**PROJECT EMBEDED SYSTEM**

**Carbon Monoxide Detector dengan Memanfaatkan NodeJS dan Web Interface**

****

**Disusun Oleh:**

**Ivan Christian Trisula 217116608**

**PROGRAM SARJANA**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS SAINS DAN TEKNOLOGI TERPADU SURABAYA**

**INSTITUT SAINS DAN TEKNOLOGI TERPADU**

**SURABAYA**

**2020**

1. **Deskripsi**

Proyek ini memanfaatkan sensor karbon monoksida MQ-7 yang dikoneksikan dengan NodeMCU dan secara terus - menerus mensinkronisasi data dengan *server* NodeJS. Jumlah ppm karbon monoksida sekitar perangkat dapat diakses oleh pengguna dengan mudah melalui sebuah *web interface*. Perangkat juga akan membunyikan alarm apabila level karbon monoksida mencapai tingkat berbahaya.

1. **Alat – Alat**

* 1 buah modul NodeMCU ESP8266
* 1 buah Breadboard
* 1 buah Buzzer Active
* 1 buah modul MQ-7
* 8 buah jumper Male to Male
* 1 buah kabel USB
* 1 buah Power Bank

1. **Perangkat Lunak & Teknologi**

* Arduino IDE

Arduino IDE adalah Integrated Development (IDE) yang berguna untuk *upload* kode program ke Arduino Development Board. Arduino IDE ditulis dalam bahasa pemograman C dan C++.

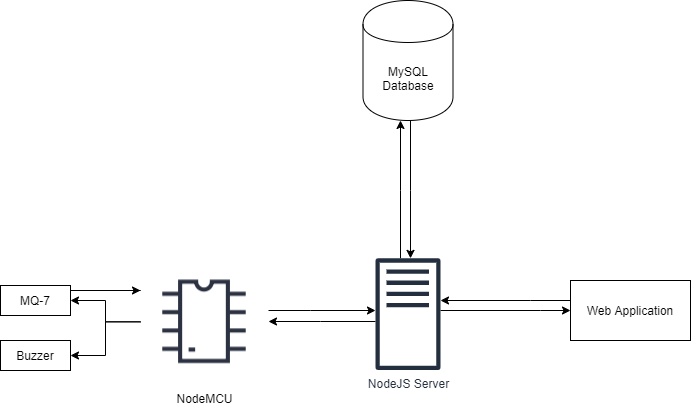
* NodeJS

NodeJS merupakan *environtment* untuk menjalankan kode JavaScript di luar *web browser*. NodeJS juga dapat menjalankan kode JavaScript untuk *server-side scripting*.

* ExpressJS

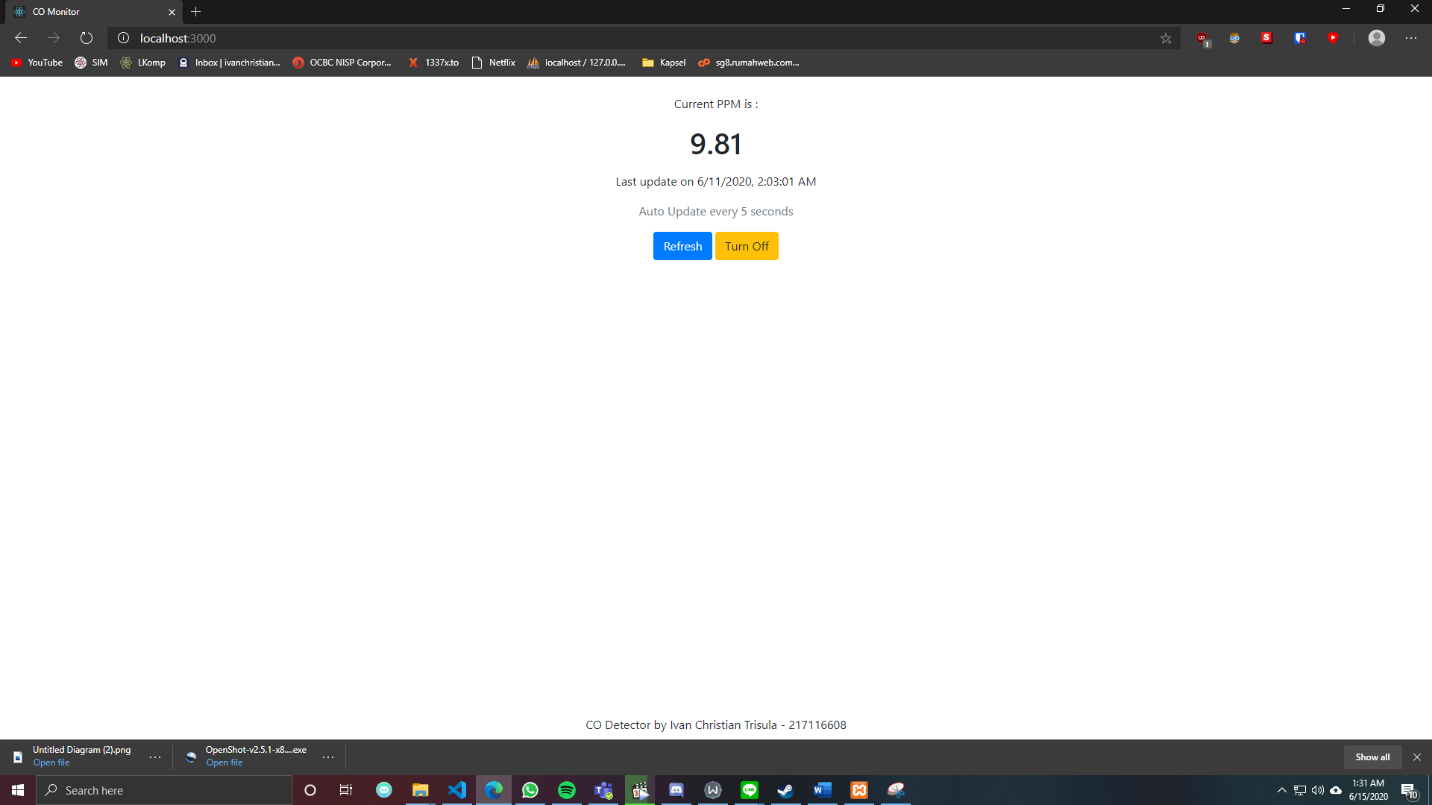
ExpressJS merupakan *framework* NodeJS untuk membuat aplikasi *back-end server* dan API.

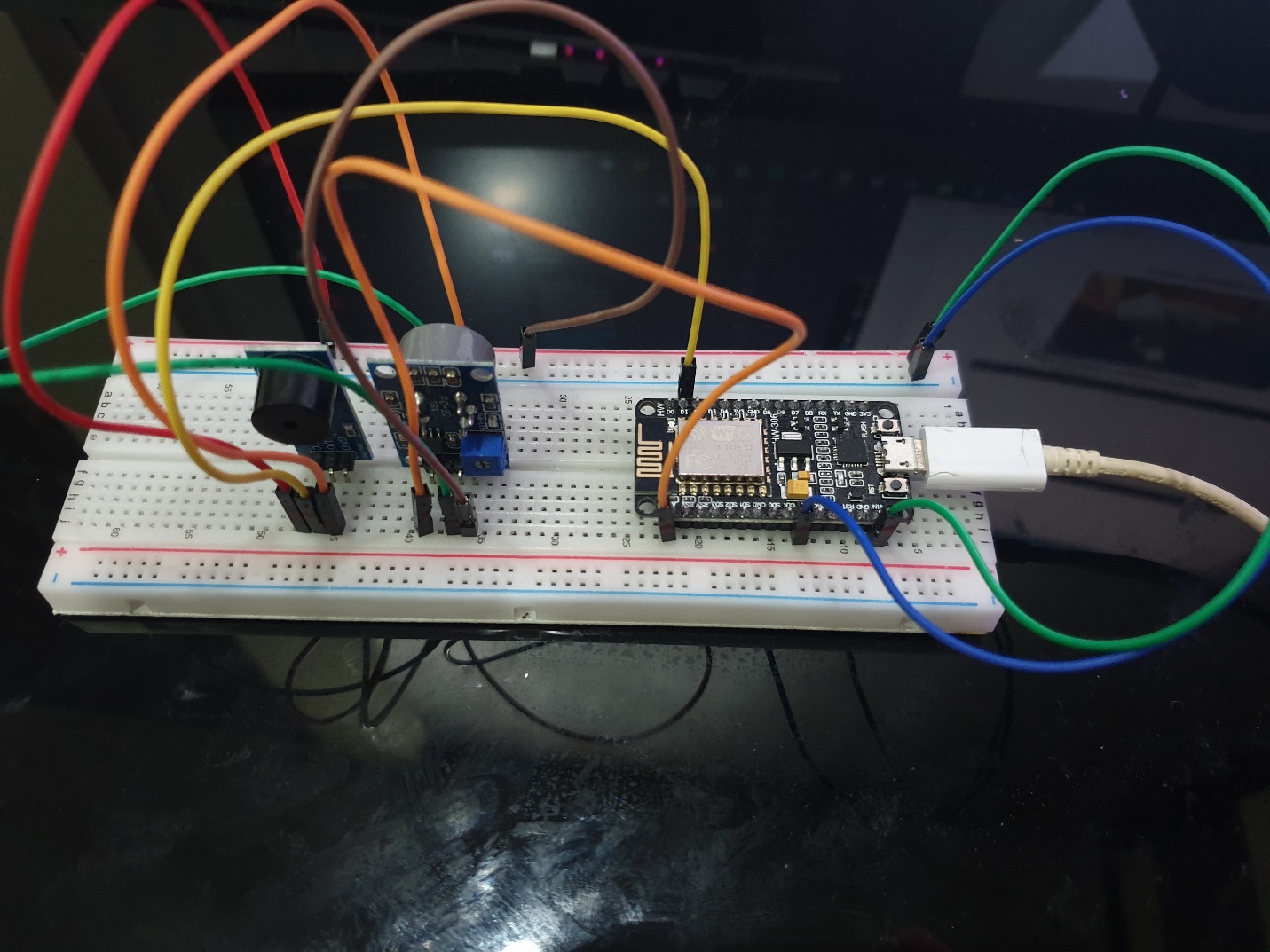
1. **Block Diagram**



Gambar 1 Block Diagram

Pada **Gambar 1 Block Diagram** di atas, dapat dilihat NodeMCU akan mengirimkan nilai ppm karbon monoksida ke ke *server* NodeJS. Kemudian NodeJS akan menyimpan data tersebut ke dalam database. Apabila nilai ppm karbon monoksida sudah melebihi batas aman, NodeMCU akan mengirimkan sinyal ke buzzer untuk menyalakan alarm sehingga bisa memberikan peringatan ke orang-orang sekitar. Pengguna juga bisa mengakses nilai ppm secara *real-time* melalui sebuah halaman web. Pada halaman tersebut, pengguna juga bisa menyalakan dan juga mematikan sensor NodeMCU.

1. **Hasil Proyek**

Gambar 2 Tampilan Web-App

Gambar 3 NodeMCU-Client

1. **Segmen Program**

Pada bab ini akan dijelaska bagian-bagian program dari sisi detector(NodeMCU) dan *server* (NodeJS)

1. **NodeMCU-Client (Arduino Source Code)**

Berikut di bawah akan dijelaskan mengenai segmen program NodeMCU

Segmen Program Arduino (1)

1. #define pinSensor A0
2. #include <ESP8266WiFi.h>
3. #include <ESP8266HTTPClient.h>
4. int PPMTreshold = 5;
5. bool alert=false;
6. long sec = 0;
7. long currMillis = 0;
8. float alertInterval = 0;
9. int stateUpdate = 0;
10. String ssid= "anugerah store";
11. String password= "12051997";
12. WiFiServer server(80);
13. float ppm = 0;
14. int state;

Pada **Segmen 1** di atas, merupakan bagian inisialisasi variable dan library. Pada baris 1 sampai 3 merupakan bagian *import library* yang digunakan agar *code* NodeMCU bisa berjalan. Pada baris 5 sampai 10 merupakan inisialisaasi variable yang digunakan pada *code*. Pada baris 12 sampai 13 digunakan untuk menyimpan data wifi, yaitu SSID dan Password. Pada baris 14 digunakan untuk mengambil object wifi dari library. Pada bagian 15 sampai 16 merupakan inisialisasi variable.

Segmen Program Arduino (2)

1. void refreshState(){
2. if (WiFi.status() == WL\_CONNECTED) {
3. HTTPClient http;
4. http.begin("http://192.168.1.51:3000/api/getState");
5. http.addHeader("Content-Type", "application/x-www-form-urlencoded");
7. int httpCode1= http.GET();
9. if (httpCode1 > 0) {
10. String res = http.getString();
11. state = res.toInt();
12. } else {
13. Serial.println("No Response");
14. }
16. //Serial.println("http code: " + String(httpCode));
17. http.end();
18. } else {
19. Serial.println("Not connected");
20. }

}

Pada **Segmen Program Arduino (2)** di atas merupakan fungsi bernama refreshState. Fungsi ini berguna untuk mengambil data state sensor dari *server* NodeJS. State yang dimaksud adalah apakah pengguna ingin sensor menyala atau tidak. Pada baris 18 merupakan pengecekan apabila sensor tersambung dengan wifi. Apabila tidak, maka sensor tidak akan mencoba mengambil data dari *server*. Pada baris 19 sampai 35 berfungsi untuk mengambil data dari server. Data dari server kemudian dimasukkan ke dalam variable state pada baris 27.

Segmen Program Arduino (3)

1. void setup()
2. {
3. WiFi.begin(ssid, password);
4. int timeout = 0;
5. while(WiFi.status() != WL\_CONNECTED && timeout!=10) {
6. delay(500);
7. Serial.println("Connecting...");
8. timeout++;
9. }
11. server.begin();
12. Serial.begin(9600);
13. Serial.println("Connected");
14. Serial.begin(9600);
15. pinMode(D0,OUTPUT);
17. refreshState();

55: }

Pada **Segmen Program Arduino (3)** di atas merupakan fungsi yang dijalankan setiap kali NodeMCU dinyalakan. Fungsi diatas hanya akan berjalan sebanyak 1 kali saja. Pada baris 40, NodeMCU akan memulai komunikasi dengan server melalui sambungan Wi-Fi. Pada baris 42 sampai 46, NodeMCU akan terus menunggu Wi-Fi tersambung dan memiliki timeout ketika NodeMCU belum tersambung ke Wi-Fi selama 5 detik. Pada baris 49 sampai 51 merupakan inisialisasi sambungan Serial. Pada baris 54, NodeMCU memanggil fungsi refreshState yang berguna untuk meminta sensor state dari server.

Segmen Program Arduino (4)

1. void loop()
2. {
3. sec=millis()/1000;
4. if(state==1){
5. if(millis()/1000>sec){
7. int sensorvalue = analogRead(pinSensor); // membaca nilai ADC dari sensor
8. float VRL= sensorvalue\*5.00/1024; // mengubah nilai ADC ( 0 - 1023 ) menjadi nilai voltase ( 0 - 5.00 volt )
9. Serial.print("VRL : ");
10. Serial.print(VRL);
11. Serial.println(" volt");
13. float Rs = ( 5.00 \* RL / VRL ) - RL;
14. Serial.print("Rs : ");
15. Serial.print(Rs);
16. Serial.println(" Ohm");
18. ppm = 100 \* pow(Rs / Ro,-1.53); // ppm = 100 \* ((rs/ro)^-1.53);
19. Serial.print("CO : ");
20. Serial.print(ppm);

Serial.println(" ppm");

Pada **Segmen Program Arduino (4)** di atas merupakan bagian dari fungsi loop. Loop merupakan fungsi bawaan arduino. Fungsi ini dieksekusi secara terus menerus selama NodeMCU masih menyala. Pada segmen ini merupakan segmen yang berfungsi untuk menghitung jumlah ppm udara. Pada baris 62, NodeMCU akan mengambil data analog dari sensor MQ-7. Value dari sensor kemudian diubah menjadi nilai volt / VRL menggunakan rumus berikut



Kemudian, sebelum mendapatkan nilai ppm, harus dicari *value* Ro dan Rs. *Value* R0 merupakan sebuah konstanta yang harus dikalibrasi terlebih dahulu dengan lingkungan sekitar. Pada proyek ini, *value* Ro ditetapkan sebesar 830. Selain itu ada juga konstanta lain bernama Rl. RL adalah total tahanan beban pada sensor. *Value* Rl yang digunakan pada proyek ini adalah 1000 ohm.

Pada baris 68, merupakan kode untuk menghitung jumlah Rs. Rs adalah tahanan pada sensor. Untuk mencari Rs, digunakan rumus sebagai berikut :



Dimana Vc adalah arus standart NodeMCU yaitu 5 volt.

Pada baris 73 dilakukan perhitungan ppm. Berikut adalah rumus yang digunakan untuk menghitung nilai ppm :



Segmen Program Arduino (5)

1. if(ppm>50&&ppm<200){
2. alertInterval=1;
3. }else if(ppm>200){
4. alertInterval=2;
5. }else{
6. alertInterval=0;
7. }
8. }

Pada **Segmen Program Arduino (5)** di atas merupakan bagian untuk mengecek apakah nilai ppm sudah diatas batas normal. Semakin tinggi alertInterval maka interval waktu alarm berbunyi akan semakin kecil menandakan tingkat keberbahayaan.

Segmen Program Arduino (6)

1. if(alertInterval==1){
2. pinMode(D1,OUTPUT);
3. digitalWrite(D0, (millis() / 1000) % 2);
4. digitalWrite(D1,(millis()/500)%2);
5. if((millis()/500)%2==0){
6. pinMode(D1,INPUT);
7. }
8. }else if(alertInterval==2){
9. pinMode(D1,OUTPUT);
10. digitalWrite(D0, (millis() / 500) % 2);
11. digitalWrite(D1,(millis()/250)%2);
12. if((millis()/250)%2==0){
13. pinMode(D1,INPUT);
14. }
15. }else{
16. digitalWrite(D0,LOW);
17. pinMode(D1,INPUT);

}

Pada **Segmen Program Arduino (6)** di atas merupakan bagian untuk membunyikan alarm apabila level ppm berbahaya. Seperti yang sudah dijelaskan pada bagian sebelumnya, tingkat keberbahayaan ppm. Apabila level ppm berada di tingkat 1 (baris 84) maka buzzer akan berbunyi setiap setengah detik. Dan apabila level ppm berada di tingkat 2 (baris 91), maka buzzer akan berbunyi setiap ¼ detik. Terakhir, apabila level ppm tidak berbahaya atau pada level 0, maka buzzer akan dimatikan (baris 98)

Segmen Program Arduino (7)

1. if(sec%5==0 && stateUpdate==0){
2. refreshState();
3. stateUpdate = 1;
4. }else if(sec%5!=0 && stateUpdate==1){
5. stateUpdate =0;
6. }

9. }

Pada **Segmen Program Arduino (7)** di atas merupakan bagian untuk meminta ke *server* state dari sensor dengan memanggil fungsi refreshState() yang sudah dijelaskan di segmen sebelumnya. NodeMCU akan melakukan *update* secara berkala setiap 5 detik.

Segmen Program Arduino (8)

1. if(sec%3==0){
2. if (WiFi.status() == WL\_CONNECTED) {
3. HTTPClient http;
4. http.begin("http://192.168.1.51:3000/sendPPM");
5. http.addHeader("Content-Type", "application/x-www-form-urlencoded");
7. int httpCode1= http.POST("ppm="+String(ppm));
9. if (httpCode1 > 0) {
10. String res = http.getString();
11. Serial.println("payload: " + res);
12. } else {
13. Serial.println("No Response");
14. }
16. //Serial.println("http code: " + String(httpCode));
17. http.end();
18. } else {
19. Serial.println("Not connected");
20. }
21. }

Pada **Segmen Program Arduino (8)** di atas merupakan kode untuk mengirimkan data ppm ke server. Pada baris 2, melakukan pengecekan apabila wifi sudah terkoneksikan. Pada baris 4, merupakan *url* yang ditujukan untuk mengirin data. Pada baris 7, data ppm dimasukkan ke dalam payload yang dikirimkan menggunakan tipe http POST

1. **NodeJS**

Berikut di bawah akan dijelaskan mengenai segmen program NodeJS yang terdapat pada *environment* NodeJS. Kode dapat ditemukan di dalam file app.js di folder server

Segmen Program NodeJS (1)

1. const express = require("express");
2. const db = require("./db\_helper");
3. const cors = require("cors");
4. const app = express();
5. app.use(cors());
6. app.set("view engine", "ejs");
7. app.use(express.json());
8. app.use(express.urlencoded({ extended: false }));
9. app.all("/\*", function (req, res, next) {
10. res.header("Access-Control-Allow-Origin", req.headers.origin);
11. res.header("Access-Control-Allow-Credentials", true);
12. res.header("Access-Control-Allow-Methods", "GET,PUT,POST,DELETE,OPTIONS");
13. res.header(
14. "Access-Control-Allow-Headers",
15. "Content-Type,Accept,X-Access-Token,X-Key,Authorization,X-Requested-With,Origin,Access-Control-Allow-Origin,Access-Control-Allow-Credentials"
16. );
17. if (req.method === "OPTIONS") {
18. res.status(200).end();
19. } else {
20. next();
21. }

});

Pada **Segmen Program NodeJS (1)** di atas merupakan bagian awal untuk inisialisasi library dan modul yang akan digunakan untuk menjalankan server.

Segmen Program NodeJS (2)

1. app.post("/sendPPM", async function (req, res) {
2. console.log(req.body.ppm);
3. let conn = await db.getConnection();
4. let query = `INSERT INTO data VALUES ('${req.body.ppm}',NULL)`;
5. await db.executeQuery(conn, query);
6. conn.release();

});

Pada **Segmen Program NodeJS (2)** di atas merupakan bagian untuk memproses *endpoint* /sendPPM. *Endpoint* ini berguna untuk menerima data ppm yang dikirim dari NodeMCU. Pada baris 49, data ppm dimasukkan ke dalam database.

Segmen Program NodeJS (3)

1. app.get("/", function (req, res) {
2. res.render("index");

});

Pada **Segmen Program NodeJS (3)** di atasmerupakan bagian untuk memproses *endpoint* /.

*Endpoint* ini digunakan untuk membuka halaman utama *web application.* Apabila diakses, *server*  akan merender halaman html.

Segmen Program NodeJS (4)

1. app.get("/api/getData", async function (req, res) {
2. let conn = await db.getConnection();
3. let query = `SELECT ppm,time FROM data ORDER BY time DESC LIMIT 1`;
4. let result = await db.executeQuery(conn, query);
5. conn.release();
6. res.send(result[0]);

});

Pada **Segmen Program NodeJS (4)** di atas merupakan fungsi *endpoint* yang berfungsi untuk mendapatkan data ppm terbaru. Pada baris 55-57, data diambil dari server. Kemudian pada baris 59, data tersebut dikirimkan ke halaman web yang meminta.

Segmen Program NodeJS (5)

1. app.post("/api/changeMode", async function (req, res) {
2. let state = req.body.state;
3. let conn = await db.getConnection();
4. let query = `INSERT INTO mode VALUES('${state}',null)`;
5. let result = await db.executeQuery(conn, query);
6. conn.release();
7. res.status(200).send("Success");

});

Pada **Segmen Program NodeJS (5)** di atas merupakan fungsi *endpoint* yang berfungsi untuk menyimpan perubahan mode / state sensor sesuai input dari pengguna. Pada baris 63 – 65, data tersebut disimpan di *database*.

Segmen Program NodeJS (6)

1. app.get("/api/getState", async function (*req*, *res*) {
2. let conn = await db.getConnection();
3. let query = `SELECT \* FROM mode ORDER BY time DESC LIMIT 1`;
4. let result = await db.executeQuery(conn, query);
5. conn.release();
6. let mode1;
7. if (result.length == 0) {
8. mode1 = 1;
9. } else {
10. mode1 = result[0].state;
11. }
12. console.log(mode1);
13. *res*.send(mode1 + "");
14. });
15. app.listen(3000);

Pada **Segmen Program NodeJS (6)** di atas merupakan fungsi *endpoint* yang berfungsi untuk mendapatkan state dari sensor. State tersebut diambil dari database pada baris 70 – 72. State tersebut dimasukkan ke dalam variable mode1 dan dikirim di baris 82. Pada baris 84 digunakan untuk menjalankan server pada port 3000.

1. **Web-Client (HTML & JavaScript Source Code)**

Berikut di bawah akan dijelaskan mengenai segmen *markup* dan program untuk *web-app* yang terdapat pada *environment* HTML & JavaScript.

Segmen Markup HTML (1)

1. <!DOCTYPE html>
2. <html lang="en">
3. <head>
4. <meta charset="UTF-8" />
5. <meta name="viewport" content="width=device-width, initial-scale=1.0" />
6. <title>CO Monitor</title>
7. <link
8. href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.0/css/bootstrap.min.css"
9. rel="stylesheet"
10. integrity="sha384-9aIt2nRpC12Uk9gS9baDl411NQApFmC26EwAOH8WgZl5MYYxFfc+NcPb1dKGj7Sk"
11. crossorigin="anonymous"
12. />

</head>

Pada **Segmen Markup HTML (1)** di atas merupakan bagian *head* pada HTML. Pada baris 7-12, digunakan untuk mengimport framework css, bootstrap.

Segmen Markup HTML (2)

1. <body>
2. <div class="container">
3. <div class="row mt-4">
4. <div class="col text-center">
5. <p>Current PPM is :</p>
6. <p><h1 id="ppm"></h1></p>
7. <p>Last update on <span id="time"></span></p>
8. <p class="text-muted text-small">
9. Auto Update every 5 seconds

</p>

Pada **Segmen Markup HTML (2)** di atas, baris 3-19 digunakan untuk membuat *log* data dari sensor dalam bentuk tabel dan baris 15-18 digunakan untuk membuat *button* yang berfungsi untuk menghapus *log* data.

Segmen Markup HTML (3)

1. <button id="btn-TriggerSensor"
2. class="button"
3. style="font-size: 1vw;">
4. ON SENSOR
5. </button>
6. <h1 class="main\_\_title"
7. style="font-size: 3vw;">
8. Analog Value Meter
9. </h1>
11. <div class="gauge-container">
12. <div class="gauge"></div>
13. </div>
14. </main>

Pada **Segmen Markup HTML (3)** di atas, baris 1-4 digunakan untuk membuat *button* yang berfungsi sebagai *trigger* (ON/OFF) dari sensor. Baris 7-14 digunakan untuk membuat *gauge meter*.

Segmen Markup HTML (4)

1. <svg width="0"
2. height="0"
3. version="1.1"
4. class="gradient-mask"
5. xmlns="http://www.w3.org/2000/svg">
6. <defs>
7. <linearGradient id="gradientGauge">
8. <stop class="color-red" offset="0%"/>
9. <stop class="color-yellow" offset="17%"/>
10. <stop class="color-green" offset="40%"/>
11. <stop class="color-yellow" offset="87%"/>
12. <stop class="color-red" offset="100%"/>
13. </linearGradient>
14. </defs>
15. </svg>

Pada **Segmen Markup HTML (4)** di atas berfungsi untuk menampilkan komponen dalam bentuk *format* SVG (Scalable Vector Graphics). Baris 7-13 berfungsi untuk menentukan gradasi warna yang akan digunakan pada *gauge meter*.

Segmen Markup HTML (5)

1. <script src="https://cdnjs.cloudflare.com/ajax/libs/toastr.js/latest/toastr.min.js"></script>
2. <script src='https://cdn3.devexpress.com/jslib/17.1.6/js/dx.all.js'></script>
3. <script src="/src/Gauge/index.js"></script>
4. <script src="/src/main.js"></script>
5. </body>
6. </html>

Pada **Segmen Markup HTML (5)** di atas berfungsi untuk melakukan *include library* JavaScript yang digunakan untuk membantu dalam pembuatan proyek yang terdiri dari GaugeJS dan ToastrJS. Selain *library* yang digunakan, juga dilakukan *include* untuk *source code* JavaScript yang terdapat pada *file* terpisah.

Segmen Program JavaScript (1)

1. toastr.options = {
2. "positionClass": "toast-bottom-right",
3. "progressBar": true,
4. "showEasing": "swing",
5. "hideEasing": "linear",
6. "showMethod": "fadeIn",
7. "hideMethod": "fadeOut"
8. }
9. let notify= false;

Pada **Segmen Program JavaScript (1)** di atas, baris 1-8 digunakan untuk men­-*setting* *property* notifikasi pada *library* ToastrJS. Baris 10 digunakan pembuatan variabel notify dengan nilai awal *false*.

Segmen Program JavaScript (2)

1. async function getSensorData() {
2. try {
3. const fetchAPI= await fetch('/api/sensorData');
4. const result= await fetchAPI.json();
5. if (result.alert) {
6. notify= true;
7. } else {
8. notify= false;
9. }
11. $('.gauge').each(function (index, item) {
12. let gauge = $(item).dxCircularGauge('instance');
13. let gaugeElement = $(gauge.\_$element[0]);
14. gaugeElement.find('.dxg-title text')
15. .last()
16. .html(result.value);
17. gauge.value((result.value));
18. });
19. } catch (error) {
20. console.error(error);
21. }
22. }

Pada **Segmen Program JavaScript (3)** di atas merupakan fungsi yang berfungsi untuk mengambil nilai sensor yang terdapat pada API *back-end server*. Baris 3-4 berfungsi untuk melakukan proses *request* data ke *back-end server* untuk mengambil nilai dari sensor. Baris 6-10 berfungsi untuk melakukan pengecekan terhadap kondisi notifikasi hidup atau mati. Apabila sensor mendeteksi adanya api maka variabel notify akan diubah ke kondisi *true*, jika tidak akan diubah ke kondisi *false*. Baris 12-20 berfungsi untuk mencari elemen HTML *gauge meter* kemudian nilai sensor akan dimasukkan ke dalam *gauge* *meter* tersebut. Baris 21-23 berfungsi untuk menangkap kegagalan apabila baris program diatas gagal dijalankan dengan menggunakan *block catch* dan menampilkan pesan errornya.

Segmen Program JavaScript (3)

1. async function getAllData() {
2. try {
3. const fetchAPI= await fetch('/api/sensorData/all');
4. const result= await fetchAPI.json();
5. console.log(result);
6. let rows= ``;
7. result.data.forEach((item, index) => {
8. rows+= `
9. <tr>
10. <td>${index+1}</td>
11. <td>${item.value\_log}</td>
12. <td>${item.time\_log}</td>
13. </tr>
14. `;
15. });
16. $('#log-data').html(rows);
17. } catch (error) {
18. console.error(error);
19. }
20. }

Pada **Segmen Program JavaScript (3)** di atas merupakan fungsi yang berfungsi untuk mengambil semua data sensor yang terdapat pada *database* dengan cara mengakses API dari *back-end server*. Baris 3-4 berfungsi untuk melakukan proses *request* data ke *back-end server* untuk mengambil semua data sensor dari *database.* Baris 6 berfungsi untuk mencetak hasil ke dalam *console website* dan baris 8 berfungsi untuk membuat variabel yang digunakan untuk menampung elemen HTML <tr></tr>. Baris 10-18 berfungsi untuk menampung data pada variabel elemen HTML yang kemudian nanti pada baris 20 akan dimasukkan ke dalam elemen tabel HTML. Baris 21-23 berfungsi untuk menangkap kegagalan apabila baris program diatas gagal dijalankan dengan menggunakan *block catch* dan menampilkan pesan errornya.

Segmen Program JavaScript (4)

1. $(document).ready(function() {
2. setInterval(async () => {
3. await getSensorData();
4. }, 100);
5. setInterval(async () => {
6. await getAllData();
7. }, 2000);
8. setInterval(() => {
9. if (notify) {
10. toastr.warning('FIRE DETECTED!');
11. }
12. }, 1000);

Pada **Segmen Program JavaScript (4)** di atas terdapat pada *block* JQuery dan dijalankan ketika semua elemen HTML telah selesai di-*load* terlebih dahulu. Baris 2-4 merupakan fungsi *interval* dimana setiap 100 mili detik melakukan *request* ke *back-end server* untuk pengambilan nilai sensor. Baris 6-8 merupakan fungsi *interval* dimana setiap 2 detik melakukan *request* ke *back-end server* untuk mendapatkan semua data sensor yang terdapat pada *database*. Baris 10-14 merupakan fungsi *interval* dimana akan melakukan pengecekan terhadap *alarm* sensor, jika *alarm* menyala maka pada *website* akan dikeluarkan pesan notifikasi.

Segmen Program JavaScript (5)

1. $('.gauge').each(function (index, item) {
2. let params = {
3. initialValue: 0,
4. higherValue: 255,
5. title: 'Value',
6. subtitle: '0'
7. };
9. let gauge= new GaugeChart(item, params);
10. gauge.init();
11. });

Pada **Segmen Program JavaScript (5)** merupakan lanjutan dari fungsi yang terdapat pada **Segmen Program JavaScript (4)**. Baris 2-7 berfungsi untuk membuat *object* yang digunakan sebagai parameter dari *objek* GaugeChart. Baris 9 berfungsi untuk membuat *object* GaugeChart dan melemparkan elemen HTML dan parameter yang sudah dibuat sebelumnya. Baris 11 berfungsi untuk menginisialisasi *object* GaugeChart.

Segmen Program JavaScript (6)

1. $('#btn-TriggerSensor').click(async function() {
2. try {
3. const fetchAPI= await fetch('/api/triggerSensor', {
4. method: 'POST',
5. headers: {
6. 'Content-Type': 'application/json'
7. },
8. body: JSON.stringify({
9. sensorOn: $(this).text() === 'ON SENSOR' ? 'OFF' : 'ON'
10. })
11. });
12. $(this).text(
13. $(this).text() === 'ON SENSOR' ? 'OFF SENSOR' : 'ON SENSOR'
14. );
15. } catch (error) {
16. console.error(error);
17. }
18. });

Pada **Segmen Program JavaScript (6)** di atas merupakan lanjutan dari fungsi yang terdapat pada **Segmen Program JavaScript (4)**. Baris 1 berfungsi untuk membuat *event* klik dengan menggunakan JQuery. Baris 3-11 berfungsi untuk melakukan pengiriman data dengan *method* POST mengakses API pada *back-end server*. Baris 13-14 berfungsi untuk mengganti tulisan pada *button* ON/OFF sensor. Baris 16-18 berfungsi untuk menangkap kegagalan apabila baris program diatas gagal dijalankan dengan menggunakan *block catch* dan menampilkan pesan errornya.

Segmen Program JavaScript (7)

1. $('#btn-clearLog').click(async function() {
2. try {
3. const fetchAPI= await fetch('/api/sensorData/all', {
4. method: 'DELETE'
5. });
6. const result= await fetchAPI.json();
8. } catch (error) {
9. console.log(error);
10. }
11. });
12. });

Pada **Segmen Program JavaScript (8)** di atas merupakan lanjutan dari fungsi yang terdapat pada **Segmen Program JavaScript (4)**. Baris 1 berfungsi untuk membuat *event* klik dengan menggunakan JQuery. Baris 3-6 berfungsi untuk melakukan *request* dengan *method* DELETE ke API *back-end server* untuk menghapus semua data *log* sensor pada *database*. Baris 8-9 berfungsi untuk menangkap kegagalan apabila baris program diatas gagal dijalankan dengan menggunakan *block catch* dan menampilkan pesan errornya.

1. **Data Diri**

****

NRP : 217116615

Nama : Jonathan Sugianto