LAPORAN PERANCANGAN DAN IMPLEMENTASI SISTEM PENERANGAN JALAN OTOMATIS BERBASIS INTERNET OF THINGS (IoT)

Kelompok 5

Disusun oleh:

Raehan Arjun Afrizal	20230140120
Sekar Asri Maghfirah	20230140128
Syihwa Moza Alika Y.P Kastella	20230140111
Dara Syauqi Darmawan	20230140140
Jihadut Tolibin	20230140125
Muhammad Bagas Prasetyo Rinaldi	20230140143

UNIVERSITAS MUHAMMADIYAH YOGYAKARTA

FAKULTAS TEKNIK

PROGRAM STUDI TEKNOLOGI INFORMASI

2025/2026

DAFTAR ISI

DAFTAR GAMBAR

BAB I PENDAHULUAN

A. Latar Belakang

Penerangan jalan umum merupakan salah satu fasilitas penting dalam menunjang aktivitas masyarakat, khususnya pada malam hari. Keberadaan lampu jalan berperan besar dalam meningkatkan keamanan, keselamatan, serta kenyamanan pengguna jalan, baik pejalan kaki maupun pengendara kendaraan. Namun, dalam penerapannya, sistem penerangan jalan yang masih bersifat konvensional sering kali menimbulkan berbagai permasalahan, salah satunya adalah pemborosan energi listrik.

Pemborosan energi listrik pada penerangan jalan umum umumnya terjadi karena lampu jalan tetap menyala meskipun kondisi lingkungan sudah cukup terang, seperti pada pagi hari atau saat cuaca cerah. Selain itu, faktor kelalaian petugas dalam mengoperasikan sakelar lampu secara manual juga menjadi penyebab utama lampu tidak dimatikan tepat waktu. Kondisi ini tidak hanya meningkatkan konsumsi energi listrik secara berlebihan, tetapi juga berdampak pada pembengkakan biaya operasional serta berkurangnya umur pakai lampu.

Seiring dengan berkembangnya teknologi, khususnya di bidang Internet of Things (IoT), permasalahan tersebut dapat diminimalisir melalui penerapan sistem otomasi. Teknologi IoT memungkinkan perangkat elektronik untuk saling terhubung dan berkomunikasi melalui jaringan internet, sehingga dapat dikendalikan dan dipantau secara jarak jauh. Dengan memanfaatkan IoT, sistem penerangan jalan dapat dirancang agar mampu menyesuaikan kondisi lingkungan secara otomatis berdasarkan data yang diperoleh dari sensor.

Salah satu pendekatan yang dapat digunakan adalah pemanfaatan sensor cahaya untuk mendeteksi intensitas cahaya lingkungan secara real-time. Data dari sensor tersebut kemudian diproses oleh mikrokontroler, seperti ESP32, untuk menentukan kapan lampu harus menyala atau mati secara otomatis. Dengan demikian, lampu jalan hanya akan menyala ketika kondisi lingkungan benar-benar membutuhkan penerangan tambahan.

Selain sistem otomatis, integrasi dengan dashboard berbasis website juga menjadi nilai tambah dalam pengelolaan penerangan jalan. Melalui antarmuka website, petugas atau pengelola dapat memantau status lampu, melihat data penggunaan energi, serta melakukan kontrol manual apabila diperlukan. Hal ini diharapkan dapat meningkatkan efisiensi pengelolaan, transparansi data, serta kemudahan dalam proses monitoring dan evaluasi sistem penerangan jalan.

Berdasarkan uraian tersebut, diperlukan sebuah sistem penerangan jalan pintar berbasis IoT yang mampu mengoptimalkan penggunaan energi listrik melalui otomatisasi dan pemantauan jarak jauh. Sistem ini diharapkan dapat menjadi solusi yang efektif dan efisien dalam mengatasi permasalahan pemborosan energi pada penerangan jalan umum.

B. **Rumusan Masalah**

Berdasarkan latar belakang yang telah diuraikan, maka rumusan masalah dalam perancangan sistem ini adalah sebagai berikut:

1. Bagaimana merancang dan membangun sistem lampu jalan otomatis yang mampu beradaptasi dengan perubahan tingkat intensitas cahaya lingkungan secara real-time menggunakan sensor dan mikrokontroler ESP32?
2. Bagaimana mengintegrasikan sensor fisik dan perangkat IoT dengan dashboard berbasis website sehingga sistem dapat dipantau dan dikendalikan secara jarak jauh?
3. Bagaimana memastikan sistem penerangan jalan yang dibangun dapat meningkatkan efisiensi penggunaan energi listrik dibandingkan sistem konvensional?

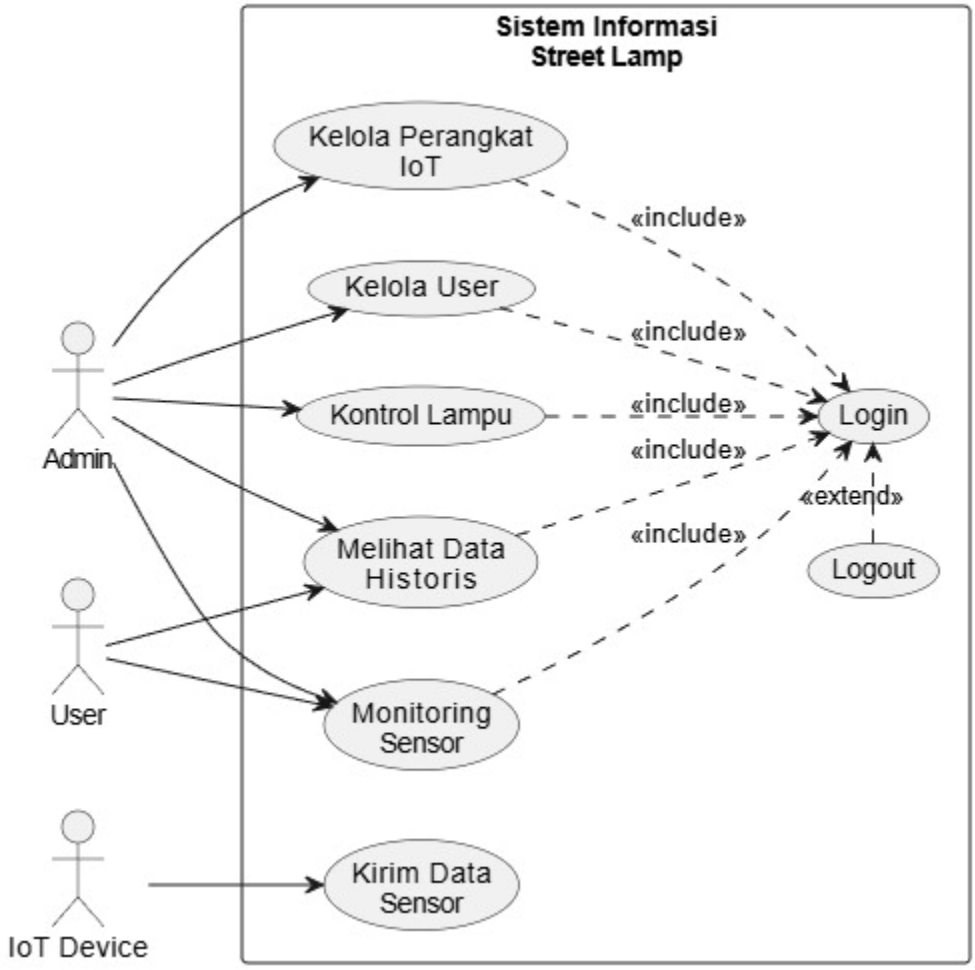
C. **Tujuan**

Adapun tujuan dari perancangan dan pembuatan sistem ini adalah sebagai berikut:

1. Membangun sistem penerangan jalan otomatis berbasis Internet of Things (IoT) dengan menggunakan mikrokontroler ESP32 dan sensor cahaya untuk mendeteksi kondisi lingkungan.
2. Mengembangkan sistem yang mampu mengontrol lampu jalan secara otomatis berdasarkan intensitas cahaya lingkungan guna mengurangi pemborosan energi listrik.
3. Menyediakan antarmuka website sebagai dashboard pemantauan yang memungkinkan pengguna untuk melihat status lampu, melakukan kontrol manual, serta menyimpan log data penggunaan sistem.
4. Meningkatkan efisiensi dan efektivitas pengelolaan penerangan jalan umum melalui penerapan teknologi otomasi dan monitoring jarak jauh.

BAB II UNIFIED MODELING LANGUAGE (UML)

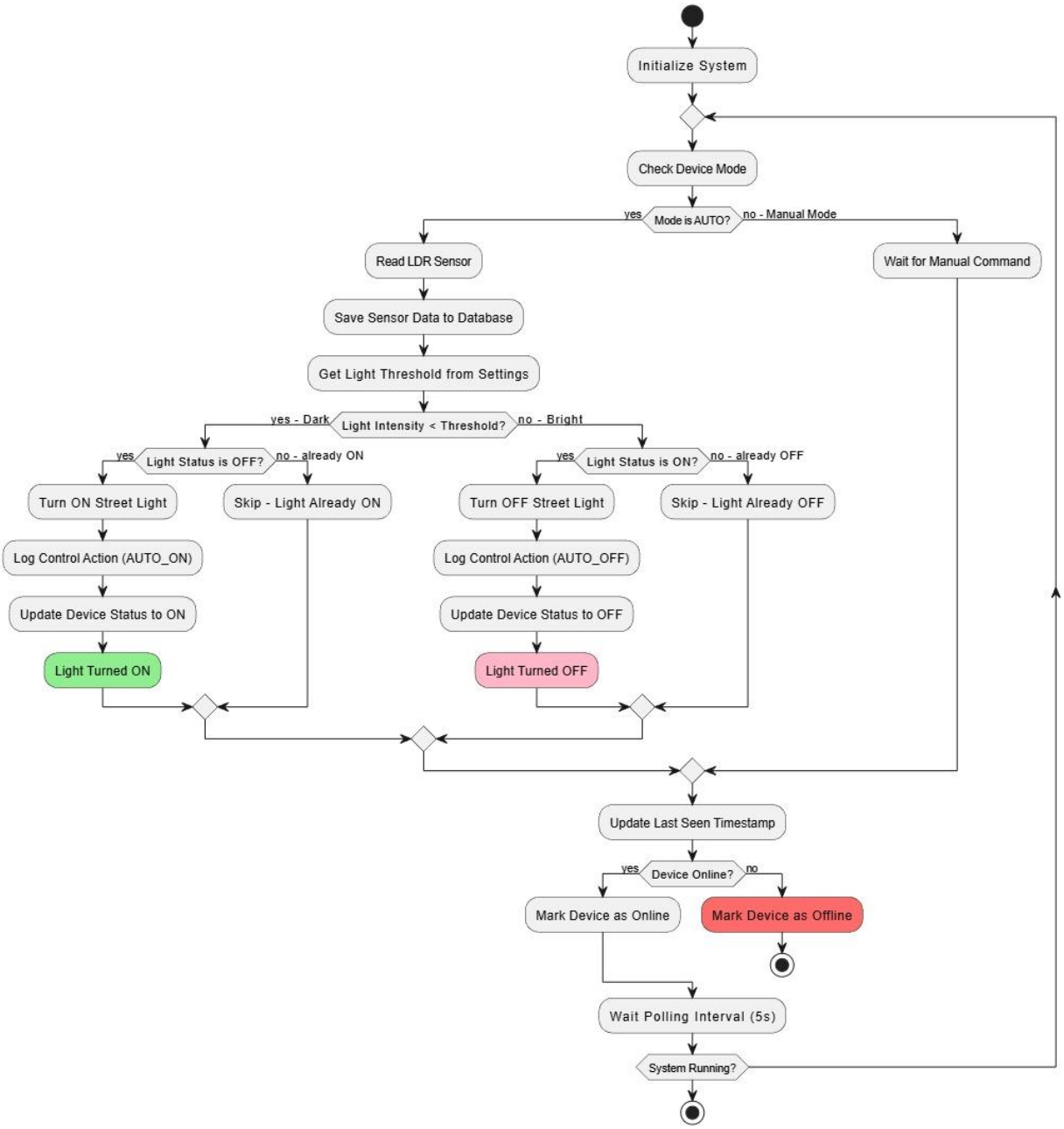
A. Use Case Diagram



Gambar 1 Use Case Diagram

Use Case Diagram ini merincikan interaksi antara tiga aktor utama dengan **Sistem Informasi Street Lamp**, yaitu Admin, User, dan IoT Device. **Admin** memiliki hak akses penuh untuk mengelola perangkat IoT, mengelola user, melakukan kontrol lampu secara manual, serta memantau data sensor dan riwayat penggunaan. Sementara itu, **User** memiliki akses terbatas yang hanya mencakup monitoring sensor dan melihat data historis saja. Seluruh fitur utama untuk aktor manusia tersebut diwajibkan melalui proses **Login** (*include*) agar dapat diakses, dengan opsi **Logout** sebagai fungsi tambahan (*extend*). Terakhir, **IoT Device** bertindak secara otomatis untuk mengirimkan data sensor ke dalam sistem tanpa memerlukan proses autentikasi login pengguna.

B. Activity Diagram



Gambar 2 Activity Diagram Kontrol Otomatis Lampu

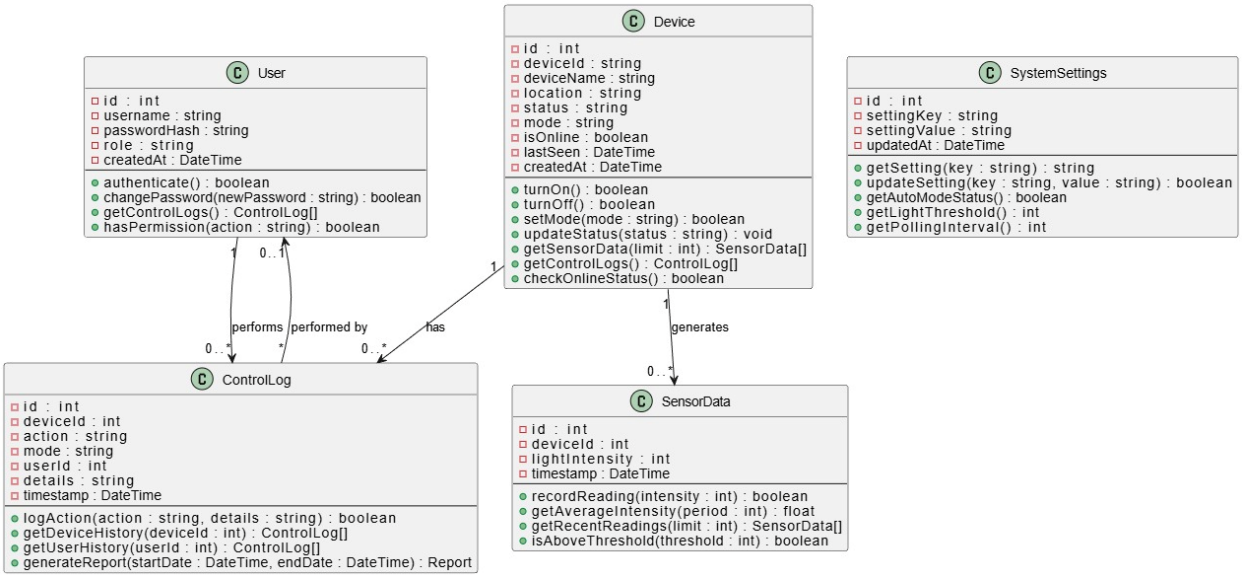
Diagram aktivitas tersebut menggambarkan mekanisme kerja sistem lampu jalan pintar yang dimulai dengan proses inisialisasi untuk memuat konfigurasi awal. Setelah aktif, sistem akan melakukan pengecekan mode operasi, yaitu antara mode otomatis atau manual. Jika sistem berada dalam mode manual, ia akan berhenti pada status menunggu instruksi langsung dari pengguna. Namun, jika berada dalam mode otomatis, sistem secara aktif akan membaca data dari sensor LDR,

menyimpannya ke database, dan mengambil nilai ambang batas (threshold) cahaya yang telah ditentukan untuk menentukan aksi selanjutnya.

Logika utama dalam mode otomatis berfokus pada efisiensi penggunaan daya dengan membandingkan intensitas cahaya sekitar terhadap ambang batas. Apabila kondisi lingkungan dideteksi gelap namun lampu masih dalam keadaan mati, sistem akan menyalakan lampu, mencatat log aktivitas, dan memperbarui status perangkat menjadi ON. Sebaliknya, jika lingkungan sudah cukup terang tetapi lampu masih menyala, sistem secara otomatis akan mematikan lampu dan memperbarui statusnya menjadi OFF. Jika status lampu sudah sesuai dengan kondisi cahaya (misalnya sudah mati saat terang), sistem akan melewati proses aktivasi tersebut agar tidak terjadi redundansi data.

Pada bagian akhir alur, sistem melakukan prosedur pemeliharaan koneksi dengan memperbarui tanda waktu (timestamp) aktivitas terakhir untuk memantau apakah perangkat masih terhubung ke jaringan. Jika perangkat terdeteksi *online*, sistem akan masuk ke fase tunggu selama 5 detik (polling interval) sebelum mengulang kembali seluruh proses pengecekan dari awal. Namun, jika koneksi terputus dan perangkat dianggap *offline*, sistem akan menandai status tersebut dan menghentikan alur kerja. Seluruh siklus ini akan terus berulang selama sistem dinyatakan masih berjalan (running).

C. Class Diagram



Gambar 3 Rancangan Class Diagram Layanan Sensor dan Autentikasi

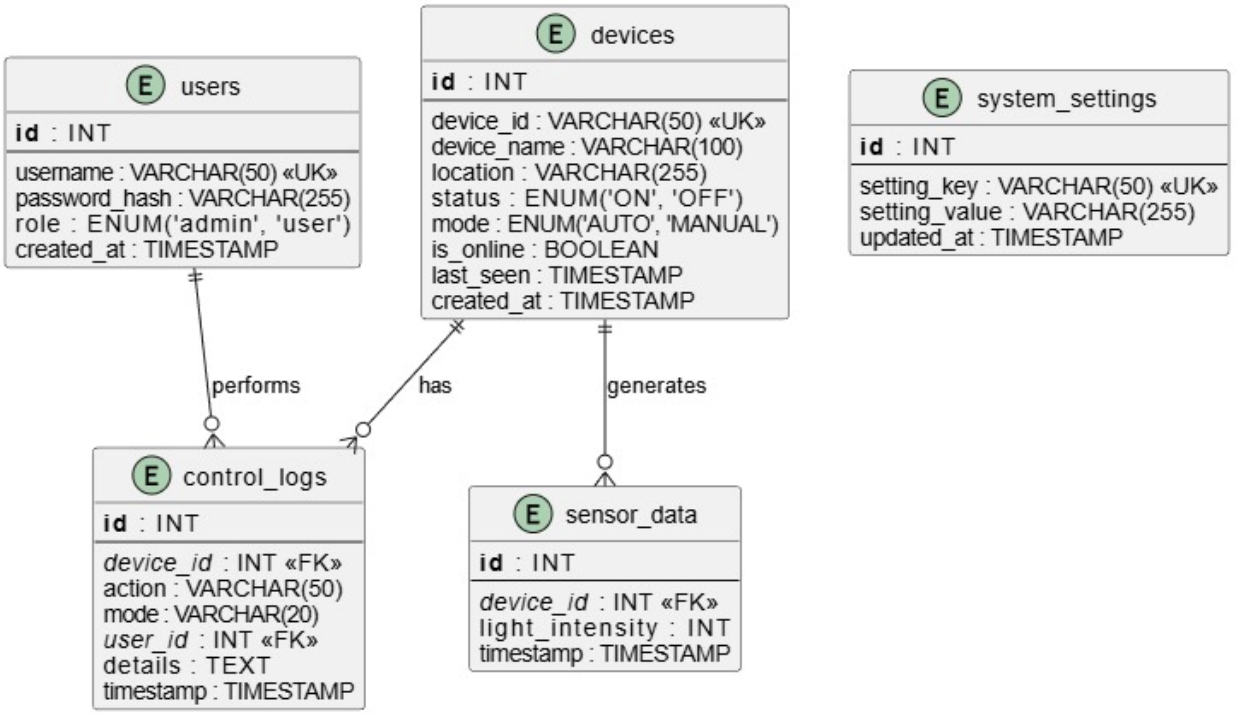
Class diagram tersebut menggambarkan struktur data dan hubungan antar objek dalam sistem **Smart Street Light**. Secara keseluruhan, sistem ini berpusat pada entitas **Device** yang merepresentasikan perangkat lampu jalan itu sendiri. Kelas Device menyimpan informasi penting

seperti lokasi, status nyala/mati, mode operasi, dan status koneksi (*isOnline*), serta memiliki metode untuk mengontrol perangkat seperti turnOn(), turnOff(), dan setMode().

Perangkat ini terhubung secara langsung dengan dua entitas data utama, yaitu **SensorData** dan **ControlLog**. Kelas Device memiliki hubungan *one-to-many* (1 ke 0..*) dengan SensorData, yang berarti satu perangkat dapat menghasilkan banyak catatan intensitas cahaya dari waktu ke waktu. Di sisi lain, Device juga berhubungan dengan ControlLog untuk mencatat setiap aksi kontrol yang dilakukan, baik secara otomatis oleh sistem maupun manual oleh pengguna. ControlLog menyimpan detail tindakan, siapa yang melakukannya (*userId*), dan kapan tindakan tersebut terjadi.

Interaksi pengguna dikelola melalui kelas **User**, yang memiliki peran (*role*) dan metode autentikasi untuk memastikan keamanan akses. Hubungan antara User dan ControlLog menunjukkan bahwa satu pengguna dapat melakukan banyak aksi kontrol (*performs*), dan setiap log aksi dapat ditelusuri kembali ke pengguna yang bertanggung jawab. Terakhir, terdapat kelas **SystemSettings** yang berfungsi secara independen untuk mengelola konfigurasi global sistem, seperti pengaturan ambang batas cahaya (*light threshold*) dan interval waktu pengambilan data (*polling interval*), yang menjadi acuan bagi perangkat dalam menjalankan mode otomatisnya.

D. Entity Relationship Diagram (ERD)



Gambar 4 Entity Relationship Diagram

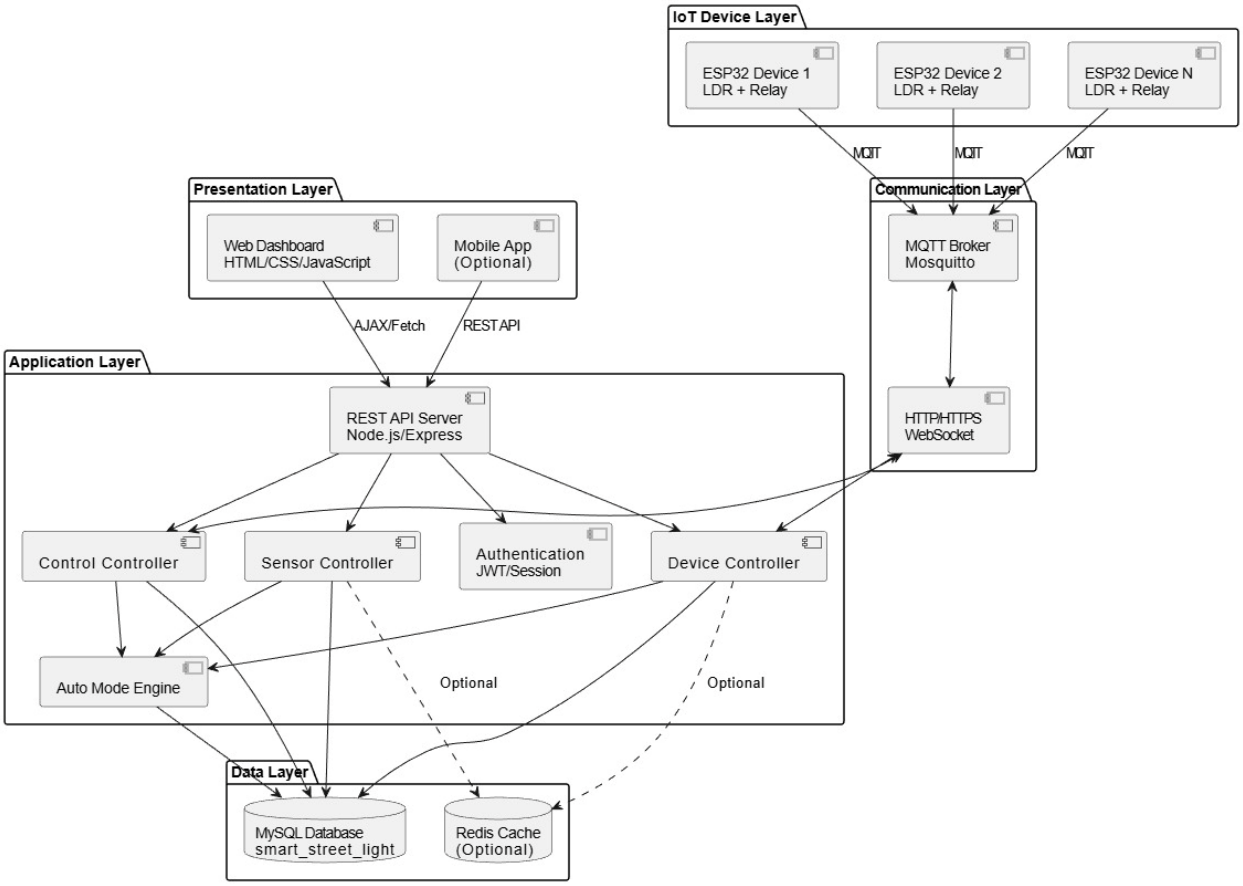
Entity Relationship Diagram (ERD) tersebut menggambarkan struktur basis data untuk sistem lampu jalan pintar yang terdiri dari lima tabel utama yang saling berelasi. Inti dari database ini adalah tabel **devices** yang menyimpan identitas unik perangkat, lokasi, status operasional (ON/OFF), serta mode kerja (AUTO/MANUAL). Tabel ini memiliki relasi *one-to-many* terhadap

tabel **sensor_data** dan **control_logs**. Artinya, satu perangkat dapat menghasilkan banyak catatan data sensor cahaya secara berkala dan dapat memiliki banyak riwayat aktivitas kontrol yang tersimpan dalam database.

Tabel **sensor_data** berfungsi sebagai tempat penyimpanan log intensitas cahaya yang dikirimkan oleh sensor LDR pada setiap perangkat melalui kunci tamu (*Foreign Key*) device_id. Sementara itu, tabel **control_logs** mencatat setiap tindakan yang terjadi pada sistem, baik yang dipicu secara otomatis oleh logika sistem maupun secara manual oleh pengguna. Tabel log ini terhubung dengan tabel **users** melalui user_id, yang memungkinkan sistem untuk melacak siapa yang melakukan perubahan status pada perangkat tertentu. Tabel **users** sendiri menyimpan informasi kredensial dan peran pengguna, seperti admin atau user biasa, untuk mengatur hak akses kontrol.

Terakhir, terdapat tabel **system_settings** yang berdiri sendiri tanpa relasi langsung ke tabel lain karena berfungsi sebagai konfigurasi global. Tabel ini menyimpan parameter penting seperti kunci pengaturan (*setting_key*) dan nilainya (*setting_value*), yang kemungkinan besar digunakan sistem untuk menentukan ambang batas cahaya (threshold) atau interval pemantauan. Secara keseluruhan, desain ERD ini memastikan integritas data antara identitas perangkat, hasil pembacaan sensor, riwayat penggunaan oleh user, dan pengaturan sistem secara terintegrasi.

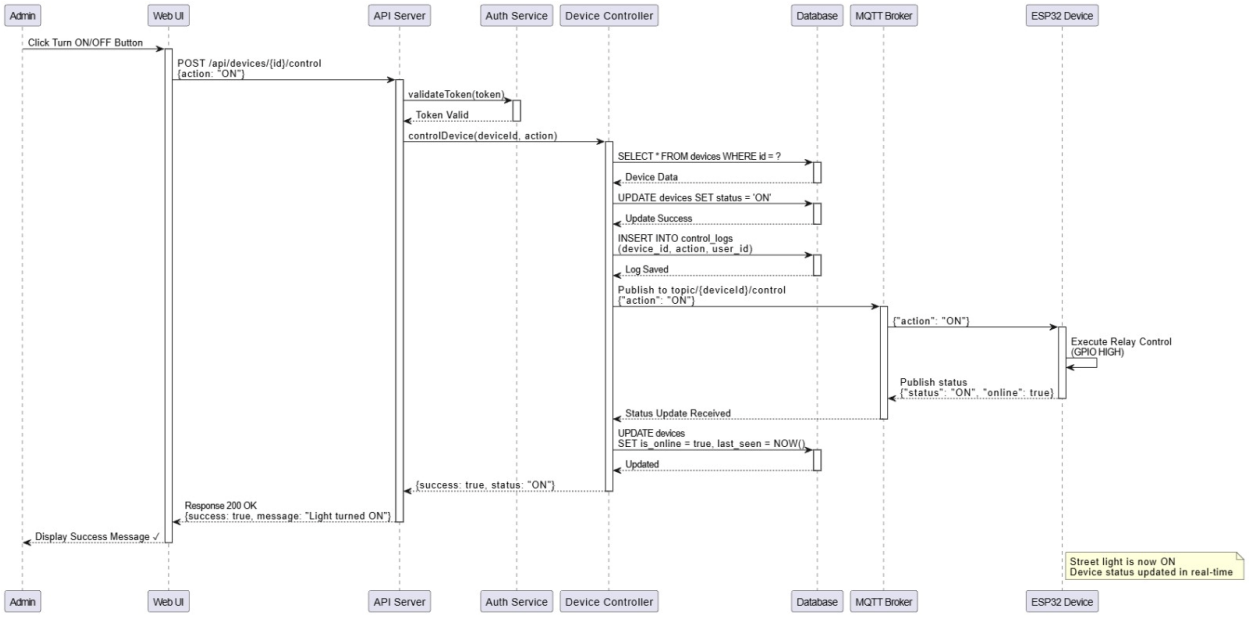
E. Arsitektur Sistem Informasi



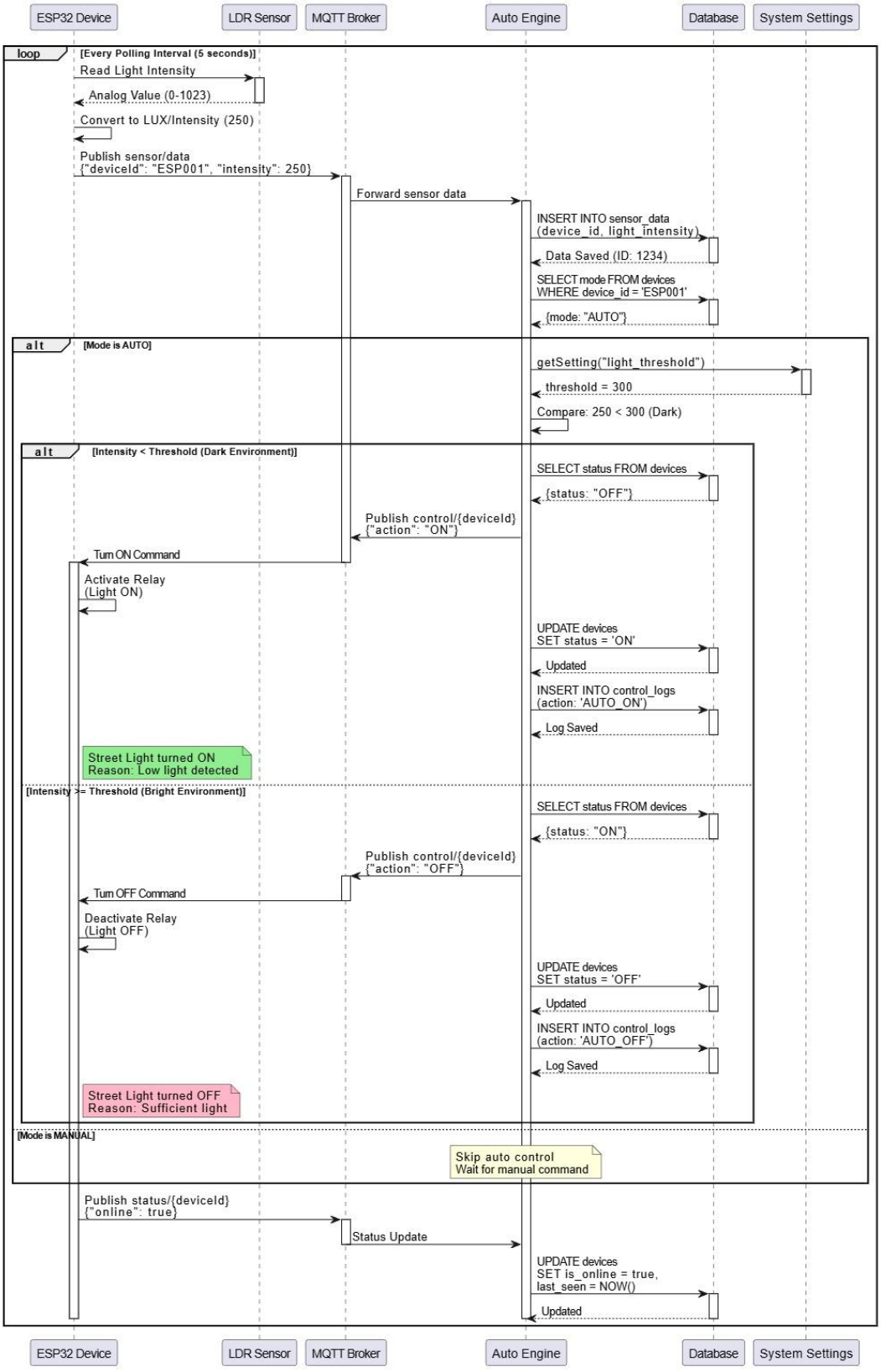
Gambar 5 Arsitektur Sistem Informasi

Sistem ini menggunakan arsitektur berlapis yang menghubungkan perangkat keras ESP32 di **IoT Device Layer** dengan server melalui protokol MQTT dan WebSocket pada **Communication Layer**. Inti operasionalnya berada di **Application Layer** berbasis Node.js/Express yang mengelola logika otomatisasi (*Auto Mode Engine*) dan kontrol perangkat, sementara seluruh data disimpan secara terpusat pada MySQL di **Data Layer**. Pengguna dapat memantau dan mengendalikan lampu secara *real-time* melalui **Presentation Layer** berupa dashboard web atau aplikasi mobile yang terhubung ke server melalui REST API.

F. Sequence



Gambar 6 Sequence Diagram

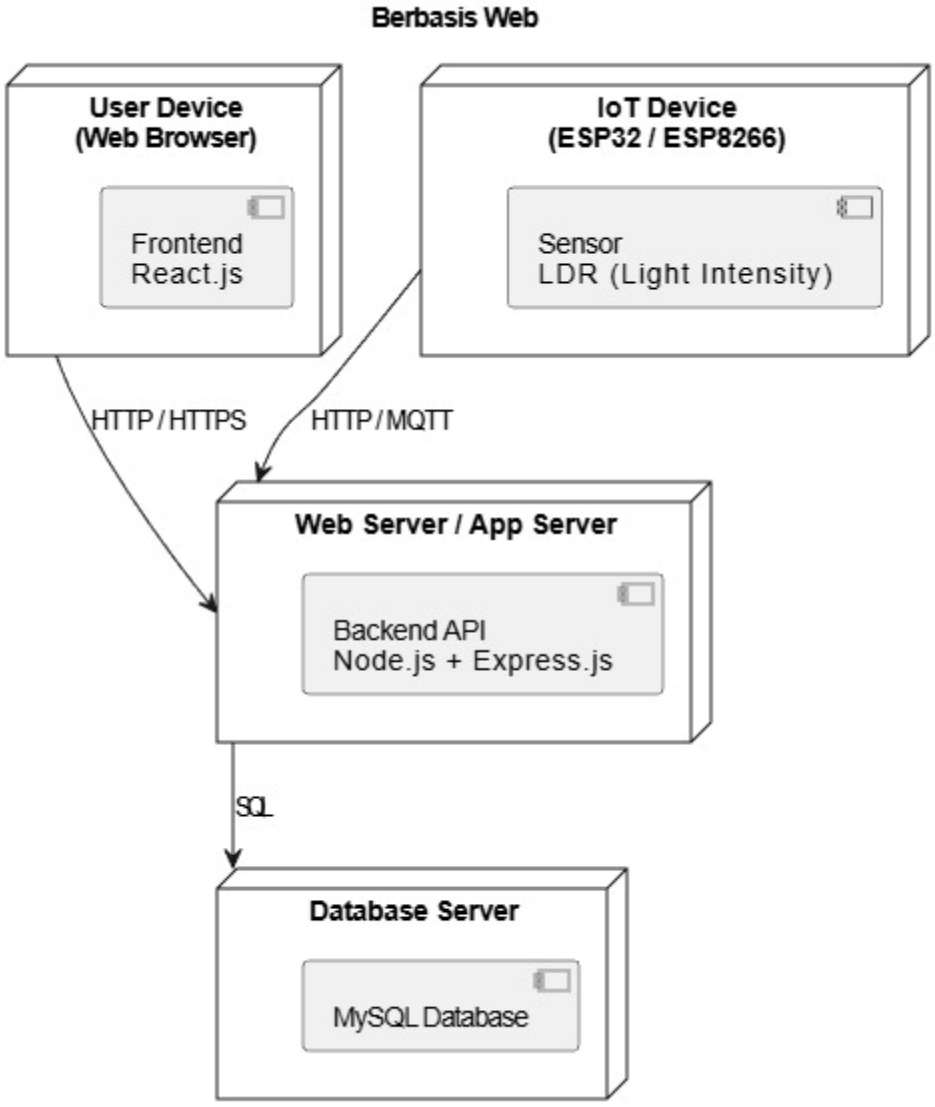


Gambar 7 Sequence Diagram

Kedua *sequence diagram* tersebut menggambarkan dua skenario pengendalian lampu jalan, yaitu melalui perintah manual pengguna dan melalui logika otomatis sistem. Pada skenario **manual**, proses dimulai ketika admin menekan tombol pada Web UI, yang kemudian mengirimkan permintaan ke API Server untuk divalidasi keamanannya sebelum instruksi diteruskan ke *Device Controller*. Pengontrol ini akan memperbarui status di database, mencatat log aktivitas, dan menerbitkan pesan melalui MQTT Broker agar perangkat ESP32 segera mengaktifkan relay lampu secara *real-time*.

Sementara itu, pada skenario **otomatis**, perangkat ESP32 secara berkala membaca intensitas cahaya dari sensor LDR dan mengirimkan datanya ke *Auto Engine* melalui MQTT Broker. *Auto Engine* kemudian menyimpan data tersebut ke database dan membandingkan nilai intensitas cahaya dengan ambang batas (*threshold*) yang diambil dari pengaturan sistem. Jika kondisi dideteksi gelap dan mode berada pada posisi "AUTO", sistem akan secara mandiri mengirimkan perintah nyala ke perangkat, memperbarui status lampu di database, dan mencatat log kontrol otomatis tanpa memerlukan intervensi manual dari pengguna.

G. Deployment Diagram

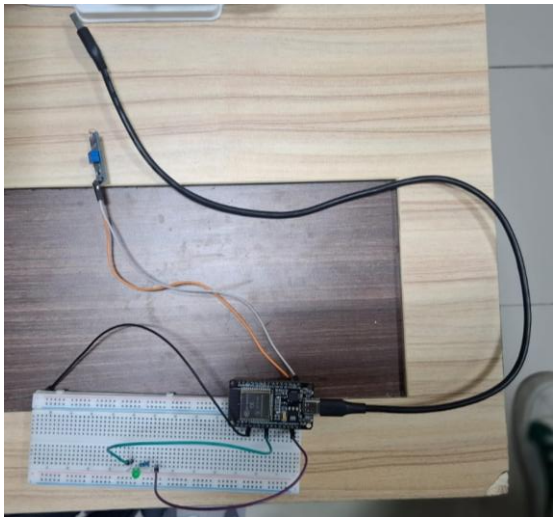


Gambar 8 Deployment Diagram Arsitektur Fisik Sistem IoT

Sistem ini menggunakan arsitektur berlapis yang menghubungkan perangkat keras ESP32 di **IoT Device Layer** dengan server melalui protokol MQTT dan WebSocket pada **Communication Layer**. Inti operasionalnya berada di **Application Layer** berbasis Node.js/Express yang mengelola logika otomatisasi (*Auto Mode Engine*) dan kontrol perangkat, sementara seluruh data disimpan secara terpusat pada MySQL di **Data Layer**. Pengguna dapat memantau dan mengendalikan lampu secara *real-time* melalui **Presentation Layer** berupa dashboard web atau aplikasi mobile yang terhubung ke server melalui REST API.

BAB III APLIKASI

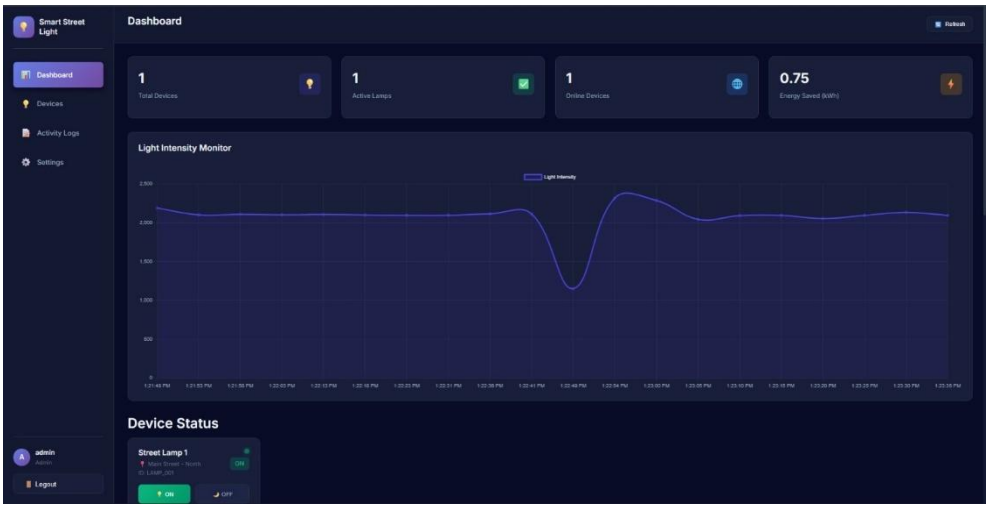
A. Foto Alat



Gambar 9 Konfigurasi Perangkat IoT Node pada Breadboard

Gambar di atas menampilkan wujud fisik purwarupa (*prototype*) Sistem Penerangan Jalan Otomatis yang telah dirakit dan diuji coba. Komponen utama yang terlihat adalah *development board* mikrokontroler **ESP32** yang terpasang pada sebuah *breadboard* putih sebagai media perakitan sirkuit tanpa solder. Rangkaian ini dilengkapi dengan modul sensor cahaya (LDR) yang terhubung ke pin input analog mikrokontroler menggunakan kabel *jumper* berwarna oranye dan putih. Sebagai indikator output, sebuah LED berwarna hijau dipasang untuk mensimulasikan lampu jalan; LED ini akan menyala ketika sensor mendeteksi penurunan intensitas cahaya. Perangkat mendapatkan suplai daya sekaligus jalur komunikasi data *serial* melalui kabel USB tipe Micro-B yang terhubung ke komputer. Implementasi perangkat keras ini membuktikan bahwa rancangan logika yang dibuat pada Activity Diagram dapat diterjemahkan menjadi sistem fisik yang berfungsi dengan baik.

B. Tampilan Website (Homepage)



Gambar 10 Tampilan Antarmuka Dashboard Monitoring Sistem Smart Street Light

Gambar 10 memperlihatkan halaman utama (*Dashboard*) dari aplikasi web "*Smart Street Light*" yang telah dikembangkan. Antarmuka ini dirancang dengan tema gelap (*dark mode*) untuk kenyamanan visual dan kesan modern. Halaman ini terbagi menjadi beberapa bagian informasi krusial sebagai berikut:

a. **Panel Statistik (Summary Cards):**

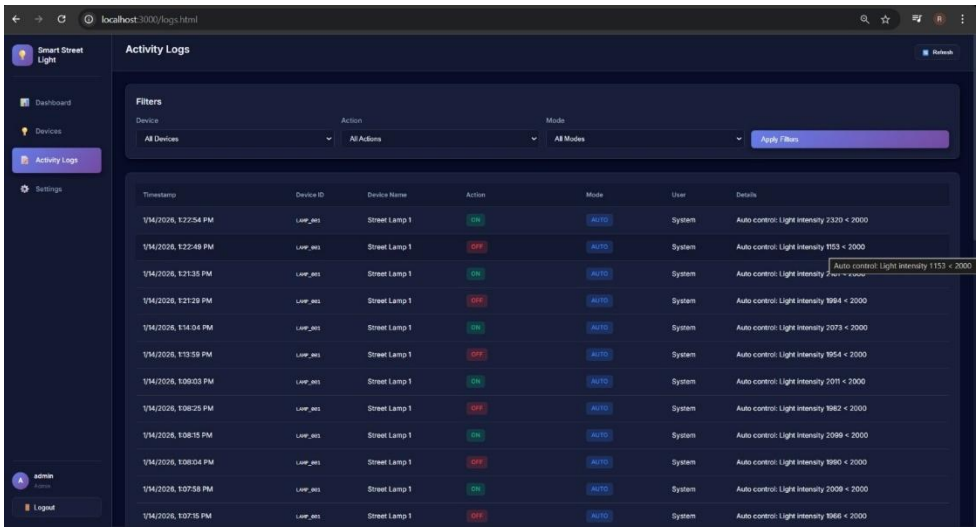
Pada bagian atas, terdapat empat kartu informasi yang menampilkan ringkasan status sistem, yaitu jumlah total perangkat (*Total Devices*), lampu yang sedang aktif (*Active Lamps*), status koneksi perangkat (*Online Devices*), serta estimasi penghematan energi (*Energy Saved*) dalam satuan kWh. Fitur ini memudahkan admin untuk mengetahui kondisi keseluruhan sistem dalam sekali pandang.

b. **Grafik Monitoring (Light Intensity Monitor)**

Bagian tengah dashboard didominasi oleh grafik garis (*line chart*) yang memvisualisasikan data intensitas cahaya yang diterima dari sensor secara *real-time*. Grafik ini bergerak dinamis mengikuti cap waktu (*timestamp*) pada sumbu horizontal, memungkinkan pengguna untuk menganalisis pola pencahayaan lingkungan sepanjang waktu.

c. **Status dan Kontrol Perangkat (Device Status):**

Pada bagian bawah, terdapat panel kontrol untuk masing-masing lampu jalan. Panel ini menampilkan identitas lampu (ID: LAMP_001), lokasi pemasangan, serta status saat ini (ON/OFF). Selain itu, terdapat tombol *toggle* manual yang memungkinkan admin untuk menyalakan atau mematikan lampu secara paksa melalui jaringan internet, mengabaikan logika otomatis sensor jika diperlukan dalam kondisi darurat atau pemeliharaan.



Gambar 11 Tampilan Halaman Activity Logs untuk Pemantauan Riwayat Sistem

Gambar 11 menyajikan halaman Activity Logs yang dirancang untuk merekam jejak digital seluruh aktivitas sistem secara kronologis. Halaman ini berfungsi sebagai fitur audit yang memungkinkan administrator untuk meninjau kinerja otomatisasi perangkat.

Terdapat beberapa komponen utama pada halaman ini:

a. **Fitur Filter:**

Di bagian atas, tersedia panel filter yang memungkinkan pengguna untuk menyortir data berdasarkan *Device* (perangkat spesifik), *Action* (jenis aksi ON/OFF), dan *Mode* operasi. Fitur ini memudahkan pencarian data spesifik di antara ribuan catatan log.

b. **Tabel Riwayat:**

Tabel utama menampilkan detail kejadian dengan kolom-kolom informasi meliputi *Timestamp* (waktu kejadian), *Device ID*, *Action* yang dieksekusi, serta *User* yang memicu aksi tersebut (baik "System" untuk otomatis maupun nama admin untuk manual).

c. **Detail Logika Otomasi:**

Pada kolom *Details*, sistem memberikan keterangan spesifik mengenai alasan pengambilan keputusan. Seperti terlihat pada gambar, sistem mencatat bahwa aksi "ON" atau "OFF" dilakukan secara otomatis (*Auto control*) karena nilai intensitas cahaya yang terbaca sensor (misalnya 1153) berada di bawah atau di atas nilai ambang batas (*threshold* 2000). Informasi ini vital untuk memverifikasi bahwa logika algoritma berjalan sesuai rencana.

Gambar 12 Tampilan Halaman Pengaturan Konfigurasi Sistem

Gambar 12 memperlihatkan halaman Settings yang berfungsi sebagai pusat kendali parameterisasi sistem. Melalui antarmuka ini, administrator dapat menyesuaikan logika operasional perangkat IoT secara dinamis tanpa perlu melakukan pemrograman ulang pada mikrokontroler.

Halaman ini terbagi menjadi dua segmen utama:

a. Konfigurasi Sistem (System Configuration):

Bagian ini memuat parameter-parameter kunci yang dapat diubah, antara lain:

a) Light Intensity Threshold: Pengaturan nilai ambang batas sensor cahaya (default: 2000). Nilai ini menjadi acuan bagi sistem untuk menentukan kondisi "gelap". Jika nilai pembacaan sensor ADC melebihi angka ini (lingkungan semakin gelap), lampu akan menyala otomatis.

b) Auto Mode Status: Opsi untuk mengaktifkan atau menonaktifkan mode otomatis secara global. Jika diset ke *Disabled*, seluruh sistem akan beralih ke kontrol manual.

c) Dashboard Refresh Interval: Pengaturan jeda waktu pembaruan data pada dashboard (default: 5000 ms atau 5 detik), yang memengaruhi seberapa *real-time* data ditampilkan kepada pengguna.

b. Informasi Sistem (System Information):

Bagian bawah menampilkan metadata statis mengenai versi perangkat lunak (v1.0.0), jenis proyek, tumpukan teknologi yang digunakan (Node.js, MySQL, ESP32), serta tujuan pengembangan sistem, yang berfungsi sebagai identitas teknis aplikasi.

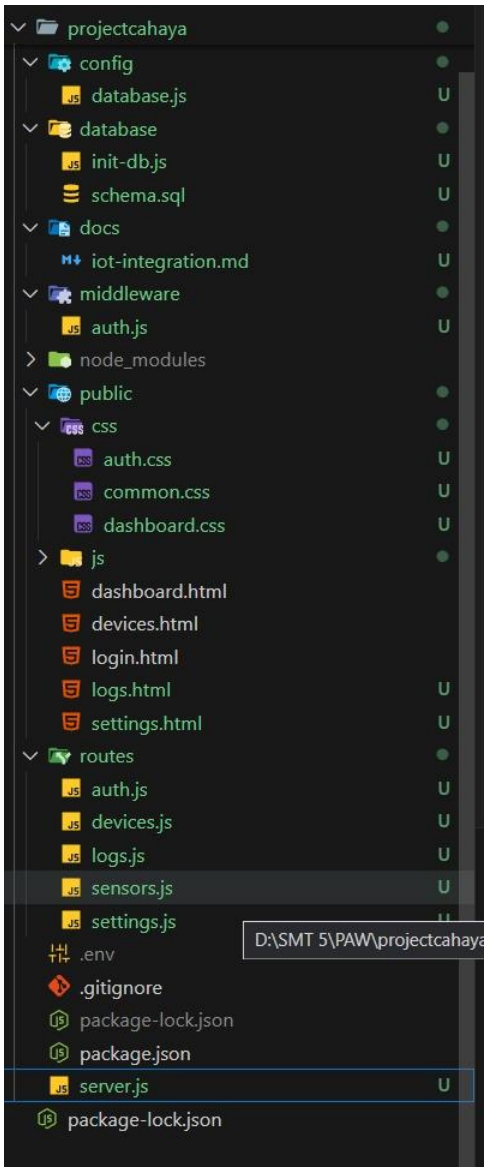
Gambar 13 Antarmuka Pengelolaan Daftar Lampu Jalan Terdaftar

Gambar 13 menampilkan halaman **Device Management** yang dirancang untuk memfasilitasi pengelolaan inventaris perangkat keras dalam sistem. Halaman ini memberikan otoritas penuh kepada administrator untuk mengatur perangkat mana saja yang diizinkan berinteraksi dengan server.

Fitur utama yang tersedia pada halaman ini meliputi:

a. **Daftar Perangkat (Device List):**
   Bagian tengah halaman menampilkan kartu informasi untuk setiap perangkat yang telah terdaftar. Informasi yang disajikan mencakup Nama Perangkat (*Street Lamp 1*), ID unik perangkat (*LAMP_001*), serta lokasi pemasangan (*Main Street - North*). Tampilan berbasis kartu (*card-based view*) ini memudahkan identifikasi visual perangkat di lapangan.
b. **Penambahan Perangkat Baru:**
   Terdapat tombol navigasi "+ Add Device" di sudut kanan atas yang berfungsi untuk membuka formulir pendaftaran perangkat baru. Fitur ini penting untuk skalabilitas sistem ketika jumlah titik lampu jalan perlu ditambah di masa mendatang.
c. **Penghapusan Perangkat:**
   Ikon tempat sampah (*trash bin*) pada sisi kanan kartu perangkat memungkinkan admin untuk menghapus perangkat yang sudah tidak aktif atau rusak dari basis data, menjaga kebersihan dan validitas data sistem.

C. Penjelasan Susunan Folder dan Source Code



Gambar 14 Struktur Folder VS Code

Struktur folder aplikasi dirancang secara modular untuk memudahkan pengembangan dan pemeliharaan sistem. Berikut penjelasan struktur folder utama:

a. Folder config
   Berisi file konfigurasi sistem, seperti :
   - Database.js : konfigurasi koneksi database MySQL.

b. Folder database
   Digunakan untuk pengelolaan baris data :
   - Init-db.js : inisialisasi database

- Schema.sql : struktur tabel database

c. Folder middleware

Berisi middleware keamanan :

- Auth.js : validasi token JWT untuk proteksi endpoint API.

d. Folder routes

Berisi routing backend :

- Auth.js : autentikasi user
- Devices.js : manajemen perangkat IoT
- Settings.js : pengaturan sistem
- Sensors.js : penerimaan dan pengelolaan data sensor
- Logs.js : pengambilan data histori

e. Folder public

Berisi file frontend :

- Folder css : styling halaman web
- Folder js : logika frontend
- File html seperti dashboard.html, login.html, dan lainnya

f. File server.js

Merupakan file utama backend yang menjalankan server Express.js dan mengatur koneksi API, middleware, serta database. Struktur ini mendukung pemisahan antara frontend, backend, dan database sehingga sistem menjadi lebih terorganisir dan scalable.

g. Source Code ESP32 (Arduino IDE)

Program dimulai dengan mengimpor library WiFi.h, HTTPClient.h, dan ArduinoJson.h yang digunakan untuk koneksi jaringan, komunikasi HTTP, serta pengolahan data JSON. Pada bagian konfigurasi, ditentukan parameter jaringan WiFi, alamat server backend, ID perangkat, serta pin yang digunakan oleh sensor LDR dan LED.

Pada fungsi setup(), ESP32 melakukan inisialisasi komunikasi serial, konfigurasi pin output, serta mencoba terhubung ke jaringan WiFi. Jika koneksi berhasil, alamat IP ESP32 akan ditampilkan melalui Serial Monitor.

Fungsi loop() merupakan inti dari sistem yang berjalan secara berulang. ESP32 membaca nilai intensitas cahaya dari sensor LDR, kemudian menentukan kondisi lampu menyala atau mati berdasarkan nilai ambang batas tertentu. Data hasil pembacaan sensor selanjutnya dikirimkan ke server backend dalam format JSON menggunakan metode HTTP POST. Proses ini dilakukan secara periodik setiap 5 detik.

BAB IV KESIMPULAN

Berdasarkan seluruh rangkaian diagram UML yang telah dibahas, dapat disimpulkan bahwa proyek Smart Street Light ini adalah sistem IoT terintegrasi yang dirancang untuk mengotomatisasi pencahayaan jalan guna meningkatkan efisiensi energi dan kemudahan pengawasan. Secara fungsional, sistem ini memberikan kendali penuh kepada Admin untuk mengelola perangkat dan pengguna, sementara User biasa hanya dapat melakukan pemantauan. Data sensor LDR menjadi acuan utama bagi sistem untuk beroperasi secara mandiri dalam mode AUTO, namun tetap menyediakan fleksibilitas bagi manusia melalui mode MANUAL.

Dari sisi teknis, sistem ini dibangun dengan arsitektur yang sangat modular. Hubungan antar entitas dalam *Class Diagram* dan ERD memastikan bahwa setiap data sensor dan aksi kontrol tercatat secara rapi di database MySQL untuk kebutuhan audit atau riwayat historis. Penggunaan protokol komunikasi MQTT dan WebSocket memungkinkan interaksi yang cepat dan stabil antara perangkat ESP32 dengan *backend* Node.js, sehingga status lampu di lapangan selalu sinkron dengan apa yang terlihat di *dashboard* pengguna secara *real-time*.

Secara keseluruhan, dokumentasi UML ini menunjukkan kesiapan sistem dari berbagai dimensi:

- Alur Kerja: Jelasnya logika pengambilan keputusan antara kondisi gelap dan terang.
- Keamanan: Adanya proses autentikasi (Login) dan pencatatan log aksi.
- Skalabilitas: Infrastruktur yang terpisah antara server aplikasi, database, dan perangkat fisik memudahkan pengembangan di masa depan.

DAFTAR PUSTAKA

(Setiadi, Sistem Penerangan Lampu Jalan Otomatis Berbasis Internet of Things Menggunakan Sensor Cahaya dengan Telegram, 2023)

(Alamsyah, Ikorasaki, & Sitinjak, 2025)

ESP32 Documentation

MySQL Official Documentation

LAMPIRAN

Kode konfigurasi koneksi database MySQL

projectcahaya > config > database.js > ...

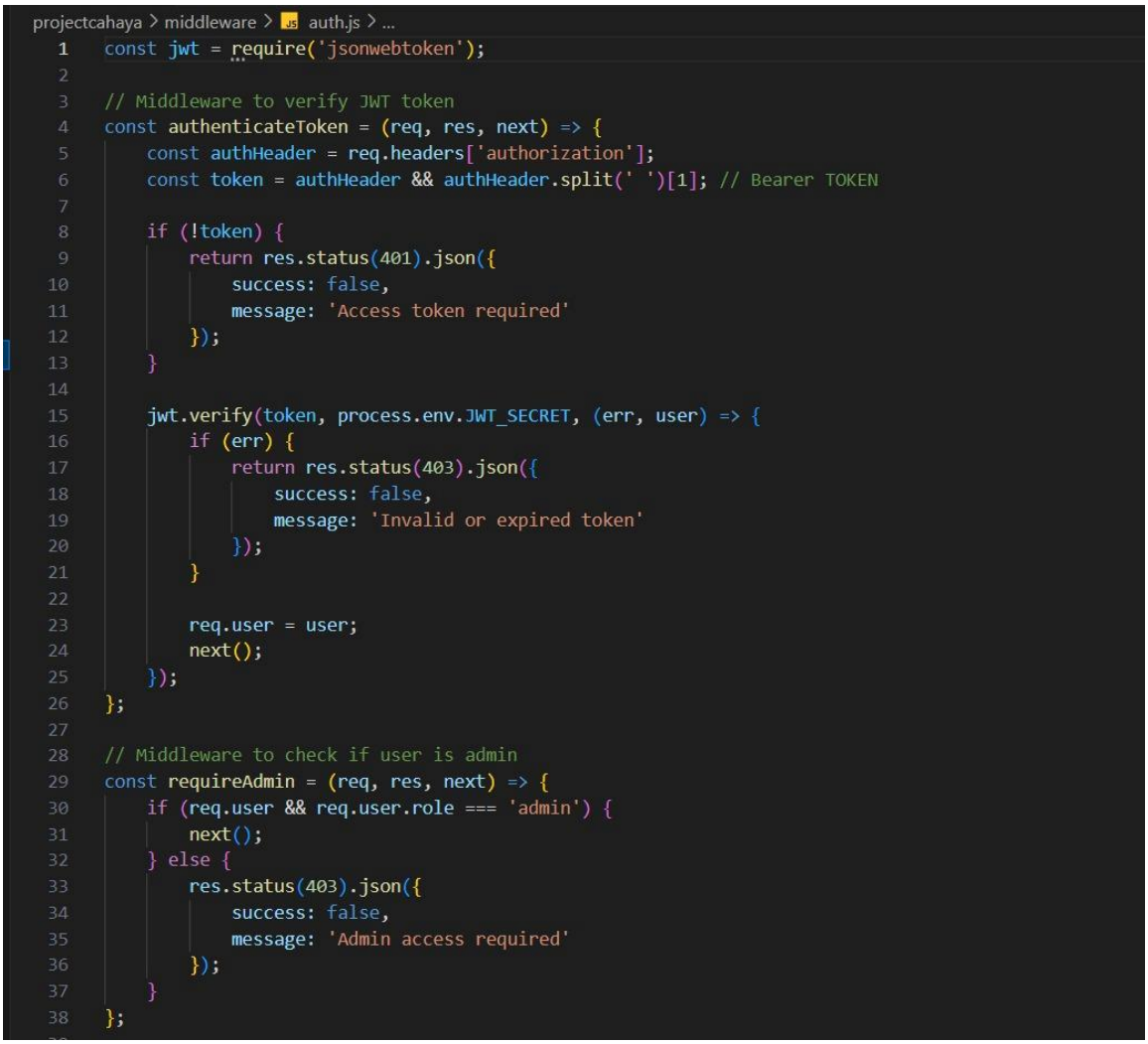
```
const mysql = require('mysql2/promise');
require('dotenv').config();

// Create connection pool
const pool = mysql.createPool({
    host: process.env.DB_HOST || '127.0.0.1',
    user: process.env.DB_USER || 'root',
    password: process.env.DB_PASSWORD || '',
    database: process.env.DB_NAME || 'smart_street_light',
    port: parseInt(process.env.DB_PORT) || 3306,
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0,
    // Add these for better compatibility with some MySQL versions
    enableKeepAlive: true,
    keepAliveInitialDelay: 0
});

// Test connection
async function testConnection() {
    try {
        const connection = await pool.getConnection();
        console.log('✅ Database connected successfully to', process.env.DB_NAME);
        connection.release();
    } catch (err) {
        console.error('❌ Database connection failed:', err.message);
        console.error('Host:', process.env.DB_HOST);
        console.error('Port:', process.env.DB_PORT);
        console.error('User:', process.env.DB_USER);
        console.error('Please ensure MySQL is running and credentials in .env are correct');
    }
}

testConnection();

module.exports = pool;
```

Kode inisialisasi struktur database

projectcahaya > database > schema.sql

```sql
-- Smart Street Light System Database Schema

-- Create database
CREATE DATABASE IF NOT EXISTS smart_street_light;
USE smart_street_light;

-- Users table for admin authentication
CREATE TABLE IF NOT EXISTS users (
    id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    role ENUM('admin', 'user') DEFAULT 'admin',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Devices table for street lamp management
CREATE TABLE IF NOT EXISTS devices (
    id INT AUTO_INCREMENT PRIMARY KEY,
    device_id VARCHAR(50) UNIQUE NOT NULL,
    device_name VARCHAR(100) NOT NULL,
    location VARCHAR(255),
    status ENUM('ON', 'OFF') DEFAULT 'OFF',
    mode ENUM('AUTO', 'MANUAL') DEFAULT 'AUTO',
    is_online BOOLEAN DEFAULT false,
    last_seen TIMESTAMP NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Sensor data table for LDR readings
CREATE TABLE IF NOT EXISTS sensor_data (
    id INT AUTO_INCREMENT PRIMARY KEY,
    device_id INT NOT NULL,
    light_intensity INT NOT NULL,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (device_id) REFERENCES devices(id) ON DELETE CASCADE,
    INDEX idx_device_timestamp (device_id, timestamp)
);
```

Kode validasi token JWT untuk proteksi endpoint API



```
projectcahaya > middleware > auth.js > ...
const jwt = require('jsonwebtoken');

// Middleware to verify JWT token
const authenticateToken = (req, res, next) => {
    const authHeader = req.headers['authorization'];
    const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

    if (!token) {
        return res.status(401).json({
            success: false,
            message: 'Access token required'
        });
    }

    jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
        if (err) {
            return res.status(403).json({
                success: false,
                message: 'Invalid or expired token'
            });
        }

        req.user = user;
        next();
    });
};

// Middleware to check if user is admin
const requireAdmin = (req, res, next) => {
    if (req.user && req.user.role === 'admin') {
        next();
    } else {
        res.status(403).json({
            success: false,
            message: 'Admin access required'
        });
    }
};
```

Kode routes pengambilan data histori

projectcahaya > routes > logs.js > ...

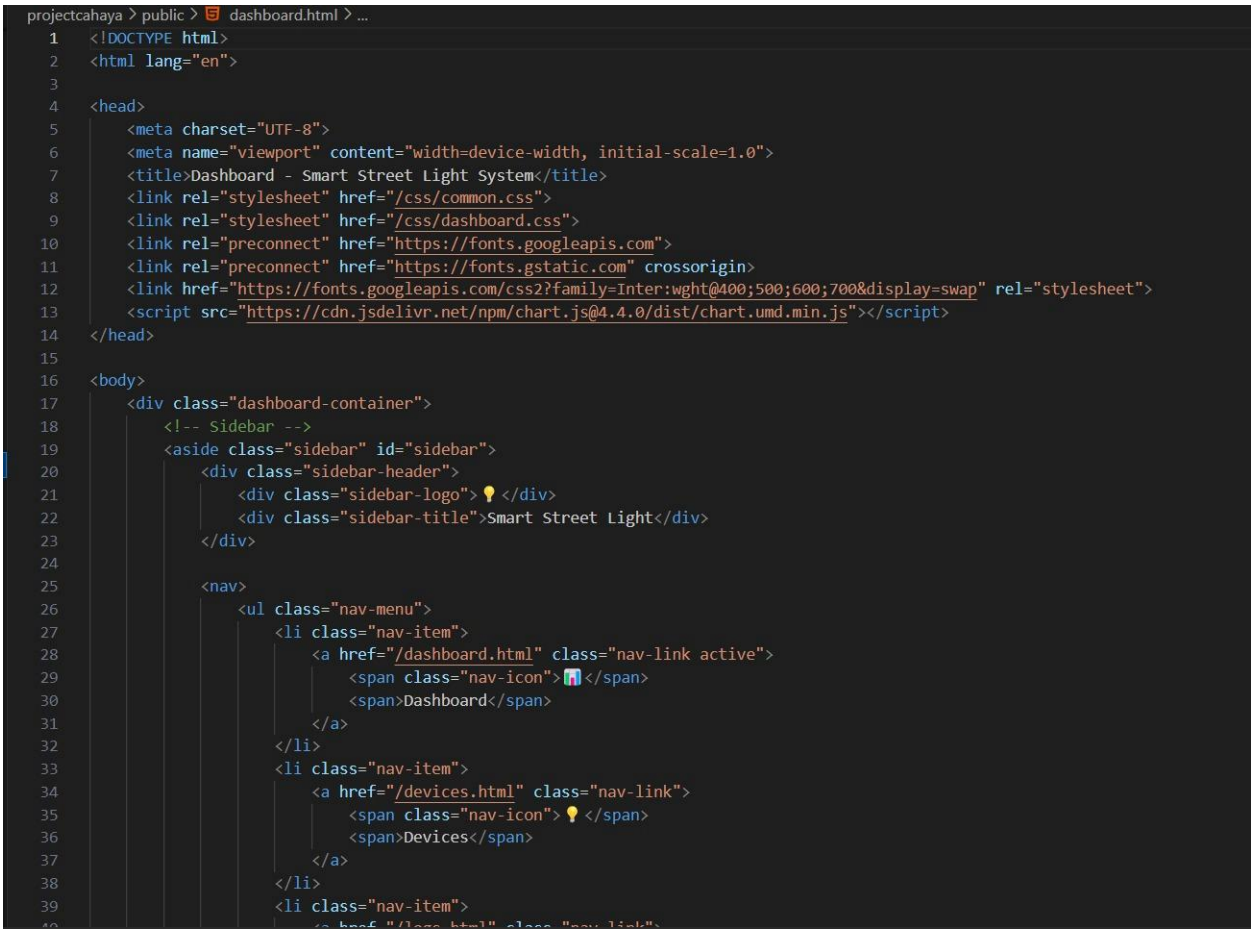
```
const express = require('express');
const router = express.Router();
const db = require('../config/database');
const { authenticateToken } = require('../middleware/auth');

// Get control logs with pagination and filtering
router.get('/', authenticateToken, async (req, res) => {
    try {
        const limit = parseInt(req.query.limit) || 50;
        const offset = parseInt(req.query.offset) || 0;
        const deviceId = req.query.device_id;
        const action = req.query.action;
        const mode = req.query.mode;

        let query = `
            SELECT
                l.id,
                l.device_id,
                d.device_name,
                d.device_id as device_identifier,
                l.action,
                l.mode,
                l.user_id,
                u.username,
                l.details,
                l.timestamp
            FROM control_logs l
            LEFT JOIN devices d ON l.device_id = d.id
            LEFT JOIN users u ON l.user_id = u.id
            WHERE 1=1
        `;

        const params = [];

        if (deviceId) {
            query += ' AND l.device_id = ?';
            params.push(deviceId);
        }

```

Kode routes untuk menerima dan mengelola data sensor

projectcahaya > routes > sensors.js > ...

```
const express = require('express');
const router = express.Router();
const db = require('../config/database');

// Receive sensor data from IoT devices
router.post('/data', async (req, res) => {
    try {
        const { device_id, light_intensity } = req.body;

        if (!device_id || light_intensity === undefined) {
            return res.status(400).json({
                success: false,
                message: 'device_id and light_intensity are required'
            });
        }

        // Get device internal ID
        const [devices] = await db.query(
            'SELECT id, mode FROM devices WHERE device_id = ?',
            [device_id]
        );

        if (devices.length === 0) {
            return res.status(404).json({
                success: false,
                message: 'Device not found. Please register device first.'
            });
        }

        const device = devices[0];

        // Insert sensor data
        await db.query(
            'INSERT INTO sensor_data (device_id, light_intensity) VALUES (?, ?)',
            [device.id, light_intensity]
        );

```

Kode tampilan dashboard



```
projectcahaya > public > dashboard.html > ...
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Dashboard - Smart Street Light System</title>
    <link rel="stylesheet" href="/css/common.css">
    <link rel="stylesheet" href="/css/dashboard.css">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
    <link href="https://fonts.googleapis.com/css2?family=Inter:wght@400;500;600;700&display=swap" rel="stylesheet">
    <script src="https://cdn.jsdelivr.net/npm/chart.js@4.4.0/dist/chart.umd.min.js"></script>
</head>

<body>
    <div class="dashboard-container">
        <!-- Sidebar -->
        <aside class="sidebar" id="sidebar">
            <div class="sidebar-header">
                <div class="sidebar-logo">💡</div>
                <div class="sidebar-title">Smart Street Light</div>
            </div>

            <nav>
                <ul class="nav-menu">
                    <li class="nav-item">
                        <a href="/dashboard.html" class="nav-link active">
                            <span class="nav-icon">📊</span>
                            <span>Dashboard</span>
                        </a>
                    </li>
                    <li class="nav-item">
                        <a href="/devices.html" class="nav-link">
                            <span class="nav-icon">💡</span>
                            <span>Devices</span>
                        </a>
                    </li>
                    <li class="nav-item">
```

Kode utama backend untuk menjalankan server Express.js

projectcahaya > server.js > ...

```
const express = require('express');
const path = require('path');
const cors = require('cors');
require('dotenv').config();

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware
app.use(cors());
app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Serve static files
app.use(express.static(path.join(__dirname, 'public')));

// Import routes
const authRoutes = require('./routes/auth');
const deviceRoutes = require('./routes/devices');
const sensorRoutes = require('./routes/sensors');
const logRoutes = require('./routes/logs');
const settingsRoutes = require('./routes/settings');

// API routes
app.use('/api/auth', authRoutes);
app.use('/api/devices', deviceRoutes);
app.use('/api/sensors', sensorRoutes);
app.use('/api/logs', logRoutes);
app.use('/api/settings', settingsRoutes);

// Root route
app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'login.html'));
});

// Health check endpoint
app.get('/api/health', (req, res) => {
    res.json({
```

Kode Perangkat IoT ESP32 (Arduino IDE)