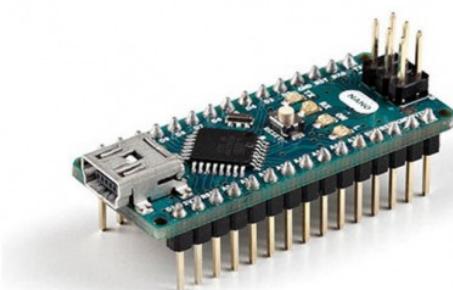
```
sketch_jan17a
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

The status bar at the bottom indicates "Arduino Nano, ATmega328P (Old Bootloader) on COM6".

Gambar 2.7 Tampilan Arduino IDE

Seperti yang terlihat pada Gambar 2.7 , bahasa pemrograman arduino memiliki dua fungsi utama yang wajib ada yaitu `setup()` dan `loop()` . Fungsi `setup()` digunakan untuk mendefinisikan apa saja yang perlu dijalankan saat pertama kali arduino mendapat daya. Fungsi `loop()` digunakan untuk mengulang – ulang kode program selama arduino mendapat daya. Perangkat arduino yang digunakan pada penelitian ini adalah Arduino Nano ditunjukkan pada Gambar 2.8.



Gambar 2.8 Arduino Nano

Arduino Nano yang ditunjukkan pada Gambar 2.8 adalah mikrokontroler yang digunakan untuk mengelola data sensor. Arduino Nano memiliki kelebihan yaitu

konsumsi daya yang rendah, performa yang baik, pin SPI untuk modul HopeRF-RFM9x berbasis Lora, serta pin analog dan digital yang digunakan untuk komunikasi dengan sensor yang digunakan untuk pemantauan parameter fisik air kolam ikan. Spesifikasi Arduino Nano ditunjukkan pada tabel 2.5.

Tabel 2.5 Spesifikasi Arduino Nano

<i>Microcontroller</i>	<i>ATmega328</i>
<i>Operating Voltage</i>	<i>5V</i>
<i>Input Voltage</i>	<i>7-12V</i>
<i>Digital I/O Pins</i>	<i>14</i>
<i>Analog Input Pins</i>	<i>8</i>
<i>Flash Memory</i>	<i>32 KB (ATmega328) of which 2 KB used by bootloader</i>
<i>SRAM</i>	<i>2 KB</i>
<i>Clock Speed</i>	<i>16MHz</i>

2.2.5 Raspberry Pi

Raspberry Pi adalah mikrokomputer yang dapat terhubung dengan *monitor* atau TV dan menggunakan *keyboard* dan *mouse* pada umumnya. Raspberry Pi dapat digunakan untuk menjelajah internet, memutar video, dan memainkan game seperti komputer pada umumnya (Raspberry Pi Foundation, 2015). Dalam penelitian ini raspberry pi berperan sebagai *gateway*. Perangkat Raspberry Pi yang digunakan ditunjukkan pada gambar 2.9.



Gambar 2.9 Raspberry Pi 3 Model B+

Raspberry Pi 3 Model B+ yang ditunjukkan pada gambar 2.9, digunakan karena struktur layout pin yang dimiliki oleh raspberry dapat diintegrasikan dengan modul HopeRF-RFM9x. Peran raspberry pada sistem ini adalah sebagai gateway pada *node server*. Selain raspberry pi 3 model b+, raspberry pi 2 juga dapat digunakan sebagai alternatif karena memiliki *layout pin* yang sama. Spesifikasi perangkat ini ditunjukkan pada tabel 2.6.

Tabel 2.6 Spesifikasi Raspberry Pi 3 Model B+

<i>SoC</i>	<i>Broadcom BCM2837B0 quad-core A53 (ARMv8) 64-bit @ 1.4GHz</i>
<i>GPU</i>	<i>Broadcom Videocore-IV</i>
<i>RAM</i>	<i>1GB LPDDR2 SDRAM</i>
<i>Networking</i>	<i>Gigabit Ethernet (via USB channel), 2.4GHz and 5GHz 802.11b/g/n/ac Wi-Fi</i>
<i>Bluetooth</i>	<i>Bluetooth 4.2, Bluetooth Low Energy (BLE)</i>
<i>Storage</i>	<i>Micro-SD</i>
<i>GPIO</i>	<i>40-pin GPIO header, populated</i>
<i>Ports</i>	<i>HDMI, 3.5mm analogue audio-video jack, 4x USB 2.0, Ethernet, Camera Serial Interface (CSI), Display Serial Interface (DSI)</i>
<i>Dimensions</i>	<i>82mm x 56mm x 19.5mm, 50g</i>

2.2.6 LoRa

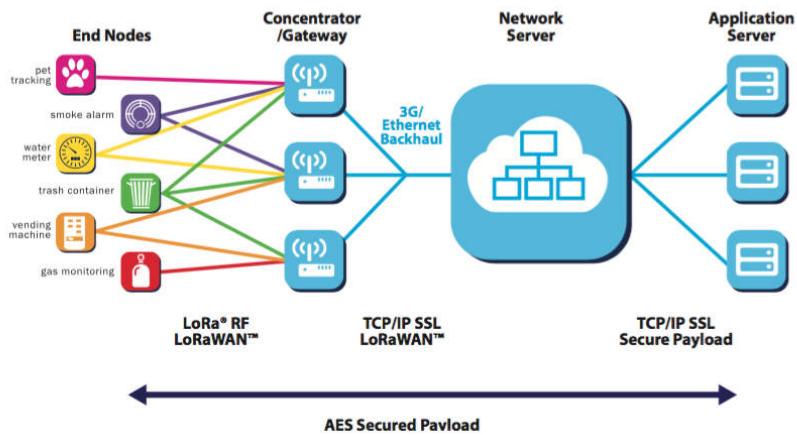
LoRa adalah teknologi nirkabel yang memiliki layanan jarak yang jauh, kebutuhan daya yang rendah, dan keamanan transmisi data untuk aplikasi IoT. Dasar dari modulasi LoRa adalah *chirp spread spectrum modulation*, yang mempunyai karakteristik daya yang rendah seperti modulasi FSK (*Frequency Shifting Key*). Yang membedakan modulasi LoRa dari modulasi FSK adalah kemampuan LoRa yang dapat digunakan untuk komunikasi jarak jauh. LoRa dapat digunakan untuk menghubungkan sensor, *gateway* dan mesin secara nirkabel.

Pada teknologi LoRa, frekuensi yang digunakan akan berbeda untuk setiap wilayah. Frekuensi yang digunakan untuk wilayah Amerika Serikat adalah 915 MHz, untuk wilayah Eropa frekuensi yang digunakan adalah 433/868 MHz, dan untuk wilayah Asia frekuensi yang digunakan adalah 433 MHz.

LoRa biasanya diartikan sebagai dua lapisan (*layer*) yang berbeda dalam *protocol stack* yaitu:

1. Lapisan fisik (*Physical Layer*) yang menggunakan teknik modulasi radio *Chirp Spread Spectrum (CSS)*. Lapisan fisik LoRa dikembangkan oleh Semtech yang memungkinkan komunikasi jarak jauh, rendah daya, dan *throughput* yang kecil. *Payload* pada setiap transmisi berkisar antara 2 sampai 255 oktet, dan *data rate* mencapai 50Kbps ketika saluran agregasi digunakan.
2. Protokol *MAC Layer* atau di dalam LoRa disebut LoRaWAN menyediakan mekanisme akses kontrol yang memungkinkan banyak *end-device* untuk berkomunikasi dengan *gateway* menggunakan modulasi LoRa.

Arsitektur LoRa ditunjukkan pada Gambar 2.10 .



Gambar 2.10 Arsitektur LoRa

Gambar 2.10 menunjukkan arsitektur jaringan LoRa pada umumnya. Jaringan LoRa menggunakan topologi *star* karena dapat meninkatkan daya hidup baterei ketika koneksi *long-range* digunakan. Jaringan LoRa terdiri dari beberapa elemen yaitu *LoRa node*, *LoRa gateways*, *network servers*. Keterangan setiap elemen jaringan LoRa adalah sebagai berikut:

a. *LoRa nodes*

LoRa nodes adalah sensor atau aplikasi dimana proses *sensing* dilakukan.

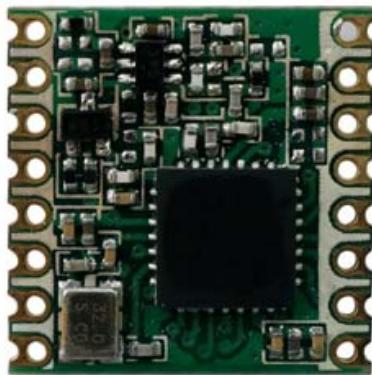
b. *LoRa gateways*

Di dalam LoRa, *node* dihubungkan dengan *gateway* tertentu. Semua data yang di transmisikan oleh *node* dikirimkan ke semua *gateway* dan setiap *gateway* yang menerima sinyal tersebut meneruskan data ke server jaringan berbasis *cloud*.

c. *Network servers*

Network servers adalah tempat untuk penyaringan paket duplikat dari beberapa *gateway* yang berbeda, melakukan pengecekan keamanan, dan meneruskan paket ke aplikasi *server* tertentu.

Modul LoRa yang digunakan pada penelitian ini adalah RFM95W ditunjukkan pada Gambar 2.11.



Gambar 2.11 Modul RFM95/96/97/98(W)

Modul RFM95/96/97/98(W) (Gambar 2.11) adalah modul komunikasi berbasis *modem* LoRa yang menyediakan layanan yaitu komunikasi penyebaran spektrum yang memiliki jangkauan sangat luas dan kekebalan tinggi terhadap gangguan sinyal serta konsumsi daya yang rendah. Spesifikasi perangkat ini ditunjukkan pada Tabel 2.7.

Tabel 2.7 Spesifikasi Modul RFM95/96/97/98(W)

<i>Lora Modem</i>
<i>168 dB maximum link budget</i>
<i>+20 dBm – 100 mW constant RF output vs. V supply</i>
<i>+14 dBm high efficiency PA</i>
<i>Programmable bit rate up to 300 kbps.</i>
<i>High sensitivity: down to -148 dBm.</i>
<i>Low RX current of 10.3 mA, 200 nA register retention.</i>
<i>Built-in temperature sensor and low battery indicator.</i>
<i>Module Size : 16*16mm</i>

2.2.7 Structure (struct)

Structure dalam bahasa pemrograman C++ adalah kumpulan variabel dari tipe data yang berbeda di dalam nama yang sama. *Structure* memiliki kesamaan dengan *object* pada bahasa pemrograman java, perbedaannya adalah *structure* tidak memiliki kemampuan untuk menyimpan fungsi atau method serta *structure* tidak dapat menjalankan fitur OOP seperti yang *object* bisa lakukan. Contoh deklarasi struct dalam kasus penyimpanan informasi mahasiswa ditunjukkan pada Tabel 2.8.

Tabel 2.8 Format struct

```
1
2 struct Mahasiswa {
3     char nama[40];
4     int nim;
5     float IPK;
6 }
```

Dalam Tabel 2.8 menunjukkan deklarasi *structure* dengan nama *Mahasiswa*. Deklarasi *structure* diawali dengan kata *struct* dan diikuti nama variabel *structure* dan kurung kurawal. Di dalam kurung kurawal dideklarasikan variabel dengan tipe data tertentu seperti pada umumnya. *Structure* mendukung semua tipe data yang ada di bahasa pemrograman C++ .

2.2.8 JSON

JavaScript Object Notation (JSON) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan di generate oleh komputer (Crockford, 2013). JSON dapat dipakai oleh semua bahasa pemrograman turunan dari C seperti C++, C#, Java, Javascript dan Python. JSON terdiri dari kumpulan pasangan *key* dan *value* seperti *object*, dan daftar nilai – nilai terurut seperti *array*. Contoh format JSON dalam kasus penyimpanan informasi mahasiswa ditunjukkan pada Tabel 2.9.

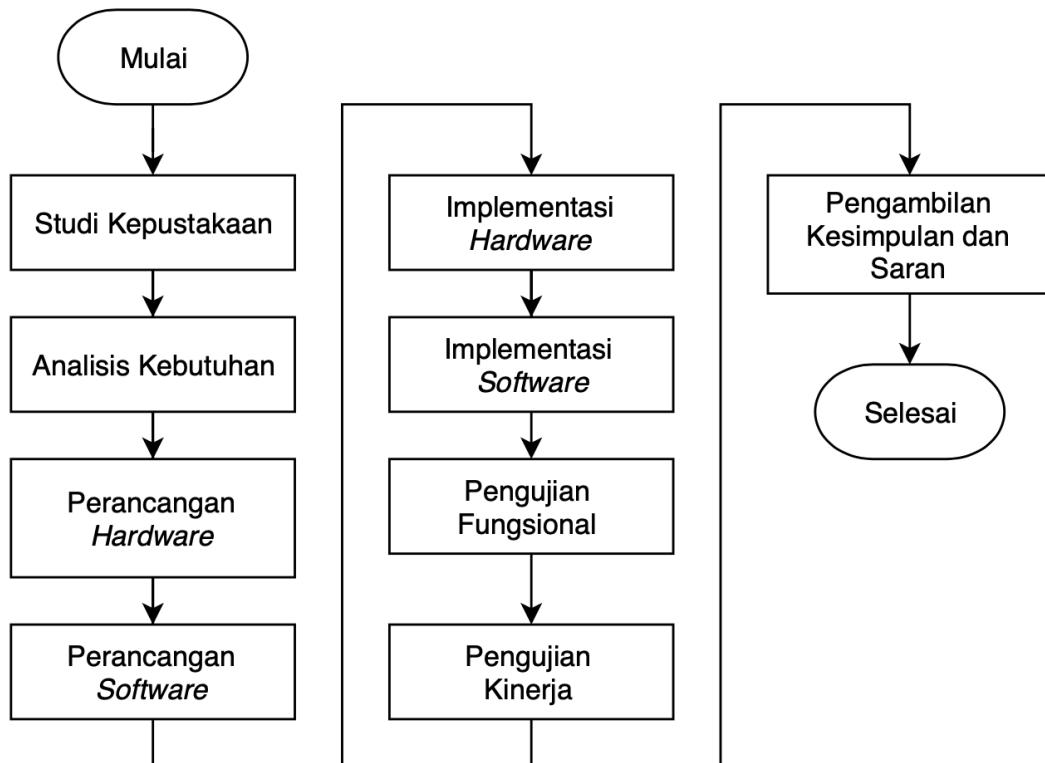
Tabel 2.9 Format JSON

```
1 {
2     "nama" : "Nida Kot Atraled",
3     "NIM" : "111222333444555666",
4     "IPK" : 3.73
5 }
```

Dalam Tabel 2.8 menunjukkan format JSON dalam kasus penyimpanan informasi mahasiswa. Format JSON diawali dan diakhiri dengan kurung kurawal ({ }), selain itu juga dapat diawali dan diakhiri dengan kurung siku ([]) untuk menyimpan data berupa *array*. *Key* dalam JSON di deklarasikan dengan menggunakan tanda kutip (" "). Untuk memisahkan *key* dan *value* dalam JSON menggunakan penanda titik dua (:) dan untuk memisahkan antara satu pasangan *key value* menggunakan tanda koma (,). Tipe data yang didukung JSON adalah *boolean*, *array*, *string*, *object*, *number*, dan *null*.

BAB 3 METODOLOGI

Bab ini membahas tentang tahapan-tahapan yang dilakukan pada sistem *monitoring* parameter fisik air kolam ikan menggunakan jaringan sensor nirkabel berbasis protokol LoRa untuk mencapai tujuan penelitian. Tahapan tersebut adalah studi kepustakaan, analisis kebutuhan, perancangan sistem, implementasi sistem, pengujian sistem, serta pengambilan kesimpulan dan saran sistem. Diagram alur tahapan-tahapan tersebut dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Alur Metodologi Penelitian

3.1 Studi Kepustakaan

Tahapan pertama dalam penelitian ini adalah studi kepustakaan. Dalam tahap ini dilakukan proses pengumpulan sumber-sumber yang berhubungan atau terkait dengan topik penelitian. Studi kepustakaan berguna untuk mengumpulkan teori yang digunakan peneliti dalam penelitian ini. Studi kepustakaan dalam mendukung penelitian ini antara lain sebagai berikut:

1. Faktor kualitas air yang baik, untuk pertumbuhan ikan.
2. Penerapan LoRa sebagai protokol komunikasi *sensor node* dan *node server*.

Kemudian dilakukan pengumpulan dasar teori mengenai pemrograman Python, *database* MongoDB, Mikrokontroller, protokol MQTT, dan data sensor. Sumber teori didapatkan dari jurnal, buku dan konferensi ilmiah.

3.2 Analisis Kebutuhan

Tahap Analisis kebutuhan sistem pada penelitian yang dilakukan ini sebagai dasar dalam melakukan implementasi, agar semua yang dibuat sesuai dengan apa yang dibutuhkan. Pada penelitian ini dibagi menjadi tiga kebutuhan utama yaitu kebutuhan fungsional, kebutuhan data dan kebutuhan nonfungsional.

1. Kebutuhan fungsional

Kebutuhan fungsional berisikan deskripsi tentang apa saja yang bisa dilakukan oleh sistem. Kebutuhan fungsional sistem secara umum meliputi kemampuan *sensor node* dalam mengakuisisi data dari sensor oksigen terlarut, keasaman, kekeruhan, dan suhu, kemampuan *node sensor* dalam melakukan mekanisme *packing* data ke dalam *struct*, dan kemampuan *gateway* dalam mengubah data *struct* menjadi JSON dan mengirimnya ke *cloud server* dengan menggunakan protokol HTTP.

2. Kebutuhan data

Kebutuhan data dalam penelitian ini berasal dari hasil *sensing* sensor oksigen terlarut, keasaman, kekeruhan dan suhu berupa *float*. Data sensor yang berhasil diakuisisi kemudian akan diproses di mikrokontroler untuk menjalankan fungsi *packing* ke dalam *struct*. Data sensor yang bertipe *struct* kemudian dikirimkan menuju *gateway* melalui protokol LoRa. Data yang berhasil terima *gateway* kemudian diubah ke dalam format JSON untuk kemudian dikirimkan menuju *cloud server*.

3. Kebutuhan nonfungsional

Kebutuhan nonfungsional berisikan kebutuhan diluar fungsi utama yang dapat mendukung kinerja sistem. Kebutuhan nonfungsional dalam penelitian ini mencangkup kebutuhan perangkat keras dan kebutuhan perangkat lunak. Perangkat keras yang dibutuhkan agar penelitian ini berjalan adalah *Arduino Nano*, *Modul LoRa HopeRF RFM95W*, sensor oksigen terlarut, sensor keasaman, sensor kekeruhan, sensor suhu dan *Raspberry Pi*. Kebutuhan perangkat lunak dalam penelitian ini yaitu *Python*, *library rf95* untuk *python*, *library HTTPlib* untuk *python*, *Arduino IDE*, *library rfm95* untuk *Arduino*, dan *putty*.

3.3 Perancangan Sistem

Perancangan yang dibuat dalam penelitian ini terdiri dari perancangan arsitektur, perancangan alur sistem, perancangan *sensor node*, perancangan *gateway*, perancangan data, perancangan komponen visualisasi dan perancangan pengujian.

1. Perancangan arsitektur sistem

Perancangan arsitektur sistem membahas tentang rancangan arsitektur sistem yang dibuat. Arsitektur sistem dalam penelitian ini terdiri atas *sensor node*, *gateway*, *cloud server*, dan komponen visualisasi data.

Sensor node terdiri dari mikrokontroler Arduino Nano, modul komunikasi LoRa HopeRF RFM95W dan empat sensor yaitu sensor oksigen terlarut, sensor keasaman, sensor kekeruhan, dan sensor suhu. *Gateway* terdiri dari mikrokomputer Raspberry Pi dan modul LoRa HopeRF RRFM95W. *Cloud server* yang digunakan adalah iot.tujuhlangit.id yang dibuat oleh (Pratama, et al., 2018). Komponen visualisasi data adalah berupa *web application* yang dibangun dengan HTML, CSS, dan JavaScript.

2. Perancangan alur sistem

Perancangan alur sistem diperlukan untuk memberi gambaran besar tentang arsitektur yang dibuat berdasarkan dengan analisis kebutuhan yang telah ditentukan. Dalam sistem yang dibuat terdapat dua format data utama yang harus ada yaitu *struct* dan JSON. *Struct* digunakan sebagai mekanisme *packing* data sensor yang terdiri dari sensor oksigen terlarut, sensor keasaman, sensor kekeruhan dan sensor suhu. Setelah mekanisme *packing* dilakukan, maka data sensor berupa *struct* akan dikirim ke *gateway*. *Gateway* kemudian akan mengubah data *struct* menjadi JSON dan kemudian dikirim menuju *cloud server* menggunakan protokol HTTP.

3. Perancangan *sensor node*

Perancangan *sensor node* dimulai dengan pembuatan perangkat keras. Perangkat keras yang digunakan untuk membangun *sensor node* adalah mikrokontroler, sensor dan modul komunikasi LoRa. Mikrokontroler yang digunakan adalah Arduino Nano, sedangkan untuk sensor yang digunakan ada empat yaitu sensor oksigen terlarut, sensor keasaman, sensor kekeruhan, dan sensor suhu. Modul komunikasi LoRa yang digunakan adalah HopeRF RFM95W. Proses *wiring* dan *pinout* yang digunakan akan dibahas lebih detail pada bab perancangan sistem.

Setelah perangkat keras berhasil dibuat, selanjutnya adalah pembuatan perangkat lunak pada *sensor node* berupa kode program. Bahasa pemrograman yang digunakan pada *sensor node* adalah Arduino yang berbasis bahasa C++. Penjelasan kode program akan dibahas lebih detail pada bab implementasi sistem.

4. Perancangan *gateway*

Perancangan *gateway* dimulai dengan pembuatan perangkat keras. Perangkat keras yang digunakan untuk membangun *gateway* adalah mikrokomputer dan modul komunikasi LoRa. Mikrokomputer yang digunakan adalah Raspberry Pi 3 B+, sedangkan untuk modul komunikasi LoRa yang digunakan adalah HopeRF RFM95W. Proses *wiring* dan *pinout* yang digunakan akan dibahas lebih detail pada bab perancangan sistem.

Setelah perangkat keras berhasil dibuat, selanjutnya adalah pembuatan perangkat lunak pada *gateway* berupa kode program. Bahasa pemrograman yang digunakan pada *gateway* adalah Python. Penjelasan kode program akan dibahas lebih detail pada bab implementasi sistem.

5. Perancangan data

Perancangan data dimulai dengan penentuan data apa saja yang akan ditransmisikan mulai dari *sensor node*, *gateway* hingga *cloud server*. Pada *sensor node* terdapat empat sensor berbeda, maka dari itu dibutuhkan mekanisme *packing* data yang memiliki ukuran data kecil agar data yang diakuisi dalam *node sensor* dapat sekaligus dikirimkan dalam satu waktu. Representasi data yang digunakan untuk mengirim data dari *sensor node* ke *gateway* adalah *struct* menggunakan LoRa sedangkan untuk pengiriman data dari *gateway* ke *cloud server* adalah JSON menggunakan protokol HTTP.

6. Perancangan komponen visualisasi data

Komponen visualisasi data dalam penelitian ini berupa *web application* yang dibangun dengan HTML, CSS, dan JavaScript. Tahap pertama adalah merancang *mockup web application* yang akan dijelaskan lebih rinci pada bab perancangan. Tahap selanjutnya menulis kode program HTML, CSS dan JavaScript. Dalam pembangunan *web application* menggunakan bantuan *framework* CSS yaitu uiikit dan *library* JavaScript yaitu chartjs. *Minimum Viable Product* (MVP) dalam aplikasi yang dibangun adalah *web application* mampu menampilkan data sensor oksigen terlarut, sensor keasaman, sensor kekeruhan dan sensor suhu.

7. Perancangan pengujian

Pada perancangan pengujian dijelaskan rincian pengujian yang dilakukan yaitu perancangan pengujian fungsionalitas dan pengujian kinerja. Perancangan pengujian fungsionalitas dibutuhkan agar pengujian tidak melebihi batasan masalah sedangkan perancangan pengujian kinerja untuk mengukur kinerja sistem yang dibuat.

3.4 Implementasi Sistem

Pada tahapan ini, implementasi akan dilakukan sesuai dengan apa yang dirancang dalam perancangan sistem. Pertama-tama implementasi akan dilakukan pada *sensor node*. Di dalam *sensor node* akan dilakukan pengambilan data sensor yang diperoleh dari sensor yang terpasang. Setelah data didapatkan, mikrokontroller ksemudian akan mengirimkan data sensor menuju gateway dengan menggunakan modul LoRa HopeRF-RFM95 dengan format struct. *Gateway* yang digunakan pada penelitian ini dibuat oleh (Arijuddin, et al., 2018) dengan modifikasi pada bagian kode programnya. Lalu pada gateway akan menerima data sensor dan mengirimkan data sensor tersebut ke *cloud server* dengan format JSON. Dalam penelitian ini *Cloud server* yang digunakan dibuat oleh (Pratama, et al., 2018). *Cloud server* yang telah dibuat oleh (Pratama, et al., 2018) menggunakan JWT Token sebagai mekanisme autentikasi dan otorisasi nya sehingga data yang dikirimkan oleh perangkat akan tervalidasi dan teridentifikasi. Setelah data berhasil dikirimkan ke *cloud server*, *web application* akan melakukan *request* data dari API lalu akan menampilkan dalam bentuk halaman web.

3.5 Pengujian Sistem

Pengujian sistem bertujuan untuk mengetahui kesalahan maupun kekurangan yang ada pada sistem serta implementasi metode yang digunakan. Proses pengujian juga ditujukan untuk mengukur performa LoRa. Pengujian yang akan dilakukan adalah pengujian fungsional dan pengujian kinerja.

1. Pengujian fungsional

Secara umum pengujian fungsionalitas pada sistem yang akan dibuat adalah menguji apakah data sensor yang dikirim dari *sensor node* hingga sampai *web application* berhasil terkirim atau tidak. Pengujian fungsional dilakukan berdasarkan analisis kebutuhan fungsional pada bab analisis kebutuhan. Dalam pengujian fungsional juga terdapat pengujian sistem dengan berbagai kondisi air yang berbeda yaitu air keruh, air panas, air dingin, air basa, dan air asam. Pembahasan lebih detail untuk pengujian fungsional terdapat pada bab pengujian sistem.

2. Pengujian kinerja

Pengujian kinerja dilakukan untuk mengukur parameter *successful rate* pada pengiriman data dari *sensor node* hingga *web application* dengan pengaruh interval waktu dan besar data yang dikirim dari *sensor node*. Pengujian jarak yang di lakukan adalah 50 meter, 150 meter, 200 meter, dan 400 meter.

3.6 Pengambilan Kesimpulan dan Saran

Tahap pengambilan kesimpulan akan membahas hasil performa LoraWAN yang menggunakan protokol LoRa. Kesimpulan akan didapat setelah setelah seluruh kegiatan perancangan, implementasi, serta pengujian telah dilakukan. Kemudian saran yang didapat untuk memperbaiki kesalahan serta melakukan perbaikan pada penulisan dan sistem serta memberi pertimbangan pada hasil yang didapatkan.

BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN SISTEM

4.1 Analisis Kebutuhan Sistem

4.1.1 Kebutuhan Fungsional Sistem

Kebutuhan fungsional adalah layanan yang harus ada di dalam sistem. Secara spesifik, kebutuhan fungsional pada sistem ini fokus pada komunikasi antar node dengan LoRa. Kebutuhan fungsional sistem ditunjukkan pada Tabel 4.1.

Tabel 4.1 Kebutuhan Fungsional Sistem

No	Kode	Kebutuhan Fungsional
1	KF-01	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan oksigen terlarut dari sensor <i>dissolved oxygen</i> .
2	KF-02	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan kadar cahaya dari sensor <i>turbidity</i> .
3	KF-03	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan keasaman air dari sensor <i>ph meter</i> .
4	KF-04	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan suhu air dari sensor <i>temperature</i> .
5	KF-05	<i>Sensor node</i> dapat melakukan <i>packing</i> data berupa <i>struct</i> dari sensor <i>dissolved oxygen</i> , <i>turbidity</i> , <i>ph meter</i> , dan suhu.
6	KF-06	<i>Sensor node</i> dapat melakukan pengiriman data sensor berupa <i>struct</i> menuju <i>gateway</i> menggunakan protokol LoRa.
7	KF-07	<i>Gateway</i> dapat melakukan <i>unpack</i> data <i>struct</i> dan mengubah format ke dalam JSON
8	KF-08	<i>Gateway</i> dapat meneruskan data JSON menuju <i>cloud server</i>
9	KF-09	<i>Web Application</i> dapat menampilkan data yang didapat dari REST API <i>cloud server</i> .

4.1.2 Kebutuhan Data

Data yang dibutuhkan dalam penelitian yaitu data yang berasal dari sensor oksigen terlarut, sensor tingkat keasaman, sensor suhu, dan sensor kekeruhan. Data yang diterima dari sensor - sensor adalah tipe data *float* yang kemudian akan diolah oleh mikrokontroler menjadi *struct*. Setelah *struct* berhasil diubah menjadi JSON, data lalu dikirim menuju *cloud server* melalui protokol HTTP. Rincian kebutuhan data dapat dilihat pada Tabel 4.2.

Tabel 4.2 Kebutuhan data

Tipe Data	Jenis Data	Keterangan
<i>float</i>	Data oksigen terlarut dalam air	Data ini diperoleh dari sensor <i>dissolved oxygen</i> berupa tipe data <i>float</i>
<i>float</i>	Data tingkat keasaman air	Data ini diperoleh dari sensor <i>ph meter</i> berupa tipe data <i>float</i>
<i>float</i>	Data kekeruhan air	Data ini diperoleh dari sensor <i>turbidity</i> berupa tipe data <i>float</i>
<i>float</i>	Data suhu air	Data ini diperoleh dari sensor <i>temperature</i> berupa tipe data <i>float</i>

4.1.3 Kebutuhan Non Fungsional

Kebutuhan non fungsional menjelaskan tentang kebutuhan pendukung yang menjabarkan tentang kualitas dari sistem ini. Pada penelitian ini kebutuhan non fungsional terdiri dari kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.1.3.1 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras untuk mengimplementasikan sistem ini dapat dilihat pada Tabel 4.3.

Tabel 4.3 Kebutuhan perangkat keras

Perangkat	Keterangan
<i>Arduino Nano</i>	Perangkat ini digunakan sebagai mikrokontroler dalam <i>sensor node</i> .
<i>Modul LoRa RFM95</i>	Perangkat ini digunakan sebagai modul komunikasi berbasis protokol LoRa.
<i>DFRobot's Analog Dissolved Oxygen</i>	Perangkat ini digunakan untuk memantau kadar oksigen terlarut di dalam air
<i>DFRobot's Analog PH Meter</i>	Perangkat ini digunakan untuk memantau kadar keasaman air
<i>Water temperature sensor probe</i>	Perangkat ini digunakan untuk memantau suhu air
<i>DFRobot's Analog turbidity sensor</i>	Perangkat ini digunakan untuk memantau kekeruhan air
<i>Raspberry Pi 3 model B+</i>	Perangkat ini digunakan sebagai <i>gateway</i> yang berfungsi untuk menerima data dari <i>sensor node</i> dan mengirimkannya ke <i>cloud server</i> .

4.1.3.2 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak yang digunakan pada proses implementasi sistem ditunjukkan pada Tabel 4.4.

Tabel 4.4 Kebutuhan perangkat lunak

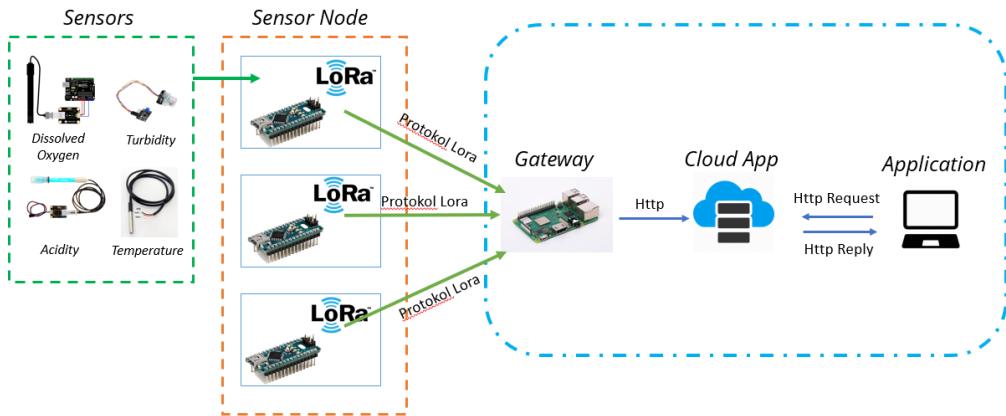
Perangkat	Keterangan
<i>Python</i>	Bahasa pemrograman yang digunakan pada <i>gateway</i> untuk menerima dan mengirimkan data menuju <i>cloud server</i> .
<i>Library rf95</i>	Pendukung bahasa pemrograman python untuk menghubungkan perangkat <i>gateway</i> dengan modul LoRa <i>RFM95</i> .
<i>Library HTTPLib</i>	Pendukung bahasa pemrograman python agar perangkat <i>gateway</i> dapat melakukan perintah – perintah umum pada protokol <i>HTTP</i> .
<i>Arduino IDE</i>	Aplikasi untuk menulis kode program pada mikrokontroler <i>Arduino Nano</i> .
<i>Library rfm95</i>	Pendukung bahasa pemrograman arduino untuk menghubungkan mikrokontroler dan modul LoRa <i>RFM95</i> .
<i>Putty</i>	Aplikasi untuk melakukan akses <i>remote</i> ke komputer lain.

4.2 Perancangan Sistem

Bagian ini akan menjelaskan tentang proses perancangan sistem mulai dari topologi sistem, perancangan *sensor node*, dan perancangan *web application*.

4.2.1 Gambaran Umum Sistem

Sistem yang akan dibuat pada penelitian ini adalah sistem *monitoring* parameter fisik air kolam ikan menggunakan jaringan sensor nirkabel berbasis protokol LoRa. *Sensor node* yang digunakan pada penelitian ini ada dua *node* yang terdiri dari satu *node* yang terpasang dengan sensor sebenarnya dan satu node lain yang akan mengirimkan data *dummy*. Data sensor yang diterima akan diolah oleh mikrokontroler *Arduino Nano*. Data yang telah diolah lalu dikirim ke *gateway* dengan menggunakan modul komunikasi LoRa *HopeRF-RFM95*. Perancangan arsitektur sistem yang akan dibagun ditunjukkan pada gambar 4.1.



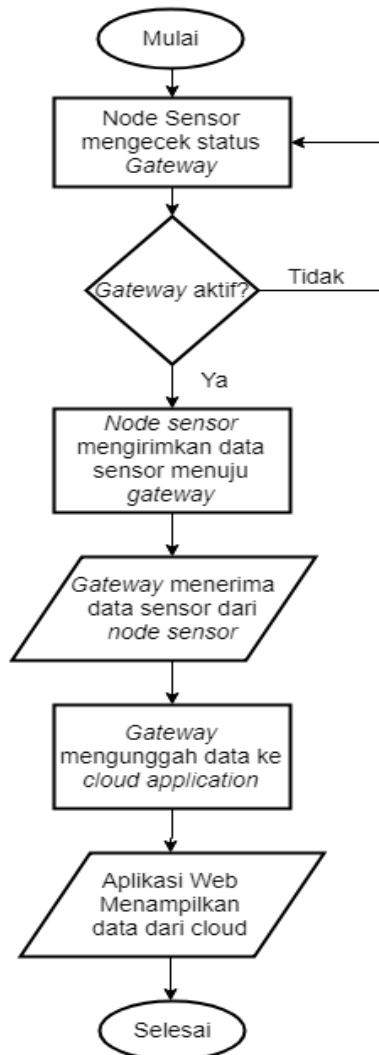
Gambar 4.1 Rancangan Arsitektur Sistem

Berdasarkan pada gambar 4.1 tersebut, poin-poin yang dapat dijelaskan untuk perancangan sistem adalah sebagai berikut:

1. *Sensor node* adalah kumpulan komponen yang terdiri atas sensor, mikrokontroller, dan modul komunikasi LoRa. Sensor yang digunakan adalah sensor oksigen Terlarut (DO), sensor keasaman (pH meter), sensor suhu dan sensor kekeruhan (*Turbidity*). Data yang didapat dari sensor akan diproses oleh Arduino Nano untuk kemudian dikirimkan ke *gateway* melalui LoRa.
2. *Gateway* merupakan perangkat yang menghubungkan *Sensor Node* dengan server. *Gateway* bertanggungjawab dalam menerima data dari *sensor node* dan mengirimkannya ke *cloud server*.
3. *Cloud server* dalam penelitian ini adalah aplikasi berbasis *cloud* yang menggunakan protokol HTTP yang bertanggungjawab untuk menyimpan data sensor dan melayani *request* oleh pengguna melalui REST API.
4. *Web Application* berfungsi untuk me *request* data dari API *cloud server* dan menampilkannya dalam bentuk tabel dan grafik.

4.2.2 Perancangan Alur Sistem

Pada sistem ini terdapat empat komponen utama yang saling berinteraksi yaitu *Sensor Node* yang menerima data *sensor* dan mengirimkannya menuju *gateway* melalui LoRa, Raspberry Pi sebagai *gateway*, *cloud server* sebagai tempat penyimpanan data sensor, dan Aplikasi Web yang akan menampilkan data yang didapat dari *Cloud Storage*. Interaksi antar komponen pada sistem ini dibagi menjadi dua yaitu pengiriman data sensor dari *sensor node* menuju *gateway* dan pengiriman data dari *gateway* menuju *cloud* untuk disimpan. *Flowchart* perancangan sistem yang dikembangkan dapat dilihat pada Gambar 4.1.



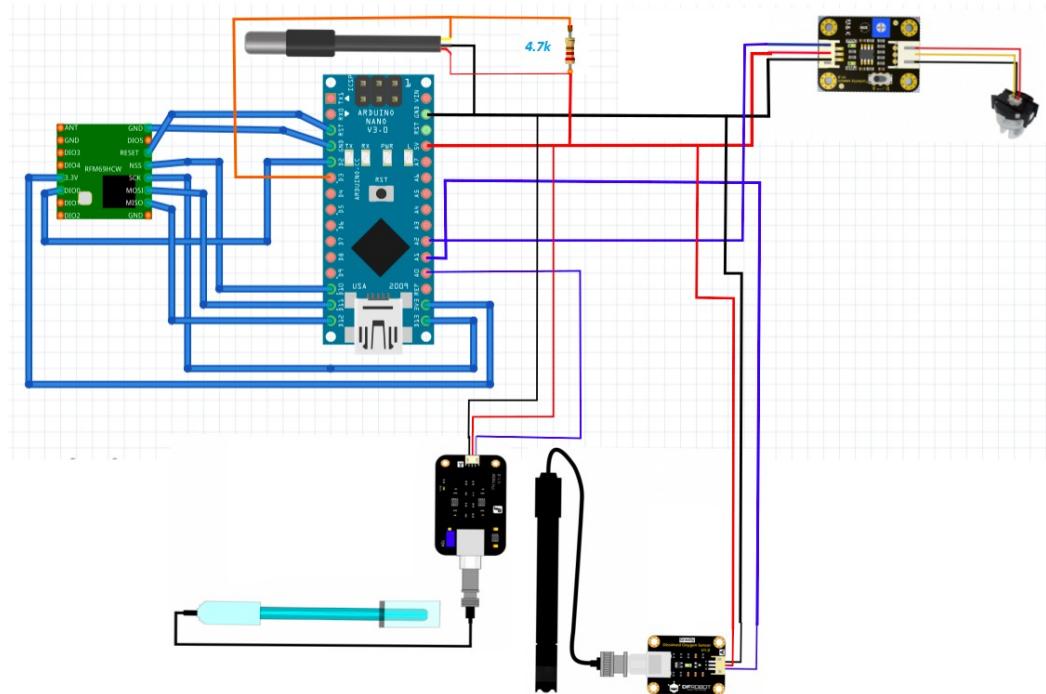
Gambar 4.2 Flowchart Perancangan Alur Sistem

Pada sistem ini terdapat empat sensor yang akan digunakan untuk pengukuran parameter fisik air kolam yaitu *dissolved oxygen* (oksigen terlarut), *ph meter* (tingkat keasaman), *turbidity* (kekeruhan) dan suhu. Sensor akan mendapat data hasil pemantauan dari lingkungan yang diteliti lalu data *sensor* yang berhasil diakuisi akan diproses dalam mikrokontroler Arduino Nano. Data *sensor* yang sudah diproses mikrokontroler kemudian akan dikirim menuju *gateway* melalui modul LoRa RFM9x pada frekuensi 915 MHz. Data yang diterima oleh *gateway* dari *sensor node* akan dikirim menuju *cloud* untuk kemudian disimpan. Data *sensor* yang telah disimpan pada *cloud* kemudian akan diakses oleh komponen visualisasi data untuk kemudian ditampilkan data hasil pemantauan secara *realtime*.

4.2.3 Perancangan *Sensor node*

Salah satu komponen utama dalam sistem ini adalah *Sensor node* yang terdiri atas sensor, mikrokontroler, dan modul komunikasi. Pada poin sebelumnya telah disebutkan bahwa sistem yang akan dibuat menggunakan empat *sensor* yaitu *Dissolved Oxygen* (oksigen terlarut), *Ph Meter* (tingkat keasaman), *Turbidity*

(kekeruhan) dan suhu. Selain keempat *sensor* yang telah disebutkan tersebut terdapat juga modul komunikasi RFM9x yang akan dihubungkan dengan mikrokontroler Arduino Nano. Rancangan *Sensor node* dapat dilihat pada Gambar 4.3.



Gambar 4.3 Perancangan *Sensor node*

Pada Gambar 4.3 adalah *wiring* yang di digunakan pada *sensor node*. Mikrokontroler yang digunakan untuk mengolah data yang ada di *sensor node* adalah Arduino Nano. Arduino Nano terhubung dengan empat sensor dan satu modul LoRa HopeRF-RFM95. Empat sensor yang terhubung dengan Arduino Nano adalah sensor oksigen terlarut, sensor keasaman, sensor kekeruhan dan sensor suhu. Untuk menyambungkan komponen – komponen tersebut dibuat sebuah PCB agar kabel *jumper* yang digunakan tidak terlalu banyak.

4.2.3.1 Perancangan modul lora pada *sensor node*

Perancangan modul lora pada *sensor node* akan membahas tentang pinout yang digunakan. Pinout modul LoRa dengan perangkat Arduino Nano dapat dilihat pada Tabel 4.5.

Tabel 4.5 Pinout Modul HopeRF-RFM95 dengan Arduino Nano

HopeRF-RFM95	Arduino Nano
GND	GND
3.3V	3V3
DIO0	2

RESET	RST
NSS	D10
SCK	D13
MOSI	D11
MISO	D12

Tabel 4.5 diatas merupakan konfigurasi *pinout* modul komunikasi LoRa HopeRF-RFM95 dengan Arduino Nano. Pada penelitian ini modul LoRa menggunakan daya 3.3V lalu dihubungkan dengan pin 3V3 pada Arduino Nano. Pin DIO0 yang digunakan untuk *Interupt* pada modul LoRa dihubungkan dengan pin D2. Untuk pin MOSI (*Master Out Slave In*) dan MISO (*Master In Slave Out*) yang digunakan untuk pengiriman data pada modul LoRa hopeRF-RFM95 dihubungkan dengan pin D11 dan D12 pada Arduino Nano.

4.2.3.2 Perancangan sensor keasaman

Perancangan sensor keasaman akan membahas tentang *pinout* dan rumus yang digunakan pada penelitian agar data sensor diakuisi secara akurat. *Pinout* sensor keasaman dengan perangkat Arduino Nano dapat dilihat pada Tabel 4.6.

Tabel 4.6 Pinout modul sensor keasaman dengan Arduino Nano

pH Meter	Arduino Nano
GND	GND
VCC	5V
Analog	A0

Tabel 4.6 diatas adalah *pinout* untuk sensor keasaman dengan Arduino Nano. Sensor keasaman menghasilkan *output* berupa data analog, sehingga pin *analog* pada sensor dihubungkan ke pin A0 pada Arduino Nano. Pin dengan huruf A pada Arduino Nano berfungsi untuk menerima data berupa analog. Untuk daya yang digunakan pada sensor keasaman adalah 5V. Agar data hasil pengamatan sensor keasaman dapat diakuisisi secara akurat maka diperlukan rumus berikut:

$$pH = \sum_{i=1}^n a \times voltage / 1024 / n \times 3.5$$

Voltage dalam rumus tersebut adalah aliran listrik yang diterima oleh sensor sedangkan *n* banyaknya data yang diterima dalam satu waktu. Penjelasan dari rumus diatas adalah rata – rata pengamatan sensor analog pH akan dikalikan dengan *voltage* yaitu 5V dan dibagi dengan 1024 untuk mendapatkan nilai milivolt. Setelah itu hasil perhitungan dikalikan dengan 3,5 untuk mendapatkan nilai pH yang sebenarnya.

4.2.3.3 Perancangan sensor oksigen terlarut

Perancangan sensor oksigen terlarut akan membahas tentang *pinout* dan rumus yang digunakan pada penelitian agar data sensor diakuisi secara akurat. *Pinout* sensor oksigen terlarut dengan perangkat Arduino Nano dapat dilihat pada Tabel 4.7.

Tabel 4.7 Pinout modul sensor oksigen terlarut dengan Arduino Nano

Dissolved Oxygen	Arduino Nano
GND	GND
VCC	5V
Analog	A1

Tabel 4.7 diatas adalah *pinout* untuk sensor oksigen terlarut dengan Arduino Nano. Sensor oksigen terlarut menghasilkan *output* berupa data analog, sehingga pin *analog* pada sensor dihubungkan ke pin A1 pada Arduino Nano. Untuk daya yang digunakan pada sensor oksigen terlarut adalah 5V. Agar data hasil pengamatan sensor keasaman dapat diakuisisi secara akurat maka diperlukan rumus berikut:

$$DO\ value = \frac{Voltage}{SaturationDoVoltage} \times SaturationDoValue$$

Voltage dari rumus diatas adalah aliran listrik yang diterima oleh sensor, dari *pinout* diketahui bahwa nilai *voltage* adalah 5 volt. Nilai *SaturationDoVoltage* dalam penelitian ini sama dengan nilai *voltage* yaitu 5 volt. Nilai *SaturationDoValue* didapatkan dari hasil *sensing* dari sensor oksigen terlarut. Penjelasan dari rumus diatas adalah *voltage* akan dibagi dengan *saturation DO voltage* dan dikali dengan *saturation DO value* untuk mendapatkan nilai oksigen terlarut yang sebenarnya.

4.2.3.4 Perancangan sensor kekeruhan

Perancangan sensor kekeruhan akan membahas tentang *pinout* dan rumus yang digunakan pada penelitian agar data sensor diakuisi secara akurat. *Pinout* sensor kekeruhan dengan perangkat Arduino Nano dapat dilihat pada Tabel 4.8.

Tabel 4.8 Pinout modul sensor kekeruhan dengan Arduino Nano

Turbidity	Arduino Nano
GND	GND
VCC	5V
Analog	A2

Tabel 4.8 diatas adalah *pinout* untuk sensor kekeruhan dengan Arduino Nano. Sensor kekeruhan menghasilkan *output* berupa data analog, sehingga pin *analog* pada sensor dihubungkan ke pin A2 pada Arduino Nano. Untuk daya yang digunakan pada sensor kekeruhan adalah 5V. Agar data hasil pengamatan sensor kekeruhan dapat diakuisisi secara akurat maka diperlukan rumus berikut:

$$\text{Turbidity} = \text{Analog value} \times \left(\frac{\text{Voltage}}{1024} \right)$$

Nilai *Analog* pada rumus diatas didapatkan dari hasil *sensing* dari sensor kekeruhan. *Voltage* dari rumus diatas adalah aliran listrik yang diterima oleh sensor, dari *pinout* diketahui bahwa nilai *voltage* adalah 5 volt. Rumus diatas adalah untuk mengkonversi pembacaan analog (rentang 0 sampai 1023) ke dalam nilai *voltage* (0 sampai 5V). Keterangan dari rumus diatas adalah nilai analog hasil pembacaan sensor dikalikan dengan nilai milivolt yang didapatkan dari hasil bagi *voltage* dengan 1024.

4.2.3.5 Perancangan sensor suhu

Perancangan sensor suhu akan membahas tentang *pinout* yang digunakan pada penelitian agar data sensor diakuisi secara akurat. *Pinout* sensor suhu dengan perangkat Arduino Nano dapat dilihat pada Tabel 4.9.

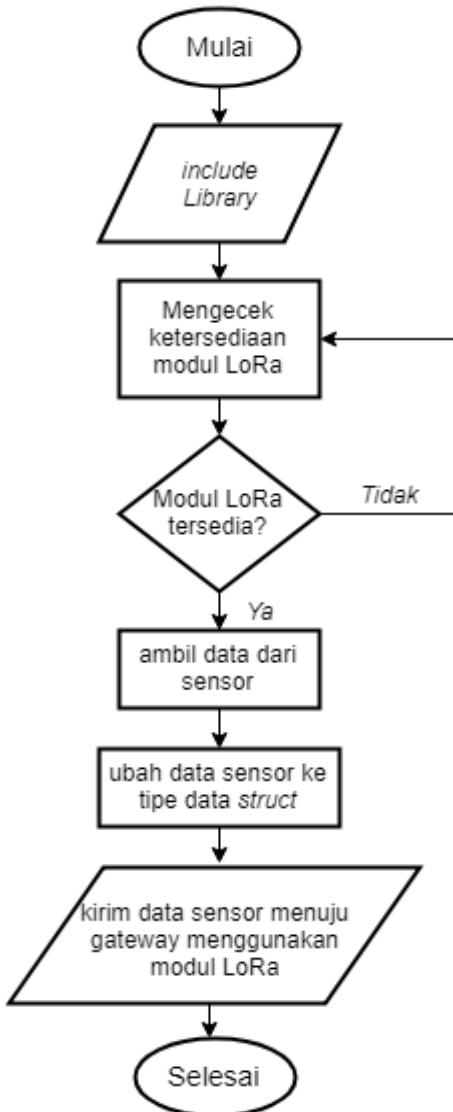
Tabel 4.9 Pinout sensor suhu (*temperature*) dengan Arduino Nano

Temperature	Arduino Nano
Ground (Hitam)	GND
VCC (Merah)	5V
Output (Kuning)	D3

Tabel 4.9 diatas adalah *pinout* untuk sensor suhu dengan Arduino Nano. Sensor suhu menghasilkan *output* berupa data digital, sehingga kabel *output* (kuning) pada sensor dihubungkan ke pin D3 pada Arduino Nano. Pin pada Arduino Nano yang berawalan huruf D digunakan untuk menerima data sensor digital. Untuk daya yang digunakan pada sensor oksigen terlarut adalah 5V.

4.2.3.6 Perancangan alur kerja sensor node

Setelah semua komponen terpasang sesuai dengan *pinout* yang telah ditentukan, maka di dalam Arduino Nano akan ditanamkan sebuah kode program yang telah dibuat pada Arduino IDE dengan bahasa pemrograman Arduino. *Library* pada Arduino Nano yang digunakan untuk memudahkan dalam menggunakan modul LoRa adalah *RH_RF95*. Untuk mengetahui alur kerja dari sisi *sensor node*, maka perlu dilakukan perancangan alur kerja *sensor node*. *Flowchart* untuk perancangan alur kerja *sensor node* dapat dilihat pada Gambar 4.4.

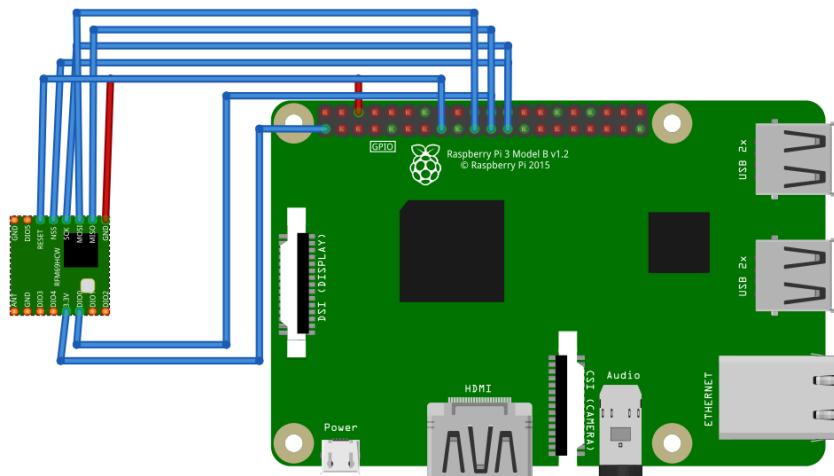


Gambar 4.4 Flowchart alur kerja *sensor node*

Keterangan dari Gambar 4.4 diatas adalah pertama – tama *include library* RH_RF95 pada Arduino Nano. Setelah itu melakukan pengecekan ketersediaan modul LoRa pada Arduino Nano, karena untuk dapat berkomunikasi dengan gateway modul LoRa harus tersedia. Jika modul LoRa tersedia *sensor node* akan mengambil data dari sensor yang telah terpasang pada *sensor node*. Data sensor yang diterima kemudian akan diubah ke dalam bentuk tipe data *struct*. Lalu data sensor akan dikirimkan ke gateway menggunakan modul LoRa HopeRF-RFM95 dan frekuensi yang dipakai adalah 915 MHz.

4.2.4 Perancangan *Gateway*

Gateway pada penelitian ini sangat diperlukan karena peran *gateway* adalah sebagai perantara *sensor node* dan *cloud server*. Perancangan *gateway* pada penelitian ini mengacu pada penelitian yang dilakukan oleh (Arijuddin, et al., 2018) dengan kode program yang dimodifikasi. Perancangan *Gateway* ditunjukkan pada Gambar 4.5.



Gambar 4.5 Arsitektur Gateway

Sumber: Arijuddin (2018)

Pada Gambar 4.5 adalah arsitektur *gateway* yang telah dibuat oleh (Arijuddin, et al., 2018). Peneliti memilih *gateway* ini karena struktur perangkat keras yang dibutuhkan untuk mendukung penelitian ini adalah sama, yaitu Raspberry Pi dan modul LoRa. Modul komunikasi LoRa yang digunakan pada *gateway* adalah HopeRF-RFM95 dengan menggunakan frekuensi 915 MHz. Modul LoRa dan Raspberry Pi dihubungkan dengan kabel *jumper female to female*. Pinout yang digunakan dapat dilihat pada Tabel 4.10.

Tabel 4.10 Pinout modul LoRa HopeRF-RFM95 dengan Raspberry Pi

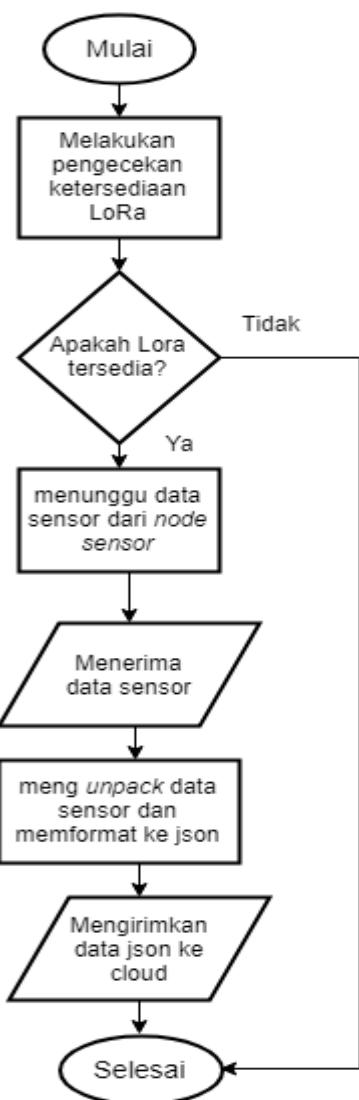
Modul LoRa HopeRF-RFM9x	Raspberry Pi
GND	GND [5]
DIO0	GPIO 25 [22]
3.3V	3.3V [1]
MISO	GPIO 9 [21]
MOSI	GPIO 10 [19]
SCK	GPIO11 [23]
NSS	GPIO 7 [26]
RESET	GPIO 22 [15]

Sumber: Arijuddin (2018)

Tabel 4.10 menunjukkan pinout modul LoRa HopeRF-RFM95 dengan Raspberry Pi. Daya yang dibutuhkan modul LoRa adalah 3.3V maka dihubungkan ke pin 1 Raspberry Pi yaitu pin untuk daya 3.3V. Untuk *interrupt* (DIO0) pada modul LoRa dihubungkan pada pin 22 Raspberry Pi. Untuk dapat mengirimkan data, Pin MISO

(Master In Slave Out) dan MOSI (Master Out Slave In) pada modul LoRa dihubungkan dengan pin 21 dan 19.

Setelah semua pin terpasang dengan benar maka akan ditanamkan sebuah program dengan bahasa Python pada Raspberry Pi. Untuk dapat menggunakan modul LoRa dengan Raspberry Pi maka dibutuhkan suatu *library* Python yaitu rf95. *Library rf95* ini digunakan untuk beberapa kebutuhan yaitu diantaranya melakukan konfigurasi LoRa, menentukan frekuensi LoRa, menerima data dari LoRa, dan mengirim data LoRa. *Library* lain yang dibutuhkan yaitu *library struct, json, dan http.client*. *Library struct* digunakan untuk melakukan *unpack* data *byte* yang dikirim melalui LoRa. *Library json* digunakan untuk melakukan format data yang akan dikirim ke *cloud server*. *Library httplib* digunakan untuk membuat koneksi ke *cloud server* menggunakan protokol http. Alur kerja dari *gateway* dapat dilihat pada Gambar 4.6.



Gambar 4.6 *Flowchart* Alur kerja *Gateway*

Pada Gambar 4.6 ditunjukkan *flowchart* alur kerja *gateway*. Pertama-tama *gateway* melakukan pengecekan modul lora, apakah modul lora sudah terpasang dengan benar atau belum. Jika modul LoRa sudah terpasang dengan benar akan ada indikator yang menampilkan bahwa modul LoRa telah terpasang. Setelah dilakukan pengecekan modul LoRa, *gateway* menunggu data sensor dari *sensor node*. Setelah data sensor dapat diterima, *gateway* akan melakukan proses *unpack* menggunakan *library struct*. *Unpack* adalah proses untuk mengubah data yang bertipe data *byte* ke tipe data yang diinginkan, dalam kasus ini tipe data yang dipakai adalah float. Setelah berhasil di *unpack* data sensor kemudian di format ke dalam bentuk *JSON* sesuai dengan format pengiriman data pada *cloud server*. Setelah data di format menjadi *JSON*, data akan dikirim ke alamat *cloud server* dengan menggunakan protokol HTTP.

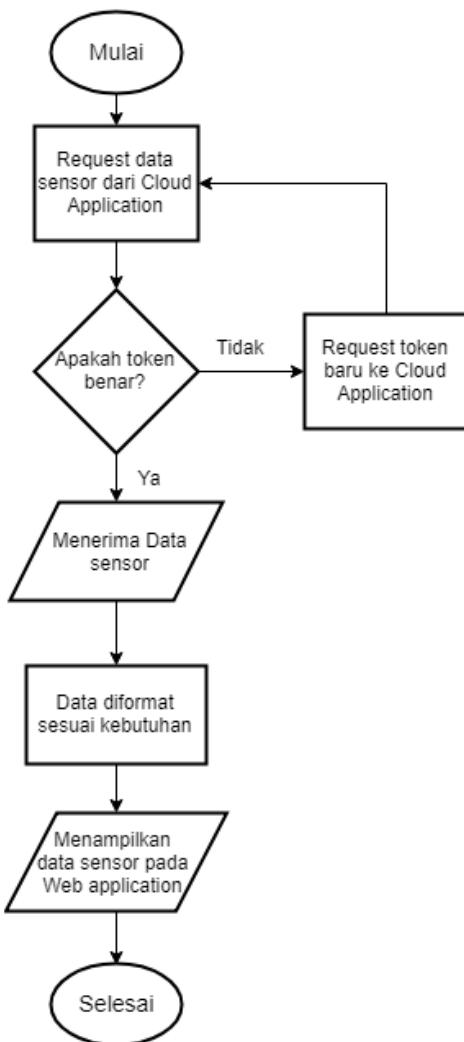
4.2.5 Perancangan Komponen Visualisasi Data

Pada penelitian ini komponen visualisasi data yang digunakan adalah aplikasi berbasis web. Bahasa yang digunakan dalam aplikasi ini adalah HTML, CSS dan Javascript. Tujuan dari aplikasi yang dibuat adalah untuk merepresentasikan hasil *monitoring* yang telah disimpan dalam *cloud* dalam bentuk grafik dan tabel. Perancangan aplikasi berbasis web dapat dilihat pada Gambar 4.7.



Gambar 4.7 Rancangan aplikasi

Untuk mendapatkan data dari *cloud*, aplikasi harus terkoneksi dengan internet. Lalu aplikasi mengirim request kepada *cloud* dengan mengirimkan juga JWT token yang telah di *generate* oleh *cloud* untuk proses autentikasi. Setelah proses autentikasi berhasil *cloud* akan mengirimkan *response* berupa data dengan format *JSON*. Perancangan alur kerja *Web application* ditunjukkan pada Gambar 4.8.



Gambar 4.8 alur kerja komponen visualisasi data

Pada Gambar 4.8 menunjukkan alur kerja dari *web application*. Yang dilakukan pertama adalah *web application* melakukan *request* data sensor yang berada di *cloud server* dengan mengirimkan JWT token. *Cloud server* lalu mengecek apakah token yang dikirim benar atau tidak. Jika token yang diterima *cloud server* salah, maka akan dilakukan proses *request* token baru ke *cloud server*. Jika token benar maka data sensor akan diterima *web application* dan kemudian akan diformat sesuai kebutuhan yaitu dalam bentuk grafik dan tabel. Data sensor yang sudah diformat kemudian akan ditampilkan ke dalam *web application* dalam bentuk grafik dan tabel.

4.2.6 Perancangan Data Sensor

Perancangan data sensor dilakukan agar data yang digunakan dalam penelitian ini sama. Terdapat dua data yang digunakan pada penelitian ini yaitu *struct* dan *JSON*.

4.2.6.1 Perancangan Struct

Struct digunakan sebagai representasi data untuk pengiriman data dari *sensor node* ke *gateway*. Selain itu, *Struct* digunakan sebagai mekanisme *packing* data sensor oksigen terlarut, keasaman, kekeruhan dan suhu yang memiliki karakteristik ukuran data yang kecil. Perancangan *struct* dalam penelitian ini ditunjukkan pada Tabel 4.11.

Tabel 4.11 Perancangan data struct

Tipe Data	Nama Variabel	Keterangan
<i>float</i>	<i>doValue</i>	Variabel digunakan untuk menyimpan data hasil pemantauan sensor <i>dissolved oxygen</i> berupa tipe data <i>float</i>
<i>float</i>	<i>phValue</i>	Variabel digunakan untuk menyimpan data hasil pemantauan sensor <i>pH meter</i> berupa tipe data <i>float</i>
<i>float</i>	<i>voltage</i>	Variabel digunakan untuk menyimpan data hasil pemantauan sensor <i>turbidity</i> berupa tipe data <i>float</i>
<i>float</i>	<i>Celsius</i>	Variabel digunakan untuk menyimpan data hasil pemantauan sensor suhu berupa tipe data <i>float</i>
<i>float</i>	<i>Fahrenheit</i>	Variabel digunakan untuk menyimpan data hasil pemantauan sensor suhu berupa tipe data <i>float</i>

4.2.6.2 Perancangan JSON

JSON digunakan sebagai representasi pengiriman data dari *gateway* menuju *cloud server*. Format JSON dipilih karena *cloud server* yang digunakan pada penelitian ini hanya menerima data dalam bentuk format JSON (Pratama, et al., 2018). Data yang dikirim adalah data sensor oksigen terlarut, keasaman, kekeruhan dan suhu. Perancangan data JSON dalam penelitian ini ditunjukkan pada Tabel 4.12.

Tabel 4.12 Perancangan data JSON

Tipe Data	Nama Variabel	Keterangan
<i>number</i>	<i>oxyg</i>	Variabel ini digunakan untuk menyimpan data hasil pemantauan sensor <i>dissolved oxygen</i> berupa tipe data <i>float</i>

<i>number</i>	<i>acid</i>	Variabel ini digunakan untuk menyimpan data hasil pemantauan sensor <i>pH meter</i> berupa tipe data <i>float</i>
<i>number</i>	<i>turb</i>	Variabel ini digunakan untuk menyimpan data hasil pemantauan sensor <i>turbidity</i> berupa tipe data <i>float</i>
<i>number</i>	<i>oxyg</i>	Variabel ini digunakan untuk menyimpan data hasil pemantauan sensor suhu berupa tipe data <i>float</i>
<i>string</i>	<i>utc_time</i>	Variabel ini digunakan untuk menyimpan data waktu pengiriman dilakukan

4.2.7 Perancangan Pengujian

Perancangan pengujian dilakukan agar pengujian yang dilakukan sesuai dengan kebutuhan yang telah didefinisikan pada bab analisis kebutuhan sistem. Perancangan yang dilakukan meliputi perancangan pengujian fungsional dan pengujian non fungsional.

4.2.7.1 Perancangan Pengujian Fungsional

Pengujian fungsional dilakukan pada penelitian ini bertujuan untuk memastikan sistem dibuat sesuai tujuan awalnya. Pengujian yang dilakukan didasari oleh kebutuhan fungsional yang telah dijelaskan pada subbab analisis kebutuhan sistem.

Tabel 4.13 Skenario Pengujian Fungsional

No	Kode	Kebutuhan Fungsional	Skenario Pengujian
1	KF-01	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan oksigen terlarut dari sensor <i>dissolved oxygen</i> .	<ol style="list-style-type: none"> 1. Sensor <i>Dissolved Oxygen</i> terpasang ke <i>Sensor node</i>. 2. <i>Arduino Nano</i> terpasang ke sumber daya. 3. Mengambil data sensor <i>Dissolved Oxygen</i>. 4. Menampilkan data sensor yang diterima dalam serial monitor
2	KF-02	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan kadar cahaya dalam	<ol style="list-style-type: none"> 1. Sensor <i>Turbidity</i> terpasang ke <i>Sensor node</i>. 2. <i>Arduino Nano</i> terpasang ke sumber daya.

		air dari sensor <i>turbidity</i> .	<ul style="list-style-type: none"> 3. Mengambil data sensor <i>Turbidity</i>. 4. Menampilkan data sensor yang diterima dalam serial monitor
3	KF-03	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan keasaman air dari sensor <i>ph meter</i> .	<ul style="list-style-type: none"> 1. Sensor <i>PH meter</i> terpasang ke <i>Sensor node</i>. 2. <i>Arduino Nano</i> terpasang ke sumber daya. 3. Mengambil data sensor <i>PH meter</i>. 4. Menampilkan data sensor yang diterima dalam serial monitor.
4	KF-04	<i>Sensor node</i> mampu mengakuisisi data hasil pemantauan suhu air dari sensor suhu.	<ul style="list-style-type: none"> 1. Sensor suhu terpasang ke <i>Sensor node</i>. 2. <i>Arduino Nano</i> terpasang ke sumber daya. 3. Mengambil data sensor suhu. 4. Menampilkan data sensor yang diterima dalam serial monitor
5	KF-05	<i>Sensor node</i> mampu melakukan <i>packing</i> data berupa <i>struct</i> dari sensor <i>dissolved oxygen</i> , <i>turbidity</i> , <i>ph meter</i> , dan suhu.	<ul style="list-style-type: none"> 1. <i>Sensor node</i> tersambung ke sumber daya 2. Sensor <i>dissolved oxygen</i>, <i>ph meter</i>, <i>turbidity</i> dan suhu sudah terpasang pada mikrokontroler. 3. Mikrokontroler melakukan <i>packing</i> data sensor yang telah terpasang menggunakan <i>struct</i>.
6	KF-06	<i>Sensor node</i> dapat melakukan pengiriman data menuju <i>gateway</i> menggunakan protokol LoRa.	<ul style="list-style-type: none"> 1. <i>Sensor node</i> tersambung ke sumber daya. 2. Modul komunikasi LoRa RFM9X sudah terpasang pada <i>sensor node</i>. 3. <i>Gateway</i> menampilkan data yang diterima dari <i>sensor node</i>.
7	KF-07	<i>Gateway</i> mampu melakukan mekanisme <i>unpack</i> data <i>struct</i> dan mengubahnya ke format JSON.	<ul style="list-style-type: none"> 1. <i>Gateway</i> terpasang ke sumber daya. 2. <i>Gateway</i> terhubung dengan modul LoRa.

			<ol style="list-style-type: none"> 3. <i>Gateway</i> melakukan <i>unpacking</i> data yang diterima dari <i>sensor node</i> berupa <i>struct</i>. 4. <i>Gateway</i> mengubah hasil <i>unpacking struct</i> menjadi format JSON.
8	KF-08	<i>Gateway</i> dapat mengirimkan data JSON menuju <i>cloud server</i>	<ol style="list-style-type: none"> 1. <i>Gateway</i> terpasang ke sumber daya. 2. <i>Gateway</i> terhubung dengan modul LoRa. 3. <i>Gateway</i> tersambung ke internet. 4. <i>Gateway</i> mengunggah data JSON ke <i>cloud server</i>. 5. Menampilkan data yang telah terunggah pada <i>dashboard administrator cloud server</i>
9	KF-09	<i>Web Application</i> dapat menampilkan data yang didapat dari REST API <i>cloud server</i> .	<ol style="list-style-type: none"> 1. <i>Web application</i> melakukan <i>request</i> ke REST API <i>cloud server</i>. 2. Menampilkan data yang di dapat ke dalam bentuk grafik dan tabel

4.2.7.2 Perancangan Pengujian Kinerja

Pengujian kinerja bertujuan untuk mengetahui kemampuan sistem dalam menerima beban sistem. Pengujian ini dilakukan dengan menggunakan jarak 50m, 100m, 200m, dan 400m. Jarak tersebut dipilih karena perangkat yang digunakan optimal pada jarak tersebut dan jika lebih dari 400 meter, paket yang berhasil terkirim hanya 5% sampai 10% dari total paket yang terkirim. Selain variasi jarak, pengujian ini juga dilakukan dengan variasi waktu *interval* 15 detik, 30 detik, dan, 30 detik. *Interval* waktu yang dipilih adalah untuk menguji bagaimana kerja sistem apabila *interval* waktu yang digunakan berbeda – beda. Pengujian yang akan dilakukan menggunakan parameter *successful rate*.

Parameter *successful rate* digunakan untuk menguji tingkat keberhasilan perangkat *gateway* dalam menerima data sensor dari *sensor node* lalu meneruskannya menuju *cloud server*.

BAB 5 IMPLEMENTASI SISTEM

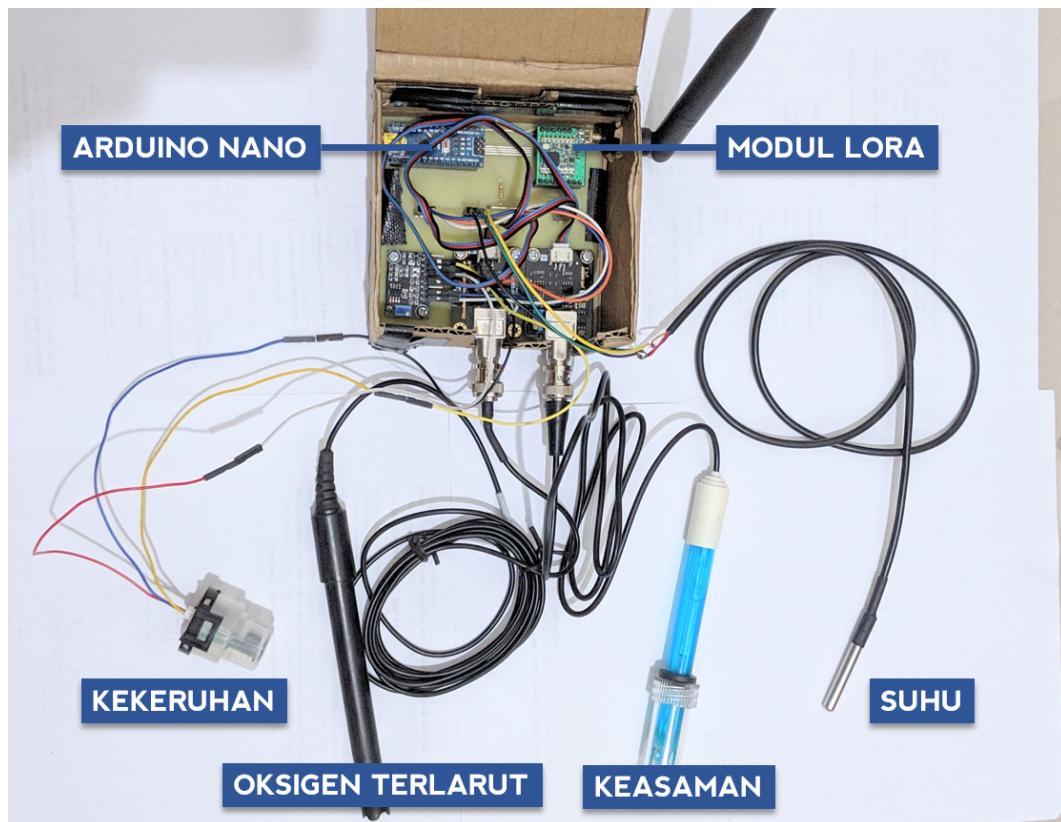
Bab ini menjelaskan tentang perancangan dan implementasi sistem. Perancangan perangkat keras sistem dilakukan dengan membuat skematik untuk masing masing perangkat. Implementasi sistem yang dilakukan adalah implementasi *sensor node*, *gateway* dan *web application*.

5.1 Implementasi Sistem

Bagian ini akan menjelaskan tentang implementasi sistem mulai dari implementasi *sensor node*, implementasi *gateway*, dan implementasi *application* berdasarkan analisis kebutuhan yang telah dilakukan sebelumnya.

5.1.1 Implementasi *Sensor node*

Sensor node pada penelitian ini menggunakan empat sensor yaitu *Dissolved Oxygen* (oksidigen terlarut), *Ph Meter* (keasaman), *Turbidity* (kekeruhan) dan suhu. Selain empat sensor tersebut, di dalam *sensor node* juga terdapat mikrokontroler arduino nano dan modul komunikasi LoRa HopeRF-RFM95. Setelah semua komponen tersedia yang dilakukan selanjutnya adalah menghubungkan semua komponen sesuai dengan rancangan yang telah dibuat. Hasil dari rancangan *sensor node* ditampilkan pada Gambar 5.1.



Gambar 5.1 Implementasi *Sensor node*

Setelah *sensor node* terpasang dengan benar maka selanjutnya adalah menanamkan kode program pada perangkat Arduino Nano. Perangkat lunak yang digunakan untuk menanamkan kode program di dalam Arduino Nano adalah Arduino IDE dengan bahasa Arduino. Ada beberapa *Library* yang harus di *include* untuk setiap komponen yang terpasang pada Arduino Nano yaitu *RH_RF95* untuk modul LoRa, *Onewire* dan *DallasTemperature* untuk sensor suhu, dan *avr/pgmspace* untuk sensor oksigen terlarut. *Pseudocode* untuk *sensor node* dapat dilihat pada Tabel 5.1.

Tabel 5.1 Pseudocode Sensor node

1	INCLUDE library
2	DEFINE dissolved oxygen sensor pin
3	DEFINE temperature sensor pin
4	DEFINE ph meter sensor pin
5	DEFINE turbidity sensor pin
6	READ dissolved oxygen sensor data
7	READ temperature sensor data
8	READ ph meter sensor data
9	READ turbidity sensor data
10	PACK sensors data into struct data type
11	IF lora initialized:
12	SET frequency
13	SET Tx Power
14	IF lora server available:
15	PRINT message
16	ELSE:
17	PRINT message
18	SEND sensor data as struct using LoRa
19	ELSE:
20	PRINT message
21	EXIT

Tabel 5.1 adalah *pseudocode* yang diimplementasikan pada *sensor node*. Pertama – tama dilakukan *include library* yang dibutuhkan oleh komponen agar dapat berkomunikasi dengan perangkat Arduino Nano. Selanjutnya dilakukan inisialisasi pin untuk setiap sensor yang dipakai. Inisialisasi pin pada setiap sensor mengacu pada subbab perancangan *sensor node* yaitu sensor keasaman pada pin A0, sensor oksigen terlarut pada pin A1, sensor kekeruhan pada pin A2, dan sensor suhu pada pin D3. Lalu dilakukan pembacaan data setiap sensor berdasarkan mekanisme masing – masing sensor karena cara membaca satu data sensor dengan sensor lainnya sangat berbeda. Setelah data sensor berhasil diambil, data sensor kemudian akan disimpan di dalam *struct*. Selanjutnya dilakukan pengecekan ketersediaan modul LoRa. Jika modul LoRa terpasang dengan benar maka akan ditampilkan pesan bahwa modul LoRa tersedia. Setelah lora berhasil dideteksi selanjutnya dilakukan inisialisasi *frequency* dan *tx power* dari LoRa. Pada penelitian ini modul LoRa yang digunakan adalah HopeRF-RFM95 dengan frekuensi 915 MHz sehingga *frequency* di dalam kode program diberi nilai 915 MHz juga. Selanjutnya dilakukan proses pengecekan ketersediaan LoRa *server*. Dalam sistem ini yang dimaksud Lora *server* adalah *gateway*. Lalu dilakukan pengiriman data sensor berupa tipe *struct* menggunakan fungsi *send()* dari *library RH_RF95*.

5.1.1.1 Implementasi data struct

Implementasi data *struct* dibuat berdasarkan perancangan yang telah dibuat pada bab perancangan. *Struct* digunakan sebagai mekanisme *packing* data sensor oksigen terlarut, keasaman, kekeruhan dan suhu yang memiliki karakteristik ukuran data yang kecil. Format data *struct* ditunjukkan pada Tabel 5.2.

Tabel 5.2 Implementasi data struct

1	...
2	struct dataSend
3	{
4	float phValue, voltage, doValue, Celsius, Fahrenheit;
5	};
6	...

Tabel 5.2 menunjukkan semua tipe data sensor berupa *float* dan berisi variabel *phValue*, *voltage*, *doValue*, *Celsius* dan *Farenheit*. Keterangan dari variabel tersebut adalah *phValue* untuk menyimpan data hasil pemantauan sensor keasaman, *voltage* untuk sensor kekeruhan, *doValue* untuk sensor oksigen terlarut, *Celsius* dan *Farenheit* untuk menyimpan data dari sensor suhu.

5.1.1.2 Implementasi sensor oksigen terlarut

Pada implementasi sensor oksigen terdapat tiga tahap implementasi kode yaitu inisialisasi pinout, pembacaan data pemantauan, dan packing hasil pemantauan ke dalam format *struct*. Implementasi sensor oksigen terlarut pada mikrokontroler dapat dilihat pada Tabel 5.3.

Tabel 5.3 Implementasi sensor oksigen terlarut

1	//inisialisasi pinout
2	#define DoSensorPin A1
3	...
4	
5	
6	
7	//membaca data hasil pemantauan oksigen terlarut pada fungsi
8	setup()
9	pinMode(DoSensorPin, INPUT);
10	readDoCharacteristicValues();
11	...
12	
13	
14	//hasil akuisi data dari sensor oksigen di packing ke dalam
15	variabel struct kirimData
16	kirimData.doValue = pgm_read_float_near(&SaturationValueTab[0] +
17	(int)(SaturationDoTemperature + 0.5) * averageVoltage /
	SaturationDoVoltage;

Tabel 5.2 menunjukkan potongan kode program Arduino untuk implementasi sensor oksigen terlarut. Kode program dimulai dengan inisialisasi pin yang dipakai yaitu A1. Pada fungsi *setup()* dilakukan pemanggilan fungsi *readDoCharacteristicValues()* untuk mengakuisi data hasil *sensing* oleh

sensor oksigen terlarut. Setelah data sensor berhasil diakuisisi, selanjutnya dilakukan *packing* data sensor oksigen terlarut ke dalam format *struct*. Pada baris 15 dilakukan implementasi rumus untuk menghitung nilai oksigen terlarut. Rumus yang digunakan yaitu `doValue = Voltage / SaturationDoVoltage * SaturationDoValue.`

5.1.1.3 Implementasi sensor keasaman

Pada implementasi sensor keasaman terdapat tiga tahap implementasi kode yaitu inisialisasi pinout, pembacaan data pemantauan, dan konversi milivolt ke nilai PH. Hasil pemantauan kemudian di simpan ke dalam format *struct*. Implementasi sensor keasaman pada mikrokontroler dapat dilihat pada Tabel 5.4.

Tabel 5.4 Implementasi sensor keasaman

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16	<pre>//inisialisasi pinout #define SensorPin 0 unsigned long int avgValue; float b; int buf[10], temp; ... //membaca data hasil pemantauan keasaman pada fungsi setup() avgValue = 0; for (int i = 2; i < 8; i++) avgValue += buf[i]; float phValue = (float)avgValue * 5.0 / 1024 / 6; ... // konversi milivolt ke nilai PH dan menyimpannya ke struct kirimData.phValue = 3.5 * phValue;</pre>
---	---

Tabel 5.4 menunjukkan potongan kode program Arduino untuk implementasi sensor keasaman. Kode program dimulai dengan inisialisasi pin yang dipakai yaitu 0. Pada fungsi *setup()* dilakukan implementasi rumus untuk mengubah sinyal analog ke dalam *minivolt*. Rumus yang digunakan adalah `avgValue * 5.0 / 1024 / 6`. Setelah data sensor berhasil diakuisisi, selanjutnya dilakukan *packing* data sensor keasaman ke dalam format *struct*. Pada baris 16 dilakukan implementasi rumus untuk mengubah *milivolt* ke dalam nilai pH. Rumus yang digunakan adalah `kirimData.phValue = 3.5 * phValue.`

5.1.1.4 Implementasi sensor kekeruhan

Pada implementasi sensor kekeruhan terdapat tiga tahap implementasi yaitu inisialisasi pinout, pembacaan data pemantauan, dan konversi hasil pembacaan analog ke nilai voltase. Hasil pemantauan kemudian di simpan ke dalam format *struct*. Implementasi kode sensor kekeruhan pada mikrokontroler dapat dilihat pada Tabel 5.5.

Tabel 5.5 Implementasi sensor kekeruhan

1 2 3 4	<pre>//inisialisasi pinout float sensorValue = analogRead(A2); ...</pre>
------------------	--

5	// konversi hasil pembacaan analog ke dalam nilai voltase dan menyimpannya ke struct
6	7 kirimData.voltage = sensorValue * (5.0 / 1024.0);
8	
9	

Tabel 5.5 menunjukkan potongan kode program Arduino untuk implementasi sensor kekeruhan. Kode program dimulai dengan inisialisasi pin yang dipakai yaitu A2. Pembacaan data *sensing* dilakukan dengan memanggil fungsi `analogRead()` dan disimpan dalam variabel `sensorValue` yang beripe `float`. Setelah data sensor berhasil diakuisisi, selanjutnya dilakukan *packing* data sensor kekeruhan ke dalam format *struct*. Pada baris 8 dilakukan implementasi rumus untuk mengubah hasil pembacaan analog ke dalam nilai voltase. Rumus yang digunakan adalah `sensorValue * (5.0 / 1024.0)`.

5.1.1.5 Implementasi sensor suhu

Sub bab ini akan membahas implementasi sensor suhu pada perangkat mikrokontroler arduino. Terdapat tiga tahap implementasi yaitu inisialisasi pinout, pembacaan data pemantauan, dan penyimpanan data hasil pemantauan ke dalam *struct*. Hasil pemantauan kemudian di simpan ke dalam format *struct*. Implementasi sensor suhu pada mikrokontroler dapat dilihat pada Tabel 5.6.

Tabel 5.6 Implementasi sensor suhu

1	//inisialisasi pinout
2	#include <OneWire.h>
3	#include <DallasTemperature.h>
4	#define ONE_WIRE_BUS 3
5	OneWire oneWire(ONE_WIRE_BUS);
6	DallasTemperature sensors(&oneWire);
7	
8	
9	...
10	
11	// menyimpan hasil pembacaan sensor suhu ke struct
12	sensors.requestTemperatures();
13	// payload temperature
14	kirimData.Celsius = sensors.getTempCByIndex(0);
15	kirimData.Fahrenheit = sensors.toFahrenheit(kirimData.Celsius);
16	

Tabel 5.6 menunjukkan potongan kode program Arduino untuk implementasi sensor suhu. Kode program dimulai dengan inisialisasi pin yang dipakai yaitu D3. Pembacaan data *sensing* dilakukan dengan memanggil fungsi `sensor.requestTemperature()`. Setelah data sensor berhasil diakuisisi, selanjutnya dilakukan *packing* data sensor suhu ke dalam format *struct*. Pada baris 14 dilakukan penyimpanan data sensor berupa suhu dalam satuan *celcius*, sedangkan pada baris 15 dilakukan penyimpanan data sensor berupa suhu dalam satuan *farenheit*.

5.1.1.6 Implementasi modul LoRa

Sub bab ini akan membahas implementasi modul LoRa pada perangkat mikrokontroler Arduino. Terdapat tiga tahap implementasi kode program untuk modul LoRa yaitu *import library*, inisialisasi pin, pengecekan modul LoRa tersedia atau tidak, inisialisasi frekuensi, dan pengiriman data. Implementasi modul LoRa pada mikrokontroler dapat dilihat pada Tabel 5.7.

Tabel 5.7 Implementasi modul LoRa

1	// import library
2	#include <RH_RF95.h>
3	#include <Arduino.h>
4	RH_RF95 rf95;
5	float frequency = 915.0;
6	
7	...
8	// set pin
9	pinMode(13, OUTPUT);
10	
11	//lorasetup
12	Serial.println("NODE SENSOR");
13	while (!rf95.init()) {
14	Serial.println("LoRa radio init failed");
15	while (1);
16	}
17	Serial.println("LoRa radio init OK!");
18	
19	// Setup ISM frequency
20	rf95.setFrequency(frequency);
21	Serial.print("Set Freq to: "); Serial.println(frequency);
22	
23	// Setup Power,dBm
24	rf95.setTxPower(13);
25	
26	// loop()
27	rf95.send((uint8_t *)&kirimData, sizeof(struct dataSend));
28	
29	

Tabel 5.7 menunjukkan potongan kode program Arduino untuk implementasi modul LoRa HopeRF RFM95W. Kode program dimulai dengan *import library*. Library yang digunakan yaitu *RH_RF95*. Kemudian dilakukan inisialisasi pin yang dipakai yaitu 13. Selanjutnya pada baris 14 hingga 16 dilakukan pengecekan apakah modul LoRa sudah terpasang dengan mikrokontroler atau belum. Lalu dilakukan inisialisasi frekuensi LoRa dengan nilai 915 MHz dengan memanggil fungsi `rf95.setFrequency(frequency)`. Lalu pada baris 28 memanggil fungsi `send()` untuk melakukan pengiriman data melalui modul LoRa. Data yang dikirim melalui LoRa berupa *struct*. Data yang dikirim melalui LoRa bertipe `uint8_t`.

5.1.2 Implementasi *Gateway*

Implementasi *Gateway* pada penelitian ini menggunakan Raspberry Pi sebagai mikrokomputer dan HopeRF-RFM95 sebagai modul komunikasi LoRa. Kedua komponen tersebut dihubungkan sesuai dengan *pinout* yang telah ditentukan

pada subbab perancangan *gateway*. Hasil Implementasi perangkat keras *gateway* dapat dilihat pada Gambar 5.2.



Gambar 5.2 Implementasi Perangkat Keras *Gateway*

Setelah perangkat keras *gateway* berhasil terpasang dengan benar maka selanjutnya adalah menanamkan kode program pada Raspberry Pi. Bahasa yang digunakan dalam penelitian ini adalah Python. Beberapa *library* dibutuhkan agar sistem yang dibuat dapat berjalan adalah *library rf95*, *struct*, *json* dan *HTTP.client*. *Library rf95* berfungsi untuk menjalankan fungsi – fungsi yang ada pada modul LoRa seperti `recv()`, `send()` dan `set_frequency()`, *Library struct* untuk melakukan *packing* dan *unpacking* data yang diterima dari *sensor node*, *library json* untuk melakukan *formating* data ke dalam bentuk json, dan *library HTTP.client* untuk melakukan komunikasi ke cloud melalui HTTP. *Pseudocode* untuk *gateway* dapat dilihat pada Tabel 5.8.

Tabel 5.8 Pseudocode Gateway

1	IMPORT library
2	DEFINE LoRa pin
3	IF LoRa is not available :
4	PRINT message "RF95 not found"
5	EXIT
6	ELSE :
7	PRINT message "RF95 LoRa mode ok"
8	SET LoRa frequency
9	SET LoRa tx power
10	WHILE true :
11	RECEIVE Lora Packet
12	ASSIGN temperature sensor data to TEMP variable
13	ASSIGN ph meter sensor data to ACID variable
14	ASSIGN dissolved oxygen sensor data to OXYG variable
15	ASSIGN turbidity sensor data to TURB variable
16	
17	UNPACK sensor datas
18	
19	ASSIGN TEMP float values to TEMP variable
20	ASSIGN ACID float values to ACID variable
21	ASSIGN OXYG float values to OXYG variable
22	ASSIGN TURB float values to TURB variable
23	
24	FORMAT sensor datas to JSON
25	
26	SEND TEMP, OXYG, ACID, and TURB values to cloud using HTTP

Tabel 5.8 adalah *pseudocode* dari *gateway* yang digunakan pada penelitian ini. Kode program diawali dengan melakukan *import library* yang dibutuhkan yaitu *rf95*, *struct*, *json*, dan *HTTP.client*. Lalu lakukan inisialisasi pin untuk modul LoRa seperti yang telah dilakukan pada subbab perancangan *gateway*. Selanjutnya melakukan pengecekan apakah modul LoRa tersedia atau tidak. Jika modul LoRa tidak tersedia maka akan mencetak keluaran “*RF95 not found*”. Jika modul LoRa tersedia maka akan menampilkan pesan “*RF95 LoRa mode ok*” lalu LoRa akan di tentukan *frequency* dan *tx power* (daya transmisi). *Frequency* LoRa pada penelitian ini menggunakan frekuensi 915 MHz dan daya transmisi 13 sesuai dengan frekuensi dan daya transmisi yang ditentukan pada *sensor node*. Selanjutnya akan dilakukan penerimaan paket menggunakan fungsi *recv()* pada *library rf95*. Paket yang berhasil diterima kemudian dipecah menjadi 4 bagian untuk masing – masing data sensor. Data sensor yang diterima dari *sensor node* masih berbentuk byte, oleh karena itu data akan di *unpack* dengan menggunakan *library struct* agar data berbentuk byte di *format* ke dalam bentuk *float*. Setelah dilakukan *unpacking* selanjutnya data sensor di kumpulkan dan di format ke dalam bentuk *json*. Setelah data sensor sudah berbentuk *json*, selanjutnya dilakukan pengiriman data sensor menuju *cloud server* menggunakan *library HTTP.client*.

5.1.2.1 Implementasi data JSON

JSON digunakan sebagai representasi pengiriman data dari *gateway* menuju *cloud application*. Data yang dikirim adalah data sensor oksigen terlarut, keasaman, kekeruhan dan suhu. Format JSON dalam penelitian ini ditunjukkan pada Tabel 5.9.

Tabel 5.9 Implementasi format JSON

```
1 "sensors" : [
2     {
3         "label" : "TEMP",
4         "value" : [
5             [temp, utc_time]
6         ]
7     },
8     {
9         "label" : "ACID",
10        "value" : [
11            [acid, utc_time]
12        ]
13    },
14    {
15        "label" : "TURB",
16        "value" : [
17            [turb, utc_time]
18        ]
19    },
20    {
21        "label" : "OXYG",
22        "value" : [
23            [oxyg, utc_time]
24        ]
25    }
26 ],
```

Tabel 5.9 menunjukkan format JSON berisi variabel *sensors* yang menyimpan data *array of objects*. Setiap *object* di dalam array terdapat label dan value. Label menandakan data sensor apa yang akan disimpan dalam value, sedangkan value adalah tempat menyimpan data sensor beserta waktu dibuatnya data tersebut.

5.1.2.2 Implementasi modul LoRa

Implementasi modul LoRa pada *gateway* terdapat dua tahapan kode program untuk modul LoRa yaitu *import library* dan inisiasi LoRa. Implementasi modul LoRa pada *gateway* dapat dilihat pada Tabel 5.10.

Tabel 5.10 Implementasi modul LoRa pada gateway

```
1 import rf95
2 ...
3
4 lora = rf95.RF95(0,0, 25,None)
5     if not lora.init(): # returns True if found
6         print("RF95 not found")
7         quit(1)
8     else:
9         print("RF95 LoRa mode ok")
10
11     lora.set_frequency(915.0)
12     lora.set_tx_power(13)
13     msg = lora.recv()
14
15
16
```

Tabel 5.10 menunjukkan potongan kode program python untuk implementasi modul LoRa HopeRF RFM95W pada *gateway*. Kode program dimulai dengan *import library rf95*. Kemudian dilakukan inisiasi objek lora dengan nilai *rf95.RF95(0, 0, 25, None)* yang berarti pin *interrupt* berada pada pin 25 dan tidak ada pin untuk *reset*. Setelah itu dilakukan pengecekan apakah modul LoRa sudah terpasang dengan mikrokomputer atau belum. Jika modul LoRa tidak terdeteksi, maka program akan berhenti. Lalu dilakukan inisialisasi frekuensi LoRa dengan nilai 915 MHz dengan memanggil fungsi *lora.set_frequency(915.0)*. Tahap selanjutnya yaitu memberi nilai daya transmisi dengan nilai 13. Baris 15 adalah fungsi LoRa untuk menerima data yang masuk dan menyimpannya dalam variabel *msg*.

5.1.2.3 Implementasi *unpacking* data sensor

Implementasi *unpacking* data diperlukan untuk melakukan translasi data *struct* menjadi *float*. Implementasi *unpacking* data sensor dapat dilihat pada Tabel 5.11.

Tabel 5.11 Implementasi *unpacking* data sensor

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22	import struct def value(msg) : packed = [chr(x) for x in msg] joinpacked = ''.join(packed) unpacked = struct.unpack('<f', joinpacked)[0] return unpacked ... msg = lora.recv() temp = msg[16:20] acid = msg[4:8] oxyg = msg[12:16] turb = msg[8:12] urutan = msg[24] waktu = time.strftime("%Y-%m-%d %H:%M:%S") temp = float(value(temp)) acid = float(value(acid)) oxyg = float(value(oxyg)) turb = float(value(turb))
---	--

Tabel 5.11 menunjukkan potongan kode program python untuk implementasi *unpacking* data. Kode program dimulai dengan melakukan *import library struct*. Tahap berikutnya membuat fungsi *value* dengan parameter *msg*. Fungsi *value* digunakan untuk mengubah data sensor yang dikirim *sensor node* yaitu data bertipe *uint8_t* menjadi data bertipe *float*. Pada baris 2 dilakukan pengubahan data integer menjadi string berdasarkan *unicode*. Lalu pada baris 4 dilakukan penggabungan array ke dalam satu string. Selanjutnya dilakukan *unpack* dengan format '*<f*' yang artinya pembacaan dimulai dari kiri dan tipe data berupa *float*.

Baris 12 hingga 15 adalah inisialisasi nilai sensor dan menyimpannya sesuai nama sensor dari data sensor yang diterima. Karena data sensor yang dikirim oleh *sensor node* berupa *uint8_t* dan direpresentasikan dalam bentuk array, maka data

yang masuk dipecah menjadi bagian – bagian sesuai dengan data sensor yang disimpan. Lalu pada baris 18 hingga 21 dilakukan *parsing* tipe data *uint8_t* menjadi float dengan memanggil fungsi *value()* yang telah didefinisikan sebelumnya.

5.1.2.4 Implementasi pengiriman data melalui protokol HTTP

Implementasi dan pengiriman data melalui protokol HTTP terdapat dua tahapan kode program yaitu *import library* dan pembuatan fungsi untuk melakukan *parsing* JSON dan mengirim data menuju *cloud server* menggunakan protokol HTTP. Kode program untuk implementasi JSON dan pengiriman data melalui protokol HTTP pada *gateway* dapat dilihat pada Tabel 5.12.

Tabel 5.12 Implementasi dan pengiriman data melalui protokol HTTP

1	import json
2	import httpplib
3	...
4	
5	
6	def send_data(temp, acid, oxyg, turb):
7	utc_time = int(time.time())
8	...
9	
10	data = {
11	"label" : "Kolam",
12	"sensors" : [
13	{
14	"label" : "TEMP",
15	"value" : [
16	[temp, utc_time]
17]
18	},
19	...
20	}
21	json_data = json.dumps(data)
22	...
23	
24	
25	ip_server = "iot.tujuhlangit.id"
26	port_server = 8080
27	conn = httpplib.HTTPConnection(ip_server, port=port_server)
28	...
29	
30	conn.request("POST", "/sensordatas/", json_data, header)
31	...
32	
33	data = conn.getresponse().read()
34	print(data)
35	
36	send_data(temp, acid, oxyg, turb)

Tabel 5.12 menunjukkan potongan kode program python untuk implementasi JSON dan pengiriman data melalui protokol HTTP pada *gateway*. Kode program dimulai dengan *import library json* dan *httpplib*. *Library json* digunakan untuk *parsing object* ke dalam format JSON, sedangkan *httpplib* digunakan untuk memanggil fungsi – fungsi umum HTTP seperti GET dan POST. Tahap selanjutnya adalah mendefinisikan fungsi *send_data*. Fungsi *send_data* memiliki empat argumen untuk masing masing sensor yaitu sensor suhu, keasaman, oksigen

terlarut, dan kekeruhan. Fungsi `send_data` digunakan untuk melakukan pengiriman data hasil pemantauan sensor menggunakan protokol HTTP dengan format JSON. Di dalam fungsi `send_data` pertama dilakukan inisialisasi *object* (*dict*) dengan nama variabel `data`. Variabel `data` menyimpan *object* yang berisi dua *key* yaitu *label* dan *sensors*. Nilai dari *key* *sensors* adalah kumpulan data sensor yang memiliki *label* dan nilai. Lalu dilakukan konversi *object* ke dalam format JSON menggunakan fungsi `json.dumps(data)`. Baris 25 dan 26 merupakan inisialisasi IP dan port dari *cloud server*. Baris 27 merupakan kode program untuk memulai koneksi HTTP kepada IP dan port yang telah ditentukan sebelumnya. Baris 30 merupakan kode untuk mengirim data *json* dengan HTTP menggunakan *method* POST. Baris 33 dan 34 adalah kode untuk menerima *response* dan menampilkannya ke terminal.

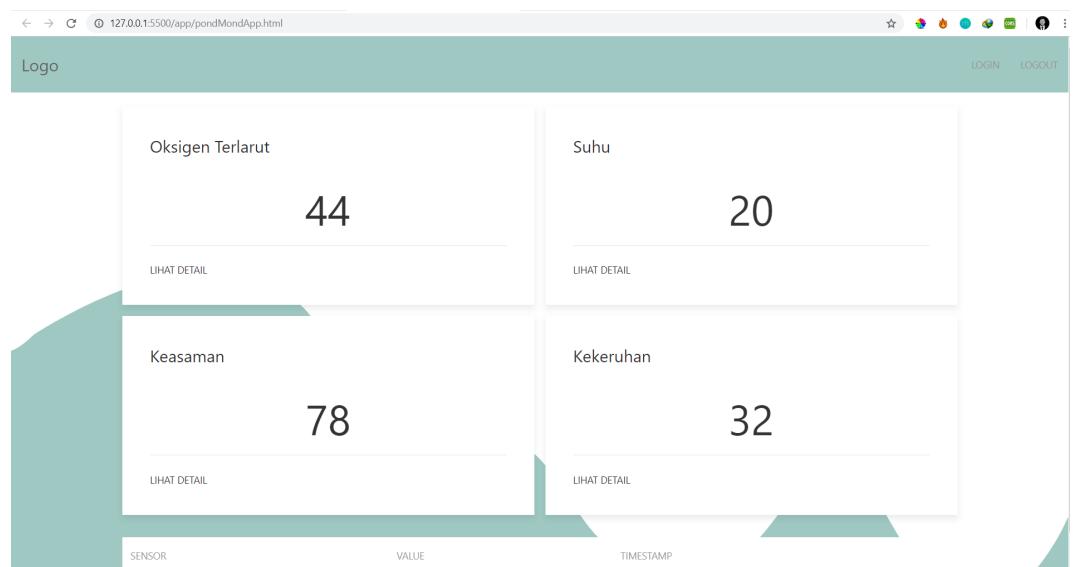
5.1.3 Implementasi Komponen Visualisasi Data

Implementasi komponen visualisasi data dalam penelitian ini menggunakan media *web application*. *Web application* dalam sistem ini menggunakan bahasa *client-side programming* yaitu HTML, CSS dan Javascript. *Web application* mendapatkan data sensor dari *cloud server* dengan menggunakan JQuery dan AJAX. AJAX digunakan untuk melakukan *request* REST API ke *cloud server*. Potongan kode program untuk mendapatkan data sensor dari *cloud server* ditunjukkan pada Tabel 5.13.

Tabel 5.13 Implementasi kode program *request REST API*

Index.js	
1 2 3 4 5 6 7 8 9 10 11 12 13 14	... \$.ajax({ type: "GET", url : "http://iot.tujuhlangit.id:8080/sensordatas/supernode/ [user-id]", dataType : 'json', headers: { "Authorization" : "Token JWT" }, success: function(data) { ... } }); ...

Potongan kode program pada Tabel 5.13 adalah untuk melakukan *request API* ke *cloud server*. Pertama dilakukan inisialisasi AJAX dengan *type* GET yang berfungsi untuk mendapatkan data dari *cloud server*. *Url* adalah *property* dari fungsi AJAX yang berisi alamat *cloud server* berada. *Data Type* adalah tipe data yang diharapkan dari *response REST API* yaitu JSON. *Headers* adalah *property* untuk menangani masalah otorisasi dalam *request* yang dilakukan karena *cloud server* hanya dapat diakses jika pengguna telah *login* dan memiliki JWT. *Success* adalah fungsi yang dijalankan apabila *request REST API* tidak ada kesalahan. Tampilan implementasi *web application* dapat dilihat pada Gambar 5.3.



Gambar 5.3 Implementasi Web Application

Gambar 5.3 adalah halaman utama dari aplikasi web pada sistem ini. Pada halaman utama terdapat informasi data sensor oksigen terlarut, suhu, keasaman dan kekeruhan. Data sensor diperoleh dari proses *request* ke *cloud server* menggunakan *REST API*.

BAB 6 PENGUJIAN SISTEM

Setelah implementasi berhasil dilakukan, selanjutnya yaitu melakukan pengujian pada sistem yang dikembangkan. Tujuan dari pengujian adalah untuk mengetahui apakah sistem yang sedang dibuat sesuai dengan kebutuhan yang telah ditentukan sebelumnya. Pengujian yang dilakukan adalah pengujian fungsional dan pengujian kinerja.

6.1 Pengujian Fungsional

Pengujian fungsional berfungsi untuk mengetahui apakah kebutuhan yang telah didefinisikan pada tahap perancangan sesuai dengan yang diimplementasikan. Pengujian fungsional yang dilakukan mengacu pada rancangan pengujian yang telah dibuat pada tahap perancangan dan implementasi.

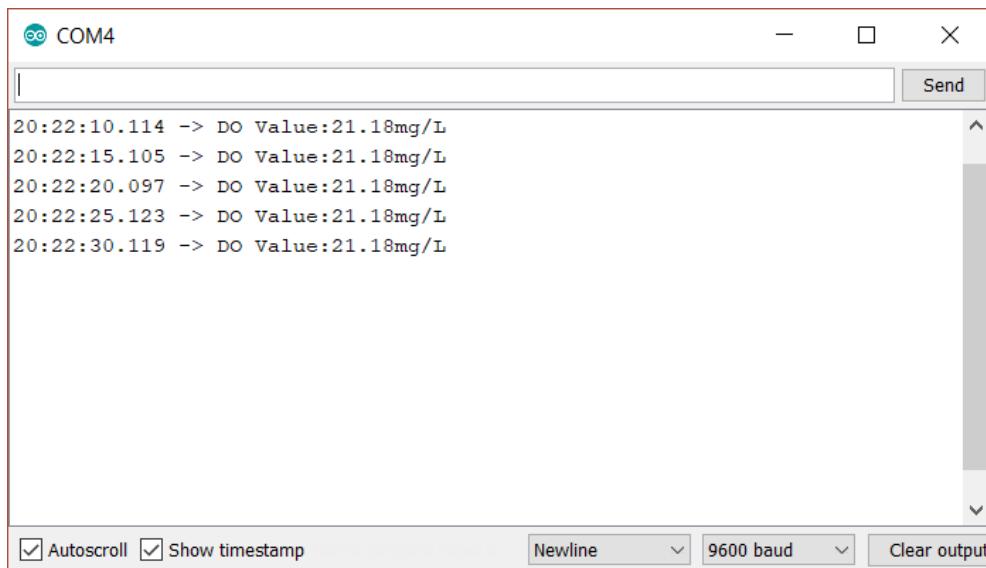
6.1.1 Pengujian Kode KF-01

Pengujian dengan kode KF-01 adalah untuk menguji kemampuan *sensor node* dalam menerima data sensor *Dissolved Oxygen*. Deskripsi lengkap mengenai pengujian dapat dilihat pada Tabel 6.1.

Tabel 6.1 Deskripsi Pengujian Kode KF-01

Kode	KF-01
Nama Kasus Uji	<i>Sensor node</i> mampu menerima data <i>sensor</i> dari sensor <i>Dissolved Oxygen</i> .
Tujuan	Mengetahui hasil pengujian kemampuan <i>sensor node</i> dalam melakukan akuisisi data hasil pemantauan sensor <i>Dissolved Oxygen</i>
Skenario Pengujian	<ol style="list-style-type: none">1. Sensor <i>Dissolved Oxygen</i> terpasang ke <i>Sensor node</i>.2. <i>Arduino Nano</i> terpasang ke sumber daya.3. Mengambil data sensor <i>Dissolved Oxygen</i>.4. Menampilkan data sensor yang diterima dalam serial monitor
Hasil yang diharapkan	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor <i>Dissolved Oxygen</i>
Hasil Pengujian	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor <i>Dissolved Oxygen</i>

Hasil dari pengujian tersebut dapat dilihat pada Gambar 6.1.



Gambar 6.1 Hasil Pengujian Sensor *Dissolved Oxygen*

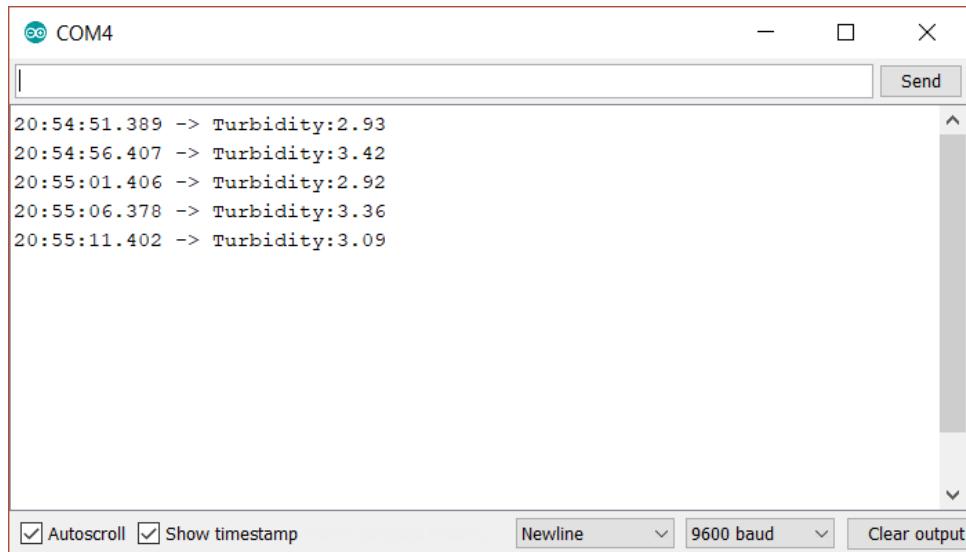
Gambar 6.1 menunjukkan bahwa data sensor *dissolved oxygen* dapat ditampilkan dalam serial monitor pada Arduino IDE. Hal tersebut menunjukkan bahwa data sensor telah berhasil didapatkan oleh *sensor node*.

6.1.2 Pengujian Kode KF-02

Pengujian dengan kode KF-02 adalah untuk menguji kemampuan *sensor node* dalam menerima data sensor *Turbidity*. Deskripsi lengkap mengenai pengujian dapat dilihat pada Tabel 6.2.

Tabel 6.2 Deskripsi Pengujian Kode KF-02

Kode	KF-02
Nama Kasus Uji	<i>Sensor node</i> mampu menerima data <i>sensor</i> dari sensor <i>Turbidity</i> .
Tujuan	Mengetahui hasil pengujian kemampuan <i>sensor node</i> dalam menerima data sensor <i>Dissolved Oxygen</i>
Skenario Pengujian	<ol style="list-style-type: none">1. Sensor <i>Turbidity</i> terpasang ke <i>Sensor node</i>.2. <i>Arduino Nano</i> terpasang ke sumber daya.3. Mengambil data sensor <i>Turbidity</i>.4. Menampilkan data sensor yang diterima dalam serial monitor
Hasil yang diharapkan	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor <i>Turbidity</i>
Hasil Pengujian	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor <i>Turbidity</i>



Gambar 6.2 Hasil Pengujian Sensor *Turbidity*

Gambar 6.2 menunjukkan bahwa data sensor *turbidity* berhasil ditampilkan dalam serial monitor pada Arduino IDE. Hal tersebut menunjukkan bahwa data sensor telah berhasil didapatkan oleh *sensor node*.

6.1.3 Pengujian Kode KF-03

Pengujian dengan kode KF-03 adalah untuk menguji kemampuan *sensor node* dalam menerima data sensor *Ph Meter*. Deskripsi lengkap mengenai pengujian dapat dilihat pada Tabel 6.3.

Tabel 6.3 Deskripsi Pengujian Kode KF-03

Kode	KF-03
Nama Kasus Uji	<i>Sensor node</i> mampu menerima data <i>sensor</i> dari sensor <i>PH meter</i> .
Tujuan	Mengetahui hasil pengujian kemampuan <i>sensor node</i> dalam menerima data sensor <i>PH Meter</i>
Skenario Pengujian	<ol style="list-style-type: none">1. Sensor <i>PH meter</i> terpasang ke <i>Sensor node</i>.2. <i>Arduino Nano</i> terpasang ke sumber daya.3. Mengambil data sensor <i>PH meter</i>.4. Menampilkan data sensor yang diterima dalam serial monitor.
Hasil yang diharapkan	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor <i>PH Meter</i>
Hasil Pengujian	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor <i>PH Meter</i>

The screenshot shows the Arduino Serial Monitor window titled 'COM4'. The window has a red border. At the top right are minimize, maximize, and close buttons. Below the title bar is a text input field with a placeholder 'Send' button. The main area displays a list of text entries. At the bottom are several control buttons: 'Autoscroll' (checked), 'Show timestamp' (checked), 'Newline' (dropdown menu), '9600 baud' (dropdown menu), and 'Clear output'.

```

21:06:27.826 -> pH:14.82
21:06:32.942 -> pH:14.82
21:06:38.036 -> pH:14.82
21:06:43.120 -> pH:14.82
21:06:48.207 -> pH:14.82
21:06:53.324 -> pH:14.82

```

Gambar 6.3 Hasil Pengujian Sensor Ph Meter

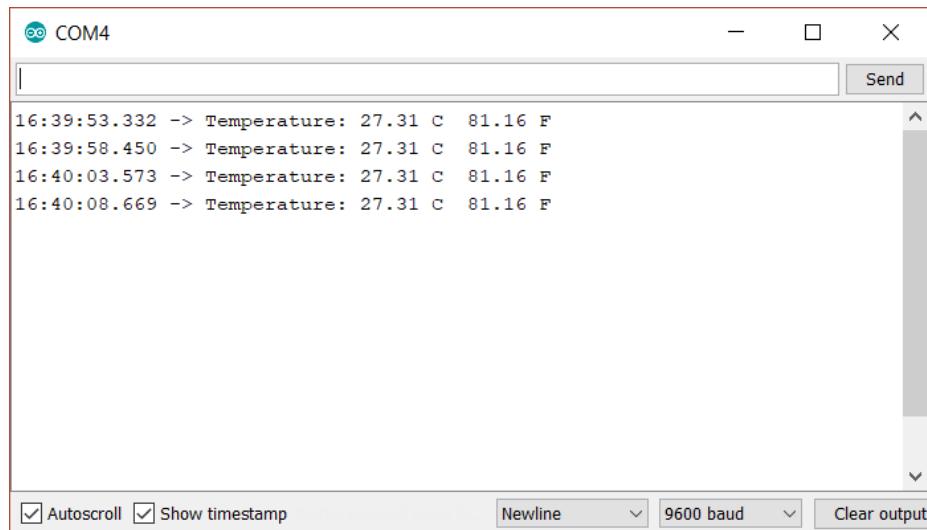
Gambar 6.3 menunjukkan bahwa data sensor *ph meter* berhasil ditampilkan dalam serial monitor pada Arduino IDE. Hal tersebut menunjukkan bahwa data sensor telah berhasil didapatkan oleh *sensor node*.

6.1.4 Pengujian Kode KF-04

Pengujian dengan kode KF-01 adalah untuk menguji kemampuan *sensor node* dalam menerima data sensor suhu. Deskripsi lengkap mengenai pengujian dapat dilihat pada Tabel 6.4.

Tabel 6.4 Deskripsi Pengujian Kode KF-04

Kode	KF-04
Nama Kasus Uji	<i>Sensor node</i> mampu menerima data <i>sensor</i> dari sensor suhu.
Tujuan	Mengetahui hasil pengujian kemampuan <i>sensor node</i> dalam menerima data sensor suhu
Skenario Pengujian	<ol style="list-style-type: none"> 1. Sensor suhu terpasang ke <i>Sensor node</i>. 2. <i>Arduino Nano</i> terpasang ke sumber daya. 3. Mengambil data sensor suhu. 4. Menampilkan data sensor yang diterima dalam serial monitor
Hasil yang diharapkan	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor Suhu
Hasil Pengujian	<i>Sensor node</i> dapat mengakuisisi data sensor dari sensor Suhu



Gambar 6.4 Hasil Pengujian Sensor Suhu

Gambar 6.4 menunjukkan bahwa data sensor suhu berhasil ditampilkan dalam serial monitor pada Arduino IDE. Hal tersebut menunjukkan bahwa data sensor telah berhasil didapatkan oleh *sensor node*.

6.1.5 Pengujian Kode KF-05

Pengujian dengan kode KF-05 adalah untuk menguji kemampuan *sensor node* dalam mengirimkan data sensor menuju *gateway* menggunakan protokol LoRa. Deskripsi lengkap mengenai pengujian Kode KF-05 dapat dilihat pada Tabel 6.5.

Tabel 6.5 Deskripsi Pengujian Kode KF-05

Kode	KF-05
Nama Kasus Uji	<i>Sensor node</i> dapat melakukan pengiriman data menuju <i>gateway</i> menggunakan protokol LoRa.
Tujuan	Mengetahui hasil pengujian kemampuan <i>sensor node</i> dalam mengirimkan data sensor menuju <i>gateway</i> menggunakan protokol LoRa.
Skenario Pengujian	<ol style="list-style-type: none"> <i>Sensor node</i> tersambung ke sumber daya. Modul komunikasi LoRa RFM9X sudah terpasang pada <i>sensor node</i>. <i>Gateway</i> menampilkan data yang diterima dari <i>sensor node</i>
Hasil yang diharapkan	<i>Sensor node</i> berhasil mengirimkan data sensor menuju <i>gateway</i> menggunakan protokol LoRa.
Hasil Pengujian	<i>Sensor node</i> berhasil mengirimkan data sensor menuju <i>gateway</i> menggunakan protokol LoRa.

```

23:05:53.327 -> Ready
23:05:53.327 -> NODE SENSOR
23:05:53.429 -> LoRa radio init OK!
23:05:53.429 -> Set Freq to: 915.00
23:05:53.463 -> Data ke: 1
23:05:53.531 -> pH:14.83
23:05:53.531 -> Turbidity:4.01
23:05:53.665 -> Temperature: 26.00 C 78.80 F
23:05:53.665 ->
23:05:58.643 -> Data ke: 2
23:05:58.749 -> pH:14.83
23:05:58.749 -> Turbidity:4.02
23:05:58.749 -> DO Value:0.00mg/L
23:05:58.887 -> Temperature: 26.00 C 78.80 F
23:05:58.887 ->
23:06:03.890 -> Data ke: 3

```

Autoscroll Show timestamp Newline 9600 baud Clear output

Gambar 6.5 Hasil Pengujian Pengiriman Data dari *Sensor node*

Gambar 6.5 menunjukkan bahwa *sensor node* berhasil dalam mengirimkan data sensor ke gateway.

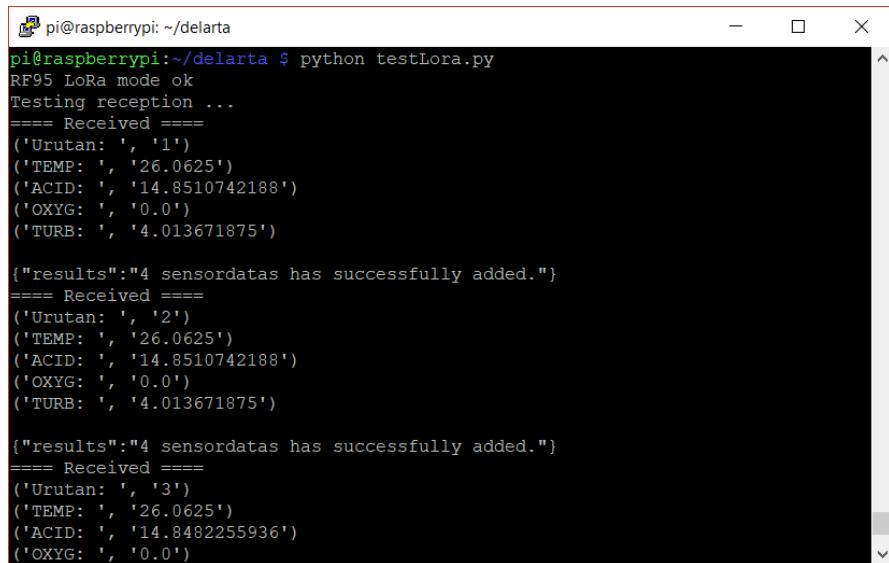
6.1.6 Pengujian Kode KF-06

Pengujian dengan kode KF-06 adalah untuk menguji kemampuan *gateway* dalam menerima data sensor dari *sensor node* dan meneruskannya menuju *cloud server*. Deskripsi lengkap mengenai pengujian kode KF-06 dapat dilihat pada Tabel 6.6.

Tabel 6.6 Deskripsi Pengujian Kode KF-06

Kode	KF-06
Nama Kasus Uji	<i>Gateway</i> mampu menerima dan meneruskan data menuju <i>cloud server</i>
Tujuan	Mengetahui hasil pengujian kemampuan <i>gateway</i> dalam menerima dan meneruskan data sensor menuju <i>cloud server</i>
Skenario Pengujian	<ol style="list-style-type: none"> 1. <i>Gateway</i> terpasang ke sumber daya. 2. <i>Gateway</i> tersambung ke internet. 3. <i>Gateway</i> menampilkan data yang diterima. 4. <i>Gateway</i> mengunggah data ke <i>cloud server</i>. 5. Menampilkan data yang telah terunggah pada <i>dashboard administrator cloud server</i>

Hasil yang diharapkan	<i>Gateway berhasil menerima dan meneruskan data menuju cloud server</i>
Hasil Pengujian	<i>Gateway berhasil menerima dan meneruskan data menuju cloud server</i>



```

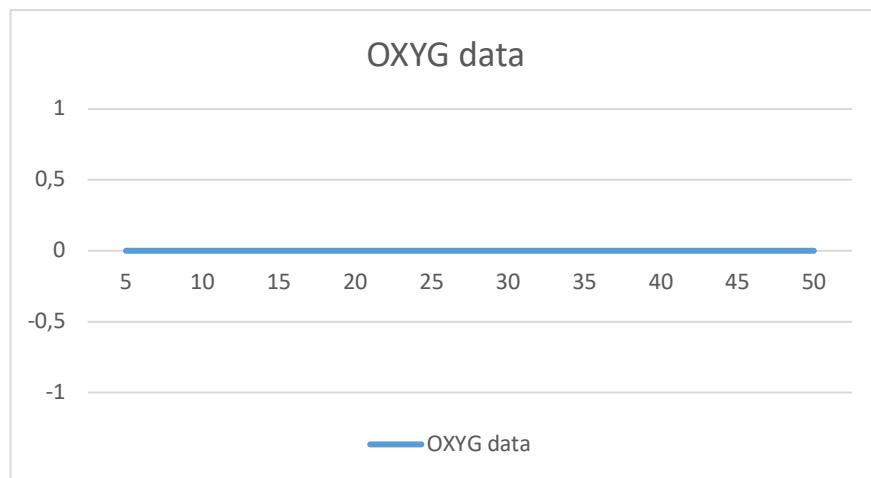
pi@raspberrypi:~/delarta
pi@raspberrypi:~/delarta $ python testLora.py
RF95 LoRa mode ok
Testing reception ...
==== Received ====
('Urutan: ', '1')
('TEMP: ', '26.0625')
('ACID: ', '14.8510742188')
('OXYG: ', '0.0')
('TURB: ', '4.013671875')

("results":"4 sensordatas has successfully added.")
==== Received ====
('Urutan: ', '2')
('TEMP: ', '26.0625')
('ACID: ', '14.8510742188')
('OXYG: ', '0.0')
('TURB: ', '4.013671875')

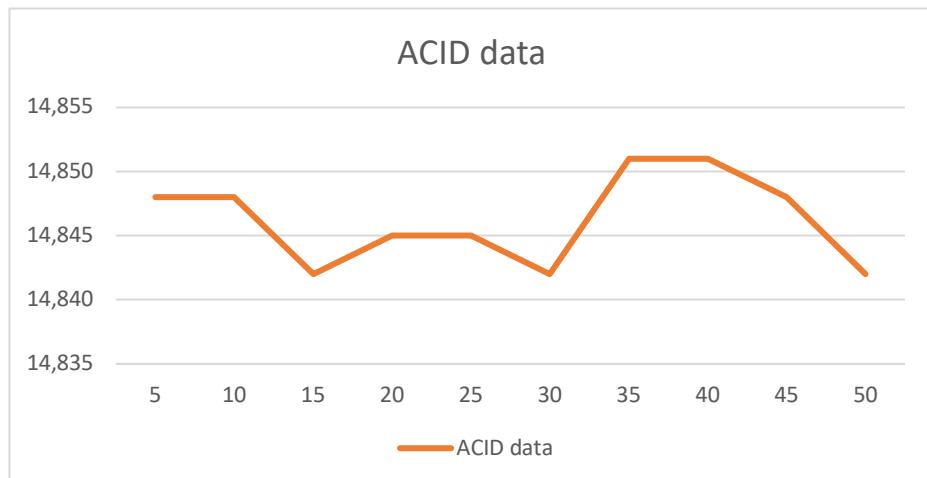
("results":"4 sensordatas has successfully added.")
==== Received ====
('Urutan: ', '3')
('TEMP: ', '26.0625')
('ACID: ', '14.8482255936')
('OXYG: ', '0.0')

```

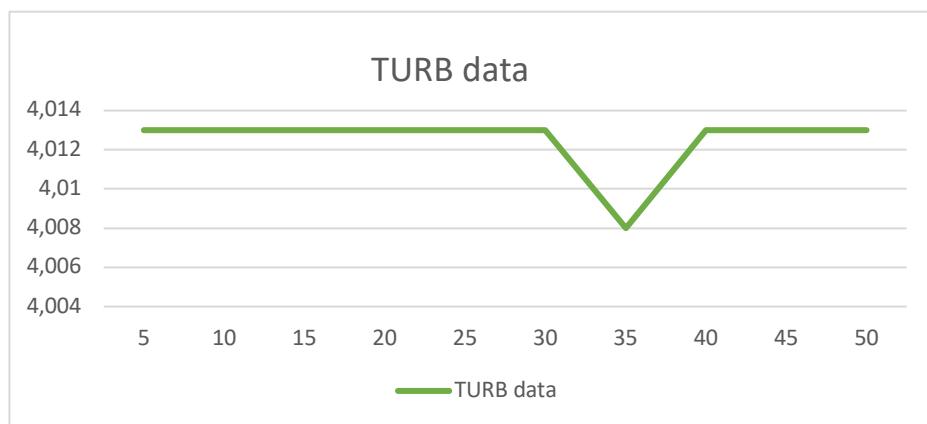
Gambar 6.6 Hasil Pengujian Penerimaan Data pada *Gateway* dan Pengiriman Data ke Cloud



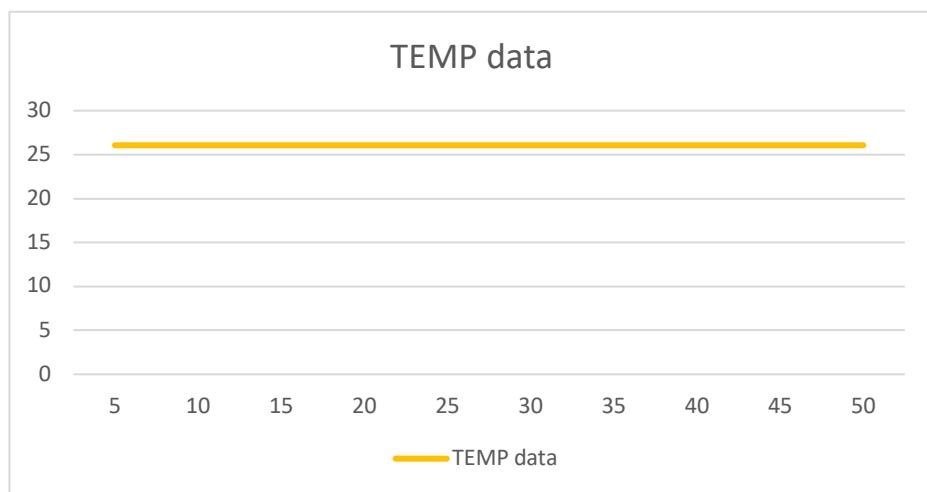
Gambar 6.7 Hasil Data Sensor *Dissolved Oxygen* diterima pada Cloud



Gambar 6.8 Hasil Data Sensor *Ph Meter* diterima pada Cloud



Gambar 6.9 Hasil Data Sensor *Turbidity* diterima pada Cloud



Gambar 6.10 Hasil Data Sensor Suhu diterima pada Cloud

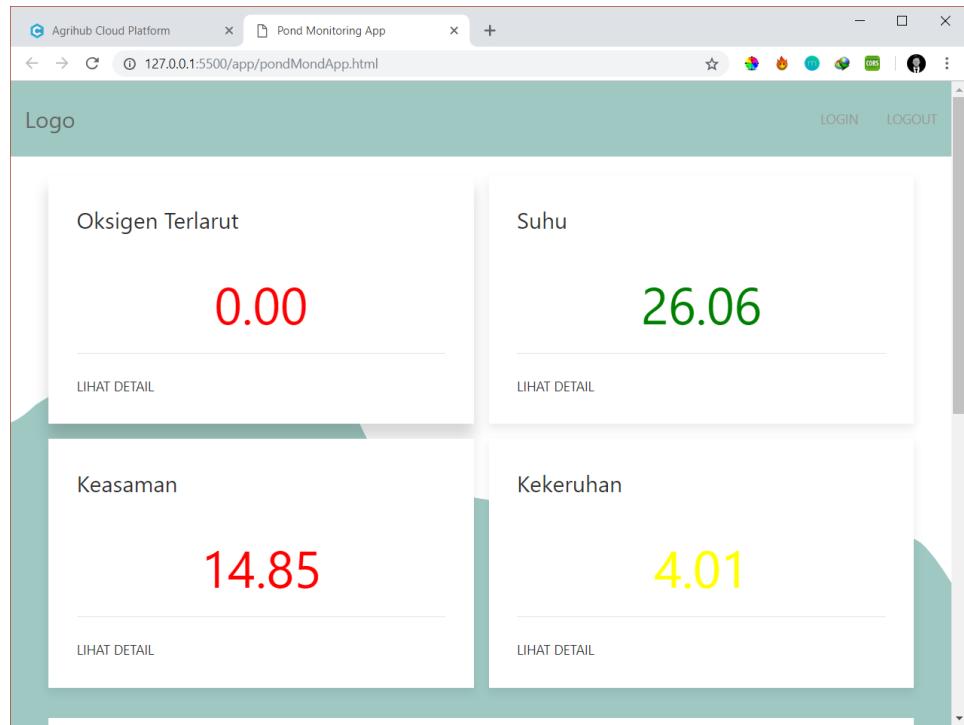
Gambar 6.6 menunjukkan tampilan gateway ketika menerima data sensor dari *sensor node* dan mengirimkannya ke *cloud server*. Gambar 6.7 sampai 6.9 adalah grafik yang ditampilkan pada *cloud server* untuk data pada setiap sensor.

6.1.7 Pengujian Kode KF-07

Pengujian dengan kode KF-07 adalah untuk menguji kemampuan *web application* menampilkan data sensor yang didapat dari *REST API cloud server*. Deskripsi lengkap mengenai pengujian kode KF-07 dapat dilihat pada Tabel 6.7.

Tabel 6.7 Deskripsi Pengujian Kode KF-07

Kode	KF-07
Nama Kasus Uji	<i>Web Application</i> dapat menampilkan data sensor yang didapat dari <i>REST API cloud server</i> .
Tujuan	Mengetahui hasil pengujian kemampuan <i>Web application</i> dalam menampilkan data yang didapat dari <i>REST API</i> milik <i>Cloud server</i>
Skenario Pengujian	<ol style="list-style-type: none"><i>Web application</i> melakukan <i>request</i> ke <i>REST API cloud server</i>.Menampilkan data yang di dapat ke dalam bentuk grafik dan tabel
Hasil yang diharapkan	<i>Web Application</i> berhasil menampilkan data sensor yang didapat dari <i>REST API cloud server</i> .
Hasil Pengujian	<i>Web Application</i> dapat menampilkan data sensor yang didapat dari <i>REST API cloud server</i> .



Gambar 6.11 Hasil Pengujian Request API Data Web Application

Gambar 6.11 menunjukkan hasil pengujian *request API* yang dilakukan oleh *web application*. Nilai yang ditampilkan pada *web application* adalah nilai data sensor yang akan di perbarui setiap 5 detik sekali.

6.1.8 Pengujian Kode KF-08

Pengujian dengan kode KF-08 adalah untuk menguji *sensor node* dalam mendeteksi perubahan kondisi air dengan menambahkan air sabun dan air panas pada lingkungan air yang diamati. Deskripsi lengkap mengenai pengujian kode KF-08 dapat dilihat pada Tabel 6.8.

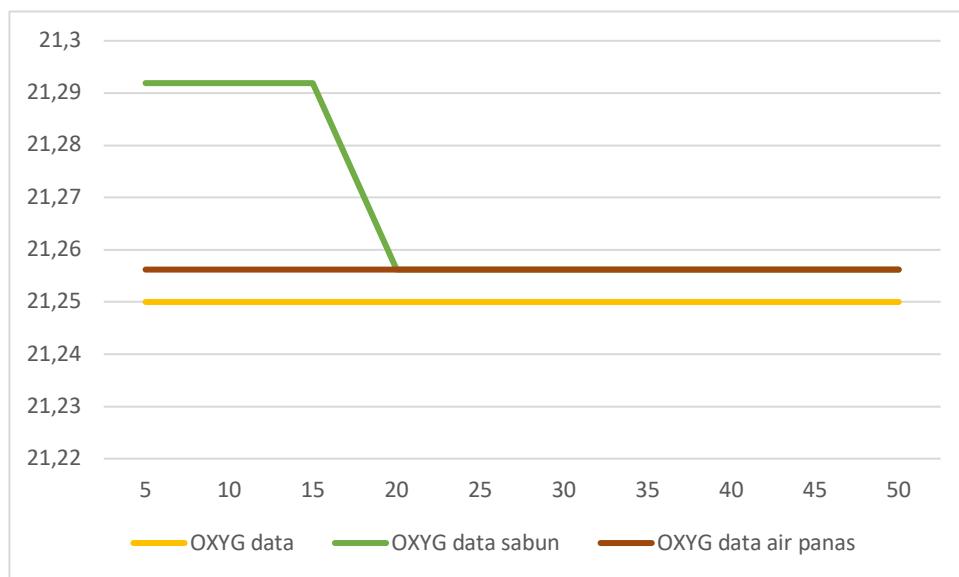
Tabel 6.8 Deskripsi Pengujian KF-08

Kode	KF-08
Nama Kasus Uji	<i>Sensor Node</i> dapat mendeteksi perubahan kondisi air.
Tujuan	Mengetahui hasil pengujian kemampuan sensor dalam mendeteksi perubahan kondisi air.
Skenario Pengujian	<ol style="list-style-type: none"> 1. <i>Sensor node</i> tersambung ke sumber daya. 2. Modul komunikasi LoRa RFM9X sudah terpasang pada <i>sensor node</i>. 3. Ditambahkan air sabun pada lingkungan air yang diamati. 4. Ditambahkan air panas pada lingkungan air yang diamati.

	5. <i>Cloud server</i> menampilkan data yang diterima dari <i>sensor node</i> .
Hasil yang diharapkan	<i>Sensor Node</i> berhasil mendeteksi perubahan kondisi air.
Hasil Pengujian	<i>Sensor Node</i> berhasil mendeteksi perubahan kondisi air.

6.1.8.1 Hasil pengujian kode KF-08 pada sensor oksigen terlarut

Grafik data sensor oksigen terlarut pada *cloud server* sebelum dan sesudah ditambahkan air sabun dan air panas ditunjukkan pada Gambar 6.12.

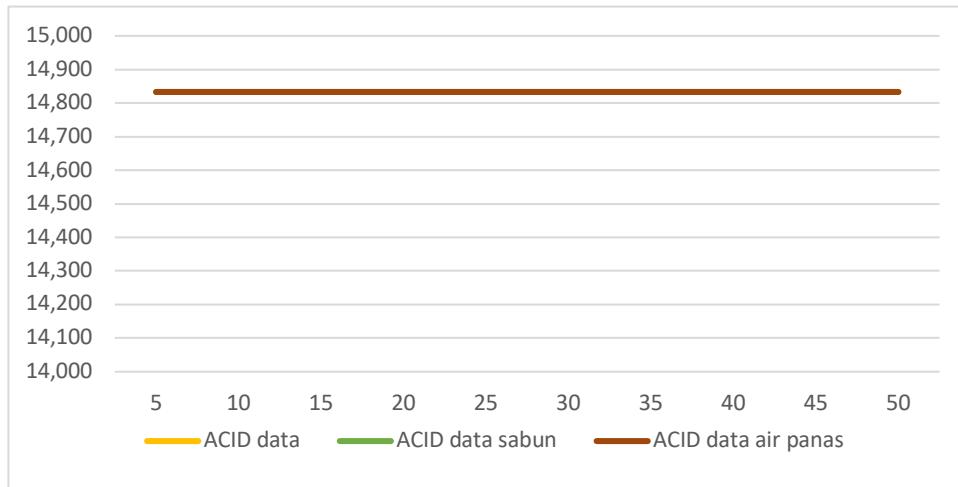


Gambar 6.12 Data sensor oksigen terlarut pada *cloud server*

Gambar 6.12 menunjukkan kondisi awal air, kondisi setelah ditambahkan air sabun, dan kondisi setelah ditambahkan air panas. Pada Gambar 6.12 menunjukkan pengaruh yang signifikan saat dilakukan penambahan air sabun. Sedangkan pada penambahan air panas pengaruhnya tidak begitu signifikan.

6.1.8.2 Hasil pengujian kode KF-08 pada sensor keasaman

Grafik data sensor keasaman yang diterima pada *cloud server* sebelum dan sesudah ditambahkan air sabun dan air panas ditunjukkan pada Gambar 6.13.

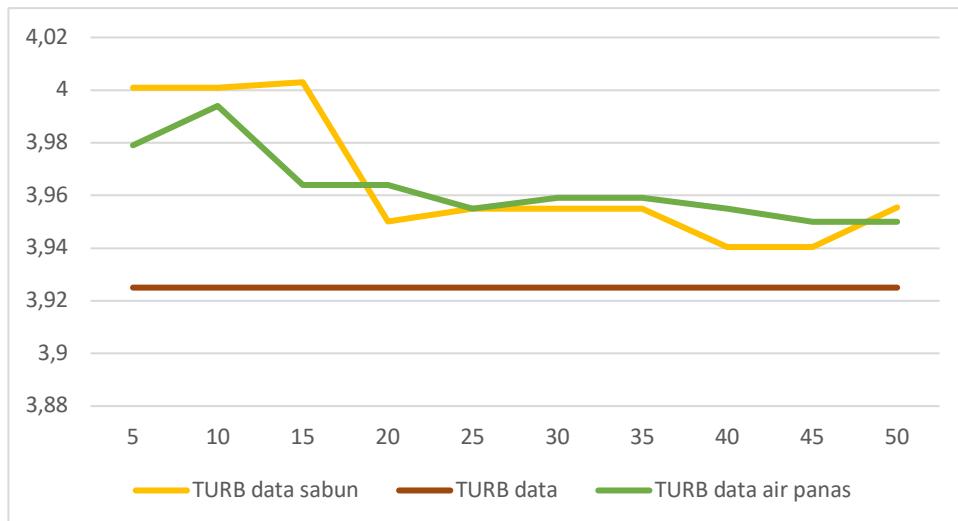


Gambar 6.13 Data sensor keasaman pada *cloud server*

Gambar 6.13 menunjukkan kondisi awal air, kondisi setelah ditambahkan air sabun, dan kondisi setelah ditambahkan air panas. Pada Gambar 6.13 menunjukkan tidak ada perubahan yang terjadi saat ditambahkan air panas dan air sabun.

6.1.8.3 Hasil pengujian kode KF-08 pada sensor kekeruhan

Grafik data sensor kekeruhan yang diterima pada *cloud server* sebelum dan sesudah ditambahkan air sabun dan air panas ditunjukkan pada Gambar 6.14.

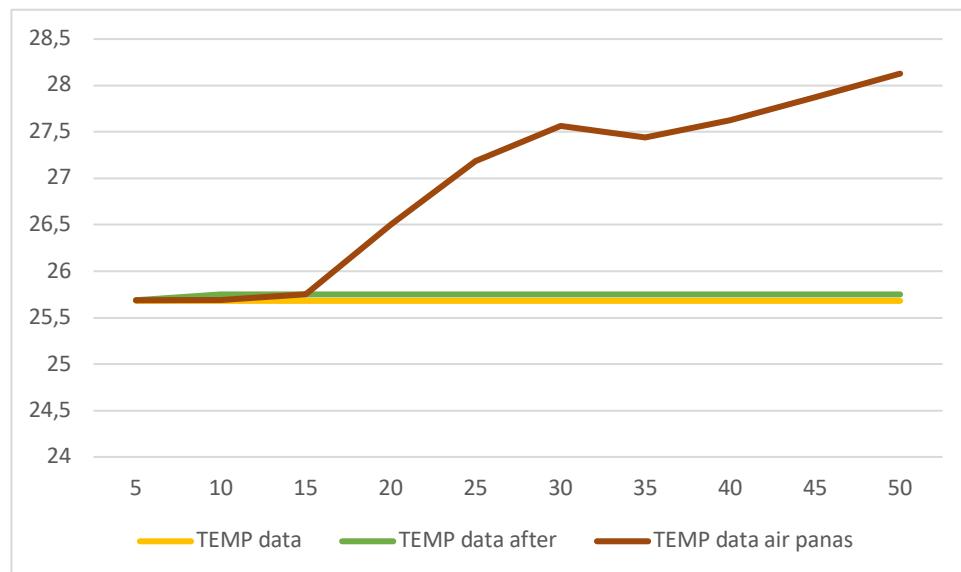


Gambar 6.14 Data sensor kekeruhan pada *cloud server*

Gambar 6.14 menunjukkan kondisi awal air, kondisi setelah ditambahkan air sabun, dan kondisi setelah ditambahkan air panas. Pada Gambar 6.14 menunjukkan pengaruh air sabun membuat air semakin keruh karena nilai sensor terus menurun. Sedangkan pada penambahan air panas juga memiliki pengaruh yang sama yaitu semakin panas suhu semakin keruh air.

6.1.8.4 Hasil pengujian kode KF-08 pada sensor suhu

Grafik data sensor suhu yang diterima pada *cloud server* sebelum dan sesudah ditambahkan air sabun dan air panas ditunjukkan pada Gambar 6.15.



Gambar 6.15 Data sensor suhu pada *cloud server*

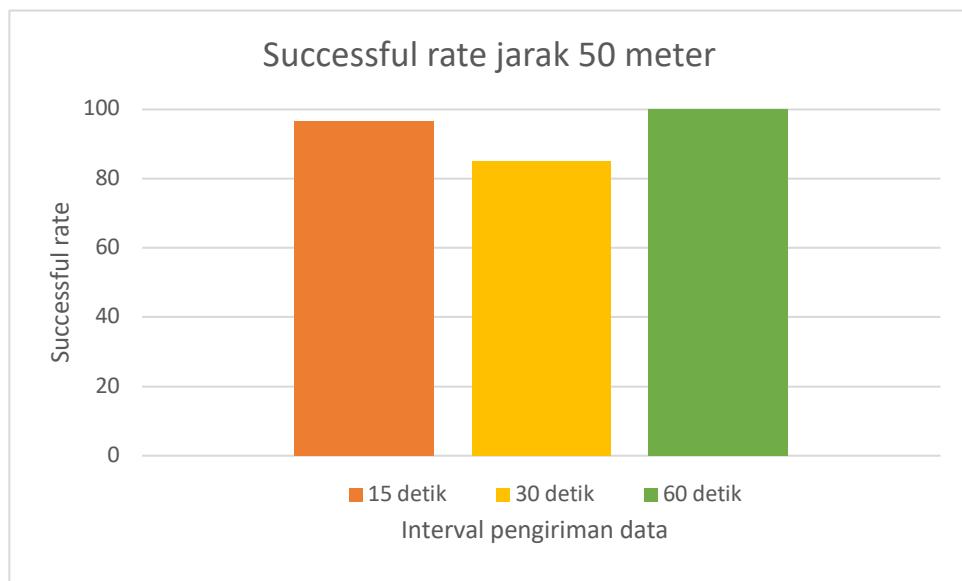
Gambar 6.15 menunjukkan kondisi awal air, kondisi setelah ditambahkan air sabun, dan kondisi setelah ditambahkan air panas. Pada Gambar 6.15 menunjukkan suhu didalam air naik ketika air panas ditambahkan. Sedangkan air sabun yang ditambahkan membuat suhu didalam air bertambah sedikit panas tetapi tidak berubah secara signifikan.

6.2 Pengujian Kinerja

Pengujian kinerja dilakukan untuk mengetahui keandalan sistem ketika melakukan fungsi pengiriman data sensor dari *sensor node* hingga *gateway*. Pengujian ini menggunakan parameter *successful rate* dengan menggunakan faktor jarak dan *interval* waktu. Pengujian dilakukan di lapangan rampal Malang karena area cukup luas dan bebas pandang. Jarak yang digunakan dalam penelitian ini adalah 50 meter, 100 meter, 200 meter, dan 400 meter. Pada pengujian ini, setiap jarak dilakukan tiga kali pengujian dengan *interval* waktu masing – masing 15 detik, 30 detik, dan 60 detik. Data sensor yang dikirim untuk masing – masing pengujian berjumlah 30 data.

6.2.1 Pengujian Jarak 50 meter

Pengujian ini dilakukan dengan menggunakan jarak 50 meter. Jarak 50 meter dipilih karena jarak tersebut jarak yang sangat terjangkau untuk LoRa. Pada jarak ini dilakukan tiga kali pengujian dengan interval waktu 15 detik, 30 detik, dan 60 detik. Hasil dari pengujian Jarak 50 meter dapat dilihat pada Gambar 6.16.

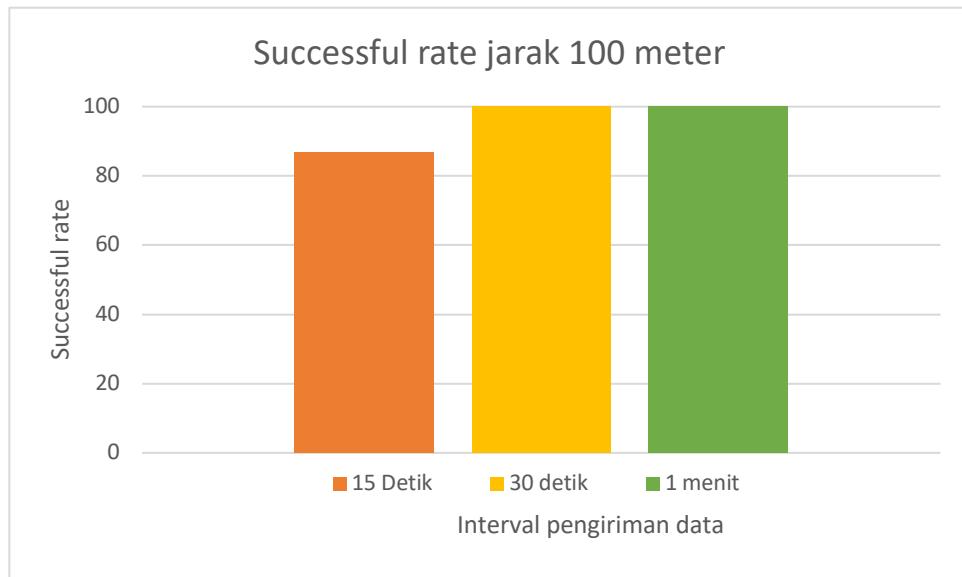


Gambar 6.16 Grafik hasil pengujian jarak 50 meter

Gambar 6.16 adalah grafik yang menunjukkan hasil dari pengujian yang dilakukan. Grafik tersebut menunjukkan pada pengujian dengan interval waktu 15 detik *successful rate* mencapai 96%. pada interval 30 detik *successful rate* mencapai 85%. Sedangkan pada interval 60 detik *successful rate* mencapai 100%. Dari data tersebut dapat disimpulkan bahwa pada jarak 50 meter interval waktu yang paling baik adalah 60 detik.

6.2.2 Pengujian Jarak 100 meter

Pengujian ini dilakukan dengan menggunakan jarak 100 meter. Pada pengujian jarak 100 meter ini dilakukan tiga kali pengujian yang sama dengan sebelumnya yaitu dengan interval waktu 15 detik, 30 detik, dan 60 detik. Hasil dari pengujian Jarak 100 meter dapat dilihat pada gambar 6.17.

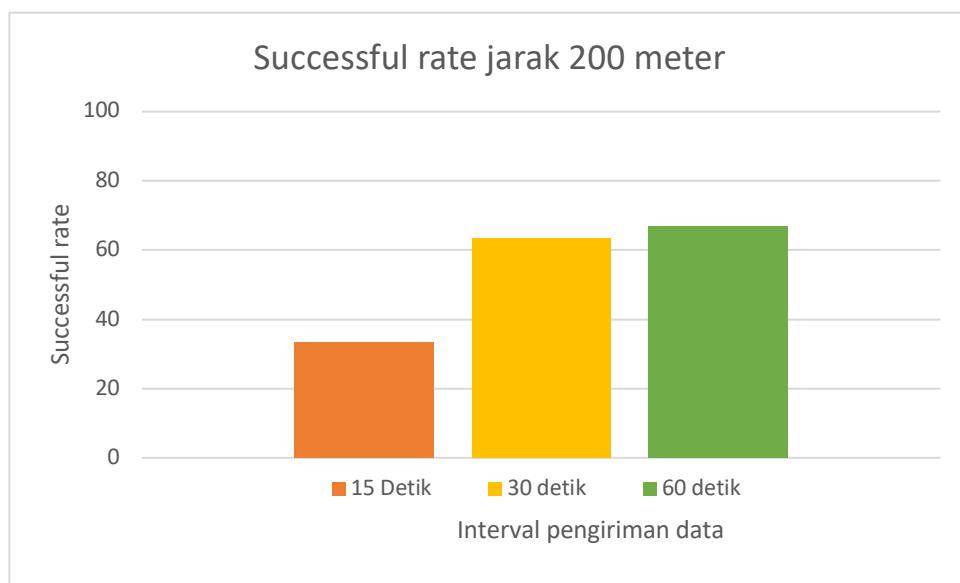


Gambar 6.17 Grafik hasil pengujian jarak 100 meter

Gambar 6.17 adalah grafik yang menunjukkan hasil dari pengujian yang dilakukan. Grafik tersebut menunjukkan pada pengujian dengan interval 15 detik *successful rate* mencapai 88%. pada interval 30 detik *successful rate* mencapai 100%. Sedangkan pada interval 60 detik *successful rate* mencapai 100%. Dari data tersebut dapat disimpulkan bahwa pada jarak 100 meter interval waktu yang paling baik adalah 30 detik dan 60 detik.

6.2.3 Pengujian Jarak 200 meter

Pengujian ini dilakukan dengan menggunakan jarak 200 meter. Pada pengujian jarak 200 meter ini dilakukan tiga kali pengujian yang sama dengan sebelumnya yaitu dengan interval waktu 15 detik, 30 detik, dan 60 detik. Hasil dari pengujian Jarak 200 meter dapat dilihat pada gambar 6.18.

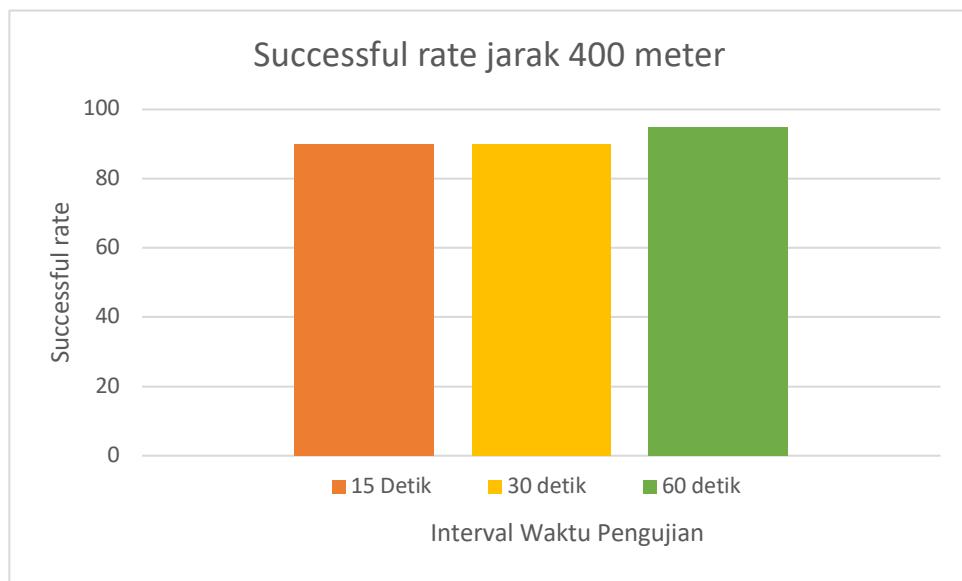


Gambar 6.18 Grafik hasil pengujian jarak 200 meter

Gambar 6.18 adalah grafik yang menunjukkan hasil dari pengujian yang dilakukan. Grafik tersebut menunjukkan pada pengujian dengan interval 15 detik *successful rate* hanya mencapai 33%. pada interval 30 detik *successful rate* mencapai 65%. Sedangkan pada interval 60 detik *successful rate* mencapai 68%. Dari data tersebut dapat disimpulkan bahwa pada jarak 200 meter interval waktu yang paling baik adalah 60 detik dengan *successful rate* 68%.

6.2.4 Pengujian Jarak 400 meter

Pengujian ini dilakukan dengan menggunakan jarak 400 meter. Pada pengujian jarak 400 meter ini dilakukan tiga kali pengujian yang sama dengan sebelumnya yaitu dengan interval waktu 15 detik, 30 detik, dan 60 detik. Hasil dari pengujian Jarak 400 meter dapat dilihat pada gambar 6.19.



Gambar 6.19 Grafik hasil pengujian jarak 400 meter

Gambar 6.19 adalah grafik yang menunjukkan hasil dari pengujian yang dilakukan. Grafik tersebut menunjukkan pada pengujian dengan interval 15 dan 30 detik *successful rate* mencapai 90%. Sedangkan pada interval 60 detik *successful rate* mencapai 95%. Dari data tersebut dapat disimpulkan bahwa pada jarak 400 meter interval waktu yang paling baik adalah 60 detik dengan *successful rate* 95%.

BAB 7 KESIMPULAN DAN SARAN

Bab ini berisi tentang kesimpulan dari penelitian yang dilakukan dan saran untuk penelitian selanjutnya.

7.1 Kesimpulan

Kesimpulan berdasarkan hasil perancangan, implementasi, dan pengujian yang dilakukan adalah sebagai berikut:

1. Sistem *monitoring* parameter fisik kolam ikan menggunakan WSN berbasis protokol LoRa terdiri dari *sensor node*, *gateway*, *cloud server* dan komponen visualisasi data. *Sensor node* berperan untuk mengakuisisi data sensor dan mengirimkannya ke *gateway* menggunakan modul LoRa. *Sensor node* terdiri dari mikrokontroler Arduino Nano, modul komunikasi LoRa, sensor oksigen terlarut, sensor keasaman, sensor kekeruhan, dan sensor suhu. *Gateway* berperan untuk menerima data yang dikirim dari *sensor node* melalui modul LoRa dan mengirimkan data ke *cloud server* menggunakan protokol HTTP. *Gateway* terdiri dari mikrokomputer Raspberry Pi dan modul komunikasi LoRa. *Cloud server* berperan untuk menyimpan data dari *gateway* dan menyediakan REST API untuk diakses oleh komponen visualisasi data. Komponen visualisasi data berperan untuk menampilkan data sensor dalam bentuk grafik dan tabel berbasis *web application*.
2. Representasi data yang digunakan untuk pengiriman data dari *sensor node* menuju *gateway* dan dari *gateway* menuju *cloud server* adalah *struct* dan JSON. *Struct* digunakan sebagai representasi data untuk mengumpulkan data hasil *sensing* dari sensor oksigen, sensor keasaman, sensor kekeruhan, dan sensor suhu pada *sensor node*. *Struct* dipilih karena memiliki karakteristik memiliki ukuran data yang kecil sehingga cocok dengan LoRa yang memiliki *datarates* yang rendah. *Struct* kemudian dikirim menuju *gateway* untuk kemudian diubah ke dalam format JSON. JSON digunakan sebagai representasi data pengiriman dari *gateway* menuju *cloud server*.
3. Mekanisme pengiriman data *sensor node* menuju *gateway* dilakukan dalam beberapa tahap yaitu akuisisi data sensor, *packing* data sensor dan *send* data dengan modul LoRa. Akuisisi data sensor dilakukan pada mikrokontroler Arduino Nano untuk mendapat hasil pemantauan dari sensor oksigen terlarut, sensor keasaman, sensor kekeruhan dan sensor suhu. Kemudian dilakukan *packing* data sensor yang berhasil diakuisi menggunakan *struct*. *Send* data menggunakan modul LoRa digunakan untuk mengirim data sensor menuju *gateway*. *Send* data dijalankan setelah semua data sensor telah di *packing*.
4. Mekanisme pengiriman data *gateway* menuju *cloud server* dilakukan dalam beberapa tahap yaitu *unpacking* data sensor, konversi data sensor ke dalam format JSON dan *send* data dengan protokol HTTP. Data sensor

yang diterima oleh *gateway* dari *node sensor* melalui protokol LoRa kemudian dilakukan proses *unpacking data*. Proses *unpacking* data sensor dilakukan dengan menggunakan *library struct* pada python. Setelah data berhasil di *unpack* selanjutnya dilakukan konversi data hasil *unpack* ke dalam format JSON menggunakan *library json*. Data JSON kemudian dikirim menuju *cloud server* menggunakan protokol HTTP.

5. Berdasarkan pengujian yang dilakukan pada jarak 50 meter, 100 meter dan 200 meter, protokol LoRa memiliki kinerja paling buruk pada jarak 200 meter dan interval 15 detik dengan *successful rate* 30%. Sedangkan kinerja paling baik pada jarak 400 meter dan pada interval waktu 60 detik dengan *successful rate* 95%.

7.2 Saran

Berdasarkan kesimpulan yang telah didapat, terdapat beberapa hal yang dapat dikembangkan untuk penelitian berikutnya. Saran peneliti untuk penelitian berikutnya adalah sebagai berikut:

1. *Sensor node* yang digunakan pada penelitian ini hanya satu buah. Diharapkan dalam penelitian berikutnya dapat menggunakan lebih dari satu *sensor node*.
2. Pengujian jarak yang dilakukan dalam penelitian ini hanya sampai 400 meter. Diharapkan selanjutnya pengujian bisa lebih dari 400 meter.
3. Parameter pengujian kinerja ditambahkan tentang *delay*, RTT dan sebagainya.

DAFTAR PUSTAKA

- Akyildyz, I., Su, W., Sankarasubramaniam & Cayirci, E., 2002. Wireless Sensor Networks: A Survey.
- Arijuddin, H., Bhawiyuga, A. & Amron, K., 2018. Pengembangan Sistem Perantara Pengiriman Data Menggunakan Modul Komunikasi LoRa dan Protokol MQTT Pada Wireless Sensor Network. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, Volume 2.
- Augustin, A., Yi, J., Claussen, T. & Townsley, W. M., 2016. A Study of LoRa: Long Range & Low Power Networks for the Internet of Things.
- Bengraine, K. & Marhaba, T. F., 2003. Using Principal Component Analysis to Monitor Spatial and Temporal Changes in Water Quality. *Journal of Hazardous Materials*, p. 179–195.
- Bhawiyuga, A. & Yahya, W., 2018. A LPWAN based Wireless Senor Node for Aquaculture Water Quality Monitoring System.
- Cario, G. et al., 2017. Long Lasting Underwater Wireless Sensors Network for Water Quality Monitoring in Fish Farms.
- Crockford, D., 2013. How JavaScript Works. In: *How JavaScript Works*. s.l.:s.n.
- Dargie, W. & Poellabauer, C., 2010. Fundamentals of Wireless Sensor Network.
- Digi International, 2007. Demystifying 802.15.4 and ZigBee. *Demystifying 802.15.4 and ZigBee*.
- Encinas, C., Ruiz, E., Cortez, J. & Espinoza, A., 2017. Design and implementation of a distributed IoT system for the monitoring of water quality in aquaculture. p. 7.
- Ira, 2014. KAJIAN KUALITAS PERAIRAN BERDASARKAN PARAMETER FISIKA DAN KIMIA DI PELABUHAN PERIKANAN SAMUDERA KENDARI SULAWESI TENGGARA. *Jurnal Ilmu Perikanan dan Sumberdaya Perairan*, p. 2.
- Karl, H. & Willig, A., 2005. *Protocols and Architectures for Wireless Sensor Network*. s.l.:s.n.
- Lora Alliance, 2015.
- Mahalik, N. P. & Kim, K., 2014. Aquaculture Monitoring and Control Systems for Seaweed and Fish Farming. *World Journal of Agricultural Research*, 2014, Vol. 2, No. 4, 176-182.
- Pratama, O. B., Bhawiyuga, A. & Amron, K., 2018. Pengembangan Perangkat Lunak IoT Cloud Platform Berbasis Protokol Komunikasi HTTP. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer*, Volume 2.
- Pule, M., Yahya, A. & Chuma, J., 2017. Wireless sensor networks: A survey on monitoring water quality. *Journal of Applied Research*.

Raspberry Pi Foundation, 2015. *About Us: Raspberry Pi Foundation*. [Online] Available at: www.raspberrypi.org [Accessed 13 May 2019].

Simbeye, D. S., Zhao, i. & Yang, S., 2014. Design and deployment of wireless sensor networks for aquaculture monitoring and control based on virtual instruments. *Computers and Electronics in Agriculture* 102 (2014).

Sung, W.-T., Chen, J.-H. & Wang, H.-C., 2014. Remote fish aquaculture monitoring system based on wireless transmission technology. p. 5.

Yani, A., 2007. *Geografi*. Jakarta: Grafindo.

LAMPIRAN A

A.1 Kode Program

```
Lora arduino.ino
//lora
#include <RH_RF95.h>
#include <Arduino.h>
RH_RF95 rf95;
float frequency = 915.0; // Change the frequency here.

///temperature
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 3

OneWire oneWire(ONE_WIRE_BUS);

DallasTemperature sensors(&oneWire);

//float Celsius = 0;
//float Fahrenheit = 0;

///DO
#include <avr/pgmspace.h>
#include <EEPROM.h>

#define DoSensorPin A1      //dissolved oxygen sensor analog output
pin to arduino mainboard
#define VREF 5000           //for arduino uno, the ADC reference is the
AVCC, that is 5000mV(TYP)
//float doValue;          //current dissolved oxygen value, unit; mg/L
float temperature = 25;    //default temperature is 25^C, you can
use a temperature sensor to read it

#define EEPROM_write(address, p) {int i = 0; byte *pp =
(byte*)&(p);for(; i < sizeof(p); i++) EEPROM.write(address+i,
pp[i]);}
#define EEPROM_read(address, p) {int i = 0; byte *pp =
(byte*)&(p);for(; i < sizeof(p); i++) pp[i]=EEPROM.read(address+i);}

#define ReceivedBufferLength 20
char receivedBuffer[ReceivedBufferLength + 1]; // store the serial
command
byte receivedBufferIndex = 0;

#define SCOUNT 30           // sum of sample point
int analogBuffer[SCOUNT]; //store the analog value in the array,
readed from ADC
int analogBufferTemp[SCOUNT];
int analogBufferIndex = 0, copyIndex = 0;

#define SaturationDoVoltageAddress 12           //the address of the
Saturation Oxygen voltage stored in the EEPROM
#define SaturationDoTemperatureAddress 16         //the address of the
Saturation Oxygen temperature stored in the EEPROM
float SaturationDoVoltage, SaturationDoTemperature;
float averageVoltage;

const float SaturationValueTab[41] PROGMEM = {           //saturation
dissolved oxygen concentrations at various temperatures
14.46, 14.22, 13.82, 13.44, 13.09,
12.74, 12.42, 12.11, 11.81, 11.53,
```

```

    11.26, 11.01, 10.77, 10.53, 10.30,
    10.08, 9.86, 9.66, 9.46, 9.27,
    9.08, 8.90, 8.73, 8.57, 8.41,
    8.25, 8.11, 7.96, 7.82, 7.69,
    7.56, 7.43, 7.30, 7.18, 7.07,
    6.95, 6.84, 6.73, 6.63, 6.53,
    6.41,
};

////PH
#define SensorPin 0           //pH meter Analog output to Arduino
Analog Input 0
unsigned long int avgValue; //Store the average value of the sensor
feedback
float b;
int buf[10], temp;

//struct data
struct dataSend
{
    float phValue = 0, voltage = 0, doValue = 0, Celsius = 0,
Fahrenheit = 0;
    int urutan = 0;
};

void setup()
{
    pinMode(13, OUTPUT);
    Serial.begin(9600);
    Serial.println("Ready"); //Test the serial monitor
    //DO
    pinMode(DoSensorPin, INPUT);
    readDoCharacteristicValues(); //read Characteristic Values
calibrated from the EEPROM

    //lorasetup
    Serial.println("NODE SENSOR");
    while (!rf95.init()) {
        Serial.println("LoRa radio init failed");
        while (1);
    }
    Serial.println("LoRa radio init OK!");

    // Setup ISM frequency
    rf95.setFrequency(frequency);
    Serial.print("Set Freq to: "); Serial.println(frequency);
    //delay(3000);
    // Setup Power, dBm
    rf95.setTxPower(13);

}

int ctData = 0;

void loop()
{
    //lora setup
    dataSend kirimData;
    ctData++;
    kirimData.urutan = ctData;
    Serial.print("Data ke: ");
    Serial.println(kirimData.urutan);
}

```

```

        for (int i = 0; i < 10; i++) //Get 10 sample value from the sensor
for smooth the value
{
    buf[i] = analogRead(SensorPin);
    delay(10);
}
for (int i = 0; i < 9; i++) //sort the analog from small to large
{
    for (int j = i + 1; j < 10; j++)
    {
        if (buf[i] > buf[j])
        {
            temp = buf[i];
            buf[i] = buf[j];
            buf[j] = temp;
        }
    }
}
avgValue = 0;
for (int i = 2; i < 8; i++) //take the average value
of 6 center sample
    avgValue += buf[i];
float phValue = (float)avgValue * 5.0 / 1024 / 6; //convert the
analog into millivolt
// payload ph
kirimData.phValue = 3.5 * phValue; //convert the
millivolt into pH value
Serial.print("pH:");
Serial.print(kirimData.phValue, 2);
Serial.println(" ");
digitalWrite(13, HIGH);
// delay(800);
digitalWrite(13, LOW);

//// turbidity
float sensorValue = analogRead(A2); // read the input on analog pin
0:
// payload turbidity
kirimData.voltage = sensorValue * (5.0 / 1024.0); // Convert the
analog reading (which goes from 0 - 1023) to a voltage (0 - 5V):
Serial.print("Turbidity:");
Serial.println(kirimData.voltage); // print out the value you
read:
//delay(500);

////DO
static unsigned long analogSampleTimepoint = millis();
if (millis() - analogSampleTimepoint > 30U) //every 30
milliseconds, read the analog value from the ADC
{
    analogSampleTimepoint = millis();
    analogBuffer[analogBufferIndex] = analogRead(DoSensorPin);
//read the analog value and store into the buffer
    analogBufferIndex++;
    if (analogBufferIndex == SCOUNT)
        analogBufferIndex = 0;
}

static unsigned long tempSampleTimepoint = millis();
if (millis() - tempSampleTimepoint > 500U) // every 500
milliseconds, read the temperature
{
    tempSampleTimepoint = millis();
    //temperature = readTemperature(); // add your temperature codes
here to read the temperature, unit:^C

```

```

}

static unsigned long printTimepoint = millis();
if (millis() - printTimepoint > 1000U)
{
    printTimepoint = millis();
    for (copyIndex = 0; copyIndex < SCOUNT; copyIndex++)
    {
        analogBufferTemp[copyIndex] = analogBuffer[copyIndex];
    }
    averageVoltage = getMedianNum(analogBufferTemp, SCOUNT) * (float)VREF / 1024.0; // read the value more stable by the median filtering algorithm
    kirimData.doValue = pgm_read_float_near( &SaturationValueTab[0] + (int)(SaturationDoTemperature + 0.5) ) * averageVoltage / SaturationDoVoltage; //calculate the do value, doValue = Voltage / SaturationDoVoltage * SaturationDoValue (with temperature compensation)
    Serial.print(F("DO Value:"));
    /// payload DO
    Serial.print( kirimData.doValue, 2);
    Serial.println(F("mg/L"));
}

if (serialDataAvailable() > 0)
{
    byte modeIndex = uartParse(); //parse the uart command received
    doCalibration(modeIndex); // If the correct calibration command is received, the calibration function should be called.
}
//temperature
sensors.requestTemperatures();
// payload temperature
kirimData.Celsius = sensors.getTempCByIndex(0);
kirimData.Fahrenheit = sensors.toFahrenheit( kirimData.Celsius);
Serial.print("Temperature: ");
Serial.print( kirimData.Celsius);
Serial.print(" C ");
Serial.print( kirimData.Fahrenheit);
Serial.println(" F");
Serial.println();

rf95.send((uint8_t *)&kirimData, sizeof(struct dataSend));
uint8_t bufrcv[RH_RF95_MAX_MESSAGE_LEN];
uint8_t len = sizeof(bufrcv);
delay(5000);
}

///DO
boolean serialDataAvailable(void)
{
    char receivedChar;
    static unsigned long receivedTimeOut = millis();
    while ( Serial.available() > 0 )
    {
        if (millis() - receivedTimeOut > 500U)
        {
            receivedBufferIndex = 0;
            memset(receivedBuffer, 0, (ReceivedBufferLength + 1));
        }
        receivedTimeOut = millis();
        receivedChar = Serial.read();
        if (receivedChar == '\n' || receivedBufferIndex == ReceivedBufferLength)
        {

```

```

        receivedBufferIndex = 0;
        strupr(receivedBuffer);
        return true;
    } else {
        receivedBuffer[receivedBufferIndex] = receivedChar;
        receivedBufferIndex++;
    }
}
return false;
}

byte uartParse()
{
    byte modeIndex = 0;
    if (strstr(receivedBuffer, "CALIBRATION") != NULL)
        modeIndex = 1;
    else if (strstr(receivedBuffer, "EXIT") != NULL)
        modeIndex = 3;
    else if (strstr(receivedBuffer, "SATCAL") != NULL)
        modeIndex = 2;
    return modeIndex;
}

void doCalibration(byte mode)
{
    char *receivedBufferPtr;
    static boolean doCalibrationFinishFlag = 0, enterCalibrationFlag = 0;
    float voltageValueStore;
    switch (mode)
    {
        case 0:
            if (enterCalibrationFlag)
                Serial.println(F("Command Error"));
            break;

        case 1:
            enterCalibrationFlag = 1;
            doCalibrationFinishFlag = 0;
            Serial.println();
            Serial.println(F(">>>Enter Calibration Mode<<<"));
            Serial.println(F(">>>Please put the probe into the saturation
oxygen water! <<<"));
            Serial.println();
            break;

        case 2:
            if (enterCalibrationFlag)
            {
                Serial.println();
                Serial.println(F(">>>Saturation Calibration Finish!<<<"));
                Serial.println();
                EEPROM_write(SaturationDoVoltageAddress, averageVoltage);
                EEPROM_write(SaturationDoTemperatureAddress, temperature);
                SaturationDoVoltage = averageVoltage;
                SaturationDoTemperature = temperature;
                doCalibrationFinishFlag = 1;
            }
            break;

        case 3:
            if (enterCalibrationFlag)
            {
                Serial.println();
                if (doCalibrationFinishFlag)

```

```

        Serial.print(F(">>>Calibration Successful"));
    else
        Serial.print(F(">>>Calibration Failed"));
    Serial.println(F("Exit Calibration Mode<<<"));
    Serial.println();
    doCalibrationFinishFlag = 0;
    enterCalibrationFlag = 0;
}
break;
}
}

int getMedianNum(int bArray[], int iFilterLen)
{
    int bTab[iFilterLen];
    for (byte i = 0; i < iFilterLen; i++)
    {
        bTab[i] = bArray[i];
    }
    int i, j, bTemp;
    for (j = 0; j < iFilterLen - 1; j++)
    {
        for (i = 0; i < iFilterLen - j - 1; i++)
        {
            if (bTab[i] > bTab[i + 1])
            {
                bTemp = bTab[i];
                bTab[i] = bTab[i + 1];
                bTab[i + 1] = bTemp;
            }
        }
    }
    if ((iFilterLen & 1) > 0)
        bTemp = bTab[(iFilterLen - 1) / 2];
    else
        bTemp = (bTab[iFilterLen / 2] + bTab[iFilterLen / 2 - 1]) / 2;
    return bTemp;
}

void readDoCharacteristicValues(void)
{
    EEPROM_read(SaturationDoVoltageAddress, SaturationDoVoltage);
    EEPROM_read(SaturationDoTemperatureAddress,
SaturationDoTemperature);
    if (EEPROM.read(SaturationDoVoltageAddress) == 0xFF &&
EEPROM.read(SaturationDoVoltageAddress + 1) == 0xFF &&
EEPROM.read(SaturationDoVoltageAddress + 2) == 0xFF &&
EEPROM.read(SaturationDoVoltageAddress + 3) == 0xFF)
    {
        SaturationDoVoltage = 1127.6; //default voltage:1127.6mv
        EEPROM_write(SaturationDoVoltageAddress, SaturationDoVoltage);
    }
    if (EEPROM.read(SaturationDoTemperatureAddress) == 0xFF &&
EEPROM.read(SaturationDoTemperatureAddress + 1) == 0xFF &&
EEPROM.read(SaturationDoTemperatureAddress + 2) == 0xFF &&
EEPROM.read(SaturationDoTemperatureAddress + 3) == 0xFF)
    {
        SaturationDoTemperature = 25.0; //default temperature is 25^C
        EEPROM_write(SaturationDoTemperatureAddress,
SaturationDoTemperature);
    }
}
}

```

gateway.py

```
#Import
from datetime import datetime
import time
import sys
import spidev
import RPi.GPIO as GPIO
import struct
import rf95
import json
import httplib
import csv

FXOSC=32000000.0
FSTEP = FXOSC / 524288.0

ip_server = "iot.tujuhlangit.id"
port_server = 8080

def value(msg):
    packed = [chr(x) for x in msg]
    #print 'Packed2: %s' % repr(packed)

    joinpacked = ''.join(packed)
    #print 'Join Packed2: %s' % repr(joinpacked)

    unpacked = struct.unpack('<f',joinpacked)[0]
    return unpacked
    #print 'Value: %s' % repr(unpacked)

def send_data(temp, acid, oxyg, turb):
    utc_time = int(time.time())
    userdata = {
        "username" : "delarta",
        "password" : "qweasd"
    }
    # json data
    data = {
        "label" : "Kolam",
        "sensors" : [
            {
                "label" : "TEMP",
                "value" : [
                    [temp, utc_time]
                ]
            },
            {
                "label" : "ACID",
                "value" : [
                    [acid, utc_time]
                ]
            },
            {
                "label" : "TURB",
                "value" : [
                    [turb, utc_time]
                ]
            },
            {
                "label" : "OXYG",
                "value" : [
                    [oxyg, utc_time]
                ]
            }
        ]
    }
    return data
```

```

        ],
        "nodes" : []
    }

    json_data = json.dumps(data)
    # end

    conn = httplib.HTTPConnection(ip_server, port=port_server)

    json_user = json.dumps(userdata)
    header_auth = {
        "Content-Type" : "application/json"
    }

    # melakukan request data user
    conn.request("POST", "/user-auth/", json_user, header_auth)
    user = conn.getresponse().read()
    user = json.loads(user)

    #Supernode
    header_supernode = {
        "Content-Type" : "application/json",
        "Authorization" : "JWT "+str(user['token'])
    }

    supernode = {
        "user": "delarta",
        "label": "PondMon",
        "secretkey": "delarta040996"
    }

    json_supernode = json.dumps(supernode)
    conn.request("POST", "/node-auth/", json_supernode,
header_supernode)
    data = conn.getresponse().read()
    sn_token = json.loads(data) ['token']

    header = {
        "Content-Type" : "application/json",
        "Authorization" : "JWT "+str(sn_token)
    }
    conn.request("POST", "/sensordatas/", json_data, header)
    # Baca responsenya
    data = conn.getresponse().read()
    print(data)

if __name__ == "__main__":
    lora = rf95.RF95(0,0, 25,None)
    if not lora.init(): # returns True if found
        print("RF95 not found")
        quit(1)
    else:
        print("RF95 LoRa mode ok")

    #set frequency, power and mode
    lora.set_frequency(915.0)
    lora.set_tx_power(13)

    print("Testing reception ...")
    with open('hasiluji50m.csv', 'w') as csvfile:
        fieldnames = ["Urutan", "TEMP", "ACID", "OXYG", "TURB",
        "Waktu"]
        writer = csv.DictWriter(csvfile, fieldnames = fieldnames)
        writer.writeheader()

```

```
while (True):
    while not lora.available():
        time.sleep(0.1)

    msg = lora.recv()
    print(msg)
    temp = msg[16:20]
    acid = msg[4:8]
    oxyg = msg[12:16]
    turb = msg[8:12]
    urutan = msg[24]
    waktu = time.strftime("%Y-%m-%d %H:%M:%S")

    temp = float(value(temp))
    acid = float(value(acid))
    oxyg = float(value(oxyg))
    turb = float(value(turb))
    print("==== Received ====")
    print("Urutan: ", str(urutan))
    print("TEMP: ", str(temp))
    print("ACID: ", str(acid))
    print("OXYG: ", str(oxyg))
    print("TURB: ", str(turb))
    print("")
    writer.writerow({'Urutan' : urutan, 'TEMP' : temp, 'ACID' :
: acid, 'OXYG' : oxyg, 'TURB' : turb, 'Waktu' : waktu})
    send_data(temp, acid, oxyg, turb)
    lora.set_mode_idle()
    time.sleep(10)
```