BAB IV

IMPLEMENTASI SISTEM

4.1 Implementasi *Database*

Adapun *listing* program yang digunakan untuk pembuatan *database* sekaligus pembuatan setiap *collection* adalah sebagai berikut :

```
C:\Users\yohan>mongo
"mongodb+srv://cluster0.mllltc5.mongodb.net/" --username
iotproject572
MongoDB shell version v4.4.25
Enter password:
connecting to: mongodb://ac-zedklvh-shard-00-
01.mllltc5.mongodb.net:27017, ac-zedklvh-shard-00-
00.mllltc5.mongodb.net:27017, ac-zedklvh-shard-00-
02.mllltc5.mongodb.net:27017/?compressors=disabled&gssapiServi
ceName=mongodb&ssl=true{"t":{"$date":"2023-10-
22T00:39:54.989Z"}, "s":"I", "c":"NETWORK","id":5490002,
"ctx": "thread1", "msg": "Started a new thread for the timer
service"}Implicit session: session { "id": UUID("92f530fb-
32e5-4d5a-93f2-df4c948ec5f5") }MongoDB server version: 6.0.11
WARNING: shell and server versions do not match
MongoDB Enterprise atlas-rmqlo4-shard-0:PRIMARY> use
Monitoring
switched to db Monitoring
MongoDB Enterprise atlas-rmq1o4-shard-0:PRIMARY>
db.createCollection('PerubahanSuhu')
      "ok": 1,
      "$clusterTime": {
            "clusterTime": Timestamp(1697935271, 21),
            "signature": {
                  "hash": BinData(0,
            "Il0kc64Pddp005Ns1xGpNrk8Vw8="),
                  "keyId": NumberLong("7244847269793497090")
            }
      "operationTime": Timestamp(1697935271, 17)
MongoDB Enterprise atlas-rmqlo4-shard-0:PRIMARY> show dbs
Monitoring  0.000GB
Admin       0.000GB
Local       12.563GB
MongoDB Enterprise atlas-rmqlo4-shard-0:PRIMARY> show
collections
PerubahanSuhu
```

Proses pembuatan *database* dilakukan melalui *Command Prompt* (*CMD*) dengan memanfaatkan fitur *Mongo Shell*. Dengan menggunakan *username* dan *password* dari *MongoDB Atlas*, pengguna dapat membuat *database* baru beserta

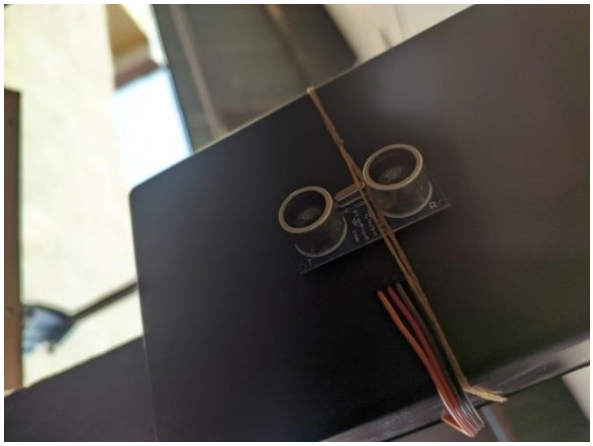
koleksinya. Alternatifnya, pembuatan *database* dan koleksi juga dapat dilakukan dengan mengakses halaman situs *web MongoDB Cloud* melalui peramban *web*.

4.2 Implementasi Perangkat *IOT*

Bagian ini merincikan implementasi perangkat *IOT* yang digunakan sebagai instrumen pengumpulan data sensor. Perangkat *IOT* yang diterapkan adalah sensor-sensor yang dibutuhkan yang terhubung ke mikrokontroler berdaya rendah, yang berfungsi sebagai agen pengirim data. Konfigurasi perangkat *IOT* melibatkan proses pengaturan sensor, pengembangan perangkat lunak pada mikrokontroler, serta pengaturan koneksi ke infrastruktur jaringan yang relevan. Penggunaan protokol komunikasi yang tepat, seperti *MQTT*, digunakan untuk memfasilitasi transmisi data yang andal dari perangkat *IOT* ke *Server*.

1. *Node* Pengaturan Pakan

   Terdapat sensor ultrasonik yang di tempatkan pada bagian atas tempat persediaan pakan agar dapat mengukur jarak ketinggian pakan yang mengukur persediaan pakan. Penermpatan sensor ultrasonik harus diatas tempat penyimpanan pakan agar dapat mengukur persediaan pakan dapat dilihat pada (Gambar 4.2) dan (Gambar 4.1).



Gambar 4.1 Sensor Ultrasonik

Adapun *list* program untuk melakukan pengukuran persediaan pakan menggunakan sensor ultrasonik adalah sebagai berikut :

```
int bacaJarak() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long durasi = pulseIn(echoPin, HIGH);
  int jarak = durasi / 58;
  return jarak;}
void loop() {
  if (!client.connected()) {
    reconnect();}
  client.loop();
  int jarak = bacaJarak();
  Serial.print("Jarak: ");
  Serial.print(jarak);
  Serial.println(" cm");
  String pesanJarak = String(jarak);
  client.publish(mqtt_topic, pesanJarak.c_str());
  delay(5000);
}
```

Dalam *node* pakan juga terdapat dua buah servo yang akan menjadi pintu penghalang keluarnya pakan. Servo akan berputar 90 derajat lalu katup pakan terbuka dan menumpakan pakan ke tempat pakan ayam setelah itu akan tertutup kembali. Penempatan servo yang semestinya dapat dilihat pada (Gambar 4.2)

Gambar 4.2 Pengaturan Pakan (Servo)

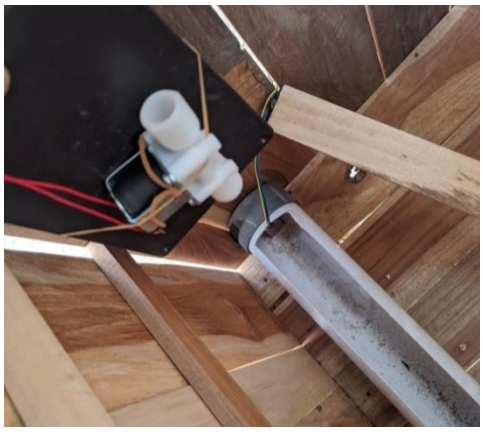
Program dibawah ini digunakan untuk membuka dan menutup mekanisme pemberian pakan melalui penggerak servo motor. Program menerima perintah untuk membuka atau menutup mekanisme pakan melalui pesan *MQTT* yang dikirimkan ke topik "pakan". Ketika pesan "*ON*" diterima, servo motor secara perlahan dibuka untuk memberikan makanan, dan ketika pesan "*OFF*" diterima, servo motor secara perlahan ditutup untuk menghentikan pemberian makanan. Ketika Server membaca waktu adalah waktu makan, maka Server akan mem-*publish* "beri_pakan" dengan topik pakan dan *node* pakan men-*subscribe* topik pakan. Adapun *listing* program yang mengatur pemberian pakan sesuai pesan yang diterima adalah sebagai berikut :

```
void callback(char* topic, byte* payload, unsigned
int length) {
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];}
  if (strcmp(topic, mqtt_topic) == 0) {
    if (message == "ON") {
      bukaServo();
    } else if (message == "OFF") {
      tutupServo();
    } else if (message == "kasih_pakan") {
      kasihPakan();}}}
void bukaServo() {
  servoControlInProgress = true;
  for (int pos = 90; pos >= 0; pos--) {
    myServo1.write(pos);
    myServo2.write(pos);
    delay(10);}
  Serial.println("Pakan ON");
  Serial.println("Kedua servo dibuka");
  servoControlInProgress = false;}
void tutupServo() {
  servoControlInProgress = true;
  for (int pos = 0; pos <= 90; pos++) {
    myServo1.write(pos);
    myServo2.write(pos);
    delay(10);}
  Serial.println("Pakan OFF");
  Serial.println("Kedua servo ditutup");
  servoControlInProgress = false;}
void kasihPakan() {
  bukaServo();
  delay(1000);
  tutupServo();}
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
}
```

2. *Node* air minum

Node sistem air minum tidak membutuhkan koneksi ke *MQTT* karena persediaan air tidak perlu di-*monitoring*. Persediaan air harus terus ada ketika level air kurang dari yang ditentaukan. Ketika *water level sensor* membaca ketinggian air rendah , maka pompa otomatis menyala tanpa harus

menggunakan perintah dari *Server*. Penempatan *node* kontrol air dapat dilihat pada (Gambar 4.3)



Gambar 4.3. Kontrol air minum

Penggunaan code dari sistem ini sangat sederhana. berikut merupakan *list code*-nya :

```
void loop() {
  int sensorValue = analogRead(sensorPin);
  float waterLevel = map(sensorValue, 0, 4095, 0,
100);
  Serial.print("Ketinggian Air: ");
  Serial.print(waterLevel);
  Serial.println(" cm");
  if (waterLevel < 50) {
    digitalWrite(relayPin, HIGH);
  } else if (waterLevel > 50) {relay
    digitalWrite(relayPin, LOW);}
  delay(1000);}
```

3. *Node* Suhu dan Kelembaban

Pada kandang tersebut, kipas ditempatkan untuk memberikan sirkulasi udara yang baik maka harus ditengah kandang atau bagian belakang, dapat dilihat pada (Gambar 4.4).

Gambar 4.4 Aktuator Kipas

Bola lampu dipasang dalam kandang untuk memberikan pemanasan tambahan, yang dapat dilihat pada (Gambar 4.5). *NodeMCU* dan sensor *DHT22*, sebagai perangkat *IOT*, ditempatkan di dalam kandang untuk membaca suhu dalam kandang melalui sensor dan mengirimkan data ke Server *MQTT*. Server *MQTT* memberikan instruksi kepada *NodeMCU* berdasarkan batas suhu yang ditentukan, yang kemudian mengontrol kipas atau lampu sesuai kebutuhan.

Gambar 4.5. Sistem Pengaturan Suhu

Data suhu dan kelembaban yang di *publish* ke topik "suhu" menjadi acuan untuk mengontrol aktuator berupa kipas dan lampu, topik yang di-

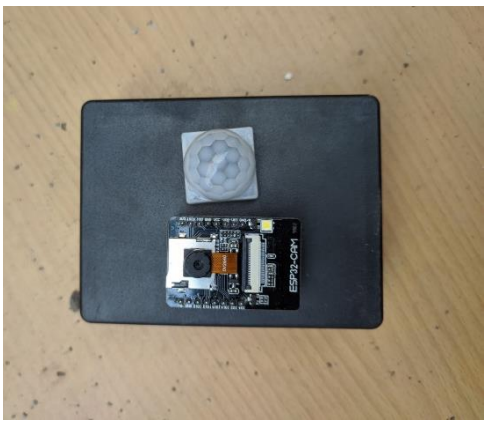
subscribe adalah kondisi suhu yang memiliki 3 kondisi. Jika pesan yang diterima dingin maka kipas mati dan lampu hidup, jika pesan panas maka kipas hidup dan lampu mati. Jika pesannya adalah normal otomatis kedua aktuator akan di matikan. Adapun *listing* program yang diimplementasikan adalah sebagai berikut :

```
void loop() {
  float suhu = dht.readTemperature();
  float kelembaban = dht.readHumidity();
  if (isnan(suhu) || isnan(kelembaban)) {
    Serial.println("Gagal membaca sensor DHT11!");
    return;}
  char suhuString[10];
  dtostrf(suhu, 5, 2, suhuString);
  char kelembabanString[10];
  dtostrf(kelembaban, 5, 2, kelembabanString);
  StaticJsonDocument<200> doc;
  doc["suhu"] = suhuString;
  doc["kelembaban"] = kelembabanString;
char jsonBuffer[256];
  serializeJson(doc, jsonBuffer);
  if (client.publish(mqttTopic, jsonBuffer)) {
    Serial.println("Data terkirim ke broker MQTT");
  } else {
    Serial.println("Gagal mengirim data ke broker MQTT");
  } delay(5000); }
```

```
void callback(char* topic, byte* payload, unsigned int
length) {
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];}
  if (strcmp(topic, mqttTopicSuhu) == 0) {
    if (message == "dingin") {
      digitalWrite(lampuPin, LOW);
      digitalWrite(kipasPin, HIGH);
      Serial.println("Dingin - Lampu hidup, Kipas mati");
    } else if (message == "panas") {
      digitalWrite(kipasPin, LOW);
      digitalWrite(lampuPin, HIGH);
      Serial.println("Panas - Lampu mati, Kipas hidup");
    } else if (message == "normal") {
      digitalWrite(lampuPin, HIGH);
      digitalWrite(kipasPin, HIGH);
      Serial.println("Normal - Lampu mati, Kipas mati");
```

4. *Node* Sistem Keamanan

Node ini bekerja dengan menggunakan *PIR* sensor dan ESP32-Cam akan mengirim perintah ke *broker MQTT* dan melakukan pengambilan gambar dan mengirimkan notifikasi ke *bot* telegram jika ada objek yang mencurigakan. Komponen node keamaan dapat dilihat pada (Gambar 4.6) yang ditempatkan pada area yang menjangkau keseluruhan kandang.



Gambar 4.6. Sistem Keamanan

Berikut ini merupakan *listing* program node keamanan :

```
void loop() {
  if (sendPhoto) {
    Serial.println("Persiapan foto");
    sendPhotoTelegram();
    sendPhoto = false;
  }
  if (adaGerakan) {
    bot.sendMessage(chatId, "ADA GERAKAN !!!", "");
    Serial.println("Ada gerakan");
    sendPhoto = true;
    publishToMQTT();
    adaGerakan = false;
  }
  if (millis() > lastTimeBotRan + botRequestDelay) {
    int numNewMessages =
bot.getUpdates(bot.last_message_received + 1);
    while (numNewMessages) {
      Serial.println("Got response");
      handleNewMessages(numNewMessages);
      numNewMessages =
bot.getUpdates(bot.last_message_received + 1);
```

4.3 Implementasi Sistem

Implementasi sistem *IOT* dan *WSN* untuk me-*monitoring* dan mengontrol peternakan ayam ini menggunakan bahasa pemrograman *javascript* dan menggunakan protokol *MQTT* serta menerapkan teknologi *MEVN* dalam pengembangan aplikasi berbasis *website*.

4.3.1 Implementasi *Server*

Pada tahap ini, dilakukan penerapan infrastruktur *Server* NodeJS sebagai komponen inti dalam penelitian ini. Infrastruktur *Server* ini dimaksudkan untuk menyimpan, mengelola, dan mengakses data yang dihasilkan oleh perangkat *IOT*. *Server* digunakan sebagai *broker MQTT*. Nodejs mendukung komunikasi *real-time* melalui protokol *Socket.IO* digunakan untuk membangun aplikasi *real-time* yang menghubungkan perangkat *IOT* dengan antarmuka pengguna dan *database*. *Server* men-*subscribe* tiga topik *MQTT*. Berikut adalah beberapa bagian dari implementasi *Server*:

1) Inisialisasi dan Konfigurasi *Server*

Adapun tahap awal dalam pengaturan *server* adalah inisialisasi dan konfigurasi semua komponen atau *library* yang akan digunakan. Berikut merupakan *listing* program yang mencakup konfigurasi awal dari *server*.

```
const express = require("express");
const mqtt = require("mqtt");
const { MongoClient } = require("mongodb");
const moment = require("moment-timezone");
const fs = require("fs");
const http = require("http");
const socketIo = require("Socket.IO");
const clientTwilio = require("twilio")(
  "AC8083b62820c67e7ec54ffd4aadd99c03",
  "4f47f72303f397a5133daa21ba9f6ec9"
);
const cors = require("cors");
const app = express();
app.use(cors());
const server = http.createServer(app);
const io = socketIo(server, {
  cors: {
    origin: "http://localhost:5173",
    methods: ["GET", "POST"],
  },
});
const port = 3000;
```

2) Konfigurasi *MQTT* dan *Database*

Tahap ini merupakan konfigurasi *MQTT*, topik-topik apa saja yang di-*subscribe* dari *MQTT Broker*, URI *MongoDB*, dan nama – nama koleksi di *database*, sekaligus menginisialisasi koneksi ke *broker MQTT* dan koneksi ke *database MongoDB* menggunakan Mongo*Client* yang dapat dilihat pada *listing* program berikut :

```
const mqttBrokerUrl = "mqtt://192.168.69.94";
const mqttTopic1 = "suhu";
const mqttTopic2 = "jarak_pakan";
const mqttTopic3 = "keamanan";

const uri = "mongodb://yohan231201:Yokef231201@ac-zedklvh-
shard-00-00.mllltc5.mongodb.net:27017,ac-zedklvh-shard-00-
01.mllltc5.mongodb.net:27017,ac-zedklvh-shard-00-
02.mllltc5.mongodb.net:27017/?ssl=true&replicaSet=atlas-
rmq1o4-shard-
0&authSource=admin&retryWrites=true&w=majority";
const dbName = "Peternakan";
const collectionName = "PerubahanSuhu";
const collectionName2 = "waktu_pakan";
const collectionName3 = "setting_suhu";
const collectionName4 = "history_pakan";
```

3) Penanganan Koneksi Socket.IO

Koneksi *Socket.IO* digunakan untuk menangani koneksi dan pesan dari antar muka *website* untuk interaksi *real-time*.

```
  io.on("connection", (socket) => {
    socket.on("lampu_ON", () => {
      console.log("Menghidupkan lampu");
      mqttClient.publish("lampu", "ON");
    });
    socket.on("lampu_OFF", () => {
      console.log("Mematikan lampu");
      mqttClient.publish("lampu", "OFF");
    });
socket.on("kipas_ON", () => {
console.log("Menghidupkan kipas");
   mqttClient.publish("kipas", "ON");
});
socket.on("kipas_OFF", () => {
   console.log("Mematikan kipas");
   mqttClient.publish("kipas", "OFF");
});
socket.on("pakan_ON", () => {
  console.log("Membuka katup pakan");
  mqttClient.publish("pakan", "ON", (err) => {
 if (err) {
 console.error("Gagal melakukan publish ke MQTT:", err);
 }else{
   insertPakanHistory("ON");
   console.log('Berhasil melakukan publish pesan "ON" ke
MQTT topik "pakan"');}
      });
    });
socket.on("pakan_OFF", () => {
  console.log("Menutup katup pakan");
   mqttClient.publish("pakan", "OFF", (err) => {
     if (err) {
      console.error("Gagal melakukan publish ke MQTT:",
err);
   } else {
      console.log('Berhasil melakukan publish pesan "OFF"
ke MQTT topic "pakan"'
```

Diatas merupakan penanganan perintah dari *client Socket.IO* yang mencakup tugas yang dilakukan pada halaman *controling*. Adapun tugas lainnya yang dapat dilakukan oleh *Server* saat menerima perintah dari *client Socket.IO*, seperti melakukan proses *CRUD (Create, Read, Update,*

Delete) ke *database.* Contoh umum yang dilakukan oleh *Server* adalah melakukan penambahan waktu pakan yang dapat dilihat dari *listing* program dibawah ini :

```
collection2.find().toArray((error, data) => {
   if (error) {
   console.error(`Gagal membaca data dari
database:${error}`);
   } else {
   io.emit("data-jadwal-pakan", data);
   socket.on("tambah-waktu-pakan", (jam) => {
   console.log(`Jam ditambahkan: ${jam}`);
   collection2.insertOne({ jam }, (error, result) => {
     if (error) {
     console.error(`Gagal menyimpan ke database:${error}`);
     } else {
     io.emit("tambah-jadwal-pakan", jam);
     collection2.find().toArray((error, data) => {
     if (error) {
     console.error(`Gagal membaca data dari database:
${error}`);
     } else {
     io.emit("data-jadwal-pakan", data);
     `
```

4) *Subscribe* ke Topik *MQTT* dan Penanganan Pesan

Tahapan ini merupakan tugas yang dilakukan untuk menangani koneksi dan pesan dari *MQTT*, termasuk penanganan pesan untuk topik suhu, jarak_pakan dan keamanan. Berikut merupakan *listing* program dari *Server* yang menerima data suhu.

```
if (topic === mqttTopic1) {
    const db = client.db(dbName);
    const collection3 = db.collection(collectionName3);
    try {
      const data = JSON.parse(message.toString());
      insertData(data, collectionName);
      const waktu = moment()
        .tz("Asia/Makassar")
        .format("YYYY-MM-DD (HH:mm:ss)");
      console.log(
        `Data terbaru: Suhu ${data.suhu}°C, Kelembaban
${data.kelembaban}%, Waktu ${waktu}`
      `
```

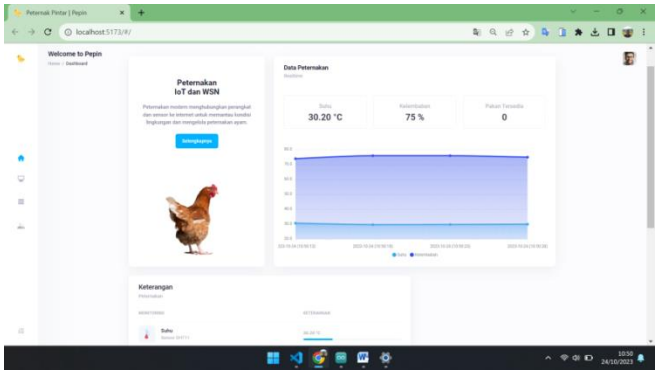
Setelah menerima data *real-time* dari node suhu, *Server* merespon kondisi saat ini sesuai nilai suhu yang dibaca. Dibawah ini merupakan *listing* program yang melakukan *publish* pesan kondisi_suhu dengan mengkondisikan dengan pengaturan suhu yang ada pada *database* setting_suhu dan menyimpan data perubahan suhu yang terjadi.

```
io.emit("suhuRealtime", { ...data, waktu });
 collection3.findOne({}, (error, settingSuhu) => {
 if (error) {
 console.error(`Gagal membaca data dari collection3:
${error}`);
 } else {
  const maximalSuhu = settingSuhu.maximal;
  const minimalSuhu = settingSuhu.minimal;
 if (data.suhu < minimalSuhu) {
     mqttClient.publish("kondisi_suhu", "dingin");
     console.log("Suhu Dingin");
 }else if(data.suhu > maximalSuhu) {
     mqttClient.publish("kondisi_suhu", "panas");
     console.log("Suhu Panas");
 } else if (data.suhu > minimalSuhu && data.suhu <
maximalSuhu){
     mqttClient.publish("kondisi_suhu", "normal");
     console.log("Suhu Normal");
 } else {
     console.log("gagal membaca data setting suhu");  }}});
async function insertData(dataToInsert, collectionName) {
   try {
   await client.connect();
   const db = client.db(dbName);
   const collection = db.collection(collectionName);
   if (dataToInsert.suhu !== lastSuhu) {
     dataToInsert.waktu =
moment().tz("Asia/Makassar").format("YYYY-MM-DD
(HH:mm:ss)");
   const result = await collection.insertOne(dataToInsert);
   console.log("Data berhasil disimpan:");
   console.log(result);
   lastSuhu = dataToInsert.suhu;
   } else {
     console.log("Data suhu sama dengan data sebelumnya,
tidak disimpan."); }
```

4.3.2 Implementasi Antarmuka *Web*

1. Halaman *Dashboard*

Halaman *Dashboard* adalah titik awal untuk peternak, menyajikan informasi singkat tentang peternakan modern dengan *WSN* dan *IOT*. Ketika ada pembaruan sensor, halaman menampilkan data suhu *real-time* melalui *Soket.IO,* disajikan dalam grafik dengan 5 data terakhir. Tautan cepat memungkinkan pengguna beralih ke halaman *Monitoring* dan *controling* untuk tindakan lebih lanjut. Lihat pada (Gambar 4.7).



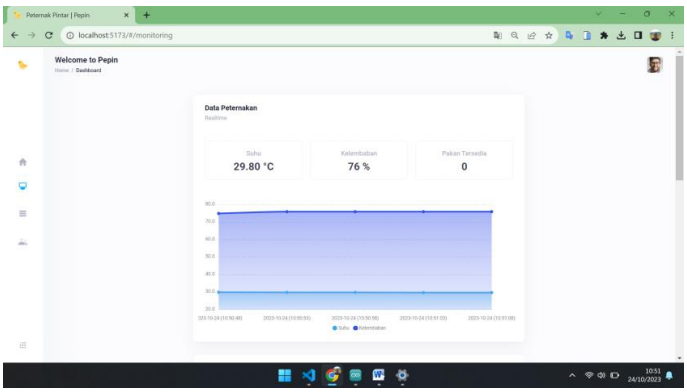
Gambar 4. 7 Halaman *Dashboard*

Adapun *listing* program dari halaman *dashboard* adalah sebagai berikut :

```
return {
   sensorData: [], currentSuhu: 0, currentKelembaban:
0, currentWaktu:
   computed: {
    recentSensorData() {
      const startIndex = Math.max(0,
this.sensorData.length - 5);
      return this.sensorData.slice(startIndex);
    created() {
    const socket = io('http://localhost:3000');
    socket.on('suhuRealtime', (data) => {
      this.sensorData.push(data);
      this.currentSuhu = data.suhu;
      this.currentKelembaban = data.kelembaban;
      this.currentWaktu = data.waktu;
      this.updateChart();});
      this.currentPakan = data.pakan;
    ``
```

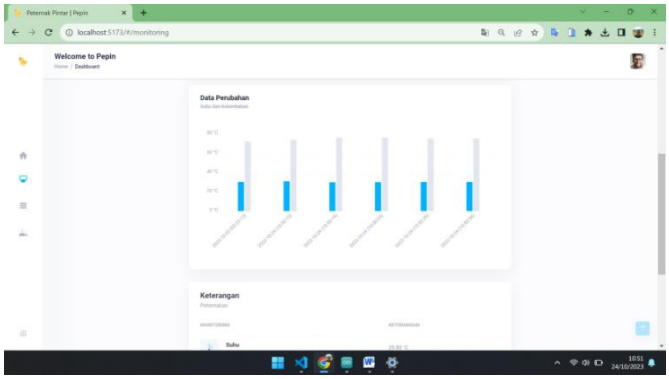
2. Halaman *Monitoring*

Halaman *Monitoring* digunakan menampilkan data sensor secara *real-time* dengan grafik dan tabel sebagai tampilan visual. Data sensor diperbarui secara dinamis saat data baru diterima melalui Socket.IO. Pada halaman ini juga menampilkan informasi perubahan suhu dan kelembaban menggunakan data suhu dalam *database* yang dibagikan melalui *Socket.IO* yang dapat dilihat pada (Gambar 4.8).



Gambar 4.8 Halaman *Monitoring*

Informasi perubahan suhu dan kelembaban yang terjadi dalam kandang ayam disajikan dalam bentuk bar chart (Gambar 4.9).



Gambar 4.9 *Chart* Perubahan suhu

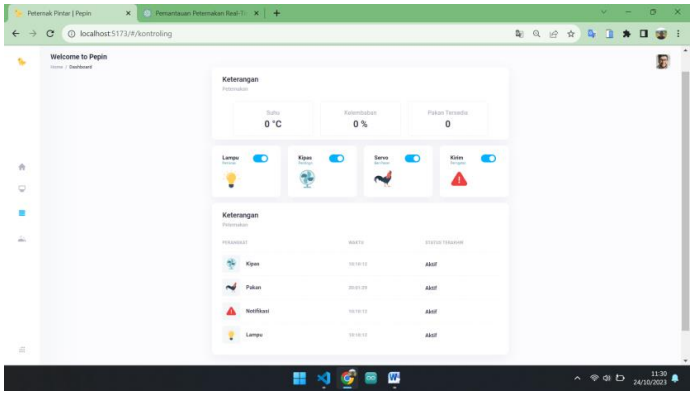
Berikut merupakan *listing* program dari antar muka *website* pada halaman *Monitoring* yang menampilkan data suhu *real-time* dan perubahan suhu yang terjadi.

```
computed: {
recentSensorData() {
   const startIndex = Math.max(0, this.sensorData.length -
5);
   return this.sensorData.slice(startIndex);},
 latestData() {
 if (this.dataList.length > 0) {
        return this.dataList[this.dataList.length - 1]; }
      return { suhu: 0, kelembaban: 0, waktu: '' };},
  created() {
    const socket = io('http://localhost:3000');
    socket.on('suhuRealtime', (data) => {
      this.sensorData.push(data);
      this.currentSuhu = data.suhu;
      this.currentKelembaban = data.kelembaban;
      this.currentWaktu = data.waktu;
      this.updateChart();});
    socket.on('dataHistoris', (data) => {
      this.dataList = data;
      this.initChartsWidget2();});
    socket.on('pakanRealtime', (data) => {
      this.currentPakan = data.pakan; });
```

3. Halaman *Controling*

Halaman *controling* ini memberikan akses kontrol terkait dengan perangkat-perangkat di dalam peternakan atau lingkungan serupa, dan memungkinkan pengguna untuk memantau dan mengendalikan status

perangkat dengan mudah (lampu , kipas, pemberian pakan dan darurat) dapat dilihat pada (Gambar 4.10).



Gambar 4.10 Halaman *Controling*

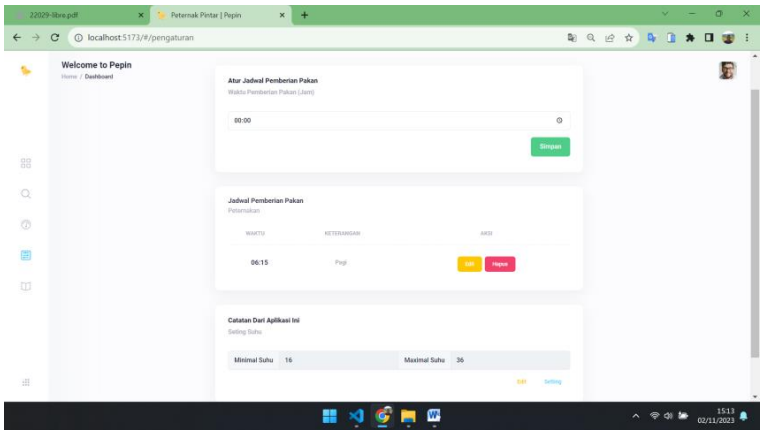
Listing dibawah ini merupakan logika yang mengubah status perangkat aktuator seperti menghidupkan atau mematikan lampu dan kipas, lalu membuka dan menutup katup pakan.

```
created() {
    this.$watch('lampuStatus', (newVal, oldVal) => {
      if (newVal !== oldVal) {
        // Emit 'lampu_ON' or 'lampu_OFF' based on the new
value of lampuStatus
        const eventName = newVal ? 'lampu_ON' : 'lampu_OFF';
        this.socket.emit(eventName); }});
    this.$watch('kipasStatus', (newVal, oldVal) => {
      if (newVal !== oldVal) {
        const eventName = newVal ? 'kipas_ON' : 'kipas_OFF';
        this.socket.emit(eventName); }});
    this.$watch('pakanStatus', (newVal, oldVal) => {
      if (newVal !== oldVal) {
        const eventName = newVal ? 'pakan_ON' : 'pakan_OFF';
        this.socket.emit(eventName); }});
    this.$watch('daruratPesan', (newVal, oldVal) => {
      if (newVal !== oldVal) {
        const eventName = newVal ? 'darurat' : 'aman';
        this.socket.emit(eventName); }});
```

4. Halaman Pengaturan

Melalui halaman ini, peternak dapat mengatur angka minimum dan maximum suhu sebagai acuan , serta jadwal pemberian pakan yang

memastikan nutrisi yang tepat pada waktu yang tepat. Dalam hal ini, halaman pengaturan memberikan fleksibilitas kepada peternak untuk menyesuaikan operasi sesuai dengan kebutuhan khusus, baik secara manual maupun dengan mengintegrasikan sistem otomatisasi yang dapat memantau dan mengendalikan perangkat kandang secara otomatis. Ini membantu meningkatkan efisiensi produksi, memastikan kualitas hidup hewan ternak, dan mengoptimalkan hasil peternakan secara keseluruhan hasil. Dapat dilihat pada (Gambar 4.11)



Gambar 4.11 Halaman Pengaturan

Data yang berhasil di perbarui selanjutnya akan dikirim ke *Server* melalui *Socket.IO* untuk melanjutkan proses *CRUD* dengan *database*, data tersebut berupa data jadwal pemberian pakan sekaligus pengaturan minimal dan maksimal suhu, yang dapat dilihat dari *listing* program dibawah ini.

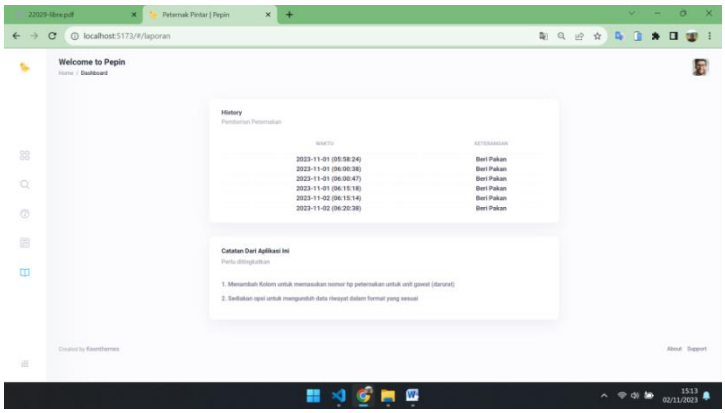
```
export default {
  data() {
   return {
      editedTime: "00:00",
      jamList: [],
      socket: null,
      editIndex: -1,
      minimalSuhu: null,
      maximalSuhu: null,
      isEditing: { minimalSuhu: false, maximalSuhu: false,
},
 created() {
    this.socket = io('http://localhost:3000');
    this.socket.on('data-jadwal-pakan', (data) => {
      this.jamList = data.map(jam => ({ ...jam, keterangan:
this.getKeterangan(jam.jam) }));
    this.socket.on('tambah-jadwal-pakan', (jam) => {
      this.jamList.push({ jam, keterangan:
this.getKeterangan(jam) });
    this.socket.on('edit-jadwal-pakan', ({ oldJam, jam })
=> {
      const index = this.jamList.findIndex(item => item.jam
=== oldJam);
      if (index !== -1) {
        this.jamList[index].jam = jam;
        this.jamList[index].keterangan =
```

```
  methods: {
    simpanWaktuPakan() {
      if (this.editIndex === -1) {
        const jamDitambahkan = this.editedTime;
        this.socket.emit('tambah-waktu-pakan',
jamDitambahkan);
      } else {
        const oldJam = this.jamList[this.editIndex].jam;
        const jamDiubah = this.editedTime;
        this.socket.emit('edit-waktu-pakan', { oldJam, jam:
jamDiubah });
      }
      this.editedTime = "00:00";
      this.editIndex = -1;
    },
    editWaktuPakan(index) {
      this.editIndex = index;
      this.editedTime = this.jamList[index].jam;
    hapusWaktuPakan(index) {
      const jamDihapus = this.jamList[index].jam;
      this.socket.emit('hapus-waktu-pakan', jamDihapus); },
    getKeterangan(jam) {
      const time = jam.split(':');
      const hour = parseInt(time[0]);
      if (hour >= 1 && hour < 11) {
        return 'Pagi';
      } else if (hour >= 11 && hour < 14) {
        return 'Siang';
      } else if (hour >= 14 && hour < 18) {
        return 'Sore';
      } else {
        return 'Malam';
      }},
    editSetting() {
      this.isEditing.minimalSuhu = true;
      this.isEditing.maximalSuhu = true; },
    simpanSetting() {
        this.socket.emit('updateSettingSuhu', {
        minimal: this.minimalSuhu,
        maximal: this.maximalSuhu,
      });
      // Matikan mode pengeditan minimal dan maksimal suhu
      this.isEditing.minimalSuhu = false;
      this.isEditing.maximalSuhu = false;}, },};
```

5. Halaman Laporan dan Riwayat

Pada halaman ini di tampilkan riwayat pemberian pakan. Jika pemberian dilakukan berdasarkan waktu yang telah ditentukan maka akan disimpan dengan keterangan makan pokok, jika pemberian pakan di

kontrol manual dengan tombol pada halaman *controling* maka keterangan akan disimpan makan tambahan. Status pemberian pakan yang dilakukan dapat dilihat pada (Gambar 4.12)



Gambar 4.12 Halaman Laporan

Pada halaman ini memiliki *listing* program yang sederhana karena hanya menampilkan riwayat pemberian pakan yang dilakukan.

```
export default {
  data() {
    return {
      historyPakan: [],
    };
  created() {
    this.socket = io('http://localhost:3000');
    this.socket.on("historyPakan", (pemberianPakan) => {
    this.historyPakan = pemberianPakan;
    });
  },
};
```