

BAB 5 IMPLEMENTASI

5.1 MQTT Broker

Implementasi MQTT broker akan dilakukan untuk membuat server yang berperan sebagai broker protokol MQTT. Implementasi MQTT broker diperlukan agar klien MQTT dapat berkomunikasi satu sama lain dengan melakukan *publish-subscribe*. Pada implementasi ini akan digunakan mosquitto broker sebagai MQTT broker dan mosquitto broker akan diimplementasi dengan mosquitto auth-plugin yaitu *plugin* yang digunakan untuk melakukan mekanisme autentikasi dan otorisasi pada mosquitto broker.

5.1.1 Implementasi Mosquitto Broker

Implementasi mosquitto broker akan dilakukan pada sebuah Virtual Box yang menjalankan OS Linux Ubuntu 16.04, mosquitto broker sendiri merupakan broker pesan *open source* yang menerapkan protokol MQTT versi 3.1 dan 3.1.1. Langkah-langkah implementasi mosquitto broker akan dimulai dengan melakukan instalasi *requirement* yang dibutuhkan oleh mosquitto broker yaitu `libssl-dev` dan `openssl 1.0.0`, yang akan dilakukan dengan menggunakan perintah sebagai berikut:

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ sudo apt-get install libssl-dev
$ wget https://openssl.org/source/openssl-1.0.0c.tar.gz
$ tar -xvzf openssl-1.0.0c.tar.gz
$ cd /path/to/openssl-1.0.0c
$ ./config
$ sudo make
$ sudo make test
$ sudo make install
```

Gambar 5.1 Perintah Install Requirement Untuk Mosquitto

Kemudian setelah berhasil melakukan instalasi *requirement* yang dibutuhkan oleh mosquitto broker, lalu langkah selanjutnya adalah melakukan instalasi mosquitto broker itu sendiri. Untuk melakukan instalasi mosquitto broker pada Ubuntu 16.04 dapat dilakukan melalui terminal dengan menggunakan perintah sebagai berikut:

```
$ wget https://mosquitto.org/files/source/mosquitto-1.3.5.tar.gz
$ tar xvzf mosquitto-1.3.5.tar.gz
$ cd /path/to/mosquitto-1.3.5
$ nano config.mk
edit >> WITH_DNS_SRV=no, WITH_UUID=no, WITH_LIBWEBSOCKETS=yes,
        WITH_TLS=no, WITH_SRV=no
$ sudo make
$ sudo make install
```

Gambar 5.2 Perintah Install Mosquitto

5.1.2 Implementasi OpenSSL pada Mosquitto

Implementasi OpenSSL akan dilakukan untuk menerapkan *tls/ssl* pada mosquitto broker untuk mengamankan *layer transport* yang akan digunakan untuk komunikasi protokol MQTT. Implementasi openssl akan dilakukan dengan membuat sertifikat yang akan digunakan untuk bertukar informasi melalui openssl. Langkah-langkah yang akan dilakukan adalah dengan membuat sebuah *key*, lalu membuat *Certificate Authority (CA)* dengan *key* tersebut, kemudian akan dibuat *key* baru untuk broker, lalu membuat sebuah *certificate sign request* berdasarkan *key* broker, terakhir akan dibuat sebuah CA untuk broker berdasarkan *certificate sign request* dan *key* milik broker. Langkah-langkah tersebut dilakukan dengan menggunakan perintah sebagai berikut:

```
$ openssl genrsa -des3 -out mqtt_ca.key 2048
enter >> new mqtt_ca.key pass phrase
$ openssl req -new -x509 -days 3650 -key mqtt_ca.key -out
mqtt_ca.crt
enter >> mqtt_ca.key pass phrase
$ openssl genrsa -out mqtt_broker.key 2048
$ openssl req -new -out mqtt_broker.csr -key mqtt_broker.key
$ openssl x509 -req -in mqtt_broker.csr -CA mqtt_ca.crt -CAkey
mqtt_ca.key -CAcreateserial -out mqtt_broker.crt -days 3650
enter >> mqtt_ca.key pass phrase
```

Gambar 5.3 Perintah Pembuatan Key & Sertifikat TLS/SSL

5.1.3 Implementasi Mosquitto Auth-Plug

Setelah mosquitto broker dan openssl berhasil diimplementasikan maka langkah selanjutnya adalah melakukan implementasi mosquitto auth-plugin yang akan digunakan untuk melakukan mekanisme otorisasi dan autentikasi pada mosquitto broker. Implementasi mosquitto auth-plugin yang akan digunakan akan menggunakan *backend* JWT. Langkah implementasi akan dilakukan dimulai dari melakukan instalasi beberapa *requirement* yang dibutuhkan dengan perintah sebagai berikut:

```
$ sudo apt-get install git
$ git clone https://github.com/redis/hiredis.git
$ cd /path/to/hiredis
$ sudo make
$ sudo make install
```

Gambar 5.4 Perintah Install Requirement Mosquitto Auth-Plug

Setelah berhasil melakukan instalasi *requirement* yang dibutuhkan, langkah selanjutnya adalah melakukan instalasi mosquitto auth-plugin sendiri. Untuk dapat membangun mosquitto auth-plugin, dibutuhkan mosquitto broker dan *library* yang akan dibutuhkan oleh *backend* yang akan digunakan. *Library backend* dari mosquitto auth-plugin sendiri terdiri dari berbagai macam yaitu MySQL, PostgreSQL, MongoDB, HTTP, JWT dll. Pada folder mosquitto authplug, lakukan *copy* file "config.mk.in" menjadi "config.mk" dan modifikasi konfigurasi dari file tersebut sesuai dengan keinginan, berdasarkan pada lokasi file mosquitto broker, openssl

dan *backend* mana yang akan digunakan. Pada penelitian ini *backend* yang akan digunakan adalah JWT. Langkah-langkah tersebut dapat dilihat dalam perintah sebagai berikut:

```
$ git clone https://github.com/jpmens/mosquitto-auth-plugin.git
$ cd /path/to/mosquitto-auth-plugin
$ cp config.mk.in config.mk
$ nano config.mk
  edit >> BACKEND_JWT = yes
  edit >> MOSQUITTO_SRC = /path/to/mosquitto-1.3.5
  edit >> OPENSLLDIR = /usr/local/openssl
$ sudo make
```

Gambar 5.5 Perintah Install Mosquitto Auth-Plug

Setelah itu dilakukan perintah “*make*” akan didapatkan file bernama “*auth-plugin.so*” yang akan digunakan sebagai referensi pada file “*mosquitto.conf*”. Karena *backend* yang akan digunakan adalah JWT, maka konfigurasi file akan dilakukan sesuai dengan konfigurasi HTTP karena akan digunakan sebuah server yang berjalan lewat HTTP untuk melakukan implementasi JWT. Untuk melakukan referensi tersebut akan dilakukan perubahan pada file “*mosquitto.conf*” yang digambarkan sebagai berikut:

Tabel 5.1 Konfigurasi File “mosquitto.conf”

1	port 8883
2	cafile /path/to/mqtt_ca.crt
3	keyfile /path/to/mqtt_broker.key
4	certfile /path/to/mqtt_broker.crt
5	tls_version tlsv1
6	auth_plugin /path/to/auth-plugin.so
7	auth_opt_backends jwt
8	auth_opt_http_ip localhost
9	auth_opt_http_port 8100
10	auth_opt_http_getuser_uri /auth
11	auth_opt_http_superuser_uri /superuser
12	auth_opt_http_aclcheck_uri /acl

5.2 Auth-Server

Implementasi Auth-Server akan dilakukan untuk membuat server yang berperan sebagai server yang akan memeriksa autentikasi dan otorisasi serta sebagai server yang akan melakukan CRUD (*Create, Read, Update, Delete*) ke database. Implementasi auth-server diperlukan agar mosquitto broker dapat memeriksa identitas dan hak akses klien MQTT melalui auth-server, mosquitto broker akan berkomunikasi dengan auth-server melalui *HTTP-method*. Pada implementasi ini akan digunakan *bottle framework* pada python sebagai auth-server dan MySQL sebagai database dengan *phpmyadmin* sebagai aplikasi manajemen database.

5.2.1 Implementasi MySQL dan PhpMyAdmin

Implementasi MySQL dan PhpMyAdmin akan dilakukan untuk menjadikan sebagai database yang akan menyimpan informasi klien MQTT dan ACL dari protokol MQTT. Informasi ini akan dibutuhkan untuk melakukan mekanisme autentikasi dan otorisasi klien MQTT yang akan dilakukan oleh auth-server. Untuk melakukan implementasi akan dilakukan instalasi LAMP (Linux, Apache, MySQL, PHP) pada Ubuntu 16.04, instalasi PhpMyAdmin, dan membuat database sesuai dengan perancangan di MySQL menggunakan PhpMyAdmin yang nantinya akan digunakan oleh auth-server. Langkah pertama yang akan dilakukan adalah instalasi LAMP dan PhpMyAdmin menggunakan perintah berikut:

```
$ sudo apt-get update
$ sudo apt-get install apache2
$ sudo apt-get install mysql-server
$ mysql_secure_installation
$ sudo apt-get install php libapache2-mod-php php-mcrypt php-
mysql
$ sudo nano /etc/apache2/mods-enabled/dir.conf
edit >> DirectoryIndex index.php index.html index.cgi index.pl
index.xhtml index.htm
$ sudo apt-get install phpmyadmin php-mbstring php-gettext
```

Gambar 5.6 Perintah Install LAMP dan PhpMyAdmin

Setelah LAMP dan PhpMyAdmin berhasil dilakukan instalasi, selanjutnya adalah membuat database dan tabel sesuai dengan skema database yang dibuat pada gambar 4.8 dan melakukan insert data pada setiap tabel sesuai dengan perancangan yang telah dilakukan.

5.2.2 Implementasi Auth-Server dengan Bottle Framework

Setelah MySQL berhasil diimplementasi dan database telah berhasil dibuat bersama dengan informasi yang ada didalamnya, langkah berikutnya adalah melakukan implementasi Auth-Server. Implementasi auth-server akan dilakukan dengan membuat sebuah *micro web-framework* pada python menggunakan *framework* Bottle yang kemudian akan berkomunikasi lewat HTTP-method dengan mosquitto broker untuk melakukan autentikasi dan otorisasi. Langkah-langkah yang akan dilakukan akan dimulai dari menginstalasi python dan *framework* bottle, yang dilakukan dengan menjalankan perintah sebagai berikut:

```
$ sudo apt-get update
$ sudo apt-get install python
$ sudo apt-get install python-pip
$ sudo pip install bottle
```

Gambar 5.7 Perintah Install Python dan Bottle Framework

Setelah instalasi python dan bottle *framework* berhasil dilakukan, maka akan dibuat sebuah *source code* yang akan dijalankan untuk berperan sebagai auth-server untuk menerima *request* dan mengembalikan *response* menggunakan HTTP-method, auth-server juga akan menjadi pihak yang terhubung dan melakukan transaksi informasi dengan database MySQL, selain itu server juga

akan melakukan *decode* dan *encode* token JWT sebagai identitas klien MQTT menggunakan *library* *jwt* pada *python*. Auth-server akan menangani mekanisme autentikasi, mekanisme otorisasi dan mekanisme cek superuser. Auth-server juga akan menangani mekanisme manipulasi data pada database pada tabel *mqtt_users*, *mqtt_roles* dan *mqtt_permissions*, yang akan dapat dimanipulasi oleh pengguna dengan peran sebagai admin. Pengguna akan melakukan request melalui *HTTP-method* yang akan diterima oleh auth-server untuk kemudian mengakses database untuk melakukan manipulasi informasi tersebut.

Tabel 5.2 Source Code Mekanisme Autentikasi pada Auth-Server

```

1  @app.route('/auth', method='POST')
2  def auth():
3      response.content_type = 'text/plain'
4      response.status = 403
5
6      encoded = request.get_header('Authorization')
7      if encoded is None:
8          return
9
10     data = decodeJWT(encoded)
11     username = data['username']
12     password = data['pw']
13
14     data_sql = userQuery(username)
15     if data_sql is None:
16         return
17
18     checkhash = pbkdf2.check_hash(password, data_sql['pw'])
19     if username == data['username'] and checkhash:
20         print "username and password match!"
21         response.status = 200
22     return
23
24 def userQuery(username):
25     cur = db.cursor()
26     sql = "SELECT * FROM mqtt_users WHERE username = '%s' "
27     % (username)
28     try:
29         cur.execute(sql)
30         result = cur.fetchone()
31         data = {'id': result[0], 'username': result[1], 'pw':
32 result[2], 'role': result[3]}
33         return data
34     except:
35         return None

```

Pada tabel 5.2 diatas, dijelaskan mengenai *source code* yang akan menangani mekanisme autentikasi auth-server. Dimulai dari baris 1-18 merupakan fungsi yang akan diakses ketika broker mengirimkan request mekanisme autentikasi melalui *HTTP-method post* pada url *'/auth'* yang akan menjalankan fungsi *auth()*. Pada baris 4 dinyatakan response status 403 yang menandakan bahwa belum terjadi mekanisme autentikasi atau autentikasi gagal. Pada baris 5-7 akan dilakukan pengambilan *header 'Authorization'* yang berisi token JWT request dan diperiksa apakah *header* ditemukan atau tidak, jika tidak maka *request* akan

dikembalikan tanpa melakukan autentikasi dengan *response status* 403. Jika *header* berhasil didapatkan maka akan dilakukan *decode* token JWT yang akan berisi *username* dan *password* klien MQTT, yang dinyatakan pada baris 8-10. Kemudian pada baris 11-13 akan diperiksa apakah *username* tersebut terdapat dalam database atau tidak, jika tidak maka *request* akan dikembalikan dengan *response status* 403. Jika *username* tersebut ada maka akan diperiksa apakah *username* dan *password* cocok atau tidak, yang dinyatakan pada baris 14-17. Jika cocok maka *response status* akan dikembalikan bernilai 200 namun jika tidak akan dikembalikan 403. Kemudian pada baris 19-28 merupakan fungsi yang diakses untuk memeriksa *username* di database. Pada baris 21 merupakan *query* yang akan diakses ke database untuk mengambil informasi *username* dan *password* jika *username* cocok dan mengembalikan nilai tersebut. Namun jika tidak cocok maka nilai yang akan dikembalikan adalah *null*.

Tabel 5.3 Source Code Mekanisme Otorisasi pada Auth-Server

```

1  @app.route('/acl', method='POST')
2  def acl():
3      response.content_type = 'text/plain'
4      response.status = 403
5
6      encoded = request.get_header('Authorization')
7      if encoded is None:
8          return
9
10     data = decodeJWT(encoded)
11     username = data['username']
12     topic    = request.forms.get('topic')
13     acc      = request.forms.get('acc')
14     result = aclQuery(username, int(acc))
15     if result is None:
16         return
17
18     for row in result:
19         if(topic == row[0]):
20             response.status = 200
21             break
22     return
23
24 def aclQuery(username, acc):
25     cur = db.cursor()
26     sql = "SELECT p.topic FROM mqtt_permissions p JOIN
27 mqtt_roles r ON r.title = p.role RIGHT JOIN mqtt_users u ON
28 u.role = r.title WHERE username = '%s' AND rw >= '%d'" %
29 (username, acc)
30     try:
31         cur.execute(sql)
32         result = cur.fetchall()
33         return result
34     except:
35         return None

```

Pada Tabel 5.3 diatas, dijelaskan mengenai *source code* yang akan menangani mekanisme otorisasi auth-server. Dimulai dari baris 1-19 merupakan fungsi yang akan diakses ketika broker mengirimkan request mekanisme otorisasi melalui

HTTP-*method post* pada url *'/acl'* yang akan menjalankan fungsi *acl()*. Pada baris 4 dinyatakan *response status 403* yang menandakan bahwa belum terjadi mekanisme autentikasi atau autentikasi gagal. Pada baris 5-7 akan dilakukan pengambilan *header 'Authorization'* yang berisi token JWT request dan diperiksa apakah *header* ditemukan atau tidak, jika tidak maka *request* akan dikembalikan tanpa melakukan autentikasi dengan *response status 403*. Jika *header* berhasil didapatkan maka akan dilakukan *decode* token JWT yang akan berisi *username* dan *password* klien MQTT, yang dinyatakan pada baris 8-9. Kemudian pada baris 10-11 akan diambil bagian *body* dari *request* untuk mendapatkan topik dan acc (*publish/subscribe*), lalu di baris 12-14 akan dilakukan pemeriksaan ACL di database terkait *username* dan acc tersebut. Jika pemeriksaan ACL gagal atau hasilnya tidak ditemukan maka akan dikembalikan *response status 403*. Namun, jika berhasil maka akan dilakukan pemeriksaan tiap hasil dari pemeriksaan ACL apakah topik yang diminta terdapat pada hasil pemeriksaan ACL tersebut pada baris 15-19, jika hasilnya adalah cocok maka akan dikembalikan *response status 200*. Tetapi bila tidak ditemukan kecocokan maka response status yang akan dikembalikan adalah 403. Kemudian pada baris 20-28 merupakan fungsi yang diakses untuk pemeriksaan ACL di database. Pada baris 21 merupakan *query* yang akan diakses ke database untuk mengambil informasi tiap topik yang cocok dengan *username*, peran dan acc lalu akan mengembalikan tiap nilai topik tersebut. Namun jika tidak cocok maka nilai yang akan dikembalikan adalah *null*.

Tabel 5.4 Source Code Mekanisme Cek Superuser pada Auth-Server

```

1  @app.route('/superuser', method='POST')
2  def superuser():
3      response.content_type = 'text/plain'
4      response.status = 403
5
6      encoded = request.get_header('Authorization')
7      if encoded is None:
8          return
9
10     data = decodeJWT(encoded)
11     username = data['username']
12     su = superQuery(username)
13     if su is None:
14         return
15     if su>0:
16         response.status = 200
17     return
18
19 def superQuery(username):
20     cur = db.cursor()
21     sql = "SELECT COUNT(*) FROM mqtt_users u JOIN mqtt_roles
22     r ON u.role = r.title WHERE username='%s' AND r.super=1" %
23     (username)
24     try:
25         cur.execute(sql)
26         result = cur.fetchone()
27         return result[0]
28     except:
29         return None

```

Pada Tabel 5.4 diatas, dijelaskan mengenai *source code* yang akan menangani mekanisme cek superuser auth-server. Dimulai dari baris 1-15 merupakan fungsi yang akan diakses ketika broker mengirimkan request mekanisme cek superuser melalui *HTTP-method post* pada url *'/superuser'* yang akan menjalankan fungsi *superuser()*. Pada baris 4 dinyatakan *response status 403* yang menandakan bahwa belum terjadi mekanisme autentikasi atau autentikasi gagal. Pada baris 5-7 akan dilakukan pengambilan *header 'Authorization'* yang berisi token JWT request dan diperiksa apakah *header* ditemukan atau tidak, jika tidak maka *request* akan dikembalikan tanpa melakukan autentikasi dengan *response status 403*. Jika *header* berhasil didapatkan maka akan dilakukan *decode* token JWT yang akan berisi *username* dan *password* klien MQTT, yang dinyatakan pada baris 8-9. Kemudian pada baris 10-12 akan dilakukan pemeriksaan superuser di database terkait *username* dan *password* tersebut. Jika pemeriksaan superuser hasilnya tidak ditemukan maka akan dikembalikan *response status 403*. Namun, jika berhasil maka akan dilakukan pemeriksaan hasil dari cek superuser di database dengan *password* apakah memiliki kecocokan pada baris 13-15, jika hasilnya adalah cocok maka akan dikembalikan *response status 200*. Tetapi bila tidak ditemukan kecocokan maka *response status* yang akan dikembalikan adalah 403. Kemudian pada baris 16-24 merupakan fungsi yang diakses untuk pemeriksaan superuser di database. Pada baris 17 merupakan *query* yang akan diakses ke database untuk mengambil informasi jumlah kecocokan *username* dan apakah perannya merupakan admin lalu akan mengembalikan jumlah nilai tersebut. Namun jika tidak cocok maka nilai yang akan dikembalikan adalah *null*.

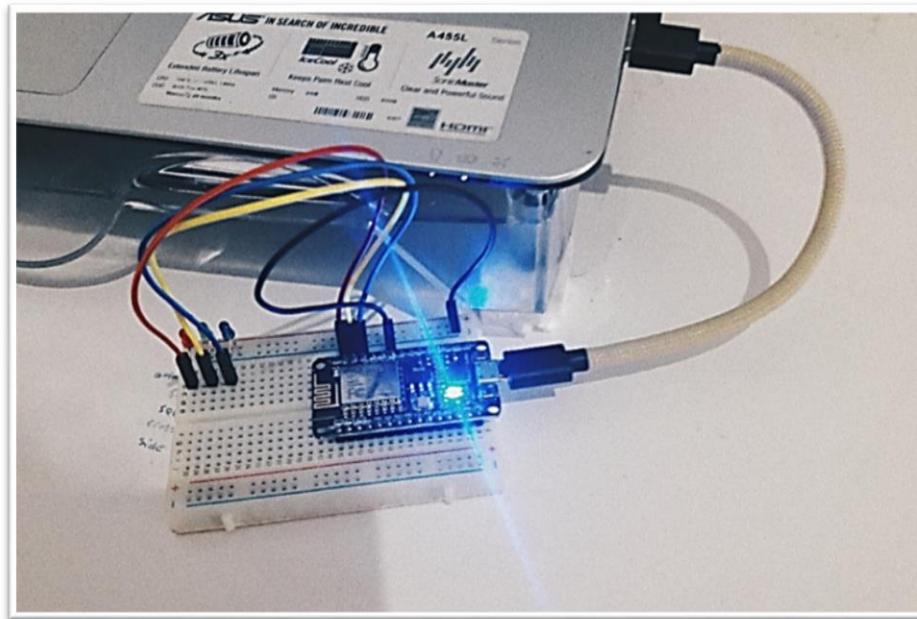
5.3 Perangkat NodeMCU

Implementasi perangkat nodeMCU akan dilakukan untuk menjadikan nodeMCU sebagai klien MQTT serta sebagai mikrokontroler untuk aktuator LED dan sensor suhu dan kelembaban. Implementasi pada perangkat nodeMCU akan dilakukan pada implementasi perangkat keras dan implementasi perangkat lunak, dimana implementasi akan dilakukan berdasarkan perancangan yang dilakukan sebelumnya. Implementasi perangkat keras dilakukan untuk menjadikan perangkat nodeMCU sebagai mikrokontroler aktuator LED dan sensor suhu dan kelembaban, Implementasi perangkat keras ini dilakukan untuk menyiapkan setiap komponen yang akan digunakan dalam melakukan implementasi perangkat lunak yang akan dilakukan setelah semua komponen tersebut berhasil terhubung dengan nodeMCU. Sementara implementasi perangkat lunak akan dilakukan dengan melakukan pemrograman menggunakan arduino IDE, yang dilakukan untuk menerapkan protokol MQTT pada nodeMCU agar dapat berkomunikasi dengan pengguna, dimana pengguna akan dapat mengakses sensor dan aktuator yang telah terhubung oleh nodeMCU.

5.3.1 Implementasi NodeMCU + LED

Implementasi yang dilakukan pada gambar 5.7 merupakan hasil implementasi perangkat keras dari nodeMCU yang dihubungkan dengan tiga LED yang berbeda warna yaitu merah, hijau dan biru. Hasil implementasi ini dilakukan dengan

mengikuti skema yang digambarkan oleh gambar 4.10 pada bab perancangan. Pin GPIO nodeMCU akan dihubungkan dengan sebuah breadboard. Ketiga LED disusun sedemikian rupa sehingga pin VCC LED merah akan terhubung dengan pin GPIO D1 nodeMCU, pin VCC LED hijau akan terhubung dengan pin GPIO D2 nodeMCU dan pin VCC LED biru akan terhubung dengan pin GPIO D3 nodeMCU. Lalu masing-masing pin GND tiap LED akan terhubung dengan GND nodeMCU, untuk menghubungkan tiap pin ini digunakan kabel jumper pada breadboard. Ketiga LED ini nantinya akan dijadikan aktuator untuk dinyalakan atau dimatikan oleh mikrokontroler nodeMCU melalui protokol MQTT pada implementasi perangkat lunak.



Gambar 5.8 Implementasi Perangkat Keras nodeMCU + LED

Setelah implementasi perangkat keras dari nodeMCU dan LED, selanjutnya akan dilakukan implementasi pada perangkat lunak yang dijadikan sebagai program dari nodeMCU sebagai mikrokontroler. Implementasi perangkat lunak milik nodeMCU akan menjadikan nodeMCU ini klien MQTT yang berperan sebagai *subscriber* dan *publisher*, yang akan *subscribe* ke topik untuk menyalakan/mematikan salah satu LED dan mengirimkan status LED yang berubah dengan melakukan *publish* setelahnya. Implementasi perangkat lunak dilakukan dengan menggunakan arduino IDE yang telah terinstal *esp8266 board*. Untuk melakukan implementasi protokol MQTT di nodeMCU akan digunakan *library* klien MQTT PubSubClient dari Nick O'Leary dan ESP8266WiFi yang akan dijadikan klien WiFi. Potongan beberapa *source code* akan ditampilkan pada beberapa tabel sebagai berikut.

Tabel 5.5 Source Code Inisialisasi Klien MQTT pada NodeMCU+LED

1	<code>#define LED_RED D1</code>
2	<code>#define LED_GREEN D2</code>
3	<code>#define LED_BLUE D3</code>
4	<code>#define WIFI_SSID "NAMA WIFI"</code>
5	<code>#define WIFI_PASS "PASSWORD WIFI"</code>

```

6  #define MQTT_SERVER "IP_BROKER"
7  #define MQTT_USERNAME "TOKEN_JWT"
8  #define MQTT_PASSWORD " "

9  WiFiClientSecure espClient;
10 PubSubClient client(MQTT_SERVER, 8883, callback, espClient);
11 void setup() {
12     Serial.begin(115200);
13     pinMode(LED_RED, OUTPUT);
14     pinMode(LED_GREEN, OUTPUT);
15     pinMode(LED_BLUE, OUTPUT);
16     pinMode(ledPin, OUTPUT);

17     setup_wifi();
18     delay(500);
19     setup_certificate();
20 }

21 void loop() {
22     if (!client.connected()) {
23         reconnect();
24     }
25     delay(500);
26     client.loop();
27 }

```

Pada Tabel 5.5 diatas, dijelaskan mengenai potongan *source code* inialisasi program dan klien MQTT. Dimulai dari baris 1-8 adalah nilai tiap pin LED, wifi ssid, *password* wifi, ip broker, *username* dalam bentuk token JWT dan *password*. Pada baris 9-10 merupakan inialisasi klien yang akan melakukan koneksi wifi dan klien yang akan melakukan koneksi ke broker melalui klien wifi tersebut. Pada baris 11-20 merupakan setup yang akan dilakukan tiap saat dijalankan dengan menyatakan tiap pin *output* led *setup* koneksi wifi dan sertifikat untuk tls. Kemudian di baris 21-27 merupakan fungsi *loop* yang akan melakukan koneksi ke broker MQTT jika belum terkoneksi dan melakukan *loop*.

Tabel 5.6 Source Code Koneksi & Subscribe pada NodeMCU+LED

```

1  void reconnect() {
2      while (!client.connected()) {
3          if (client.connect("nodeMCU_LED", MQTT_USERNAME,
4 MQTT_PASSWORD)) {
5              boolean sub;
6              do{
7                  delay(10);
8                  sub = client.subscribe("nodeMCU/LED/+toggle");
9              }while(!sub);
10             updateStatus();
11             publishStatus("red");
12             publishStatus("green");
13             publishStatus("blue");
14         } else {
15             delay(5000);
16         }
17     }
18 }

```

Pada Tabel 5.6 diatas, dijelaskan mengenai potongan *source code* untuk melakukan koneksi dengan broker dan publish status LED untuk pertama kali. Dimulai dari baris 2 yang akan melakukan perulangan selama klien belum terhubung dengan broker, lalu di baris 3 akan melakukan percobaan koneksi, jika berhasil maka akan melakukan perulangan *subscribe* pada topik "nodeMCU/LED/+toggle" hingga berhasil pada baris 4-8. Setelah berhasil subscribe maka akan *publish* status tiap LED untuk pertama kali yang dinyatakan pada baris 9-12. Pada baris 13-15, merupakan seleksi ketika di baris 3 gagal melakukan koneksi yang kemudian akan melakukan *delay* selama 5 detik sebelum mencoba melakukan koneksi lagi.

Tabel 5.7 Source Code Publish pada NodeMCU+LED

1	void publishStatus(String color){
2	char buffer[5];
3	if(color=="red"){
4	dtostrf(redStatus,0, 0, buffer);
5	client.publish("nodeMCU/LED/red/status", buffer);
6	}else if(color=="green"){
7	dtostrf(greenStatus,0, 0, buffer);
8	client.publish("nodeMCU/LED/green/status", buffer);
9	}else if(color=="blue"){
10	dtostrf(blueStatus,0, 0, buffer);
11	client.publish("nodeMCU/LED/blue/status", buffer);
12	}
13	}

Pada Tabel 5.7 diatas, dijelaskan mengenai potongan *source code* untuk melakukan *publish* sesuai dengan perubahan status yang terjadi pada salah satu LED ke broker. Dimulai dari baris 1, yang dinyatakan sebuah parameter *color* untuk seleksi LED mana yang terjadi perubahan status. Lalu di baris 2 disiapkan sebuah *buffer* yang akan diisi dengan nilai status LED tersebut. Kemudian pada baris 3-12 akan dilakukan seleksi dari nilai parameter *color* apakah *red*, *green* atau *blue*, untuk kemudian dibaca status LED tersebut dan dimasukkan nilainya ke *buffer* untuk di-*publish* sesuai dengan topik.

Tabel 5.8 Source Code Penanganan Pesan Masuk pada NodeMCU+LED

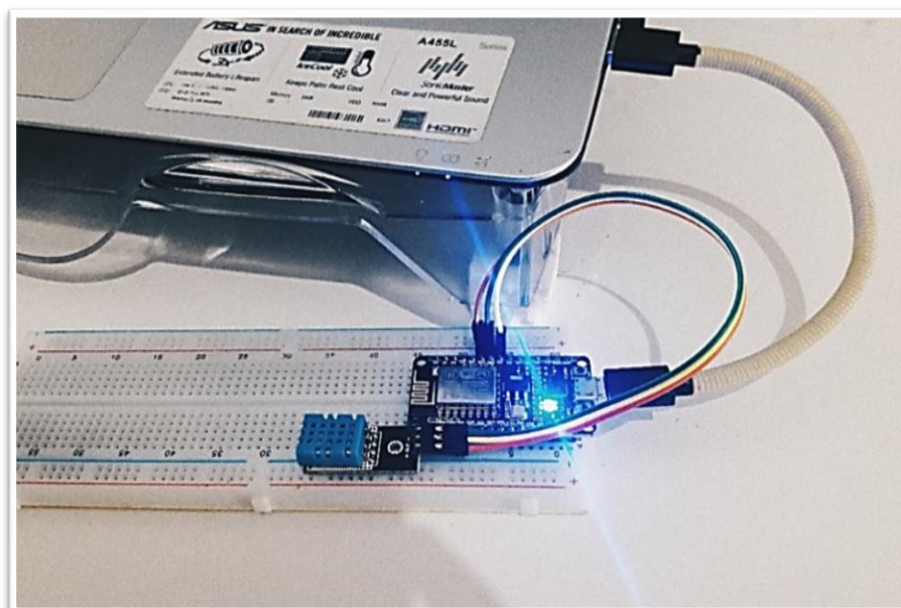
1	void callback(char* topic, byte* payload, unsigned int
2	length) {
	boolean state;
3	for (int i=0;i<length;i++) {
4	char receivedChar = (char)payload[i];
5	if (receivedChar == '1'){
6	state = true;
7	}
8	if (receivedChar == '0'){
9	state = false;
10	}
11	}
12	String color = splitString((String)topic, '/', 2);
13	if(color=="red"){
14	digitalWrite(LED RED, state);

```
15 }else if(color=="green"){
16     digitalWrite(LED_GREEN, state);
17 }else if(color=="blue"){
18     digitalWrite(LED_BLUE, state);
19 }
20 updateStatus();
21 publishStatus(color);
22 }
```

Pada Tabel 5.8 diatas, dijelaskan mengenai potongan *source code* untuk melakukan penanganan ketika ada pesan yang masuk melalui topik yang sudah di-*subscribe*. Dimulai dari baris 3-11 yang akan memeriksa nilai *payload* apakah 1 atau 0 untuk menentukan apakah akan menyalakan atau mematikan LED. Kemudian di baris 12-19 akan dilakukan seleksi terkait topik yang masuk apakah LED yang akan dinyalakan/dimatikan adalah yang berwarna merah, hijau atau biru. Setelah berhasil diubah keadaan LED tersebut maka akan di-*publish* perubahan status LED tersebut pada baris 20-21.

5.3.2 Implementasi NodeMCU + DHT11

Implementasi yang dilakukan pada gambar 5.8 merupakan hasil implementasi perangkat keras dari nodeMCU yang dihubungkan dengan DHT11. Hasil implementasi ini dilakukan dengan mengikuti skema yang digambarkan oleh gambar 4.12 pada bab perancangan. Pin GPIO nodeMCU akan dihubungkan dengan sebuah breadboard. Pin VCC DHT11 akan dihubungkan dengan pin GPIO 3V3 nodeMCU, pin DATA DHT11 akan dihubungkan dengan pin GPIO D5 nodeMCU dan pin GND DHT11 akan dihubungkan dengan pin GPIO GND nodeMCU, untuk menghubungkan tiap pin ini digunakan kabel jumper pada breadboard. DHT11 ini nantinya akan bertindak sebagai sensor dan membaca informasi suhu dan kelembaban dan dikirimkan oleh mikrokontroler nodeMCU melalui protokol MQTT pada implementasi perangkat lunak.



Gambar 5.9 Implementasi Perangkat Keras nodeMCU + DHT11

Setelah implementasi perangkat keras dari nodeMCU dan DHT11, selanjutnya akan dilakukan implementasi pada perangkat lunak yang dijadikan sebagai program dari nodeMCU sebagai mikrokontroler. Implementasi perangkat lunak milik nodeMCU akan menjadikan nodeMCU ini klien MQTT yang berperan sebagai *publisher*, yang akan mengirimkan status data suhu dan kelembaban dengan melakukan *publish* keduanya pada topik yang berbeda. Implementasi perangkat lunak dilakukan dengan menggunakan arduino IDE yang telah terinstal esp8266 *board*. Untuk melakukan implementasi protokol MQTT di nodeMCU akan digunakan *library* klien MQTT PubSubClient dari Nick O’Leary, DHT sensor library oleh Adafruit dan ESP8266WiFi yang akan dijadikan klien WiFi. Potongan beberapa *source code* akan ditampilkan pada tabel berikut.

Tabel 5.9 Source Code Inisialisasi Klien MQTT pada NodeMCU+DHT11

1	#define DHTPIN D5
2	#define DHTTYPE DHT11
3	#define WIFI_SSID "NAMA_WIFI"
4	#define WIFI_PASS "PASSWORD_WIFI"
5	#define MQTT_SERVER "IP_BROKER"
6	#define MQTT_USERNAME "TOKEN_JWT"
7	#define MQTT_PASSWORD " "
8	int defPin = LED_BUILTIN;
9	unsigned long readTime;
10	DHT dht(DHTPIN, DHTTYPE);
11	WiFiClientSecure espClient;
12	PubSubClient client(MQTT_SERVER, 8883, callback, espClient);
13	void setup() {
14	Serial.begin(115200);
15	pinMode(defPin, OUTPUT);
16	setup_wifi();
17	delay(500);
18	setup_certificate();
19	dht.begin();
20	readTime = 0;
21	}
22	void loop() {
23	if (!client.connected()) {
24	reconnect();
25	}
26	delay(500);
27	client.loop();
28	if(millis() > readTime+60000){
29	publishStatus();
30	}
31	}

Pada Tabel 5.9 diatas, dijelaskan mengenai potongan *source code* inisialisasi program dan klien MQTT. Dimulai dari baris 1-7 adalah nilai pin yang digunakan untuk DHT11, tipe DHT yang digunakan, wifi ssid, *password* wifi, ip broker, *username* dalam bentuk token JWT dan *password*. Di baris 8-10 diinisialisasi pin default, dan variabel readTime untuk membaca waktu. Pada baris 11-12 merupakan inisialisasi klien yang akan melakukan koneksi wifi dan klien yang akan

melakukan koneksi ke broker melalui klien wifi tersebut. Pada baris 13-21 merupakan setup yang akan dilakukan tiap saat dijalankan dengan menyatakan pin *output* led default, menjalankan DHT11, *setup* koneksi wifi dan sertifikat untuk tls. Kemudian di baris 22-31 merupakan fungsi *loop* yang akan melakukan koneksi ke broker MQTT jika belum terkoneksi dan melakukan *loop* untuk publish suhu dan kelembaban setiap 5 detik.

Tabel 5.10 Source Code Koneksi pada NodeMCU+DHT11

1	void reconnect() {
2	while (!client.connected()) {
3	if (client.connect("nodeMCU_DHT", MQTT_USERNAME,
	MQTT_PASSWORD)) {
4	
5	} else {
6	delay(5000);
7	}
8	}
9	}

Pada Tabel 5.10 diatas, dijelaskan mengenai potongan *source code* untuk melakukan koneksi dengan broker untuk pertama kali. Dimulai dari baris 2 yang akan melakukan perulangan selama klien belum terhubung dengan broker, lalu di baris 3-7 akan melakukan percobaan koneksi, jika gagal melakukan koneksi yang kemudian akan melakukan *delay* selama 5 detik sebelum mencoba melakukan koneksi lagi.

Tabel 5.11 Source Code Publish pada NodeMCU+DHT11

1	void publishStatus(){
2	float t, h;
3	readTime = millis();
4	do{
5	t = dht.readTemperature();
6	h = dht.readHumidity();
7	}while (isnan(h) isnan(t));
8	char bufferTemp[10];
9	char bufferHum[10];
10	dtostrf(t,0, 0, bufferTemp);
11	dtostrf(h,0, 0, bufferHum);
12	client.publish("nodeMCU/DHT/temperature/status",
	bufferTemp);
13	delay(50);
14	client.publish("nodeMCU/DHT/humidity/status", bufferHum);
15	delay(50);
16	}

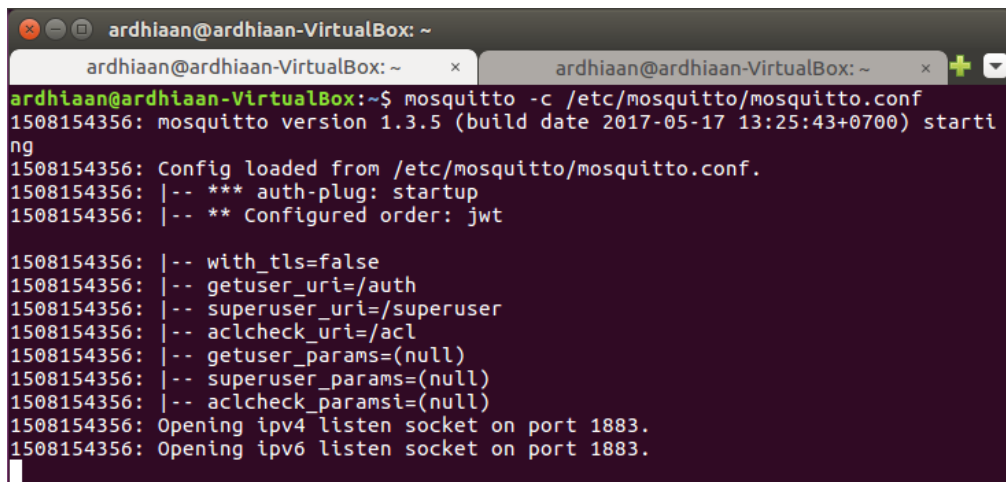
Pada Tabel 5.11 diatas, dijelaskan mengenai potongan *source code* untuk melakukan pembacaan data suhu dan kelembaban oleh sensor DHT11 dan melakukan publish data tersebut. Dimulai dari baris 2 yang menyatakan variabel untuk suhu dan kelembaban, lalu di baris 3 akan dibaca waktu saat ini yang merupakan waktu ketika dilakukan publish terakhir kali. Pada baris 4-7 akan dilakukan perulangan untuk membaca data suhu dan kelembaban oleh sensor hingga kedua data berhasil didapatkan. Kemudian data tersebut akan

dimasukkan kedalam sebuah buffer pada baris 8-11, lalu data suhu akan dipublish pada topik “nodeMCU/DHT/temperature/status” dan data kelembaban akan dipublish pada topik “nodeMCU/DHT/humidity/status” di baris 12-15.

5.4 Setup Sistem

Setup sistem merupakan implementasi untuk mempersiapkan arsitektur sistem secara keseluruhan dengan menjalankan tiap komponen yang telah diimplementasikan. Setup sistem dimulai dari menjalankan mosquitto broker, kemudian menjalankan auth-server, lalu melakukan persiapan untuk setup kedua perangkat nodeMCU dengan meng-*upload source code* ke perangkat tersebut. Berikut merupakan langkah yang dilakukan:

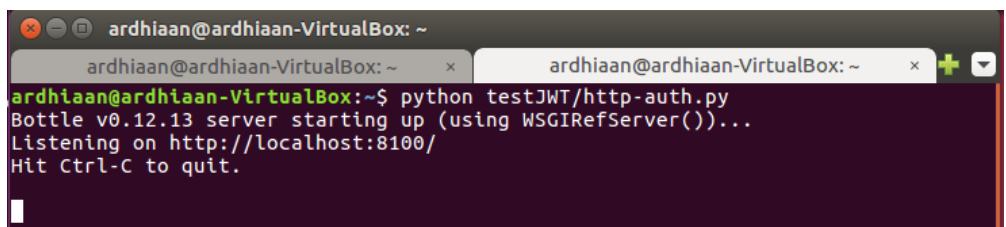
1. Jalankan MQTT broker, dilakukan dengan menjalankan mosquitto broker menggunakan perintah “`mosquitto -c path/to/mosquitto.conf`” pada terminal, untuk menjalankan mosquitto broker dengan *file* konfigurasi yang telah dibuat sesuai kebutuhan sistem.



```
ardhiaan@ardhiaan-VirtualBox: ~  
ardhiaan@ardhiaan-VirtualBox:~$ mosquitto -c /etc/mosquitto/mosquitto.conf  
1508154356: mosquitto version 1.3.5 (build date 2017-05-17 13:25:43+0700) starting  
1508154356: Config loaded from /etc/mosquitto/mosquitto.conf.  
1508154356: |-- *** auth-plugin: startup  
1508154356: |-- ** Configured order: jwt  
  
1508154356: |-- with_tls=false  
1508154356: |-- getuser_uri=/auth  
1508154356: |-- superuser_uri=/superuser  
1508154356: |-- aclcheck_uri=/acl  
1508154356: |-- getuser_params=(null)  
1508154356: |-- superuser_params=(null)  
1508154356: |-- aclcheck_params=(null)  
1508154356: Opening ipv4 listen socket on port 1883.  
1508154356: Opening ipv6 listen socket on port 1883.
```

Gambar 5.10 Menjalankan Mosquitto Broker

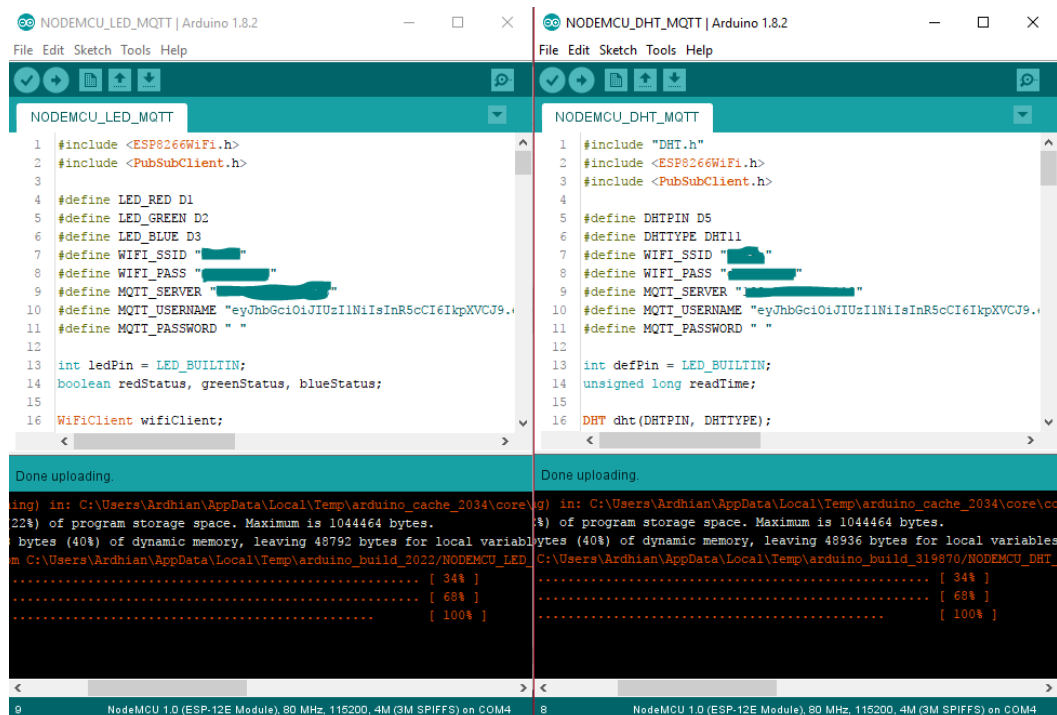
2. Jalankan auth-server, dilakukan dengan menjalankan *source code* auth-server dengan python menggunakan perintah “`python <nama-file-source-code-auth-server>.py`” pada terminal, auth-server dan mosquitto broker akan dijalankan pada waktu yang bersamaan agar keduanya dapat saling berkomunikasi.



```
ardhiaan@ardhiaan-VirtualBox: ~  
ardhiaan@ardhiaan-VirtualBox:~$ python testJWT/http-auth.py  
Bottle v0.12.13 server starting up (using WSGIRefServer())...  
Listening on http://localhost:8100/  
Hit Ctrl-C to quit.
```

Gambar 5.11 Menjalankan Auth-Server

3. Selanjutnya adalah melakukan *upload* kedua *source code* untuk nodeMCU dengan LED dan DHT11 melalui arduino IDE, agar kedua perangkat nodeMCU dapat menjalankan kode program yang sudah dibuat.



Gambar 5.12 Upload Source Code NodeMCU