BAB II

TINJAUAN PUSTAKA

2.1. Penelitian Terkait

Terdapat beberapa penelitian terkait *IoT* berbasis lampu dan perangkat *IoT* yang terhubung ke server.

2.1.1 *An IOT by Information Retrieval approach: Smart Lights controlled using WiFi*

Penelitian dengan judul "*An IOT by Information Retrieval approach: Smart Lights controlled using WiFi*" dibuat oleh Navjot Kaur Walia, Parul Kalra dan Deepti Mehrotra. Penelitian ini menggunakan protokol MQTT untuk mengkomunikasikan antara perangkat IoT lampu dengan perangkat *smartphone* pengguna melalui *Cloud Services* yang diinstal MQTT *broker*. Hasil dari penelitian ini adalah lampu dapat dikontrol dari mana saja selama ada koneksi internet [2]. Penelitian yang dibuat juga akan menggunakan lampu yang dikontrol dengan MQTT dan mengembangkan broker yang digunakan sebagai penghubung antar server dengan alat.

2.1.2 *Wireless Mesh Networks in IoT Networks*

Penelitian dengan judul "*Wireless Mesh Networks in IoT Networks*" ini merupakan penelitian terkait penggunaan jaringan mesh nirkabel.

Penelitian ini membahas mengenai kelebihan dan kekurangan WMN dibandingkan dengan penggunaan topologi lainnya. Makalah ini berisi pengenalan bagaimana mengintegrasikan jaringan mesh ke dalam jaringan perangkat IoT yang sudah ada. Penelitian ini juga menyimpulkan bahwa topologi dinamik seperti WMN ini dapat dicapai walau dengan perangkat IoT yang memiliki *microcontroller* kecil. Selain itu sistem IoT yang menggunakan WMN dapat lebih mudah untuk memperbesar skalanya [4]. Penelitian yang dibuat akan menggunakan jaringan WMN dengan menggunakan ESP8266 pada setiap lampu.

2.1.3 *Smart Dog Feeder Design using Wireless Communication, MQTT, and Android Client*

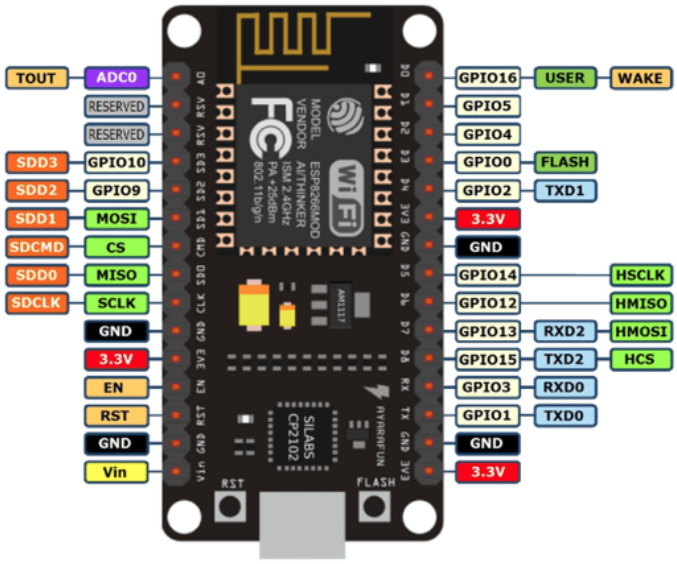
Penelitian dengan judul "*Smart Dog Feeder Design using Wireless Communication, MQTT, and Android Client*" dibuat oleh Vania di Universitas Multimedia Nusantara pada tahun 2016. Penelitian ini menggunakan Arduino dengan ESP8266 sebagai pengendali mikro dari *Smart Dog Feeder* yang dihubungkan dengan *WiFi* ke server Raspberry Pi melalui protokol MQTT. Hasil dari penelitian tersebut server berupa sebuah Raspberry Pi dapat terhubung dengan alat melalui protokol MQTT [7]. Penelitian yang dibuat akan menggunakan server yang sama sebagai broker MQTT, yaitu Raspberry Pi.

2.2. ESP8266

ESP8266 merupakan sebuah modul *wifi* yang ada pada beberapa pengendali mikro, sehingga sesuai untuk diterapkan kedalam *Internet of Things* (IoT). [8].

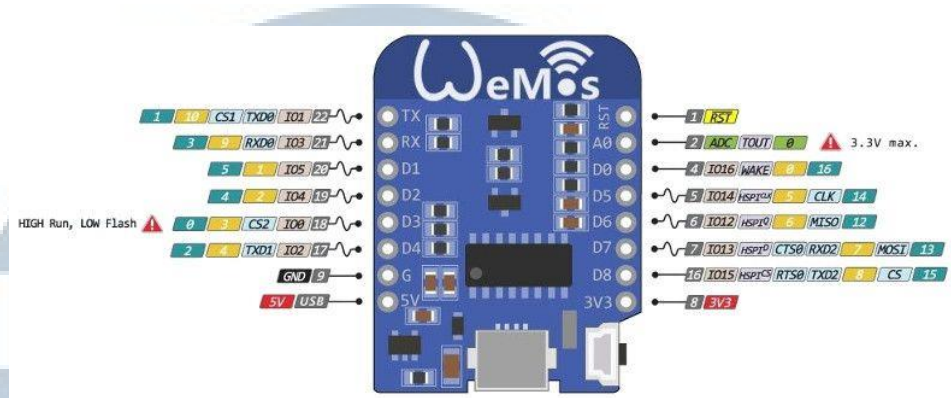
Karena adanya modul *wifi* ESP8266 pada pengendali mikro tersebut, maka dapat digunakan juga sebagai sebuah *access point* untuk menyebar *wifi* dan juga dapat digunakan sebagai server HTTP sederhana yang dapat mengembalikan sebuah HTML dan diakses oleh kontroler atau perangkat lain [8].

NodeMCU adalah salah satu pengendali mikro yang menggunakan ESP8266 dan memiliki 30 pin seperti pada Gambar 2.1. sedangkan WeMos memiliki 16 pin dengan ukuran yang lebih kecil seperti pada Gambar 2.2.



Gambar 2.1 NodeMCU Pin Input Output Map [9]

Sama halnya dengan NodeMCU, WeMos juga menggunakan ESP8266 dan juga didukung oleh sebuah WiFi Module. Penggunaan WeMos dan NodeMCU pun sangat mirip, hanya berbeda pada jumlah pin yang lebih sedikit.

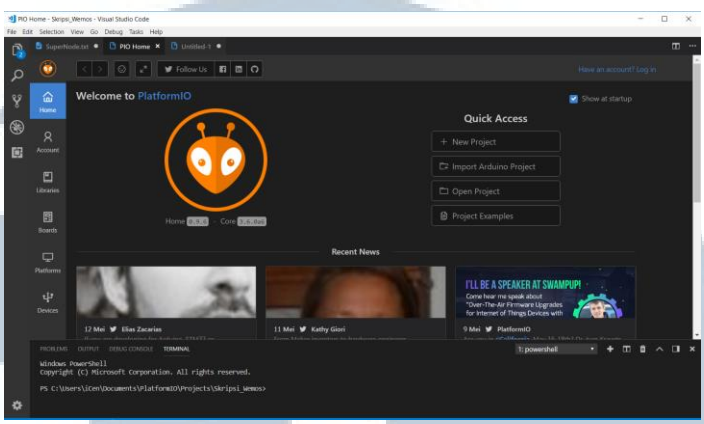


Gambar 2.2 WeMos Pin Input Output Map [10]

2.4. PlatformIO pada Visual Studio Code

Visual Studio Code atau lebih dikenal dengan VSCode merupakan *text editor* yang sangat fleksibel. VSCode dapat diintegrasikan dengan aplikasi-aplikasi pendukung lainnya, dan juga memiliki terminal window sendiri, sehingga sangat cocok untuk melakukan mengembangkan website maupun aplikasi lainnya.

PlatformIO merupakan salah satu *extention* yang ada pada VSCode. Selain VSCode, PlatformIO juga dapat diinstal di Atom. Platformio merupakan sebuah IDE atau *Integrated Development Environment* yang sangat mendukung untuk mengembangkan perangkat *Internet of Things*. PlatformIO mendukung lebih dari 400 *development board* dari lebih dari 20 *platform*. Tampilan utama PlatformIO pada Visual Studio Code dapat dilihat pada Gambar 2.3.

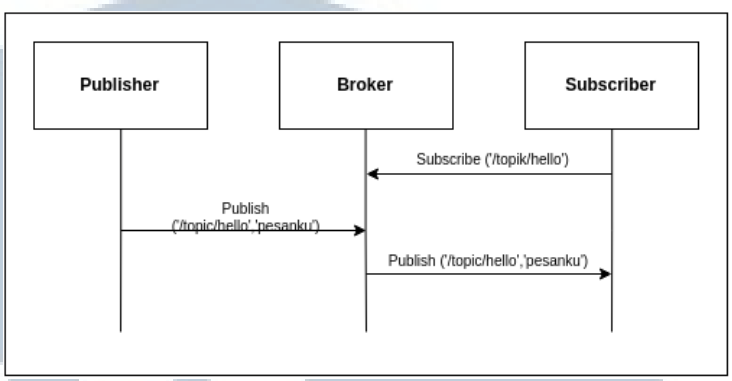


Gambar 2.3 *Screen Shoot* Tampilan Home PlatformIO pada VSCode

2.5. Message Queuing Telemetry Transport (MQTT)

MQTT merupakan kependekan dari *Message Queue Telemetry Transport*. MQTT merupakan sebuah protokol yang menggunakan arsitektur dengan model *topic-based publish-subscribe* [7], [11].

Pada MQTT, paling tidak akan ada tiga pemeran utama, yaitu *publisher*, *subscriber*, dan *broker*. Seperti pada Gambar 2.4, *subscriber* berperan sebagai klien yang melakukan *subscribe* ke sebuah topik tertentu pada sebuah *broker*. Kemudian *publisher* berperan melakukan *publish* ke sebuah *broker* yang berisi nama topik dan juga pesan yang ingin dikirimkan. Ketika *publisher* melakukan *publish* dan *subscriber* melakukan *subscribe* dengan topik yang sama ke sebuah *broker*, maka *subscriber* akan mendapatkan isi pesan dari *publisher* tersebut. Jadi peran *broker* adalah sebagai server yang menerima dan meneruskan *publish* dari *publisher* ke *subscriber* [11].



Gambar 2.4 Cara Kerja Protokol MQTT [12]

MQTT menggunakan protokol TCP layaknya HTTP. Namun *header* paket yang dimiliki oleh MQTT lebih kecil dibandingkan HTTP sehingga dapat lebih menghemat sumber daya. MQTT juga mendukung tiga macam QoS, yaitu QoS versi 0, 1, dan 2. Semakin tinggi tingkat versinya akan semakin ketat pula MQTT dalam aturan penerimaan paketnya[11].

Pada ranah IoT, MQTT menjadi salah satu solusi yang paling populer setelah HTTP, karena selain lebih hemat sumber daya, penggunaan MQTT cenderung lebih mudah. Beberapa contoh broker adalah Mosquitto, RabbitMQ, HiveMQ, dan lain-lain.

2.6. VueJS

VueJS adalah *framework* JavaScript yang digunakan untuk membangun tampilan antarmuka. Vue dari awal dibuat dengan tujuan agar mudah dikembangkan dan mudah beradaptasi secara bertahap. Vue hanya berfokus pada lapisan antarmuka saja, dan mudah untuk diintegrasikan ke pustaka yang sudah ada. Selain itu Vue juga sangat mendukung pembuatan *single page website*.

2.7. Jaringan Mesh

Sebuah jaringan mesh nirkabel (WMN) adalah sebuah jaringan yang terdiri dari banyak *node* yang saling berkomunikasi sehingga membentuk seperti jala. Sifat jaringan yang terdistribusi seperti WMN ini sangatlah ideal untuk digunakan pada sebuah jaringan IoT untuk memanfaatkan cakupan jaringan agar lebih luas dan juga tetap menjaga tampilan perangkat IoT karena modulnya kecil dan tersebar. Selain itu dengan cara ini jaringan menjadi lebih toleransi terhadap suatu kegagalan [4].

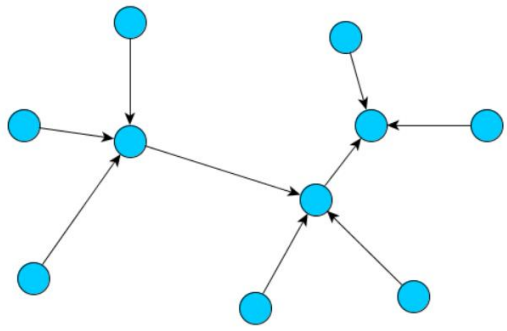
Jaringan mesh nirkabel ini juga memiliki banyak sekali kelebihan apabila diimplementasikan kedalam suatu jaringan IoT. Salah satu kelebihan yang paling menonjol adalah jaringan menjadi sangat fleksibel. Untuk menambahkan sebuah perangkat IoT baru kedalam sebuah jaringan WMN hanya perlu menempatkan perangkat IoT tersebut ke dalam jaringan WMN yang sudah ada, selain itu jangkauan cakupan dari jaringan juga dapat menjadi lebih besar, karena perangkat tersebut tidak perlu terhubung langsung dengan server, tetapi mencari *node* yang terdekat dengannya. Selain dari segi fleksibilitas dan *scalability*, WMN juga menjamin jaringan yang toleran terhadap kegagalan. Apabila salah satu *node* pada jaringan tersebut rusak, maka *node* lain akan secara otomatis kembali terhubung dengan *node* yang lain [4].

2.8. PainlessMesh

PainlessMesh merupakan sebuah *library* yang berfungsi untuk membuat jaringan mesh (jala) sederhana menggunakan *library* ESP8266 secara otomatis

tanpa perlu membuat struktur dari topologi jaringan tersebut. PainlessMesh menggunakan JSON sebagai basis dari pertukaran data yang dilakukan antar node. PainlessMesh menggunakan JSON karena mudah dibaca dan dipahami, selain itu juga mudah diintegrasikan dengan javascript *front-end* maupun web aplikasi lainnya [13].

Setiap node pada mesh berfungsi sebagai *Access Point* (AP) untuk node lain, dan juga sebagai *client* yang terhubung ke AP atau node lain seperti pada Gambar 2.5. Batas maksimal koneksi ke satu AP adalah 4 nodes untuk ESP8266 dan 10 nodes untuk ESP32. Setiap node yang belum terhubung ke suatu AP akan melakukan *scanning* secara berkala terhadap AP atau node yang belum ada pada daftar node yang terkoneksi, hal ini dilakukan agar tidak mengakibatkan adanya *loop* pada jaringan [14].



Gambar 2.5 Tata Letak Jaringan pada PainlessMes

PainlessMesh tidak menggunakan *Internet Protocol* (IP) address untuk bertukar data, tetapi setiap node dibedakan dengan id unik sejumlah 32-bit yang didapatkan dari esp8266 dengan perintah *system_get_chip_id()*. Selanjutnya pesan dapat dikirimkan ke semua node yang terhubung dalam topologi mesh atau dikirimkan secara spesifik ke id node yang diindetifikasi dari chip [13].

PainlessMesh menggunakan *library* ArduinoJson, TaskScheduler, dan juga ESPAsyncTCP atau AsyncTCP. Arduino Json digunakan untuk mengelola Json string [15] dalam pengiriman maupun penerimaan data pada *painlessMesh*, TaskScheduler digunakan untuk melakukan perintah secara terjadwal tanpa harus menggunakan *delay*[16], sedangkan ESPAsyncTCP digunakan agar *fungsi* pada *painlessMesh* dapat berjalan secara *asynchronous* [17]. Tabel 2.1 adalah beberapa fungsi yang ada pada *library* painlessMesh dan digunakan dalam penelitian ini.

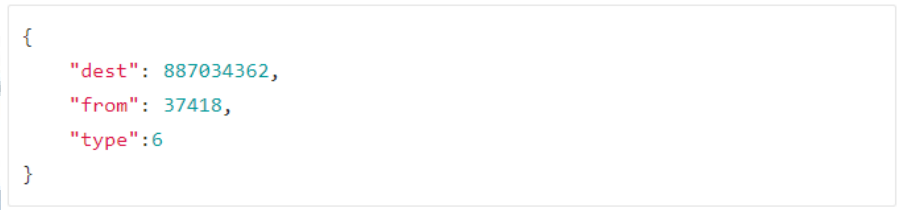
Tabel 2.1 Beberapa Fungsi yang Digunakan pada PainlessMesh [18]

Fungsi	Keterangan
void init (ssid, password, port = 5555)	Diletakkan pada fungsi setup( ). Digunakan untuk inisialisasi jaringan mesh. Default port yang digunakan oleh painlessMesh adalah 5555.
void update( )	Diletakkan pada fungsi loop( ). Untuk melakukan update data yang diperlukan oleh mesh.
void onReceive (&receivedCallback)	Fungsi yang dijalankan ketika node menerima pesan. Fungsi &receiveCallback memiliki parameter *from* yaitu nodeId asal dan *msg* yaitu isi dari pesan tersebut.
void onNewConnection (&newConnectionCallback)	Fungsi ini dijalankan ketika ada node baru yang tersambung. &newConnectionCallback memiliki parameter *nodeId* yang berisi nodeId dari node yang baru saja tersambung.

void onChangedConnection (&changedConnectionCallBack)	Fungsi ini dijalankan ketika ada perubahan pada jaringan mesh, apakah ada node baru maupun ada node yang terlepas dari mesh.
void onNodeTimeAdjusted (&nodeTimeAdjustedCallback)	Fungsi yang dijalankan ketika ada perubahan pada *internal clock* di dalam node.
bool sendBroadcast (&msg, includeSelf = false)	Perintah untuk mengirimkan sebuah pesan *msg* ke seluruh jaringan, parameter kedua merupakan parameter apakah pesan tersebut juga dikirimkan ke dirinya sendiri atau tidak.
bool sendSingle(dest, &msg)	Untuk mengirim pesan ke satu node, parameter *dest* merupakan *nodeId* tujuan.
std::list getNodeList( )	Meminta list semua nodes yang terhubung di dalam jaringan.
int getNodeId( )	Mengambil nodeId pada node saat ini.
int getNodeTime(void)	Mengambil *internal clock* pada node saat ini (dalam *micro second*). Dan akan kembali 0 setiap 71 menit dari node pertama.
void stationManual (ssid, password)	Untuk menghubungkan mesh dengan Access Point diluar jaringan mesh. Node akan membuka koneksi TCP terhubung dengan WiFi.

Pesan yang digunakan oleh mesh untuk bertukar data antar node nya adalah berbasis JSON. Pesan yang dikirimkan akan dibagi menjadi *control messages* dan *user messages*. *Control messages* dikirimkan oleh node untuk bertukar informasi mengenai *routing* dan sinkronisasi waktu antar nodes. Sedangkan *user messages* digunakan untuk mengirim pesan. Skema JSON dasar yang dikirimkan oleh node untuk semua pesan dapat dilihat pada Gambar 2.6. "*dest*" adalah nodeID dari

tujuan, sedangkan "*from*" adalah node sebelumnya yang dilalui oleh pesan, dan "*type*" adalah tipe dari pesan.



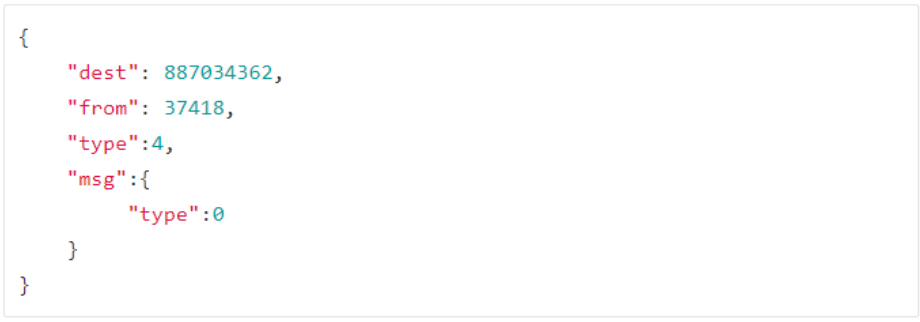
Gambar 2.6 Skema JSON Dasar untuk Pengiriman Pesan

Clock pada setiap node bersifat *internal*, tetapi disinkronisasi sehingga semua node memiliki waktu yang sama agar mereka dapat mengerjakan perintah dengan bersama-sama. Waktu pada sebuah node disinkronisasi dengan node yang berada di sebelahnya. Ketika ada node baru pada jaringan *mesh*, sinkronisasi waktu akan dijalankan. Sinkronisasi waktu juga akan dijalankan secara periodik dengan selang waktu 7-13 menit sekali pada setiap node nya.

Sebelum menginisasi sinkronisasi waktu, node harus mengetahui waktu siapa yang akan digunakan. Node dengan koneksi dan sub-koneksi yang lebih sedikit akan mengikuti node yang memiliki lebih banyak koneksi, apabila jumlah node yang terkoneksi sama, maka waktu pada node yang menjadi AP akan diadopsi [6].

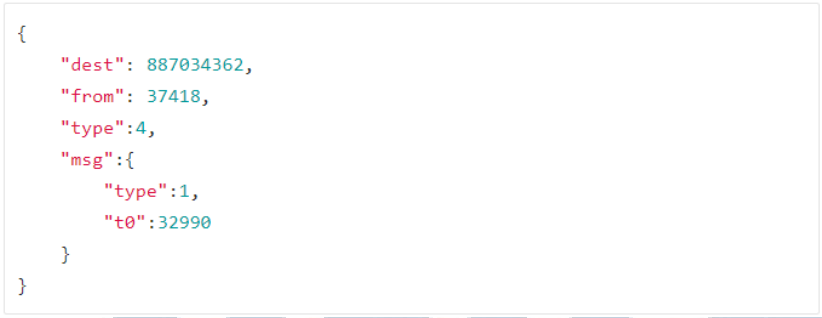
Node yang akan menginisasi sinkronisasi waktu akan melakukan *request* dengan mengirimkan pesan JSON. Apabila node tersebut tidak perlu mengadopsi waktu dari node sebelah, ia akan mengirim request ke node sebelahnya seperti pada contoh Gambar 2.7. "*type*" untuk menandakan bahwa JSON untuk sinkronisasi

waktu adalah 4. Sedangkan “type” dalam “msg” mengartikan bahwa pesan berupa pesan *request* untuk melakukan sinkronisasi waktu .



Gambar 2.7 Pesan JSON Request Time Sync [6]

Apabila node yang melakukan inisiasi sinkronisasi waktu harus mengadopsi waktu, atau node tersebut mendapatkan pesan *request* seperti pada Gambar 2.7, maka node tersebut akan mengirim JSON dengan format seperti pada Gambar 2.8. “t0” adalah *internal clock* milik node ketika pesan dibuat.



Gambar 2.8 Pesan JSON Balasan Request Time Sync (2) [6]

Kemudian node yang waktunya akan diadopsi akan membalas pesan tersebut dengan menambahkan 2 buah *clock*, yaitu “t1” dan “t2”. “t1” adalah *timestamp* internal milik node saat *request* diterima, sedangkan “t2” adalah *timestamp* internal ketika pesan dibuat, seperti tampak pada Gambar 2.9.

```
{
    "dest": 37418,
    "from": 887034362,
    "type":4,
    "msg":{
        "type":2,
        "t0":32990,
        "t1":448585896,
        "t2":448596056,
    }
}
```


Gambar 2. 9 Pesan JSON Balasan JSON Request Time Sync (3) [6]

Setelah menerima pesan balasan, node yang mengadopsi akan menghitung "t3" sebagai *timestamp* saat pesan balasan diterima. Setelah itu node akan menghitung waktu offset dan tripDelay berdasarkan rumus seperti pada Gambar 2.10, dan menghitung *internal clock* nya. *Offset* adalah waktu perbedaan waktu rata-rata antar 2 node, sedangkan *tripDelay* adalah waktu rata-rata yang diperlukan untuk mengirim dan menerima pesan

$$\text{offset} = \frac{t1 - t0}{2} + \frac{t2 - t3}{2}$$

$$\text{tripDelay} = (t3 - t0) - (t2 - t1)$$

Gambar 2.10 Rumus Perhitungan Offset dan tripDelay [6]