LAPORAN TEMPAT SAMPAH DENGAN PENDETEKSI GERAKAN

PENGEMBANGAN APLIKASI WEB

Nama Anggota:

1. Pradipa Yogananda 20230140103
2. Imron Satrio Purnomo 20230140122
3. M. Alfiansyah Azad 20230140126
4. Harits Zhafran Nayotama 20230140141
5. Muhammad Razzin Ayyuri 20230140121
6. Rafli Ramadhani Oktavianto Khufi 20230140127

FAKULTAS TEKNIK
PROGRAM STUDI TEKNOLOGI INFORMASI
UNIVERSITAS MUHAMMADIYAH YOGYAKARTA

2025/2026

DAFTAR ISI

BAB 1 PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi informasi dan Internet of Things (IoT) mendorong terciptanya solusi cerdas untuk permasalahan sehari-hari, salah satunya pengelolaan sampah. Tempat sampah konvensional masih memiliki keterbatasan, seperti kontak fisik langsung dan kurangnya efisiensi pemantauan. Oleh karena itu, dikembangkan sistem **Tempat Sampah dengan Pendeteksi Gerakan** yang terintegrasi dengan **aplikasi web** untuk memantau dan mengelola data secara digital.

1.2 Rumusan Masalah

1. Bagaimana merancang sistem tempat sampah dengan pendeteksi gerakan?
2. Bagaimana memodelkan sistem menggunakan UML?
3. Bagaimana mengimplementasikan aplikasi web sebagai media monitoring dan pengelolaan data?

1.3 Tujuan Penelitian

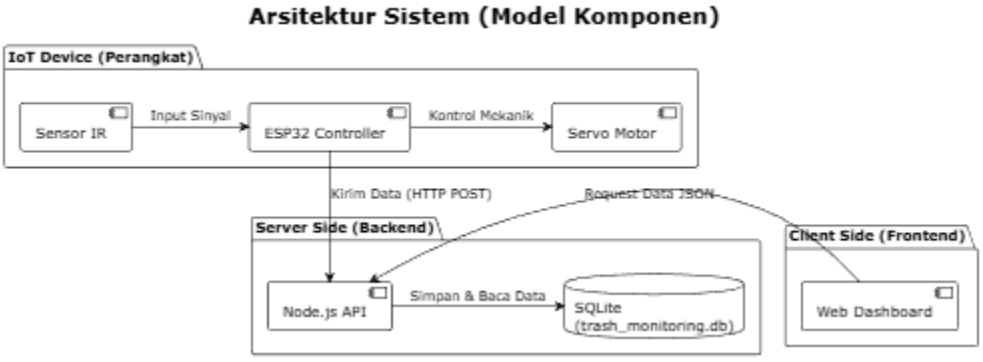
1. Merancang sistem tempat sampah otomatis berbasis pendeteksi gerakan.
2. Membuat pemodelan sistem menggunakan UML.
3. Mengembangkan aplikasi web untuk monitoring dan pengelolaan data.

1.4 Manfaat

- Meningkatkan kebersihan dan kenyamanan pengguna.
- Memudahkan pengelolaan dan pemantauan tempat sampah.
- Sebagai media pembelajaran pengembangan aplikasi web.

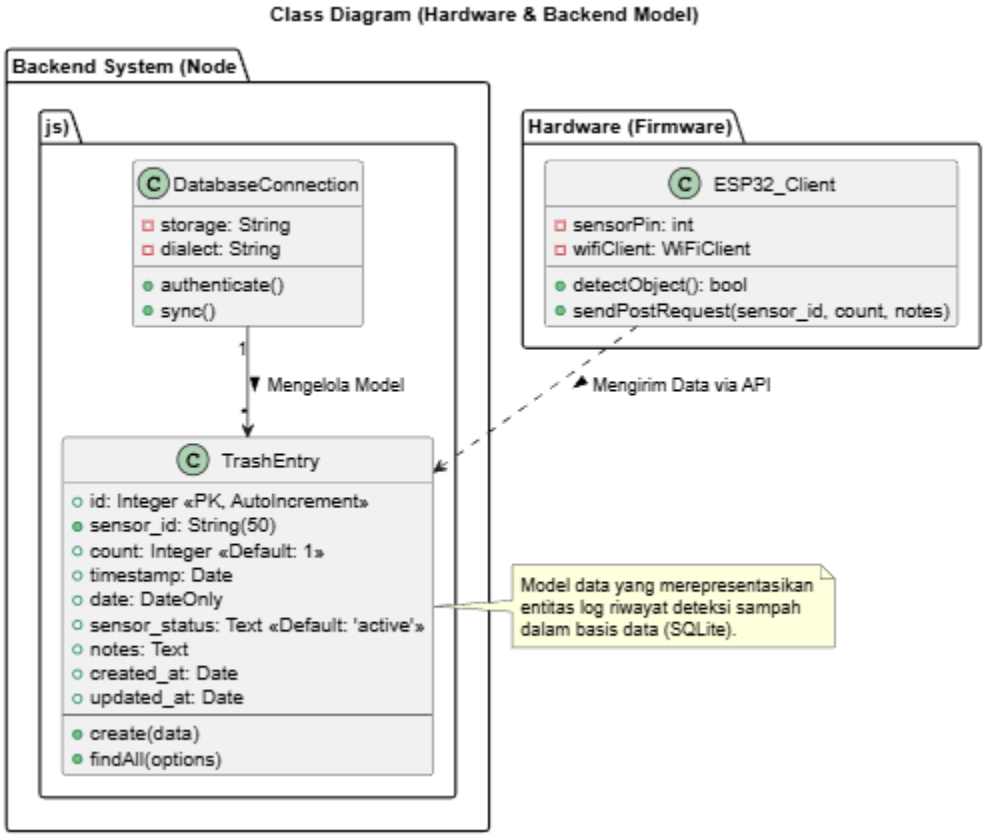
BAB 2 UML

2.1 Arsitektur



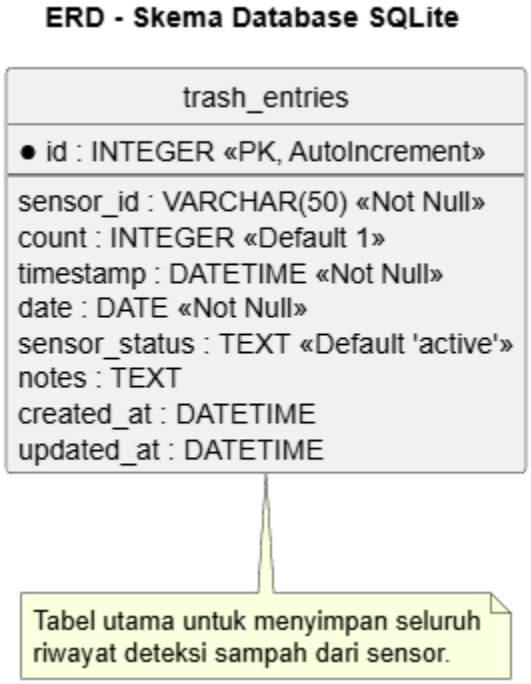
Deskripsi: Diagram arsitektur ini dirancang menggunakan model komponen untuk menggambarkan topologi sistem secara sederhana. Sensor IR dan Motor Servo terhubung ke ESP32 sebagai pengendali utama. ESP32 mengirimkan data deteksi melalui protokol HTTP ke backend berbasis Node.js. Data tersebut kemudian disimpan ke dalam database lokal SQLite (trash_monitoring.db) dan ditampilkan ke pengguna melalui Web Dashboard.

2.2 Class Diagram



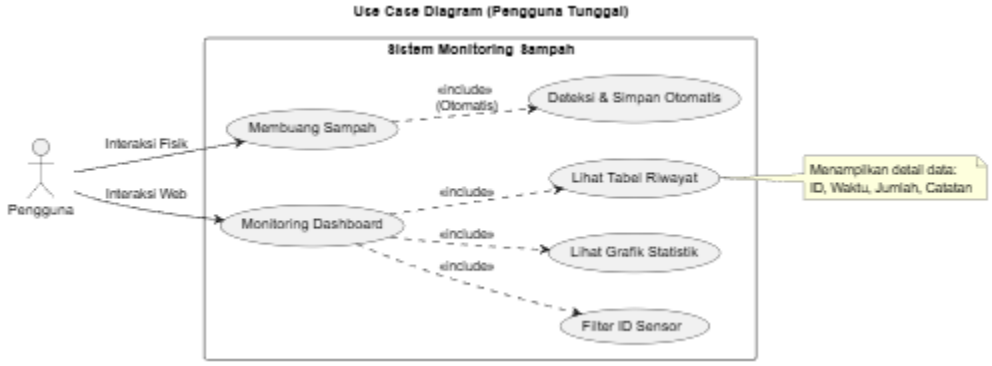
Penjelasan: Class diagram menggambarkan struktur kelas pada sisi firmware dan backend. Kelas ESP32_Client menangani pembacaan sensor dan komunikasi jaringan. Di sisi server, kelas TrashEntry merepresentasikan model data yang dibangun menggunakan Sequelize ORM. Atribut kelas ini (seperti sensor_id, count, notes) disesuaikan dengan kebutuhan pencatatan log pada database SQLite.

2.3 Entity Relationship Diagram



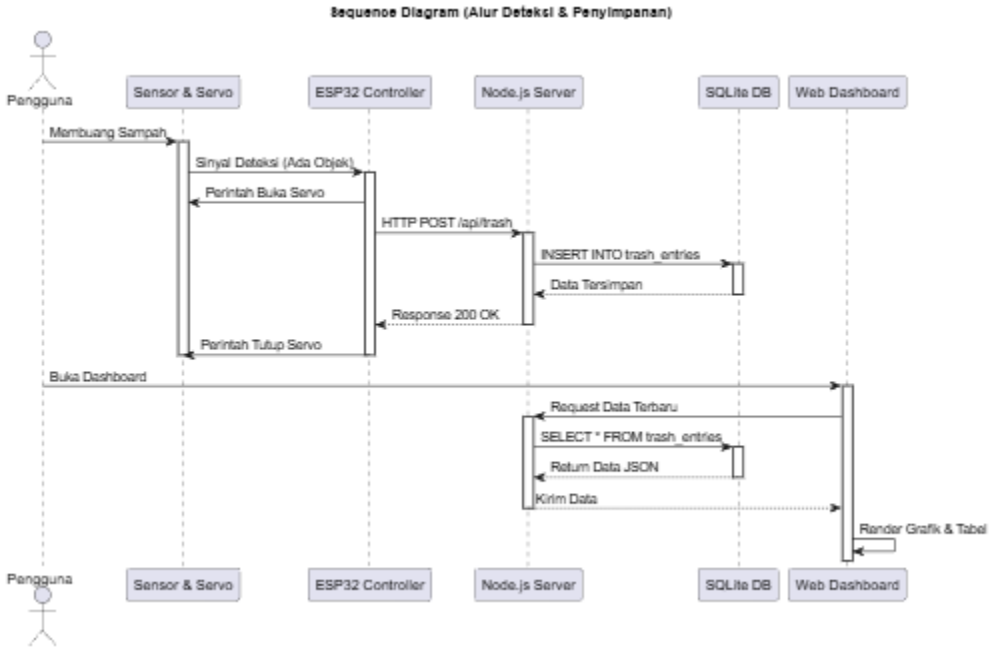
Penjelasan: ERD ini menjelaskan desain basis data SQLite yang digunakan. Sistem hanya memiliki satu tabel utama yaitu trash_entries. Kolom-kolom kunci meliputi sensor_id (identitas alat), timestamp (waktu kejadian), dan count (jumlah deteksi). Struktur tabel ini dirancang untuk mendukung fitur visualisasi grafik dan tabel riwayat pada dashboard web.

2.4 Use Case Diagram



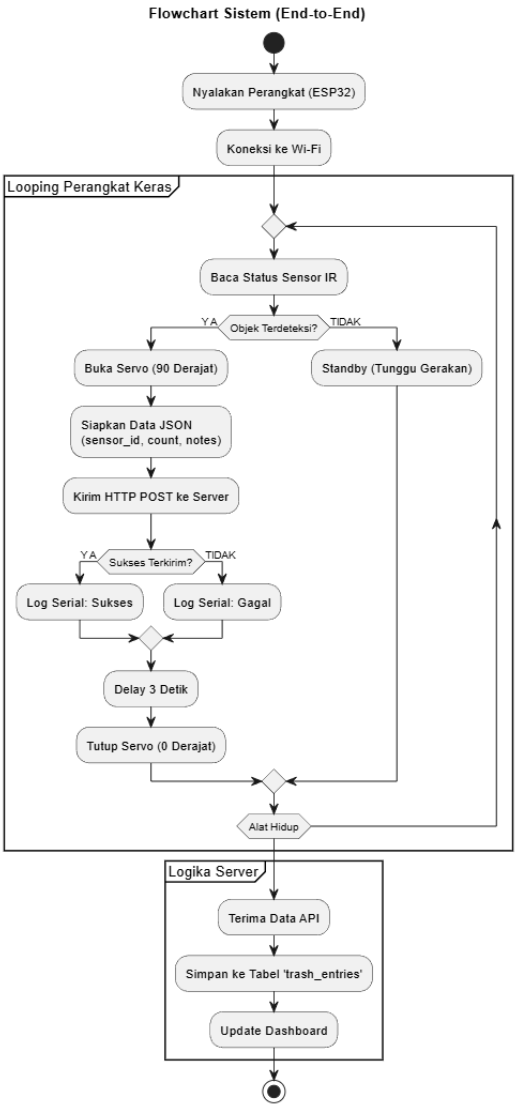
Penjelasan: Use Case Diagram menunjukkan interaksi antara Pengguna dengan sistem. Pengguna memiliki peran ganda: melakukan interaksi fisik (membuang sampah) yang memicu pencatatan otomatis, dan melakukan interaksi digital (monitoring) melalui web dashboard. Fitur web mencakup melihat grafik statistik mingguan, memfilter data sensor, dan memeriksa tabel riwayat deteksi.

2.5 Sequence Diagram



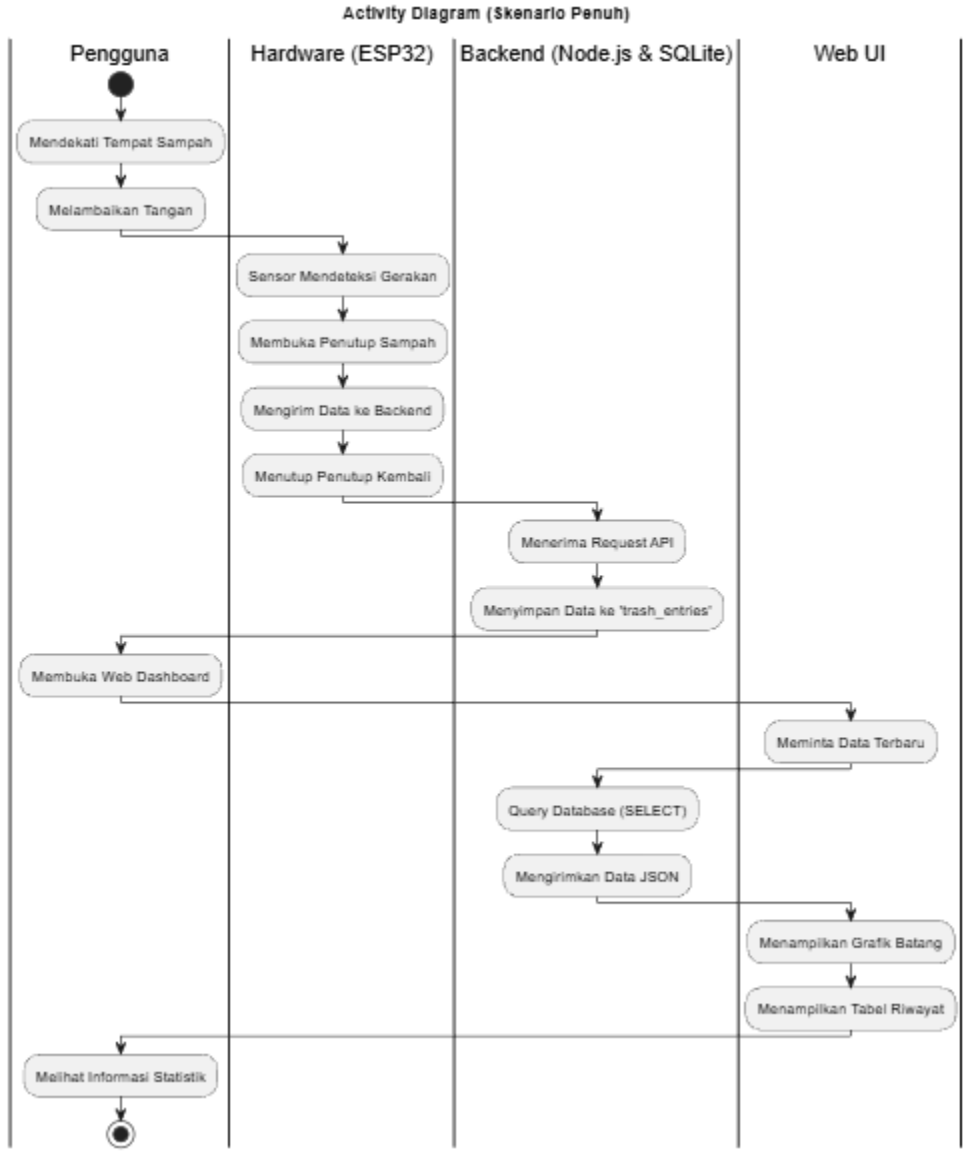
Penjelasan: Diagram ini merinci urutan proses berdasarkan waktu. Ketika sensor mendeteksi objek, ESP32 menggerakkan motor servo sekaligus mengirim data ke server Node.js. Server kemudian menyimpan data tersebut ke database SQLite. Diagram juga memvisualisasikan proses saat pengguna membuka dashboard, di mana sistem mengambil data terbaru dari database untuk ditampilkan.

2.6 Flowchart Sistem



Penjelasan: Flowchart menggambarkan algoritma yang berjalan pada mikrokontroler. Setelah terhubung ke Wi-Fi, sistem masuk ke perulangan utama (loop) untuk memantau sensor. Jika gerakan terdeteksi, sistem melakukan aksi mekanik (buka tutup) dan aksi jaringan (kirim data ke server). Logika ini memastikan data hanya dikirim saat ada aktivitas nyata, untuk efisiensi penyimpanan database.

2.7 Activity Diagram



Penjelasan: Activity Diagram memetakan alur kerja sistem secara menyeluruh menggunakan swimlanes untuk memisahkan tanggung jawab. Diagram ini memperjelas batasan tugas antara aktivitas fisik pengguna, respon otomatis perangkat keras, pemrosesan data di server backend, hingga visualisasi informasi pada antarmuka web.

BAB 3 APLIKASI

3.1 Deskripsi Aplikasi

Aplikasi web tempat sampah dengan pendeteksi gerakan berfungsi sebagai media monitoring dan pengelolaan data. Aplikasi ini menampilkan informasi status tempat sampah secara real-time.

3.2 Teknologi yang Digunakan

- Bahasa Pemrograman: HTML, CSS, JavaScript, dan PHP
- Basis Data: MySQL
- Perangkat Pendukung: Sensor pendeteksi gerakan

3.3 Fitur Aplikasi

1. Monitoring status tempat sampah
2. Pengelolaan data pengguna
3. Tampilan data berbasis web

3.4 Implementasi Sistem

Implementasi sistem dilakukan dengan mengintegrasikan perangkat keras (sensor) dengan aplikasi web melalui basis data.

3.5 Pengujian Sistem

Pengujian dilakukan untuk memastikan setiap fitur berjalan sesuai dengan fungsinya.

3.6 Tampilan Halaman Web/Aplikasi

Implementasi Antarmuka Dashboard

1. Halaman dashboard IR Trash Monitoring System berfungsi sebagai pusat pemantauan data sensor secara real-time. Tampilan antarmuka terdiri dari empat komponen utama:
2. Panel Filter & Statistik: Bagian atas memuat fitur Filter Sensor (opsi: Semua, BIN-01, BIN-02, BIN-03) untuk memilah data berdasarkan perangkat spesifik. Disandingkan dengan kartu ringkasan yang menampilkan jumlah deteksi hari ini dan total keseluruhan.

3. Grafik Mingguan: Bagian tengah menampilkan Grafik 7 Hari Terakhir dalam bentuk diagram batang, memberikan visualisasi tren aktivitas pembuangan sampah selama satu minggu ke belakang.
4. Tabel Riwayat Deteksi: Bagian bawah menyajikan log data detail dalam format tabel. Kolom yang ditampilkan meliputi ID, Sensor ID, Tanggal, Waktu, Count, dan Notes, sesuai dengan data yang dikirimkan oleh mikrokontroler.
5. Status Koneksi: Pada bagian bawah halaman (footer), terdapat indikator API Status berwarna hijau ("Connected") yang menandakan sistem terhubung stabil dengan server backend.

BAB IV KESIMPULAN

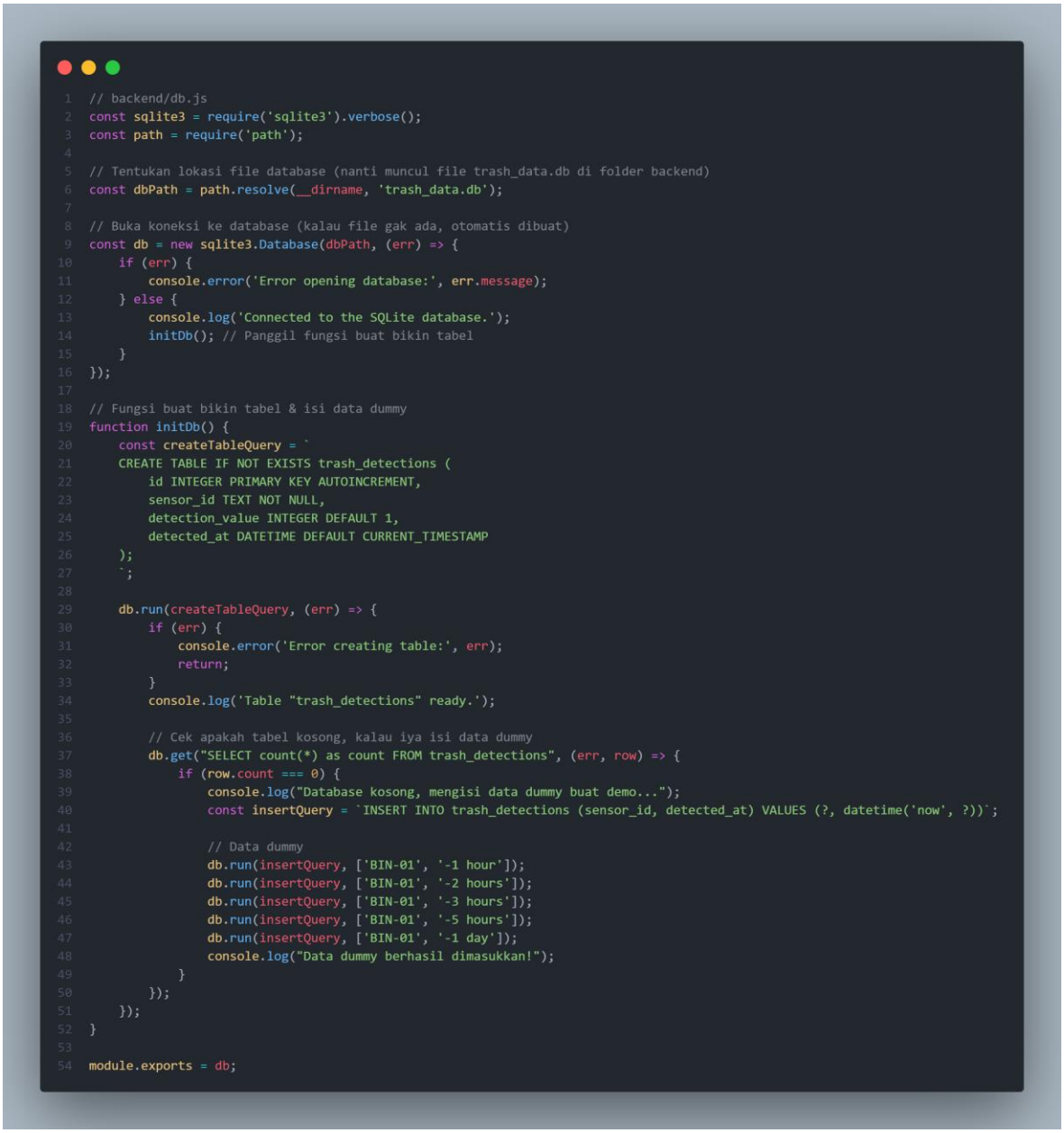
Berdasarkan hasil perancangan dan implementasi, dapat disimpulkan bahwa aplikasi web tempat sampah dengan pendeteksi gerakan berhasil dikembangkan dan dapat membantu pengelolaan sampah secara lebih efisien. Sistem ini juga menjadi sarana pembelajaran yang baik dalam pengembangan aplikasi web.

DAFTAR PUSTAKA

1. Pressman, R. S. (2015). *Software Engineering*. McGraw-Hill.
2. Sommerville, I. (2016). *Software Engineering*. Pearson Education.

LAMPIRAN

File db.js



```
// backend/db.js
const sqlite3 = require('sqlite3').verbose();
const path = require('path');

// Tentukan lokasi file database (nanti muncul file trash_data.db di folder backend)
const dbPath = path.resolve(__dirname, 'trash_data.db');

// Buka koneksi ke database (kalau file gak ada, otomatis dibuat)
const db = new sqlite3.Database(dbPath, (err) => {
    if (err) {
        console.error('Error opening database:', err.message);
    } else {
        console.log('Connected to the SQLite database.');
        initDb(); // Panggil fungsi buat bikin tabel
    }
});

// Fungsi buat bikin tabel & isi data dummy
function initDb() {
    const createTableQuery = `
    CREATE TABLE IF NOT EXISTS trash_detections (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        sensor_id TEXT NOT NULL,
        detection_value INTEGER DEFAULT 1,
        detected_at DATETIME DEFAULT CURRENT_TIMESTAMP
    );
    `;

    db.run(createTableQuery, (err) => {
        if (err) {
            console.error('Error creating table:', err);
            return;
        }
        console.log('Table "trash_detections" ready.');

        // Cek apakah tabel kosong, kalau iya isi data dummy
        db.get("SELECT count(*) as count FROM trash_detections", (err, row) => {
            if (row.count === 0) {
                console.log("Database kosong, mengisi data dummy buat demo...");
                const insertQuery = `INSERT INTO trash_detections (sensor_id, detected_at) VALUES (?, datetime('now', ?))`;

                // Data dummy
                db.run(insertQuery, ['BIN-01', '-1 hour']);
                db.run(insertQuery, ['BIN-01', '-2 hours']);
                db.run(insertQuery, ['BIN-01', '-3 hours']);
                db.run(insertQuery, ['BIN-01', '-5 hours']);
                db.run(insertQuery, ['BIN-01', '-1 day']);
                console.log("Data dummy berhasil dimasukkan!");
            }
        });
    });
}

module.exports = db;
```

File TrashSensorMain.ino

```
import { NextResponse } from 'next/server';
import { sequelize, ensureDbReady } from '@/lib/db';

export const runtime = 'nodejs';

function addDays(dateStr, delta) {
  const d = new Date(dateStr + 'T00:00:00.000Z');
  d.setUTCDate(d.getUTCDate() + delta);
  return d.toISOString().slice(0, 10);
}


export async function GET(req) {
  await ensureDbReady();

  const { searchParams } = new URL(req.url);
  const days = Math.min(90, Math.max(1, parseInt(searchParams.get('days') || '7', 10)));
  const sensor_id = searchParams.get('sensor_id');

  const end = new Date().toISOString().slice(0, 10);
  const start = addDays(end, -(days - 1));

  const whereSql = sensor_id ? 'AND sensor_id = :sensor_id' : '';
  const replacements = sensor_id ? { start, end, sensor_id } : { start, end };

  const [rows] = await sequelize.query(
    `SELECT date, COALESCE(SUM(count), 0) as total
     FROM trash_entries
     WHERE date BETWEEN :start AND :end ${whereSql}
     GROUP BY date
     ORDER BY date ASC`,
    { replacements }
  );

  // biar tanggal yang gak ada data tetap muncul (0)
  const map = new Map(rows.map(r => [r.date, Number(r.total || 0)]));
  const series = [];
  for (let i = 0; i < days; i++) {
    const d = addDays(start, i);
    series.push({ date: d, total: map.get(d) ?? 0 });
  }

  return NextResponse.json({
    success: true,
    data: { start, end, days, series },
  });
}

```

File route.js

```
import { NextResponse } from 'next/server';
import { sequelize, ensureDbReady } from '@/lib/db';

export const runtime = 'nodejs';

function addDays(dateStr, delta) {
  const d = new Date(dateStr + 'T00:00:00.000Z');
  d.setUTCDate(d.getUTCDate() + delta);
  return d.toISOString().slice(0, 10);
}

export async function GET(req) {
  await ensureDbReady();

  const { searchParams } = new URL(req.url);
  const days = Math.min(90, Math.max(1, parseInt(searchParams.get('days') || '7', 10)));
  const sensor_id = searchParams.get('sensor_id');

  const end = new Date().toISOString().slice(0, 10);
  const start = addDays(end, -(days - 1));

  const whereSql = sensor_id ? 'AND sensor_id = :sensor_id' : '';
  const replacements = sensor_id ? { start, end, sensor_id } : { start, end };

  const [rows] = await sequelize.query(
    `SELECT date, COALESCE(SUM(count), 0) as total
     FROM trash_entries
     WHERE date BETWEEN :start AND :end ${whereSql}
     GROUP BY date
     ORDER BY date ASC`,
    { replacements }
  );

  // biar tanggal yang gak ada data tetap muncul (0)
  const map = new Map(rows.map(r => [r.date, Number(r.total || 0)]));
  const series = [];
  for (let i = 0; i < days; i++) {
    const d = addDays(start, i);
    series.push({ date: d, total: map.get(d) ?? 0 });
  }

  return NextResponse.json({
    success: true,
    data: { start, end, days, series },
  });
}
```