RANCANG BANGUN PERANGKAT LUNAK IOT CLOUD PLATFORM BERBASIS PROTOKOL

KOMUNIKASI HTTP

SKRIPSI

Untuk memenuhi sebagian persyaratan   
memperoleh gelar Sarjana Komputer

Disusun oleh:

Ocki Bagus Pratama

NIM: 135150207111060



TEKNIK INFORMATIKA

INFORMATIKA

FAKULTAS ILMU KOMPUTER

UNIVERSITAS BRAWIJAYA

MALANG

2016

PENGESAHAN

RANCANG BANGUN PERANGKAT LUNAK IOT GATEWAY DARI FIELD KE CLOUD BERBASIS PROTOKOL

KOMUNIKASI HTTP

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan

memperoleh gelar Sarjana Komputer

Disusun Oleh :

Ocki Bagus Pratama

NIM: 135150207111060

Skripsi ini telah diuji dan dinyatakan lulus pada

2 Januari 2015

Telah diperiksa dan disetujui oleh:

|  |  |
| --- | --- |
| Dosen Pembimbing I  Adhitya Bhawiyuga, S.Kom, M.S  NIK: 201405 890720 1 001 | Dosen Pembimbing II  Kasyful Amron, S.T, M.Sc  NIP: 19750803 200312 1 003 |

Mengetahui

Ketua Jurusan Informatika

Nama Ketua Jurusan

NIP: 123456789

/\*jika terdapat NIP\*/

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 1 Januari 2017



­

Ocki Bagus Pratama

NIM: 135150207111060

KATA PENGANTAR

Bagian ini memuat pernyataan resmi untuk menyampaikan rasa terima kasih penulis kepada berbagai pihak yang telah membantu penyelesaian skripsi ini. Nama-nama penerima ucapan terima kasih sebaiknya dituliskan lengkap, termasuk gelar akademik, dan pihak-pihak yang tidak terkait dihindari untuk dituliskan. Bahasa yang digunakan seharusnya mengikuti kaidah bahasa Indonesia yang baku. Kata pengantar boleh diakhiri dengan paragraf yang menyatakan bahwa penulis menerima kritik dan saran untuk pengembangan penelitian selanjutnya. Terakhir, kata pengantar ditutup dengan mencantumkan kota dan tanggal penulisan kata pengantar, lalu diikuti dengan kata “Penulis”.

Malang, 1 Januari 2017

Ocki Bagus Pratama

ocki.bagus.p@gmail.com

ABSTRAK

*Internet of Things* (IoT) dalam beberapa tahun terakhir telah berkembang menjadi sebuah tren teknologi internet. IoT bertujuan untuk memperluas manfaat dari konektivitas internet yang terhubung secara terus-menerus. Perangkat dalam IoT seringkali memiliki keterbatasan dalam hal kapasitas penyimpanan dan kemampuan komputasi. Hal tersebut menjadikan perangkat IoT memerlukan integrasi dengan sistem yang memiliki spesifikasi yang lebih besar, salah satunya yaitu *cloud*. Namun, integrasi antara perangkat IoT dan *cloud* memiliki tantangan dalam hal *network communication*, *security and privacy* dan manajemen perangkat. *Network communication* yaitu banyaknya teknologi jaringan yang digunakan, sehingga pemilihan protokol komunikasi harus mempertimbangkan dukungan oleh banyak perangkat. *Security and privacy* terjadi karena banyaknya perangkat IoT yang dapat dengan mudah terhubung *cloud*, sehingga peran otentikasi dan otorisasi diperlukan untuk mengidentifikasi dan memvalidasi perangkat yang mengirimkan data. Manajemen perangkat diperlukan karena banyaknya perangkat IoT yang dapat digunakan membutuhkan mekanisme untuk menajemen perangkat. Berdasarkan sebelumnya, diusulkan sebuah rancang bangun IoT *cloud* *platform* menggunakan protokol komunikasi HTTP untuk menyelesaikan kendala *network communication* dan manajemen perangkat. Sedangkan untuk mengidentifikasi dan memvalidasi perangkat yang mengirimkan data, digunakan mekanisme otentikas dan otorisasi menggunakan JSON Web Token. Hasil pengujian kehandalan sistem, didapati bahwa sistem yang dibangun mampu menangani hingga 100 pengguna dalam tiap detik dengan *error rate* sebesar 0%.

Kata kunci: *IoT*, *CloudIoT*, *cloud platform,* RESTful *web service*, HTTP

ABSTRACT

*Internet of Things (IoT) in recent years has evolved into an internet technology trends. IoT aims to expand the benefits of continuously connected Internet connectivity. Devices in IoT often have limitations in terms of storage capacity and computing capabilities. This makes the IoT device requires integration with systems that have a larger specification, one of them is cloud. However, the integration between IoT and cloud devices has challenges in terms of network communication, security and privacy and device management. Network communication is the number of network technology used, so the selection of communication protocols should consider the support by many devices. Security and privacy occurs because of the large number of IoT devices that can easily connect to the cloud, so an authentication and authorization role is required to identify and validate devices that transmit the data. Device management is required because the number of IOT devices that can be used requires a mechanism for device management. Based on earlier, it was proposed an IoT cloud platform design using HTTP communication protocol to solve network communication constraints and device management. As for identifying and validating devices that transmit data, use an authentication and authorization mechanisms using JSON Web Token. From system reliability test results, it was found that the built system capable of handling up to 100 users in every second with an error rate of 0%.*

*Keywords*: *IoT*, *CloudIoT*, *cloud platform,* RESTful *web service*, HTTP

DAFTAR ISI

[PENGESAHAN ii](#_Toc490453768)

[PERNYATAAN ORISINALITAS iii](#_Toc490453769)

[KATA PENGANTAR iv](#_Toc490453770)

[ABSTRAK v](#_Toc490453771)

[ABSTRACT vi](#_Toc490453772)

[DAFTAR ISI vii](#_Toc490453773)

[DAFTAR TABEL x](#_Toc490453774)

[DAFTAR GAMBAR xii](#_Toc490453775)

[DAFTAR LAMPIRAN xiv](#_Toc490453776)

[BAB 1 PENDAHULUAN 1](#_Toc490453777)

[1.1 Latar belakang 1](#_Toc490453778)

[1.2 Rumusan masalah 3](#_Toc490453779)

[1.3 Tujuan 3](#_Toc490453780)

[1.4 Manfaat 3](#_Toc490453781)

[1.5 Batasan masalah 3](#_Toc490453782)

[1.6 Sistematika penulisan 4](#_Toc490453783)

[BAB 2 LANDASAN KEPUSTAKAAN 6](#_Toc490453784)

[2.1 Kajian Pustaka 6](#_Toc490453785)

[2.2 Internet of Things 6](#_Toc490453786)

[2.3 Cloud 7](#_Toc490453787)

[2.4 HTTP 8](#_Toc490453788)

[2.4.1 Mekanisme Kerja 8](#_Toc490453789)

[2.4.2 Metode HTTP 9](#_Toc490453790)

[2.4.3 Header HTTP 10](#_Toc490453791)

[2.4.4 Kode status HTTP 12](#_Toc490453792)

[2.5 Web Service 13](#_Toc490453793)

[2.5.1 RESTful 15](#_Toc490453794)

[2.6 JSON 18](#_Toc490453795)

[2.7 MongoDB 18](#_Toc490453796)

[2.7.1 Dokumen 19](#_Toc490453797)

[2.7.2 Koleksi 19](#_Toc490453798)

[2.7.3 Tipe Data 20](#_Toc490453799)

[2.7.4 Reference 21](#_Toc490453800)

[2.8 JSON Web Token 22](#_Toc490453801)

[2.8.1 Struktur 22](#_Toc490453802)

[2.8.2 Otentikasi dan Otorisasi 23](#_Toc490453803)

[BAB 3 METODOLOGI 25](#_Toc490453804)

[3.1 Jenis Penelitian 25](#_Toc490453805)

[3.2 Metodologi Penelitian 25](#_Toc490453806)

[3.2.1 Studi Literatur 25](#_Toc490453807)

[3.2.2 Analisis Kebutuhan Sistem 26](#_Toc490453808)

[3.2.3 Perancangan Sistem 26](#_Toc490453809)

[3.2.4 Implementasi 27](#_Toc490453810)

[3.2.5 Pengujian dan Analisis Hasil Pengujian 28](#_Toc490453811)

[3.2.6 Kesimpulan dan Saran 29](#_Toc490453812)

[BAB 4 ANALISIS KEBUTUHAN DAN PERANCANGAN 30](#_Toc490453813)

[4.1 Deskripsi Umum Sistem 30](#_Toc490453814)

[4.2 Analisis Kebutuhan 32](#_Toc490453815)

[4.2.1 Kebutuhan Fungsional 32](#_Toc490453816)

[4.2.2 Kebutuhan Non-Fungsional 33](#_Toc490453817)

[4.3 Perancangan 34](#_Toc490453818)

[4.3.1 Perancangan Komponen Komunikasi 34](#_Toc490453819)

[4.3.2 Perancangan Komponen Manajemen Data 34](#_Toc490453820)

[4.3.3 Perancangan Komponen Manajemen Perangkat 44](#_Toc490453821)

[4.3.4 Perancangan Komponen Web Console 64](#_Toc490453822)

[BAB 5 IMPLEMENTASI 70](#_Toc490453823)

[5.1 Implementasi Komponen Komunikasi 70](#_Toc490453824)

[5.1.1 Instalasi Apache Web Server 70](#_Toc490453825)

[5.1.2 Instalasi Django Web Framework 70](#_Toc490453826)

[5.1.3 Integrasi Apache dan Django 71](#_Toc490453827)

[5.2 Implementasi Komponen Manajemen Data 72](#_Toc490453828)

[5.2.1 Instalasi MongoDB 72](#_Toc490453829)

[5.2.2 Implementasi Data Model 72](#_Toc490453830)

[5.2.3 Implementasi Data Access 76](#_Toc490453831)

[5.3 Implementasi Komponen Manajemen Perangkat 83](#_Toc490453832)

[5.3.1 Implementasi Otentikasi dan Otorisasi 83](#_Toc490453833)

[5.3.2 Implementasi Menejemen Perangkat 87](#_Toc490453834)

[5.4 Implementasi Komponen Web Console 92](#_Toc490453835)

[5.4.1 Instalasi Angular 93](#_Toc490453836)

[5.4.2 Integrasi Web Console dengan Web service 94](#_Toc490453837)

[BAB 6 PENGUJIAN DAN ANALISIS HASIL PENGUJIAN 109](#_Toc490453838)

[6.1 Perancangan Pengujian 109](#_Toc490453839)

[6.1.1 Perancangan Pengujian Fungsional 109](#_Toc490453840)

[6.1.2 Perancangan Pengujian Kehandalan Sistem 1](#_Toc490453841)

[6.2 Hasil dan Analisis Pengujian 2](#_Toc490453842)

[6.2.1 Hasil dan Analisis Pengujian Fungsional 2](#_Toc490453843)

[6.2.2 Hasil dan Analisis Pengujian Kehandalan Sistem 5](#_Toc490453844)

[BAB 7 KESIMPULAN DAN SARAN 10](#_Toc490453845)

[7.1 Kesimpulan 10](#_Toc490453846)

[7.2 Saran 10](#_Toc490453847)

[DAFTAR PUSTAKA 12](#_Toc490453848)

DAFTAR TABEL

[Tabel 2.1 Metode HTTP umum 9](#_Toc490453877)

[Tabel 2.2 Kode status HTTP yang digunakan 13](#_Toc490453878)

[Tabel 2.3 RESTful: empat tindakan terhadap data 15](#_Toc490453879)

[Tabel 4.1 Kebutuhan Fungsional Sistem 32](#_Toc490453880)

[Tabel 4.2 Kebutuhan Perangkat Keras 33](#_Toc490453881)

[Tabel 4.3 Kebutuhan Perangkat Lunak 33](#_Toc490453882)

[Tabel 4.4 Dokumen Users 36](#_Toc490453883)

[Tabel 4.5 Dokumen Sensors 36](#_Toc490453884)

[Tabel 4.6 Dokumen Nodes 36](#_Toc490453885)

[Tabel 4.7 Dokumen Sensordatas 37](#_Toc490453886)

[Tabel 4.8 Akses Data Sensor Berdasarkan Pengguna 38](#_Toc490453887)

[Tabel 4.9 Akses Data Sensor Berdasarkan Perangkat Node 40](#_Toc490453888)

[Tabel 4.10 Akses Data Sensor Berdasarkan Sensor 42](#_Toc490453889)

[Tabel 4.11 Perangkat Node Mendapatkan Token Akses JWT 45](#_Toc490453890)

[Tabel 4.12 Aplikasi Client Mendapatkan Token Akses JWT 46](#_Toc490453891)

[Tabel 4.13 Registrasi Pengguna 48](#_Toc490453892)

[Tabel 4.14 Menerima Data Sensor dari Perangkat Node 50](#_Toc490453893)

[Tabel 4.15 Membuat Perangkat Node 54](#_Toc490453894)

[Tabel 4.16 Melihat Perangkat Node 56](#_Toc490453895)

[Tabel 4.17 Melihat Perangkat Node Berdasarkan Id 58](#_Toc490453896)

[Tabel 4.18 Mengubah Perangkat Node 60](#_Toc490453897)

[Tabel 4.19 Menghapus Perangkat Node 62](#_Toc490453898)

[Tabel 6.1 Skenario Pengujian Fungsional 110](#_Toc490453899)

[Tabel 6.2 Hasil Pengujian Fungsional 3](#_Toc490453900)

[Tabel 6.3 Hasil Pengujian Performa Dalam Mengakses Data Sensor Berdasarkan Pengguna 5](#_Toc490453901)

[Tabel 6.4 Hasil Pengujian Performa Dalam Mengakses Data Sensor Berdasarkan Perangkat Node 5](#_Toc490453902)

[Tabel 6.5 Hasil Pengujian Performa Dalam Mengakses Data Sensor Berdasarkan Sensor 5](#_Toc490453903)

[Tabel 6.6 Hasil Pengujian Performa Dalam Mengirimkan Data Sensor 7](#_Toc490453904)

DAFTAR GAMBAR

[Gambar 2.1 Komponen Internet of Things 7](#_Toc490453905)

[Gambar 2.2 Arsitektur Cloud 8](#_Toc490453906)

[Gambar 2.3 Mekanisme HTTP request-response 9](#_Toc490453907)

[Gambar 2.4 Web Service menyediakan lapisan abstraksi 14](#_Toc490453908)

[Gambar 2.5 Komponen utama Web service 15](#_Toc490453909)

[Gambar 2.6 Contoh dokumen JSON 18](#_Toc490453910)

[Gambar 2.7 MongoDB: dokumen 19](#_Toc490453911)

[Gambar 2.8 MongoDB: koleksi dokumen 20](#_Toc490453912)

[Gambar 2.9 MongoDB: reference 22](#_Toc490453913)

[Gambar 2.10 Contoh JSON Web Token 23](#_Toc490453914)

[Gambar 2.11 Contoh Otorisasi Menggunakan JSON Web Token 24](#_Toc490453915)

[Gambar 3.1 Flowchart tahapan penelitian 25](#_Toc490453916)

[Gambar 3.2 Tahap Perancangan 26](#_Toc490453917)

[Gambar 4.1 Komponen Sistem 32](#_Toc490453918)

[Gambar 4.2 Alur Komunikasi Antar Entitas 34](#_Toc490453919)

[Gambar 4.3 Skema Basis Data 35](#_Toc490453920)

[Gambar 4.4 Alur Mengakses Data Sensor Berdasarkan Pengguna 39](#_Toc490453921)

[Gambar 4.5 Alur Mengakses Data Sensor Berdasarkan Perangkat Node 41](#_Toc490453922)

[Gambar 4.6 Alur Mengakses Data Sensor Berdasarkan Sensor 43](#_Toc490453923)

[Gambar 4.7 Alur Perangkat Node Mendapatkan Token Akses JWT 45](#_Toc490453924)

[Gambar 4.8 Alur Aplikasi Client Mendapatkan Token Akses JWT 47](#_Toc490453925)

[Gambar 4.9 Alur Registrasi Pengguna 49](#_Toc490453926)

[Gambar 4.10 Alur Menerima Data Sensor dari Perangkat Node 51](#_Toc490453927)

[Gambar 4.11 Flowchart Mengatur Ulang Bilangan Counter Pembatasan Pengiriman Semua Perangkat Node 53](#_Toc490453928)

[Gambar 4.12 Alur Membuat Perangkat Node 55](#_Toc490453929)

[Gambar 4.13 Alur melihat perangkat node 57](#_Toc490453930)

[Gambar 4.14 Alur melihat perangkat node berdasarkan id 59](#_Toc490453931)

[Gambar 4.15 Alur Mengubah Perangkat Node 61](#_Toc490453932)

[Gambar 4.16 Alur Menghapus Perangkat Node 63](#_Toc490453933)

[Gambar 4.17 Perancangan Web Console: Lihat Semua Perangkat Node 64](#_Toc490453934)

[Gambar 4.18 Perancangan Web Console: Lihat Perangkat Node 65](#_Toc490453935)

[Gambar 4.19 Perancangan Web Console: Buat Perangkat Sensor 66](#_Toc490453936)

[Gambar 4.20 Perancangan Web Console: Ubah Perangkat Node 67](#_Toc490453937)

[Gambar 4.21 Perancangan Web Console: Lihat Data Sensor 68](#_Toc490453938)

[Gambar 5.1 Konfigurasi Apache 72](#_Toc490453939)

[Gambar 5.2 Implementasi Aplikasi Web Console: Lihat Semua Perangkat Node 95](#_Toc490453940)

[Gambar 5.3 Implementasi Aplikasi Web Console: Lihat Perangkat Node 97](#_Toc490453941)

[Gambar 5.4 Implementasi Aplikasi Web Console: Buat Perangkat Node 99](#_Toc490453942)

[Gambar 5.5 Implementasi Aplikasi Web Console: Ubah Perangkat Node 102](#_Toc490453943)

[Gambar 5.6 Implementasi Aplikasi Web Console: Lihat Data Sensor 106](#_Toc490453944)

[Gambar 6.1 Perancangan Pengujian Kehandalan Sistem 1](#_Toc490453945)

[Gambar 6.2 Grafik Perbandingan Nilai Throughput Pada Fitur Mengakses Data Sensor 6](#_Toc490453946)

[Gambar 6.3 Grafik Perbandingan Nilai Latency Pada Fitur Mengakses Data Sensor 7](#_Toc490453947)

[Gambar 6.4 Grafik Perbandingan Nilai Throughput Pada Fitur Mengakses Data Sensor 8](#_Toc490453948)

[Gambar 6.5 Grafik Perbandingan Nilai Latency Pada Fitur Mengakses Data Sensor 9](#_Toc490453949)

DAFTAR LAMPIRAN

[LAMPIRAN A PERSYARATAN FISIK DAN TATA LETAK **Error! Bookmark not defined.**](#_Toc428800332)

[A.1 Kertas **Error! Bookmark not defined.**](#_Toc428800333)

[A.2 Margin **Error! Bookmark not defined.**](#_Toc428800334)

[A.3 Jenis dan ukuran huruf **Error! Bookmark not defined.**](#_Toc428800335)

[A.4 Spasi **Error! Bookmark not defined.**](#_Toc428800336)

[A.5 Kepala bab, sub bab, dan seksi **Error! Bookmark not defined.**](#_Toc428800337)

[A.6 Nomor halaman **Error! Bookmark not defined.**](#_Toc428800338)

[LAMPIRAN B PENGGUNAAN BAHASA **Error! Bookmark not defined.**](#_Toc428800339)

# PENDAHULUAN

## Latar belakang

*Internet of Things* (IoT) merujuk pada konsep dimana perangkat fisik dapat saling terhubung di berbagai jaringan menggunakan berbagai protokol berbeda (Guoqiang et. al., 2013). Konsep ini bertujuan untuk memperluas manfaat dari konektivitas internet yang terhubung secara terus-menerus. IoT secara khusus dapat dibagi ke dalam 6 komponen berbeda: *identification, sensing*, *communication*, *computation*, *services* dan *semantics* (Al-Fuqaha, et. al., 2015). Komponen-komponen tersebut membuat perangkat dalam IoT dapat diidentifikasi secara unik, mengumpulkan dan mengirimkan data, serta mengakses suatu layanan yang relevan melalui internet. Konsep IoT dapat diaplikasikan ke dalam berbagai aktivitas di segala bidang sehingga pertukaran informasi melalui internet menjadi lebih mudah dan efisien. Salah satu penerapannya digunakan dalam sistem untuk mengambil data dari berbagai perangkat yang tersebar di suatu tempat. Sistem ini memungkinkan seorang petani dan peneliti di bidang *agriculture* untuk mengakses informasi terkait kondisi suhu dan kelembaban di lahan yang mereka punya melalui internet.

*Internet of Things* secara umum merepresentasikan perangkat-perangkat kecil yang di distribusikan secara luas dan memiliki keterbatasan dalam hal kapasitas penyimpanan dan kemampuan komputasi. Hal tersebut menciptakan kekhawatiran dalam hal keandalan, kinerja, keamanan, dan privasi pada layanan yang akan dibangun (Botta, et. al., 2016). Untuk mengatasi kekhawatiran tersebut, perangkat IoT dapat diintegrasikan dengan sistem lain yang menawarkan spesifikasi perangkat keras yang lebih baik, salah satunya *cloud computing*. Menurut Zhang, et., all (2010) *cloud computing* merupakan model komputasi baru dimana sumber daya komputasi dapat dikonfigurasi sesuai kebutuhan pengguna dengan mudah melalui internet. Integrasi antara IoT dan *cloud computing* menciptakan paradigma teknologi baru yang disebut CloudIoT untuk internet dimasa depan (Botta, et. al., 2016).

Salah satu contoh penerapan CloudIoT digunakan dalam sistem untuk mengambil data dari perangkat node di lapangan. Pada sistem tersebut, data sensor sering kali diambil secara terus menerus, disisi lain perangkat node memiliki keterbatasan dalam hal spesifikasi perangkat keras. Hal tersebut membuat penyimpanan dan pengolahan data yang besar tidak bisa dilakukan di perangkat node dan harus dilakukan oleh *cloud*. Selain itu, dengan mengirimkan data sensor ke *cloud* membuatnya dapat diakses secara luas tanpa harus berada di tempat tersebut (Botta, et. al., 2016). Ini menjadikan sumber penyimpanan dan *cloud computing* menghadirkan pilihan terbaik bagi IoT untuk menyimpan dan memproses data dalam jumlah besar.

Diantara berbagai keuntungan dari integrasi antara IoT dan *cloud computing*,masih terdapat tantangan seperti yang dibahas dalam penelitian Botta, et. al. (2016). Tantangan tersebut meliputi masalah *network communication* dan *security and privacy*. Masalah *network communication* terjadi karena CloudIoTmelibatkan banyak teknologi jaringan, sehingga pemilihan protokol komunikasi harus mempertimbangkan dukungan oleh banyak perangkat. Sedangkan masalah s*ecurity and privacy* terjadi karena banyaknya perangkat di luar sana yang dapat dengan mudah terhubung dengan CloudIoT, sehingga peran otentikasi dan otorisasi diperlukan untuk mengidentifikasi dan memvalidasi perangkat yang mengirimkan data. Masalah selanjutnya menurut Al-Fuqaha, et. al., (2015) adalah dengan banyaknya perangkat IoT yang digunakan membutuhkan mekanisme untuk menajemen perangkat (Al-Fuqaha, et. al., 2015).

Sebagai salah satu solusi dari permasalahan dalam mengintegrasikan IoT dan *cloud computing* diatas, penelitian ini membuat rancang bangun IoT *cloud platform* yangmemungkinkan perangkat IoT terhubung dan berinteraksi dengan aplikasi *cloud* dengan aman. IoT *cloud platform* merupakan aplikasi yang menawarkan layanan untuk manajemen keamanan, manajemen perangkat, manajemen *policy*, dan lain-lain (Singh dan Viniotis, 2016). Untuk menyelesaikan masalah *network communication*, pada IoT *cloud platform* inidigunakan protokol komunikasi HTTP. Protokol ini dipilih karena merupakan protokol sederhana yang sejak era awal perkembangan internet telah banyak digunakan, sehingga didukung oleh banyak perangkat. Sedangkan untuk menyelesaikan masalah s*ecurity and privacy*, digunakan mekanisme otentikasi dan otorisasi menggunakan JWT (JSON Web Token).

Sistem ini secara garis besar dapat dibagi ke dalam 4 komponen, yaitu: komponen komunikasi, komponen *security*, komponen manajemen data dan komponen *web console*. Komponen komunikasi merupakan RESTful *web service* yang menangani komunikasi dari perangkat node melalui protokol HTTP. Komponen *security* merupakan bagian yang menangani manajemen perangkat, otentikasi dan otorisasi yang berbasis JSON Web Token. Komponen manajemen data menangani mekanisme penyimpanan datadan mekanisme mengakses data sensor melalui RESTful *web service*. Sedangakan komponen *web console* merupakan *single-page application* yang menyediakan antarmuka pengguna untuk melihat data sensor dan memanajemen perangkat.

## Rumusan masalah

Berdasarkan latar belakang diatas, diperoleh rumusan masalah yang menjadi pedoman dalam penelitian ini adalah sebagai berikut:

1. Bagaimana menerapakan protokol HTTP untuk komunikasi pada IoT *cloud platform*?
2. Bagaimana mekanisme pengiriman data antara perangkat node, IoT *cloud platform*, dan aplikasi *client*?
3. Bagaimana mekanisme manajemen perangkat, mengirimkan dan mengakses data sensor pada IoT *cloud platform* yang dibangun?
4. Bagaimana kehandalan dari IoT *cloud platform* yang dibangun?

## Tujuan

Tujuan yang ingin dicapai dalam penelitian ini berdasarkan rumusan masalah di atas adalah sebagai berikut:

1. Merancang dan mengimplementasikan IoT *cloud platform* yang memiliki fitur manajemen perangkat, mengidentifikasi dan memvalidasi perangkat yang mengirimkan data sensor, dan mengkakses data sensor secara luas melalui internet.
2. Menerapkan RESTful *web service* yang menyediakan antarmuka bagi perangkat node dan *cloud* untuk saling terhubung menggunakan protokol komunikasi HTTP.
3. Mengetahui kehandalan dari IoT *cloud* platform yang dibangun.

## Manfaat

Penelitian ini diharapkan dapat memberikan manfaat baik untuk penulis, penelitian selanjutnya, maupun umum. Manfaat yang diperoleh penulis adalah penulis mendapat pengetahuan mengenai protokol komunikasi HTTP dalam proses komunikasi serta mendapat pengetahuan terkait integrasi antara perangkat IoT dan *cloud* secara aman dan efisien. Untuk penelitian selanjutnya dapat menambah referensi penelitian di bidang jaringan khusunya mengenai protokol HTTP, konsep Internet of Things, RESTFul *web service* dan IoT *cloud platform*. Sedangkan untuk umum menyediakan sebuah IoT *cloud platform* berbasis RESTFul *web service* yang dapat digunakan untuk mengumpulkan data dari perangkat IoT di suatu tempat dan mengksesnya secara luas melalui internet.

## Batasan masalah

Agar pembahasan tidak melebar, diperlukan batasan-batasan untuk mempersempit lingkup penelitian. Batasan masalah pada penelitian ini adalah sebagai berikut:

Perangkat node dapat langsung mengirim data ke IoT *cloud platfotm* tanpa melewati *relay* terlebih dahulu. Untuk penambahan *relay*, nantinya dapat dikembangkan menjadi topik penelitian tersendiri.

Format data dari atau menuju IoT *cloud platform* menggunakan tipe JSON.

Otentikasi dan otorisasi menggunakan JWT (JSON Web Token).

Penyimpanan token di dalam kukis browser pada aplikasi *web console* tidak menerapkan suatu proses pengamanan tertentu, misalnya enkripsi.

Data sensor yang disimpan di *cloud* merupakan data mentah yang tidak dianalisa dan divisualisasikan dalam bentuk tertentu.

## Sistematika penulisan

Sistematika penulisan menjelaskan mengenai struktur penulisan dari penelitian ini yang dapat dibagi ke dalam 7 bab sebagai berikut:

|  |  |  |
| --- | --- | --- |
| **BAB 1** | **:** | **Pendahuluan**  Menguraikan masalah yang diangkat secara umum dan menjelaskan mengenai struktur penulisan yang digunakan dalam penelitian. Bab ini terdiri dari: latar belakang, rumusan masalah, tujuan, manfaat, batasan masalah dan sistematika penulisan. |
| **BAB 2** | **:** | **Landasan Kepustakaan**  Membahas kajian pustaka dan penelitian sebelumnya yang berhubungan dengan rancang bangun IoT *cloud platform*. Kajian pustaka ini berasal dari referensi-referensi berkaitan dan mendukung dalam penelitian ini. |
| **BAB 3** | **:** | **Metodologi Penelitian**  Menguraikan tahapan-tahapan secara sistematik dalam rancang bangun perangkat lunak IoT *cloud platform* berbasis protokol komunikasi HTTP. Tahapan-tahapan yang dijelaskan pada bab metodologi penelitian diantaranya: studi literatur, analisis kebutuhan dan perancangan, implementasi, pengujian dan analisis hasil pengujian, serta kesimpulan dan saran. |
| **BAB 4** | **:** | **Analisis Kebutuhan dan Perancangan**  Membahas tentang kebutuhan dan fitur apa saja yang dibutuhkan dari perangkat lunak IoT *cloud platform* yang dibangun. Berdasarkan kebutuhan dan fitur tersebut kemudian dibuat rancangan arsitektur serta dijelaskan bagaimana proses alur data dari perangkat node ke *cloud* dan sebaliknya. |
| **BAB 5** | **:** | **Implementasi**  Menjelaskan proses implementasi sistem berdasarkan rancangan pada bab sebelumnya dengan menyertakan potongan kode proses-proses utama dalam perangkat lunak IoT *cloud gateway*. Bab ini juga membahas tentang persiapan lingkungan sistem yang dibutuhkan untuk menjalankan perangkat lunak seperti instalasi dan konfigurasi web server, sistem basis data, dll. |
| **BAB 6** | **:** | **Pengujian dan Analisis Hasil Pengujian**  Membahas tentang tahapan pengujian terhadap perangkat lunak IoT *cloud platform* yang dibangun. Bab ini juga membahas mengenai skenario pengujian yang digunakan dalam tahap pengujian. Pengujian dilakukan berdasarkan scenario uji untuk memastikan kebutuhan telah terpenuhi dan sesuai. |
| **BAB 7** | **:** | **Penutup**  Memuat kesimpulan yang diperoleh dari penelitian ini berdasarkan rumusan masalah sebelumnya. Pada bab ini juga disertakan saran yang dapat digunakan untuk penelitian-penelitian selanjutnya. |

# LANDASAN KEPUSTAKAAN

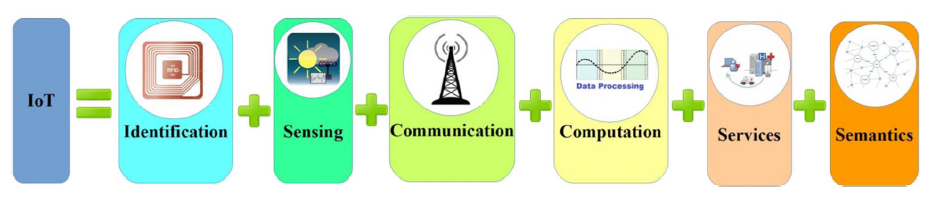
Bab ini berisi ulasan mengenai kajian dan dasar teori yang berkaitan dengan penelitian ini. Kajian pustaka membahas mengenai penelitian-penelitian terdahulu yang terkait dengan penelitian yang diusulkan. Dasar teori berisi ulasan mengenai teori berkaitan dengan teknologi yang digunakan, diantaranya: HTTP, *webservice*, sistem basis data MongoDB, JSON Web Token, serta dasar teori lainnya yang mendukung.

## Kajian Pustaka

Penelitain terkait dengan judul ***Modular and generic IoT management on the cloud*** (Douzis, et. al., 2016) mengajukan perangkat lunak untuk memanajemen perangkat IoT yang dibangun diatas lingkungan *sandbox* Fiware lab. Fiware lab merupakan *platform* yang berjalan diatas OpenStack. Hal tersebut menjadikan perangkat lunak tidak dapat dijalankan pada *server cloud* non-OpenStack. Penelitian lainnya yang dilakukan oleh Gangluy (2016) dengan judul ***Selecting the right IoT Cloud Platform*** membahas fitur-fitur dari beberapa IoT *cloud platform* populer diantaranya: Kaa, DeviceHive, OpenIoT dan ThingSpeak. Dari beberapa IoT *cloud platform* yang dibahas dalam penelitian tersebut, tidak ditemukan atau patut diduga tidak tersedia fitur untuk membatasi suatu perangkat dalam mengirimkan data sensor setiap harinya. Fitur pembatasan pengiriman tersebut dapat mencegah dikimkannya data sensor dari suatu perangkat node secara terus-menerus setiap harinya, sehingga dapat menekan penggunaan sumber daya penyimpanan di *cloud* untuk perangkat node yang diinginkan. Fitur lain yang tidak ditemukan atau patut diduga tidak tersedia yaitu fitur visibilitas perangkat node yang memungkinkan pengguna dapat mengakses data sensor dari perangkat node milik pengguna lainnya. Fitur tersebut sangat berguna jika beberapa pengguna sistem ingin saling berbagi data sensor dari perangkat yang mereka miliki.

## Internet of Things

Menurut Guoqiang et. al. (2013), *Internet of Things* (IoT) merujuk pada suatu konsep jaringan yang menghubungkan hal-hal yang dapat diidentifikasi dalam dunia fisik di berbagai jaringan dengan berbagai protokol berbeda. Konsep ini bertujuan untuk memperluas manfaat dari konektivitas internet yang terhubung secara terus-menerus. IoT membuat perangkat di dunia fisik dapat diidentifikasi secara unik, mengumpulkan dan mengirimkan data, serta mengakses suatu layanan yang relevan melalui internet. Konsep IoT dapat diaplikasikan kedalam berbagai aktivitas disegala bidang sehingga pertukaran informasi melalui internet menjadi lebih mudah dan efisien. IoT secara khusus dapat dibagi ke dalam 6 komponen berbeda yaitu *identification, sensing*, *communication*, *computation*, *services* dan *semantics* (Al-Fuqaha, et. al., 2015). Komponen-komponen tersebut dapat dilihat pada Gambar 2.1.



Gambar 2.1 Komponen Internet of Things

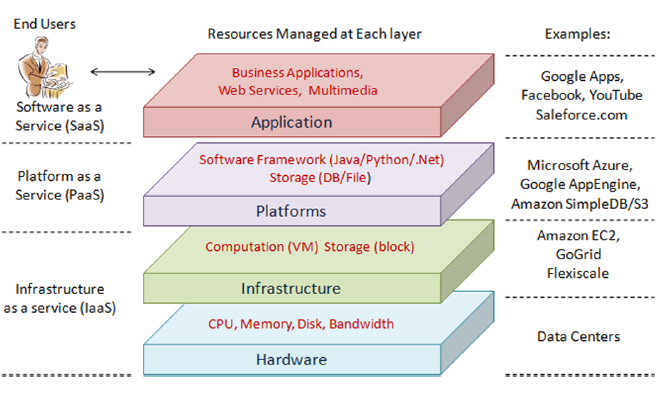
Sumber: Al-Fuqaha, et. al. (2015)

Komponen *identification* memungkinkan *things* dapat di identifikasi berdasarkan *id* objek dan pengalamatannya di dalam jaringan komunikasi. Komponen *sensing* bertugas mengumpulkan data dari objek IoT *sensor* dan mengirimkannya kembali ke *warehouse*, *database*, atau *cloud*. Data objek IoT *sensor* tersebut dikirimkan oleh komponen *communication* melalui teknologi seperti WiFi, Bluetooth, IEEE 802.15.4, Z-wave, dan LTE-Advanced. Kemampuan komunikasi dari IoT didukung oleh komponen *computation* berupa perangkat keras dan perangkat lunak yang relevan. Sedangkan, komponen *services* menyediakan layanan tertentu untuk IoT dan elemen *semantic* bertugas menggali informasi dengan cerdas untuk menyediakan layanan yang dibutuhkan (Al-Fuqaha, et. al., 2015).

Perangkat-perangkat dalam IoT umumnya merupakan perangkat kecil yang di distribusikan secara luas dan memiliki keterbatasan dalam hal kapasitas penyimpanan dan kemampuan komputasi. Hal tersebut menciptakan kekhawatiran dalam hal keandalan, kinerja, keamanan, dan privasi pada layanan yang akan dibangun (Botta, et. al., 2016). Untuk mengatasi kekhawatiran tersebut, perangkat IoT dapat diintegrasikan dengan sistem lain yang menawarkan spesifikasi perangkat keras yang lebih baik, salah satunya *cloud computing*. Integrasi antara IoT dan *cloud computing* menciptakan paradigma teknologi baru yang disebut CloudIoT untuk internet dimasa depan (Botta, et. al., 2016).

## Cloud

Menurut Zhang, et., all (2010) *cloud computing* merupakan model komputasi baru dimana sumberdaya komputasi dapat dikonfigurasi sesuai kebutuhan pengguna dengan mudah melalui internet. Arsitektur *cloud computing* dapat dibagi ke dalam empat lapisan, yaitu: *hardware, infrastructure, platform,* dan *application*. Lapisan *hardware* merupakan sumber daya fisik dari *cloud* yang meliputi server fisik, router, switch, dan sebagainya. Sumber tersebut divirtualisasikan oleh lapisan *infrastructure* untuk menciptakan kumpulan media penyimpanan dan komputasi virtual. Lapisan *platform* berisi sistem operasi dan *framework* untuk aplikasi diatasnya. Sedangkan lapisan *application* merupakan hierarki tertinggi untuk aplikasi *cloud*, misalnya multimedia dan *web service*.



Gambar 2.2 Arsitektur *Cloud*

Sumber: Zhang, et., all (2010)

## HTTP

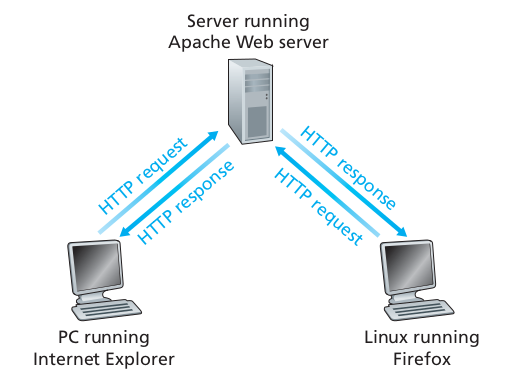
Hypertext Transfer Protocol (HTTP) merupakan sebuah protokol jaringan di layer aplikasi untuk mendistribusikan [informasi](http://id.wikipedia.org/wiki/Sistem_terdistribusi) dengan banyak tipe datasecara kolaboratif (IETF, 1999). HTTP dikembangkan untuk proyek World Wide Web (WWW) oleh Tim Berners-Lee beserta koleganya pada tahun 1990. WWW menyediakan mekanisme untuk mengambil sumber informasi yang saling terhubung dengan tautan menggunakan [Uniform Resource Identifier](https://id.wikipedia.org/wiki/Uniform_Resource_Identifier) (URI). URI dapat dianalogikan seperti alamat pos di Internet, ia memberikan identifikasi secara unik terhadap sumber informasi di seluruh dunia (Gourley, et. al., 2002).

HTTP diimplementasikan ke dalam dua program, yaitu program *client* dan program *server*. Kedua program tersebut, dijalankan pada *end system* yang berbeda dan berkomunikasi satu sama lain dengan bertukar pesan HTTP (Kurose dan Ross, 2013:98). Program *client* dapat berupa [*web browser*](https://id.wikipedia.org/wiki/Web_browser), sistem dan lain sebagainya yang mengirimkan pesan HTTP *request*. Program *server* atau disebut juga sebagai *web server*, bertugas menyimpan sumber daya tertentu, memproses pesan HTTP *request* yang dikirimkan *client* dan mengembalikan *request* yang diminta melalui pesan HTTP *response*. Protokol HTTP mendefiniskan bagaimana program *client* meminta sumber daya dari *server* dan bagaimana *server* mengirimkan sumber daya yang diminta tersebut kepada *client* (Kurose dan Ross, 2013:99).

### Mekanisme Kerja

Mekanisme kerja dari HTTP menggunakan kaidah *request*-*response* antara *client* dan *server*. *Client* mengirimkan pesan HTTP *request* kemudian server mengirimkan pesan HTTP *response* sesuai dengan format yang telah ditentukan *client*.  *Server* tidakakan mengirimkan sebuah informasi jika *client* tidak melakukan permintaan terlebih dahulu (Kurose dan Ross, 2013:100). Mekansime kerja dari HTTP dapat dilihat pada Gambar 2.3.

HTTP berjalan diatas protokol TCP yang *reliable* atau dapat diandalkan. TCP yang merupakan protokol di lapisan *transport* memastikan data yang dikirim oleh protokol HTTP sampai ke tujuan. Hal ini menyiratkan bahwa setiap pesan HTTP *request* dari *client* akan sampai secara utuh ke *server;* begitu juga sebaliknya,setiap pesan HTTP *response* yang dikirim oleh *server* akan sampai secara utuh ke *client* (Kurose dan Ross, 2013:100).



Gambar 2.3 MekanismeHTTP request-response

Sumber: Kurose dan Ross (2013:100)

### Metode HTTP

HTTP mendukung beberapa metode *request* untuk membedakan aksi yang akan dijalankan oleh *server* nantinya. Terdapat tujuh metode HTTP yang umum digunakan. Beberapa diantaranya memiliki badan pesan pada pesan HTTP *request*-nya, dan sebagian lagi tidak. Badan pesan digunakan untuk mengirimkan data teks kepada *server* untuk diproses atau disimpan, seperti yang dimiliki metode POST dan PUT (Gourley, et. al., 2002).

Tabel 2.1 Metode HTTP umum

|  |  |  |
| --- | --- | --- |
| Metode | Deskripsi | Badan pesan? |
| GET | Mendapatkan dokumen dari server. | Tidak |
| HEAD | Mendapatkan *header* untuk sebuah dokumen dari server. | Tidak |
| POST | Mengirimkan data ke server untuk diproses. | Ya |
| PUT | Menyimpan data di badan pesan pada server. | Ya |
| TRACE | Menelusuri pesan melalui server proxy ke server. | Tidak |
| OPTONS | Mendapatkan informasi mengenai metode apa saja yang bisa beroperasi pada server. | Tidak |
| DELETE | Menghapus dokumen dari server. | Tidak |

Sumber: Gourley, et. al., (2002:48)

### Header HTTP

Menurut Richardson, Leonard, et al (2013), *header* HTTP merupakan serangkaian bit dari *metadata* yang mendeskripsikan semantik protokol dari pesan HTTP *request* atau pesan HTTP *response. Header* HTTP memungkinkan *client* dan *server* bertukar informasi tambahan untuk menentukan apa yang akan *client* dan *server* lakukan (Mozilla, 2017). Beberapa *header* digunakan secara spesifik untuk setiap jenis pesan dan beberapa digunakan untuk tujuan yang lebih umum, keduanya memiliki tujuan yang sama dalam hal menyediakan informasi pada pesan *request* dan *response*. *Header* HTTP dapat dikelompokkan ke dalam lima kelompok utama berdasarkan fungsinya (Gourley, et. al., 2002:67) :

* 1. *General headers*

*General headers* merupakan *header* generik yang menawarkan informasi yang sangat mendasar tentang sebuah pesan. *Header* dalam kelas ini sama-sama digunakan oleh *client* dan *server* untuk tujuan umum dalam berkomunikasi satu sama lain. Salah satu contohnya adalah *header Date* yang mengindikasikan waktu dan tanggal pada saat pesan dibuat:

Date: Tue, 3 Oct 1974 02:16:00 GMT

* 1. [*Request header*](https://developer.mozilla.org/en-US/docs/Glossary/Request_header)*s*

*Request headers* secara spesifik hanya digunakan pada pesan *request* yang dikirimkan *client*. *Header* ini berisi beberapa informasi tambahan kepada *server* mengenai tipe dari sumber daya yang diminta. Salah satu contohnya adalah *header Accept* yang memberitahukan *server* bahwa *client* menerima tipe media apapun yang cocok dengan *request* tersebut:

Accept: \*/\*

* 1. *Response headers*

*Response headers* secara spesifik hanya digunakan pada pesan *response* yang dikirimkan oleh *server*. *Header* ini berisi informasi mengenai *server* untuk *client*, misalnya *server* dengan tipe apa yang berkomunikasi dengan *client* tersebut. Salah satu contohnya *header Server* memberitahu *client* untuk berkomunikasi menggunakan Tiki-Hut server versi 1.0:

Server: Tiki-Hut/1.0

* 1. [*Entity header*](https://developer.mozilla.org/en-US/docs/Glossary/Entity_header)

*Entity headers* berisi informasi tambahan yang berkaitan dengan entitas yang dibawa dalam badan pesan HTTP. Sebagai contoh, *header* *Content-Type* dan *header charset* dapat memberi tahu informasi dari entitas terkait dengan tipe data dan metode pengkodeannya:

Content-Type: text/html; charset=iso-latin-1

* 1. *Extension headers*

*Extension headers* merupakan *header* non-standar yang dibuat oleh pengembang aplikasi. Hal tersebut dimungkinkan untuk tujuan tertentu, misalnya menerapkan sekuritas tambahan yang tidak ada dalam spesifikasi protokol HTTP. Program HTTP yang dibuat, secara spesifik perlu mentolerir dan meneruskan *header* ekstensi, bahkan jika mereka tidak tahu apa arti dari *header* tersebut. Sebagai contoh*, header X-XSS-Protection* dan *header mode* untuk memberi tahu diaktifkannya *Cross-site scripting (XSS) filter*:

X-XSS-Protection: 1; mode=block

Beberapa *header* standar HTTP yang digunakan dalam penelitian:

* 1. *Allow*

Merupakan *header* yang termasuk ke dalam kelompok *response headers*. *Header* ini digunakan untuk memberi tahu *client* mengenaimetode HTTP apa saja yang akan ditanggapi *server* pada suatu lokasi sumber daya. Sebagai contoh suatu lokasi sumber daya hanya mendukung metode GET dan HEAD:

Allow: GET,HEAD

* 1. *Authorization*

Merupakan *header* yang termasuk ke dalam *request headers*. *Header* ini dikirimkan oleh *client* untuk tujan otorisasi dengan skema yang disepakati. Skema apa saja dapat digunakan selama *client* dan *server* mengetahui itu. Meski begitu skema umum yang paling banyak digunakan adalah OAuth dan HTTP Basic. Sintaks dasar dari *header* ini adalah:

Authorization: authentication-scheme

#authentication-param

Contoh:

Authorization: Basic YnJpYW4tdG90dHk6T3ch

* 1. *Content****-****Type*

Merupakan *header* yang termasuk ke dalam kelompok *entity headers*. *Header* ini digunakan oleh *client* dan *server* untuk menentukan tipe data dan metode pengkodean yang digunakan di dalam pesan. Sebagai contoh *client* dan *server* menggunakan tipe konten text/html dengan set karakter iso-latin-1:

Content-Type: text/html; charset=iso-latin-1

* 1. *WWW****-****Authenticate*

Merupakan *header* yang termasuk ke dalam kelompok *response headers*. *Header* ini digunakan pada pesan HTTP *response* dengan status 401 *Unauthorized* yang menunjukkan penolakan skema otentikasi kepada *client*.

WWW-Authenticate: Basic realm="Your Private Travel Profile"

### Kode status HTTP

Kode status HTTP adalah bilangan tiga-digit yang dilampirkan pada setiap pesan HTTP *response* untuk memberi tahu *client* apa yang terjadi saat *server* mencoba menangani *request* tersebut. Dalam spesifikasi dari protokol HTTP didefinisikan lima kelompok berbeda berdasarkan kode status dengan awalan digit 1 sampai 5 (Richardson, Leonard, et. al., 2013):

* 1. 1xx*: Informational*

Kode status ini hanya digunakan oleh *client* dan server dalam proses negosiasi untuk menangani sambungan HTTP.

* 1. 2xx: *Successful*

Kode status ini menunjukkan bahwa operasi apapun yang diminta oleh *client* telah berhasil dilakukan.

* 1. 3xx: *Redirection*

Kode status ini menunjukkan bahwa operasi yang diminta oleh *client* belum berhasil dilakukan. Secara umum, *client* perlu membuat *request* baru dengan mengubah lokasi awal ke lokasi alternatif dari sumber daya yang diminta.

* 1. 4xx: *Client* *Error*

Kode status ini menunjukkan bahwa operasi yang diminta oleh *client* tidak berhasil dilakukan, karena terdapat masalah dengan *HTTP request-*nya. Masalah tersebut dapat berupa: otentikasi, format dari representasi sumber daya, waktu dari *request*, atau kesalahan dari HTTP *client* itu sendiri. Pada akhirnya, *client* harus memperbaiki sesuatu untuk membuat *request* selanjutnya berhasil dilakukan.

* 1. 5xx: *Server* Error

Kode status ini menunjukkan bahwa operasi yang diminta oleh *client* tidak berhasil dilakukan, karena terdapat masalah pada sisi *server*. Tidak ada yang dapat dilakukan *client* untuk membuat *request* selanjutnya berhasil dilakukan, sebelum masalah pada sisi *server* diperbaiki.

Tabel 2.2 menunjukkan beberapa kode status HTTP yang digunakan dalam penelitian:

Tabel 2.2 Kode status HTTP yang digunakan

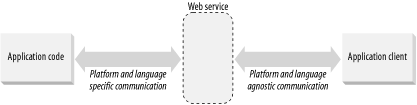
|  |  |  |
| --- | --- | --- |
| Kode status | Frase Sebab | Arti |
| 200 | OK | *Request* berhasil dilakukan. |
| 201 | Created | *Server* berhasil membuat objek baru atas permintaan *client* (mis., PUT). |
| 204 | No Content | Pesan *response* yang hanya berisi *header* dan kode status tanpa ada badan pesan. Biasanya digunakan untuk memperbarui *browser* tanpa berpindah ke laman baru (mis., Menyegarkan halaman formulir). |
| 301 | Moved Permanently | Lokasi sumber daya yang diminta telah dipindahkan. *Response* seharusnya berisi informasi mengenai lokasi saat ini dari sumber daya tersebut. |
| 400 | Bad Request | *Client* mengirimkan *request* yang salah. |
| 401 | Unauthorized | *Client* perlu melakukan otorisasi sebelum bisa mengakses sumber daya yang diminta. |
| 403 | Forbidden | *Request* ditolak oleh server karena alasan tertentu. *Server* bisa saja memberi tahu atau tidak mengenai alasan penolakannya tersebut. |
| 404 | Not Found | Sumber daya yang diminta tidak ada pada lokasi tersebut. |
| 500 | Internal  Server Error | Terdapat kesalahan di sisi *server*, sehingga mencegahnya untuk melayani request dari *client*. |

Sumber: Gourley, et. al., (2002:48)

## Web Service

*Web service* merupakan API yang berjalan pada aplikasi web. API (*Application Programming Interface)* merupakan kumpulan perintah, fungsi, protokol, dan objek yang dapat digunakan oleh *programmer* untuk membangun aplikasi atau berinteraksi dengan sistem dari luar. Selain web, API juga dapat berjalan di dalam sistem operasi, sistem basisdata, perangkat keras komputer atau pustaka dari aplikasi. API memberi *programmer* suatu perintah standar untuk melakukan operasi umum sehingga mereka tidak perlu menuliskan kode tersebut dari awal (Techterms, 2016).

Menurut Tidwell (2011), *web service* merupakan antarmuka untuk memanggil fungsi dari suatu aplikasi yang dibangun menggunakan standar teknologi internet. *Web service* berperan sebagai lapisan abstraksi yang memisahkan antara *platform* dan bahasa pemrograman yang digunakan--serta menyembunyikan rincian tentang bagaimana kode aplikasi sebenarnya dieksekusi. Lapisan standar ini membuat bahasa pemrograman apapun yang mendukung *web service* dapat mengakses fungsionalitas dari aplikasi tersebut.



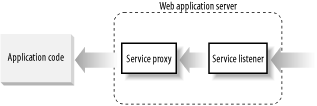
Gambar 2.4 Web Service menyediakan lapisan abstraksi

antara aplikasi dan kode

Sumber: Tidwell (2001)

Karena lapisan abstraksi yang disediakan merupakan antarmuka yang terstandar, menjadi tidak masalah jika sebuah servis aplikasi ditulis dalam bahasa pemrograman Java sedangkan *browser* ditulis dalam C++. Perbedaan *platform* juga menjadi tidak masalah, sebagai contoh sebuah servis aplikasi dibangun pada sistem Unix sementara *browser* dibangun pada sistem Windows. *Web service* memungkinkan antar *platform* dapat saling mengertisatu sama lain sehingga perbedaan *platform* terhadap masalah kompatibilitas menjadi tidak relevan (Tidwell, 2001).

*Web service* terdiri dari beberapa komponen utama, yaitu *Application code*, *Service proxy* dan *Service listener*. *Application code* bertugas untuk menangani semua logika bisnis dan kode yang memang melakukan sesuatu hal (melihat daftar buku, menambahkan buku ke dalam keranjang belanja, membayar buku, dll). *Service listener* bertugas untukmenangani *transport protocol* (HTTP, SOAP, Jabber, dll) dan menerima *request* yang masuk. *Service Proxy* bertugas untuk men-decode *request* yang masuk ke dalam pemanggilan kode aplikasi yang sesuai. *Service proxy* kemudian dapat meng-encode *response* untuk diteruskan untuk *Service listener*, namun langkah ini bersifat optional dan memungkinkan untuk dihilangkan (Tidwell, 2001). Komponen utama dari *web service* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Komponen utama *Web service*

Sumber: Tidwell (2001)

*Web service* berbeda dengan aplikasi web biasa seperti blog ataupun toko online. Salah satu perbedaan utamanya terletak dalam hal representasi data. *Web service* merepresentasikan data dalam format tertentu yang mudah di-decode oleh program, contohnya JSON dan XML. Sedangkan aplikasi web biasa merepresentasikan data ke dalam format HTML yang kemudian ditampilkan oleh *web browser* menjadi suatu laman web utuh. Bisa dikatakan bahwa, *web service* dibuat untukdigunakan oleh aplikasi lain, sedangkan aplikasi web biasa dibuat untuk digunakan langsung oleh pengguna.

### RESTful

REST (*REpresentational State Transfer*) bukanlah sebuah arsitektur melainkan sekumpulan batasan yang jika diterapkan dalam mendesain suatu sistem akan mencipatakan gaya arsitektur perangkat lunak. Batasan tersebut tidak mengatur teknologi apa yang digunakan, tetapi mendefinisikan pedoman bagaimana data ditransfer antar komponen. Pedoman tersebut menghasilkan abstraksi dari *resource*, representasi, URI, dan jenis HTTP *request* menjadi antarmuka seragam yang digunakan untuk *client* dan *server* bertukar data (Sandoval, 2009). Sistem yang mendefinisikan REST disebut dengan RESTful *system* dan *web service* yang mendefinisikan REST disebut dengan RESTful *web service*.

RESTful memodelkan transfer data antar komponen berbasis *resource*, dimana setiap *resource* dipetakan ke dalam URI ([Uniform Resource Identifier](https://id.wikipedia.org/wiki/Uniform_Resource_Identifier)) yang unik. Setiap *resource* dapat direpresentasikan dalam XML, objek JSON, CSV bergantung pada apa yang diminta oleh *client* pada pesan HTTP nya(Richardson, 2013). RESTful mengubah keadaan *resource* melalui empat tindakan spesifik yang dapat dilakukan, yaitu *Create, Retrieve, Update,* dan *Delete (CRUD)*. Tindakan terhadap *resource* dapat dipetakan ke dalam metode HTTP POST, GET, PUT, dan DELETE sebagai berikut (Sandoval, 2009) :

Tabel 2.3 RESTful: empat tindakan terhadap data

|  |  |
| --- | --- |
| Tindakan Data | Equivalen dengan HTTP method |
| *CREATE* | *POST* |
| *RETRIEVE* | *GET* |
| *UPDATE* | *PUT* |
| *DELETE* | *DELETE* |

Sumber: Sandoval (2009)

Dibawah ini dijelaskan lebih rinci mengenai empat jenis HTTP *request* dan bagaimana masing-masing digunakan dalam memodifikasi keadaan *resource*. Untuk itu diambil satu contoh, misalnya terdapat RESTful *web service* yang beralamat di api.example.com. *Web service* tersebut menyediakan *resource* data mahasiswa.

***GET***

Metode *GET* digunakan *client* untuk meminta *resource* seluruh mahasiswa atau *resource* satu mahasiswa. Sehingga dalam contoh ini, URI api.example.com/mahasiswa untuk mengakses *resource* seluruh mahasiswa dan URI api.example.com/mahasiswa/{id} untuk mengakses *resource* satu mahasiswa yang memiliki identifikasi unik dari nilai id. Kode status HTTP yang paling umum untuk permintaan *GET* adalah 200 (*OK*). Kode pengalihan seperti 301 (*Moved Permanently*) juga umum digunakan (Richardson, 2013).

* Mendapatkan *resource* seluruh mahasiswa:

HTTP *Request*

GET /mahasiswa/ HTTP/1.1

Host: api.example.com

Content-Type: application/json

HTTP *Response*

“mahasiswa”: [

{

“id”: 1,

“nama”: “Budi”,

“jurusan”: “Teknik Informatika”

},

{

“id”: 2,

“nama”: “Joko”,

“jurusan”: “Sistem Informasi”

}

]

* Mendapatkan *resource* mahasiswa Budi:

HTTP *Request*

GET /mahasiswa/1/ HTTP/1.1

Host: api.example.com

Content-Type: application/json

HTTP *Response*

{

“id”: 1,

“nama”: “Budi”,

“jurusan”: “Teknik Informatika”

}

***POST***

Metode *POST* digunakan *client* untuk membuat *resource* mahasiswa baru. Dalam contoh ini, *client* perlu mengirimkan HTTP *POST* ke URI api.example.com/mahasiswa. Kode status HTTP yang paling umum untuk HTTP *POST* adalah 201 (*CREATED*), kode tersebut memberi tahu *client* bahwa *resource* baru telah dibuat. Kode lain seperti 202 (*Accepted*) juga umum digunakan, hanya saja *server* baru menerimapermintaan tersebut untuk diproses, namun pengolahannya belum selesai dilakukan (Richardson, 2013).

* Membuat *resource* mahasiswa baru:

HTTP *Request*

POST /mahasiswa/ HTTP/1.1

Host: api.example.com

Content-Type: application/json

{

“nama”: “Bambang”,

“jurusan”: “Teknik Komputer”

}

***PUT***

Metode *PUT* digunakan *client* untuk mengubah *resource* dari satu mahasiswa. Untuk mengubah *resource* dari mahasiwa Budi, *client* perlu mengirimkan HTTP *PUT* ke URI api.example.com/mahasiswa/1. Kode status HTTP yang paling umum untuk HTTP PUT adalah 200 (*OK*) atau 202 (*No Content*) (Richardson, 2013).

* Mengubah data jurusan mahasiswa Budi:

HTTP *Request*

POST /mahasiswa/1/ HTTP/1.1

Host: api.example.com

Content-Type: application/json

{

“jurusan”: “Sistem Informasi”

}

***DELETE***

Metode *DELETE* digunakan *client* untuk menghapus *resource* semua mahasiswa atau *resource* satu mahasiswa. Sehingga dalam contoh ini, URI api.example.com/mahasiswa untuk menghapus *resource* seluruh mahasiswa dan URI api.example.com/mahasiswa/{id} untuk menghapus *resource* satu mahasiswa yang memiliki identifikasi unik dari nilai id. Kode status HTTP yang paling umum untuk permintaan *DELETE* adalah 200 (*OK*)*, 202* (*Accepted*), atau 204 (*No Content*) (Richardson, 2013).

* Menghapus *resource* seluruh mahasiswa:

HTTP *Request*

DELETE /mahasiswa/ HTTP/1.1

Host: api.example.com

* Menghapus *resource* mahasiswa Budi:

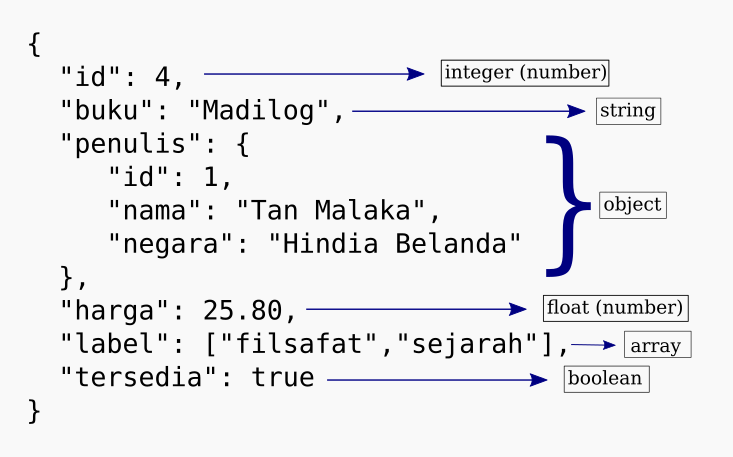
HTTP *Request*

DELETE /mahasiswa/1/ HTTP/1.1

Host: api.example.com

## JSON

JSON (*JavaScript Object Notation*) merupakan format pertukaran data berbasis teks yang ringan dan tidak bergantung pada bahasa pemrograman terentu. JSON dapat mewakili empat tipe data primitif seperti *string*, *number*, *boolean*, dan *null* ditambah dua tipe data struktural seperti *object* dan *array* (IETF, 2014). Format data ini menggunakan aturan penulisan yang sederhana dan membutuhkan pengelompokan antara definisi data dan nilai data. Pertama mengharuskan setiap elemen ditutup dengan karakter kurung kurawal: { dan }. Kedua, nilai dari elemen selalu berpasangan mengikuti struktur “nama”:”nilai”, dan dipisahkan menggunakan koma. Ketiga, *array* ditutup dengan karakter kurung siku: [ dan ] (Richardson, 2013). Contoh dari dokumen JSON dapat dilihat pada Gambar 2.6.



Gambar 2.6 Contoh dokumen JSON

Sumber: Penulis

## MongoDB

MongoDB merupakan sistem basis data non-relasional berorientasi dokumenyang *powerful*, fleksibel, dan *scalable*. Sistem basis data yang berorientasi dokumen menggantikan konsep “baris” dengan model yang lebih fleksibel yaitu “dokumen”. Dokumen tidak memiliki skema tetap yang didefinisikan diawal, hal ini menjadikan kunci dan nilai dari dokumen dapat berbeda dalam jenis dan ukuran. Hal tersebut membuat penambahan dan pengurangan *field* yang dibutuhkan dikemudian hari menjadi jauh lebih mudah (Kristina, 2013).

MongoDB dirancang untuk basis data terdistribusi yang menangani jumlah data yang besar. Model data yang berorientasi dokumen membuat MongoDB lebih mudah untuk membagi data di beberapa server berbeda. MongoDB secara otomatis menangani penyeimbangan data dan beban di *cluster*, mendistribusikan ulang dokumen secara otomatis dan mengarahkan permintaan pengguna ke mesin yang benar. Hal tersebut memungkinkan pengembang untuk fokus pada pembuatan kode aplikasi, bukan memikirkan bagaimana menangani jumlah data yang besar (Kristina, 2013).

### Dokumen

Dokumen adalah model data untuk MongoDB yang kurang lebih setara dengan satu baris dalam sistem basis data relasional (Kristina, 2013). Dokumen merupakan struktur data yang terdiri dari pasangan *field* dan nilai, menjadikannya identik dengan JSON. Nilai dari *field* dapat berisi dokumen lain, *array*, dan *array* dari dokumen sehingga dapat mengurangi kebutuhan untuk *join*. MongoDB menyimpan data dalam dokumen BSON. BSON (*Binary* JSON) merupakan representasi biner dari dokumen JSON yang mendukung lebih banyak tipe data (MongoDB, 2017).

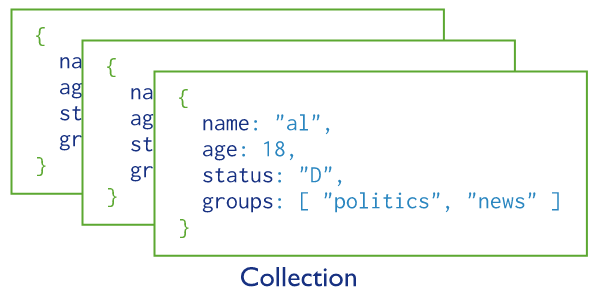


Gambar 2.7 MongoDB: dokumen

Sumber: MongoDB (2017)

### Koleksi

Koleksi merupakan sekumpulan dari dokumen BSON. Jika dokumen dalam MongoDB dianalogikan sebagai baris di basis data relasional, maka koleksi dapat dianalogikan sebagai tabel (Kristina, 2013). Koleksi tidak menerapkan skema tetap. Dokumen dalam koleksi dapat memiliki *field* yang berbeda. Namun biasanya, semua dokumen dalam koleksi memiliki tujuan yang sama atau terkait satu sama lain (MongoDB, 2017).



Gambar 2.8 MongoDB: koleksi dokumen

Sumber: MongoDB (2017)

### Tipe Data

MongoDB menyimpan data dalam dokumen BSON yang dapat dikatakan secara konseptual mirip dengan JSON. Dimana boleh dikatakan juga bahwa JSON kurang ekspresif untuk sistem basis data karena dukungan tipe data yang terbatas (*null*, *boolean*, *numeric*, *string*, *array*, dan *object*). BSON menambahkan dukungan terhadap sejumlah tipe data sekaligus menjaga sifat dasar JSON. Berikut beberapa tipe data BSON yang umum digunakan (Kristina, 2013):

* null

Tipe data untuk merepresentasikan nilai *null* dan *field* yang tidak ada:

{“x” : null}

* *boolean*

Tipe data untuk menyimpan nilai kebenaran “*true*” atau “*false*”:

{“x” : true}

* *number*

Tipe data untuk menyimpan nilai dalam bentuk angka. Secara *default* angka disimpan dalam bilangan titik mengambang sebesar 64-*bit*. Untuk bilangan bulat, dapat digunaan *NumberInt* atau *NumberLong*, yang masing-masing mewakili bilangan bulat sebesar 4-*byte* dan 8-*byte*:

{“x” : 3.14, “y” : NumberInt(“3”)}

* *string*

Tipe data untuk menyimpan setiap *string* dari karakter UTF-8:

{“x” : “foobar”}

* *date*

Tipe data untuk menyimpan tanggal dalam bentuk milidetik tanpa zona waktu:

{“x” : new Date()}

* *regular expression*

Digunakan pada kueri pencarian pola string menggunakan sintaks *regular expression* milik JavaScript:

{“x” : /foobar/i}

* *array*

Tipe data untuk menyimpan kumpulan atau daftar dari sebuah nilai dengan tipe data apapun di dalam sebuah larik:

{“x” : [“a”, “b”, “c”]}

* *embedded document*

Tipe data ini menyematkan keseluruhan dokumen menjadi nilai dari suatu *field* di dokumen utamanya:

{“x” : {“foo” : “bar”}}

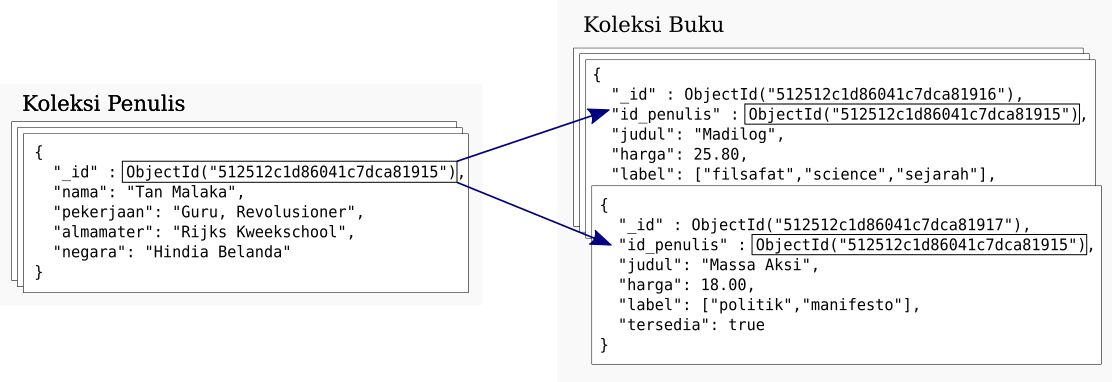
* *object id*

Object Id merupakan identifikasi sebesar 12-*byte* untuk dokumen. Setiap dokumen yang disimpan di MongoDB harus memiliki kunci “\_id” yang unik satu sama lain. Kunci “\_id” dapat menggunakan tipe data apapun, namun secara *default* menggunakan tipe *ObjectId*:

{"x" : ObjectId()}

### *Reference*

MongoDB menyediakan fasilitas *reference* untuk normalisasi data. Normalisasi membagi data ke dalam beberapa koleksi untuk menekan redudansi, yaitu keadaan dimana dokumen berbeda menyimpan data yang sama. *Reference* membuat bagian data yang berada dalam satu koleksi dapat dirujuk oleh dokumen lainnya. Caranya adalah dengan menyimpan *field* \_id dari satu dokumen oleh dokumen lainnya sebagai rujukan. Dengan tidak adanya fasilitas *join* seperti halnya basis data relasional, ini menjadikan proses membaca dokumen dari beberapa koleksi membutuhkan lebih dari satu kueri (Kristina, 2013). Kueri pertama untuk dokumen utamanya, kueri setelahnya untuk dokumen lainnya sebagai rujukan.



Gambar 2.9 MongoDB: reference

Sumber: Penulis

## JSON Web Token

*JSON Web Token* (JWT) merupakan token yang digunakan sebagai sarana untuk mewakili klaim otoritas dalam komunikasi di antara dua pihak. Klaim dikodekan dalam objek JSON yang juga berisi informasi berbasis teks didalamnya (IETF, 2015). Informasi tersebut dapat diverifikasi dan dipercaya karena telah dibubuhkan tanda tangan digital. JWT ditandatangani dan diverifikasi menggunakan algoritma *hash* seperti HMAC dan RSA untuk menjaga kerahasiaan dan keaslian datanya (jwt.io).

JWT menghasilkan token yang *compact* dalam ukuran kecil sehingga dapat langung dikirimkan melalui *header* HTTP *Authorization* dan parameter kueri lewat alamat URI (IETF, 2015). Ukuran yang *compact* membuat JWT dapat ditransmisikan dengan cepat. JWT dapat digunakan untuk menggantikan HTTP *session* dengan menyimpan semua informasi tentang pengguna di dalamnya. Sama halnya seperti HTTP *session*, JWT juga memiliki masa berlaku yang dapat diatur sesuai kebutuhan.

### Struktur

Struktur JWT secara keseluruhan terdiri dari 3 bagian, yaitu: *header*, *payload*, dan *signature*. Setiap bagian dikodekan dalam skema *base64url* dan dipisahkan oleh karakter titik (‘.’). Sehingga JWT akan memiliki format seperti xxxxx.yyyyy.zzzzz.

* *header*

Bagian ini biasanya hanya terdiri dari dua *field* untuk mengatur algoritma *hash* dan jenis token yang digunakan. Contoh:

{

"alg": "HS256",

"typ": "JWT"

}

* *payload*

Bagian ini berisi klaim otoritas (biasanya berupa informasi pengguna) dan metadata tambahan. Metadata tambahan yang paling umum digunakan adalah *exp* yang mendefinisikan masa berlaku dari suatu token. *Exp* haruslah berasal dari tipe data *NumericDate*. Contoh:

{

"id": "1234567890",

"name": "John Doe",

"admin": true,

"exp": 1416471934

}

* *signature*

Bagian ini berisi informasi untuk memverifikasi tanda tangan digital sesuai dengan algoritma *hash* yang disepakati di bagian *header*. Tanda tangan digital tersebut dibuat berdasarkan *secret* berupa string rahasia yang ditentukan sendiri. Token dan *secret* selalu berkaitan. Jika saat diverifikasi token dan *secret* tidak sesuai, maka token dianggap tidak valid. Contoh:

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload), secret

)

Jika *header* dan *payload* pada contoh diatas dikodekan menggunakan skema *base64url*, kemudian juga telah dibubuhkan tanda tangan digital akan menghasilkan JWT secara utuh seperti Gambar 2.10 dibawah ini.



Gambar 2.10 Contoh JSON Web Token

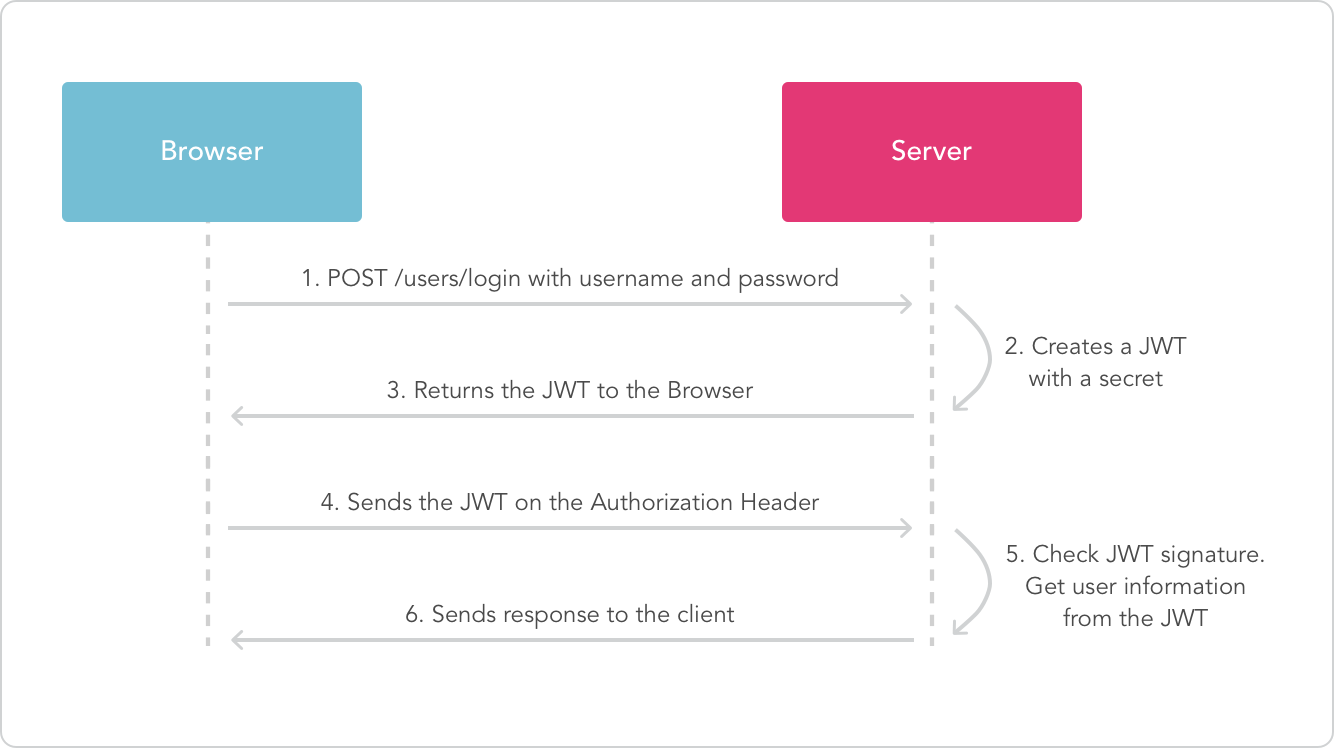
Sumber: jwt.io

### Otentikasi dan Otorisasi

Pada proses otentikasi, *user* yang telah melakukan proses *login* dapat dibedakan menggunakan *credentials* yang dimiliki. Dari *credentials*, dapat ditentukan apakah suatu *user* memiliki hak akses untuk operasi atau sumberdaya yang dilindungi. JSON Web Token menyimpan *credentials* dan data *user* secara lokal. Itu berarti *server* tidak perlu menyimpan informasi tersebutkarena sudah terwakili oleh JWT. Sedangkan pada pendekatan otentikasi tradisonal, *credentials* dan data *user* disimpan didalam HTTP *session* di *server* dan mengirimkan kembali *cookie* untuk disimpan (mis., oleh *browser*).

Setiap kali *user* ingin menjalankan operasi atau mengakses sumberdaya yang dilindungi, *user* harus mengirimkan JWT untuk keperluan otentikasi. JWT biasanya disisipkan dalam *header Authorization* menggunakan skema *Bearer*:

Authorization: Bearer <token>



Gambar 2.11 Contoh Otorisasi Menggunakan JSON Web Token

Sumber: jwt.io

Sebelum permintaan *user* diproses, *server* akan memeriksa kevalidan JWT pada *header Authorization* terlebih dahulu. Jika valid, *user* akan diizinkan melakukan operasi dan mengakses sumberdaya yang dilindungi. Jika tidak, *server* akan menolak permintaan tersebut sehingga operasi dan sumberdaya dapat terlindungi dari *user* yang tidak memiliki wewenang. Karena JWT bersifat mandiri, *credentials* dan data user yang diperlukan ada didalam token, sehingga mengurangi kebutuhan untuk mengeksekusi kueri basis data berkali-kali.

# METODOLOGI

## Jenis Penelitian

Penelitian yang dilakukan penulis termasuk dalam jenis penelitian rancang bangun. Keluaran dari penelitian jenis ini adalah menghasilkan sebuah perangkat lunak yang dibangun secara sistematis. Penelitian dimulai dengan analisis kebutuhan, perancangan, implementasi ke dalam kode sumber program, serta pengujian terhadap produk perangkat lunak yang dihasilkan.

## Metodologi Penelitian

Metodologi penelitian mengambarkan panduan alur pengerjaan dalam penelitian yang ditujukan agar penelitian dapat berjalan secara sistematis dan berurutan. Berikut merupakan gambaran metodologi penelitian berupa diagram alir yang ditunjukan pada Gambar 3.1.



Gambar 3.1 Flowchart tahapan penelitian

Sumber: Penulis

### Studi Literatur

Studi literatur merupakan kegiatan yang dilakukan untuk mempelajari dan memahami teori dan referensi yang digunakan pada penelitian. Teori dan referensi tersebut dapat berasal dari jurnal, buku, dan penelitian sebelumnya. Adapun bahan studi literatur yang perlu dijelaskan adalah: protokol HTTP, Web Service, format data JSON, sistem basis data MongoDB, dan JSON Web Token. Studi literatur tersebut dirancang sebagai pedoman pengetahuan dasar dalam melakukan analisis, perancangan, implementasi dan pengujian dalam tahap-tahap penelitian. Hal ini bertujuan agar mendapat pemahaman yang lebih mendalam terhadap pokok bahasan yang diangkat.

### Analisis Kebutuhan Sistem

Analisis kebutuhan merupakan tahapan dalam menentukan apa yang dibutuhkan dan dapat dilakukan oleh sistem. Analisis kebutuhan diperlukan agar dapat memetakan kebutuhan sistem mengenai fitur dan komponen untuk menghindari penggunaan sumberdaya yang tidak perlu. Kebutuhan yang digunakan dalam penelitian ini dapat dibagi ke dalam dua bagian yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak. Keluaran dari tahapan ini menjadi landasan dalam tahap perancangan.

### Perancangan Sistem

Perancangan sistem memberikan gambaran mengenai alur kerja sistem berdasarkan kebutuhan yang telah ditentukan. Perancangan dibuat sebagai pedoman untuk mempermudah tahap implementasi. Perancangan dapat dibagi ke dalam empat komponen berdasarkan peran dan fungsinya, yakni: perancangan komponen komunikasi, perancangan komponen manajemen data, perancangan komponen *security*, dan perancangan komponen *web* console. Secara umum, alur perancangan dapat dilihat pada Gambar 3.2dibawah ini:



Gambar 3.2 Tahap Perancangan

Adapun detail dari tahap reancangan diatas adalah sebagai berikut:

1. Pada perancangan komponen komunikasi akan digambarkan bagaimana alur komunikasi antar perangkat node dan aplikasi *client* ke *web service* dan sebaliknya.
2. Pada perancangan komponen manajemen data akan dijelaskan mengenai skema basis data dan alur untuk mengakses data tersebut melalui komponen komunikasi.
3. Perancangan komponen *security* menjelaskan alur dalam manajemen perangkat serta alur otentikasi yang berbeda antara perangkat dan aplikasi *client* menggunakan token akses JWT. Dijelaskan juga tentang bagaimana alur untuk mendapatkan token akses JWT, registrasi pengguna, dan menerima data sensor dari perangkat node.
4. Perancangan komponen *web console* menjelaskan rancangan antarmuka pengguna untuk memanajemen perangkat dan melihat data sensor.

### Implementasi

Implementasi merupakan tahapan dalam pembuatan IoT *cloud platform* yang diajukan dengan mengacu pada kebutuhan di tahap perancangan. Implementasi meliputi:

1. Implementasi komponen komunikasi, menjelaskan terkait dengan instalasi dan konfigurasi yang diperlukan untuk menjalankan RESTful *Web Service*.
2. Implementasi komponen manajemen data, meliputi:

* Implementasi data model, mengubah skema basis data pada tahap perancangan ke dalam *Model Class* MongoEngine. MongoEngine merupakan sebuah *module* Python tambahan yang memungkinkan aplikasi yang dibangun dapat bekerja dengan sistem basis data MongoDB.
* Implementasi data *access*, berisi kode sumber untuk mengakses data sensor melalui RESTful *Web Service*.

1. Implementasi komponen *security*, meliputi beberapa implementasi sebagai berikut:

• Implementasi otorisasi dan otentikasi, berisi kode sumber untuk membuat token akses JWT, registrasi pengguna dan menerima data sensor dari perangkat node.

• Implementasi manajemen perangkat, berisi kode sumber untuk menambah, mengubah, dan menghapus perangkat.

1. Implementasi komponen *web console*, berisi hasil akhir dari antarmuka pengguna beserta kode sumbernya.

### Pengujian dan Analisis Hasil Pengujian

Pengujian dilakukan untuk mengetahui apakah IoT *cloud platform* yang dibangun sudah sesuai dengan kebutuhan yang telah didefiniskan di tahapan analisis dan perancangan. Sistem juga harus diuji apakah dapat menyelesaikan masalah yang diangkat yang ada pada pendahuluan dan rumusan masalah. Pada penelitan ini terdapat dua jenis pengujian yaitu pengujian fungsional dan pengujian kehandalan sistem.

Pengujian fungsionalitas dilakukan dengan beberapa skenario untuk menguji fitur-fitur pada perangkat lunak IoT *cloud platform* dapat berjalan dengan benar. Tahap pengujian terdiri atas beberapa skenario:

1. Peneliti menguji IoT *cloud platform* dalam proses menerima data sensor dari perangkat node dan menyimpannya ke dalam basis data.
2. Peneliti menguji proses otentikasi dan otorisasi dari perangkat node dan aplikasi *client* berdasarkan JSON Web Token yang dikirimkan.
3. Peneliti menguji IoT *cloud platform* dalam proses manajemen perangkat node dan lihat data sensor.

Pengujian kehandalan sistem dilakukan dengan beberapa skenario untuk mengetahui performa perangkat lunak IoT *cloud platform* yang telah dibangun. Tahap pengujian terdiri atas beberapa skenario:

1. Performa fitur mengirimkan data sensor

Peneliti mengukur berapa banyak HTTP *request* yang dapat ditangani sistem ketika perangkat node mengirimkan data sensor. Pengujian ini menggunakan parameter uji *throughput*, *latency* dan *error rate*. Nilai dari parameter uji didapatkan ketika HTTP *request* dikirimkan oleh user sebanyak 50, 100, dan 150 secara bersamaan.

1. Performa fitur mengakses data sensor

Peneliti mengukur berapa banyak HTTP *request* yang dapat ditangani sistem ketika aplikasi *client* mengakses data sensor. Pengujian ini secara spesifik menguji tiga kriteria dalam mengakses data sensor, yaitu kriteria berdasarkan pengguna, perangkat node dan sensor. Parameter uji yang digunakan untuk mengukur performa fitur ini adalah *throughput*, *latency* dan *error rate*. Nilai dari parameter uji didapatkan ketika HTTP *request* dikirimkan oleh user sebanyak 50, 100, dan 150 secara bersamaan.

Berdasarkan pengujian diatas, hasil kemudian diolah dan dianalisis untuk mengetahui apakah telah sesuai dengan tahap kebutuhan dan perancangan sebelumnya, memiliki otorisasi dan otentikasi yang sesuai, dan kehandalan yang diharapkan.

### Kesimpulan dan Saran

Pengambilan kesimpulan dapat dilakukan jika semua tahapan sebelumnya telah selesai dibangun dan dilakukan. Kesimpulan diambil melalui hasil data pengujian terhadap sistem perangkat lunak yang telah dibangun. Pada bagian ini juga memuat saran-saran untuk penelitian selanjutanya.

# ANALISIS KEBUTUHAN DAN PERANCANGAN

Pada bab ini akan dibahas mengenai kebutuhan fungsional dan non-fungsional yang harus dipenuhi oleh sistem. Kebutuhan fungsional merupakan layanan, fitur, atau proses apa saja yang disediakan untuk penguna sistem. Sedangkan kebutuhan non-fungsional merupakan perilaku dan batasan arsitektur yang ada pada sistem. Kebutuhan tersebut kemudian dikembangkan menjadi sebuah rancangan yang menjadi landasan dalam mengimplementasikan sistem. Tahap perancangan terdiri dari perancangan komponen komunikasi, perancangan komponen manajemen data, perancangan komponen manajemen perangkat dan perancangan komponen *web console*.

## Deskripsi Umum Sistem

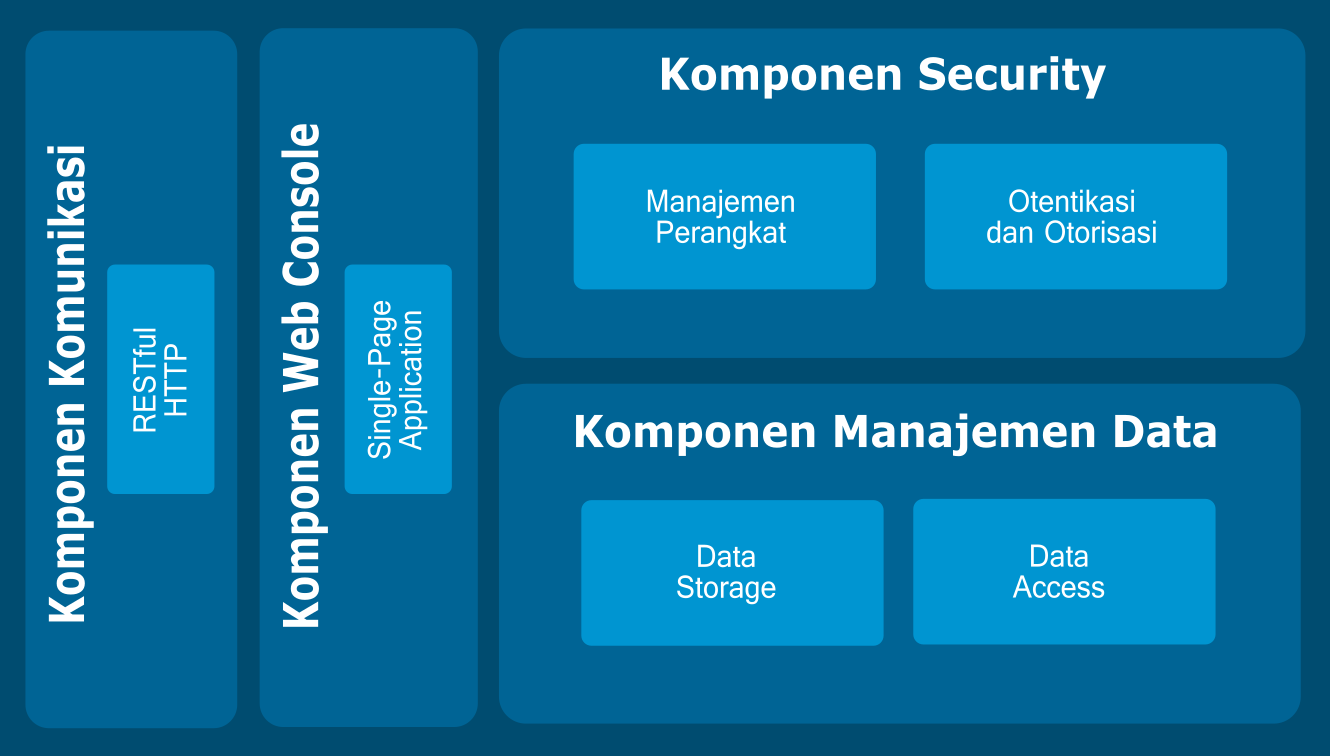
Pada penelitian ini dikembangkan perangkat lunak IoT *cloud platform* yang menyediakan fitur untuk menerima data sensor, mengakses data sensor dan memanajemen perangkat melalui protokol HTTP. Terdapat dua entitas yang berkomunikasi dengan perangkat lunak ini, yaitu perangkat node dan aplikasi *client*. Perangkat node merupakan *microcontroller* dengan modul sensor tertentu yang mengirimkan data sensor ke perangkat lunak IoT *cloud* *platform*. Sedangkan aplikasi *client* merupakan perangkat lunak multi-platformyang berkomunikasi dengan perangkat lunak IoT *cloud* *platform* untuk mengakses data sensor dan memanajemen perangkat. Perangkat lunak IoT *cloud platform* ini menyediakan RESTful *web service* yang memiliki antarmuka terstandar dan format yang baku dalam melakukan komunikasi. Hal ini membuat *developer* hanya perlu fokus pada pengiriman datanya dan dapat mengembangkan aplikasi *client* sendiri untuk platform dan Bahasa pemrograman yang lain.

Untuk dapat menggunakan perangkat lunak IoT *cloud platform*, pengguna diharuskan melakukan registrasi terlebih dahulu. Setiap pengguna nantinya dapat memiliki lebih dari satu perangkat node dan setiap perangkat node dapat memiliki lebih dari satu sensor. Pengguna dapat memanajemen perangkat node yang ia punya melalui operasi CRUD (*create, read, update,* dan *delete*). Melalui manajemen perangkat juga dapat diatur pembatasan pengiriman dan visibilitas perangkat terhadap pengguna yang lain. Fitur pembatasan pengiriman membuat perangkat node dapat dibatasi dalam jumlah tertentu untuk mengirimkan data sensor per-harinya. Sedangkan fitur visibilitas perangkat memberikan *policy* dalam mengakses data perangkat menggunakan label visibilitas “*public*” atau “*private*”. Pengguna dapat melihat data sensor dari perangkatnya sendiri dan dari perangkat pengguna lain yang memiliki label visibilitas “*public”*’.

Perangkat node yang didaftarkan memiliki identifikasi unik berupa label dan *secretkey*. Keduanya digunakan untuk mendapatkan token akses yang digunakan dalam proses otentikasi dan otorisasi mengirimkan data sensor ke perangkat lunak IoT *cloud platform*. Dalam mengirimkan data sensor, akan disediakan suatu format pengirimakn tertentu untuk mengabungkan data sensor dari beberapa sensor yang dimiliki perangkat node tersebut, sehingga dapat dikirimkan dalam satu HTTP *request* yang sama. Pengiriman data juga dapat dibatasi pada operasi manajemen perangkat. Jika dibatasi, pengiriman data per-harinya akan berjumlah seperti yang sudah ditentukan. Untuk itu, digunakan bilangan *counter* yang akan berkurang satu setiap perangkat melakukan pengiriman. Jika bilangan *counter* habis atau sama dengan nol maka perangkat tidak dapat mengirimkan data sensor ke *cloud*. Bilangan *counter* tersebut akan kembali di atur ulang secara otomatis setiap pukul 24:00.

IoT *cloud platform* yang dikembangkan menerapkan fitur sekuritas seperti otentikasi dan otorisasi menggunakan token akses JWT. Dua entitas yang berinteraksi dengan IoT *cloud platform* yakni perangkat node dan aplikasi *client* akan memiliki token akses yang berbeda. Token perangkat node digunakan untuk pengiriman data sensor sedangkan token aplikasi *client* digunakan untuk mengakses data sensor dan memanajemen perangkat. Penggunaan token harus sesuai, misalnya token aplikasi *client* tidak dapat digunakan untuk mengirimkan data sensor begitu juga sebaliknya token perangkat node tidak dapat digunakan untuk memanajemen perangkat.

Sistem secara umum terbagi ke dalam empat komponen berbeda yakni komponen komunikasi, komponen *security*, komponen manajemen data dan komponen *web console*. Komponen komunikasi merupakan RESTful *web service* yang menangani komunikasi dari perangkat dan aplikasi *client* melalui protokol HTTP. Komponen manajemen data menangani mekanisme yang berkaitan dengan *data storage* dan menyediakan mekanismeuntuk mengakses data sensor melalui RESTful *web service*. Komponen *security* menyediakan mekanisme untuk otentikasi dan otorisasi; dan memanajemen perangkat. Sedangakan komponen web *console* merupakan *single-page application* yang menyediakan antarmuka pengguna untuk mengakses data sensor dan memanajemen perangkat. Keempat komponen-komponen tersebut dapat dilihat pada Gambar 4.1.



Gambar 4.1 Komponen Sistem

## Analisis Kebutuhan

Analisis kebutuhan merupakan tahapan dalam mengumpulkan informasi tentang apa-apa saja yang dapat dilakukan dan dibutuhkan oleh sistem. Kebutuhan dibagi ke dalam dua yaitu kebutuhan fungsional dan kebutuhan non-fungsional.

### Kebutuhan Fungsional

Kebutuhan fungsional merupakan layanan, fitur, atau proses apa saja yang harus dipenuhi untuk penguna sistem. Kebutuhan fungsional dari sistem yang diajukan pada penelitian ini dijelaskan pada Tabel 4.1 :

Tabel 4.1 Kebutuhan Fungsional Sistem

|  |  |
| --- | --- |
| Nomor | Kebutuhan |
| 1 | Sistem dapat menyediakan layanan bagi perangkat node dan aplikasi *client* untuk mendapatkan token akses JWT. |
| 2 | Sistem dapat menyediakan layanan bagi aplikasi *client* untuk melakukan registrasi pengguna. |
| 3 | Sistem dapat menyediakan layanan bagi perangkat node untuk mengirimkan data sensor. |
| 4 | Sistem dapat menyediakan layanan bagi aplikasi *client* untuk mengakses data sensor. |
| 5 | Sistem dapat menyediakan layanan bagi aplikasi *client* untuk memanajemen perangkat node. |
| 6 | Sistem dapat menerapkan otentikasi menggunakan token akses JWT. |
| 7 | Sistem dapat menerapkan otorisasi menggunakan token akses JWT. |

### Kebutuhan Non-Fungsional

Kebutuhan non-fungsional merupakan perilaku dan batasan arsitektur yang ada pada sistem. Kebutuhan non-fungsional terbagi menjadi dua yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

#### 4.2.2.1 Kebutuhan Perangkat Keras

Kebutuhan perangkat keras merupakan spesifikasi perangkat keras yang digunakan dalam mengambangkan perangkat lunak IoT *cloud platform*, spesifikasi dijelaskan pada Tabel 4.2 :

Tabel 4.2 Kebutuhan Perangkat Keras

|  |  |
| --- | --- |
| Nama Komponen | Spesifikasi |
| Prosessor | Intel Core i3 – 4030U, 1.9 Ghz |
| Memori (RAM) | 2 GB DDR 3L x 2 |
| Harddisk | 500 GB 5200 RPM |

#### 4.2.2.2 Kebutuhan Perangkat Lunak

Kebutuhan perangkat lunak merupakan spesifikasi perangkat lunak yang digunakan dalam mengambangkan IoT *cloud platform*, spesifikasi dijelaskan pada Tabel 4.3 :

Tabel 4.3 Kebutuhan Perangkat Lunak

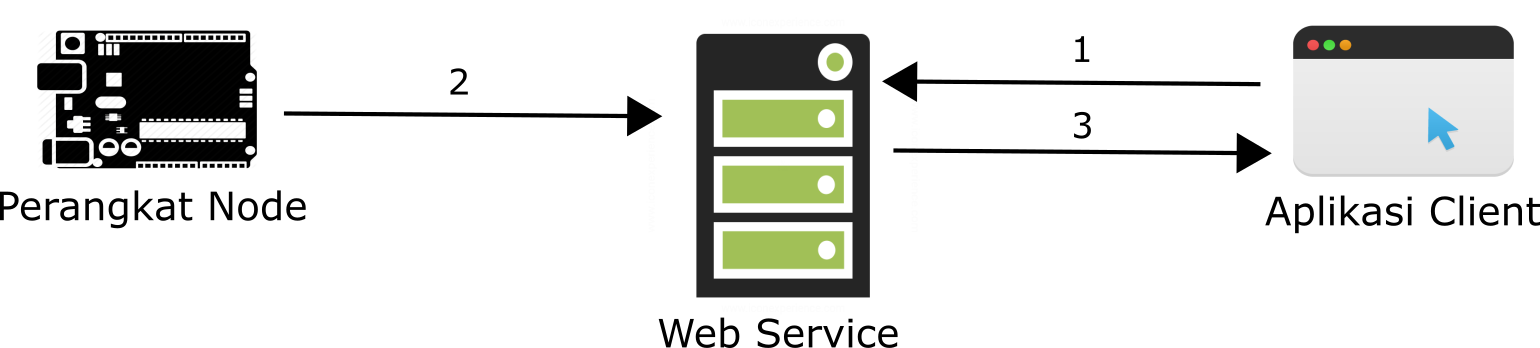
|  |  |  |
| --- | --- | --- |
| Nama Komponen | Keterangan | Detail |
| Fedora Workstation | Sistem operasi yang digunakan untuk merancang dan mengembangkan sistem. | Fedora 25 (Workstation) |
| Python | Bahasa pemrograman. | Versi 2.7 |
| Django Framework | Web Framework untuk mengembangkan RESTful Web Api. | Versi 1.9 |
| Node.js | Framework yang dibutuhkan Angular. | Versi 4.2.6 |
| Angular | Framework untuk mengembangkan Aplikasi Manajemen Perangkat. | Versi 2.4.0 |
| MongoDB | Sistem basis data non-relasional. | Versi 3.4.2 |
| PyCharm | Editor pemrograman untuk memprogram RESTful *Web Service*. | Versi 2016.1 |
| Visual Studio Code | Editor pemrograman untuk memprogram aplikasi manajemen perangkat. | Versi 11 |
| Oracle VirtualBox | Perangkat lunak untuk menjalankan mesin virtual. | Versi 5.1 |
| Apache Jmeter | Perangkat lunak pengujian untuk menguji kehandalan sistem yang dibangun | Versi 3.2 |

## Perancangan

Perancangan meliputi perancangan komponen komunikasi, perancangan komponen manajemen data, perancangan komponen sekuritas, dan perancangan aplikasi manajemen perangkat.

### Perancangan Komponen Komunikasi

Perancangan komponen komunikasi menggambarkan alur komunikasi antara tiga entitas yaitu perangkat node, perangkat lunak IoT *cloud platform*, dan aplikasi *client*. Alur komunikasi antar entitas dibagi menjadi dua yakni pengiriman data dari perangkat node ke IoT *cloud platform* dan aplikasi *client* kepada IoT *cloud platform* dalam mengakses data sensor dan memanajemen perangkat. Kedua alur komunikasi pada sistem yang dikembangkan dapat dilihat pada Gambar 4.2.



Gambar 4.2 Alur Komunikasi Antar Entitas

Keterangan pada Gambar 4.2 diatas adalah:

1. Aplikasi *client* melakukan manajemen perangkat.
2. Perangkat yang sudah didaftarkan mengirimkan data sensor ke perangkat lunak IoT *cloud platform*.
3. Aplikasi *client* mendapatkan data sensor dari perangkat lunak IoT *cloud platform*.

### Perancangan Komponen Manajemen Data

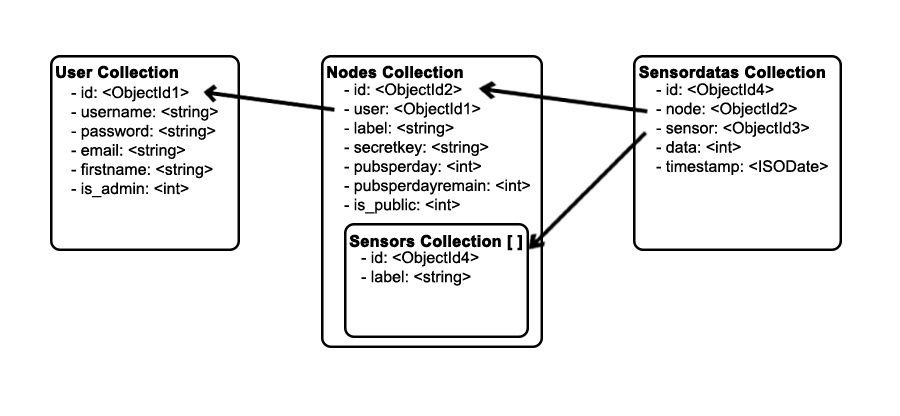
Perancangan komponen manajemen data dibagi menjadi dua yakni perancangan data model dan perancangan data *access*. Perancangan data model menggambarkan mengenai skema basis data yang digunakan. Sedangkan perancangan data access menggambarkan alur komunikasi dalam mengakses data dari data model.

#### 4.3.2.1 Perancangan Data Model

Pada perancangan data model akan digambarkan mengenai skema basis data untuk menyimpan informasi mengenai pengguna, data sensor, dan perangkat. Perancangan disesuaikan dengan sistem basis data MongoDB yang berorientasi dokumen. Pada tahap ini diidentifikasi dokumen, *field* beserta tipe data disetiap dokumen. Untuk menekan redudansi data, diidentifikasi juga hubungan antar dokumen menggunakan *reference* dari satu dokumen ke dokumen lainnya.

Dalam proses identifikasi diketahui bahwa perangkat lunak IoT *cloud platform* memerlukan empat dokumen yaitu *User, Nodes*, *Sensors*, dan *Sensordatas*. Dokumen *Users* menyimpan informasi mengenai data pengguna sistem, dokumen *Nodes* menyimpan informasi mengenai perangkat node, dokumen *sensor* menyimpan informasi mengenai sensor yang dimiliki suatu perangkat node, dan dokumen *Sensordatas* menyimpan data yang dikirimkan oleh perangkat node di lapangan. Penjelasan lebih detail mengenai *field* dan tipe data dari setiap dokumen tersebut dapat dilihat pada Tabel 4.4, Tabel 4.5, Tabel 4.6, dan Tabel 4.7.

Dari keempat dokumen tersebut, kemudian dilakukan identifikasi hubungan antar dokumen untuk menekan redudansi data. Hubungan antar dokumen tersebut dapat dilihat pada Gambar 4.3.



Gambar 4.3 Skema Basis Data

Penjelasan dari Gambar 4.3 adalah:

Dokumen *Users* memiliki hubungan satu-ke-banyak ke dokumen *Nodes* karena satu pengguna dapat memiliki beberapa perangkat node.

Dokumen *Nodes* memiliki hubungan satu-ke-banyak ke dokumen *Sensors* karena dalam satu perangkat node dapat memiliki beberapa sensor.

Dokumen *Sensors* merupakan *EmbeddedDocument* ber tipe data *Array*. Secara *default*, *EmbeddedDocument* tidak memiliki *field* id yang di-*generate* secara otomatis. Pada dokumen ini *field* id dibutuhkan untuk mempermudah dalam melakukan operasi CRUD. Oleh karena itu *field* id harus di-*generate* secara manual.

Dokumen *Sensordatas* memiliki hubungan satu-ke-banyak ke dokumen *Nodes* dan dokumen *Sensors* sehingga dapat diketahui dari mana data sensor tersebut diambil. Dokumen ini bukan merupakan *EmbeddedDocument* sebagaimana dokumen *Sensors* dengan pertimbangan data sensor akan terus bertambah seiring dengan berjalannya waktu. Dengan begitu, memisahkannya menjadi dokumen sendiri akan membuat performa penulisan menjadi lebih cepat (Kristina, 2013:153).

Tabel 4.4 Dokumen Users

|  |  |  |
| --- | --- | --- |
| Nama *field* | Tipe data | Keterangan |
| id | ObjectId | Nilai id unik untuk membedakan satu dokumen terhadap dokumen lainnya dalam koleksi *Users*. |
| username | String | Username pengguna. |
| password | String | Kata sandi pengguna. |
| email | String | Alamat surel pengguna. |
| firstname | String | Nama depan pengguna. |
| last\_name | String | Nama akhir pengguna. |
| is\_admin | Int | Bilangan integer yang menunjukkan hak akses pengguna. Nilai 1 berarti pengguna memiliki hak akses Admin, sedangkan nilai 0 berarti pengguna memiliki hak akses pengguna biasa. |

Tabel 4.5 Dokumen Sensors

|  |  |  |
| --- | --- | --- |
| Nama *field* | Tipe data | Keterangan |
| id | ObjectId | Nilai id unik untuk membedakan satu dokumen terhadap dokumen lainnya dalam koleksi *Sensors*. |
| label | String | Label dari sensor. |

Tabel 4.6 Dokumen Nodes

|  |  |  |
| --- | --- | --- |
| Nama *field* | Tipe data | Keterangan |
| id | ObjectId | Nilai id unik untuk membedakan satu dokumen terhadap dokumen lainnya dalam koleksi *Nodes*. |
| user | ObjectId | Nilai id dari dokumen *Users* sebagai rujukan. |
| label | String | Label dari perangkat node. |
| secretkey | String | Karakter rahasia untuk keperluan otentikasi. |
| pubsperday | Int | Jumlah pengiriman yang dapat dilakukan perangkat node per harinya. Nilai -1 berarti pengiriman tidak dibatasi. |
| pubsperdayremain | Int | Bilangan *counter* dari *field* pubsperday. Pada awalnya akan bernilai sama seperti pubsperday dan akan berkurang satu setiap kali perangkat melakukan pengiriman data sensor. *Field* ini akan di-*reset* setiap harinya pada pukul 24:00. |
| is\_public | Int | Bilangan integer yang menunjukkan visibilitas perangkat terhadap pengguna lain. Nilai 1 berarti data sensor dapat dilihat oleh pengguna lain, sedangkan nilai 0 berarti data sensor hanya dapat dilihat oleh pengguna yang bersangkutan. |
| sensors | Array<Sensors> | *EmbeddedDocument* bertipe *array*, menyimpan daftar sensor yang dimiliki oleh suatu perangkat node. |

Tabel 4.7 Dokumen Sensordatas

|  |  |  |
| --- | --- | --- |
| Nama *field* | Tipe data | Keterangan |
| id | ObjectId | Nilai id unik untuk membedakan satu dokumen terhadap dokumen lainnya dalam koleksi *Sensordatas*. |
| node | ObjectId | Nilai id dari dokumen *Nodes* sebagai rujukan. |
| sensor | ObjectId | Nilai id dari dokumen *Sensors* sebagai rujukan. |
| data | Int | Data dari sensor. |
| timestamp | ISOdate | Waktu ketika data sensor diambil. |

#### 4.3.2.2 Perancangan Data *Access*

Perancangan data *access* digunakan untuk menggambarkan alur dan proses yang terjadi dalam mengakses data sensor. Dalam IoT *cloud platform* yang dirancang ini, peneliti menyediakan API berupa *web service* untuk mengakses data sensor yang ada di *cloud*. Terdapat tiga kriteria yang dapat digunakan, yakni berdasarkan pengguna, berdasarkan perangkat node, dan berdasarkan sensor. Pada kriteria berdasarkan perangkat node dan kriteria berdasarkan sensor memungkinkan untuk mengakses data sensor milik pengguna lain selama perangkat node tersebut memiliki label visibilitas publik. Selain itu, data sensor juga dapat diakses berdasarkan kriteria waktu yang dapat dikombinasikan dengan tiga kriteria sebelumnya dan bersifat optional. Kriteria waktu ditentukan oleh dua parameter HTTP GET yaitu: *start* dan *end*.

Parameter *start* digunakan untuk mengakses data sensor dari waktu yang ditentukan tersebut sampai data sensor yang paling baru. Sedangkan parameter *end* digunakan untuk mengakses data sensor dari awal sampai data sensor dari waktu yang ditentukan. Penggunaan kedua parameter secara bersamaan akan menghasilkan kriteria berdasarkan rentang waktu dari *start* sampai *end*. Sebagai contoh, pengguna menggunakan kriteria perangkat node X dan kriteria waktu dengan parameter *start* 2016-11-01 00:00:00 dan parameter *end* 2016-12-01 00:00:00. Hal tersebut membuat pengguna dapat mengakses semua data sensor yang dimiliki oleh perangkat node X pada rentang waktu 2016-11-01 00:00:00 sampai 2016-12-01 00:00:00.

Dalam mengakses data sensor digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman. Setiap HTTP *request* akan memberikan HTTP *response* berupa halaman dengan maksimal berisi 10 data di dalamnya. Untuk menggunakan fitur *pagination*, dibutuhkan parameter HTTP GET tambahan yaitu *page* untuk menentukan halaman yang ingin diakses. Parameter *page* merupakan bilangan integer dengan nilai *default* sama dengan 1. Jumlah halaman didapatkan menggunakan rumus: jumlah keseluruhan data/10. Sebagai contoh, jika suatu kriteria data sensor menghasilkan sejumlah 800 data, maka jumlah halaman pada kriteria tersebut adalah 80 halaman.

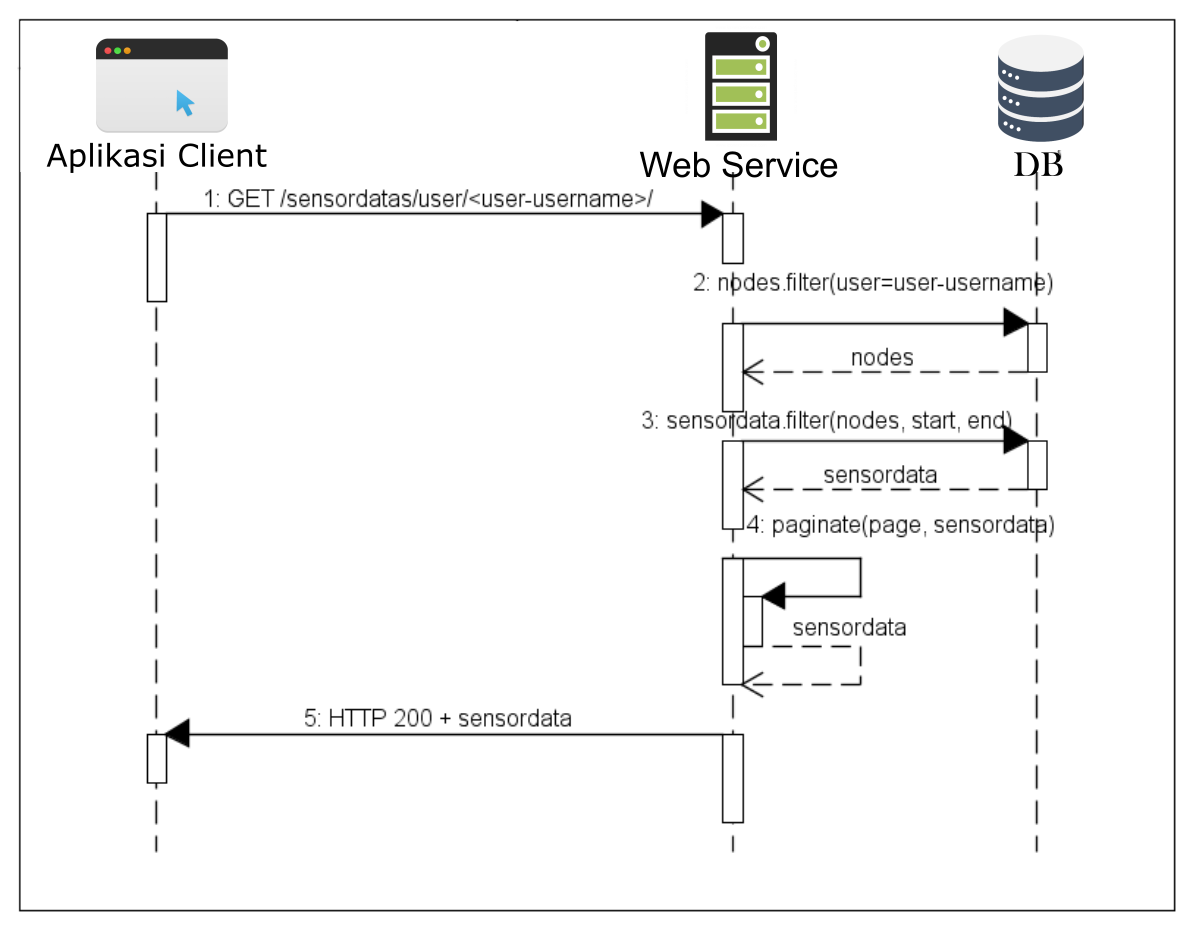
Dibawah ini dijelaskan mengenai alur interaksi dalam mengakses data sensor berdasarkan pengguna, berdasarkan perangkat node, dan berdasarkan sensor.

1. Berdasarkan Pengguna

Penjelasan mengenai parameter GET dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.8. Sedangkan alur dan proses yang terjadi dalam mengakses data sensor berdasarkan pengguna digambarkan pada Gambar 4.4.

Tabel 4.8 Akses Data Sensor Berdasarkan Pengguna

|  |  |  |
| --- | --- | --- |
| *Request* GET /sensordatas/ | | |
| Deskripsi | Parameter | Keterangan |
| Menampilkan data sensor dari semua perangkat node yang dimiliki pengguna. | * + *start*   + *end*   + *page* | * + parameter *start* dan *end* bersifat opsional.   + parameter *page* menentukan nomor halaman yang diminta.   + Jika tidak didefinisikan parameter *page* memiliki nilai sama dengan 1. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *count* | integer | Jumlah data sensor yang didapat berdasarkan kriteria tertentu. |
| *next* | string atau null | URL dari halaman selanjutnya. |
| *previous* | string atau null | URL dari halaman sebelumnya. |
| *results* | array dari dokumen Sensordatas | Sejumlah maksimal 10 data sensor pada halaman saat ini. |



Gambar 4.4 Alur Mengakses Data Sensor Berdasarkan Pengguna

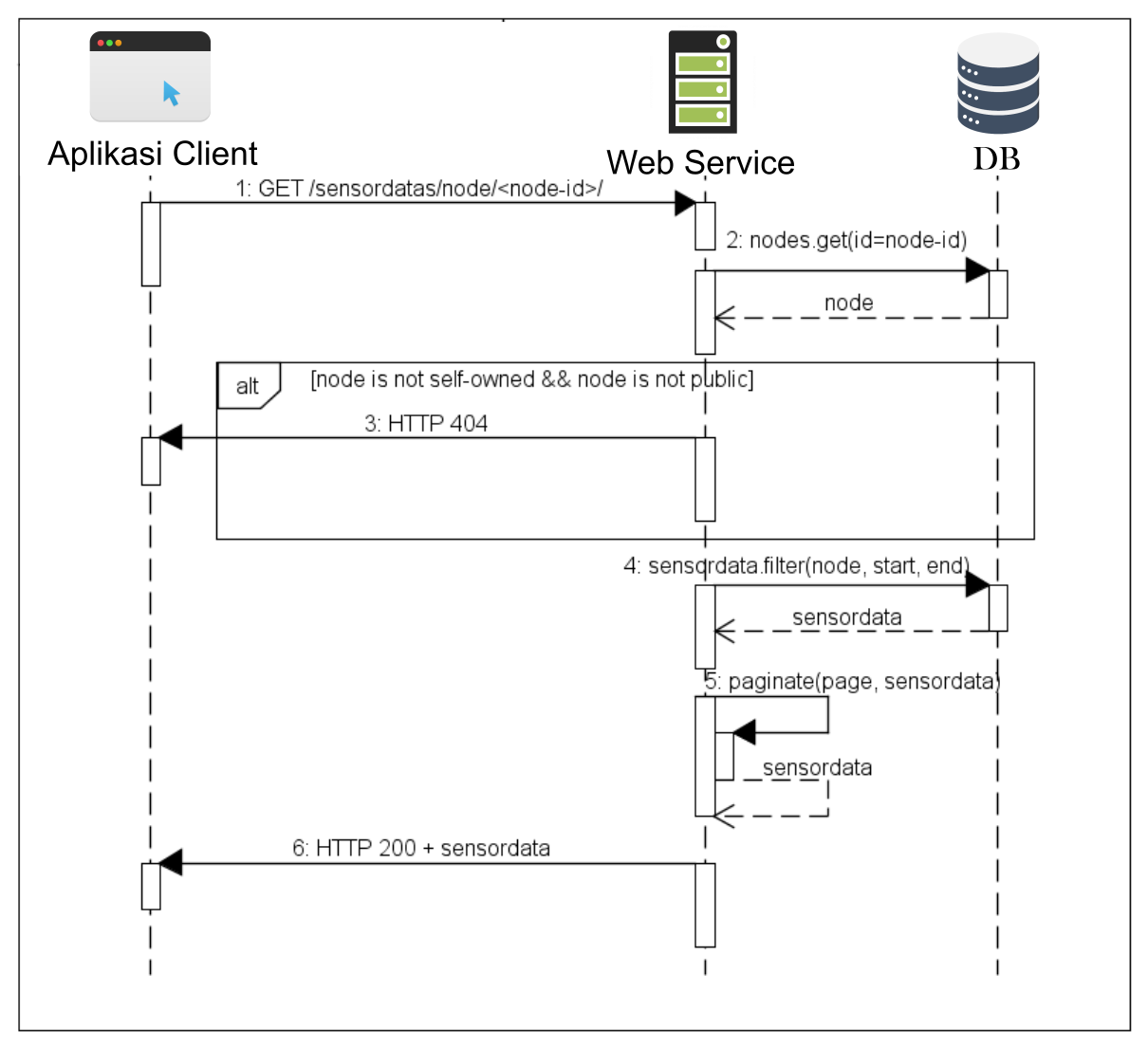
Penjelasan dari Gambar 4.4 adalah:

1. Aplikasi *client* mengirimkan *request* HTTP GET.
2. *Web service* memanggil fungsi nodes.filter(user=user-username) untuk mendapatkan semua perangkat node berdasarkan *username* pengguna. Setelah itu, DB mengembalikan data yang diminta dalam variabel nodes.
3. *Web service* memanggil fungsi sensordatas.filter(nodes, start, end) untuk mendapatkan data sensor dari semua perangkat node yang dimiliki pengguna. Parameter fungsi nodesdiambil dari proses nomor 2, sedangkan parameter startdan enddidapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP GET. Jika kedua parameter *query string* tersebut tidak dikirimkan maka parameter fungsi startdan end akan bernilai sama dengan null. Setelah fungsi sensordata.filter() dieksekusi, DB mengembalikan data yang diminta dalam variabel sensordata.
4. *Web service* memanggil fungsi paginate(page, sensordata) untuk membagi data berdasarkan halaman yang diminta. Parameter fungsi pagedidapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP GET, sedangkan parameter fungsi sensordata diambil dari proses nomor 3. Jika parameter *query string* page tidak dikirimkan maka parameter fungsi page akan bernilai sama dengan 1. Setelah fungsi paginate() dieksekusi, DB mengembalikan data yang diminta dalam variable sensordata.
5. *Web service* mengirimkan HTTP status 200 beserta data sensor sesuai dengan kriteria dan halaman yang diminta.
6. Berdasarkan Perangkat Node

Penjelasan mengenai parameter GET dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.9. Sedangkan alur dan proses yang terjadi dalam mengakses data sensor berdasarkan perangkat node digambarkan pada Gambar 4.5.

Tabel 4.9 Akses Data Sensor Berdasarkan Perangkat Node

|  |  |  |
| --- | --- | --- |
| *Request* GET /sensordatas /node/<node-id>/ | | |
| Deskripsi | Parameter | Keterangan |
| Menampilkan data sensor yang disortir berdasarkan perangkat node tertentu. | * + *start*   + *end*   + *page* | * + parameter *start* dan *end* bersifat opsional.   + parameter *page* menentukan nomor halaman yang diminta.   + Jika tidak didefinisikan parameter *page* memiliki nilai sama dengan 1. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *count* | integer | Jumlah data sensor yang didapat berdasarkan kriteria tertentu. |
| *next* | string atau null | URL dari halaman selanjutnya. |
| *previous* | string atau null | URL dari halaman sebelumnya. |
| *results* | array dari dokumen Sensordatas | Sejumlah 10 data sensor pada halaman saat ini. |



Gambar 4.5 Alur Mengakses Data Sensor Berdasarkan Perangkat Node

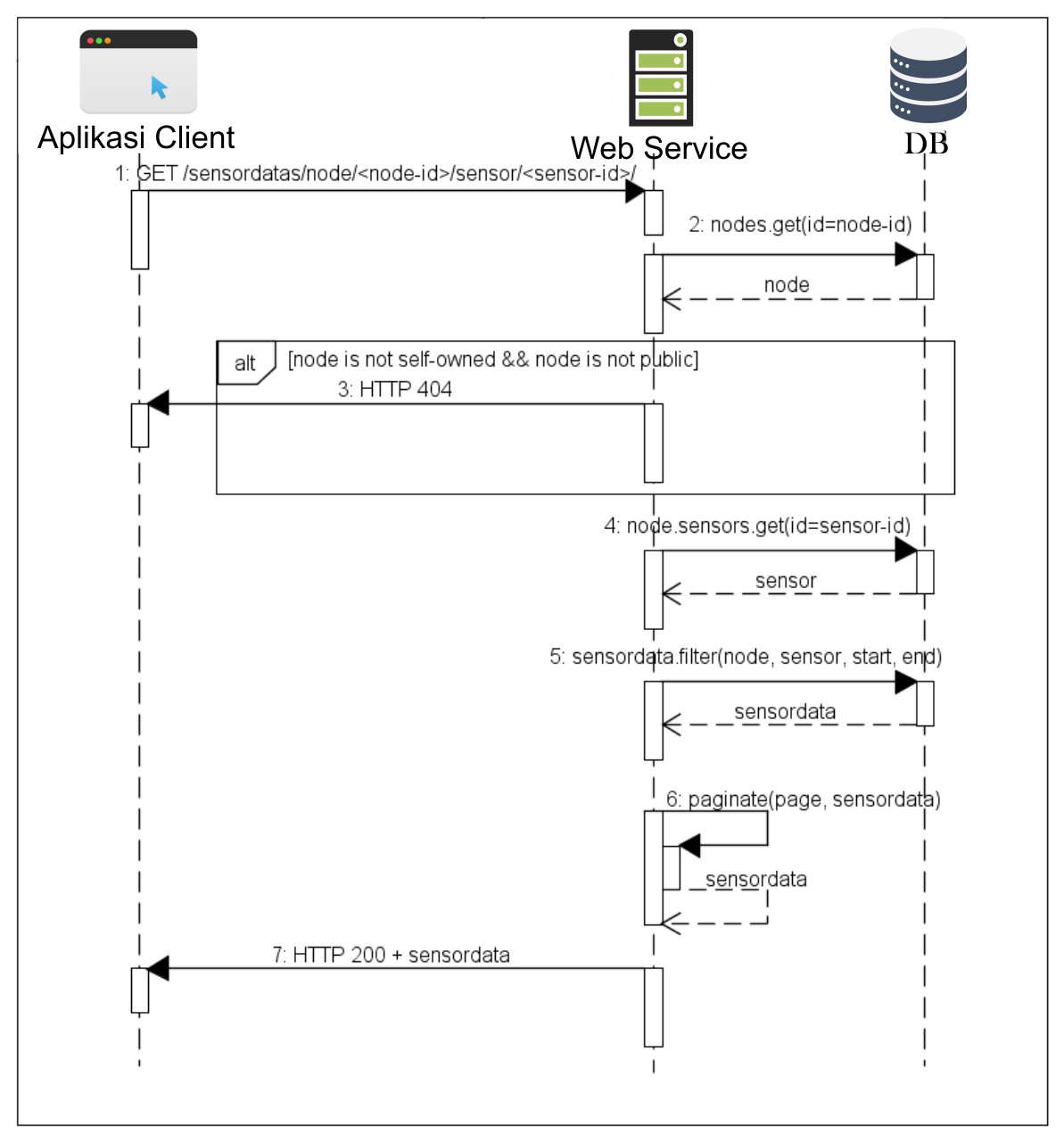
Penjelasan dari Gambar 4.5 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP GET.
2. *Web service* memanggil fungsi nodes(node-id) untuk mendapatkan data perangkat node berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel node.
3. Jika perangkat node yang diakses bukan milik pengguna dan perangkat node tersebut tidak memiliki label visibilitas publik, *web service* akan mengembalikan HTTP status 404.
4. Jika kondisi nomor 3 tidak terpenuhi, *web service* memanggil fungsi sensordatas(node, start, end) untuk mendapatkan data sensor dari perangkat node tersebut. Parameter fungsi nodediambil dari proses nomor 2, sedangkan parameter start dan enddidapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP GET. Jika kedua parameter *query string* tersebut tidak dikirimkan maka parameter fungsi startdan end akan bernilai sama dengan null. Setelah fungsi sensordata() dieksekusi, DB mengembalikan data yang diminta dalam variable sensordata.
5. *Web service* memanggil fungsi paginate(page, sensordata) untuk membagi data berdasarkan halaman yang diminta. Parameter fungsi pagedidapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP GET, sedangkan parameter fungsi sensordata diambil dari proses nomor 3. Jika parameter *query string* page tidak dikirimkan maka parameter fungsi page akan bernilai sama dengan 1. Setelah fungsi paginate() dieksekusi, DB mengembalikan data yang diminta dalam variable sensordata.
6. *Web service* mengirimkan HTTP status 200 beserta data sensor sesuai dengan kriteria dan halaman yang diminta.
7. Berdasarkan Sensor

Penjelasan mengenai parameter GET dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.10. Sedangkan alur dan proses yang terjadi dalam mengakses data sensor berdasarkan sensor digambarkan pada Gambar 4.6.

Tabel 4.10 Akses Data Sensor Berdasarkan Sensor

|  |  |  |
| --- | --- | --- |
| *Request* GET / sensordatas /node/<node-id>/sensor/<sensor-id>/ | | |
| Deskripsi | Parameter | Keterangan |
| Menampilkan data sensor yang disortir berdasarkan perangkat node dan sensor tertentu. | * + *start*   + *end*   + *page* | * + parameter *start* dan *end* bersifat opsional.   + parameter *page* menentukan nomor halaman yang diminta.   + Jika tidak didefinisikan parameter *page* memiliki nilai sama dengan 1. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *count* | integer | Jumlah data sensor yang didapat berdasarkan kriteria tertentu. |
| *next* | string atau null | URL dari halaman selanjutnya. |
| *previous* | string atau null | URL dari halaman sebelumnya. |
| *results* | array dari dokumen Sensordatas | Sejumlah 10 data sensor pada halaman saat ini. |



Gambar 4.6 Alur Mengakses Data Sensor Berdasarkan Sensor

Penjelasan dari Gambar 4.6 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP GET.
2. *Web service* memanggil fungsi nodes(node-id) untuk mendapatkan data perangkat node berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel node.
3. Jika perangkat node yang diakses bukan milik pengguna dan perangkat node tersebut tidak memiliki label visibilitas publik, *web service* akan mengembalikan HTTP status 404.
4. Jika kondisi nomor 3 tidak terpenuhi, *web service* memanggil fungsi sensordatas(node, start, end) untuk mendapatkan data sensor dari perangkat node tersebut. Parameter fungsi nodediambil dari proses nomor 2, sedangkan parameter startdan enddidapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP GET. Jika kedua parameter *query string* tersebut tidak dikirimkan maka parameter fungsi startdan end akan bernilai sama dengan null. Setelah fungsi sensordata() dieksekusi, DB mengembalikan data yang diminta dalam variabel sensordata.
5. *Web service* memanggil fungsi paginate(page, sensordata) untuk membagi data berdasarkan halaman yang diminta. Parameter fungsi pagedidapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP GET, sedangkan parameter fungsi sensordata diambil dari proses nomor 3. Jika parameter *query string* page tidak dikirimkan maka parameter fungsi page akan bernilai sama dengan 1. Setelah fungsi paginate() dieksekusi, DB mengembalikan data yang diminta dalam variable sensordata.
6. *Web service* mengirimkan HTTP status 200 beserta data sensor sesuai dengan kriteria dan halaman yang diminta.

### Perancangan Komponen *Secutity*

Perancangan komponen sekuritas dibagi menjadi dua yakni perancangan otentikasi dan otorisasi; dan perancangan manajemen perangkat.

#### 4.3.3.1 Perancangan Otentikasi dan Otorisasi

Perancangan otentikasi dan otorisasi menggambarkan mengenai alur dalam mendapatkan token akses JWT, registrasi pengguna dan menerima data sensor yang dikirimkan perangkat node.

1. Mendapatkan token akses JWT

Perangkat node dan aplikasi *client* memiliki token akses JWT yang berbeda. Kedua token tersebut berisi informasi *credentials* mewakili otorisasi yang berbeda pula. Token perangkat node digunakan untuk pengiriman data sensor ke IoT *cloud platform* sedangkan token aplikasi *client* digunakan untuk melihat data sensor dan memanajemen perangkat.

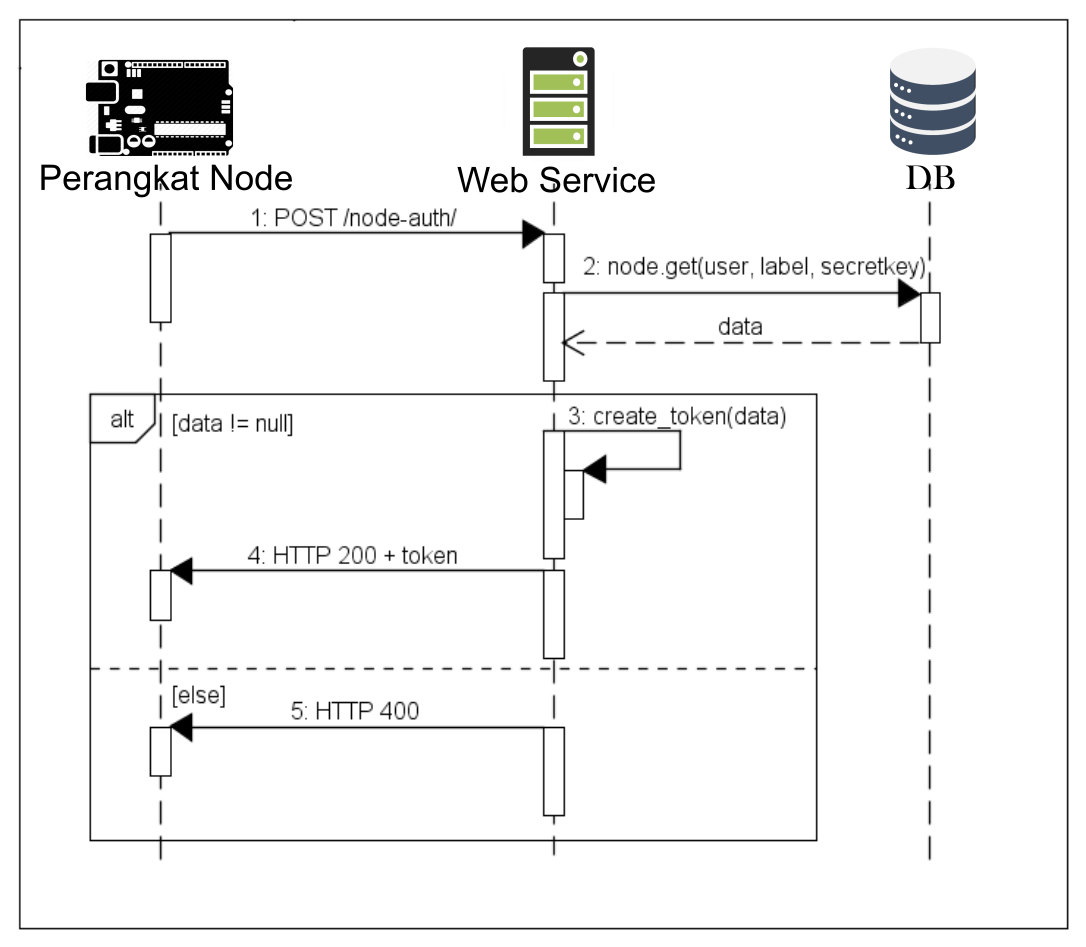
* Perangkat node

Untuk mendapatkan token akses JWT perangkat node perlu mengirimkan HTTP *request* dengan *payload* berisi *user*, *label*, dan *secretkey*. *Payload* tersebut berisi tiga *field* utama pada dokumen Nodes untuk mendapatkan informasi perangkat node yang dimaksud. Informasi tersebut kemudian di-*generate* menjadi *credentials* yang mewakili otoritas pada token akses yang dihasilkan.

Penjelasan mengenai *payload* dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.11. Sedangkan alur dan proses yang terjadi dalam mengakses data sensor berdasarkan sensor digambarkan pada Gambar 4.7.

Tabel 4.11 Perangkat Node Mendapatkan Token Akses JWT

|  |  |  |
| --- | --- | --- |
| *Request* POST /node-auth/ | | |
| Deskripsi | *Payload* | Keterangan |
| Men-*generate* token akses JWT untuk perangkat node. | * + *user*   + *label*   + *secretkey* | * + *payload user*, *label*, dan *secretkey* dibutuhkan.   + *payload* *user* merupakan *username* dari akun pengguna.   + *payload label* merupakan label dari perangkat node.   + *payload* *secretkey* merupakan *secretkey* dari perangkat node. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *node* | *object* dari dokumen nodes | Data perangkat node tersebut. |
| *token* | string | Akses token JWT. |



Gambar 4.7 Alur Perangkat Node Mendapatkan Token Akses JWT

Penjelasan dari Gambar 4.7 adalah sebagai berikut:

1. Perangkat nodemengirimkan *request* HTTP POST dengan *payload* berisi *user*, *label*, dan *secretkey*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.10.
2. *Web service* memanggil fungsi nodes.get(user, label, secretkey) untuk mendapatkan data perangkat node. Kemudian, DB mengembalikan data yang diminta dalam variabel data.
3. Jika variable *data* tidak berisi null, *web service* memanggil fungsi create\_token() dengan parameter fungsi datauntuk meng-*generate* token.
4. *Web service* mengirimkan *response* dengan HTTP status 200 beserta data perangkat node tersebut dan token akses JWT-nya. Informasi lebih lengkap mengenai informasi *response* dapat dilihat pada Tabel 4.11.
5. Jika variable databerisi null, *web service* mengirimkan *response* dengan HTTP status 400.

* Akses token JWT untuk aplikasi *client*

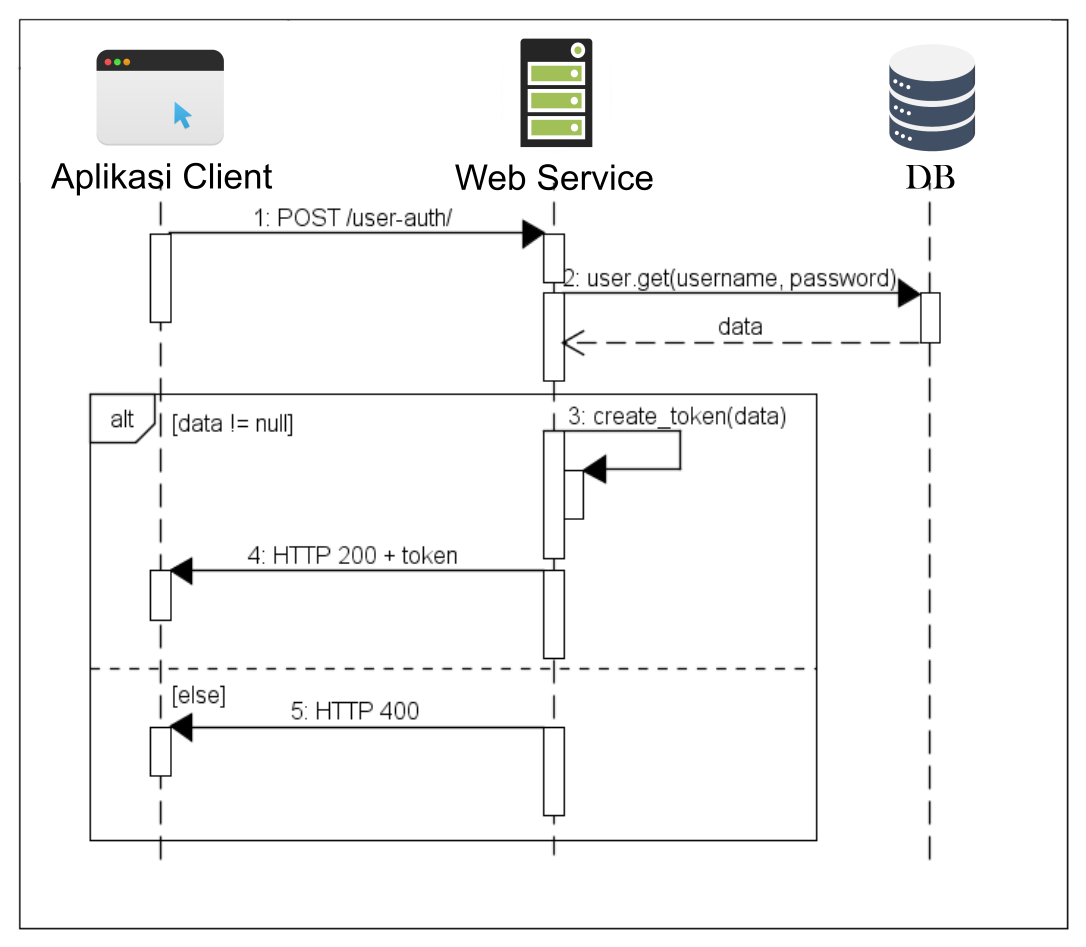
Aplikasi *client* merupakan perangkat lunak multi-platform yang berkomunikasi dengan IoT *cloud platform* melalui RESTful *web service*. Pengguna adalah pihak yang mengoperasikan aplikasi *client* sehingga token akses JWT yang digunakan mewakili *credentials* pengguna itu sendiri.

Untuk mendapatkan token akses, aplikasi *client* perlu mengirimkan HTTP *request* dengan *payload* berisi *username*, dan *password*. *Payload* tersebut berisi dua *field* utama pada dokumen Users yang diperlukan untuk mendapatkan informasi pengguna yang dimaksud. Informasi tersebut kemudian di-*generate* menjadi *credentials* yang mewakili otoritas pada token akses yang dihasilkan.

Penjelasan mengenai *payload* dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.12. Sedangkan alur dan proses yang terjadi dalam mengakses data sensor berdasarkan sensor digambarkan pada Gambar 4.8.

Tabel 4.12 Aplikasi *Client* Mendapatkan Token Akses JWT

|  |  |  |
| --- | --- | --- |
| *Request* POST /user-auth/ | | |
| Deskripsi | *Payload* | Keterangan |
| Men-*generate* token akses JWT untuk aplikasi *client*. | * + *username*   + *password* | * + *payload username* dan *password* dibutuhkan.   + *payload* *username* merupakan *username* dari akun pengguna.   + *password* merupakankata sandidari akun pengguna. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *user* | *object* dari dokumen Users | Data pengguna. |
| *token* | string | Akses token JWT. |



Gambar 4.8 Alur Aplikasi *Client* Mendapatkan Token Akses JWT

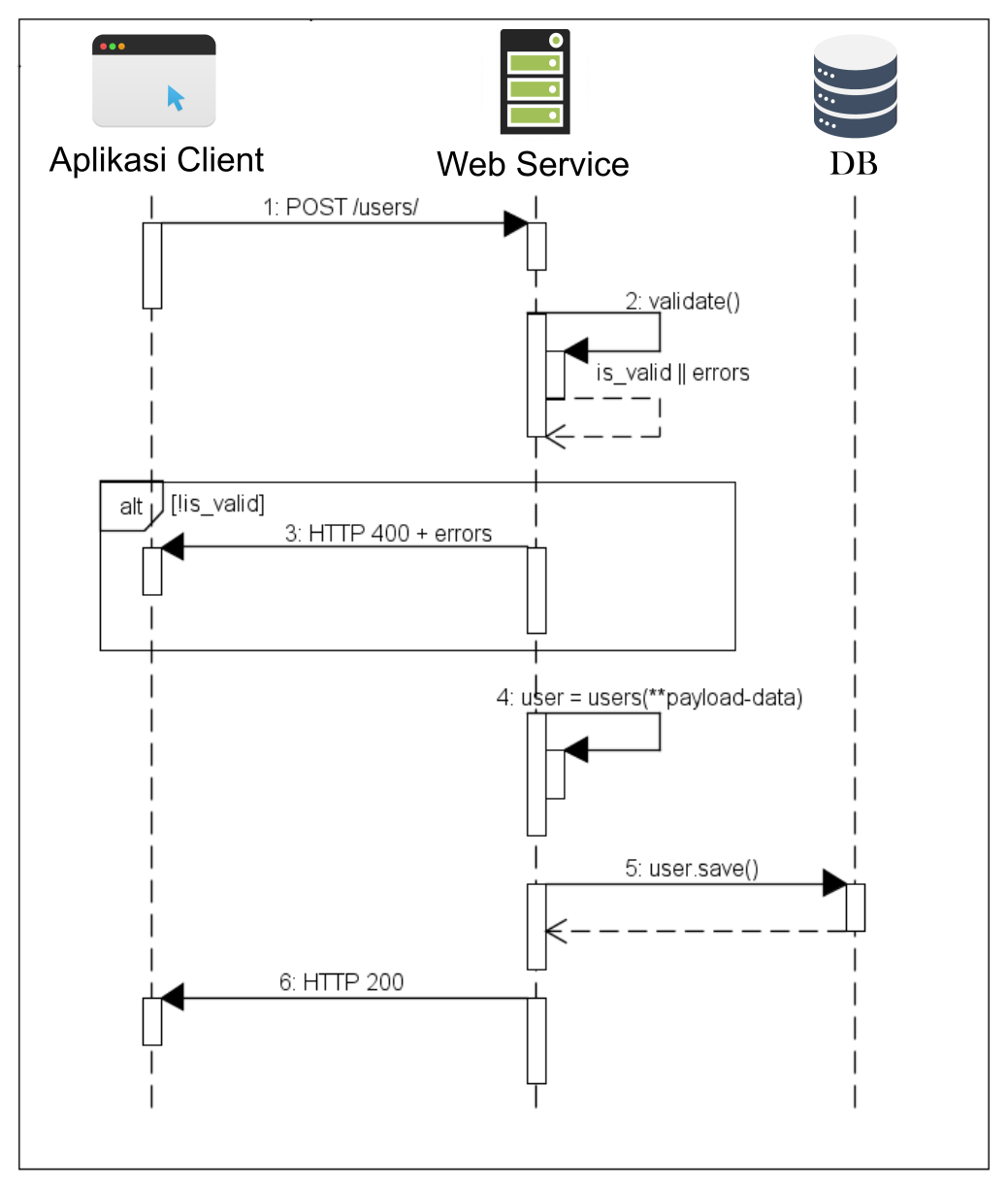
Penjelasan dari Gambar 4.8 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP POST dengan *payload* berisi *username*, dan *password*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.12.
2. *Web service* memanggil fungsi users.get(username, password) untuk mendapatkan data perngguna. Kemudian, DB mengembalikan data yang diminta dalam variabel data.
3. Jika variable *data* tidak berisi null, *web service* memanggil fungsi create\_token() dengan parameter fungsi *data* untuk meng-*generate* token.
4. *Web service* mengirimkan *response* dengan HTTP status 200 beserta data pengguna tersebut dan token akses JWTnya. Informasi lebih lengkap mengenai informasi *response* dapat dilihat pada Tabel 4.13.
5. Jika variable databerisi null, w*eb service* mengirimkan *response* dengan HTTP status 400.
6. Registrasi Pengguna

Penjelasan mengenai *payload* dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.13. Sedangkan alur dan proses yang terjadi dalam registrasi pengguna digambarkan pada Gambar 4.9.

Tabel 4.13 Registrasi Pengguna

|  |  |  |
| --- | --- | --- |
| *Request* POST /users/ | | |
| Deskripsi | *Payload* | Keterangan |
| Registrasi pengguna. | * + *username*   + *first\_name*   + *last\_name*   + *password*   + *email*   + *password* | * + semua *payload username* dan *password* dibutuhkan. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *user* | *object* dari dokumen Users | Data pengguna yang dibuat. |



Gambar 4.9 Alur Registrasi Pengguna

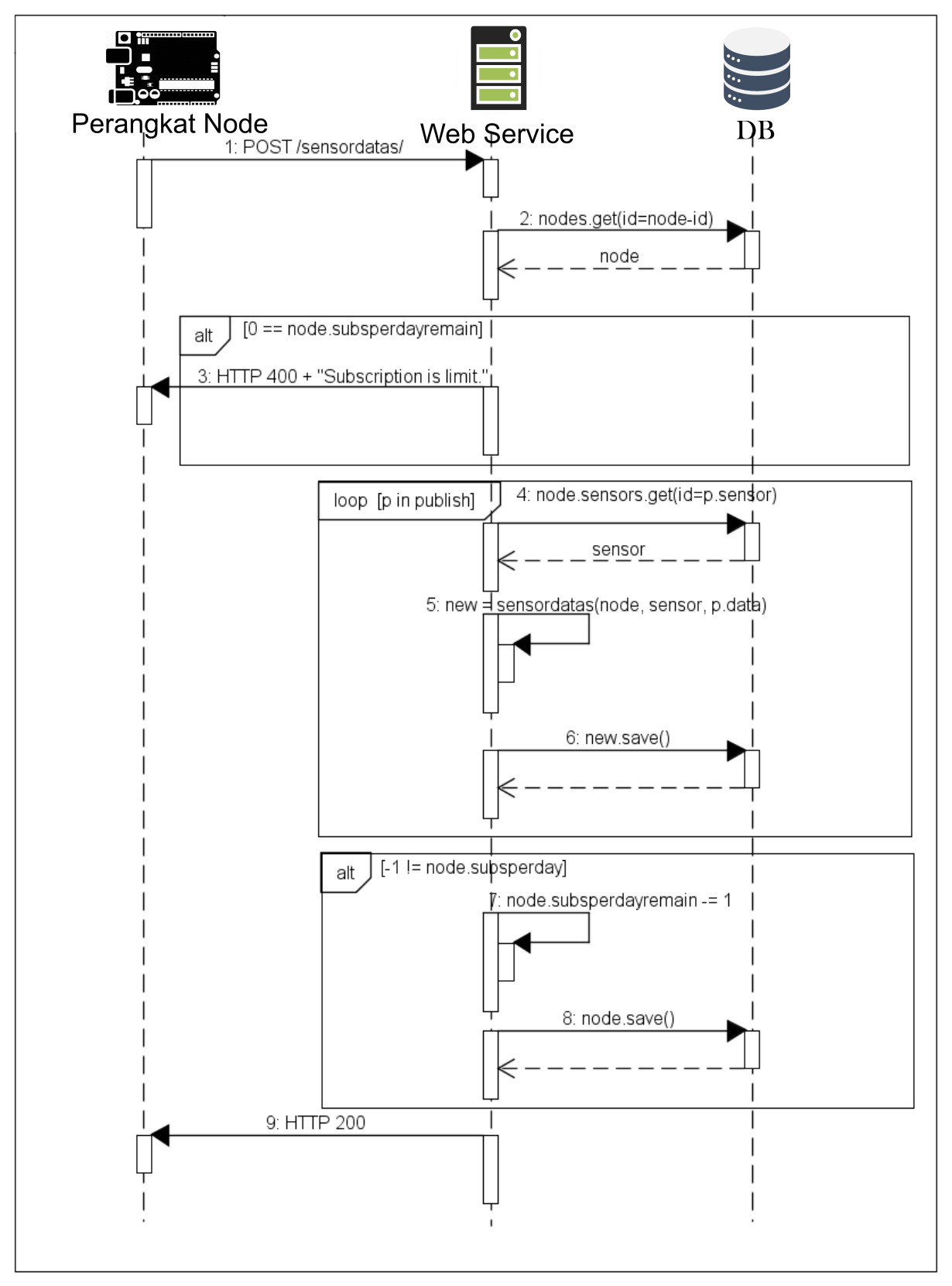
Penjelasan dari Gambar 4.9 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP POST dengan *payload* berisi *username*, *first\_name*, *last\_name*, *email*,dan *password*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.13.
2. *Web service* memanggil fungsi validate() untuk memastikan *payload* yang dikirim sesuai dengan yang dibutuhkan. Fungsi ini juga memeriksa keunikan *field username* dari pengguna baru terhadap pengguna lainnya.
3. Jika proses validasi gagal, *web service* mengirimkan *response* dengan HTTP status 400 dan informasi mengenai kegagalan validasi pada *erors*.
4. *Web service* memanggil fungsi users(\*\*payload-data) untuk membuat *object* baru dari dokumen Users. *Object* baru tersebut disimpan dalam variabel user.
5. *Web service* memanggil fungsi user.save() untuk menyimpan *object* baru pada nomor 4 ke dalam basis data.
6. *Web service* mengirimkan *response* dengan HTTP status 200.
7. Menerima data sensor dari perangkat node

Penjelasan mengenai *payload* dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.14. Sedangkan alur dan proses yang terjadi dalam menerima data sensor dari perangkat node digambarkan pada Gambar 4.10.

Tabel 4.14 Menerima Data Sensor dari Perangkat Node

|  |  |  |
| --- | --- | --- |
| *Request* POST /sensordatas/ | | |
| Deskripsi | *Payload* | Keterangan |
| Mengirimkan data sensor dari perangkat node dilapangan. | * + *publish*[]   + *testing* | * + *payload publish* dibutuhkan.   + *payload testing bersifat opsional*.   + *payload* *publish* merupakan array yang menyimpan data bertipe *object*. Setiap *object* berisi *field* sensor dan data.   + *payload testing* merupakan *field* bernilai *boolean*. Digunakan untuk mengecek format *payload* dalam pengiriman, tanpa meyimpan data sensor ke dalam basis data. |
| Response | | |
| Nama *field* | Tipe data | Keterangan |
| *results* | *array berisi object* dari dokumen sensordatas. | List data sensor yang telah dibuat. |

****

Gambar 4.10 Alur Menerima Data Sensor dari Perangkat Node

Penjelasan dari Gambar 4.10 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP POST dengan *payload* berisi *publish*, dan *testing*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.14.
2. *Web service* memanggil fungsi nodes.get(id=node-id) untuk mendapatkan data perangkat node. Kemudian, DB mengembalikan data yang diminta dalam variabel node.
3. Jika pubsperdayremain dari perangkat node sudah habis atau sama dengan nol. *Request* dibatalkan dan *web service* mengirimkan *response* dengan HTTP status 400.

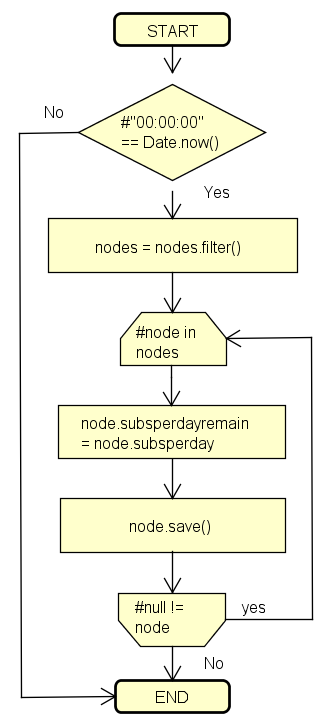
Nomor 4, 5, dan 6 berada dalam iterasi sebanyak *array items* pada *field* publish yang terdapat pada *payload*. Pada setiap iterasi, *item* pada *field* *publish* dapat diakses menggunakan variabel p.

1. *Web service* memanggil fungsi nodes.sensors.get(id=p.sensor) untuk mendapatkan data sensor. Kemudian, DB mengembalikan data yang diminta dalam variabel sensor.
2. *Web service* memanggil fungsi sensordatas(node, sensor, p.data) untuk membuat *object* baru dari dokumen Sensordatas. *Object* baru tersebut disimpan dalam variabel new.
3. *Web service* memanggil fungsi new.save() untuk menyimpan *object* baru pada nomor 5 ke dalam basis data.

Nomor 7 dan 8 berada dalam kondisi saat perangkat node tidak memiliki pembatasan pengiriman data sensor.

1. Nilai pubsperdayremain dari perangkat node dikurangi satu.
2. Perubahan data perangkat node pada nomor 7 disimpan ke dalam basis data.
3. *Web service* mengirimkan *response* dengan HTTP status 200.
4. Mengatur ulang bilangan *counter* pembatasan pengiriman semua perangkat node.

Fitur pembatasan pengiriman data sensor per hari bagi perangkat node memerlukan satu bilangan *counter* yang akan berkurang satu setiap kali perangkat mengirimkan data. Bilangan *counter* tersebut harus diatur ulang pada hari berikutnya agar perangkat node dapat kembali mengirimkan datanya. Untuk itu diperlukan suatu layanan untuk melakukannya setiap pukul 12 malam. Dibawah ini digambarkan alur dalam mengatur ulang bilangan *counter* ada pada gambar Gambar 4.11.



Gambar 4.11 Flowchart Mengatur Ulang Bilangan *Counter* Pembatasan Pengiriman Semua Perangkat Node

#### 4.3.3.2 Perancangan Manajemen Perangkat

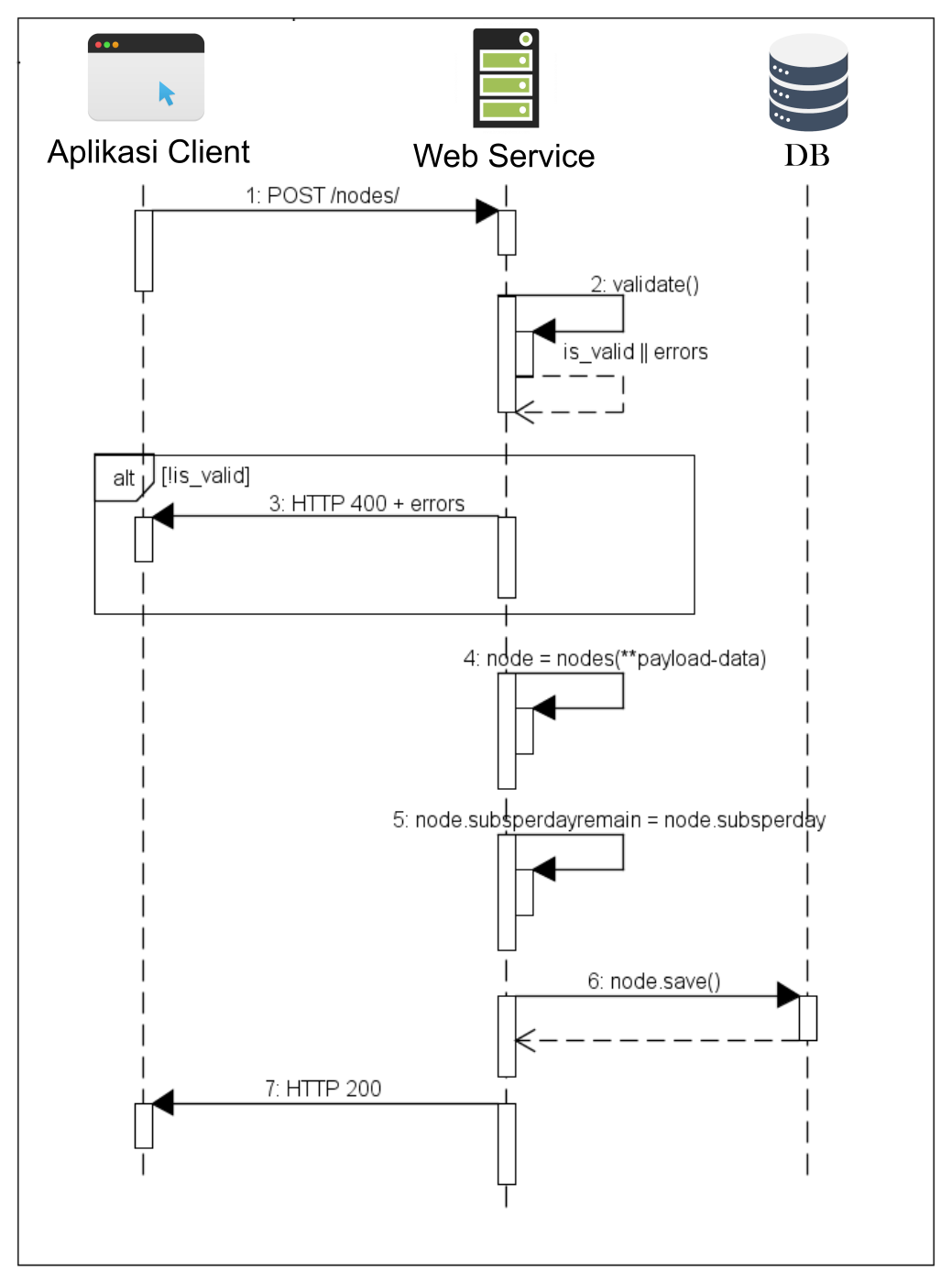
Perancangan manajemen perangkat menjelaskan alur aplikasi *client* dalam memanajemen perangkat.

1. Membuat perangkat node

Penjelasan mengenai *payload* dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.15. Sedangkan alur dan proses yang terjadi dalam membuat perangkat node digambarkan pada Gambar 4.12.

Tabel 4.15 Membuat Perangkat Node

|  |  |  |
| --- | --- | --- |
| *Request* POST /nodes/ | | |
| Deskripsi | *Payload* | Keterangan |
| Mendaftarkan perangkat node baru yang dimiliki pengguna. | * + *user*   + *label*   + *secretkey*   + *pubsperday*   + *is\_public* | * + semua *payload* dibutuhkan.   + *payload user* berisi username pengguna.   + *payload pubsperday* berisi bilangan bulat untuk membatasi pengiriman data sensor. Nilai -1 berarti pengiriman tidak dibatasi.   + *payload* *is\_public* berisi bilangan bulat yang menunjukkan visibilitas perangkat terhadap pengguna lain. Nilai 1 berarti perangkat node bersifat publik, sedangkan nilai 0 berarti bersifat privat. |
| Response | | |
| Nama *field* | Tipe data | Keterangan |
| *results* | dokumen nodes. | Perangkat node yang dibuat. |



Gambar 4.12 Alur Membuat Perangkat Node

Penjelasan dari Gambar 4.12 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP POST dengan *payload* berisi user, *label*, *secretkey*, *subsperday*, dan *is­\_public*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.16.
2. *Web service* memanggil fungsi validate() untuk memastikan *payload* yang dikirim sesuai dengan yang dibutuhkan. Fungsi ini juga memeriksa keunikan *field* label dari perangkat node baru terhadap perangkat node lainnya milik pengguna.
3. Jika proses validasi gagal, *web service* mengirimkan *response* dengan HTTP status 400 dan informasi mengenai kegagalan validasi pada *erors*.
4. *Web service* memanggil fungsi nodes(\*\*payload-data) untuk membuat *object* baru dari dokumen Nodes. *Object* baru tersebut disimpan dalam variabel node.
5. Nilai pubsperdayremain diatur dengan nilai awal sama dengan pubsperday.
6. *Web service* memanggil fungsi node.save() untuk menyimpan *object* baru pada nomor 4 ke dalam basis data.
7. *Web service* mengirimkan *response* dengan HTTP status 200.
8. Melihat perangkat node

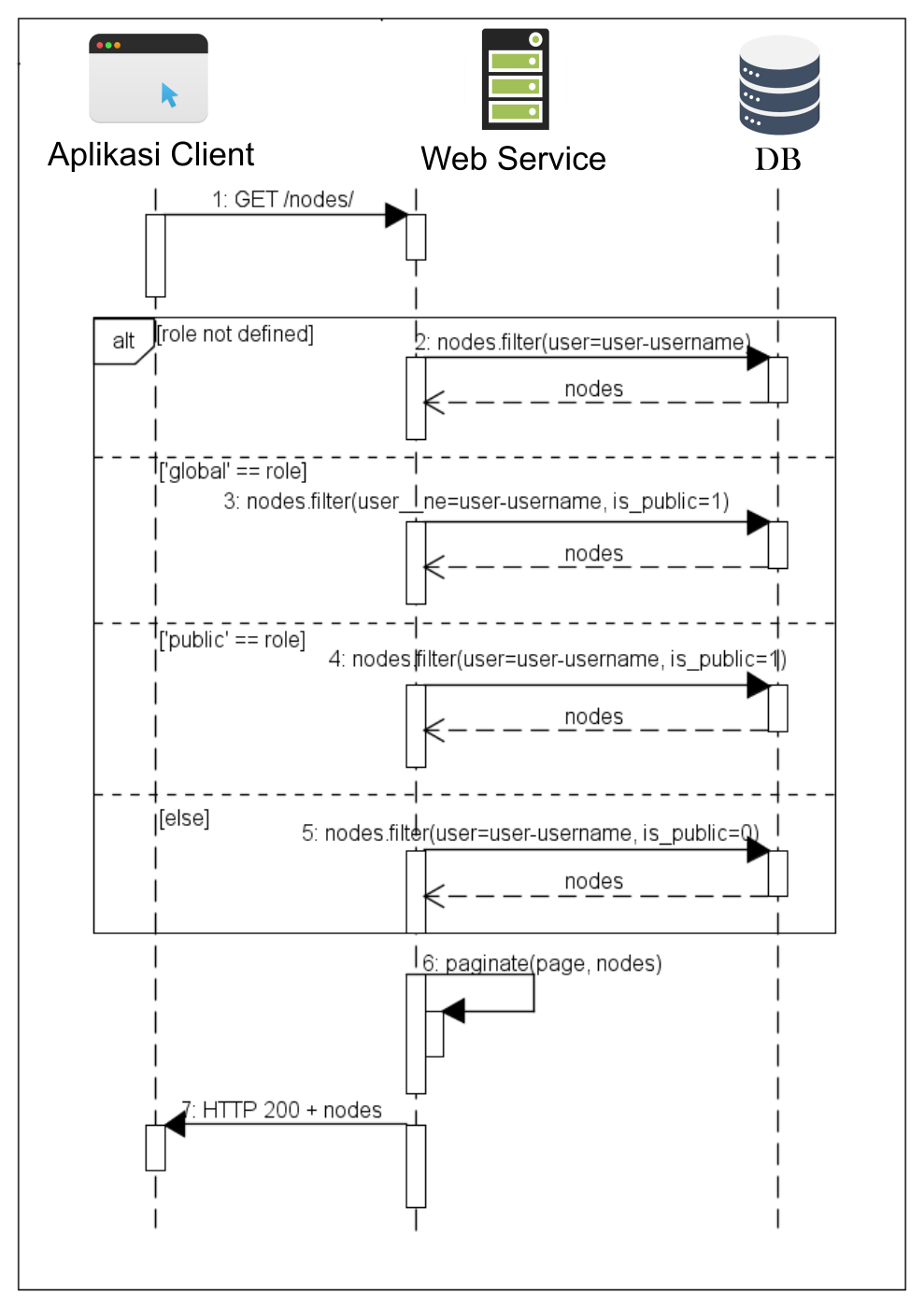
Terdapat tiga kriteria untuk mengakses perangkat node berdasarkan visibilitas perangkat. Kriteria ditentukan oleh parameter HTTP GET *role* yang dapat berisi nilai: “*public”*, *“private”*, dan “*global”*.Nilai “*global”* digunakan untuk mengakses perangkat node dari pengguna lain yang memiliki visibilitas publik. Jika parameter *role* tidak didefinisikan, *web service* akan mengembalikan semua perangkat node milik pengguna yang sedang terotentikasi.

Dalam melihat perangkat node digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman. Setiap HTTP *request* akan memberikan HTTP *response* berupa halaman dengan maksimal berisi 10 data di dalamnya. Untuk menggunakan fitur *pagination*, dibutuhkan parameter HTTP GET tambahan yaitu *page* untuk menentukan halaman yang ingin diakses. Parameter *page* merupakan bilangan integer dengan nilai *default* sama dengan 1.

Penjelasan mengenai parameter GET dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.16. Sedangkan alur dan proses yang terjadi dalam melihat semua perangkat node digambarkan pada Gambar 4.13.

Tabel 4.16 Melihat Perangkat Node

|  |  |  |
| --- | --- | --- |
| *Request* GET /nodes/ | | |
| Deskripsi | Parameter | Keterangan |
| Menampilkan perangkat node. | * + *role*   + *page* | * + parameter *role* dan *page* bersifat opsional.   + parameter *page* menentukan nomor halaman yang diminta.   + Jika tidak didefinisikan parameter *page* memiliki nilai sama dengan 1. |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| *count* | integer | Jumlah perangkat node yang didapat berdasarkan kriteria tertentu. |
| *next* | string atau null | URL dari halaman selanjutnya. |
| *previous* | string atau null | URL dari halaman sebelumnya. |
| *results* | array berisi *object* dari dokumen Sensordatas | Sejumlah 10 perangkat node pada halaman saat ini. |

****

Gambar 4.13 Alur melihat perangkat node

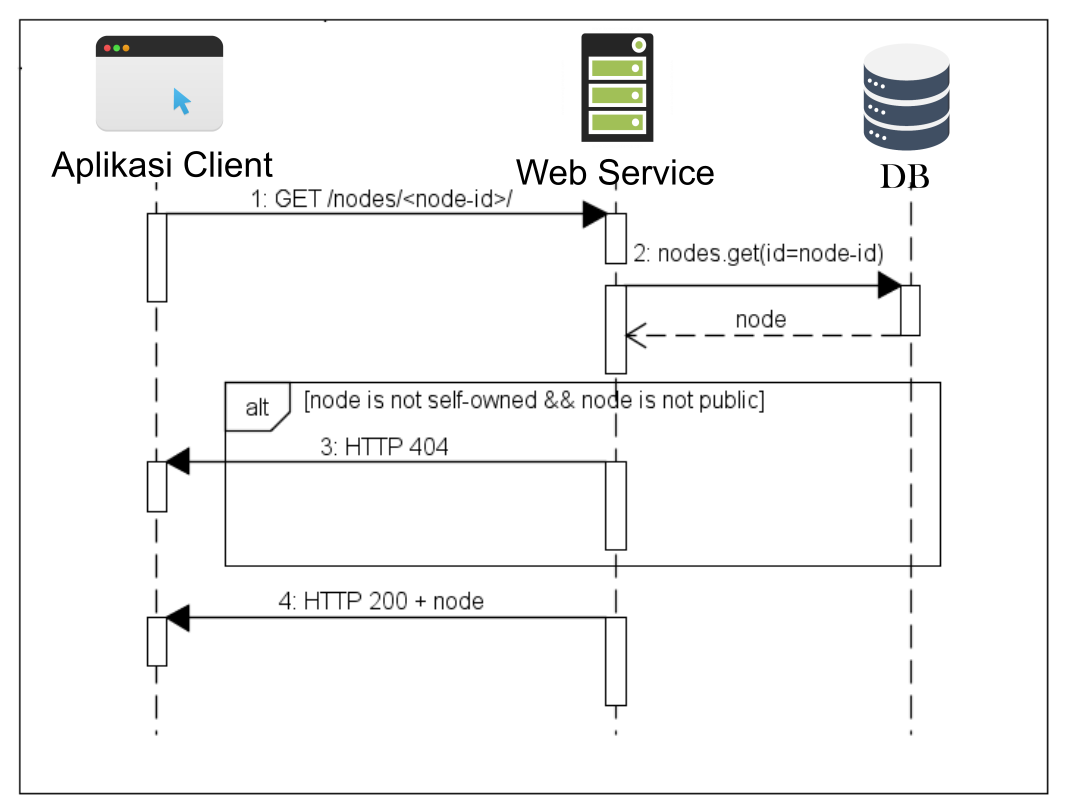
Penjelasan dari Gambar 4.13 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP GET dengan atau tanpa parameter *role*. Informasi lebih lengkap mengenai parameter *role* dapat dilihat pada Tabel 4.18.
2. Jika parameter roletidak didefinisikan, *web service* memanggil fungsi nodes.filter(user=user-username) untuk mendapatkan semua perangkat node milik pengguna. Setelah fungsi nodes.filter() dieksekusi, DB mengembalikan data yang diminta dalam variable nodes.
3. Jika parameter rolebernilai “*global*’, *web service* akan memanggil fungsi nodes.filter(user\_\_ne=user-username, is\_public=1) untuk mendapatkan semua perangkat node milik pengguna lain yang memiliki label visibilitas publik. Setelah fungsi nodes.filter() dieksekusi, DB mengembalikan data yang diminta dalam variable nodes.
4. Jika parameter rolebernilai “*public*”, *web service* akan memanggil fungsi nodes.filter(user=user-username,is\_public=1)untuk mendapat-kan semua perangkat node milik pengguna yang memiliki label visibilitas publik. Setelah fungsi nodes.filter() dieksekusi, DB mengembalikan data yang diminta dalam variable nodes.
5. Kondisi lainnya, *web service* akan memanggil fungsi nodes.filter(user=user-username,is\_public=0)untuk mendapat-kan semua perangkat node milik pengguna yang memiliki label visibilitas privat. Setelah fungsi nodes.filter() dieksekusi, DB mengembalikan data yang diminta dalam variable nodes.
6. *Web service* memanggil fungsi paginate(page, nodes) untuk membagi data berdasarkan halaman yang diminta. Setelah fungsi paginate() dieksekusi, DB mengembalikan data yang diminta dalam variable nodes.
7. *Web service* mengirimkan *response* dengan HTTP status 200 beserta *list* perangkat node sesuai dengan kriteria dan halaman yang diminta.
8. Melihat perangkat node berdasarkan id

Penjelasan mengenai parameter HTTP GET dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.17. Sedangkan alur dan proses yang terjadi dalam mengakses semua perangkat node digambarkan pada Gambar 4.14.

Tabel 4.17 Melihat Perangkat Node Berdasarkan Id

|  |  |  |
| --- | --- | --- |
| *Request* GET /nodes/<node-id>/ | | |
| Deskripsi | Parameter | Keterangan |
| Menampilkan perangkat node berdasarkan id. | - |  |
| *Response* | | |
| Nama *field* | Tipe data | Keterangan |
| Tidak ada (langsung berupa *object*) | *object* dari dokumen Nodes. | * + Perangkat node. |

****

Gambar 4.14 Alur melihat perangkat node berdasarkan id

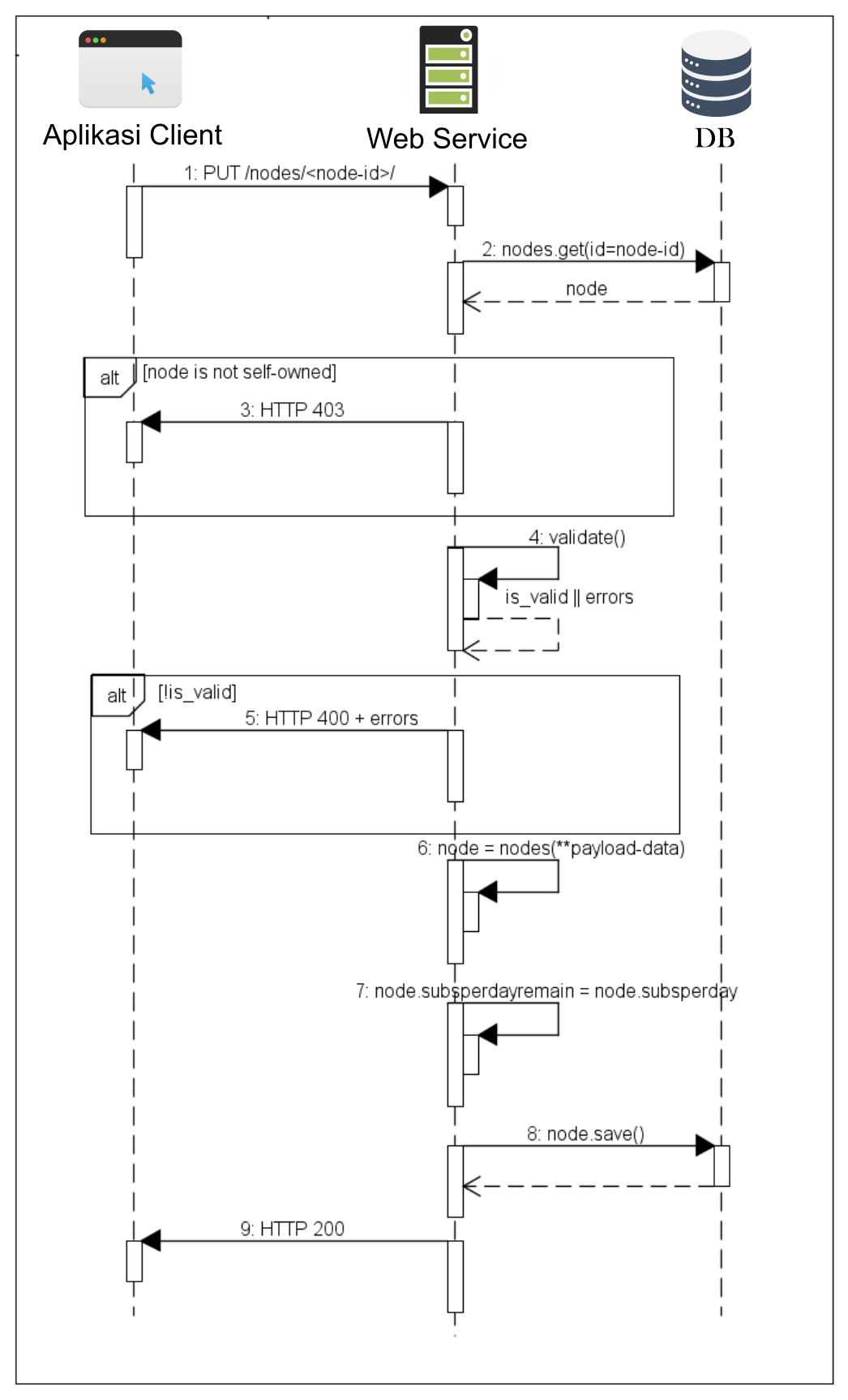
Penjelasan dari Gambar 4.14 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP GET.
2. *Web service* memanggil fungsi nodes.get(id=node-id) untuk mendapatkan data perangkat node berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel node.
3. Jika perangkat node yang diakses bukan milik pengguna dan perangkat node tersebut tidak memiliki label visibilitas publik, *web service* akan mengembalikan HTTP status 404.
4. *Web service* mengirimkan *response* dengan HTTP status 200 beserta data perangkat node yang diminta.
5. Mengubah perangkat node

Penjelasan mengenai *payload* dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.18. Sedangkan alur dan proses yang terjadi dalam mengubah perangkat node digambarkan pada Gambar 4.15.

Tabel 4.18 Mengubah Perangkat Node

|  |  |  |
| --- | --- | --- |
| *Request* PUT /nodes/<node-id>/ | | |
| Deskripsi | *Payload* | Keterangan |
| Mendaftarkan perangkat node baru yang dimiliki pengguna. | * + *user*   + *label*   + *secretkey*   + *subsperday*   + *is\_public* | * + *payload* bersifat opsional, kirim hanya bagianyang ingin dirubah. |
| Response | | |
| Nama *field* | Tipe data | Keterangan |
| Tidak ada (langsung berupa *object*) | *object* dari dokumen Nodes. | Perangkat node yang dirubah. |

****

Gambar 4.15 Alur Mengubah Perangkat Node

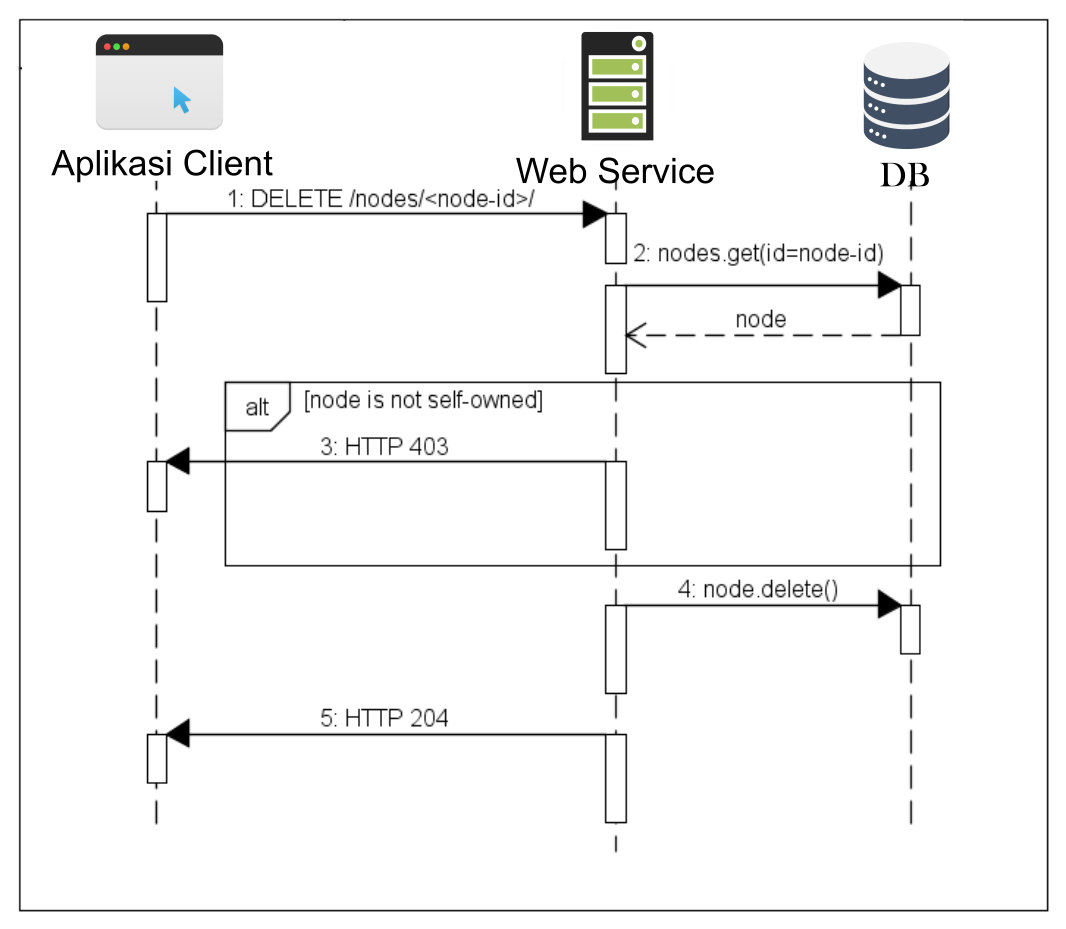
Penjelasan dari Gambar 4.15 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP PUT dengan *payload* berisi hanya *field* yang ingin dirubah. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.22.
2. *Web service* memanggil fungsi nodes.get(id=node-id) untuk mendapatkan data perangkat node berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel node.
3. Jika perangkat node yang diakses bukan milik pengguna, *web service* akan mengembalikan HTTP status 403.
4. *Web service* memanggil fungsi validate() untuk memastikan *payload* yang dikirim sesuai dengan yang dibutuhkan. Fungsi ini juga memeriksa keunikan *field* label dari perangkat node baru terhadap perangkat node milik pengguna yang sudah ada.
5. Jika proses validasi gagal, *web service* akan mengirimkan *response* dengan HTTP status 400 dan informasi mengenai kegagalan validasi pada *erors*.
6. *Web service* memanggil fungsi nodes(\*\*payload-data) untuk membuat *object* baru dari dokumen nodes. *Object* baru tersebut disimpan dalam variabel node.
7. Nilai pubsperdayremain diatur dengan nilai awal sama dengan pubsperday.
8. *Web service* memanggil fungsi node.save() untuk menyimpan *object* baru pada nomor 4 ke dalam basis data.
9. *Web service* mengirimkan *response* dengan HTTP status 200.
10. Menghapus perangkat node

Informasi dalam mengirimkan HTTP *request* dan informasi mengenai data yang didapatkan dari HTTP *response* ada pada Tabel 4.19. Sedangkan alur dan proses yang terjadi dalam menghapus perangkat node digambarkan pada Gambar 4.16.

Tabel 4.19 Menghapus Perangkat Node

|  |  |  |
| --- | --- | --- |
| *Request* DELETE /nodes/<node-id>/ | | |
| Deskripsi | *Payload* | Keterangan |
| Menghapus perangkat node berdasarkan id. | - |  |
| Response | | |
| Nama *field* | Tipe data | Keterangan |
| *-* | - | - |

****

Gambar 4.16 Alur Menghapus Perangkat Node

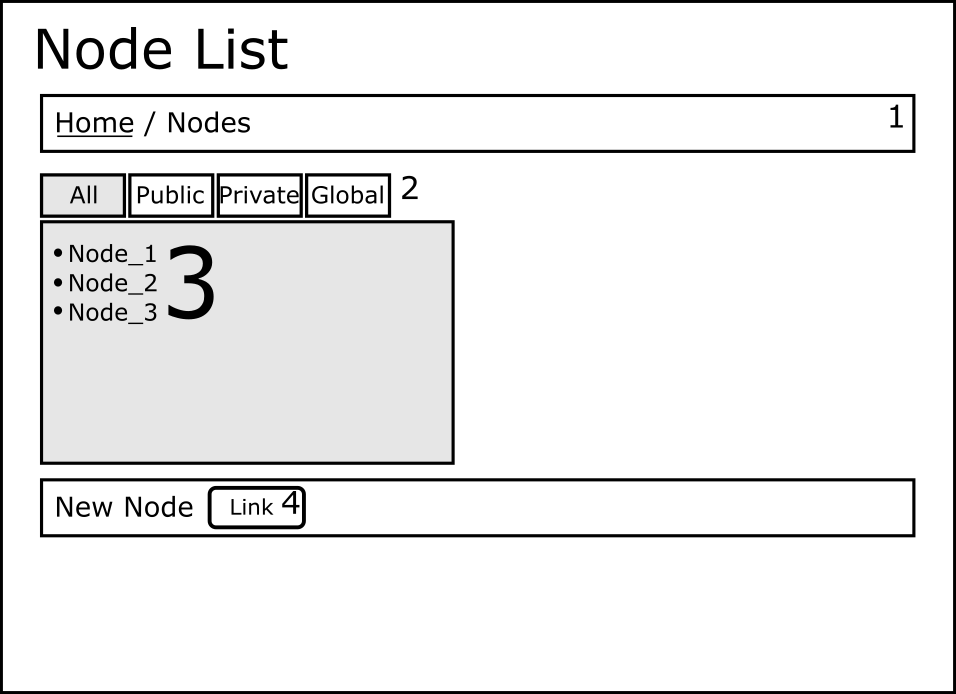
Penjelasan dari Gambar 4.16 adalah sebagai berikut:

1. Aplikasi *client* mengirimkan *request* HTTP GET.
2. *Web service* memanggil fungsi nodes.get(id=node-id) untuk mendapatkan data perangkat node berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel node.
3. Jika perangkat node yang diakses bukan milik pengguna, *web service* akan mengembalikan HTTP status 403.
4. *Web service* memanggil fungsi node.delete() untuk menghapus perangkat node di dalam basis data.
5. *Web service* mengirimkan *response* dengan HTTP status 200 beserta data perangkat node yang diminta.

### Perancangan Komponen *Web Console*

Perancangan komponen *web console* menjelaskan mengenai rancangan antarmuka pengguna untuk memanajemen perangkat dan melihat data sensor.

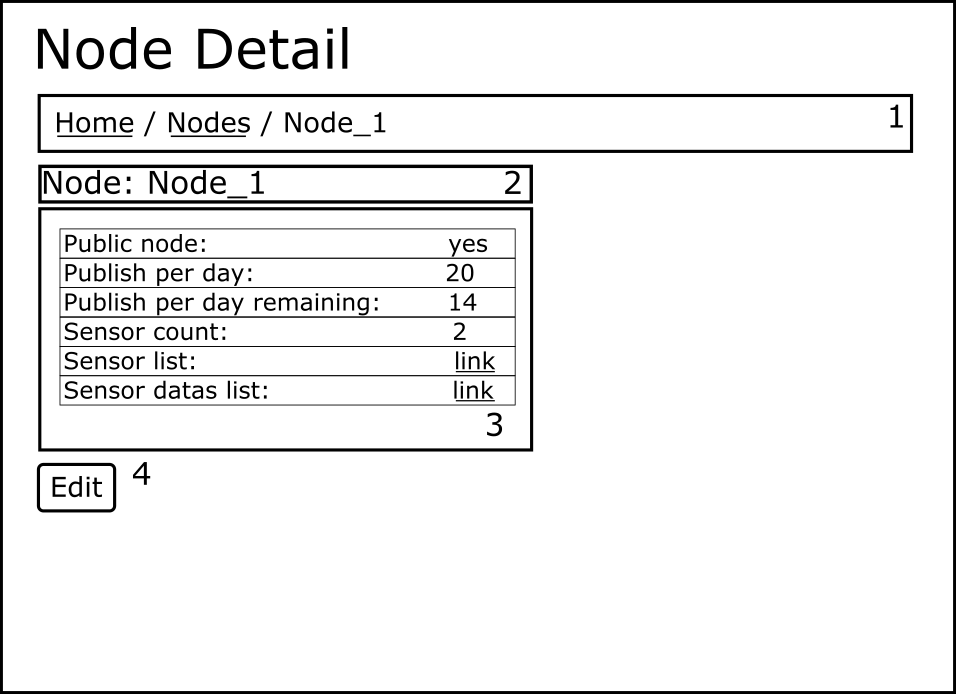
1. Lihat Semua Perangkat Node



Gambar 4.17 Perancangan Web Console: Lihat Semua Perangkat Node

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.17 diatas adalah:

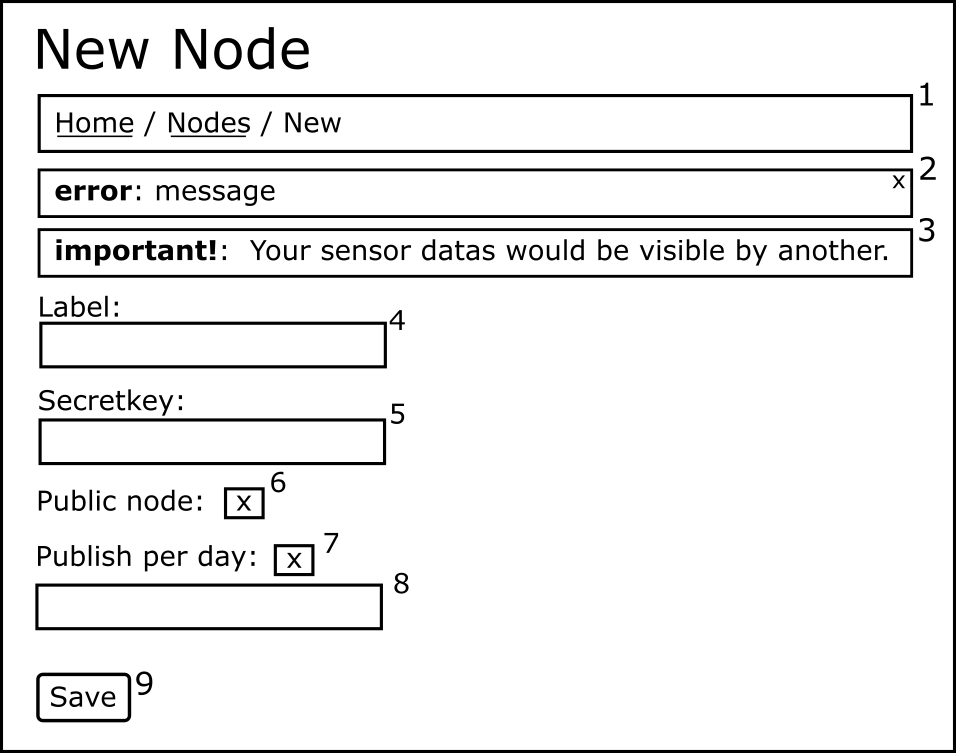
1. Elemen navigasi yang memungkinkan pengguna untuk kembali ke URL halaman sebelumnya.
2. Elemen tabulasi yang memungkinkan pengguna melihat daftar perangkat node berdasarkan kriteria yang ada. Tabulasi yang aktif pada saat halaman pertama kali dimuat adalah “*All*”.
3. Elemen *list* berisi daftar perangkat node sesuai dengan kriteria yang diminta.
4. Elemen *button* yang ketika ditekan akan mengarahkan pengguna ke halaman buat perangkat node.
5. Lihat Perangkat Node



Gambar 4.18 Perancangan Web Console: Lihat Perangkat Node

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.18 diatas adalah:

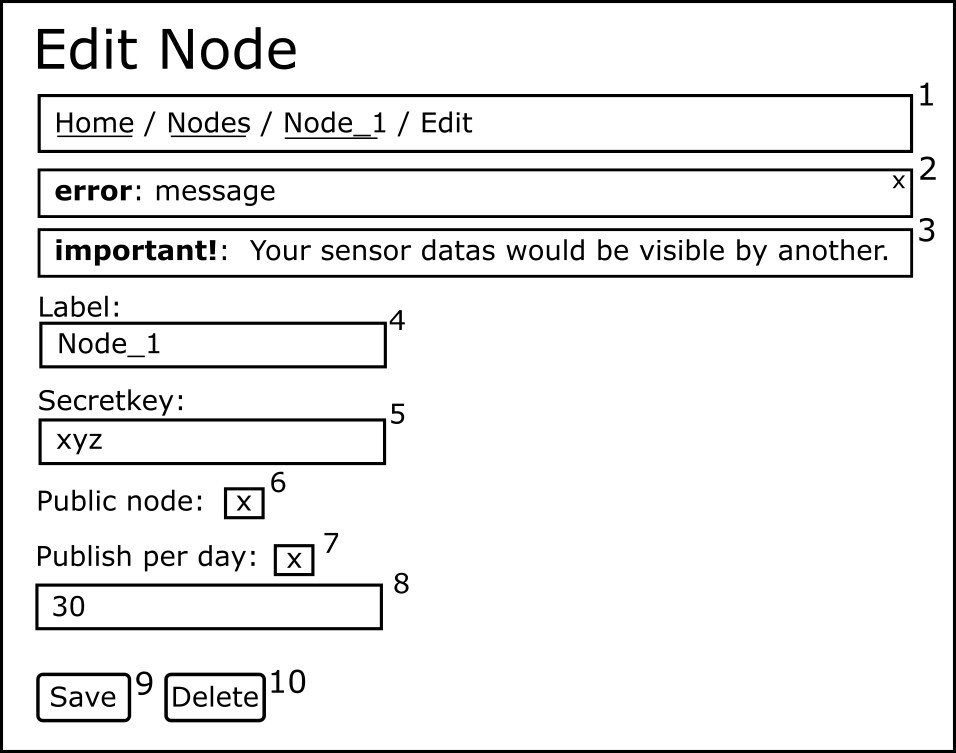
1. Elemen navigasi yang memungkinkan pengguna untuk kembali ke URL halaman sebelumnya.
2. Elemen yang menampilkan label dari perangkat node yang dimaksud.
3. Elemen tabel yang menampilkan informasi lainnya secara lengkap dari perangkat node yang dimaksud.
4. Elemen *button* yang ketika ditekan akan mengarahkan pengguna ke halaman ubah perangkat node.
5. Buat Perangkat Node



Gambar 4.19 Perancangan Web Console: Buat Perangkat Sensor

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.19 diatas adalah:

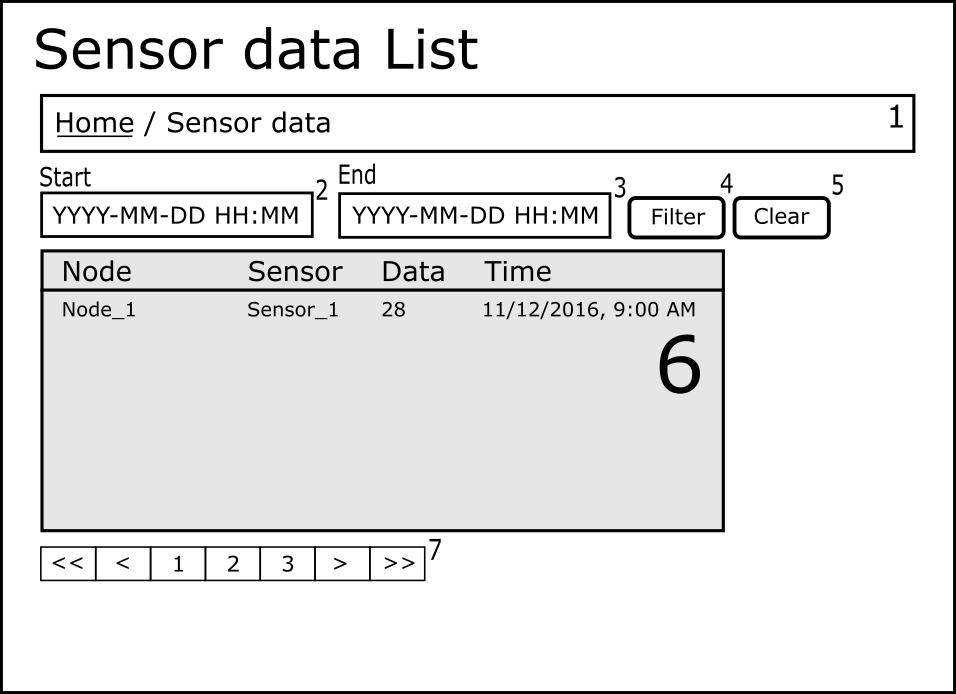
1. Elemen navigasi yang memungkinkan pengguna untuk kembali ke URL halaman sebelumnya.
2. Elemen yang nantinya akan menampilkan pesan kesalahan dari *web service* ketika pengguna menekan tombol “*Save*” dan data form input yang dikirimkan tersebut tidak valid.
3. Elemen yang berisi pesan bahwa data sensor perangkat dapat dilihat oleh pengguna lain. Elemen ini akan ditampilkan hanya ketika elemen *check-box* pada nomor 6 dalam keadaan tercentang.
4. Elemen *text-input* yang menampung label dari perangkat node.
5. Elemen *text-input* yang menampung *secretkey* dari perangkat node.
6. Elemen *check-box* yang menampung visibilitas perangkat node terhadap pengguna lain. Jika elemen ini tercentang, maka perangkat node yang dibuat akan memiliki visibilitas publik, jika tidak perangkat node akan memiliki visibilitas privat.
7. Elemen *check-box* yang menentukan pembatasan pengiriman data sensor dari perangkat node setiap harinya. Jika elemen ini tercentang, maka elemen *input* pada nomor 8 akan ditampilkan untuk diisi jumlah batasan pengirimannya tersebut. Sedangkan jika elemen ini tidak tercentang, pengiriman data sensor dari perangkat node yang dibuat tidak dibatasi.
8. Elemen *input* yang menampung jumlah data sensor yang dapat dikirimkan oleh perangkat node baru setiap harinya. Elemen ini akan ditampilkan hanya ketika elemen *check-box* pada nomor 7 dalam keadaan tercentang.
9. Elemen *button* yang ketika ditekan akan mengirimkan *request* ke *web service* untuk menyimpan perangkat node dan mengarahkan pengguna ke halaman lihat semua perangkat node.
10. Ubah Perangkat Node



Gambar 4.20 Perancangan Web Console: Ubah Perangkat Node

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.20 diatas adalah:

1. Elemen navigasi yang memungkinkan pengguna untuk kembali ke URL halaman sebelumnya.
2. Elemen yang nantinya akan menampilkan pesan kesalahan dari *web service* ketika pengguna menekan tombol “*Save*” dan data form input yang dikirimkan tersebut tidak valid.
3. Elemen yang berisi pesan bahwa data sensor perangkat dapat dilihat oleh pengguna lain. Elemen ini akan ditampilkan hanya ketika elemen *check-box* pada nomor 6 dalam keadaan tercentang.
4. Elemen *text-input* yang menampung label dari perangkat node.
5. Elemen *text-input* yang menampung *secretkey* dari perangkat node.
6. Elemen *check-box* yang menampung visibilitas perangkat node terhadap pengguna lain. Jika elemen ini tercentang, maka perangkat node yang dibuat akan memiliki visibilitas publik, jika tidak perangkat node akan memiliki visibilitas privat.
7. Elemen *check-box* yang menentukan pembatasan pengiriman data sensor dari perangkat node setiap harinya. Jika elemen ini tercentang, maka elemen *input* pada nomor 8 akan ditampilkan untuk diisi jumlah batasan pengirimannya tersebut. Sedangkan jika elemen ini tidak tercentang, pengiriman data sensor dari perangkat node yang dibuat tidak dibatasi.
8. Elemen *input* yang menampung jumlah data sensor yang dapat dikirimkan oleh perangkat node baru setiap harinya. Elemen ini akan ditampilkan hanya ketika elemen *check-box* pada nomor 7 dalam keadaan tercentang.
9. Elemen *button* yang ketika ditekan akan mengirimkan *request* ke *web service* untuk menyimpan perangkat node dan mengarahkan pengguna ke halaman lihat semua perangkat node.
10. Elemen *button* yang ketika ditekan akan menampilkan komfirmasi hapus. Jika dikonfirmasi, *request* hapus akan dikirimkan ke *web service* dan mengarahkan pengguna ke halaman lihat semua perangkat node.
11. Lihat Data Sensor



Gambar 4.21 Perancangan Web Console: Lihat Data Sensor

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.21 diatas adalah:

1. Elemen navigasi yang memungkinkan pengguna untuk kembali ke URL halaman sebelumnya.
2. Elemen *text-input* yang menampung kriteria “awal” untuk memfilter data sensor berdasarkan waktu.
3. Elemen *text-input* yang menampung kriteria “akhir” untuk memfilter data sensor berdasarkan waktu.
4. Elemen *button* yang ketika ditekan akan memuat ulang data sensor pada elemen 6, sesuai dengan kriteria waktu yang diminta.
5. Elemen *button* yang ketika ditekan akan memuat ulang data sensor tanpa kriteria waktu kemudian mengosongkan nilai pada elemen 2 dan elemen 3.
6. Elemen *table* yang menampilkan data sensor dalam satu paginasi, dimana data sensor pada setiap paginasi ditampilkan maksimal berjumlah 10.
7. Elemen *pagination* untuk pindah ke halaman paginasi yang dipilih.

# IMPLEMENTASI

Pada bagian ini dijelaskan mengenai implementasi IoT *coud platform* dengan mengacu pada bab analisis kebutuhan dan perancangan. Pada penelitian ini, implementasi meliputi: implementasi komponen komunikasi, implementasi komponen manajemen data, implementasi komponen manajemen perangkat dan implementasi komponen *web console*. Proses implementasi dilakukan pada mesin virtual dengan sistem operasi Ubuntu Server versi 14.04.

## Implementasi Komponen Komunikasi

Implementasi komponen komunikasi membahas mengenai instalasi dan konfigurasi yang diperlukan untuk menjalankan RESTful *Web Service*. Pada penelitian ini *web service* dikembangkan menggunakan Django Web Framework yang berbahasa pemrograman Python. *Web framework* tersebut kemudian dijalankan menggunakan Apache Web Server. Pembahasan pada sub-bab ini meliputi instalasi Apache Web Server, instalasi Django Web Framework, dan integrasi Apache Web Server dan Django Web Framework.

### Instalasi Apache Web Server

Apache Web Server merupakan perngkat lunak yang digunakan untuk menjalankan aplikasi web sehingga dapat melayani permintaan dari *client*. Web server ini dikembangkan oleh Apache Software Foundation dan dapat digunakan secara gratis.

Untuk menginstal nya, langkah yang diperlukan adalah mengeksekusi perintah dibawah ini:

|  |
| --- |
| $ sudo apt-get install apache2 |

Setelah proses instalasi dilakukan, jalankan *web server* dengan mengeksekusi perintah:

|  |
| --- |
| $ sudo systemctl start apache2 |

### Instalasi Django Web Framework

Django merupakan salah satu framework web yang ditulis menggunakan Bahasa pemrograman Python. Setiap aplikasi yang menggunakan Django memiliki pola arsitektur MVT (*model*, *view*, *template*) yang memisahkan bagian antara data, lojik, dan antarmuka untuk pengguna. Hal tersebut mempermudah dalam hal melakukan *maintenance*, karena perubahan pada satu bagian tidak akan mempengaruhi bagian yang lain. Django dikembangkan oleh Django Software Foundation dan dapat digunakan secara bebas dan gratis.

Untuk menginstal nya, langkah pertama yang diperlukan adalah melakukan instalasi *python package manager* pip lalu mengeksekusi perintah:

|  |
| --- |
| $ pip install -e django/ |

Setelah proses instalasi dilakukan, langkah selanjutnya adalah membuat project menggunakan Django dengan nama agri-hub, untuk itu jalankan perintah:

|  |
| --- |
| $ django-admin startproject agri-hub |

Aplikasi Django yang dikembangkan membutuhkan beberapa modul *dependencies* yang harus dipenuhi. Salah satunya modul mongoengine untuk mendukung sistem basis data MongoDB yang secara resmi tidak didukung. Untuk mempermudah dalam proses instalasinya, dapat dibuat file baru bernama requirements.txt yang mendefinisikan modul-modul yang dibutuhkan tersebut. Hal itu membuat proses instalasi *dependencies* dapat dilakukan sekaligus dengan satu baris perintah.

**requirements.txt**

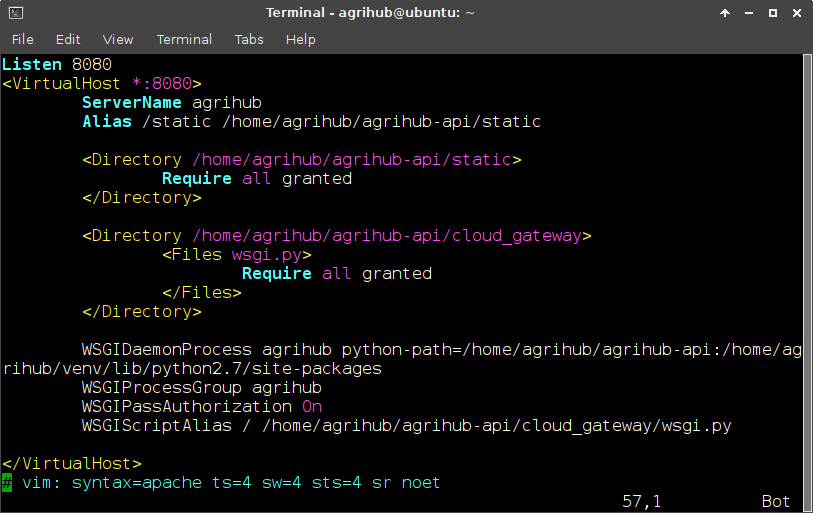
|  |
| --- |
| pymongo==3.3.1  Django==1.10.3  mongoengine==0.10.6  pymongo==3.3.1  djangorestframework\_jwt==1.8.0  django\_rest\_framework\_mongoengine==3.3.1  PyJWT==1.4.2  djangorestframework==3.5.3  django-cors-headers==1.3.1 |

Untuk menginstal semua *dependencies* yang didefinisikan pada file requirements.txt adalah dengan mengeksekusi perintah:

|  |
| --- |
| $ pip install -r requirements.txt |

### Integrasi Apache dan Django

Untuk menjalankan Django di Apache *web server* dibutuhkan modul tambahan yakni mod\_wsgi. Modul tersebut memungkinkan aplikasi yang ditulis menggunakan Bahasa pemrograman Python dijalankan sebagai CGI. Untuk itu Apache perlu dikonfigurasi dengan mengubah file /etc/apache2/sites-enabled/000-default.conf. Konfigurasi yang harus di tambahkan ke dalam file konfigurasi itu adalah: alamat CGI, alamat interpreter Python, alamat alias untuk static file (css, dan JS bawaan Django), dan alamat *port* dimana aplikasi Djangotersebut dijalankan. Berikut konfigurasi mod\_wsgi pada Apache ditampilkan pada Gambar 5.1.



Gambar 5.1 Konfigurasi Apache

## Implementasi Komponen Manajemen Data

Implementasi komponen manajemen data membahas instalasi sistem basis data dan implementasi komponen program yang menangani data yang disimpan di IoT *cloud platform*. Pembahasan pada bagian ini meliputi: instalasi MongoDB, implementasi data model dan implementasi data *access*.

### Instalasi MongoDB

Sistem basis data MongoDB secara resmi telah tersedia pada repositori Ubuntu. MongoDB membutuhkan beberapa komponen paket yaitu server, *daemon*, *shell*, dan *tools*. Komponen paket tersebut dapat diinstall sekaligus menggunakan paket mongodb-org yang berisi *metadata* untuk menginstall komponen paket yang dibutuhkan. Untuk menginstall sistem basis data MongoDB, jalankan perintah:

|  |
| --- |
| $ sudo apt-get install mongodb-org |

### Implementasi Data Model

Implementasi data sensor berisi kode sumber yang dihasilkan berdasarkan skema basis data pada tahap perancangan data model. Pada bagian ini dibahas kode sumber dari dokumen pengguna, dokumen perangkat node, dokumen sensor dan dokumen data sensor.

1. Dokumen Pengguna

Dokumen ini menyimpan informasi mengenai pengguna dari IoT *cloud platform* yang dirancang. Kode sumber dari dokumen ini disimpan pada file bernama users/models.py.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | from mongoengine.document import Document  from mongoengine import StringField, EmailField, IntField  class User(Document):  email = EmailField(required=True)  username = StringField(min\_length=4, max\_length=16, unique=True, required=True)  password = StringField(min\_length=8, max\_length=128, required=True)  first\_name = StringField(max\_length=50, required=True)  last\_name = StringField(max\_length=50, required=True)  is\_admin = IntField(default=0)  meta = {  'indexes': [  {  'fields': ['-username'],  'unique': True  },  ],  } |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 1-2, memuat modul-modul yang diperlukan.
2. Baris 5, deklarasi kelas User yang mewarisi atribut dan operasi dari kelas Document.
3. Baris 6, deklarasi atribut kelas bernama emailuntuk menampung alamat email pengguna. Atribut ini harus didefinisikan dengan alamat email yang valid.
4. Baris 7-8, deklarasi atribut kelas bernama usernameuntuk menampung *username* pengguna. Atribut ini harus didefinisikan sebagai *string* dengan 4-16 karakter.
5. Baris 9-10, deklarasi atribut kelas bernama passworduntuk menampung *password* pengguna. Atribut ini harus didefinisilan sebagai *string* dengan 8-128 karakter.
6. Baris 11, deklarasi atribut kelas bernama firstnameuntuk menampung nama depan pengguna. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 50 karakter.
7. Baris 12, deklarasi atribut kelas bernama lastnameuntuk menampung nama belakang pengguna. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 50 karakter.
8. Baris 13, deklarasi atribut kelas bernama is\_adminuntuk menampung level akses dari pengguna. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.
9. Baris 15-22, deklarasi *metadata* kelas untuk memastikan atribut kelas usernamememiliki nilai yang unik terhadap pengguna lainnya.
10. Dokumen Perangkat Node

Dokumen ini menyimpan informasi mengenai perangkat node yang dimiliki pengguna. Kode sumber dari dokumen ini disimpan pada file bernama nodes/models.py.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | from mongoengine.document import Document  from mongoengine import StringField, ReferenceField, EmbeddedDocumentListField, IntField, CASCADE  from sensors.models import Sensors  from users.models import User  class Nodes(Document):  user = ReferenceField(User, reverse\_delete\_rule=CASCADE)  label = StringField(max\_length=28)  secretkey = StringField(required=True, max\_length=16)  is\_public = IntField(default=0)  pubperday = IntField(default=0)  pubperdayremain = IntField(default=0)  sensors = EmbeddedDocumentListField(document\_type=Sensors) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 1-5, memuat modul-modul yang diperlukan.
2. Baris 6, deklarasi kelas Nodes yang mewarisi atribut dan operasi dari kelas Document.
3. Baris 6, deklarasi atribut kelas bernama useruntuk menampung nilai id dari pengguna yang memiliki perangkat node itu.
4. Baris 10, deklarasi atribut kelas bernama labeluntuk menampung label dari perangkat node. Atribut ini harus didefinisilan sebagai *string* dengan panjang maksimal 28 karakter.
5. Baris 11, deklarasi atribut kelas bernama secretkeyuntuk menampung kunci rahasia dari perangkat node. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 16 karakter.
6. Baris 12, deklarasi atribut kelas bernama is\_publicuntuk menampung visibilitas perangkat. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.
7. Baris 13, deklarasi atribut kelas bernama pubperdayuntuk menampung batasan pengiriman data sensor milik perangkat node. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.
8. Baris 14, deklarasi atribut kelas bernama pubperdayremainuntuk menampung nilai *counter* dari batasan pengiriman data sensor milik perangkat node setiap harinya. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.
9. Baris 15, deklarasi atribut kelas bernama sensorsuntuk menyimpan daftar sensor yang dimiliki perangkat node. Atribut ini didefinisikan sebagai *EmbeddedDocument* dari dokumen sensor.
10. Dokumen Sensor

Dokumen ini menyimpan daftar sensor yang dimiliki suatu perangkat node. Kode sumber dari dokumen ini disimpan pada file bernama sensors/models.py.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | from \_\_future\_\_ import unicode\_literals  import bson  from mongoengine import StringField, ObjectIdField  from mongoengine.document import EmbeddedDocument  class Sensors(EmbeddedDocument):  id = ObjectIdField(default=bson.objectid.ObjectId())  label = StringField(max\_length=28)  meta = {  'indexes': [  {  'fields': ['-label'],  'unique': True,  'types': False  },  ],  }  def \_\_unicode\_\_(self):  return self.label |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 1-4, memuat modul-modul yang diperlukan.
2. Baris 7, deklarasi kelas Sensors yang mewarisi atribut dan operasi dari kelas EmbeddedDocument.
3. Baris 8, deklarasi atribut kelas bernama iduntuk menampung nilai id dari sensor yang dimaksud. Atribut ini secara umum tidak tersedia di *EmbeddedDocument* dan harus didefinisikan secara manual untuk memudahkan operasi CRUD.
4. Baris 9, deklarasi atribut kelas bernama labeluntuk menampung label dari sensor. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 28 karakter.
5. Baris 11-19, deklarasi *metadata* kelas untuk memastikan atribut kelas labelmemiliki nilai yang unik terhadap sensor lainnya.
6. Baris 21-22, deklarasi *method* yang merepresentasikan *object* dari dokumen sensor berdasarkan nama labelnya.
7. Dokumen Data Sensor

Dokumen ini menyimpan informasi mengenai data sensor yang dikirimkan oleh perangkat node. Kode sumber dari dokumen ini disimpan pada file bernama sensordatas/models.py.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | from mongoengine import Document, ObjectIdField  from mongoengine import ReferenceField, IntField, DateTimeField, CASCADE  from nodes.models import Nodes  import datetime  class Sensordatas(Document):  node = ReferenceField(Nodes, reverse\_delete\_rule=CASCADE)  sensor = ObjectIdField(required=True)  data = IntField()  timestamp = DateTimeField(default=datetime.datetime.now()) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 1-5, memuat modul-modul yang diperlukan.
2. Baris 8, deklarasi kelas Sensorsdatas yang mewarisi atribut dan operasi dari kelas Document.
3. Baris 9, deklarasi atribut kelas bernama nodeuntuk menampung nilai id dari perangkat node yang mengirimkan data sensor tersebut.
4. Baris 10, deklarasi atribut kelas bernama sensoruntuk menampung nilai id dari sensor yang mengambil nilai dari data sensor tersebut.
5. Baris 11, deklarasi atribut kelas bernama datauntuk menampung nilai data sensor.
6. Baris 12, deklarasi atribut kelas bernama timestampuntuk menampung nilai data sensor. Atribut ini harus didefinisikan sebagai *object* dari kelas datetime.

### Implementasi Data Access

Implementasi data *access* berisi kode sumber untuk mengakses data sensor melalui RESTful *Web Service*. Terdapat tiga kriteria yang dapat digunakan, yakni berdasarkan pengguna, berdasarkan perangkat node, dan berdasarkan sensor.

1. Berdasarkan pengguna

Kriteria ini digunakan untuk mengakses data sensor dari semua perangkat node yang dimiliki pengguna. Kode sumber yang menangani kriteria ini disimpan pada file sensordatas/views.py dengan nama kelas SensordataFilterUser.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | ...  class SensordataFilterUser(ListAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  serializer\_class = SensordataSerializer  def get\_queryset(self):  user = self.request.user  if user.username != self.kwargs.get('user'):  raise PermissionDenied(detail="Your credential and URL prefix must be same.")  nodes = Nodes.objects.filter(user=user)  tmp = []  filter\_from = self.request.GET.get('start')  filter\_last = self.request.GET.get('end')  for node in nodes:  if filter\_from and filter\_last :  all\_subs = Sensordatas.objects.filter(  node=node, timestamp\_\_gte=filter\_from, timestamp\_\_lte=filter\_last  )  elif filter\_from:  all\_subs = Sensordatas.objects.filter(node=node, timestamp\_\_gte=filter\_from)  elif filter\_last:  all\_subs = Sensordatas.objects.filter(node=node, timestamp\_\_lte=filter\_last)  else:  all\_subs = Sensordatas.objects.filter(node=node)  for sub in all\_subs:  tmp.append(sub)  return tmp |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas SensorsdataFilterUser yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa kriteria ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 6, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk representasi data sensor sebelum dikirimkan kepada *client*. Kelas yang menangani representasi data sensor adalah SensordataSerializer.
5. Baris 8, *override* *method* get\_queryset() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasikan *object* yang akan ditampilkan.
6. Baris 9, deklarasi variabel user yang menampung *credentials* pengguna dari *payload* token akses JWT.
7. Baris 11-13, jika user.username tidak sesuai dengan username pada alamat URL, *web service* akan mengembalikan HTTP status 403.
8. Baris 15, deklarasi variabel nodes yang menampung daftar perangkat node yang dimiliki pengguna.
9. Baris 16, deklarasi variabel tmp yang menampung *object* data sensor.
10. Baris 18, deklarasi variabel filter\_from yang menampung parameter HTTP GET 'start' yang didapat dari URL.
11. Baris 19, deklarasi variabel filter\_last yang menampung parameter HTTP GET 'end' yang didapat dari URL.
12. Baris 21-37, iterasi sebanyak *object items* pada variabel nodes. Pada setiap iterasi, item perangkat node dapat diakses menggunakan variabel node.
13. Baris 22-34, mendapatkan data sensor dari *object* node sesuai dengan kriteria waktu yang didefinisikan dalam parameter HTTP GET 'start' dan 'end'. Data sensor tersebut ditampung dalam variabel all\_subs.
14. Baris 36-37, iterasi sebanyak data sensoryang ada pada variabel all\_subs. Pada setiap iterasi data sensor tersebut dimasukkan ke dalam variabel tmp.
15. Baris 38, *method* mengembalikan variabel tmp.
16. Berdasarkan perangkat node

Kriteria ini digunakan untuk mengakses data sensor milik perangkat node tertentu. Kode sumber yang menangani kriteria ini disimpan pada file sensordatas/views.py dengan nama kelas SensordataFilterNode.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | ...  class SensordataFilterNode(ListAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  serializer\_class = SubscriptionSerializer  ...  def get\_queryset(self):  nodeid = self.kwargs['node']  node = self.checknode(nodeid)  if self.request.user != node.user and 0 == node.is\_public:  return 'unauthorized'  filter\_from = self.request.GET.get('start')  filter\_last = self.request.GET.get('end')  if filter\_from and filter\_last:  return Sensordatas.objects.filter(  node=node, timestamp\_\_gte=filter\_from, timestamp\_\_lte=filter\_last  )  elif filter\_from:  return Sensordatas.objects.filter(node=node, timestamp\_\_gte=filter\_from)  elif filter\_last:  return Sensordatas.objects.filter(node=node, timestamp\_\_lte=filter\_last)  else:  return Sensordatas.objects.filter(node=node)  def get(self, request, \*args, \*\*kwargs):  raw\_queryset = self.get\_queryset()  if 'unauthorized' == raw\_queryset:  return Response({  'detail': 'Not found.'  }, status=status.HTTP\_404\_NOT\_FOUND)  queryset = self.filter\_queryset(raw\_queryset)  page = self.paginate\_queryset(queryset)  ...  serializer = SensordataSerializer(page, many=True, context={'request': request})  return self.get\_paginated\_response(serializer.data) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas SensorsdataFilterUser yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa kriteria ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 6, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk representasi data sensor sebelum dikirimkan kepada *client*. Kelas yang menangani representasi data sensor adalah SensordataSerializer.
5. Baris 8, *override* *method* get\_queryset() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasikan *object* yang akan ditampilkan.
6. Baris 9, deklarasi variabel nodeid yang berisi id perangkat node, diambil dari alamat URL.
7. Baris 10, deklarasi variabel node yang menampung *object* perangkat node berdasarkan id nya.
8. Baris 11-12, jika perangkat node yang diakses bukan milik pengguna dan perangkat node tersebut tidak memiliki label visibilitas publik, *method* akan mengembalikan *string* dengan nilai 'unauthorized'.
9. Baris 14, deklarasi variabel filter\_from yang menampung parameter HTTP GET 'start' yang didapat dari URL.
10. Baris 15, deklarasi variabel filter\_last yang menampung parameter HTTP GET 'end' yang didapat dari URL.
11. Baris 17-29, mendapatkan data sensor dari *object* node sesuai dengan kriteria waktu yang didefinisikan dalam parameter HTTP GET 'start' dan 'end'.
12. Baris 31, *override* *method* get() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP GET.
13. Baris 32, deklarasi variabel raw\_queryset yang menampung data mentah dari *method* get\_queryset().
14. Baris 33-36, jika nilai dari variabel raw\_queryset yang berisi *string* 'unauthorized', *web service* akan mengembalikan HTTP status 404.
15. Baris 37, deklarasi variabel queryset yang menampung data sensor yang sudah diolah sehingga dapat digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman.
16. Baris 38, deklarasi variabel page yang menampung data sensor sesuai dengan halaman yang diminta.
17. Baris 40-41, deklarasi variabel serializer yang menampung representasi data sensor sesuai dengan kriteria dan halaman yang diminta.
18. Baris 42, *web service* mengembalikan HTTP status 200 beserta data sensor yang diminta.
19. Berdasarkan sensor

Kriteria ini digunakan untuk mengakses data sensor milik sensor tertentu. Kode sumber yang menangani kriteria ini disimpan pada file sensordatas/views.py dengan nama kelas SensordataFilterSensor.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55 | ...  class SensordataFilterNodeSensor(ListAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  serializer\_class = SensordataSerializer  ...  def get\_queryset(self):  nodeid = self.kwargs['node']  sensorid = self.kwargs['sensor']  node\_sensor = self.checknode(nodeid, sensorid)  if self.request.user != node\_sensor.get('node').user and 0 == node\_sensor.get('node').is\_public:  return 'unauthorized'  filter\_from = self.request.GET.get('start')  filter\_last = self.request.GET.get('end')  if filter\_from and filter\_last:  return Sensordatas.objects.filter(  node=node\_sensor.get('node').id, sensor=node\_sensor.get('sensor').id,  timestamp\_\_gte=filter\_from, timestamp\_\_lte=filter\_last  )  elif filter\_from:  return Sensordatas.objects.filter(  node=node\_sensor.get('node').id, sensor=node\_sensor.get('sensor').id,  timestamp\_\_gte=filter\_from  )  elif filter\_last:  return Sensordatas.objects.filter(  node=node\_sensor.get('node').id, sensor=node\_sensor.get('sensor').id,  timestamp\_\_lte=filter\_last  )  else:  return Sensordatas.objects.filter(node=node\_sensor.get('node').id, sensor=node\_sensor.get('sensor').id)  def get(self, request, \*args, \*\*kwargs):  raw\_queryset = self.get\_queryset()  if 'unauthorized' == raw\_queryset:  return Response({  'detail': 'Not found.'  }, status=status.HTTP\_404\_NOT\_FOUND)  queryset = self.filter\_queryset(raw\_queryset)  page = self.paginate\_queryset(queryset)  ...  serializer = SensordataSerializer(page, many=True, context={'request': request})  return self.get\_paginated\_response(serializer.data) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas SensorsdataFilterSensor yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa kriteria ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 6, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk representasi data sensor sebelum dikirimkan kepada *client*. Kelas yang menangani representasi data sensor adalah SensordataSerializer.
5. Baris 8, *override* *method* get\_queryset() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasikan *object* yang akan ditampilkan.
6. Baris 9, deklarasi variabel nodeid yang berisi id perangkat node, diambil dari alamat URL.
7. Baris 10, deklarasi variabel sensorid yang berisi id sensor, diambil dari alamat URL.
8. Baris 12, deklarasi variabel node\_sensor yang menampung *object* perangkat node dan sensor.
9. Baris 13-15, jika perangkat node yang diakses bukan milik pengguna dan perangkat node tersebut tidak memiliki label visibilitas publik, *method* akan mengembalikan *string* dengan nilai 'unauthorized'.
10. Baris 17, deklarasi variabel filter\_from yang menampung parameter HTTP GET 'start' yang didapat dari URL.
11. Baris 18, deklarasi variabel filter\_last yang menampung parameter HTTP GET 'end' yang didapat dari URL.
12. Baris 20-42, mendapatkan data sensor sesuai dengan kriteria waktu yang didefinisikan dalam parameter HTTP GET 'start' dan 'end'.
13. Baris 44, *override* *method* get() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP GET.
14. Baris 45, deklarasi variabel raw\_queryset yang menampung data mentah dari *method* get\_queryset().
15. Baris 46-49, jika nilai dari variabel raw\_queryset yang berisi *string* 'unauthorized', *web service* akan mengembalikan HTTP status 404.
16. Baris 50, deklarasi variabel queryset yang menampung data sensor yang sudah diolah sehingga dapat digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman.
17. Baris 51, deklarasi variabel page yang menampung data sensor sesuai dengan halaman yang diminta.
18. Baris 53-54, deklarasi variabel serializer yang menampung representasi data sensor sesuai dengan kriteria dan halaman yang diminta.
19. Baris 55, *web service* mengembalikan HTTP status 200 beserta data sensor yang diminta.

## Implementasi Komponen *Security*

Implementasi komponen *security* dibagi menjadi dua yakni implementasi otentikasi dan otorisasi; dan implementasi manajemen perangkat.

### Implementasi Otentikasi dan Otorisasi

Implementasi otentikasi dan otorisasi berisi kode sumber dalam mendapatkan token akses JWT, registrasi pengguna, menerima data sensor dari perangkat node, dan mengatur ulang bilangan *counter* pembatasan pengiriman semua perangkat node.

1. Mendapatkan token akses JWT

Token akses JWT dibagi menjadi dua sesuai dengan otorisasi yang diberikan yakni token untuk perangkat node dan token untuk aplikasi *client*.

* Perangkat node

Kode sumber yang menangani pembuatan token akses untuk perangkat node disimpan pada file authenticate/views.py dengan nama kelas NodeTokenCreator.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | ...  class NodeTokenCreator(APIView):  def post(self, request, format=None):  form = NodeAuthForm(request.data)  if form.is\_valid():  return Response({  'node': NodeSerializer(form.node, context={'request': request}).data,  'token': self.create\_token(form.node)  })  return Response(form.errors, status=status.HTTP\_400\_BAD\_REQUEST)  @staticmethod  def create\_token(node):  payload = node\_jwt\_payload\_handler(node)  token = jwt.encode(payload, settings.SECRET\_KEY)  return token.decode('unicode\_escape') |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas NodeTokenCreator yang mewarisi atribut dan *method* dari kelas APIView.
2. Baris 4, *override* *method* post() dari kelas APIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP POST.
3. Baris 5, deklarasi variabel form yang menampung *object* dari kelas NodeAuthForm. *Object* tersebut digunakan untuk memvalidasi isi *payload* yang dikirimkan lewat HTTP POST.
4. Baris 6-11, jika validasi lewat form.is\_valid() berhasil, *web serice* akan mengembalikan HTTP status 200 beserta *object* perangkat node dan token akses. Token akses dihasilkan dengan memanggil *method* create\_token().
5. Baris 12-13, jika validasi gagal, *web service* akan mengembalikan HTTP status 400.
6. Baris 15-19, deklarasi *method* create\_token() untuk menghasilkan token akses.

* Aplikasi *client*

Kode sumber yang menangani pembuatan token akses untuk aplikasi *client* disimpan pada file authenticate/views.py dengan nama kelas UserTokenCreator.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | ...  class UserTokenCreator(APIView):  def post(self, request, format=None):  form = UserAuthForm(request.data)  if form.is\_valid():  return Response({  'user': UserSerializer(form.user).data,  'token': self.create\_token(form.user)  })  return Response(form.errors, status=status.HTTP\_400\_BAD\_REQUEST)  @staticmethod  def create\_token(user):  payload = user\_jwt\_payload\_handler(user)  token = jwt.encode(payload, settings.SECRET\_KEY)  return token.decode('unicode\_escape') |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas UserTokenCreator yang mewarisi atribut dan *method* dari kelas APIView.
2. Baris 4, *override* *method* post() dari kelas APIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP POST.
3. Baris 5, deklarasi variabel form yang menampung *object* dari kelas UserAuthForm. *Object* tersebut digunakan untuk memvalidasi isi *payload* yang dikirimkan lewat HTTP POST.
4. Baris 6-10, jika validasi lewat form.is\_valid() berhasil, *web serice* akan mengembalikan HTTP status 200 beserta *object* pengguna dan token akses. Token akses dihasilkan dengan memanggil *method* create\_token().
5. Baris 11-12, jika validasi gagal, *web service* akan mengembalikan HTTP status 400.
6. Baris 14-18, deklarasi *method* create\_token() untuk menghasilkan token akses.
7. Registrasi Pengguna

Kode sumber untuk registrasi pengguna disimpan pada file users/views.py dengan nama kelas ResearcherRegistration.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | ...  class ResearcherRegistration(GenericAPIView):  serializer\_class = UserSerializer  @staticmethod  def post(request):  ...  serializer = UserSerializer(data=request.data, context={'request': request})  if serializer.is\_valid():  serializer.save()  return Response(serializer.data, status=status.HTTP\_201\_CREATED)  return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas UserTokenCreator yang mewarisi atribut dan *method* dari kelas APIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk validasi isi *payload*.
3. Baris 6-7, *override* *method* post() dari kelas APIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP POST.
4. Baris 9, deklarasi variabel serializer yang menangani validasi *object* pengguna sebelum disimpan ke dalam basis data.
5. Baris 11-14, jika validasi berhasil, *object* pengguna baru akan disimpan dan *web service* akan mengembalikan HTTP status 201.
6. Baris 11-12, jika validasi gagal, *web service* akan mengembalikan HTTP status 400 beserta infomasi kegagalan dalam melakukan validasi.
7. Menerima data sensor dari perangkat node

Kode sumber untuk menerima data sensor dari perangkat node disimpan pada file sensordatas/views.py dengan nama kelas SensordatasList.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26 | ...  class SensordatasList(ListAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsAuthenticated,)  ...  serializer\_class = SensordataSerializer  @staticmethod  def post(request):  if not isinstance(request.user, Nodes):  raise exceptions.AuthenticationFailed("You do not have permission to perform this action.")  serformat = SensordataFormatSerializer(data=request.data, context={'request': request})  if serformat.is\_valid():  data = serformat.save()  return Response(  { "results": SensordataSerializer(data, many=True, context={'request': request}).data },  status=status.HTTP\_201\_CREATED  )  else:  return Response(serformat.errors, status=status.HTTP\_400\_BAD\_REQUEST) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas SensordatasList yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini hanya dapat dilakukan oleh pihak yang telah terotentikasi. Kelas yang menangani otorisasi tersebut adalah IsAuthenticated.
4. Baris 7, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk untuk validasi isi *payload*. Kelas yang menangani representasi data sensor adalah SensordataSerializer.
5. Baris 9-10, *override* *method* post() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP POST.
6. Baris 11-13, jika token akses yang digunakan bukan mewakili otoritas perangkat node, *web service* akan mengenbalikan HTTP status 403.
7. Baris 15-16, deklarasi variabel serializer yang menangani validasi *object* data sensor sebelum disimpan ke dalam basis data.
8. Baris 17-23, jika validasi berhasil, *object* data sensor akan disimpan dan *web service* akan mengembalikan HTTP status 201.
9. Baris 24-26, jika validasi gagal, *web service* akan mengembalikan HTTP status 400 beserta infomasi kegagalan dalam melakukan validasi.

### Implementasi Menejemen Perangkat

Implementasi manajemen perangkat berisi kode sumber untuk memanajemen perangkat.

1. Membuat perangkat node baru

Kode sumber untuk membuat perangkat node baru disimpan pada file nodes/views.py dengan nama kelas NodeList.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | ...  class NodesList(ListAPIView):    authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  serializer\_class = NodeSerializer  ...  @staticmethod  def post(request):  ...  serializer = NodeSerializer(data=request.data, context={'request': request})  if serializer.is\_valid():  serializer.save()  return Response(serializer.data, status=status.HTTP\_201\_CREATED)  return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas NodesList yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 6, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 7, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk representasi data sensor sebelum dikirimkan kepada *client*. Kelas yang menangani representasi data sensor adalah NodeSerializer.
5. Baris 9-10, *override* *method* post() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP POST.
6. Baris 12-13, deklarasi variabel serializer yang menangani validasi *object* perangkat node sebelum disimpan ke dalam basis data.
7. Baris 14-17, jika validasi berhasil, *object* perangkat node akan disimpan dan *web service* akan mengembalikan HTTP status 201.
8. Baris 18-19, jika validasi gagal, *web service* akan mengembalikan HTTP status 400 beserta infomasi kegagalan dalam melakukan validasi.
9. Melihat perangkat node

Kode sumber untuk melihat perangkat node disimpan pada file nodes/views.py dengan nama kelas NodeList.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | ...  class NodesList(ListAPIView):    authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  serializer\_class = NodeSerializer  ...  def get(self, request, \*args, \*\*kwargs):  queryset = self.filter\_queryset(self.get\_nodes(request.user, request.GET.get('role')))  page = self.paginate\_queryset(queryset)  if page is not None:  serializer = NodeSerializer(page, many=True, context={'request': request})  return self.get\_paginated\_response(serializer.data)  serializer = self.get\_serializer(queryset, many=True)  return Response(serializer.data) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas NodesList yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 6, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 7, deklarasi atribut kelas yang mendeskripsikan *serializer* untuk representasi data sensor sebelum dikirimkan kepada *client*. Kelas yang menangani representasi data sensor adalah NodeSerializer.
5. Baris 9, *override* *method* get() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP GET.
6. Baris 10-12, deklarasi variabel queryset yang menampung perangkat node sehingga dapat digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman.
7. Baris 13, deklarasi variabel page yang menampung perangkat node sesuai dengan halaman yang diminta.
8. Baris 14-17, jika parameter HTTP GET page didefinisikan, *web service* mengembalikan HTTP status 200 beserta perangkat node sesuai dengan halaman yang diminta.
9. Baris 19-20, jika parameter HTTP GET page tidak didefinisikan, *web service* mengembalikan HTTP status 200 beserta semua perangkat node.
10. Melihat perangkat node berdasarkan id

Kode sumber untuk melihat perangkat node berdasarkan id disimpan pada file nodes/views.py dengan nama kelas NodeDetail.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | ...  class NodeDetail(GenericAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  @staticmethod  def get\_object(pk):  try:  return Nodes.objects.get(pk=pk)  except Exception:  raise Http404  def get(self, request, pk, format=None):  node = self.get\_object(pk)  if request.user != node.user and 0 == node.is\_public:  return Response({  'detail': 'Not found.'  }, status=status.HTTP\_404\_NOT\_FOUND)  serializer = NodeSerializer(node, context={'request': request})  return Response(serializer.data) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas NodesDetail yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 7-12, deklarasi *method* get\_object()yang mengembalikan *object* perangkat node berdasarkan id.
5. Baris 14, *override* *method* get() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP GET.
6. Baris 15, deklarasi variabel node yang menampung *object* perangkat node dari *method* get\_object().
7. Baris 16-19, jika perangkat node yang diakses bukan milik pengguna dan perangkat node tersebut tidak memiliki label visibilitas publik, *web service* mengembalikan HTTP status 404.
8. Baris 20-21, *web service* mengembalikan HTTP status 200 beserta perangkat node yang diminta.
9. Mengubah perangkat node

Kode sumber untuk mengubah perangkat node disimpan pada file nodes/views.py dengan nama kelas NodeDetail.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27 | ...  class NodeDetail(GenericAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  @staticmethod  def get\_object(pk):  try:  return Nodes.objects.get(pk=pk)  except Exception:  raise Http404  def put(self, request, pk, format=None):  node = self.get\_object(pk)  if request.user != node.user:  return Response({  'detail': 'You can not update another person node.'  }, status=status.HTTP\_403\_FORBIDDEN)  serializer = NodeSerializer(node, data=request.data, context={'request': request}, partial=True)  if serializer.is\_valid():  serializer.save()  return Response(serializer.data, status=status.HTTP\_200\_OK)  return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas NodesDetail yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 7-12, deklarasi *method* get\_object()yang mengembalikan *object* perangkat node berdasarkan id.
5. Baris 14, *override* *method* put() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP PUT.
6. Baris 15, deklarasi variabel node yang menampung *object* perangkat node dari *method* get\_object().
7. Baris 16-19, jika perangkat node yang diakses bukan milik pengguna, *web service* mengembalikan HTTP status 403.
8. Baris 20-21, deklarasi variabel serializer yang menangani validasi *object* perangkat node sebelum disimpan ke dalam basis data.
9. Baris 22-25, jika validasi berhasil, *object* perangkat node akan disimpan dan *web service* akan mengembalikan HTTP status 200.
10. Baris 18-19, jika validasi gagal, *web service* akan mengembalikan HTTP status 400 beserta infomasi kegagalan dalam melakukan validasi.
11. Menghapus perangkat node

Kode sumber untuk menghapus perangkat node disimpan pada file nodes/views.py dengan nama kelas NodeDetail.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | ...  class NodeDetail(GenericAPIView):  authentication\_classes = (JSONWebTokenAuthentication,)  permission\_classes = (IsUser,)  @staticmethod  def get\_object(pk):  try:  return Nodes.objects.get(pk=pk)  except Exception:  raise Http404  def delete(self, request, pk, format=None):  node = self.get\_object(pk)  if request.user != node.user:  return Response({  'detail': 'You can not delete another person node.'  }, status=status.HTTP\_403\_FORBIDDEN)  node.delete()  return Response(status=status.HTTP\_204\_NO\_CONTENT) |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3, deklarasi kelas NodesDetail yang mewarisi atribut dan *method* dari kelas ListAPIView.
2. Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa otentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani otentikasi tersebut adalah JSONWebTokenAuthentication.
3. Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini membutuhkan otorisasi sebagai pengguna. Kelas yang menangani otorisasi tersebut adalah IsUser.
4. Baris 7-12, deklarasi *method* get\_object()yang mengembalikan *object* perangkat node berdasarkan id.
5. Baris 14, *override* *method* delete() dari kelas ListAPIView. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP DELETE.
6. Baris 15, deklarasi variabel node yang menampung *object* perangkat node dari *method* get\_object().
7. Baris 16-19, jika perangkat node yang diakses bukan milik pengguna, *web service* mengembalikan HTTP status 403.
8. Baris 20, menghapsu *object* perangkat node di basis data.
9. Baris 21, *web service* akan mengembalikan HTTP status 204 menandakan perangkat node berhasil dihapus.

## Implementasi Komponen *Web Console*

Implementasi komponen *web console* menjelaskan mengenai konfigurasi, hasil akhir antarmuka pengguna dan kode sumber untuk memanajemen perangkat dan melihat data sensor. Kkomponen *web console* menerjemahkan interaksi pengguna ke dalam *request* yang sesuai dan mengirimkannya ke RESTful *web service*. Aplikasi ini dibangun berdasarkan *platform* web menggunakan *framework* JavaScript Angular versi 2.0.

### Instalasi Angular

Angular merupakan *framework front-end* yang dikembangkan oleh Google untuk membangun *single-page application* (SPA). *Single-page application* merupakan aplikasi berbasis web yang memuat *object* secara dinamis berdasarkan interaksi pengguna tanpa memuat ulang halaman. Hal ini membuat *user experience* menjadi lebih baik karena tidak perlu berkali-kali memuat ulang halaman, seperti halnya aplikasi *desktop*.

Angular terdiri dari beberapa modul berbeda dan membutuhkan modul-modul lainnya untuk dapat digunakan. Untuk itu dalam pembuatan *project* Angular diperlukan file bernama package.json yang salah satunya berisi modul-modul yang dibutuhkan tersebut.

|  |
| --- |
| {  "name": "angular2-agri-hub",  "version": "1.0.0",  "description": "Agri Hub Client Using Angular2",  "scripts": {  "start": "webpack-dev-server --inline --progress --port 8080",  "build": "rimraf dist && webpack --config config/webpack.prod.js --progress --profile --bail"  },  "keywords": [],  "author": "ocki.bagus.p@gmail.com",  "license": "MIT",  "dependencies": {  "@angular/common": "~2.4.0",  "@angular/compiler": "~2.4.0",  "@angular/core": "~2.4.0",  "@angular/forms": "~2.4.0",  "@angular/http": "~2.4.0",  "@angular/platform-browser": "~2.4.0",  "@angular/platform-browser-dynamic": "~2.4.0",  "@angular/router": "~3.4.0",  "@ng-bootstrap/ng-bootstrap": "1.0.0-alpha.18",  "angular-in-memory-web-api": "~0.2.2",  "angular2-cookie": "~1.2.4",  "core-js": "^2.4.1",  "ng2-datetime-picker": "^0.14.1",  "reflect-metadata": "^0.1.8",  "rxjs": "5.0.1",  "systemjs": "0.19.40",  "zone.js": "^0.7.4",  "angular2-template-loader": "^0.6.0",  "awesome-typescript-loader": "^3.0.4",  "css-loader": "^0.26.1",  "extract-text-webpack-plugin": "2.0.0-beta.5",  "file-loader": "^0.9.0",  "html-loader": "^0.4.3",  "html-webpack-plugin": "^2.16.1",  "jasmine-core": "^2.4.1",  "null-loader": "^0.1.1",  "raw-loader": "^0.5.1",  "rimraf": "^2.5.2",  "style-loader": "^0.13.1",  "typescript": "~2.0.10",  "webpack": "2.2.1",  "webpack-dev-server": "2.4.1",  "webpack-merge": "^3.0.0"  },  "devDependencies": {  "concurrently": "^3.1.0",  "lite-server": "^2.2.2",  "typescript": "~2.0.10",  "canonical-path": "0.0.2",  "http-server": "^0.9.0",  "tslint": "^3.15.1",  "lodash": "^4.16.4",  "jasmine-core": "~2.4.1",  "protractor": "~4.0.14",  "rimraf": "^2.5.4",  "@types/node": "^6.0.46",  "@types/jasmine": "2.5.41"  },  "repository": {}  } |

Untuk menginisiasi *project* dengan modul-modul yang didefinisikan pada file package.json, perlu melakukan eksekusi perintah sebagai berikut:

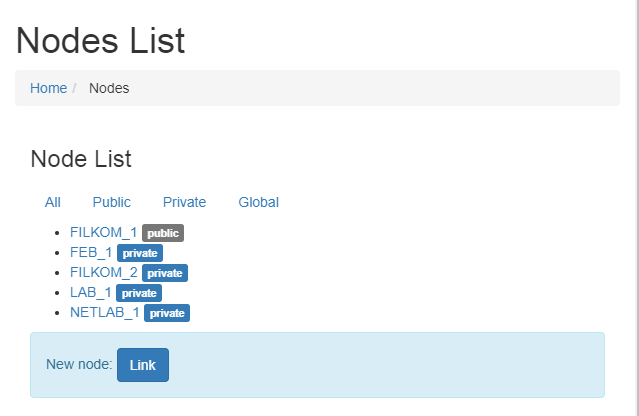
|  |
| --- |
| $ npm install |

### Integrasi *Web Console* dengan *Web service*

Bagian ini membahas mengenai hasil akhir dari antarmuka pengguna beserta kode sumber untuk menangani interaksi pengguna dan integrasi dengan *web service*. Kode sumber ditulis menggunakan Bahasa pemrograman TypeScript dengan file ekstensi .ts. Fitur aplikasi *web console* pada bagian ini meliputi lihat semua perangkat node, lihat perangkat node, buat perangkat node, ubah perangkat node dan lihat data sensor.

1. Lihat Semua Perangkat Node

Terdapat empat kriteria yang dapat dipilih berdasarkan visibilitas perangkat. Kriteria-kriteria tersebut dipisahkan ke dalam elemen tabulasi yang berbeda. Hasil akhir dari implementasi lihat semua perangkat node dapat dilihat pada Gambar 5.2.



Gambar 5.2 Implementasi Aplikasi Web Console: Lihat Semua Perangkat Node

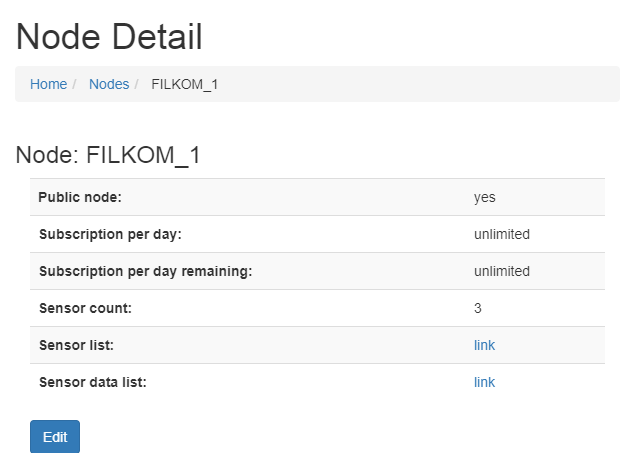
Kode sumber untuk melihat semua perangkat node disimpan pada file nodes/node.component.ts dengan nama kelas NodeComponent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38 | ...  @Component({  ...  })  export class NodeComponent implements OnInit {    nodes: Node[];  links: any[];  activeId = "all";  constructor(  private nodeService: NodeService,  public router: Router,  ) {}    ngOnInit(): void {  this.links = [  { label: "Home", url: "/" },  { label: "Nodes", is\_active: true}  ]  this.getNodes();  }  getNodes(role: string=""): void {  this.nodeService.getNodes(role)  .subscribe(  res => this.nodes = res.results as Node[],  error => console.log(error)  );  }  tabChange($event: NgbTabChangeEvent): void {  this.router.navigateByUrl(`/nodes?visibility=${$event.nextId}`);  this.getNodes($event.nextId);  }  } |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3-6, deklarasi *component* NodeComponent yang mengimplementasi *interface* OnInit.
2. Baris 8, deklarasi variabel nodesuntuk menampung *array* dari *object* perangkat node yang nantinya didapatkan dari web service.
3. Baris 9, deklarasi variabel linksuntuk menampung navigasi URL yang tersedia.
4. Baris 10, deklarasi variabel activeId yang berisi nilai awal “all”. Variabel ini digunakan untuk menyimpan id elemen tabulasi yang sedang aktif.
5. Baris 12-15, *constructor* yang berisi deklarasi *object* dari *class* lain yang digunakan pada *class* ini.
6. Baris 17-23, *override* dari *method* ngOnInit() dari *interface* OnInit. *Method* ini adalah bagian pertama yang akan dijalankan ketika *component* dibuat.
7. Baris 18-21, memberikan nilai dari variabel links.
8. Baris 22, memanggil *method* getNodes().
9. Baris 25-31, deklarasi *method* getNodes() untuk mendapatkan data perangkat node dari *web service* sesuai dengan nilai dari parameter *role*.
10. Baris 35-37, dekalarsi *method* tabChange() yang menangani *event* perubahan elemen tabulasi yang aktif. Setiap ada perubahan, *method* getNodes() akan dipanggil untuk memperbaharui perangkat node yang ditampilkan.
11. Lihat Perangkat Node

Halaman ini menampilkan informasi secara lengkap mengenai suatu perangkat node. Hasil akhir dari implementasi lihat perangkat node dapat dilihat pada Gambar 5.3.



Gambar 5.3 Implementasi Aplikasi Web Console: Lihat Perangkat Node

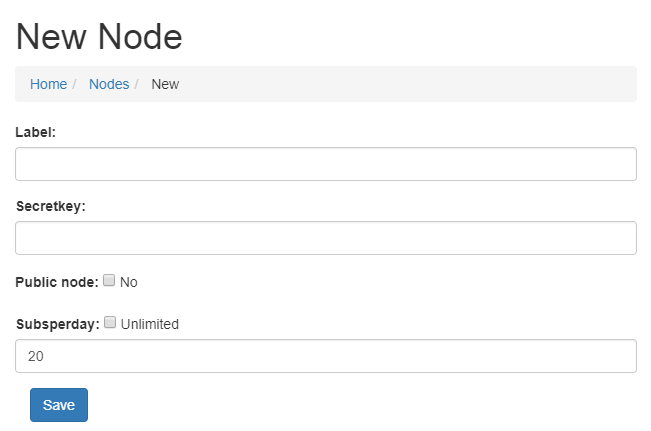
Kode sumber untuk melihat perangkat node disimpan pada file nodes/node-detail.component.ts dengan nama kelas NodeDetailComponent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43 | ...  @Component({  ...  })  export class NodeDetailComponent implements OnInit {  node: Node;  links: any[]; // breadcrumb  is\_mine: boolean; // 'edit' button visibility  constructor(  private nodeService: NodeService,  private route: ActivatedRoute,  private credentialsService: CredentialsService,  private router: Router  ) {}  ngOnInit() {  this.route.params  .switchMap((params: Params) => this.nodeService.getNode(params['id']))  .subscribe(  node => this.setUpNode(node),  error => console.log(error)  );  }  edit(): void {  this.router.navigate(['/nodes/edit', this.node.id]);  }  private setUpNode(node: Node): void {  this.node = node;  this.is\_mine = (node.user == this.credentialsService.getUser().username) ?  true: false;  this.links = [  { label: "Home", url: "/" },  { label: "Nodes", url: "/nodes/" },  { label: this.node.label, url: "/" , is\_active: true}  ];  }  } |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3-6, deklarasi component NodeDetailComponent yang mengimplementasi interface OnInit.
2. Baris 7, deklarasi variabel nodeuntuk menampung *object* perangkat node yang nantinya didapatkan dari web service.
3. Baris 8, deklarasi variabel linksuntuk menampung navigasi URL yang tersedia.
4. Baris 9, deklarasi variabel is\_minebertipe data *Boolean*. Nilai dari variabel ini akan menentukan ditampilkan atau tidaknya *button* Edit. Jika perangkat node tersebut merupakan milik pengguna yang sedang terotentikasi, variabel ini akan bernilai true, jika tidak akan bernilai false.
5. Baris 11-16, *constructor* yang berisi deklarasi *object* dari *class* lain yang digunakan pada *class* ini.
6. Baris 18-26, *override* dari *method* ngOnInit() dari *interface* OnInit. *Method* ini adalah bagian pertama yang akan dijalankan ketika *component* dibuat.
7. Baris 19-25, mendapatkan data perangkat node dari *web service*.
8. Baris 23, *object* perangkat node yang diterima dijadikan parameter untuk method setUpNode() .
9. Baris 32-42, deklarasi *method* setUpNode() untuk memberikan nilai terhadap variabel node, links, dan is\_mine.
10. Buat Perangkat Node

Halaman ini berisi form input untuk membuat perangkat node baru. Ketika pertama kali dimuat, semua form input dalam kedaan kosong. Hasil akhir dari implementasi buat perangkat node dapat dilihat pada Gambar 5.4.



Gambar 5.4 Implementasi Aplikasi Web Console: Buat Perangkat Node

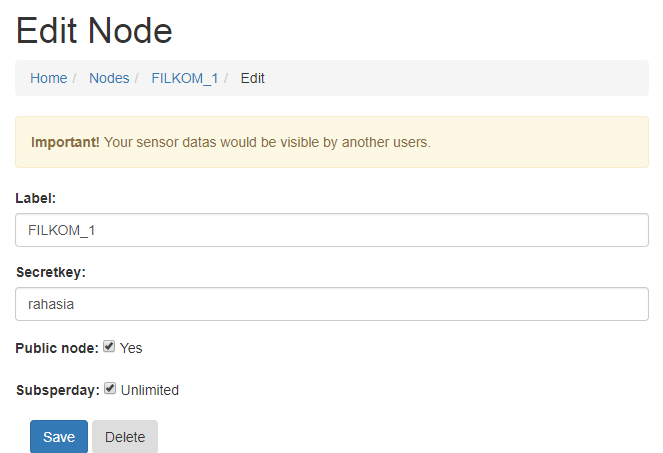
Kode sumber untuk membuat perangkat node disimpan pada file nodes/node-new.component.ts dengan nama kelas NodeNewComponent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | ...  @Component({  ...  })  export class NodeNewComponent {  ...  links: any[];  node: Node = new Node;  unlimited: boolean;  errors: Array<{ field: string, message: string}>;  constructor(  private nodeService: NodeService,  private router: Router  ) {}  ngOnInit() {  this.links = [  { label: "Home", url: "/" },  { label: "Nodes", url: "/nodes/" },  { label: "New", is\_active: true }  ];  }  unlimitedStateChange(): void {  if (this.unlimited) {  this.node.pubsperday = -1;  } else {  this.node.pubsperday = 0;  }  }  save(): void {  this.node.is\_public = this.node.is\_public ? 1 : 0;  this.nodeService.save(this.node)  .subscribe(  node => this.router.navigate(['/nodes']),  error => this.extractErrors(error)  );  }  private extractErrors(err: any): void {  let errorsParse = JSON.parse(err.\_body);  this.errors = [];  for(let index in errorsParse) {  if(errorsParse.hasOwnProperty(index)) {  this.errors.push({  field: index,  message: errorsParse[index]  })  }  }  }  public closeAlert(alert: any) {  const index: number = this.errors.indexOf(alert);  this.errors.splice(index, 1);  }  } |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3-6, deklarasi component NodeNewComponent yang mengimplementasi interface OnInit.
2. Baris 8, deklarasi variabel linksuntuk menampung navigasi URL yang tersedia.
3. Baris 9, deklarasi variabel nodeuntuk menampung data dari form input.
4. Baris 10, deklarasi variabel unlimitedyang digunakan oleh form input bertipe *checkbox*. Nilai dari variabel ini nantinya akan dikonversi ke dalam bilangan *integer* yang menentukan pembatasan pengiriman dari perangkat node yang dibuat.
5. Baris 12, deklarasi variabel errorsuntuk menampung pesan kegagalan validasi dalam pembuatan perangkat node.
6. Baris 14-17, *constructor* yang berisi deklarasi *object* dari *class* lain yang digunakan pada *class* ini.
7. Baris 19-25, *override* dari *method* ngOnInit() dari *interface* OnInit. *Method* ini adalah bagian pertama yang akan dijalankan ketika *component* dibuat.
8. Baris 20-24, memberikan nilai dari variabel links.
9. Baris 27-33, deklarasi *method* unlimitedStateChange() untuk mengkonversi nilai dari variabel unlimited ke dalam nilai awal node.subsperday.
10. Baris 35-42, deklarasi *method* save() untuk menangani *event* ketika pengguna menekan tombol “*Save*”.
11. Baris 36, mengkonversi nilai dari form input node.pubsperday ke dalam bilangan *integer* 1 atau 0;
12. Baris 37-41, mengirim nilai dari *object* perangkat node baru ke *web service* untuk disimpan.
13. Baris 39, jika pembuatan perangkat node berhasil, pengguna akan diarahkan ke halaman lihat semua perangkat node.
14. Baris 40, jika pembuatan perangkat node tidak berhasil, pesan error yang diterima dijadikan parameter untuk method extractErrors().
15. Baris 44-55, deklarasi *method* extractErrors() untuk mengkonversi pesan error ke dalam *array* milik variabel errors sehingga dapat ditampilkan.
16. Baris 57-60, deklarasi *method* closeAllert() untuk menangani *event* ketika pengguna menutup pesan error yang ditampilkan.
17. Uban Perangkat Node

Halaman ini berisi form input untuk mengubah informasi perangkat node tertentu. Ketika pertama kali dimuat, form input akan berisi informasi dari perangkat node yang dimaksud. Hasil akhir dari implementasi buat perangkat node dapat dilihat pada Gambar 5.5.



Gambar 5.5 Implementasi Aplikasi Web Console: Ubah Perangkat Node

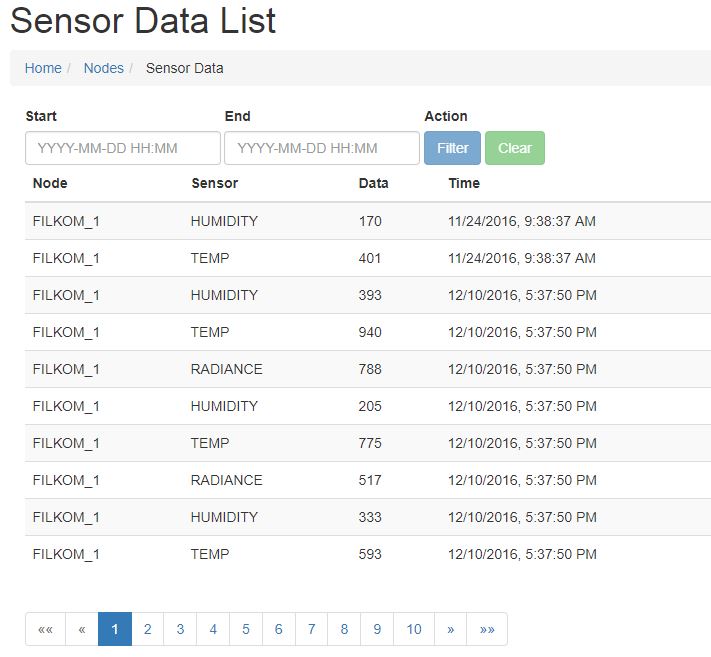
Kode sumber untuk melihat perangkat node disimpan pada file nodes/node-edit.component.ts dengan nama kelas NodeEditComponent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98 | ...  @Component({  ...  })  export class NodeEditComponent {  links: any[];  node: Node;  unlimited: boolean;  \_initial\_subsperday: number;  errors: Array<{ field: string, message: string}>;  constructor(  private nodeService: NodeService,  private route: ActivatedRoute,  private credentialsService: CredentialsService,  private router: Router  ) {}  ngOnInit() {  this.route.params  .switchMap((params: Params) => this.nodeService.getNode(params['id']))  .subscribe(  node => this.setUpNode(node),  error => console.log(error)  );  this.node = new Node;  }  private setUpNode(node: Node): void {  // raise 403 when node is not owned by this auth user  if (node.user != this.credentialsService.getUser().username) {  this.router.navigateByUrl('/403', { skipLocationChange: true });  }  this.node = node;  this.unlimited = (-1 == node.subsperday);  this.\_initial\_pubsperday = node.subsperday;  this.links = [  { label: "Home", url: "/" },  { label: "Nodes", url: "/nodes/" },  { label: this.node.label, url: `/nodes/view/${node.id}` },  { label: "Edit", is\_active: true }  ];  }  unlimitedStateChange(): void {  if (this.unlimited) {  this.node.pubsperday = -1;  } else {  this.node.pubsperday = (-1 == this.\_initial\_subsperday) ? 0 :  this.\_initial\_subsperday;  }  }  save(): void {  this.node.is\_public = this.node.is\_public ? 1 : 0;  this.nodeService.save(this.node)  .subscribe(  node => this.router.navigate(['/nodes/view', node.id]),  error => this.extractErrors(error)  );  }  delete(): void {  if(confirm("Are you sure?")) {  this.nodeService.delete(this.node.url)  .subscribe(  () => this.router.navigate(['/nodes/']),  error => this.extractErrors(error)  );  }  }  private extractErrors(err: any): void {  let errorsParse = JSON.parse(err.\_body);  this.errors = [];  for(let index in errorsParse) {  if(errorsParse.hasOwnProperty(index)) {  this.errors.push({  field: index,  message: errorsParse[index]  })  }  }  }  public closeAlert(alert: any) {  const index: number = this.errors.indexOf(alert);  this.errors.splice(index, 1);  }  } |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3-6, deklarasi component NodeEditComponent yang mengimplementasi interface OnInit.
2. Baris 7, deklarasi variabel nodeuntuk menampung data dari form input.
3. Baris 8, deklarasi variabel linksuntuk menampung navigasi URL yang tersedia.
4. Baris 9, deklarasi variabel unlimitedyang digunakan oleh form input bertipe *checkbox*. Nilai dari variabel ini nantinya akan dikonversi ke dalam bilangan *integer* yang menentukan pembatasan pengiriman dari perangkat node yang dibuat.
5. Baris 10, deklarasi variabel \_initial\_pubsperday untuk menyimpan nilai pembatasan pengiriman dari perangkat node sebelum dilakukan perubahan.
6. Baris 12, deklarasi variabel errorsuntuk menampung pesan kegagalan validasi dalam pembuatan perangkat node.
7. Baris 14-19, *constructor* yang berisi deklarasi *object* dari *class* lain yang digunakan pada *class* ini.
8. Baris 21-30, *override* dari *method* ngOnInit() dari *interface* OnInit. *Method* ini adalah bagian pertama yang akan dijalankan ketika *component* dibuat.
9. Baris 22-28, mendapatkan data perangkat node dari *web service*.
10. Baris 26, *object* perangkat node yang diterima dijadikan parameter untuk method setUpNode() .
11. Baris 27, memberikan nilai awal dari variabel node dengan *object* perangkat node. Hal tersebut untuk menghindari error ketika melakukan *render* form input karena variabel yang digunakan disana tidak memiliki nilai.
12. Baris 32-49, deklarasi *method* setUpNode() untuk memberikan nilai terhadap variabel node, links, unlimited, dan \_initial\_subsperday.
13. Baris 41-43, jika perangkat node tersebut bukan merupakan milik pengguna yang sedang terotentikasi, maka pengguna akan diarahkan ke halaman 403.
14. Baris 51-59, deklarasi *method* unlimitedStateChange() untuk mengkonversi nilai dari variabel unlimited ke dalam nilai awal node.subsperday.
15. Baris 61-69, deklarasi *method* save() untuk menangani *event* ketika pengguna menekan tombol “*Save*”.
16. Baris 62, mengkonversi nilai dari form input node.pubsperday ke dalam bilangan *integer* 1 atau 0.
17. Baris 63-68, mengirim nilai dari *object* perangkat node baru ke *web service* untuk disimpan.
18. Baris 65-66, jika pembuatan perangkat node berhasil, pengguna akan diarahkan ke halaman detail dari perangkat node yang baru dibuat tersebut.
19. Baris 67, jika pembuatan perangkat node tidak berhasil, pesan error yang diterima dijadikan parameter untuk method extractErrors().
20. Baris 71-79, deklarasi *method* delete() untuk menangani *event* ketika pengguna menekan tombol “*Delete*”.
21. Baris 72, membuka dialog konfirmasi hapus.
22. Baris 73-77, mengirim permintaan hapus perangkat node tersebut ke *web service*.
23. Baris 81-92, deklarasi *method* extractErrors() untuk mengkonversi pesan error ke dalam *array* milik variabel errors sehingga dapat ditampilkan.
24. Baris 94-97, deklarasi *method* closeAllert() untuk menangani *event* ketika pengguna menutup pesan error yang ditampilkan.
25. Lihat Data Sensor

Halaman ini berisi form input untuk mengubah informasi perangkat node tertentu. Ketika pertama kali dimuat, form input akan berisi informasi dari perangkat node yang dimaksud. Hasil akhir dari implementasi buat perangkat node dapat dilihat pada Gambar 5.6.



Gambar 5.6 Implementasi Aplikasi Web Console: Lihat Data Sensor

Kode sumber untuk melihat data sensor disimpan pada file sensordatas/sensordata.component.ts dengan nama kelas SensorDataComponent.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61 | ...  @Component({  ...  })  export class SensorDataComponent implements OnInit {  sensordatas: SensorData[];  links: any[];  ...  page = 1;  maxSize = 10;  collectionSize: number;  date\_start: string;  date\_end: string;  constructor(  private sensorDataService: SensorDataService,  private route: ActivatedRoute,  private router: Router,  ) {}    ngOnInit() {  this.getSensorData();  this.links = [  { label: "Home", url: "/" },  { label: "Nodes", url: "/nodes/" },  { label: "Sensor Data", is\_active: true }  ];  }  getSensorData(): void {  this.sensorDataService.getSensorDataByUser(this.page, this.date\_start, this.date\_end)  .subscribe(  sensordatas => {  this.collectionSize = sensordatas.count;  this.sensordatas = sensordatas.results as SensorData[];  },  error => console.log(error)  );  }  pageChange(): void {  this.router.navigateByUrl(`sensordata?page=${this.page}`);  this.getSensorData();  }  filter(): void {  this.page = 1;  this.pageChange();  }  clearFilter() {  this.page = 1;  this.date\_start = "";  this.date\_end = "";  this.pageChange();  }  } |

Penjelasan dari potongan kode program diatas adalah:

1. Baris 3-6, deklarasi component SensorDataComponent yang mengimplementasi interface OnInit.
2. Baris 7, deklarasi variabel sensordatasuntuk menampung *object* data sensor yang nantinya didapatkan dari web service.
3. Baris 8, deklarasi variabel linksuntuk menampung navigasi URL yang tersedia.
4. Baris 10, deklarasi variabel page untuk menampung nomor halaman paginasi yang sedang aktif. Variabel ini berisi nilai awal 1.
5. Baris 11, deklarasi variabel maxSize untuk menentukan jumlah sensor data yang ditampilkan setiap halaman paginasi berjumlah maksimal 10 data.
6. Baris 12, deklarasi variabel collectionSize untuk menampung jumlah semua data sensor sebelum dilakukan paginasi; yang nantinya didapatkan dari web service.
7. Baris 14, deklarasi variabel date\_start untuk menampung kriteria waktu “awal” dari *form input*.
8. Baris 15, deklarasi variabel date\_end untuk menampung kriteria waktu “akhir” dari *form input*.
9. Baris 17-21, *constructor* yang berisi deklarasi *object* dari *class* lain yang digunakan pada *class* ini.
10. Baris 23-30, *override* dari *method* ngOnInit() dari *interface* OnInit. *Method* ini adalah bagian pertama yang akan dijalankan ketika *component* dibuat.
11. Baris 25-28, memberikan nilai pada variabel links.
12. Baris 32-43, deklarasi *method* getSensorData() untuk mendapatkan data sensor dari *web service* kemudian menyimpannya ke dalam variabel sensordatas.
13. Baris 45-48, deklarasi *method* pageChange() untuk menangani *event* perubahan halaman paginasi. Ketika halaman berubah, nomor halaman aktif akan ditambahkan ke dalam URL, kemudian memanggil *method* getSensorData() untuk memperbarui data sensor yang ditampilkan.
14. Baris 50-53, deklarasi *method* filter()untuk menangani *event* ketika pengguna menekan tombol “*Filter*”. Jika ditekan, *method* getSensorData() akan dipanggil untuk memperbarui data sensor yang ditampilkan berdasarkan kriteria waktu yang diminta.
15. Baris 55-60, deklarasi *method* clear() untuk menangani *event* ketika pengguna menekan tombol “*Clear*”. Jika ditekan, nilai dari *form input* kriteria waktu akan dikosongkan kemudian mengatur ulang nilai dari variabel *page* menjadi 1.

# PENGUJIAN DAN ANALISIS HASIL PENGUJIAN

Pada bagian ini dibahas mengenai pengujian dan analisis hasil pengujian dari IoT *cloud platform* yang dikembangkan. Pengujian dilakukan untuk memastikan terpenuhinya kebutuhan pada bab analisis dan perancangan. Pengujian juga dapat memberi jawabah apakah IoT *cloud platform* yang dikembangkan dapat menyelesaikan masalah yang diangkat pada pendahuluan dan rumusan masalah.

## Perancangan Pengujian

Perancangan pengujian membahas mengenai batasan dan skenario yang digunakan dalam pengujian. Pada penelitian ini, perancangan pengujian meliputi: perancangan pengujian fungsional, perancangan pengujian sekuritas dan perancangan pengujian kehandalan sistem.

### Perancangan Pengujian Fungsional

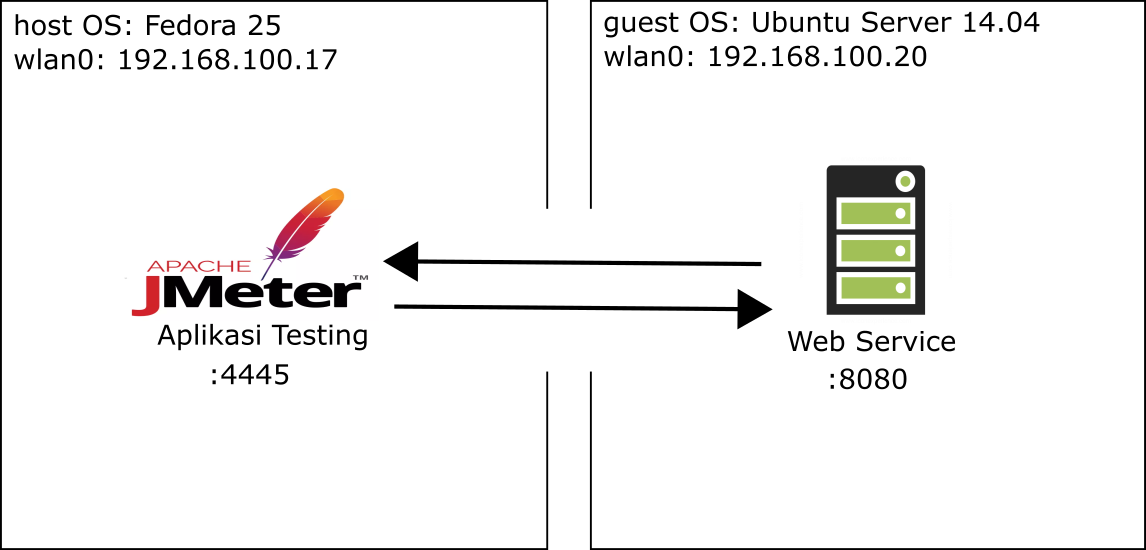
Perancangan pengujian fungsional berisi skenario untuk memastikan semua fitur telah sesuai dan dapat berjalan dengan benar. Keberhasilan pengujian ini ditentukan oleh kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Informasi mengenai skenario dalam melakukan pengujian fungsional tersebut dijelaskan pada Tabel 6.1 dibawah ini:

Tabel 6.1 Skenario Pengujian Fungsional

|  |  |  |
| --- | --- | --- |
| Kode | *Test Case* | Skenario |
| PF\_001 | Pengujian membuat token akses JWT bagi perangkat node. | 1. Perangkat node mengirimkan request HTTP POST dengan JSON payload: user, label, dan secretkey. 2. Sistem akan mengembalikan HTTP status 200 beserta token akses JWT untuk perangkat node. |
| PF\_002 | Pengujian membuat token akses JWT bagi aplikasi client. | 1. Perangkat node mengirimkan request HTTP POST dengan JSON payload: username dan password. 2. Sistem akan mengembalikan HTTP status 200 beserta token akses JWT untuk aplikasi client |
| PF\_003 | Pengujian registrasi pengguna. | 1. Aplikasi client mengirimkan request HTTP POST dengan JSON payload: username, password, email, firstname, dan lastname. 2. Sistem akan menyimpan data pengguna baru ke dalam sistem basis data. 3. Sistem akan mengembalikan HTTP status 201. |
| PF\_004 | Pengujian perangkat node mengirimkan data sensor ke IoT cloud platform untuk disimpan. | 1. Perangkat node mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish. 2. Sistem akan menyimpan data sensor ke dalam basis data. 3. Sistem akan mengembalikan HTTP status 200. |
| PF\_005 | Pengujian mengakses data sensor. | 1. Menguji kriteria berdasarkan pengguna 2. Aplikasi client mengirimkan request HTTP GET. 3. Sistem akan mengembalikan HTTP status 200 beserta data sensor. 4. Menguji kriteria berdasarkan perangkat node 5. Aplikasi client mengirimkan request HTTP GET. 6. Sistem akan mengembalikan HTTP status 200 beserta data sensor. 7. Menguji kriteria berdasarkan sensor 8. Aplikasi client mengirimkan request HTTP GET. 9. Sistem akan mengembalikan HTTP status 200 beserta data sensor. |
| PF\_006 | Pengujian membuat perangkat node. | 1. Aplikasi client mengirimkan request HTTP POST dengan JSON payload: user, label, is\_public, secretkey dan pubsperday. 2. Sistem akan menyimpan data perangkat node ke dalam basis data. 3. Sistem akan mengembalikan HTTP status 200. |
| PF\_007 | Pengujian melihat perangkat node. | 1. Aplikasi client mengirimkan request HTTP GET. 2. Sistem akan mengembalikan HTTP status 200 beserta semua perangkat node. |
| PF\_008 | Pengujian melihat perangkat node berdasarkan id. | 1. Aplikasi client mengirimkan request HTTP GET. 2. Sistem akan mengembalikan HTTP status 200 beserta perangkat node. |
| PF\_009 | Pengujian mengubah data perangkat node. | 1. Aplikasi client mengirimkan request HTTP POST dengan JSON payload berisi hanya field yang ingin diubah. 2. Sistem akan menyimpan perubahan data perangkat node ke dalam basis data. 3. Sistem akan mengembalikan HTTP status 200. |
| PF\_010 | Pengujian menghapus perangkat node. | 1. Aplikasi client mengirimkan request HTTP DELETE. 2. Sistem akan menghapus perangkat node di dalam basis data. 3. Sistem akan mengembalikan HTTP status 204. |
| PF\_011 | Pengujian Otentikasi | 1. Pengujian perangkat node mengirimkan data sensor tanpa token akses JWT 2. Perangkat node mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish ke alamat URL /sensordatas. 3. Sistem akan mengembalikan HTTP status 401 beserta pesan "Authentication credentials were not provided." 4. Pengujian aplikasi client mengakses data sensor tanpa token akses JWT 5. Aplikasi client mengirimkan request HTTP GET ke alamat URL /sensordatas/user/<user-username>. 6. Sistem akan mengembalikan HTTP status 401 beserta pesan "Authentication credentials were not provided." |
| PF\_012 | Pengujian Otorisasi | 1. Pengujian perangkat node mengirimkan data sensor menggunakan token akses JWT milik aplikasi client 2. Perangkat node mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish ke alamat URL /sensordatas. 3. Sistem akan mengembalikan HTTP status 403 beserta pesan "You do not have permission to perform this action." 4. Pengujian aplikasi client mengakses data sensor menggunakan token akses JWT milik perangkat node 5. Aplikasi client mengirimkan request HTTP GET ke alamat URL /sensordatas/user/<user-username>. 6. Sistem akan mengembalikan HTTP status 403 beserta pesan "You do not have permission to perform this action." 7. Pengujian mengirim data sensor oleh perangkat node yang telah habis batas pengiriman per harinya 8. Perangkat node mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish ke alamat URL /sensordatas. 9. Sistem akan mengembalikan HTTP status 403 beserta pesan "Publish is limit." |

### Perancangan Pengujian Kehandalan Sistem

Pengujian kehandalan sistem dilakukan untuk mengetahui kinerja dari IoT *cloud platform* yang diajukan. Kinerja tersebut berkaitan dengan dari protokol komunikasi yang digunakan, yaitu HTTP. Pengujian dilakukan dengan mengamati proses komunikasi antar entitas menggunakan beberapa nilai parameter yang ditentukan seperti *troughput*, *latency* dan *error rate*. Agar pengujian dapat dilakukan dan memberikan hasil yang tepat, diperlukan rancangan topologi untuk menggambarkan bagaimana entitas saling terhubung dan mengirimkan data. Rancangan topologi ini dapat dilihat pada Gambar 6.1 berikut:



Gambar 6.1 Perancangan Pengujian Kehandalan Sistem

Pengujian ini menggunakan dua sistem berbeda yaitu sistem operasi Fedora sebagai host dan sistem virtual dengan sistem operasi Ubuntu server sebagai *guest*. Sistem operasi host menjalankan aplikasi testing JMeter pada port 4445 untuk menguji kehandalan dari *web service* yang dikembangkan. Sedangkan sistem operasi virtual berperan sebagai *cloud* yang menjalankan *web service* pada port 8080. Pada sistem operasi host terdapat *interface* wlan0 dengan alamat ip local 192.168.100.17. Aplikasi testing JMeter pada sistem operasi host berkomunikasi dengan *web service* yang beralamat di 192.168.100.20: 8080. Baik sistem operasi host atau sistem operasi guest sama-sama terhubung menggunakan interface wlan0, dimana gateway dari jaringan tersebut berada pada alamat 192.168.100.1. Fitur yang dilakukan pengujian kehandalan merupakan fitur utama dari IoT *cloud platform* yang diajukan, yakni: fitur mengirimkan data sensor dan fitur mengakses data sensor.

Pada fitur pengiriman data sensor, aplikasi pengujian JMeter berperan sebagai perangkat node dengan menggunakan token akses JWT milik perangkat dengan label FILKOM\_1. Untuk itu, aplikasi JMeter perlu diatur untuk mengirimkan *request* HTTP POST pada alamat /sensordatas/ dengan format seperti yang telah ditentukan pada Tabel 4.14. Karena perangkat node tersebut memiliki dua sensor yaitu HUMIDITY dan TEMP, *payload* yang dikirimkan memiliki format seperti dibawah ini:

{

"publish": [

{

"sensor": "HUMIDITY",

"data": "145"

},

{

"sensor": "TEMP",

"data": "30"

}

]

}

Skenario pada pengujian ini menggunakan jumlah *request* yang dikirimkan secara bersamaan dengan jumlah *request* sebanyak 50, 100, dan 150.

Sedangkan pada fitur mengakses data sensor, aplikasi pengujian JMeter berperan sebagai aplikasi *client* dengan menggunakan token akses JWT milik pengguna dengan username basukicahya. Kriteria yang digunakan dalam mengakses data sensor adalah kriteria untuk mendaatkan semua data sensor milik pengguna. Untuk itu, aplikasi JMeter perlu diatur untuk mengirimkan *request* HTTP GETke alamat /sensordatas/user/basukicahya/. Skenario pada pengujian ini menggunakan jumlah *request* yang dikirimkan secara bersamaan bersamaan dengan jumlah *request* sebanyak 50, 100, dan 150.

Pengujian dari kedua fitur tersebut, dijalankan masing-masing skenario sebanyak 3 kali untuk mendapatkan hasil yang akurat. Setelah itu, hasil yang didapat dari pengujian kehandalan sistem dilakukan proses analisis dengan membandingkan beberapa parameter uji terkait kehandalan sistem yaitu *troughput*, *latency* dan *error rate*.

## Hasil dan Analisis Pengujian

Pengujian terhadap sistem dilakukan berdasarkan skenario-skenario dari perancangan pengujian diatas. Hasil dari pengujian tersebut kemudian diolah dan dianalisis untuk mengetahui apakah telah sesuai dengan tahap kebutuhan dan perancangan sebelumnya, memiliki otorisasi dan otentikasi yang sesuai, dan kehandalan yang diharapkan.

### Hasil dan Analisis Pengujian Fungsional

Bagian ini membahas mengenai hasil dan analisis dari pengujian fungsional yang dilakukan. Pengujian ini dilakukan untuk memastikan semua fitur telah sesuai dan dapat berjalan dengan benar. Keberhasilan pengujian ditentukan oleh kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Hasil dari pengujian fungsional dapat dilihat pada Tabel 6.2.

Tabel 6.2 Hasil Pengujian Fungsional

|  |  |  |  |
| --- | --- | --- | --- |
| Kode | Deskripsi | Hasil yang Didapat | Status |
| PF\_001 | Pengujian membuat token akses JWT bagi perangkat node. | * Sistem mengembalikan HTTP status 200 beserta token akses JWT untuk perangkat node. | valid |
| PF\_002 | Pengujian membuat token akses JWT bagi aplikasi client. | * Sistem akan mengembalikan HTTP status 200 beserta token akses JWT untuk aplikasi *client* | valid |
| PF\_003 | Pengujian registrasi pengguna. | * Sistem menyimpan data pengguna baru ke dalam sistem basis data * Sistem mengembalikan HTTP status 201 | valid |
| PF\_004 | Pengujian perangkat node mengirimkan data sensor ke IoT cloud platform untuk disimpan. | * Sistem menyimpan data sensor ke dalam basis data * Sistem mengembalikan HTTP status 200 | valid |
| PF\_005 | Pengujian mengakses data sensor. | * Sistem mengembalikan HTTP status 200 beserta data sensor | valid |
| PF\_006 | Pengujian membuat perangkat node. | * Sistem menyimpan data perangkat node ke dalam basis data * Sistem mengembalikan HTTP status 200 | valid |
| PF\_007 | Pengujian melihat perangkat node. | * Sistem mengembalikan HTTP status 200 beserta semua perangkat node | valid |
| PF\_008 | Pengujian melihat perangkat node berdasarkan id. | * Sistem mengembalikan HTTP status 200 beserta perangkat node | valid |
| PF\_009 | Pengujian mengubah data perangkat node. | * Sistem menyimpan perubahan data perangkat node ke dalam basis data * Sistem mengembalikan HTTP status 200 | valid |
| PF\_010 | Pengujian menghapus perangkat node. | * Sistem menghapus perangkat node di dalam basis data * Sistem mengembalikan HTTP status 204 | valid |
| PF\_011 | Pengujian otentikasi. | * Sistem mengembalikan HTTP status 401 beserta pesan "Authentication credentials were not provided." | valid |
| PF\_012 | Pengujian otorisasi. | * Sistem mengembalikan HTTP status 403 beserta pesan "You do not have permission to perform this action." | valid |

Dari Tabel 6.2 diatas didapati bahwa semua fitur yang diuji telah sesuai dan dapat berjalan dengan benar. Pada pengujian 1 dan 2 tentang pembuatan token akses JWT untuk perangkat node dan aplikasi *client*, diketahui bahwa sistem dapat menghasilkan token sesuai dengan payload yang dikirimkan. Pengujian fitur registrasi pada nomor 3 berhasil membuat akun pengguna baru sesuai dengan data pengguna yang terdapat pada payload. Selain itu, pengujian pengiriman data sensor pada nomor 4 berhasil dilakukan, karena sistem berhasil menyimpan data sensor tersebut ke dalam basis data. Pengujian nomor 5 tentang fitur melihat data sensor, pengujian fitur manajemen perangkat pada nomor 6-10, serta pengujian otorisasi dan otentikasi pada nomor 11-12 dinyatakan berhasil karena telah sesuai dengan keluaran yang diharapkan

Kesimpulan dari hasil pengujian ini adalah pengujian fitur pada semua skenario memiliki kesesuaian antara hasil yang didapat dengan keluaran yang diharapkan. Hal tersebut dapat memberikan kesimpulan lain bahwa pengujian fungsional telah berhasil dilakukan dan sistem dapat menjalankan operasi-operasi yang dibutuhkan. Selain itu, keberhasilan fitur juga menggambarkan bahwa integrasi antara perangkat node dan aplikasi *client* terhadap *web service* dapat terhubung dan saling berkomunikasi dengan baik melalui protokol komunikasi HTTP.

### Hasil dan Analisis Pengujian Kehandalan Sistem

Bagian ini membahas mengenai hasil dan analisis dari pengujian kehandalan sistem. Pengujian dilakukan berdasarkan rancangan topologi dan skenario uji yang telah ditentukan pada sub bab 6.2.3. Masing-masing skenario dijalankan sebanyak 3 kali dengan menggunakan aplikasi pengujian JMeter. Fitur yang dilakukan pengujian kehandalan merupakan fitur utama dari IoT *cloud platform* yang diajukan, yakni: fitur mengirimkan data sensor dan fitur mengakses data sensor.

1. Pengujian Fitur Mengakses Data Sensor

Pada IoT *cloud platform* yang dikembangkan ini, terdapat tiga kriteria dalam mengakses data sensor yakni berdasarkan pengguna, berdasarkan perangkat node dan berdasarkan sensor. Sehingga pada fitur ini dilakukan pengujian sebanyak tiga kali berdasarkan tiga kriteria tersebut. Hasil dari pengujian ini ditampilkan pada Tabel 6.3, Tabel 6.4 dan Tabel 6.5. Mengacu pada hasil tersebut didapati bahwa ketika *request* dikirimkan secara bersamaan (dapat dianalogikan satu *request* degan satu user) berjumlah 50 dan 100 secara bersamaan, sistem menangani permintaan tersebut dengan keberhasilan 100%. Disisi lain, ketika *request* yang dikirimkan berjumlah 150, terdapat 2.08% dan 1.78% *request* yang tidak tertangani untuk kriteria berdasarkan pengguna dan berdasarkan sensor.

**Tabel 6.3 Hasil Pengujian Performa Dalam Mengakses Data Sensor Berdasarkan Pengguna**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Thread (users) | Expected | Actual | Success rate % | Request Rejected rate % | Throughput / s | Latency (ms) |
| 50 | 150 | 150 | 100% | 0% | 18.07649 | 1777.753333 |
| 100 | 300 | 300 | 100% | 0% | 17.95819333 | 1800.44 |
| 150 | 450 | 441 | 97.92% | 2.08% | 17.69729333 | 1861.106667 |

**Tabel 6.4 Hasil Pengujian Performa Dalam Mengakses Data Sensor Berdasarkan Perangkat Node**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Thread (users) | Expected | Actual | Success rate % | Request Rejected rate % | Throughput / s | Latency (ms) |
| 50 | 150 | 150 | 100% | 0% | 46.35336333 | 668.2866667 |
| 100 | 300 | 300 | 100% | 0% | 45.48874333 | 768.8733333 |
| 150 | 450 | 450 | 100% | 0% | 43.97515667 | 910.7933333 |

**Tabel 6.5 Hasil Pengujian Performa Dalam Mengakses Data Sensor Berdasarkan Sensor**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Thread (users) | Expected | Actual | Success rate % | Request Rejected rate % | Throughput / s | Latency (ms) |
| 50 | 150 | 150 | 100% | 0% | 34.59481667 | 636.6062222 |
| 100 | 300 | 300 | 100% | 0% | 30.74651 | 1210.92 |
| 150 | 450 | 442 | 98.22% | 1.78% | 20.17453333 | 1754.48 |

Gambar 6.2 Grafik Perbandingan Nilai Throughput Pada Fitur Mengakses Data Sensor

Perbandingan nilai *throughput* diantara ketiga kriteria dalam mengakses data sensor dapat memberitahukan berapa banyak *request* yang dapat ditangani sistem setiap detiknya. Berdasarkan grafik perbandingan nilai *throughput* pada Gambar 6.2 diatas, terlihat bahwa mengakses data sensor menggunakan kriteria berdasarkan sensor menghasilkan *throughput* terendah dan kriteria berdasarkan perangkat node menghasilkan *throughput* tertinggi. Hal ini berkaitan dengan jumlah dan kompleksitas kueri yang diperlukan untuk mendapatkan data sensor yang sesuai. Seperti yang sudah dijelaskan pada bab kajian pustaka, bahwa sistem basis data MongoDB tidak memiliki fitur join sehingga untuk mendapatkan data dari dokumen berbeda memerlukan lebih dari satu kueri. Pada kriteria berdasarkan sensor, diperlukan dua kueri kompleks yaitu kueri untuk mendapatkan *list* sensor milik perangkat node yang dimaksud dan kueri untuk melakukan filter data sensor berdasarkan *list* sensor. Sedangkan untuk kriteria berdasarkan perangkat node hanya diperlukan satu kueri standar, yaitu melakukan filter data sensor berdasarkan id perangkat node yang terdapat dalam URL. Hal tersebut menjadikan eksekusi kueri untuk kriteria berdasarkan perangkat node menjadi jauh lebih cepat.

Gambar 6.3 Grafik Perbandingan Nilai Latency Pada Fitur Mengakses Data Sensor

*Latency* merupakan nilai jeda yang dibutuhkan untuk mengirimkan pesan dari pengirim ke penerima dalam jaringan komputer. Pada kasus ini *latency* yang dihitung adalah jeda waktu dari *web service* ke aplikasi pengujian JMeter ketika memberikan *response* berisi data sensor yang diminta. Perbandingan nilai *latency* dapat dilihat pada Gambar 6.3. Berdasarkan gambar tersebut didapati bahwa ketika *request* yang dikirimkan berjumlah 50 secara bersamaan, *latency* yang didapat adalah 1777.75/ms untuk kriteria berdasarkan pengguna, 668.28/ms untuk kriteria berdasarkan perangkat node, dan 1025.78/ms untuk kriteria berdasarkan sensor. Ketika *request* yang dikirimkan berjumlah 100 secara bersamaan, *latency* yang didapat meningkat menjadi 1800.44/ms, 786.87/ms dan 1210,92/ms. Sedangkan untuk *request* secara bersamaan dengan jumlah 150, nilai *latency* justru meningkat cukup signifikan menjadi 1861.10/ms, 910.79/ms dan 1754.48/ms.

1. Pengujian Fitur Mengirim Data Sensor

Pengujian kehandalan berikutnya yakni pengujian terhadap fitur mengirimkan data sensor, hasil dari aplikasi JMeter ditampilakan pada Tabel 6.6. Berdasarkan pada hasil tersebut didapati bahwa ketika *request* dikirimkan secara bersamaan (dapat dianalogikan satu *request* degan satu user) berjumlah 50, 100 dan 150 secara bersamaan, sistem dapat menangani permintaan tersebut dengan keberhasilan 100%.

**Tabel 6.6 Hasil Pengujian Performa Dalam Mengirim Data Sensor**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Thread (users) | Expected | Actual | Success rate % | Request Rejected rate % | Throughput / s | Latency (ms) |
| 50 | 150 | 150 | 100% | 0% | 26.28006 | 1194.26 |
| 100 | 300 | 300 | 100% | 0% | 25.98759667 | 1270.373333 |
| 150 | 450 | 450 | 100% | 0% | 22.94452 | 1737.88 |

Gambar 6.4 Grafik Perbandingan Nilai Throughput Pada Fitur Mengirim Data Sensor

Perbandingan nilai *throughput* dari fitur mengirim data sensor dapat dilihat pada Gambar 6.4. Perbandingan tersebut dapat memberitahu berapa banyak user yang dapat ditangani sistem setiap detiknya. Berdasarkan gambar tersebut, didapati bahwa ketika *request* yang dikirimkan oleh 50 user secara bersamaan, nilai *throughput* yang didapat adalah 26.28/detik. Ketika user berjumlah 100, *throughput* menurun menjadi 25.98/detik karena jumlah *request* yang dikirimkan lebih besar dari sebelumnya. Begitu juga hasil yang didapat dari jumlah user sebanyak 150, nilai *throughput* juga turun menjadi 22.94/detik.

Gambar 6.5 Grafik Perbandingan Nilai Latency Pada Fitur Mengirim Data Sensor

Pada kasus ini *latency* yang dihitung adalah jeda waktu dari aplikasi pengujian JMeter ke *web service* ketika memberikan *request* berisi data sensor yang ingin disimpan. Perbandingan nilai *latency* dapat dilihat pada Gambar 6.5. Berdasarkan gambar tersebut didapati bahwa ketika *request* yang dikirimkan oleh user berjumlah 50 secara bersamaan, *latency* yang didapat adalah 1194.26/ms. Ketika *request* yang dikirimkan berjumlah 100 secara bersamaan, *throughput* yang didapat meningkat menjadi 1270.37/ms dikarenakan semakin banyak *request* yang dikirimkan semakin lama pula jeda waktu yang dibutuhkan. Seperti halnya pada perbandingan *throughput*, hal menarik juga terjadi pada *request* secara bersamaan dengan jumlah 150, nilai *latency* justru menurun cukup signifikan menjadi 1737.88/ms.

Kesimpulan dari hasil pengujian ini adalah pengujian kehandalan pada semua scenario, sistem dapat menangani <100 pengguna dengan keberhasilan 100%.

# KESIMPULAN DAN SARAN

Bab ini berisikan uraian kesimpulan beserta saran untuk penelitian selanjutnya. Uraian kesimpulan dibuat berdasarkan hasil dari tahap perancangan, implementasi dan pengujian yang telah dilakukan sebelumnya.

## Kesimpulan

Berdasarkan hasil dari tahap perancangan, implementasi dan pengujian yang telah dilakukan sebelumnya, untuk menjawab pertanyaan pada rumusan masalah dapat disimpulkan bahwa:

Protokol HTTP dapat diterapkan dengan menjalankan aplikasi *web server* sehingga aplikasi web dapat melayani permintaan dari *client*. Untuk menjalankan aplikasi web yang ditulis menggunakan Bahasa pemrograman Python dibutuhkan modul mod\_wsgi. Modul mod\_wsgi memungkinkan aplikasi web berbahasa Python dapat dijalankan sebagai CGI.

Pengiriman data antara perangkat node dan aplikasi *client* terhadap IoT *cloud* *platform* harus melalui RESTful *web service* yang menyediakan aturan baku untuk menjalankan fitur-fitur yang disediakan.

Mekanisme manajemen perangkat, mengirimkan dan mengakses data sensor dapat dilakukan dengan menyediakan RESTful *web service* sebagai antarmuka standar untuk berkomunikasi. Dalam hal ini terdapat dua entitas yang berkomunikasi dengan *web service*, yaitu perangkat node yang bertindak mengirimkan data sensor dan alplikasi *client* yang bertindak mengakses data sensor dan memanajemen perangkat.

Berdasarkan hasil pengujian kehandalan sistem didapati bahwa sistem dapat menangani 100 user secara bersamaan dengan *success rate* sebesar100%.

## Saran

Setelah menyelesaikan penelitian ada beberapa saran yang dapat disampaikan oleh penulis guna untuk mengembangkan perangkat lunak IoT Cloud Platform berbasis protokol komunikasi HTTP:

Penelitian selanjutnya dapat mengembangkan perangkat *relay* antara perangkat node dengan IoT *cloud platform*.

IoT *cloud platform* ini dapat dikembangkan lebih lanjut dengan menambahkan komputasi cerdas untuk mengolah data sensor untuk keperluan yang lebih luas.

IoT *cloud platform* ini dapat dikembangkan lebih lanjut dengan menambahkan fitur untuk memvisualisasikan data sensor kedalam format tertentu, misalnya grafik.

# DAFTAR PUSTAKA

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., & Ayyash, M. (2015). Internet of things: A survey on enabling technologies, protocols, and applications. *IEEE Communications Surveys & Tutorials Vol. 17(4)*, 2347-2376.

Botta, A., De Donato, W., Persico, V., & Pescapé, A. (2016). Integration of Cloud computing and Internet of Things: A survey. *Future Generation Computer Systems*, 684–700.

Chodorow, K. (2013). *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage.* O'Reilly Media.

Doug, T., Snell, James, & Pavel, K. (2001). *Programming web services with SOAP: building distributed applications.* O'Reilly Media, Inc.

Douzis, K., Sotiriadis, S., Petrakis, E. G., & Amza, C. (2016). Modular and generic IoT management on the cloud. *Future Generation Computer Systems*.

Ganguly, P. (2016). Selecting the right IoT cloud platform. *Internet of Things and Applications (IOTA), International Conference on. IEEE*.

Gourley, D., & Brian, T. (2002). *HTTP: the definitive guide.* O'Reilly Media, Inc.

Guoqiang, S., Yanming, C., Chao, Z., & Yanxu, Z. (2013). Design and implementation of a smart IoT gateway. *Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing*.

IETF. (1999). *Hypertext Transfer Protocol--HTTP/1.1.* Retrieved from Internet Engineering Task Force: https://www.ietf.org/rfc/rfc2616.txt

IETF. (2014). *The JavaScript Object Notation (JSON) Data Interchange Format*. Retrieved April 29, 2017, from Internet Engineering Task Force: https://tools.ietf.org/pdf/rfc7159

IETF. (2015). *JSON Web Token (JWT)*. Retrieved Mei 30, 2017, from Internet Engineering Task Force: https://tools.ietf.org/pdf/rfc7159

Kurose, J. F. (2013). *Computer Network: A Top-Down Approach Featuring the Internet, 6/E.* Pearson Education India.

MongoDB. (2017). *The MongoDB 3.4 Manual.* Retrieved May 1, 2017, from http://docs.mongodb.com/master/MongoDB-manual.epub

Mozilla. (2017). *HTTP Header*. Retrieved April 22, 2017, from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

Richardson, Leonard, Mike, A., & Sam, R. (O'Reilly Media, Inc). *RESTful Web APIs: Services for a Changing World.* 2013.

Sandoval, J. (2009). *Restful java web services: Master core rest concepts and create restful web services in Java.* Packt Publishing Ltd.

Singh, Anand, & Yannis, V. (2016). An SLA-based resource allocation for IoT applications in cloud environments. *Cloudification of the Internet of Things (CIoT) (pp. 1-6)*.

Techterms. (2016, Juny). *API Definition*. Retrieved April 26, 2017, from https://techterms.com/definition/api

Zhang, Q., Lu, C., & Raouf, B. (2010). Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications 1.1*, 7-18.