Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer e-ISSN: 2548-964X
Vol. 3, No. 10, Oktober 2019, hlm. 9357-9366 http://j-ptiik.ub.ac.id

Penerapan Metode *Token-Based* sebagai Mekanisme Autentikasi pada IoT *Middleware*

Aditya Chamim Pratama[1], Eko Sakti Pramukantoro[2], Achmad Basuki[3]

Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Brawijaya
Email: [1]adityacprtm@gmail.com, [2]ekosakti@ub.ac.id, [3]abazh@ub.ac.id

Abstrak

IoT *middleware* yang ada saat ini telah dikembangkan untuk menjawab tantangan *syntactical interoperability* dengan membuat *gateway* agar dapat berkomunikasi melalui protokol CoAP, MQTT dan WebSocket. Namun, *middleware* tersebut dapat menerima data yang berasal dari mana saja dan berbagai *node* yang tidak berhak untuk terhubung dengan *middleware*. Sehingga dibutuhkan mekanisme tambahan pada IoT *middleware* untuk melakukan autentikasi terhadap *node* yang akan berkomunikasi. Autentikasi merupakan jaminan bahwa semua yang terlibat dalam komunikasi sistem adalah entitas yang valid. Oleh karena itu, pada penelitian ini dilakukan pengembangan pada IoT *middleware* dengan menambahkan mekanisme autentikasi *JSON Web Token* (JWT). JWT adalah *token* yang merepresentasikan sebuah pesan ringkas yang disebut *claims* dan dilindungi dengan digital *signature*, selain itu *token* JWT memiliki *expiration time*. Penggunaan *token* JWT dikarenakan ukuran yang ringkas sehingga memungkinkan untuk dikirim melalui URL, *query* ataupun *header*. Berdasarkan hasil pengujian, mekanisme autentikasi yang diterapkan mampu melakukan autentikasi *token* yang dikirim oleh *node*. Dampak penerapan mekanisme autentikasi pada IoT *middleware* menyebabkan peningkatan waktu yang diperlukan *node* ketika membangun koneksi dan melakukan *publish* atau *subscribe*. Dibandingkan dengan IoT *middleware* tanpa autentikasi, rata-rata selisih waktu yang dihasilkan yaitu di bawah 30 milidetik.

Kata kunci: *autentikasi, Internet of Things (IoT), middleware, JSON Web Token (JWT)*

Abstract

The existing IoT middleware has been developed to answer the challenge of syntactical interoperability by creating a gateway so that it can communicate through the CoAP, MQTT and WebSocket protocols. However, the middleware can accept data from anywhere and various nodes that are not entitled to connect to the middleware. So we need an additional mechanism on the IoT middleware to authenticate the node that will communicate. Authentication is a guarantee that all involved in system communication are valid entities. Therefore, in this research, the development of IoT middleware is done by adding the JSON Web Token (JWT) authentication mechanism. JWT is a token that represents a concise message called claims and is protected by a digital signature, in addition, JWT tokens have an expiration time. JWT tokens are used because of their compact size so they can be sent via URL, query or header. Based on the test results, the authentication mechanism that is applied is able to authenticate the token sent by the node. The impact of applying the authentication mechanism to the IoT middleware causes the node to increase the time it takes to establish connections and publish or subscribe. Compared to IoT middleware without authentication, the average time difference generated is below 30 milliseconds.

Keywords: *authentication, Internet of Things (IoT), middleware, JSON Web Token (JWT)*

1. PENDAHULUAN

Secara umum *Internet of Things* (IoT) membutuhkan infrastruktur yang memilik tiga elemen utama yaitu sensor atau aktuator, internet *gateway* (*middleware*) dan data *center* atau *cloud* (Hamid, Pramukantoro and Siregar, 2018). *Middleware* pada IoT memiliki peranan penting yaitu melakukan integrasi perangkat yang heterogen, memungkinkan perangkat untuk berkomunikasi dan menjamin keamanan perangkat dalam berkomunikasi. Anwari,

Pramukantoro & Hanafi, (2017) telah membangun sebuah *middleware* yang menyediakan *gateway* dengan multi-protokol untuk menjawab salah satu permasalahan dalam lingkungan IoT yaitu *Syntactical Interoperability*. *Syntactical Interoperability* merujuk pada beragam protokol yang dipakai untuk proses pengiriman data atau pertukaran data seperti protokol CoAP, MQTT, HTTP, XMPP, dan lainnya. *Gateway* yang dibuat pada *middleware* tersebut meliputi *gateway* MQTT, CoAP dan WebSocket yang dihubungkan dengan broker. *Middleware* yang dibangun mampu menerima data yang dikirim oleh *publisher* melalui protokol CoAP dan MQTT serta mampu mengirimkan data ke *subscriber* melalui protokol WebSocket. Namun, *middleware* tersebut dapat menerima data atau koneksi yang bisa berasal dari mana saja dan bahkan dari berbagai *node* yang tidak berhak untuk terhubung dengan *middleware*. Dalam mengatasi hal tersebut dapat dengan menerapkan sebuah mekanisme autentikasi (Lin et al., 2017). Autentikasi merupakan jaminan bahwa semua yang terlibat dalam komunikasi sistem adalah entitas yang valid (Tiburski et al., 2016).

Beberapa penelitian terdahulu telah melakukan penerapan mekanisme autentikasi dalam bidang IoT. Penelitian terdahulu (Bhawiyuga, Data & Warda, 2018) telah dilakukan perancangan dan implementasi autentikasi berbasis *token* menggunakan *token* JWT pada protokol MQTT di perangkat terbatas. Desain pada penelitian tersebut terdiri dari empat komponen yaitu *publisher*, *subscriber*, MQTT broker dan server autentikasi JWT. Pada penelitian lain (Putra, Pramukantoro & Bakhtiar, 2019) dilakukan implementasi protokol CoAP pada IoT *semantic* Web Service dengan menerapkan proses autentikasi dan otorisasi menggunakan *JSON Web Token* (JWT). Kedua penelitian tersebut memiliki alur komunikasi yang hampir sama yaitu *publisher* atau *client* meminta *token* dengan mengirimkan kredensial, kemudian menggunakan *token* tersebut untuk melakukan pertukaran data. Selain itu, terdapat penelitian yang dilakukan oleh Hamid, Pramukantoro & Siregar, (2018) yang menerapkan mekanisme autentikasi berbasis *token* menggunakan JWT untuk membatasi aplikasi dalam melakukan POST ataupun GET data pada data *storage* IoT.

Berangkat dari uraian sebelumnya, dalam penelitian ini dilakukan pengembangan dan penambahan fitur pada IoT *middleware* sebelumnya agar mampu melayani komunikasi dan pertukaran data hanya dengan entitas yang valid. Pada *middleware* tersebut ditambahkan mekanisme autentikasi berbasis *token* menggunakan *JSON Web Token* (JWT). JWT adalah *token* yang merepresentasikan sebuah pesan ringkas yang disebut *claims* dan dilindungi dengan digital *signature*, selain itu *token* JWT memiliki *expiration time*. Penggunaan *token* JWT dikarenakan ukuran yang ringkas sehingga memungkinkan untuk dikirim melalui URL, *query* ataupun *header*. Hal ini memungkinkan *token* JWT dapat digunakan dalam lingkungan IoT. Penelitian ini membahas penerapan mekanisme autentikasi berbasis *token* pada IoT *middleware* dan mengetahui kinerja mekanisme autentikasi dalam melakukan autentikasi.

2. LANDASAN KEPUSTAKAAN

Middleware yang dibangun oleh Anwari, Pramukantoro & Hanafi, (2017) terbagi menjadi tiga sektor utama yaitu sektor sensor *gateway*, sektor *service unit* dan sektor *application gateway*. Sensor *gateway* sendiri berperan sebagai antarmuka untuk *node* sensor atau *publisher* agar dapat berkomunikasi dengan *middleware*, sedangkan *application gateway* berperan sebagai antarmuka untuk aplikasi atau *subscriber* agar dapat berkomunikasi dengan *middleware*. Bagian *service* unit berperan menjadi broker untuk menjembatani komunikasi sensor *gateway* dengan *application gateway* dengan menyediakan beberapa antarmuka.

2.1 Constrained Application Protocol (CoAP)

CoAP merupakan bentuk sederhana dari HTTP dengan menggunakan *method* dari bagian REST dan dilakukan optimasi agar mendukung komunikasi M2M. *Method* yang digunakan yakni POST, PUT, GET dan DELETE. Skema URI CoAP diawali dengan kode *coap* atau *coaps* untuk mengenali CoAP Resource. Setiap pesan memiliki *Message ID* yang digunakan untuk mendeteksi kemungkinan pesan terduplikasi dan untuk pilihan *reliability* yaitu *Confirmable* (CON) atau *Non-confirmable* (NON) (Shelby, Hartke and Bormann, 2014).

2.2 Message Queuing Telemetry Transport (MQTT)

Dengan karakteristik yang ringan (*lightweight*) dan mudah dalam penerapan

menjadikan MQTT dapat dipakai dalam banyak situasi termasuk lingkungan IoT. MQTT memakai pola *publish-subscribe* yang komunikasi antar *client endpoint* tidak dilakukan secara langsung. Sehingga *publish-subscribe* terdiri dari dua bagian yaitu MQTT *Client* dan MQTT Broker. MQTT *Client* sendiri terbagi menjadi dua jenis yaitu *publisher* dan *subscriber*. MQTT memiliki tiga level *Quality of Service* (QoS) dalam pengiriman pesan yaitu level 0 (*at most once*), level 1 (*at least once*) dan level 2 (*exactly once*). MQTT memiliki fitur keamanan berupa autentikasi yang berada pada level *transport* dan level *application*. Pada level *application*, MQTT menyediakan *field username-password* pada pesan CONNECT untuk mekanisme autentikasi (HiveMQ, 2015).

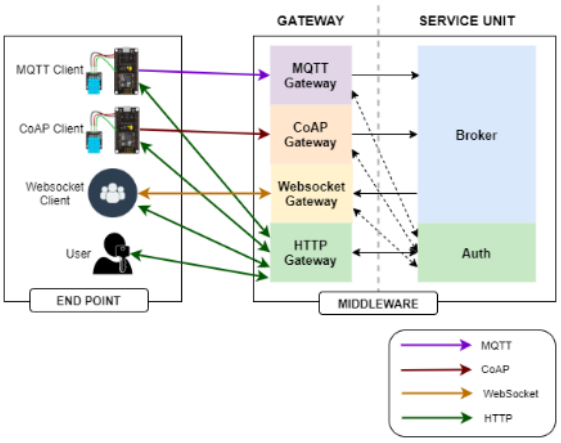
2.3 WebSocket

WebSocket didesain untuk menggantikan penggunaan protokol HTTP dalam skema komunikasi yang terjadi dua arah (Fette and Melnikov, 2011). Protokol WebSocket merupakan bagian dalam konektivitas dari spesifikasi HTML5 yang memungkinkan pembuatan *full-duplex*, koneksi *bidirectional* antara *client-server* melalui *subscriber*. Protokol WebSocket bersifat asinkron yang artinya selama koneksi WebSocket terbuka. aplikasi memungkinkan untuk berkomunikasi dengan menggunakan WebSocket Event (Skvorc, Horvat and Srbljic, 2014).

2.4 JSON Web Token (JWT)

JWT merupakan gambaran *claims* ringkas yang ditujukan kepada *space constrained environments* seperti *Authorization headers* pada HTTP dan parameter *query* pada URI. Data informasi dalam *claims* dapat dikonfirmasi dan dipercaya karena adanya *digitally signed* atau *integrity protected* menggunakan *Message Authentication Code* (MAC) (B. Jones, Bradley and Sakimura, 2015). Selain untuk transfer *claims* antara dua pihak, JWT dapat digunakan untuk beberapa aplikasi yakni *authentication*, *authorization*, *client-side sessions* (*stateless sessions*) dan *client-side secrets* (Sebastián and Auth0 Inc., 2018). Struktur JWT direpresentasikan dengan tiga bagian yang dipisahkan oleh karakter titik (“.”), bagian-bagian tersebut adalah *header*, *payload* dan *signature*. *Header* berisi informasi algoritma *hashing*, *payload* berisi informasi *user* atau informasi lainnya, dan *signature* untuk memastikan *token* tidak berubah selama pengiriman (Sebastián and Auth0 Inc., 2018).

3. ANALISIS KEBUTUHAN



Gambar 1. Lingkungan penelitian saat ini

Penelitian yang dilakukan bersifat implementasi pengembangan dari penelitian sebelumnya yang telah dilakukan oleh Anwari, Pramukantoro & Hanafi, (2017). Pada penelitian ini dilakukan penambahan mekanisme autentikasi pada IoT *middleware*. Gambar 1 memperlihatkan yang dikerjakan dalam penelitian ini, yaitu menambahkan mekanisme autentikasi pada IoT *middleware*. Untuk menunjang mekanisme autentikasi JWT, dilakukan penambahan fitur meliputi modul *Auth* pada *service unit*, HTTP *Gateway* dan komunikasi melalui protokol HTTP. Untuk dapat berkomunikasi dengan *middleware*, *node* harus terautentikasi dengan mengirimkan *token* yang didapat setelah melakukan *request* melalui HTTP *Gateway*. Untuk mendapatkan *token*, *node* harus didaftarkan oleh *user* ke *middleware*. Tiap *gateway* terintegrasi dengan modul *auth* untuk proses autentikasi.

3.1 Kebutuhan Fungsional

Daftar kebutuhan fungsional dari sistem dalam penelitian ini disajikan pada Tabel 1.

3.2 Kebutuhan Non-fungsional

Kebutuhan non-fungsional pada penelitian ini diklasifikasikan jadi dua bagian yakni kebutuhan perangkat keras yang ditunjukkan pada Tabel 2 dan kebutuhan perangkat lunak pada Tabel 3.

Tabel 1. Kebutuhan fungsional

No	Kebutuhan Fungsional
1	*Middleware* mampu menghasilkan *token* dari *request* yang valid
2	*Middleware* mampu menolak *request token* yang tidak valid
3	*Middleware* mampu menolak koneksi tanpa *token*
4	*Middleware* mampu melakukan autentikasi terhadap *token* yang valid
5	*Middleware* mampu melakukan validasi terhadap *token* yang kedaluwarsa
6	*Middleware* mampu melakukan validasi peran *node* pada *token* yang valid
7	*Middleware* mampu menerima data valid yang berasal dari *node* sensor berbasis CoAP dan mengirimkan ke *subscriber* melalui protokol WebSocket.
8	*Middleware* mampu menerima data valid yang berasal dari *node* sensor berbasis MQTT dan mengirimkan ke *subscriber* melalui protokol WebSocket.
9	Middleware mampu menangani login *user*
10	*Middleware* mampu melayani CRUD (*Create*, *Read*, *Update* dan *Delete*) akun
11	*Middleware* mampu melayani CRUD (*Create*, *Read*, *Update* dan *Delete*) *things*

Tabel 2. Kebutuhan perangkat keras

Perangkat	Keterangan
Raspberry Pi	Perangkat yang digunakan untuk menjalankan middleware
NodeMCU	Digunakan sebagai *microcontroller* bagi sensor suhu dan kelembapan
DHT11	Modul sensor berfungsi mengambil nilai suhu dan kelembapan
Laptop	Digunakan untuk menjalankan *subscriber*

Tabel 3 Kebutuhan perangkat lunak

Perangkat	Keterangan
Redis	Berfungsi sebagai broker dan sebagai media penyimpanan data sementara didalam memori
Raspbian	Sistem operasi yang dijalankan pada perangkat Raspberry Pi
SQLite	Untuk penyimpanan data node dan user
Node.js	*Framework dengan bahasa JavaScript yang digunakan dalam menjalankan middleware*
NodeMCU Firmware	*Firmware untuk microcontroller NodeMCU berbasis LUA*
ESPlorer	Sebagai IDE untuk pemrograman ESP32
Postman	Digunakan dalam melakukan pengujian

4. PERANCANGAN

Hubungan antar komponen terdiri dari dua bagian yaitu pengiriman data dari *publisher* ke *middleware* dan antara *middleware* dengan *subscriber*. *Middleware* diterapkan mekanisme autentikasi JWT terhadap *node* yang ingin berkomunikasi. Untuk menunjang mekanisme autentikasi, terdapat interaksi tambahan yaitu antara *user* dengan *middleware* untuk mendaftarkan *node*.

JWT memiliki fitur *expiration time* pada *token* yang dihasilkan, sehingga *token* tidak dapat digunakan kembali ketika *token* telah kedaluwarsa atau waktu telah habis. *Token* akan di-*generate* oleh sistem untuk *node* yang melakukan *request* dengan ketentuan: (1) belum pernah melakukan *request* dan belum memiliki *token*, (2) *token* telah kedaluwarsa.

Pada bagian *service unit* ditambahkan modul *auth* yang berfungsi untuk melakukan mekanisme autentikasi. Modul *auth* memiliki tiga fungsi utama yaitu manajemen akun dan *things* serta *database*. Sehingga pada *service unit* ditambahkan *interface* yang ditunjukkan pada Tabel 4 guna autentikasi.

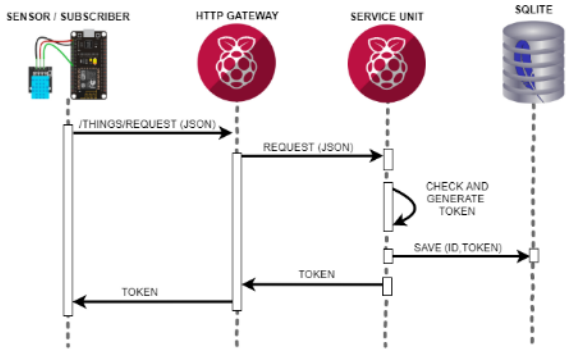
Tabel 4. Interface service unit

Interface	Keterangan
Request (payload)	Meminta untuk melakukan *generate token*
Validity (token)	Melakukan validasi *token*

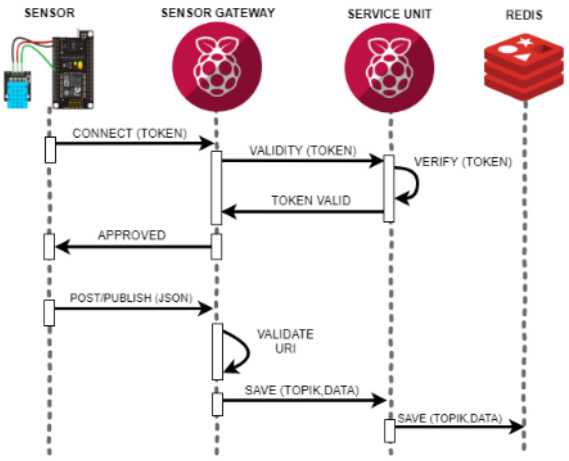
Tiap *gateway* yang tersedia pada IoT *middleware* diterapkan mekanisme autentikasi JWT, yaitu pada CoAP *gateway*, MQTT *gateway* dan WebSocket *gateway*.

Sebelum *node* sensor dapat melakukan pengiriman data, terlebih dahulu melakukan *connect* dengan menyertakan *token*. *Token* didapat dari proses POST *request token* ke HTTP *Gateway* dengan mengirimkan kredensial berupa *things_id* dan *things_password* seperti yang ditunjukkan pada Gambar 2. *Things_id* dan *things_password* didapat ketika *node* telah didaftarkan *user* melalui *web service* yang disediakan oleh *middleware*. Untuk MQTT, *token* dikirimkan ketika proses membangun koneksi (*connect*) dengan *middleware*. Sedangkan pada CoAP, *token* dapat diletakkan pada *query* atau *payload* karena tidak memiliki proses membangun koneksi seperti halnya MQTT. Selanjutnya sensor *gateway* akan melakukan validasi *token* yang diterima ke *service unit* melalui *interface validity*. Jika *node*

sensor melakukan *publish* data tanpa *token*, *gateway* akan menolak. Proses pada sensor *gateway* ditampilkan dalam *sequence diagram* yang terlihat di Gambar 3.



Gambar 2. HTTP gateway



Gambar 3. Autentikasi sensor gateway

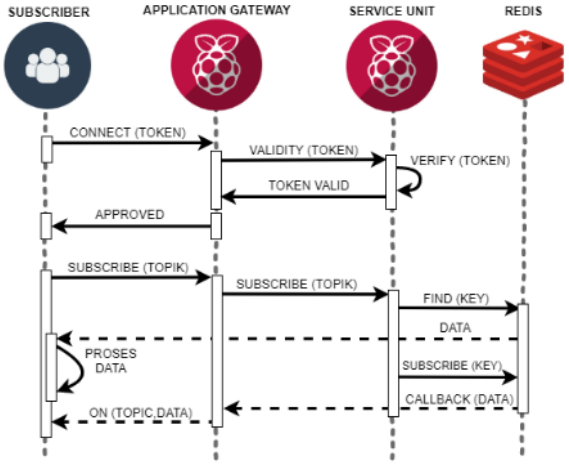
Sama halnya dengan sensor *gateway*, *application gateway* juga menerapkan mekanisme autentikasi JWT. Pada proses *connect*, *subscriber* menyertakan *token* yang telah didapat sebelumnya melalui HTTP *Gateway*. Proses detail yang terjadi pada *application gateway* ditampilkan dalam bentuk *sequence* diagram yang ditampilkan di Gambar 4.

5. IMPLEMENTASI

Bagian ini menjelaskan penambahan mekanisme autentikasi pada IoT *middleware* dan juga implementasi *publisher* serta *subscriber*.

5.1 Implementasi *Middleware*

Dalam implementasi *middleware* dilakukan integrasi antara sistem *middleware* yang ada dengan penambahan mekanisme autentikasi meliputi Web Services dan modul *Auth*.



Gambar 4. Autentikasi application gateway

5.1.1 Implementasi Web Service

Untuk menerapkan Web Service diperlukan sebuah Web Server. Penelitian ini menggunakan Node.js yang memiliki banyak *package* untuk menjalankan web server, salah satunya Express. Sebagai penyimpanan data digunakan *SQLite*. Untuk melakukan instalasi *package* tersebut, digunakan *tool* yang tersedia dalam Node.js yaitu NPM (*Node Package Manager*). Web Service berfungsi untuk mendaftarkan *node* sensor atau aplikasi yang akan berkomunikasi dengan *middleware* untuk mendapatkan *things_id* dan *things_password*.

5.1.2 Implementasi Modul *Auth*

Didalam modul *Auth* terdapat dua *interface* utama, yaitu *interface request* dan *validity*. *Interface request* berfungsi untuk melakukan *generate token* dari *request* yang berasal dari HTTP *Gateway*. Dalam fungsi *request* yang ditunjukkan pada Tabel 5 dilakukan pengecekan kredensial *things_id* dan *things_password* yang diterima apakah tersimpan di dalam *database*.

Interface validity digunakan untuk melakukan validasi *token* yang berasal dari tiap *gateway*. Fungsi *validity* yang ditunjukkan pada Tabel 6 akan mengecek status *token* apakah valid atau tidak valid dan apakah kedaluwarsa atau belum.

Tabel 5. Pseudocode request

```
Fungsi Request
1  DO function request(payload)
2     GET data from database DO function
3        IF error
4           RETURN error
5        END
6        IF data != null
7           SET token = get data token
8           IF token != null
9              DO verifyToken(token)
```

```
            ELSE
                DO generateToken(data)
                    RETURN roken
                END
            END
        ELSE
            RETURN 'Things Not Registered'
        END
    END
END
```

Tabel 6. Pseudocode validity

```
Fungsi Validity
DEFINE function validity(token)
    DO verifyToken(token)
        IF error
            RETURN error, status = invalid
        ELSE
            Check things_id in database
            IF things_id exist
                RETURN status = valid
            ELSE
                RETURN status = invalid
            END
        END
    END
END
```

5.1.3 Implementasi Mekanisme Autentikasi

Tiap *gateway* pada IoT *middleware* yang meliputi CoAP *gateway*, MQTT *gateway* dan WebSocket *gateway* diterapkan mekanisme autentikasi. Penerapan mekanisme autentikasi tersebut dengan melakukan integrasi *gateway* dengan modul *Auth* agar dapat melayani request token dan melakukan validasi *token*. Sensor dan aplikasi *gateway* akan menggunakan *interface validity* untuk melakukan validasi *token* yang diterima oleh *node* yang ingin terhubung, sedangkan HTTP *gateway* digunakan untuk *request token* oleh *node* dan aplikasi serta akses web *service* oleh *user*.

5.2 Implementasi Publisher dan Subscriber

Publisher dijalankan pada perangkat NodeMCU dengan menggunakan bahasa pemrograman LUA dan dihubungkan dengan modul DHT11. Dalam melakukan implementasi *node* sensor sebagai *publisher* ini, terdapat tiga langkah yaitu *build firmware*, *flash firmware* dan *upload code* aplikasi. Terdapat dua *publisher* yang masing-masing berfungsi mengirimkan data melalui protokol CoAP dan MQTT. Publisher bertugas mengirimkan data berupa *payload* berformat JSON seperti pada Tabel 7.

Subscriber berupa aplikasi dengan menjalankan *script* menggunakan *javascript* yang berfungsi untuk meminta data. Pada implementasinya *subscriber* memerlukan *package socket.io-client* dan *sync-request*.

Tabel 7. Struktur data

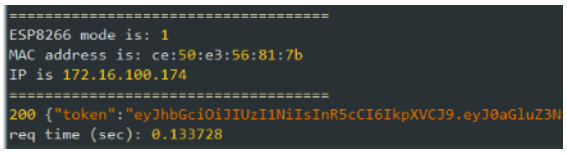
```
{
    "protocol": "string",
    "timestamp": "string",
    "topic": "string",
    "humidity": {
        "value": "number",
        "unit": "string"
    },
    "temperature": {
        "value": "number",
        "unit": "string"
    }
}
```

6. PENGUJIAN DAN PEMBAHASAN

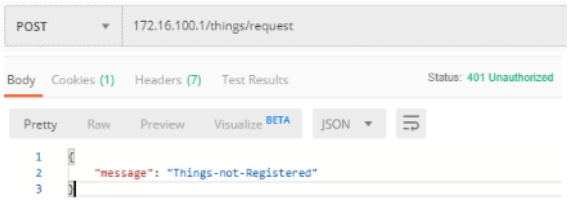
Pengujian dilakukan untuk melihat kemampuan dan kesesuaian sistem. Pengujian terdiri dari pengujian *request token*, pengujian autentikasi, pengujian validitas data, pengujian web *service* dan kinerja.

6.1 Pengujian Request Token

Pengujian dilakukan dengan menggunakan dua jenis *request* yakni valid dan tidak valid. *Request* yang valid merupakan *request* dengan menyertakan kredensial berupa *things_id* dan *things_password*. Sedangkan request yang tidak valid yaitu tanpa menyertakan kredensial berupa *things_id* dan *things_password*. Request dikirim ke HTTP *Gateway* pada *middleware*. Gambar 5 menunjukkan *node* menerima *token* setelah melakukan *request* valid. Sedangkan Gambar 6 menunjukkan hasil dari *request* yang tidak valid.



Gambar 5. Hasil pengujian request valid



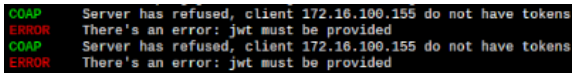
Gambar 6. Hasil pengujian request tidak valid

6.2 Pengujian Autentikasi

Pengujian dilakukan dengan menggunakan tiga jenis uji yakni tanpa token, dengan token yang valid dan token yang kedaluwarsa.

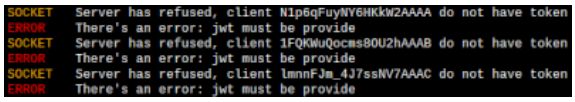
6.2.1 Pengujian tanpa Token

Pengujian dijalankan dengan skenario melakukan *publish* melalui CoAP dan MQTT dan melakukan *subscribe* melalui WebSocket tanpa menyertakan *token*. Hasil yang didapat menunjukkan *middleware* menolak *publish* atau koneksi yang dilakukan seperti yang terlihat pada Gambar 7 untuk CoAP, Gambar 8 untuk MQTT dan Gambar 9 untuk WebSocket.



Gambar 7. Hasil autentikasi tanpa token CoAP

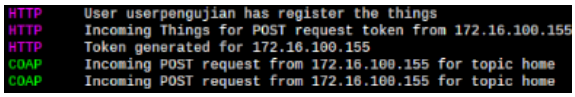
Gambar 8. Hasil autentikasi tanpa token MQTT



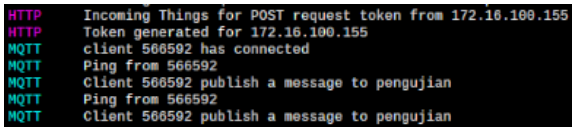
Gambar 9. Hasil autentikasi tanpa token WebSocket

6.2.2 Pengujian dengan Token yang Valid

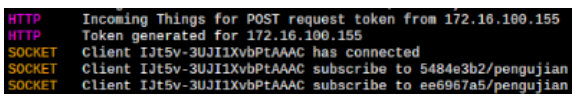
Pengujian dijalankan dengan skenario melakukan *publish* melalui CoAP dan MQTT dan melakukan *subscribe* melalui WebSocket dengan menyertakan *token* yang valid. Hasil yang didapat menunjukkan bahwa *middleware* menerima koneksi dan data *publish* yang dilakukan seperti yang terlihat pada Gambar 10 untuk CoAP, Gambar 11 untuk MQTT dan Gambar 12 untuk WebSocket.



Gambar 10. Hasil autentikasi token valid CoAP



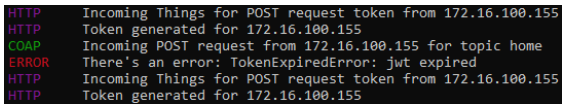
Gambar 11. Hasil autentikasi token valid MQTT



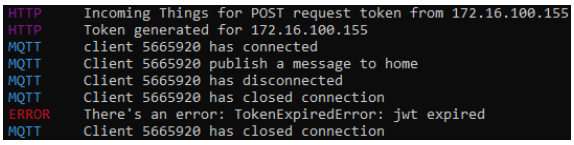
Gambar 12. Hasil autentikasi token valid WebSocket

6.2.3 Pengujian dengan Token yang Kedaluwarsa

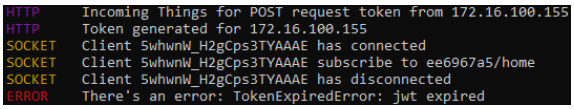
Pengujian dijalankan dengan skenario melakukan *publish* melalui CoAP dan MQTT dan melakukan *subscribe* melalui WebSocket menggunakan *token* yang telah kedaluwarsa. Hasil yang didapat menunjukkan bahwa *middleware* menolak *publish* atau koneksi yang dilakukan seperti yang terlihat pada Gambar 13 untuk CoAP, Gambar 14 untuk MQTT dan Gambar 15 untuk WebSocket.



Gambar 13. Hasil autentikasi token kedaluwarsa CoAP



Gambar 14. Hasil autentikasi token kedaluwarsa MQTT

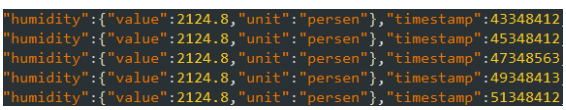


Gambar 15. Hasil autentikasi token kedaluwarsa WebSocket

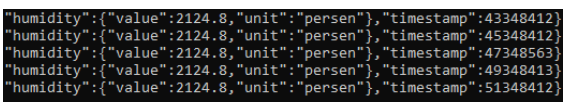
6.3 Pengujian Validitas Data

Pengujian dijalankan dengan mengecek kesamaan data antara data yang di *publish* melalui CoAP dan MQTT dengan data yang didapat disisi *subscriber* dengan topik sama.

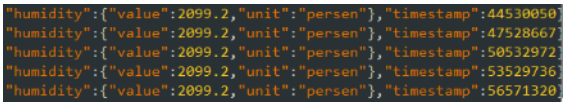
Hasil perbandingan data menunjukkan data yang dikirim oleh *publisher* melalui CoAP pada Gambar 16 memiliki kesamaan data disisi *subscriber* pada Gambar 17. Selain itu, data yang dikirim oleh *publisher* berbasis MQTT pada Gambar 18 memiliki kesamaan data yang didapat disisi *subscriber* pada Gambar 19. Kesesuaian data tersebut dapat dilihat dari nilai *timestamp*.



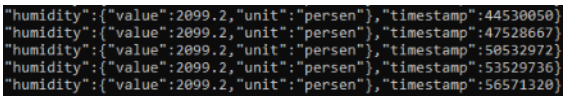
Gambar 16. Data publish berbasis CoAP



Gambar 17. Data CoAP yang diterima subscriber



Gambar 18. Data publish berbasis MQTT



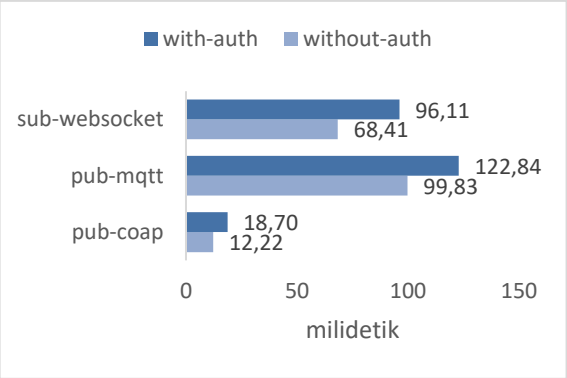
Gambar 19. Data MQTT yang diterima subscriber

6.4 Pengujian Kinerja

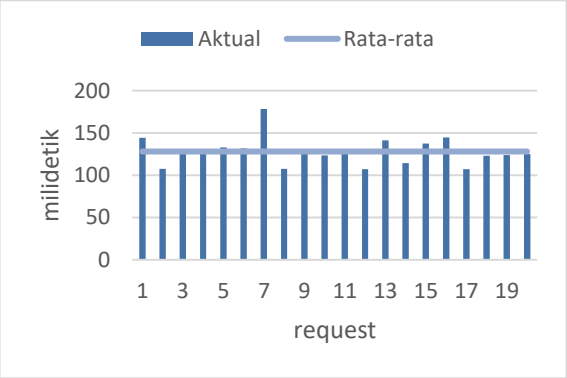
Pengujian kinerja dimaksudkan untuk melihat kompetensi dari *middleware* dengan penerapan mekanisme autentikasi. Pengujian ini berfokus pada parameter waktu yang dibutuhkan dalam melakukan *publish* data sebanyak 100 kali untuk mendapatkan hasil rata-rata waktu yang dibutuhkan. Keberhasilan *publish* data ditentukan jika *publisher* menerima *ACK*. Untuk mendapatkan data parameter yang digunakan, pencatatan waktu dilakukan disisi *node*. Selain *publish*, juga dilakukan pengujian waktu yang dibutuhkan dalam melakukan *subscribe* melalui WebSocket sebanyak 100 kali. Pengujian ini dilakukan pada *middleware* sebelumnya tanpa mekanisme autentikasi dan *middleware* penelitian saat ini dengan mekanisme autentikasi. Hal itu dilakukan untuk melihat perbandingan dan dampak yang dihasilkan dari penerapan mekanisme autentikasi pada IoT *middleware*.

Gambar 20 memperlihatkan grafik perbandingan waktu ketika melakukan *publish* dan *subscribe* antara IoT *middleware* dengan mekanisme autentikasi dan yang tidak. Hasil menunjukkan penerapan mekanisme autentikasi memberi dampak peningkatan waktu dalam melakukan *publish* oleh *client* CoAP dan juga MQTT. Hal yang sama ditunjukkan ketika *client* WebSocket melakukan *subscribe* yang memerlukan tambahan waktu.

Gambar 21 menunjukkan grafik waktu yang diperlukan ketika melakukan *request token* hingga menerima *token* dengan waktu rata-rata yaitu 128.05 milidetik.



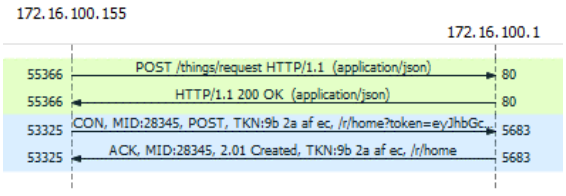
Gambar 20. Grafik perbandingan waktu



Gambar 21. Grafik waktu request token

6.5 Network Flow Proses Autentikasi

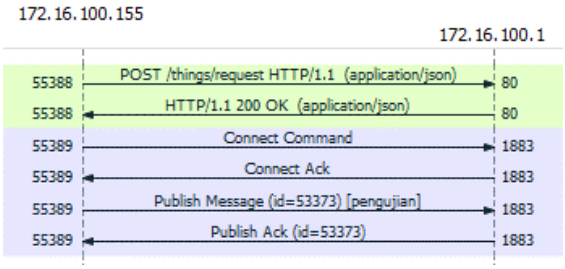
Gambar 22 menunjukkan proses autentikasi dengan *token*, yang mana *middleware* menerima koneksi dari *client* CoAP ketika melakukan *publish*. Sebelum melakukan *publish*, *client* CoAP melakukan *request token* melalui protokol HTTP, kemudian *middleware* merespons dengan mengirimkan *token*. Selanjutnya *client* CoAP melakukan *publish* dengan mengirimkan POST *request* berisi *payload* dan *token* yang valid, *middleware* menerima data dan merespons dengan mengirimkan ACK dengan kode respons 2.01 (*Created*). Pada CoAP, *token* dapat dimasukkan dalam *query* ataupun *payload* ketika melakukan *publish*.



Gambar 22. Network flow autentikasi CoAP

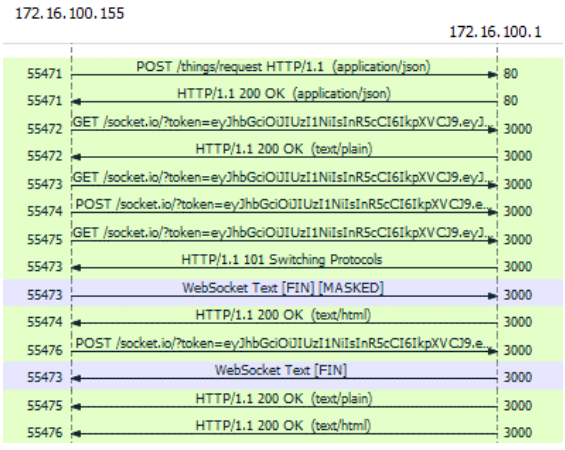
Gambar 23 menunjukkan proses autentikasi dengan *token*, *middleware* menerima koneksi

dari *client* MQTT ketika melakukan *connect*. Sebelum melakukan *publish*, *client* MQTT melakukan *request token* melalui protokol HTTP, kemudian *middleware* merespons dengan mengirimkan *token*. Selanjutnya *client* MQTT membangun koneksi dengan mengirimkan *Connect Message* dengan *username* berupa *token*, *middleware* menerima koneksi dengan mengirimkan ACK kode respons 0 (*Connection accepted*). Kemudian *client* dapat melakukan *publish*.



Gambar 23. Network flow autentikasi MQTT

Gambar 24 menunjukkan proses autentikasi dengan *token*, *middleware* menerima koneksi dari *client* WebSocket ketika membangun koneksi. Setelah proses perubahan koneksi ke WebSocket berhasil, *client* WebSocket membangun koneksi dengan menyertakan *token* pada bagian *query*. Kemudian *gateway* melakukan validasi *token* yang dikirimkan, karena *token* valid maka koneksi antara *client* WebSocket dan *middleware* terbangun.



Gambar 24. Network flow autentikasi WebSocket

7. KESIMPULAN

Bersumber pada hasil perancangan dan implementasi serta proses pengujian, dapat ditarik kesimpulan bahwa metode autentikasi berbasis *token* dengan menggunakan *JSON Web Token* (JWT) dapat diimplementasikan pada IoT *middleware*. Mekanisme autentikasi tersebut diterapkan pada tiap *gateway* yang tersedia pada *middleware*. Semua *gateway* terintegrasi dengan modul *auth* yang berfungsi untuk proses autentikasi. Diperlukan pendaftaran *node* oleh *user* ke web *service* untuk mendapatkan kredensial yang digunakan oleh *node* ketika melakukan *request token*.

Penerapan autentikasi menyebabkan peningkatan waktu yang diperlukan oleh *node* ketika membangun koneksi dan melakukan *publish* ataupun *subscribe*. Dibandingkan dengan IoT *middleware* tanpa autentikasi, rata-rata selisih waktu yang dihasilkan yaitu di bawah 30 milidetik. Dilain sisi, peningkatan waktu tersebut memberikan dampak positif yaitu tidak sembarang *node* dapat mengirim atau meminta data ke *middleware*. *Middleware* mampu menolak koneksi yang dibangun oleh *node* tanpa menyertakan *token* dan mampu menerima koneksi dari *node* dengan *token* yang valid. *Middleware* juga dapat memvalidasi *token* yang telah kedaluwarsa.

8. DAFTAR PUSTAKA

Anwari, H., Pramukantoro, E.S. and Hanafi, M.H., 2017. Pengembangan Iot Middleware Berbasis Event-Based dengan Protokol Komunikasi CoAP, MQTT dan Websocket. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer (J-PTIIK) Universitas Brawijaya*.

B. Jones, M., Bradley, J. and Sakimura, N., 2015. *JSON Web Token (JWT)*. [online] Internet Engineering Task Force (IETF). Available at: <https://tools.ietf.org/html/rfc7519> [Accessed 1 Apr. 2019].

Bhawiyuga, A., Data, M. and Warda, A., 2018. Architectural design of token based authentication of MQTT protocol in constrained IoT device. *Proceeding of 2017 11th International Conference on Telecommunication Systems Services and Applications, TSSA 2017*, 2018-Janua, pp.1–4.

Fette, I. and Melnikov, A., 2011. *The WebSocket Protocol*. [online] Internet Engineering Task Force (IETF). Available at: <https://tools.ietf.org/html/rfc6455> [Accessed 2 Apr. 2019].

Hamid, T.W., Pramukantoro, E.S. and Siregar, R.A., 2018. Pengembangan Web Service Pada IoT Data Storage Dengan Pendekatan Semantik dan Penambahan Mekanisme Autentikasi. 2(12), pp.6824–6827.

HiveMQ, 2015. *MQTT Security Fundamentals*. [online] Available at: <https://www.hivemq.com/blog/mqtt-security-fundamentals-authentication-username-password/> [Accessed 6 Mar. 2019].

Lin, J., Yu, W., Zhang, N., Yang, X., Zhang, H. and Zhao, W., 2017. A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications. *IEEE Internet of Things Journal*.

Putra, M.R., Pramukantoro, E.S. and Bakhtiar, F.A., 2019. Implementasi Constrained Application Protocol (CoAP) Pada Semantic IoT Web Service. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer (J-PTIIK) Universitas Brawijaya*, 3(5), pp.4671–4679.

Sebastián, P. and Auth0 Inc., 2018. *The JWT Handbook*. 0.14.1 ed.

Shelby, Z., Hartke, K. and Bormann, C., 2014. *The Constrained Application Protocol (CoAP)*. [online] Internet Engineering Task Force (IETF). Available at: <https://tools.ietf.org/html/rfc7252> [Accessed 5 Mar. 2019].

Skvorc, D., Horvat, M. and Srbljic, S., 2014. Performance evaluation of Websocket protocol for implementation of full-duplex web streams. *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014 - Proceedings*, (May), pp.1003–1008.

Tiburski, R.T., Amaral, L.A., De Matos, E., De Azevedo, D.F.G. and Hessel, F., 2016. The Role of Lightweight Approaches Towards the Standardization of a Security Architecture for IoT Middleware Systems. *IEEE Communications Magazine*.