PENGEMBANGAN 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

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2017

PENGESAHAN

PENGEMBANGAN PERANGKAT LUNAK IOT CLOUD PLATFORM 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
28 Desember 2017
Telah diperiksa dan disetujui oleh:

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

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

IDENTITAS TIM PENGUJI

Ketua Majelis (Dosen Penguji I)	: Fariz Andri Bakhtiar, S.T, M.Kom
Dosen Penguji II	: Widhi Yahya, S.Kom, M.Sc
Dosen Pembimbing I	: Adhitya Bhawiyuga, S.Kom, M.S
Dosen Pembimbing II	: Kasyful Amron, S.T, M.Sc

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, 10 Januari 2018

Ocki Bagus Pratama

NIM: 135150207111060

DAFTAR RIWAYAT HIDUP

DATA PRIBADI :

Nama : Ocki Bagus Pratama
Tempat / Tanggal Lahir : Tangerang, 06 Oktober 1995
Jenis Kelamin : Laki-laki
Status : Belum Menikah
Agama : Islam
Kebangsaan : Indonesia
Alamat : Jln. Darma Kusuma Kp. Pasir Limus, RT/RW 007/004, Ds. Wangun Harja, Kec. Cikarang Utara, Kab. Bekasi, Jawa Barat
No. Telp : 0812 3388 5220

PENDIDIKAN :

SD Negeri Wangun Harja 01 (2001-2007)
SMP Negeri 4 Cikarang Utara (2007-2010)
MA Negeri 1 Kab. Bekasi (2010-2011)
SMA Negeri 1 Cikarang Utara (2011-2013)
Program Studi Teknik Informatika Fakultas Ilmu Komputer Universitas Brawijaya

UCAPAN TERIMA KASIH

Puji dan syukur atas kehadirat Allah SWT atas limpahan rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi dengan judul "Pengembangan Perangkat Lunak IoT *Cloud Platform* Berbasis Protokol Komunikasi HTTP". Skrispi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer.

Untuk menyelesaikan penulisan skripsi ini, penulis tidak lepas dari bantuan dari berbagai pihak yang telah banyak memberikan bantuan serta dukungan dalam hal sekecil apapun. Dalam kesempatan ini penulis ingin mengucapkan terima kasih kepada :

1. Bapak Wayan Firdaus Mahmudy, S.Si, M.T, Ph.D, Selaku Dekan Fakultas Ilmu Komputer Program Studi Teknik Informatika Universitas Brawijaya yang telah memberikan ijin penelitian.
2. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D, Selaku Ketua Jurusan Informatika Fakultas Ilmu Komputer Universitas Brawijaya yang telah menyetujui permohonan penyusunan Skripsi.
3. Bapak Agus Wahyu Widodo, S.T, M.Cs, Selaku Ketua Program Studi Fakultas Ilmu Komputer Program Studi Teknik Informatika Universitas Brawijaya.
4. Bapak Adhitya Bhawiyuga, S.Kom, M.S selaku dosen pembimbing yang telah dengan sabar, tekun, tulus, dan ikhlas meluangkan waktu, tenaga, serta pikiran dalam memberikan arahan, bimbingan, motivasi dan saran-saran yang sangat berharga kepada penulis selama penyusunan skripsi ini.
5. Bapak Kasyful Amron, S.T, M.Sc selaku dosen pembimbing yang telah dengan sabar, tekun, tulus, dan ikhlas meluangkan waktu, tenaga, serta pikiran dalam memberikan arahan, bimbingan, motivasi dan saran-saran yang sangat berharga kepada penulis selama penyusunan skripsi ini.
6. Orang tua, keluarga, *team rocket*, teman-teman dan seluruh orang yang telah banyak memberikan dukungan, semangat dan doa untuk penyelesaian skripsi ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak terdapat kekurangan, sehingga penulis mengharapkan adanya saran dan kritik yang bersifat membangun. Semoga skripsi ini dapat bermanfaat bagi semua dan berguna untuk pengembangan ilmu pengetahuan.

Malang, 10 Januari 2018

Ocki Bagus Pratama

ocki.bagus.p@gmail.com

ABSTRAK

Internet of Things (IoT) bertujuan untuk memperluas manfaat dari konektivitas internet dengan menjadikan benda-benda di sekitar kita dapat dikontrol dan diakses melalui internet. Tetapi, perangkat dalam IoT sering kali memiliki keterbatasan dalam hal kapasitas penyimpanan dan kemampuan komputasi. Hal tersebut menjadikan perangkat IoT memerlukan integrasi dengan sistem yang memiliki kemampuan komputasi yang lebih baik. Salah satunya adalah sistem komputasi berbasis *cloud*. Namun, integrasi antara perangkat IoT dan *cloud* memiliki tantangan dalam hal jaringan komunikasi, keamanan dan manajemen perangkat. Jaringan komunikasi yaitu banyaknya teknologi jaringan yang digunakan, sehingga pemilihan protokol komunikasi harus mempertimbangkan dukungan oleh banyak perangkat. Keamanan terjadi karena banyaknya perangkat IoT yang dapat dengan mudah terhubung *cloud*, sehingga peran autentikasi dan otorisasi diperlukan untuk mengidentifikasi dan memvalidasi perangkat yang mengirimkan data. Manajemen perangkat diperlukan karena dengan banyaknya perangkat IoT yang digunakan membutuhkan mekanisme untuk memanajemen perangkat. Berdasarkan penjelasan sebelumnya, diusulkan sebuah rancang bangun IoT *cloud platform* menggunakan protokol komunikasi HTTP untuk menyelesaikan kendala jaringan komunikasi dan RESTful untuk manajemen perangkat. Sedangkan untuk mengidentifikasi dan memvalidasi perangkat yang mengirimkan data, digunakan mekanisme autentikasi dan otorisasi menggunakan JSON *Web Token*. Hasil pengujian performa sistem menunjukan, sistem yang dibangun mampu menangani hingga 100 pengguna secara bersamaan.

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

ABSTRACT

The primary aim of Internet of Things (IoT) is to expand the benefits of Internet connectivity by connecting objects around us to the internet, thus enabling remote control and accessibility. IoT devices often have limitations in terms of storage capacity and computing capabilities. This requires the IoT device to be integrated with systems that have better computing capabilities One of them is cloud-based computing system. However, the integration between IoT and cloud devices has challenges in terms of network communication, security, and device management. Network communication is the number of network technologies being used, thus the selection of communication protocols should consider its support for a variety of devices. Security and privacy are a concern because of the large number of IoT devices that can be easily connect to the cloud, requiring authentication and authorization that is required to identify and validate devices that transmit the data. Device management is required due to the large number of IoT devices that will be used, it requires a device management mechanism. Based on those problems, a solution was proposed on IoT cloud platform design using HTTP communication protocol to solve network communication and RESTful for device management constraints. As for identifying and validating devices that transmit data, an authentication and authorization mechanism using JSON Web Token is utilized. The result of system performance testing, is that the built systems are capable of handling up to 100 users simultaneously.

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

PENGANTAR

Puji dan syukur atas kehadirat Allah SWT atas limpahan rahmat dan karunia-Nya, sehingga penulis dapat menyelesaikan skripsi dengan judul "Pengembangan Perangkat Lunak IoT *Cloud Platform* Berbasis Protokol Komunikasi HTTP". Skrispi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer.

Untuk menyelesaikan penulisan skripsi ini, penulis tidak lepas dari bantuan dari berbagai pihak yang telah banyak memberikan bantuan serta dukungan dalam hal sekecil apapun. Dalam kesempatan ini penulis ingin mengucapkan terima kasih kepada :

1. Bapak Adhitya Bhawiyuga, S.Kom, M.S selaku dosen pembimbing yang telah dengan sabar, tekun, tulus, dan ikhlas meluangkan waktu, tenaga, serta pikiran dalam memberikan arahan, bimbingan, motivasi dan saran-saran yang sangat berharga kepada penulis selama penyusunan skripsi ini.
2. Bapak Kasyful Amron, S.T, M.Sc selaku dosen pembimbing yang telah dengan sabar, tekun, tulus, dan ikhlas meluangkan waktu, tenaga, serta pikiran dalam memberikan arahan, bimbingan, motivasi dan saran-saran yang sangat berharga kepada penulis selama penyusunan skripsi ini.
3. Orang tua, keluarga, *team rocket*, teman-teman dan seluruh orang yang telah banyak memberikan dukungan, semangat dan doa untuk penyelesaian skripsi ini.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak terdapat kekurangan, sehingga penulis mengharapkan adanya saran dan kritik yang bersifat membangun. Semoga skripsi ini dapat bermanfaat bagi semua dan berguna untuk pengembangan ilmu pengetahuan.

Malang, 10 Januari 2018

Ocki Bagus Pratama

ocki.bagus.p@gmail.com

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Internet of Things (IoT) merujuk pada suatu jaringan yang menghubungkan perangkat fisik di berbagai jaringan menggunakan berbagai protokol berbeda (Guoqiang et. al., 2013). IoT bertujuan untuk memperluas manfaat dari konektivitas internet dengan menjadikan benda-benda di sekitar kita dapat terhubung ke internet, sehingga dapat dikontrol dan diakses dari jarak jauh. 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. IoT dapat diaplikasikan ke dalam berbagai aktivitas di segala bidang sehingga pertukaran informasi melalui internet menjadi lebih mudah. 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 didistribusikan secara luas dan memiliki keterbatasan dalam hal kapasitas penyimpanan dan kemampuan komputasi. Hal tersebut menciptakan kekhawatiran dalam hal performa, kinerja, keamanan, dan privasi pada layanan yang akan dibangun (Botta, et. al., 2016). Untuk mengatasi kekhawatiran tersebut, perangkat IoT perlu diintegrasikan dengan sistem lain yang menawarkan kemampuan komputasi yang lebih baik. Salah satunya adalah sistem komputasi berbasis *cloud*. Menurut Zhang, et., all (2010) *cloud computing* merupakan model komputasi baru di mana 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 di lapangan. Pada sistem tersebut, data sensor sering kali diambil secara terus menerus sehingga menghasilkan data dalam ukuran yang besar. Keterbatasan pada perangkat IoT membuat penyimpanan dan data yang besar tidak bisa dilakukan di perangkat IoT 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 (Al-Fuqaha, et. al., 2015).

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 jaringan komunikasi dan keamanan. Masalah jaringan komunikasi terjadi karena CloudIoT melibatkan banyak teknologi jaringan, sehingga pemilihan protokol komunikasi harus mempertimbangkan dukungan oleh banyak perangkat. Sedangkan masalah keamanan terjadi karena banyaknya perangkat di luar sana yang dapat dengan mudah terhubung dengan CloudIoT, sehingga peran autentikasi dan otorisasi diperlukan untuk mengidentifikasi dan memvalidasi perangkat yang mengirimkan data. Tantangan selanjutnya adalah dengan banyaknya perangkat IoT yang digunakan, dibutuhkan mekanisme untuk memanajemen perangkat.

Sebagai salah satu solusi dari permasalahan dalam mengintegrasikan IoT dan *cloud computing*, penelitian ini membuat rancang bangun IoT *cloud platform* yang memungkinkan perangkat IoT terhubung dan berinteraksi dengan aplikasi *cloud* dengan aman dan tervalidasi. 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 jaringan komunikasi, pada IoT *cloud platform* ini digunakan protokol komunikasi HTTP. Protokol HTTP dipilih karena merupakan protokol sederhana yang telah banyak digunakan, sehingga didukung oleh banyak perangkat. Sedangkan untuk menyelesaikan masalah keamanan, digunakan mekanisme autentikasi 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 IoT melalui protokol HTTP. Komponen *security* merupakan bagian yang menangani manajemen perangkat, autentikasi dan otorisasi yang berbasis JSON *Web Token*. Komponen manajemen data menangani mekanisme penyimpanan data dan 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.

1.2 Rumusan masalah

Berdasarkan latar belakang di atas, 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 IoT, IoT *cloud platform*, dan *web client*?
3. Bagaimana mekanisme manajemen perangkat, mengirimkan dan mengakses data sensor pada IoT *cloud platform* yang dibangun?
4. Bagaimana performa dari IoT *cloud platform* yang dibangun?

1.3 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 IoT dan *cloud* untuk saling terhubung menggunakan protokol komunikasi HTTP.
3. Mengetahui performa dari IoT *cloud* platform yang dibangun.

1.4 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, 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.

1.5 Batasan masalah

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

1. Perangkat IoT dapat langsung mengirim data ke IoT *cloud platfotm* tanpa melewati *relay* terlebih dahulu.
2. Format data dari atau menuju IoT *cloud platform* menggunakan tipe JSON.
3. Autentikasi dan otorisasi menggunakan JWT (JSON *Web Token*).
4. Penyimpanan token di dalam kukis browser pada aplikasi *web console* tidak menerapkan suatu proses pengamanan tertentu, misalnya enkripsi.
5. Data sensor yang disimpan di *cloud* merupakan data mentah yang tidak dianalisis dan divisualisasikan dalam bentuk tertentu.

1.6 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 pengembangan perangkat lunak 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 pengembangan 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 IoT 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 platform*. 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 yang digunakan dalam tahap pengujian. Pengujian dilakukan berdasarkan skenario uji untuk memastikan kebutuhan telah terpenuhi dan sesuai dengan hasil yang diharapkan.

BAB 7 : Kesimpulan dan Saran

Memuat kesimpulan yang diperoleh dari penelitian ini berdasarkan rumusan masalah sebelumnya. Pada bab ini juga disertakan saran yang dapat digunakan untuk penelitian-penelitian selanjutnya.

BAB 2 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 terkait dengan teknologi yang digunakan, diantaranya: HTTP, *webservice*, sistem basis data MongoDB, JSON *Web Token*, serta dasar teori lainnya yang mendukung.

2.1 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 di atas lingkungan *sandbox* Fiware lab. Fiware lab merupakan *platform* yang berjalan di atas 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 dikirimkannya data sensor dari suatu perangkat IoT secara terus-menerus setiap harinya, sehingga dapat menekan penggunaan sumber daya penyimpanan di *cloud* untuk perangkat IoT yang diinginkan. Fitur lain yang tidak ditemukan atau patut diduga tidak tersedia yaitu fitur visibilitas perangkat IoT yang memungkinkan pengguna dapat mengakses data sensor dari perangkat IoT milik pengguna lainnya. Fitur tersebut sangat berguna jika beberapa pengguna sistem ingin saling berbagi data sensor dari perangkat yang mereka miliki.

2.2 Internet of Things

Menurut Guoqiang et. al. (2013), *Internet of Things* (IoT) merujuk pada suatu jaringan yang menghubungkan perangkat fisik di berbagai jaringan menggunakan berbagai protokol berbeda. IoT bertujuan untuk memperluas manfaat dari konektivitas internet dengan menjadikan benda-benda di sekitar kita dapat terhubung ke internet, sehingga dapat dikontrol dan diakses dari jarak jauh. IoT membuat perangkat di dunia fisik dapat diidentifikasi secara unik, mengumpulkan dan mengirimkan data, serta mengakses suatu layanan yang relevan melalui internet. IoT dapat diaplikasikan ke dalam berbagai aktivitas di segala bidang sehingga pertukaran informasi melalui internet menjadi lebih mudah. 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)

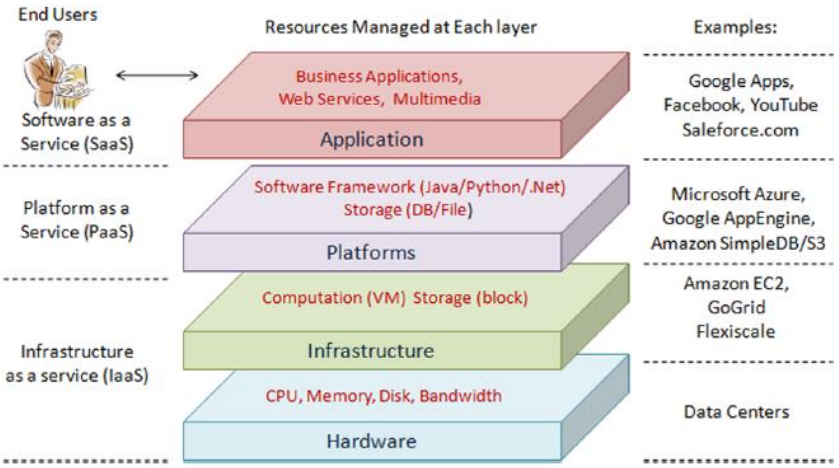
Komponen *identification* memungkinkan *things* dapat diidentifikasi berdasarkan *id* objek dan pengalamatannya di dalam jaringan komunikasi. Komponen *sensing* bertugas mengumpulkan data dari objek IoT *sensor* dan mengirimkannya kembali ke *warehouse*, *database*, atau *cloud*. Data objek IoT *sensor* 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 didistribusikan secara luas dan memiliki keterbatasan dalam hal kapasitas penyimpanan dan kemampuan komputasi. Hal tersebut menciptakan kekhawatiran dalam hal performa, 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 kemampuan komputasi yang lebih baik. Salah satunya adalah sistem komputasi berbasis *cloud*. Integrasi antara IoT dan *cloud computing* menciptakan paradigma teknologi baru yang disebut CloudIoT untuk internet dimasa depan (Botta, et. al., 2016).

2.3 Cloud

Menurut Zhang, et., all (2010) *cloud computing* merupakan model komputasi baru di mana sumber daya 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 di atasnya. Sedangkan lapisan *application* merupakan hierarki tertinggi untuk aplikasi *cloud*, misalnya multimedia dan *web service*.



Gambar 2.2 Arsitektur *Cloud*

Sumber: Zhang, et., all (2010)

2.4 HTTP

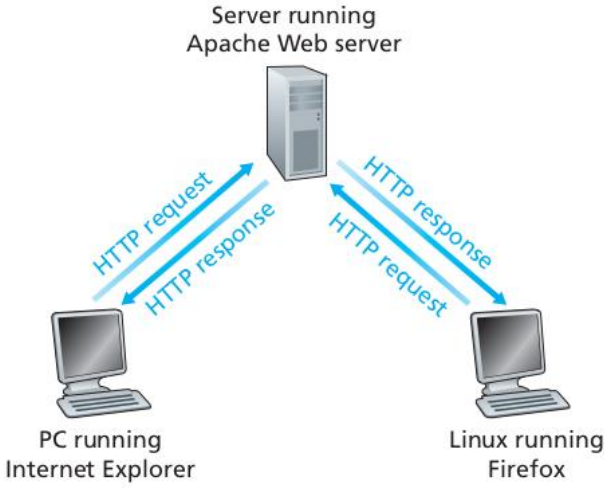
Hypertext Transfer Protocol (HTTP) merupakan sebuah protokol jaringan di layer aplikasi untuk mendistribusikan informasi dengan banyak tipe data secara kolaboratif (IETF, 1999). HTTP dikembangkan untuk proyek *World Wide Web* (WWW) oleh Tim Berners-Lee beserta koleganya pada tahun 1990. WWW menyediakan mekanisme untuk mengambil sumber informasi yang saling terhubung dengan tautan menggunakan *Uniform Resource Identifier* (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). Program *client* dapat berupa *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).

2.4.1 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* tidak akan mengirimkan sebuah informasi jika *client* tidak melakukan permintaan terlebih dahulu. Mekansime kerja dari HTTP dapat dilihat pada Gambar 2.3.

HTTP berjalan di atas 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).



Gambar 2.3 Mekanisme HTTP request-response

Sumber: Kurose dan Ross (2013)

2.4.2 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)

2.4.3 HTTP Header

Menurut Richardson, Leonard, et al (2013), HTTP *header* merupakan serangkaian bit dari *metadata* yang mendeskripsikan semantik protokol dari pesan HTTP *request* atau pesan HTTP *response.* HTTP *header* 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*. HTTP *header* dapat dikelompokkan ke dalam lima kelompok utama berdasarkan fungsinya (Gourley, et. al., 2002) :

a. *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
   ```

b. *Request headers*

   Request headers secara spesifik hanya digunakan pada pesan *request* yang dikirimkan *client*. *Header* ini berisi beberapa informasi tambahan kepada *server* mengenai tipe data 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: */*
```

c. *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
```

d. *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
```

e. *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:

a. *Allow*

Merupakan *header* yang termasuk ke dalam kelompok *response headers*. *Header* ini digunakan untuk memberi tahu *client* mengenai metode 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
```

b. *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
```

c. *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
```

d. *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 autentikasi kepada *client*.

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

2.4.4 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):

a. 1xx*: Informational*

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

b. 2xx: *Successful*

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

c. 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.

d. 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: autentikasi, format dari representasi sumber daya, waktu dari *request*, atau kesalahan dari HTTP *request* dari *client* itu sendiri. Pada akhirnya, *client* harus memperbaiki sesuatu untuk membuat *request* selanjutnya berhasil dilakukan.

e. 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

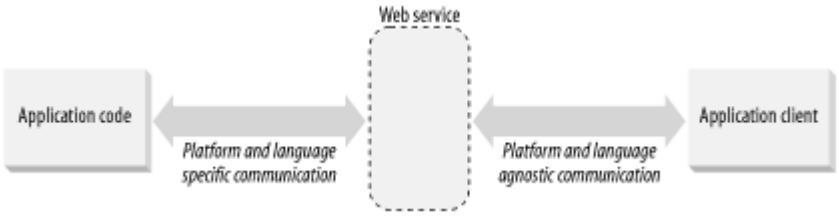
Kode status	Frase Sebab	Arti
200	OK	*Request* berhasil dilakukan.
201	Created	*Server* berhasil membuat objek baru atas permintaan *client* (mis., PUT).
204	No Content	Pesan *response* yang hanya berisi *header* dan kode status tanpa ada badan pesan. Biasanya digunakan untuk memperbarui *browser* tanpa 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)

2.5 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 basis data, 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.



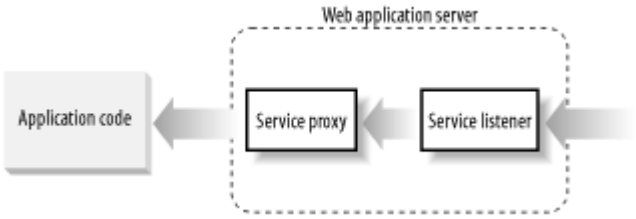
Gambar 2.4 Web Service menyediakan lapisan abstraksi antara aplikasi dan kode

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 mengerti satu 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 untuk menangani *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 untuk digunakan oleh aplikasi lain, sedangkan aplikasi web biasa dibuat untuk digunakan langsung oleh pengguna.

2.5.1 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 data, 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*, di mana setiap *resource* dipetakan ke dalam URI (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` disediakan 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 menerima permintaan 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*

```
PUT /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` disediakan 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
  ```

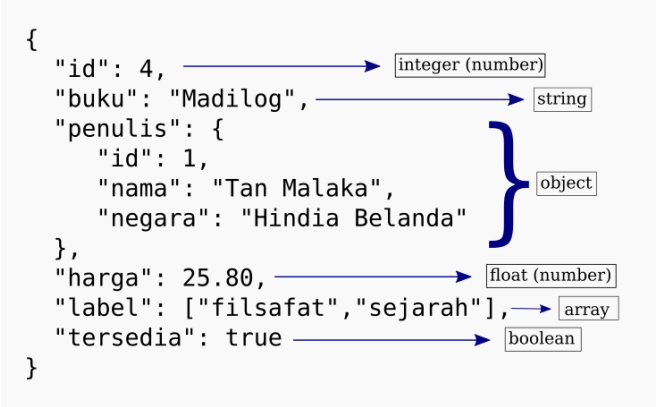
- Menghapus *resource* mahasiswa Budi:

  HTTP *Request*

  ```
  DELETE /mahasiswa/1/ HTTP/1.1
  Host: api.example.com
  ```

2.6 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

2.7 MongoDB

MongoDB merupakan sistem basis data non-relasional berorientasi dokumen yang *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 di awal, hal ini menjadikan kunci dan nilai dari dokumen dapat berbeda dalam jenis dan ukuran. Hal tersebut membuat penambahan dan pengurangan *field* yang dibutuhkan di kemudian 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).

2.7.1 Dokumen

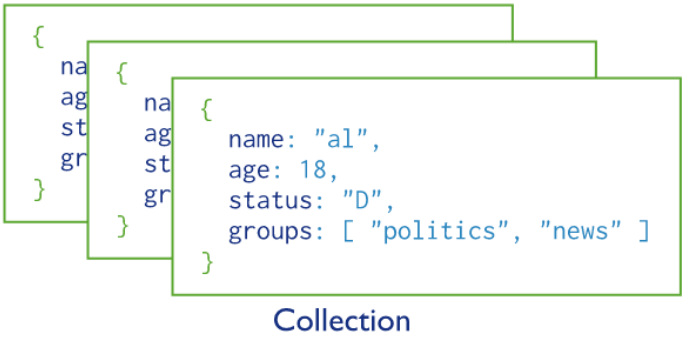
Dokumen adalah model data untuk MongoDB yang kurang lebih setara dengan satu baris dalam sistem basis data relasional (Kristina, 2013). Dokumen merupakan struktur data yang terdiri dari pasangan *field* dan nilai, menjadikannya identik dengan JSON. Nilai dari *field* dapat berisi dokumen lain, *array*, 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)

2.7.2 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)

2.7.3 Tipe Data

MongoDB menyimpan data dalam dokumen BSON yang dapat dikatakan secara konsep mirip dengan JSON. Di mana 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"}}
  ```

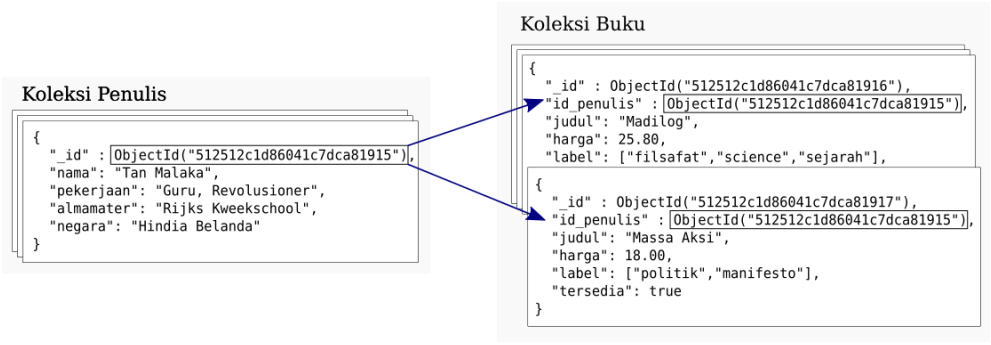
- *object id*

  Object Id merupakan identifikasi sebesar 12-*byte* untuk dokumen yang di-*generate* secara otomatis ketika dokumen baru dibuat. Object Id di-*generate* berdasarkan CPU ID dan *timestamp*. 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()}
```

2.7.4 *Reference*

MongoDB menyediakan fasilitas *reference* untuk normalisasi data. Normalisasi membagi data ke dalam beberapa koleksi untuk menekan redudansi, yaitu keadaan di mana 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

2.8 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.

2.8.1 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 di atas dikodekan menggunakan skema *base64url*, kemudian juga telah dibubuhkan tanda tangan digital akan menghasilkan JWT secara utuh seperti Gambar 2.10 dibawah ini.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvaG4
gRG9lIiwiaXNTb2NpYWwiOnRydWV9.
4pcPyMD09olPSyXnrXCjTwXyr4BsezdI1AVTmud2fU4
```

Gambar 2.10 Contoh JSON *Web Token*

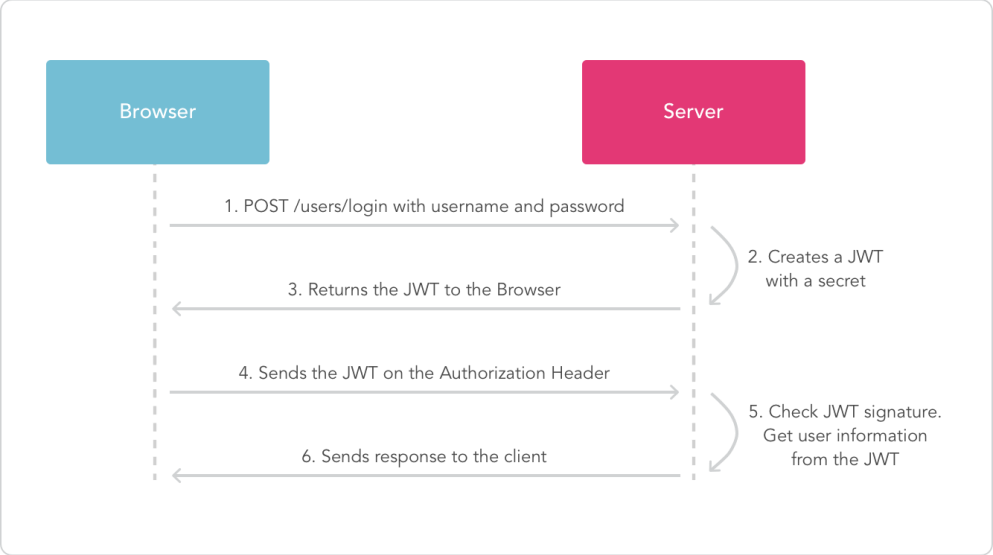
Sumber: jwt.io

2.8.2 Autentikasi dan Otorisasi

Pada proses autentikasi, *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 sumber daya yang dilindungi. JSON *Web Token* menyimpan *credentials* dan data *user* secara lokal. Itu berarti *server* tidak perlu menyimpan informasi tersebut karena sudah terwakili oleh JWT. Sedangkan pada pendekatan autentikasi 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 sumber daya yang dilindungi, *user* harus mengirimkan JWT untuk keperluan autentikasi. 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 sumber daya yang dilindungi. Jika tidak, *server*

akan menolak permintaan tersebut sehingga operasi dan sumber daya dapat terlindungi dari *user* yang tidak memiliki wewenang. Karena JWT bersifat mandiri, *credentials* dan data user yang diperlukan ada di dalam token, sehingga mengurangi kebutuhan untuk mengeksekusi kueri basis data berkali-kali.

2.9 Parameter Pengujian

- Skalabilitas: kapasitas sistem dalam mengakomodasi peningkatan jumlah pengguna tanpa ada penurunan metrik performa (Litoiu, 2002). Uji skalabilitas dapat digunakan untuk mengetahui jumlah pengguna yang dapat ditangani oleh sistem dalam satu detik.
- *Latency*: waktu yang dibutuhkan pada saat *client* memanggil layanan web tertentu sampai *response* diterima (Litoiu, 2002).
- Error rate: persentase jumlah *request* yang tidak tertangani oleh sistem dari total *request* yang dikirimkan.

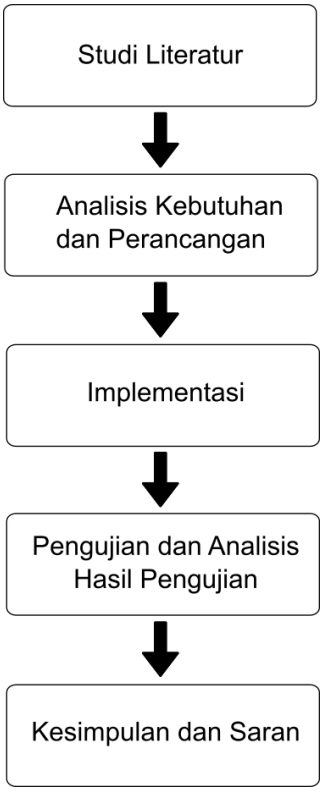
BAB 3 METODOLOGI

3.1 Jenis Penelitian

Penelitian yang dilakukan penulis termasuk dalam jenis penelitian pengembangan perangkat lunak. 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.

3.2 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

3.2.1 Studi Literatur

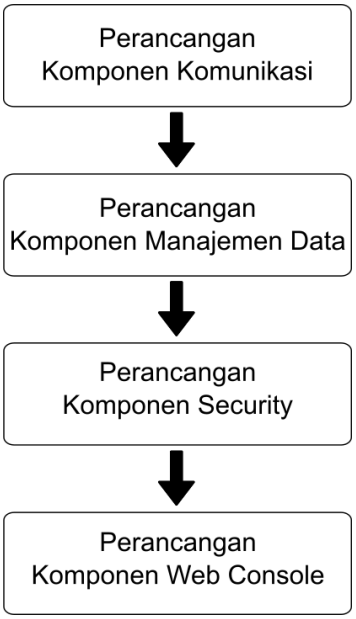
Studi literatur merupakan kegiatan yang dilakukan untuk mempelajari dan memahami teori dan referensi yang digunakan pada penelitian. Teori dan

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.

3.2.2 Analisis Kebutuhan dan Perancangan

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 sumber daya yang tidak diperlukan. 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 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 3.2 dibawah ini:



Gambar 3.2 Tahap perancangan

Adapun detail dari tahap perancangan di atas adalah sebagai berikut:

1. Pada perancangan komponen komunikasi akan digambarkan bagaimana alur komunikasi antar perangkat IoT dan *web 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 autentikasi yang berbeda antara perangkat IoT dan *web client* menggunakan token akses JWT. Dijelaskan juga tentang bagaimana alur untuk mendapatkan token akses JWT, registrasi pengguna, dan menerima data sensor dari perangkat IoT.
4. Perancangan komponen *web console* menjelaskan rancangan antarmuka pengguna untuk memanajemen perangkat dan melihat data sensor.

3.2.3 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*.
3. Implementasi komponen *security*, meliputi beberapa implementasi sebagai berikut:
    - Implementasi autentikasi dan otorisasi, berisi kode sumber untuk membuat token akses JWT, registrasi pengguna dan menerima data sensor dari perangkat IoT.
    - Implementasi manajemen perangkat, berisi kode sumber untuk menambah, mengubah, dan menghapus perangkat IoT.

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

3.2.4 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 tiga jenis pengujian yaitu pengujian fungsional, pengujian performa dan pengujian *overhead*.

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

Pengujian performa 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 IoT mengirimkan data sensor. Parameter uji yang digunakan adalah skalabilitas, *latency*, dan *error rate*. Pengujian dilakukan dengan mengirimkan HTTP *request* sejumlah 50, 100, dan 150 secara bersamaan.

2. Performa fitur mengakses data sensor

   Peneliti mengukur berapa banyak HTTP *request* yang dapat ditangani sistem ketika *web client* mengakses data sensor. Pengujian ini secara spesifik menguji tiga kriteria dalam mengakses data sensor, yaitu kriteria berdasarkan pengguna, perangkat IoT dan sensor. Parameter uji yang digunakan adalah skalabilitas, *latency*, dan *error rate*. Pengujian dilakukan dengan mengirimkan HTTP *request* sejumlah 50, 100, dan 150 secara bersamaan.

Pengujian *overhead* dilakukan untuk mengetahui kerugian yang ditimbulkan dari proses autentikasi dan otorisasi menggunakan JSON Web Token. Kerugian

yang dimaksud adalah besaran *Bytes* tambahan pada HTTP *header* yang diperlukan untuk mengirimkan token. Tahap pengujian terdiri atas dua skenario:

1. *Overhead* pada HTTP GET

   Peneliti membandingkan ukuran HTTP *header* pada metode HTTP GET dengan dan tanpa menggunakan token.

2. *Overhead* pada HTTP POST

   Peneliti membandingkan ukuran pesan HTTP pada metode HTTP POST dengan dan tanpa menggunakan token. Pada skenario ini dapat diketahui juga mana yang memiliki ukuran yang lebih kecil diantara token di HTTP *header* dan *payload* data yang dikirimkan.

Berdasarkan pengujian di atas, hasil kemudian diolah dan dianalisis untuk mengetahui apakah telah sesuai dengan tahap kebutuhan dan perancangan sebelumnya, memiliki autentikasi dan otorisasi yang sesuai, dan performa yang diharapkan.

3.2.5 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.

BAB 4 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*.

4.1 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 IoT dan *web client*. Perangkat IoT merupakan *microcontroller* atau komputer papan tunggal dengan modul sensor tertentu yang mengirimkan data. Sedangkan *web client* merupakan perangkat lunak tertentu (mis., *web browser*) yang digunakan oleh pengguna untuk mengakses data sensor dan memanajemen perangkat

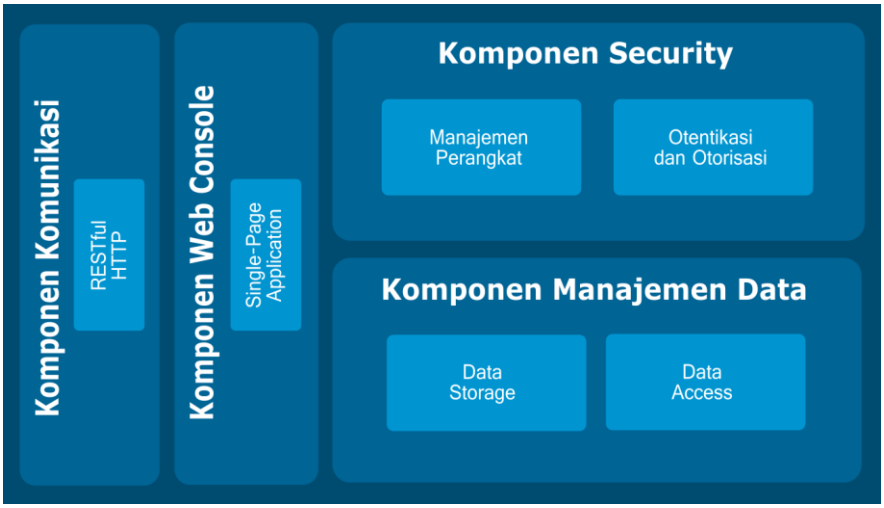
Untuk dapat menggunakan perangkat lunak IoT *cloud platform*, pengguna diharuskan melakukan registrasi terlebih dahulu. Setiap pengguna nantinya dapat memiliki lebih dari satu perangkat IoT dan setiap perangkat dapat memiliki lebih dari satu sensor. Pengguna dapat mengelola perangkat melalui operasi 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 dapat dibatasi dalam jumlah tertentu untuk mengirimkan data sensor per-harinya. Sedangkan fitur visibilitas perangkat memberikan *policy* dalam mengakses data perangkat dengan mengatur label visibilitas menjadi "*public*" atau "*private*". Pengguna dapat melihat data sensor dari perangkat miliknya dan dari perangkat pengguna lain yang memiliki label visibilitas "*public*"".

Perangkat IoT yang didaftarkan memiliki identifikasi unik berupa label dan *secretkey*. Keduanya digunakan untuk mendapatkan token akses yang digunakan dalam proses autentikasi dan otorisasi mengirimkan data sensor ke perangkat lunak IoT *cloud platform*. Dalam mengirimkan data sensor, akan disediakan suatu format pengiriman tertentu untuk mengabungkan data sensor dari beberapa sensor yang dimiliki perangkat 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 diatur ulang secara otomatis setiap pukul 24:00.

IoT *cloud platform* yang dikembangkan menerapkan fitur sekuritas seperti autentikasi dan otorisasi menggunakan token akses JWT. Dua entitas yang berinteraksi dengan IoT *cloud platform* yakni perangkat IoT dan *web client* akan memiliki token akses yang berbeda. Token perangkat IoT digunakan untuk pengiriman data sensor sedangkan token *web client* digunakan untuk mengakses data sensor dan memanajemen perangkat. Penggunaan token harus sesuai, misalnya token *web client* tidak dapat digunakan untuk mengirimkan data sensor begitu juga sebaliknya token perangkat IoT 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 IoT dan *web client* melalui protokol HTTP. Komponen manajemen data menangani mekanisme yang berkaitan dengan *data storage* dan menyediakan mekanisme untuk mengakses data sensor melalui RESTful *web service*. Komponen *security* menyediakan mekanisme untuk autentikasi 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

4.2 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.

4.2.1 Kebutuhan Fungsional

Kebutuhan fungsional merupakan layanan, fitur, atau proses apa saja yang harus dipenuhi untuk penguna sistem. Kebutuhan fungsional dari sistem, dibagi berdasarkan penyelesaian dari masalah yang diangkat dijelaskan pada Tabel 4.1 :

Tabel 4.1 Kebutuhan fungsional sistem

Nomor	Kebutuhan
Masalah jaringan komunikasi	
1	Sistem dapat menyediakan layanan bagi perangkat IoT untuk mengirimkan data sensor melalui protokol HTTP
2	Sistem dapat menyediakan layanan bagi pengguna untuk mengakses data sensor melalui protokol HTTP
Masalah keamanan	
3	Sistem dapat menyediakan layanan bagi perangkat IoT dan *web client* untuk mendapatkan token akses JWT
4	Sistem dapat menyediakan layanan untuk melakukan registrasi pengguna
5	Sistem dapat menerapkan autentikasi menggunakan token akses JWT
6	Sistem dapat menerapkan otorisasi menggunakan token akses JWT

Masalah manajemen perangkat	
7	Sistem dapat menyediakan layanan bagi pengguna untuk memanajemen perangkat IoT

4.2.2 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)	6 GB DDR 3L
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 *web console*.	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

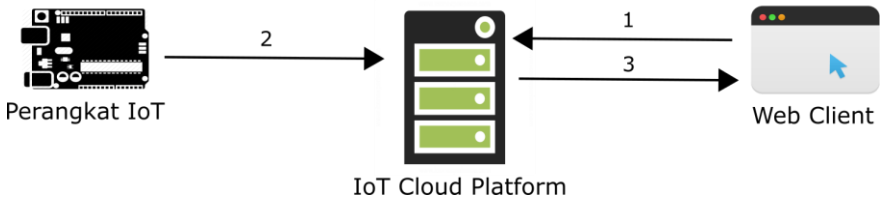
Visual Studio Code	Editor pemrograman untuk memprogram aplikasi *web console*.	Versi 11
Oracle VirtualBox	Perangkat lunak untuk menjalankan mesin virtual.	Versi 5.1
Apache Jmeter	Perangkat lunak pengujian untuk menguji performa sistem yang dibangun	Versi 3.2
Wireshark	Perangkat lunak yang digunakan untuk menangkap dan melihat detail ukuran pesan HTTP yang dikirimkan lewat jaringan.	Versi 2.4

4.3 Perancangan

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

4.3.1 Perancangan Komponen Komunikasi

Perancangan komponen komunikasi menggambarkan alur komunikasi antara tiga entitas yaitu perangkat IoT, perangkat lunak IoT *cloud platform*, dan *web client*. Alur komunikasi antar entitas dibagi menjadi dua yakni pengiriman data dari perangkat IoT ke IoT *cloud platform* dan *web 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 di atas adalah:

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

4.3.2 Perancangan Komponen Manajemen Data

Perancangan komponen manajemen data dibagi menjadi dua yakni perancangan data model dan perancangan data *access*. Perancangan data model

menggambarkan skema basis data yang digunakan. Sedangkan perancangan data access menggambarkan alur komunikasi dalam mengakses data dari data model.

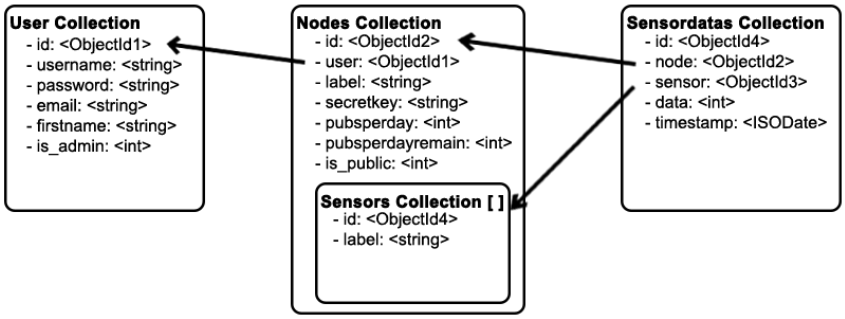
4.3.2.1 Perancangan Data Model

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 IoT, dokumen *sensor* menyimpan informasi mengenai sensor yang dimiliki suatu perangkat IoT, dan dokumen *Sensordatas* menyimpan data yang dikirimkan oleh perangkat IoT 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:

1. Dokumen *Users* memiliki hubungan satu-ke-banyak ke dokumen *Nodes* karena satu pengguna dapat memiliki beberapa perangkat IoT.
2. Dokumen *Nodes* memiliki hubungan satu-ke-banyak ke dokumen *Sensors* karena dalam satu perangkat IoT dapat memiliki beberapa sensor.
3. 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.

4. 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).

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 IoT.

secretkey	String	Karakter rahasia untuk keperluan autentikasi.
pubsperday	Int	Jumlah pengiriman yang dapat dilakukan perangkat IoT 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 IoT.

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 IoT, dan berdasarkan sensor. Pada kriteria berdasarkan perangkat IoT dan kriteria berdasarkan sensor memungkinkan untuk mengakses data sensor milik pengguna lain selama perangkat IoT 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 IoT 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 IoT 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 IoT, dan berdasarkan sensor.

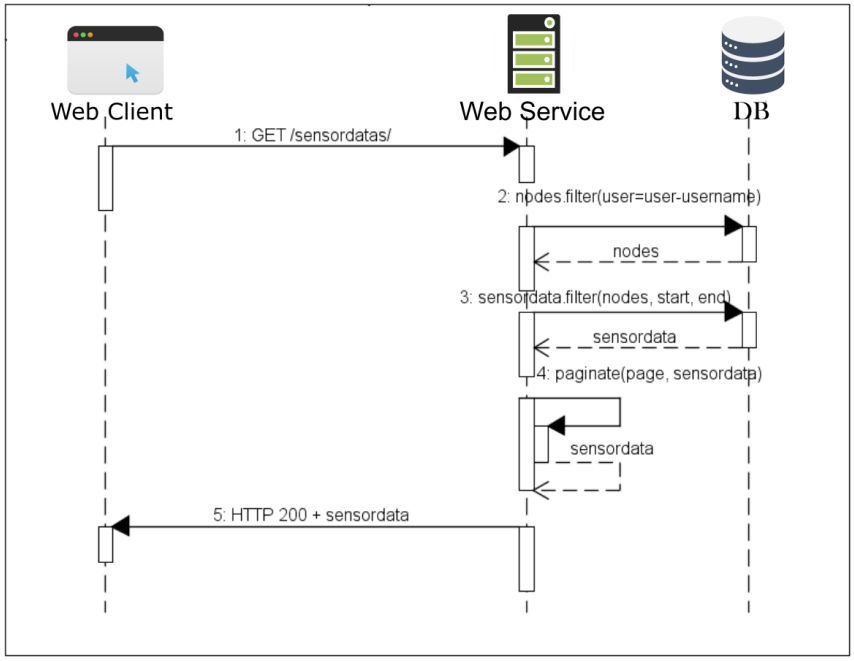
a. Berdasarkan Pengguna

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 IoT 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) *Web client* mengirimkan HTTP *request* dengan metode GET.
2) *Web service* memanggil fungsi `nodes.filter(user=user-username)` untuk mendapatkan semua perangkat IoT 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 IoT yang dimiliki pengguna. Parameter fungsi `nodes` diambil dari proses nomor 2, sedangkan parameter `start` dan `end` didapatkan dari parameter *query string* yang dikirimkan bersamaan dengan *request* HTTP *request* dengan metode GET. Jika kedua parameter *query string* tersebut tidak dikirimkan maka parameter fungsi `start` dan `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 `page`

didapatkan dari parameter *query string* yang dikirimkan bersamaan dengan HTTP *request* dengan metode 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.

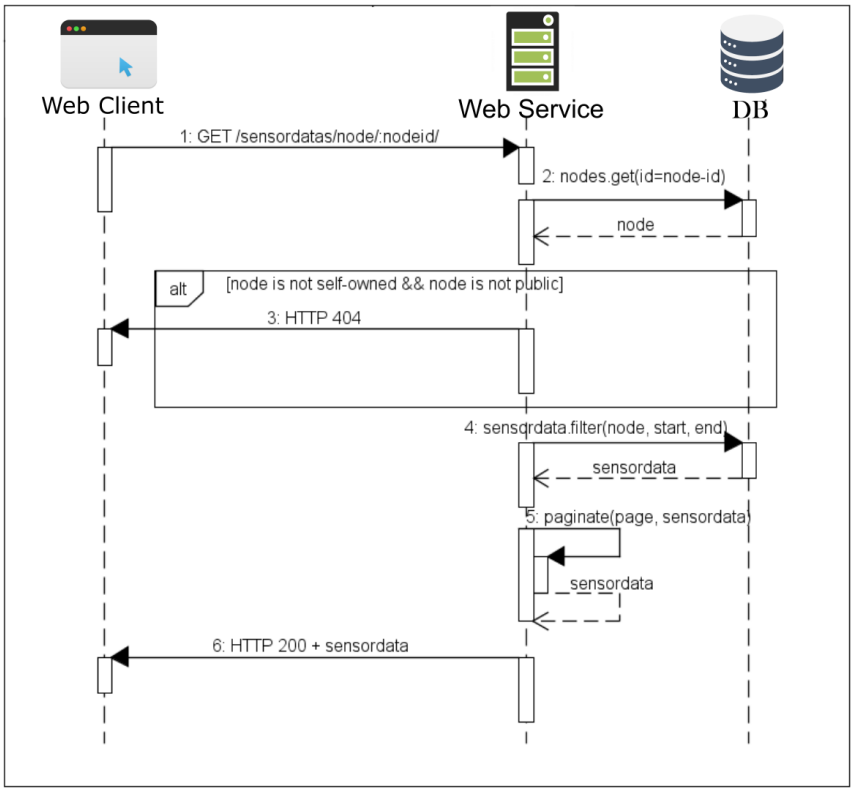
5) *Web service* mengirimkan HTTP status 200 beserta data sensor sesuai dengan kriteria dan halaman yang diminta.

b. Berdasarkan Perangkat IoT

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 IoT digambarkan pada Gambar 4.5.

Tabel 4.9 Akses data sensor berdasarkan perangkat IoT

***Request* GET /sensordatas /node/:nodeid/**		
Deskripsi	**Parameter**	**Keterangan**
Menampilkan data sensor yang disortir berdasarkan perangkat IoT 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 IoT

Penjelasan dari Gambar 4.5 adalah sebagai berikut:

1) *Web client* mengirimkan HTTP *request* dengan metode GET.
2) *Web service* memanggil fungsi `nodes(node-id)` untuk mendapatkan data perangkat IoT berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel `node`.
3) Jika perangkat IoT yang diakses bukan milik pengguna dan perangkat IoT 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 IoT tersebut. Parameter fungsi `node` diambil dari proses nomor 2, sedangkan parameter `start` dan `end` didapatkan dari parameter *query string* yang dikirimkan bersamaan dengan HTTP *request* dengan metode GET. Jika kedua parameter *query string* tersebut tidak dikirimkan maka parameter fungsi `start` dan `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 `page`

didapatkan dari parameter *query string* yang dikirimkan bersamaan dengan HTTP *request* dengan metode 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.

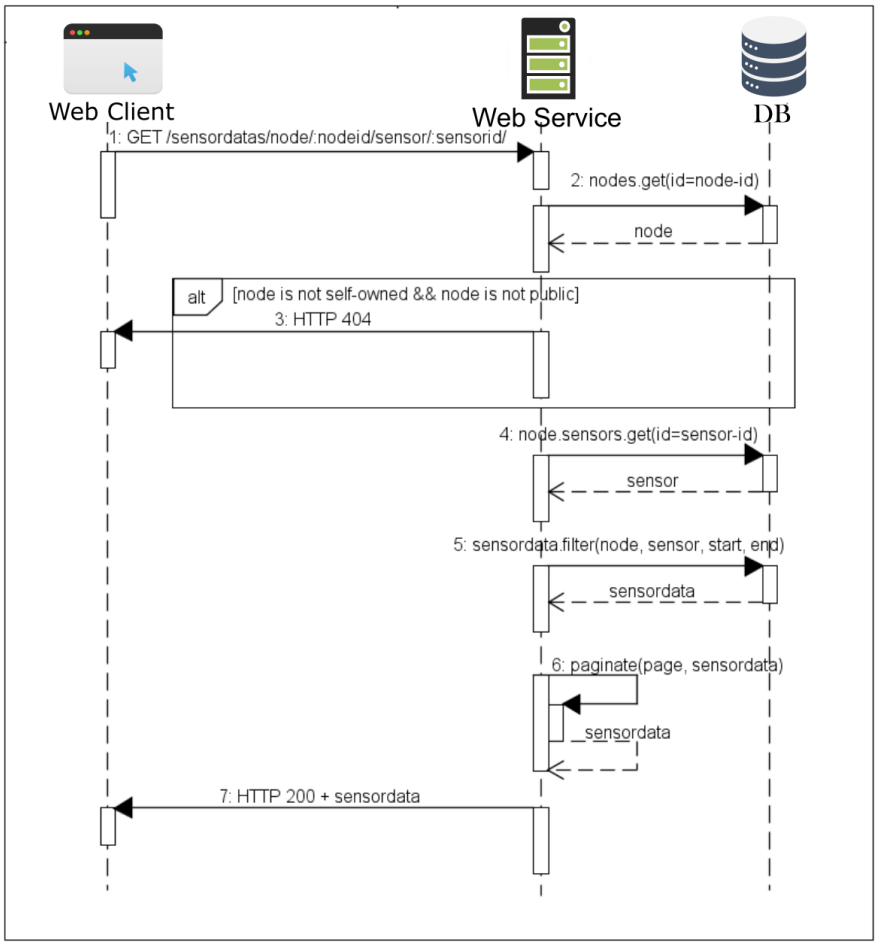
6) *Web service* mengirimkan HTTP status 200 beserta data sensor sesuai dengan kriteria dan halaman yang diminta.

c. 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/:nodeid/sensor/:sensorid/**		
Deskripsi	**Parameter**	**Keterangan**
Menampilkan data sensor yang disortir berdasarkan perangkat IoT 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) *Web client* mengirimkan HTTP *request* dengan metode GET.
2) *Web service* memanggil fungsi `nodes(node-id)` untuk mendapatkan data perangkat IoT berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel `node`.
3) Jika perangkat IoT yang diakses bukan milik pengguna dan perangkat IoT 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 IoT tersebut. Parameter fungsi `node` diambil dari proses nomor 2, sedangkan parameter `start` dan `end` didapatkan dari parameter *query string* yang dikirimkan bersamaan dengan HTTP *request* dengan metode GET. Jika kedua parameter *query string* tersebut tidak dikirimkan maka parameter fungsi `start` dan `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 `page` didapatkan dari parameter *query string* yang dikirimkan bersamaan dengan HTTP *request* dengan metode 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.

4.3.3 Perancangan Komponen *Security*

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

4.3.3.1 Perancangan Autentikasi dan Otorisasi

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

a. Mendapatkan token akses JWT

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

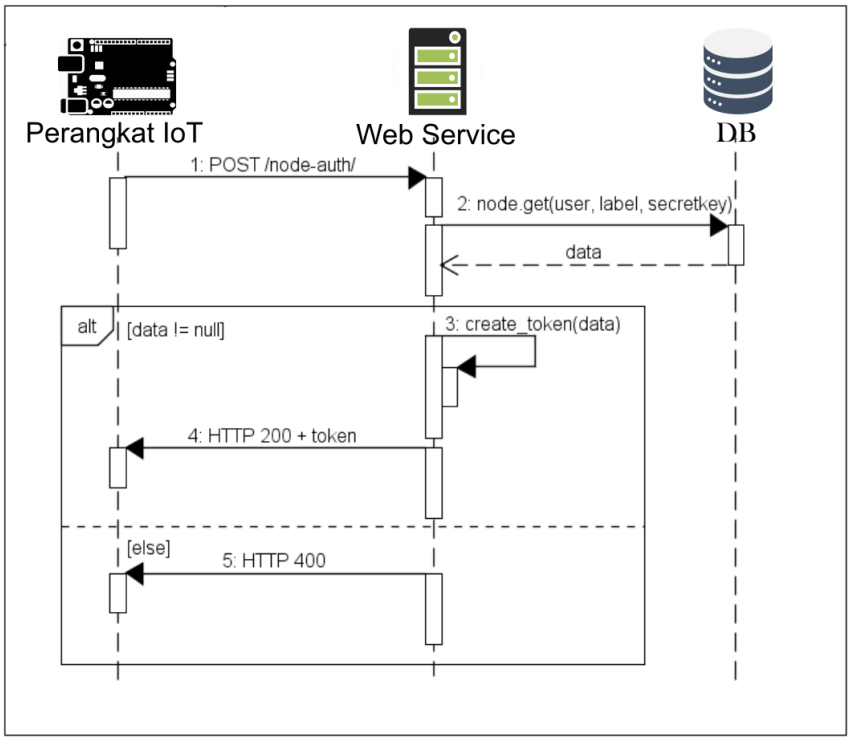
- Perangkat IoT

  Untuk mendapatkan token akses JWT perangkat IoT perlu mengirimkan HTTP *request* dengan *payload* berisi *user*, *label*, dan *secretkey*. *Payload* tersebut berisi tiga *field* utama pada dokumen Nodes untuk mendapatkan informasi perangkat IoT 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 IoT mendapatkan token akses JWT

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



Gambar 4.7 Alur Perangkat IoT mendapatkan token akses JWT

Penjelasan dari Gambar 4.7 adalah sebagai berikut:

1) Perangkat IoT mengirimkan HTTP *request* dengan metode POST dengan *payload* berisi *user*, *label*, dan *secretkey*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.11.
2) *Web service* memanggil fungsi `nodes.get(user, label, secretkey)` untuk mendapatkan data perangkat IoT. 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 perangkat IoT tersebut dan token akses JWT-nya. Informasi lebih lengkap mengenai informasi *response* dapat dilihat pada Tabel 4.11.
5) Jika variable `data` berisi `null`, *web service* mengirimkan *response* dengan HTTP status 400.

- Akses token JWT untuk *web client*

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

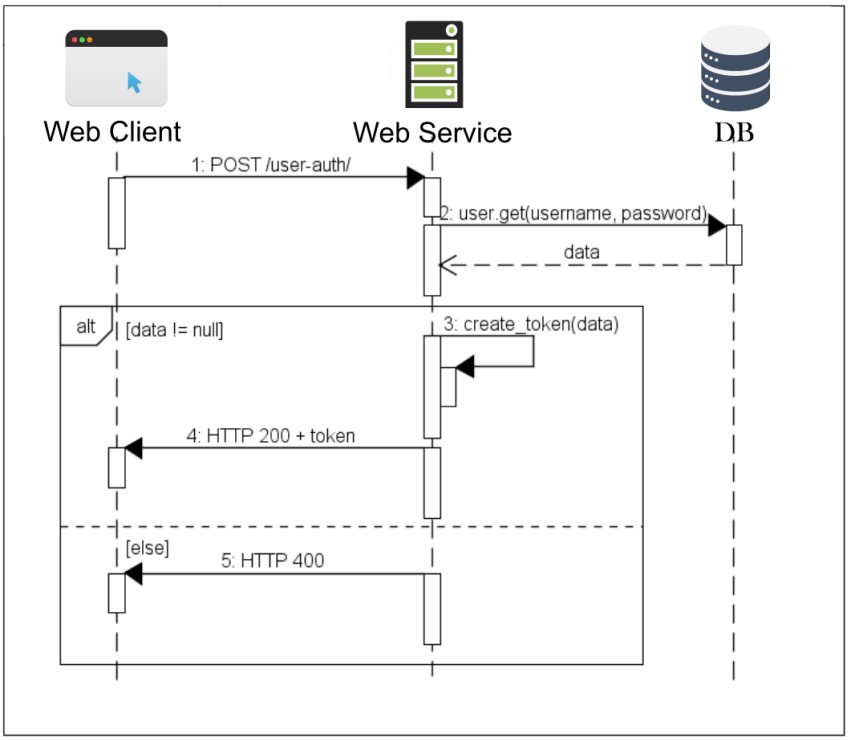
  Untuk mendapatkan token akses, *web client* perlu mengirimkan HTTP *request* dengan *payload* berisi *username*, dan *password*. *Payload* tersebut berisi dua *field* utama pada dokumen Users yang diperlukan untuk mendapatkan informasi pengguna yang dimaksud. Informasi tersebut kemudian di-*generate*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

***Request* POST /user-auth/**		
Deskripsi	***Payload***	**Keterangan**
Men-*generate* token akses JWT untuk *web client*.	- *username* - *password*	- *payload username* dan *password* dibutuhkan. - *payload username* merupakan *username* dari akun pengguna. - *password* merupakan kata sandi dari akun pengguna.

Response		
Nama *field*	**Tipe data**	**Keterangan**
user	*object* dari dokumen Users	Data pengguna.
token	string	Akses token JWT.



Gambar 4.8 Alur *Web client* mendapatkan token akses JWT

Penjelasan dari Gambar 4.8 adalah sebagai berikut:

1) *Web client* mengirimkan HTTP *request* dengan metode 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 pernggguna. 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 4.12.

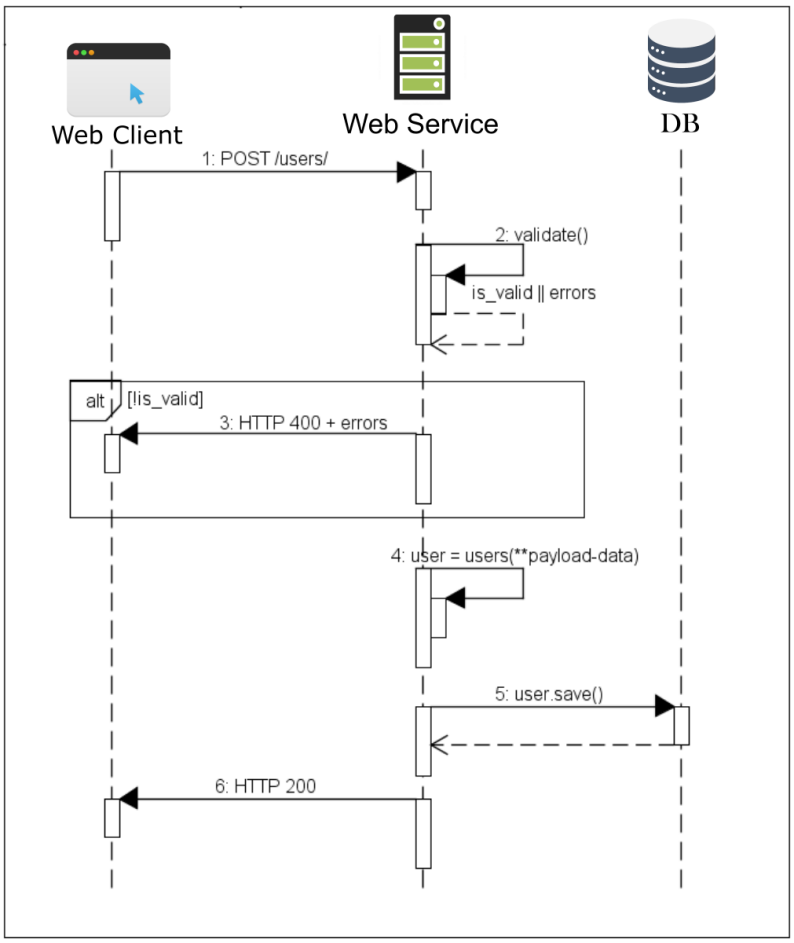
5) Jika variable `data` berisi `null`, w*eb service* mengirimkan *response* dengan HTTP status 400.

b. 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:

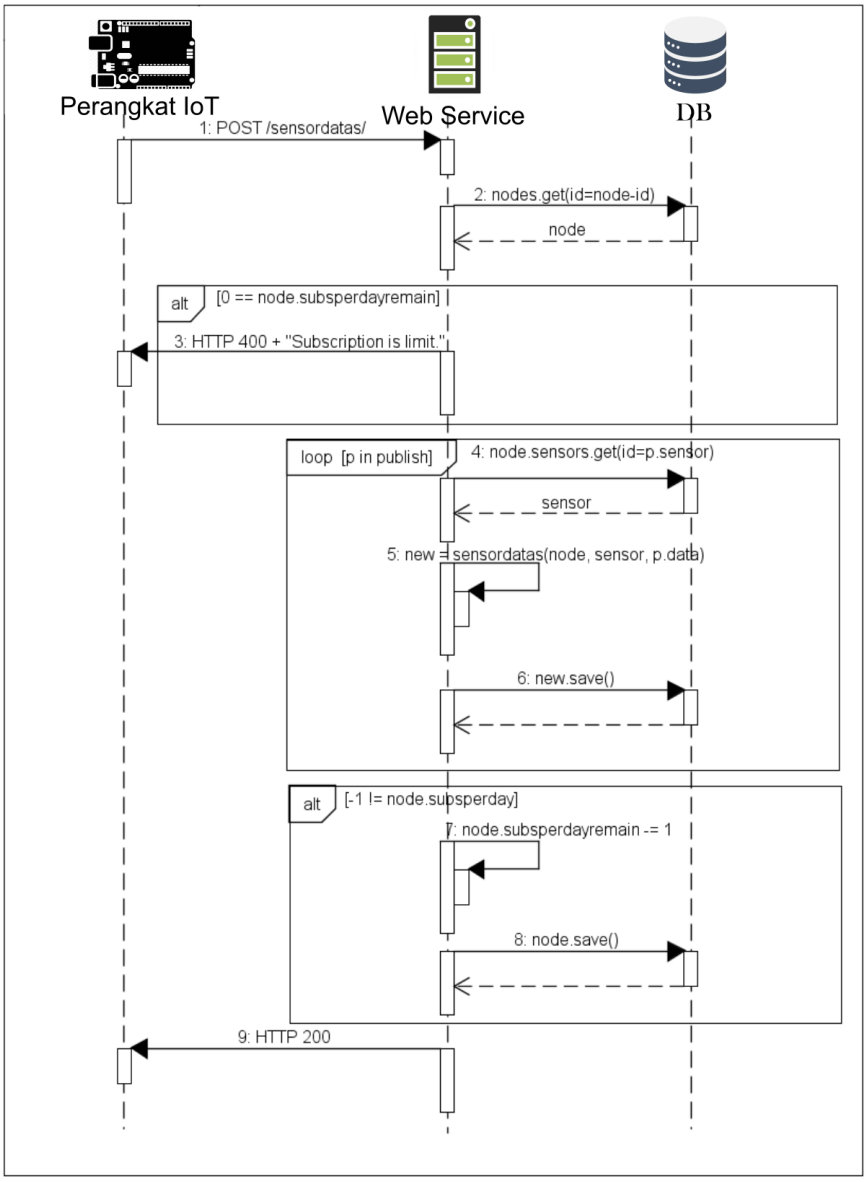
1) *Web client* mengirimkan HTTP *request* dengan metode POST dengan *payload* berisi *username*, *first_name*, *last_name*, *email*, dan *password*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.13.
2) *Web service* memanggil fungsi `validate()` untuk memastikan *payload* yang 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.

c. Menerima data sensor dari perangkat IoT

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 IoT digambarkan pada Gambar 4.10.

Tabel 4.14 Menerima data sensor dari perangkat IoT

***Request* POST /sensordatas/**		
Deskripsi	***Payload***	**Keterangan**
Mengirimkan data sensor dari perangkat IoT 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 IoT

Penjelasan dari Gambar 4.10 adalah sebagai berikut:

1) *Web client* mengirimkan HTTP *request* dengan metode 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 IoT. Kemudian, DB mengembalikan data yang diminta dalam variabel `node`.

3) Jika `pubsperdayremain` dari perangkat IoT 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`.

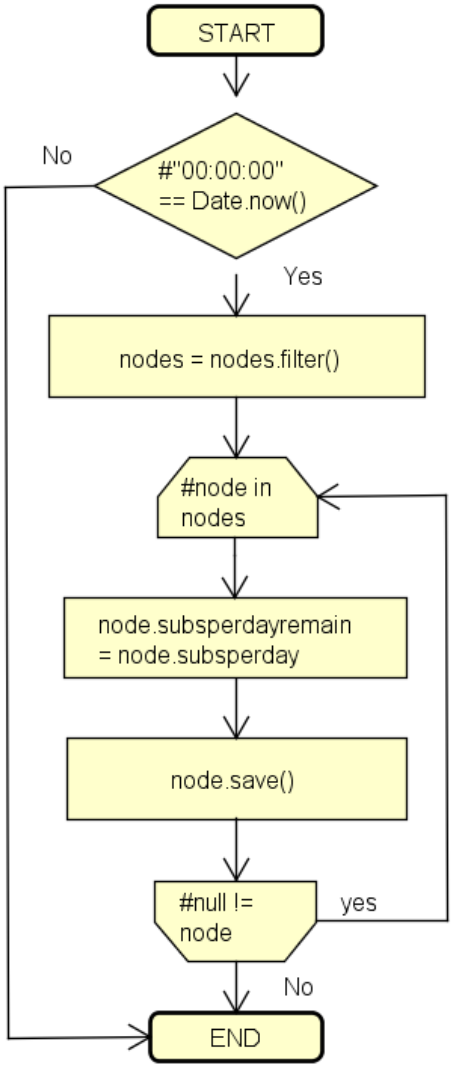
4) *Web service* memanggil fungsi `nodes.sensors.get(id=p.sensor)` untuk mendapatkan data sensor. Kemudian, DB mengembalikan data yang diminta dalam variabel `sensor`.
5) *Web service* memanggil fungsi `sensordatas(node, sensor, p.data)` untuk membuat *object* baru dari dokumen Sensordatas. *Object* baru tersebut disimpan dalam variabel `new`.
6) *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 IoT tidak memiliki pembatasan pengiriman data sensor.

7) Nilai `pubsperdayremain` dari perangkat IoT dikurangi satu.
8) Perubahan data perangkat IoT pada nomor 7 disimpan ke dalam basis data.
9) *Web service* mengirimkan *response* dengan HTTP status 200.

d. Mengatur ulang bilangan *counter* pembatasan pengiriman semua perangkat IoT.

Fitur pembatasan pengiriman data sensor per hari bagi perangkat IoT memerlukan satu bilangan *counter* yang akan berkurang satu setiap kali perangkat mengirimkan data. Bilangan *counter* tersebut harus diatur ulang pada hari berikutnya agar perangkat IoT 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 IoT

4.3.3.2 Perancangan Manajemen Perangkat

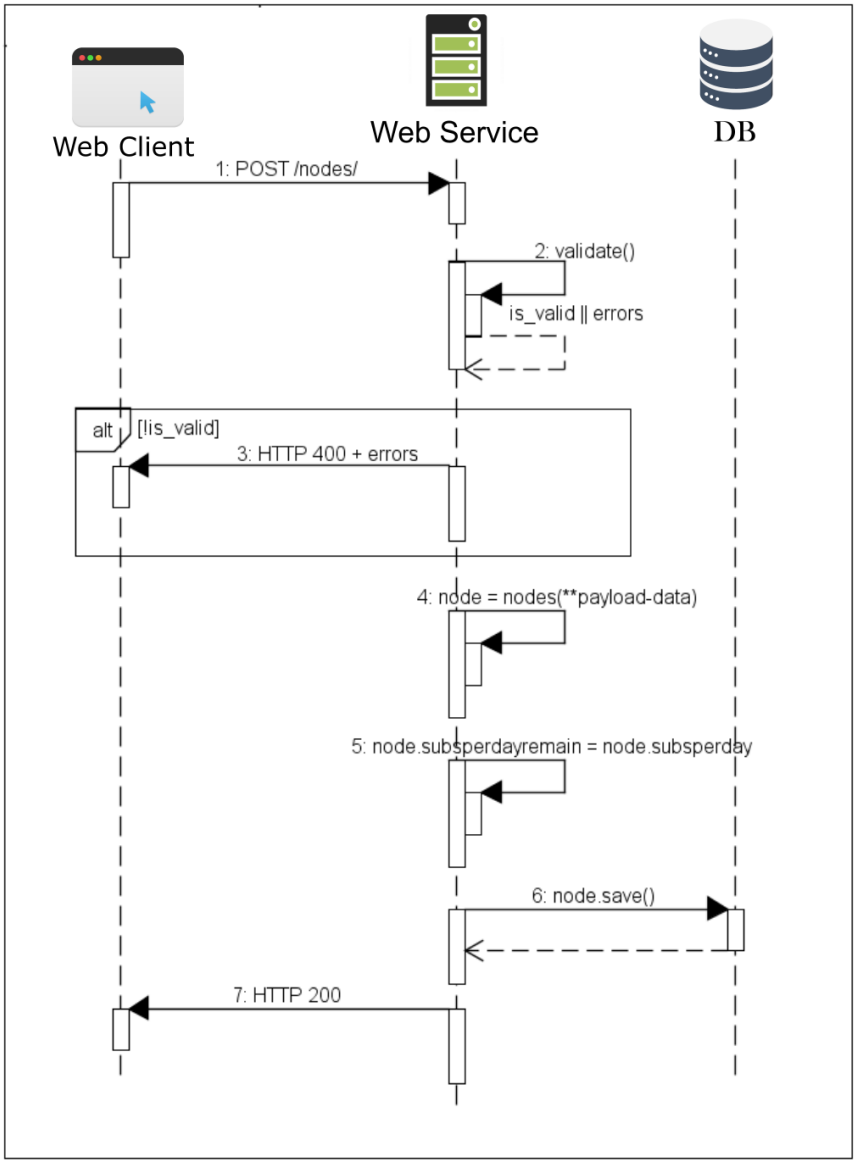
Perancangan manajemen perangkat menjelaskan alur *web client* dalam memanajemen perangkat.

a. Membuat perangkat IoT

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 IoT digambarkan pada Gambar 4.12.

Tabel 4.15 Membuat perangkat IoT

***Request* POST /nodes/**		
Deskripsi	***Payload***	**Keterangan**
Mendaftarkan perangkat IoT 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 IoT bersifat publik, sedangkan nilai 0 berarti bersifat privat.
Response		
Nama *field*	**Tipe data**	**Keterangan**
results	dokumen nodes.	Perangkat IoT yang dibuat.



Gambar 4.12 Alur membuat perangkat IoT

Penjelasan dari Gambar 4.12 adalah sebagai berikut:

1) *Web client* mengirimkan HTTP *request* dengan metode POST dengan *payload* berisi user, *label*, *secretkey*, *subsperday*, dan *is_public*. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.15.
2) *Web service* memanggil fungsi `validate()` untuk memastikan *payload* yang dikirim sesuai dengan yang dibutuhkan. Fungsi ini juga memeriksa keunikan *field* label dari perangkat IoT baru terhadap perangkat IoT lainnya milik pengguna tersebut.
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.

b. Melihat perangkat IoT

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

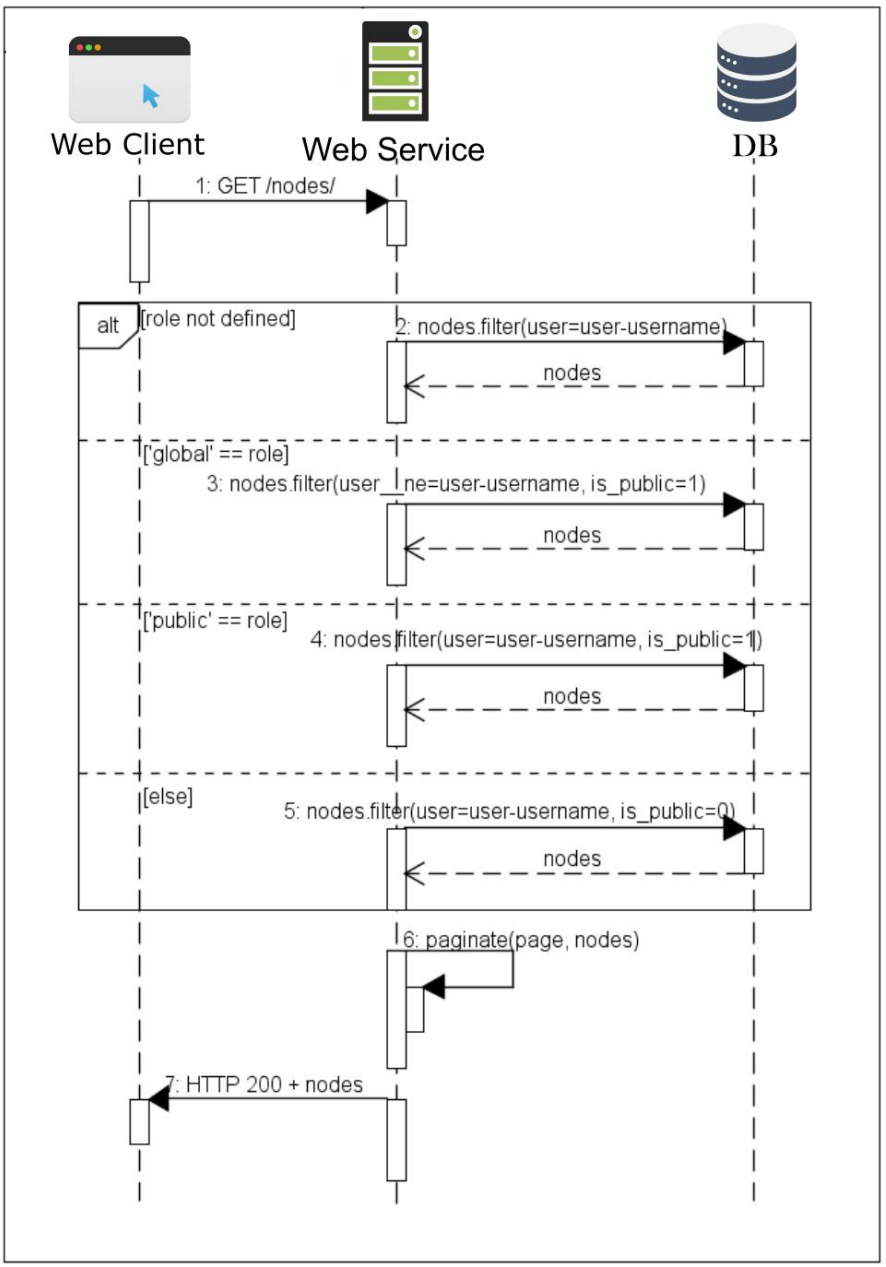
Dalam melihat perangkat IoT digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman. Setiap HTTP *request* akan memberikan HTTP *response* berupa halaman dengan maksimal berisi 10 data di dalamnya. Untuk menggunakan fitur *pagination*, dibutuhkan parameter HTTP GET tambahan yaitu *page* 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 IoT digambarkan pada Gambar 4.13.

Tabel 4.16 Melihat perangkat IoT

***Request* GET /nodes/**		
Deskripsi	**Parameter**	**Keterangan**
Menampilkan perangkat IoT.	- *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 IoT 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 IoT pada halaman saat ini.



Gambar 4.13 Alur melihat perangkat IoT

Penjelasan dari Gambar 4.13 adalah sebagai berikut:

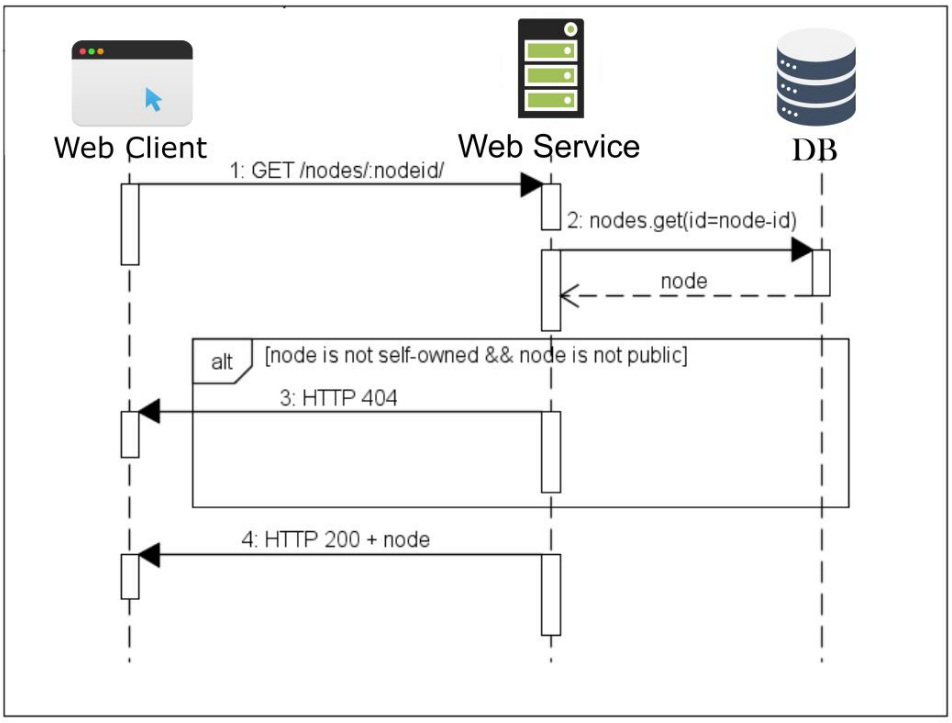
1) *Web client* mengirimkan HTTP *request* dengan metode GET dengan atau tanpa parameter *role*. Informasi lebih lengkap mengenai parameter *role* dapat dilihat pada Tabel 4.16.
2) Jika parameter `role` tidak didefinisikan, *web service* memanggil fungsi `nodes.filter(user=user-username)` untuk mendapatkan semua perangkat IoT milik pengguna. Setelah fungsi `nodes.filter()` dieksekusi, DB mengembalikan data yang diminta dalam variable `nodes`.
3) Jika parameter `role` bernilai "*global*", *web service* akan memanggil fungsi `nodes.filter(user__ne=user-username, is_public=1)` untuk mendapatkan semua perangkat IoT milik pengguna lain yang memiliki label visibilitas publik. Setelah fungsi `nodes.filter()` dieksekusi, DB mengembalikan data yang diminta dalam variable `nodes`.
4) Jika parameter `role` bernilai "*public*", *web service* akan memanggil fungsi `nodes.filter(user=user-username,is_public=1)`untuk mendapatkan semua perangkat IoT 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 mendapatkan semua perangkat IoT 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 IoT sesuai dengan kriteria dan halaman yang diminta.

c. Melihat perangkat IoT 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 IoT digambarkan pada Gambar 4.14.

Tabel 4.17 Melihat perangkat IoT berdasarkan id

***Request* GET /nodes/:nodeid/**		
Deskripsi	**Parameter**	**Keterangan**
Menampilkan perangkat IoT berdasarkan id.	-	-
Response		
Nama *field*	**Tipe data**	**Keterangan**
Tidak ada (langsung berupa *object*)	*object* dari dokumen Nodes.	- Perangkat IoT.



Gambar 4.14 Alur melihat perangkat IoT berdasarkan id

Penjelasan dari Gambar 4.14 adalah sebagai berikut:

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

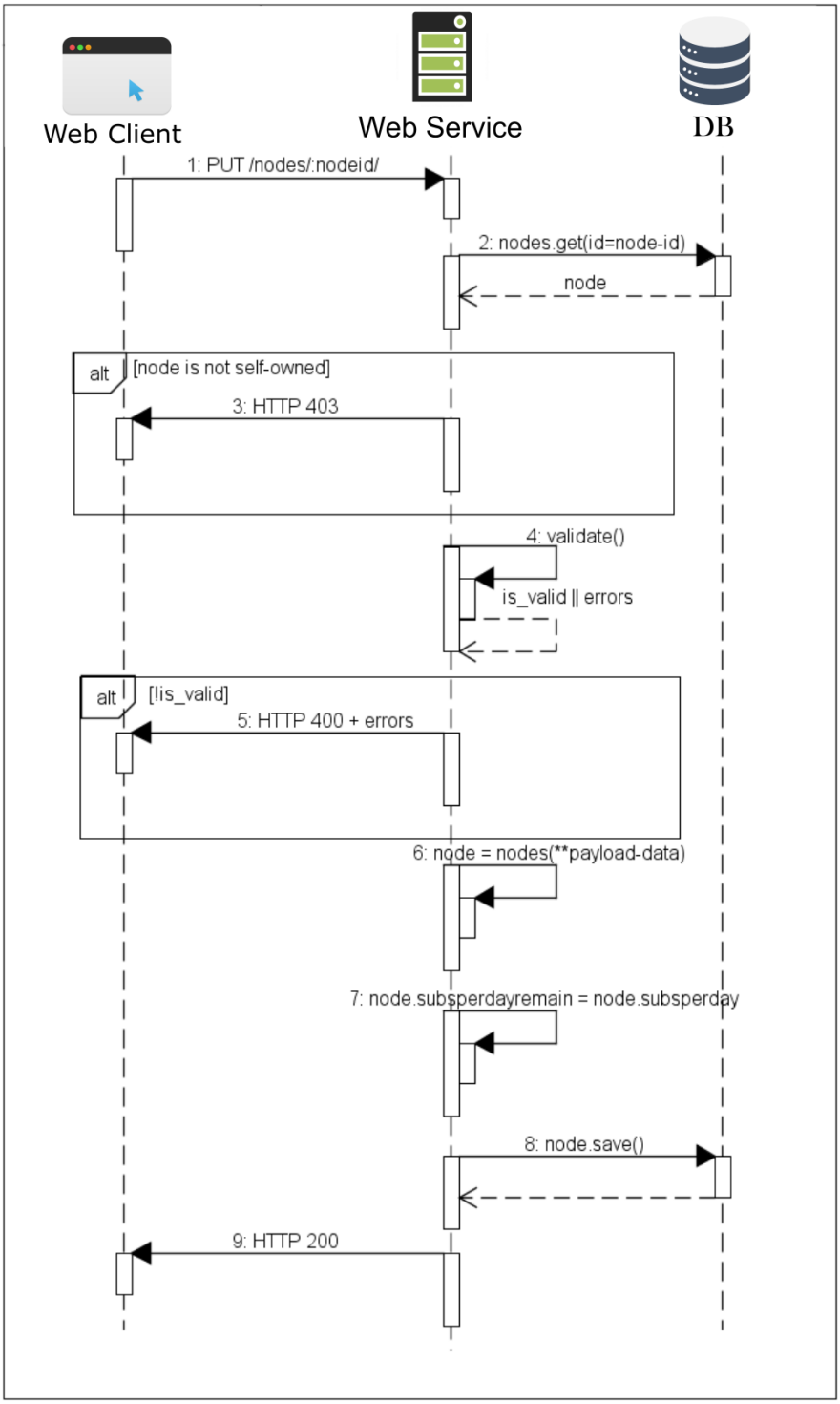
4) *Web service* mengirimkan *response* dengan HTTP status 200 beserta data perangkat IoT yang diminta.

d. Mengubah perangkat IoT

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 IoT digambarkan pada Gambar 4.15.

Tabel 4.18 Mengubah perangkat IoT

***Request* PUT /nodes/:nodeid/**		
Deskripsi	***Payload***	**Keterangan**
Mengubah data perangkat IoT yang dimiliki pengguna.	- *user* - *label* - *secretkey* - *subsperday* - *is_public*	- *payload* bersifat opsional, kirim hanya bagian yang ingin dirubah.
Response		
Nama *field*	**Tipe data**	**Keterangan**
Tidak ada (langsung berupa *object*)	*object* dari dokumen Nodes.	Perangkat IoT yang dirubah.



Gambar 4.15 Alur mengubah perangkat IoT

Penjelasan dari Gambar 4.15 adalah sebagai berikut:

1) *Web client* mengirimkan HTTP *request* dengan metode PUT dengan *payload* berisi hanya *field* yang ingin dirubah. Informasi lebih lengkap mengenai *payload* dapat dilihat pada Tabel 4.18.

2) *Web service* memanggil fungsi `nodes.get(id=node-id)` untuk mendapatkan data perangkat IoT berdasarkan id. Kemudian, DB mengembalikan data yang diminta dalam variabel `node`.

3) Jika perangkat IoT 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 IoT baru terhadap perangkat IoT 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.

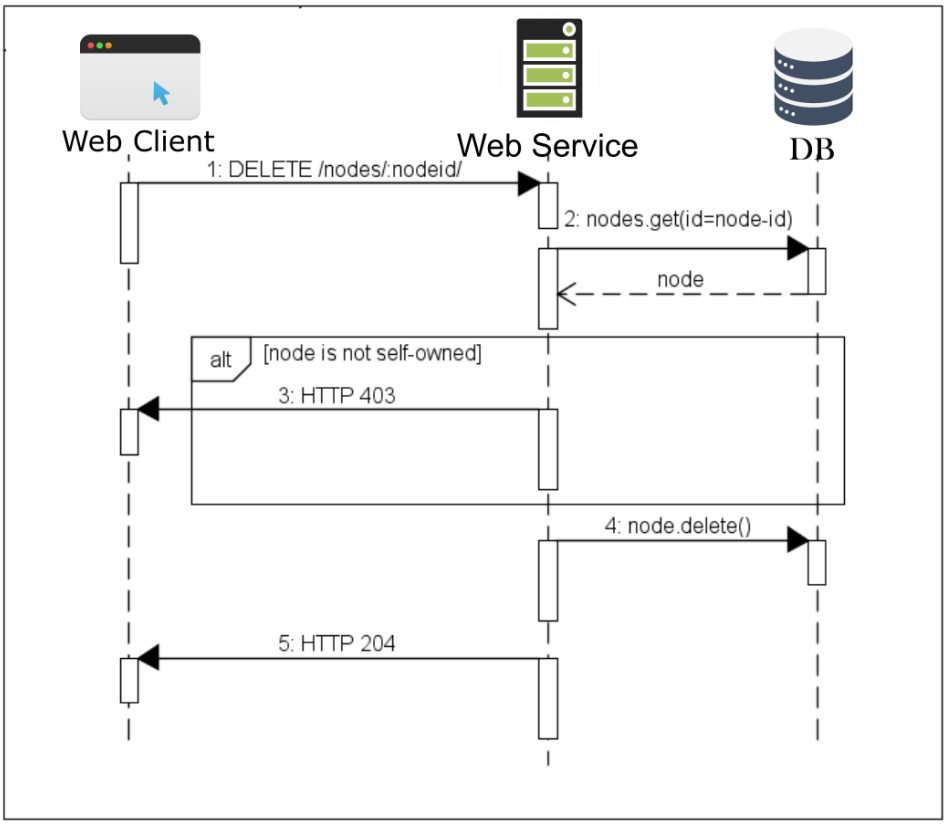
9) *Web service* mengirimkan *response* dengan HTTP status 200.

e. Menghapus perangkat IoT

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 IoT digambarkan pada Gambar 4.16.

Tabel 4.19 Menghapus perangkat IoT

***Request* DELETE /nodes/:nodeid/**		
Deskripsi	***Payload***	**Keterangan**
Menghapus perangkat IoT berdasarkan id.	-	-
Response		
Nama *field*	**Tipe data**	**Keterangan**
-	-	-



Gambar 4.16 Alur menghapus perangkat IoT

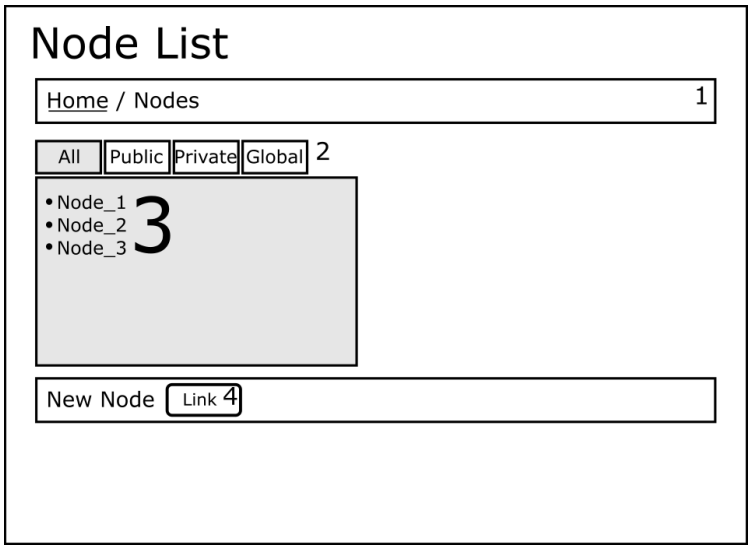
Penjelasan dari Gambar 4.16 adalah sebagai berikut:

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

4.3.4 Perancangan Komponen *Web Console*

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

a. Lihat Semua Perangkat IoT

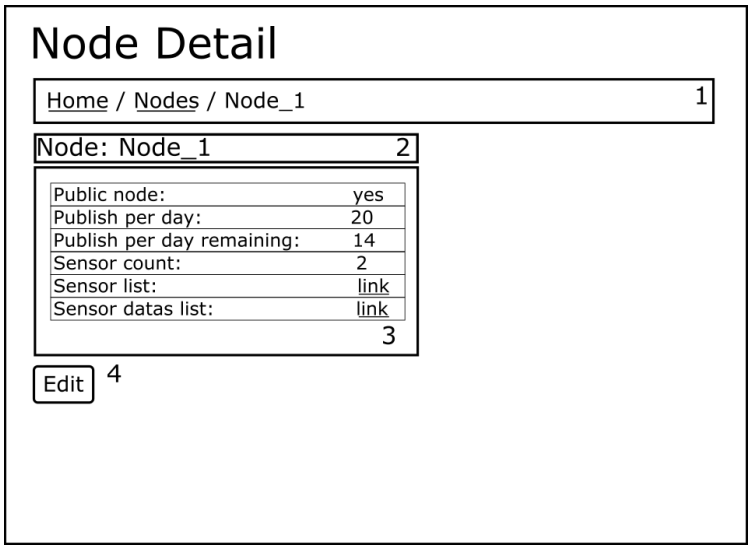


Gambar 4.17 Perancangan *web console*: lihat semua perangkat IoT

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.17 di atas adalah:

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

b. Lihat Perangkat IoT

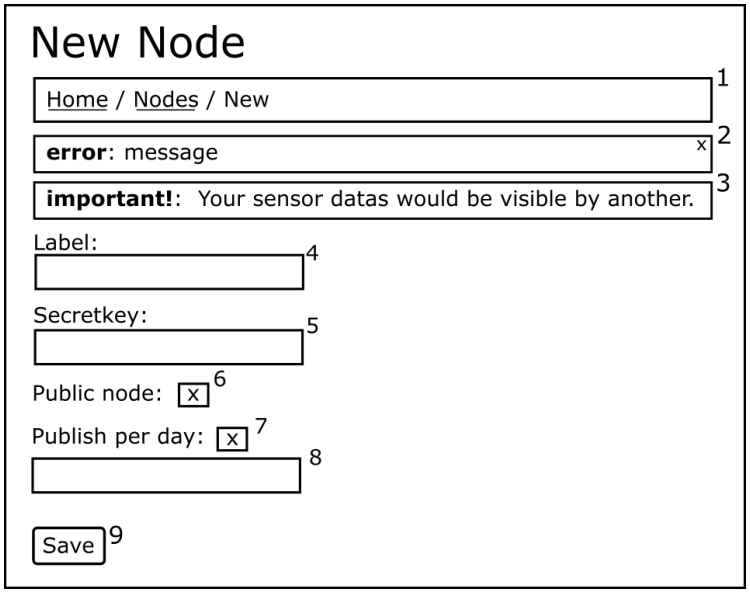


Gambar 4.18 Perancangan *web console*: lihat perangkat IoT

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.18 di atas adalah:

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

c. Buat Perangkat IoT



Gambar 4.19 Perancangan *web console*: buat perangkat sensor

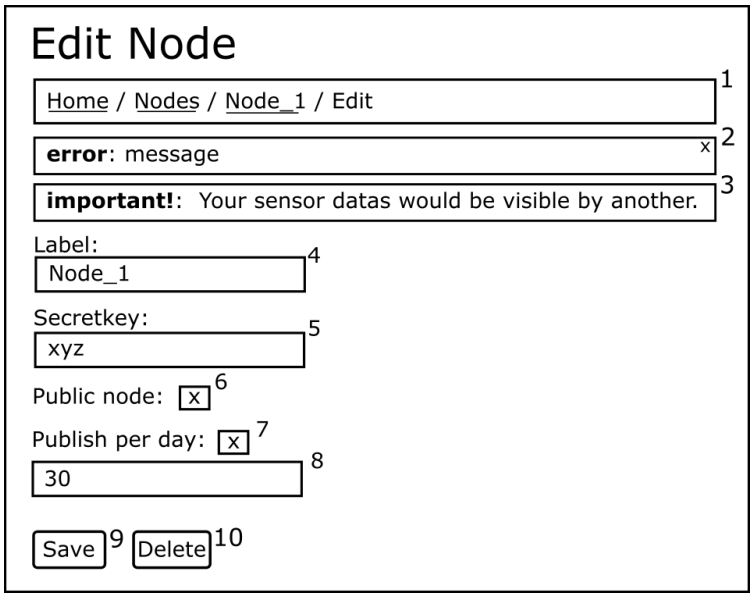
Penjelasan dari rancangan antar muka pengguna pada Gambar 4.19 di atas adalah:

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 IoT.
5) Elemen *text-input* yang menampung *secretkey* dari perangkat IoT.
6) Elemen *check-box* yang menampung visibilitas perangkat IoT terhadap pengguna lain. Jika elemen ini tercentang, maka perangkat IoT yang dibuat akan memiliki visibilitas publik, jika tidak perangkat IoT akan memiliki visibilitas privat.
7) Elemen *check-box* yang menentukan pembatasan pengiriman data sensor dari perangkat IoT 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 IoT yang dibuat tidak dibatasi.

8) Elemen *input* yang menampung jumlah data sensor yang dapat dikirimkan oleh perangkat IoT 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 IoT dan mengarahkan pengguna ke halaman lihat semua perangkat IoT.

d. Ubah Perangkat IoT



Gambar 4.20 Perancangan *web console*: ubah perangkat IoT

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.20 di atas 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 IoT.

5) Elemen *text-input* yang menampung *secretkey* dari perangkat IoT.

6) Elemen *check-box* yang menampung visibilitas perangkat IoT terhadap pengguna lain. Jika elemen ini tercentang, maka perangkat IoT yang dibuat akan memiliki visibilitas publik, jika tidak perangkat IoT akan memiliki visibilitas privat.

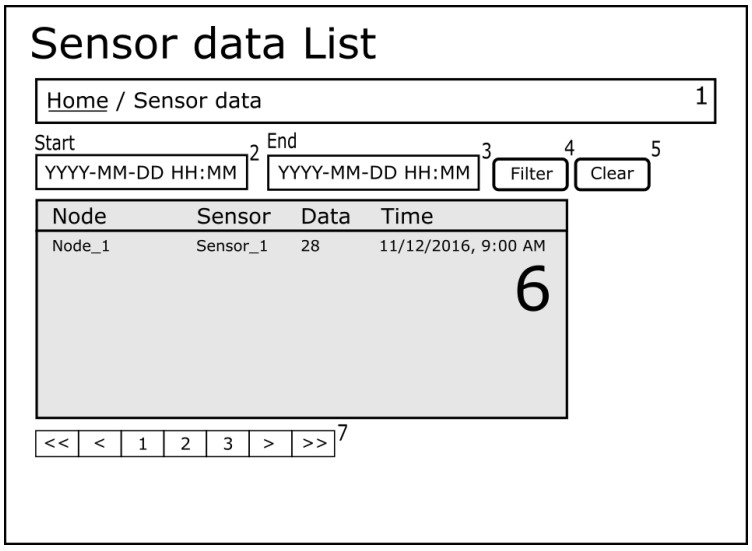
7) Elemen *check-box* yang menentukan pembatasan pengiriman data sensor dari perangkat IoT setiap harinya. Jika elemen ini tercentang, maka elemen *input* pada nomor 8 akan ditampilkan untuk diisi jumlah batasan pengirimannya tersebut. Sedangkan jika elemen ini tidak tercentang, pengiriman data sensor dari perangkat IoT yang dibuat tidak dibatasi.

8) Elemen *input* yang menampung jumlah data sensor yang dapat dikirimkan oleh perangkat IoT 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 IoT dan mengarahkan pengguna ke halaman lihat semua perangkat IoT.

10) Elemen *button* yang ketika ditekan akan menampilkan konfirmasi hapus. Jika dikonfirmasi, *request* hapus akan dikirimkan ke *web service* dan mengarahkan pengguna ke halaman lihat semua perangkat IoT.

e. Lihat Data Sensor



Gambar 4.21 Perancangan *web console*: lihat data sensor

Penjelasan dari rancangan antar muka pengguna pada Gambar 4.21 di atas 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, di mana data sensor pada setiap paginasi ditampilkan maksimal berjumlah 10.
7) Elemen *pagination* untuk pindah ke halaman paginasi yang dipilih.

BAB 5 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.

5.1 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.

5.1.1 Instalasi Apache Web Server

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
```

5.1.2 Instalasi Django Web Framework

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

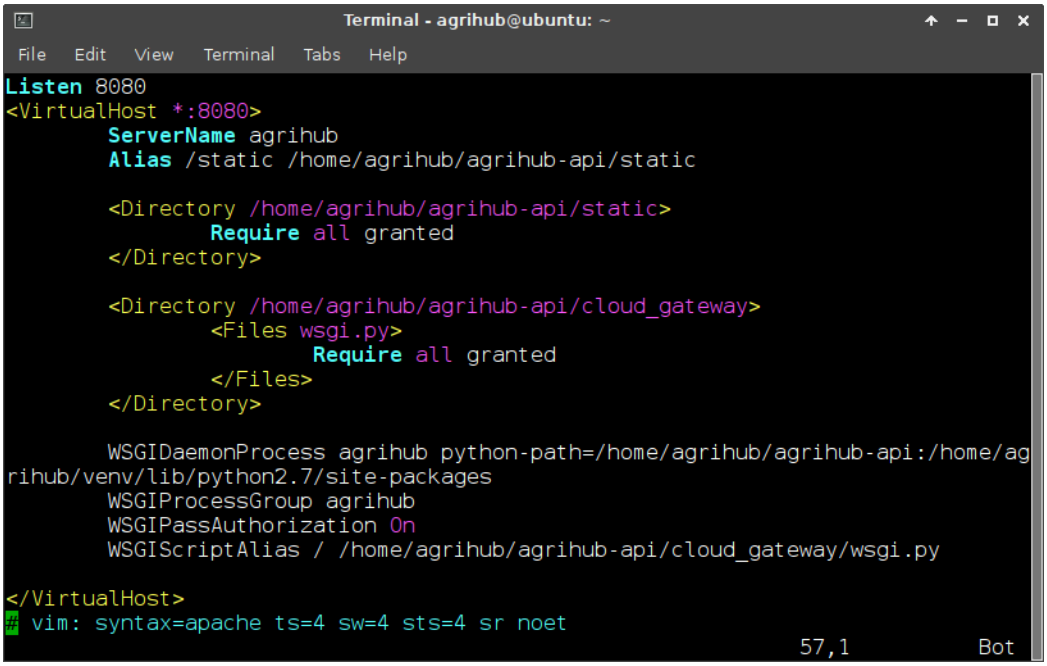
```
pymongo==3.3.1
Django==1.10.3
mongoengine==0.10.6
pymongo==3.3.1
djangorestframework_jwt==1.8.0
django_rest_framework_mongoengine==3.3.1
PyJWT==1.4.2
djangorestframework==3.5.3
django-cors-headers==1.3.1
```

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

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

5.1.3 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 ditambahkan ke dalam file konfigurasi itu adalah: alamat CGI, alamat interpreter Python, alamat alias untuk static file (css, dan JS bawaan Django), dan alamat *port* di mana aplikasi Django tersebut dijalankan. Berikut konfigurasi mod_wsgi pada Apache ditampilkan pada Gambar 5.1.



Gambar 5.1 Konfigurasi Apache

5.2 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*.

5.2.1 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
```

5.2.2 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 IoT, dokumen sensor dan dokumen data sensor.

a. 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  from mongoengine.document import Document
2  from mongoengine import StringField, EmailField, IntField
3
4
5  class User(Document):
6      email = EmailField(required=True)
7      username = StringField(min_length=4, max_length=16,
8  unique=True, required=True)
9      password = StringField(min_length=8, max_length=128,
10 required=True)
11     first_name = StringField(max_length=50, required=True)
12     last_name = StringField(max_length=50, required=True)
13     is_admin = IntField(default=0)
14
15     meta = {
16         'indexes': [
17             {
18                 'fields': ['-username'],
19                 'unique': True
20             },
21         ],
22     }
```

Penjelasan dari potongan kode program di atas 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 `email` untuk menampung alamat email pengguna. Atribut ini harus didefinisikan dengan alamat email yang valid.
4) Baris 7-8, deklarasi atribut kelas bernama `username` untuk menampung *username* pengguna. Atribut ini harus didefinisikan sebagai *string* dengan 4-16 karakter.
5) Baris 9-10, deklarasi atribut kelas bernama `password` untuk menampung *password* pengguna. Atribut ini harus didefinisilan sebagai *string* dengan 8-128 karakter.
6) Baris 11, deklarasi atribut kelas bernama `firstname` untuk menampung nama depan pengguna. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 50 karakter.
7) Baris 12, deklarasi atribut kelas bernama `lastname` untuk menampung nama belakang pengguna. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 50 karakter.

8) Baris 13, deklarasi atribut kelas bernama `is_admin` untuk 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 `username` memiliki nilai yang unik terhadap pengguna lainnya.

b. Dokumen Perangkat IoT

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

```
1  from mongoengine.document import Document
2  from mongoengine import StringField, ReferenceField,
3  EmbeddedDocumentListField, IntField, CASCADE
4  from sensors.models import Sensors
5  from users.models import User
6
7
8  class Nodes(Document):
9      user = ReferenceField(User, reverse_delete_rule=CASCADE)
10     label = StringField(max_length=28)
11     secretkey = StringField(required=True, max_length=16)
12     is_public = IntField(default=0)
13     pubperday = IntField(default=0)
14     pubperdayremain = IntField(default=0)
15     sensors = EmbeddedDocumentListField(document_type=Sensors)
```

Penjelasan dari potongan kode program di atas 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 `user` untuk menampung nilai id dari pengguna yang memiliki perangkat IoT itu.

4) Baris 10, deklarasi atribut kelas bernama `label` untuk menampung label dari perangkat IoT. Atribut ini harus didefinisilan sebagai *string* dengan panjang maksimal 28 karakter.

5) Baris 11, deklarasi atribut kelas bernama `secretkey` untuk menampung kunci rahasia dari perangkat IoT. Atribut ini harus didefinisikan sebagai *string* dengan panjang maksimal 16 karakter.

6) Baris 12, deklarasi atribut kelas bernama `is_public` untuk menampung visibilitas perangkat. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.

7) Baris 13, deklarasi atribut kelas bernama `pubperday` untuk menampung batasan pengiriman data sensor milik perangkat IoT. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.

8) Baris 14, deklarasi atribut kelas bernama `pubperdayremain` untuk menampung nilai *counter* dari batasan pengiriman data sensor milik perangkat IoT setiap harinya. Atribut ini harus didefinisikan sebagai bilangan bulat yang memiliki nilai bawaan 0.

9) Baris 15, deklarasi atribut kelas bernama `sensors` untuk menyimpan daftar sensor yang dimiliki perangkat IoT. Atribut ini didefinisikan sebagai *EmbeddedDocument* dari dokumen sensor.

c. Dokumen Sensor

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

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

Penjelasan dari potongan kode program di atas 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 `id` untuk 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 `label` untuk 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 `label` memiliki nilai yang unik terhadap sensor lainnya.

6) Baris 21-22, deklarasi *method* yang merepresentasikan *object* dari dokumen sensor berdasarkan nama labelnya.

d. Dokumen Data Sensor

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

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

Penjelasan dari potongan kode program di atas 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 `node` untuk menampung nilai id dari perangkat IoT yang mengirimkan data sensor tersebut.
4) Baris 10, deklarasi atribut kelas bernama `sensor` untuk menampung nilai id dari sensor yang mengambil nilai dari data sensor tersebut.
5) Baris 11, deklarasi atribut kelas bernama `data` untuk menampung nilai data sensor.
6) Baris 12, deklarasi atribut kelas bernama `timestamp` untuk menampung nilai data sensor. Atribut ini harus didefinisikan sebagai *object* dari kelas `datetime`.

5.2.3 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 IoT, dan berdasarkan sensor.

a. Berdasarkan pengguna

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

```
1  ...
2
3  class SensordataFilterUser(ListAPIView):
4      authentication_classes = (JSONWebTokenAuthentication,)
5      permission_classes = (IsUser,)
6      serializer_class = SensordataSerializer
7
8      def get_queryset(self):
9          user = self.request.user
10
11         if user.username != self.kwargs.get('user'):
12             raise PermissionDenied(detail="Your credential and URL
13 prefix must be same.")
14
15         nodes = Nodes.objects.filter(user=user)
16         tmp = []
17
18         filter_from = self.request.GET.get('start')
19         filter_last = self.request.GET.get('end')
20
21         for node in nodes:
22             if filter_from and filter_last :
23                 all_subs = Sensordatas.objects.filter(
24                     node=node, timestamp__gte=filter_from,
25 timestamp__lte=filter_last
26                 )
27             elif filter_from:
28                 all_subs = Sensordatas.objects.filter(node=node,
29 timestamp__gte=filter_from)
30             elif filter_last:
31                 all_subs = Sensordatas.objects.filter(node=node,
32 timestamp__lte=filter_last)
33             else:
34                 all_subs = Sensordatas.objects.filter(node=node)
35
36             for sub in all_subs:
37                 tmp.append(sub)
38         return tmp
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `SensorsdataFilterUser` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.
2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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.

6) Baris 11-13, jika `user.username` tidak sesuai dengan username pada alamat URL, *web service* akan mengembalikan HTTP status 403.

7) Baris 15, deklarasi variabel `nodes` yang menampung daftar perangkat IoT yang dimiliki pengguna.

8) Baris 16, deklarasi variabel `tmp` yang menampung *object* data sensor.

9) Baris 18, deklarasi variabel `filter_from` yang menampung parameter HTTP GET `'start'` yang didapat dari URL.

10) Baris 19, deklarasi variabel `filter_last` yang menampung parameter HTTP GET `'end'` yang didapat dari URL.

11) Baris 21-37, iterasi sebanyak *object items* pada variabel `nodes`. Pada setiap iterasi, item perangkat IoT dapat diakses menggunakan variabel `node`.

12) 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`.

13) Baris 36-37, iterasi sebanyak data sensor yang ada pada variabel `all_subs`. Pada setiap iterasi data sensor tersebut dimasukkan ke dalam variabel `tmp`.

14) Baris 38, *method* mengembalikan variabel `tmp`.

b. Berdasarkan perangkat IoT

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

```
1  ...
2
3  class SensordataFilterNode(ListAPIView):
4      authentication_classes = (JSONWebTokenAuthentication,)
5      permission_classes = (IsUser,)
6      serializer_class = SubscriptionSerializer
7  ...
8      def get_queryset(self):
9          nodeid = self.kwargs['node']
10         node = self.checknode(nodeid)
11         if self.request.user != node.user and 0 == node.is_public:
12             return 'unauthorized'
13
```

```
14          filter_from = self.request.GET.get('start')
15          filter_last = self.request.GET.get('end')
16
17          if filter_from and filter_last:
18              return Sensordatas.objects.filter(
19                  node=node, timestamp__gte=filter_from,
20  timestamp__lte=filter_last
21              )
22          elif filter_from:
23              return Sensordatas.objects.filter(node=node,
24  timestamp__gte=filter_from)
25          elif filter_last:
26              return Sensordatas.objects.filter(node=node,
27  timestamp__lte=filter_last)
28          else:
29              return Sensordatas.objects.filter(node=node)
30
31      def get(self, request, *args, **kwargs):
32          raw_queryset = self.get_queryset()
33          if 'unauthorized' == raw_queryset:
34              return Response({
35                  'detail': 'Not found.'
36              }, status=status.HTTP_404_NOT_FOUND)
37          queryset = self.filter_queryset(raw_queryset)
38          page = self.paginate_queryset(queryset)
39          ...
40          serializer = SensordataSerializer(page, many=True,
41  context={'request': request})
42              return self.get_paginated_response(serializer.data)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `SensorsdataFilterUser` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.
2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT, diambil dari alamat URL.

7) Baris 10, deklarasi variabel `node` yang menampung *object* perangkat IoT berdasarkan id nya.

7) Baris 11-12, jika perangkat IoT yang diakses bukan milik pengguna dan perangkat IoT tersebut tidak memiliki label visibilitas publik, *method* akan mengembalikan *string* dengan nilai `'unauthorized'`.

8) Baris 14, deklarasi variabel `filter_from` yang menampung parameter HTTP GET `'start'` yang didapat dari URL.

9) Baris 15, deklarasi variabel `filter_last` yang menampung parameter HTTP GET `'end'` yang didapat dari URL.

10) Baris 17-29, mendapatkan data sensor dari *object* `node` sesuai dengan kriteria waktu yang didefinisikan dalam parameter HTTP GET `'start'` dan `'end'`.

11) Baris 31, *override method* `get()` dari kelas `ListAPIView`. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP GET.

12) Baris 32, deklarasi variabel `raw_queryset` yang menampung data mentah dari *method* `get_queryset()`.

13) Baris 33-36, jika nilai dari variabel `raw_queryset` yang berisi *string* `'unauthorized'`, *web service* akan mengembalikan HTTP status 404.

14) Baris 37, deklarasi variabel `queryset` yang menampung data sensor yang sudah diolah sehingga dapat digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman.

15) Baris 38, deklarasi variabel `page` yang menampung data sensor sesuai dengan halaman yang diminta.

16) Baris 40-41, deklarasi variabel `serializer` yang menampung representasi data sensor sesuai dengan kriteria dan halaman yang diminta.

17) Baris 42, *web service* mengembalikan HTTP status 200 beserta data sensor yang diminta.

c. 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  class SensordataFilterNodeSensor(ListAPIView):
4      authentication_classes = (JSONWebTokenAuthentication,)
5      permission_classes = (IsUser,)
6      serializer_class = SensordataSerializer
7  ...
8      def get_queryset(self):
9          nodeid = self.kwargs['node']
```

```
10          sensorid = self.kwargs['sensor']
11
12          node_sensor = self.checknode(nodeid, sensorid)
13          if self.request.user != node_sensor.get('node').user and 0
14  == node_sensor.get('node').is_public:
15              return 'unauthorized'
16
17          filter_from = self.request.GET.get('start')
18          filter_last = self.request.GET.get('end')
19
20          if filter_from and filter_last:
21              return Sensordatas.objects.filter(
22                  node=node_sensor.get('node').id,
23  sensor=node_sensor.get('sensor').id,
24                  timestamp__gte=filter_from,
25  timestamp__lte=filter_last
26              )
27          elif filter_from:
28              return Sensordatas.objects.filter(
29                  node=node_sensor.get('node').id,
30  sensor=node_sensor.get('sensor').id,
31                  timestamp__gte=filter_from
32              )
33          elif filter_last:
34              return Sensordatas.objects.filter(
35                  node=node_sensor.get('node').id,
36  sensor=node_sensor.get('sensor').id,
37                  timestamp__lte=filter_last
38              )
39          else:
40              return
41  Sensordatas.objects.filter(node=node_sensor.get('node').id,
42  sensor=node_sensor.get('sensor').id)
43
44      def get(self, request, *args, **kwargs):
45          raw_queryset = self.get_queryset()
46          if 'unauthorized' == raw_queryset:
47              return Response({
48                  'detail': 'Not found.'
49              }, status=status.HTTP_404_NOT_FOUND)
50          queryset = self.filter_queryset(raw_queryset)
51          page = self.paginate_queryset(queryset)
52          ...
53          serializer = SensordataSerializer(page, many=True,
54  context={'request': request})
55          return self.get_paginated_response(serializer.data)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `SensorsdataFilterSensor` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.
2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT, 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 IoT dan sensor.

8) Baris 13-15, jika perangkat IoT yang diakses bukan milik pengguna dan perangkat IoT tersebut tidak memiliki label visibilitas publik, *method* akan mengembalikan *string* dengan nilai `'unauthorized'`.

9) Baris 17, deklarasi variabel `filter_from` yang menampung parameter HTTP GET `'start'` yang didapat dari URL.

10) Baris 18, deklarasi variabel `filter_last` yang menampung parameter HTTP GET `'end'` yang didapat dari URL.

11) Baris 20-42, mendapatkan data sensor sesuai dengan kriteria waktu yang didefinisikan dalam parameter HTTP GET `'start'` dan `'end'`.

12) Baris 44, *override method* `get()` dari kelas `ListAPIView`. *Method* ini di-*override* untuk menspesifikasi penanganan permintaan HTTP GET.

13) Baris 45, deklarasi variabel `raw_queryset` yang menampung data mentah dari *method* `get_queryset()`.

14) Baris 46-49, jika nilai dari variabel `raw_queryset` yang berisi *string* `'unauthorized'`, *web service* akan mengembalikan HTTP status 404.

15) Baris 50, deklarasi variabel `queryset` yang menampung data sensor yang sudah diolah sehingga dapat digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman.

16) Baris 51, deklarasi variabel `page` yang menampung data sensor sesuai dengan halaman yang diminta.

17) Baris 53-54, deklarasi variabel `serializer` yang menampung representasi data sensor sesuai dengan kriteria dan halaman yang diminta.

18) Baris 55, *web service* mengembalikan HTTP status 200 beserta data sensor yang diminta.

5.3 Implementasi Komponen *Security*

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

5.3.1 Implementasi Autentikasi dan Otorisasi

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

a. Mendapatkan token akses JWT

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

- Perangkat IoT

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

```
1   ...
2
3   class NodeTokenCreator(APIView):
4       def post(self, request, format=None):
5           form = NodeAuthForm(request.data)
6           if form.is_valid():
7               return Response({
8                   'node': NodeSerializer(form.node,
9   context={'request': request}).data,
10                  'token': self.create_token(form.node)
11              })
12          return Response(form.errors,
13  status=status.HTTP_400_BAD_REQUEST)
14
15      @staticmethod
16      def create_token(node):
17          payload = node_jwt_payload_handler(node)
18          token = jwt.encode(payload, settings.SECRET_KEY)
19          return token.decode('unicode_escape')
```

Penjelasan dari potongan kode program di atas 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 IoT 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.

- *Web client*

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

```
1  ...
2
3  class UserTokenCreator(APIView):
4      def post(self, request, format=None):
5          form = UserAuthForm(request.data)
6          if form.is_valid():
7              return Response({
8                  'user': UserSerializer(form.user).data,
9                  'token': self.create_token(form.user)
10             })
11         return Response(form.errors,
12 status=status.HTTP_400_BAD_REQUEST)
13
14     @staticmethod
15     def create_token(user):
16         payload = user_jwt_payload_handler(user)
17         token = jwt.encode(payload, settings.SECRET_KEY)
18         return token.decode('unicode_escape')
```

Penjelasan dari potongan kode program di atas 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.

b. Registrasi Pengguna

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

```
1  ...
2
3  class ResearcherRegistration(GenericAPIView):
4      serializer_class = UserSerializer
5
6      @staticmethod
7      def post(request):
8          ...
9          serializer = UserSerializer(data=request.data,
10 context={'request': request})
11         if serializer.is_valid():
12             serializer.save()
13             return Response(serializer.data,
14 status=status.HTTP_201_CREATED)
15         return Response(serializer.errors,
16 status=status.HTTP_400_BAD_REQUEST))
```

Penjelasan dari potongan kode program di atas 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.

c. Menerima data sensor dari perangkat IoT

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

```
1  ...
2
3  class SensordatasList(ListAPIView):
4      authentication_classes = (JSONWebTokenAuthentication,)
5      permission_classes = (IsAuthenticated,)
6      ...
7      serializer_class = SensordataSerializer
8
9      @staticmethod
10     def post(request):
11         if not isinstance(request.user, Nodes):
12             raise exceptions.AuthenticationFailed("You do not have
13 permission to perform this action.")
14
15         serformat = SensordataFormatSerializer(data=request.data,
16 context={'request': request})
17         if serformat.is_valid():
18             data = serformat.save()
19             return Response(
20                 { "results": SensordataSerializer(data, many=True,
21 context={'request': request}).data },
22                 status=status.HTTP_201_CREATED
23             )
24         else:
25             return Response(serformat.errors,
26 status=status.HTTP_400_BAD_REQUEST)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `SensordatasList` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.
2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi tersebut adalah `JSONWebTokenAuthentication`.
3) Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa layanan ini hanya dapat dilakukan oleh pihak yang telah terautentikasi. 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 IoT, *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.

5.3.2 Implementasi Menejemen Perangkat

Implementasi manajemen perangkat berisi kode sumber untuk memanajemen perangkat.

a. Membuat perangkat IoT baru

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

```
1  ...
2
3  class NodesList(ListAPIView):
4
5      authentication_classes = (JSONWebTokenAuthentication,)
6      permission_classes = (IsUser,)
7      serializer_class = NodeSerializer
8      ...
9      @staticmethod
10     def post(request):
11         ...
12         serializer = NodeSerializer(data=request.data,
13 context={'request': request})
14         if serializer.is_valid():
15             serializer.save()
16             return Response(serializer.data,
17 status=status.HTTP_201_CREATED)
18         return Response(serializer.errors,
19 status=status.HTTP_400_BAD_REQUEST)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `NodesList` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.

2) Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT sebelum disimpan ke dalam basis data.

7) Baris 14-17, jika validasi berhasil, *object* perangkat IoT 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.

b. Melihat perangkat IoT

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

```
1  ...
2
3  class NodesList(ListAPIView):
4
5      authentication_classes = (JSONWebTokenAuthentication,)
6      permission_classes = (IsUser,)
7      serializer_class = NodeSerializer
8      ...
9      def get(self, request, *args, **kwargs):
10         queryset =
11 self.filter_queryset(self.get_nodes(request.user,
12 request.GET.get('role')))
13         page = self.paginate_queryset(queryset)
14         if page is not None:
15             serializer = NodeSerializer(page, many=True,
16 context={'request': request})
17             return self.get_paginated_response(serializer.data)
18
19         serializer = self.get_serializer(queryset, many=True)
20         return Response(serializer.data)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `NodesList` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.

2) Baris 5, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT sehingga dapat digunakan fitur *pagination* untuk membagi data ke dalam beberapa halaman.

7) Baris 13, deklarasi variabel `page` yang menampung perangkat IoT sesuai dengan halaman yang diminta.

8) Baris 14-17, jika parameter HTTP GET `page` didefinisikan, *web service* mengembalikan HTTP status 200 beserta perangkat IoT 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 IoT.

c. Melihat perangkat IoT berdasarkan id

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

```
1  ...
2
3  class NodeDetail(GenericAPIView):
4      authentication_classes = (JSONWebTokenAuthentication,)
5      permission_classes = (IsUser,)
6
7      @staticmethod
8      def get_object(pk):
9          try:
10             return Nodes.objects.get(pk=pk)
11         except Exception:
12             raise Http404
13
14     def get(self, request, pk, format=None):
15         node = self.get_object(pk)
16         if request.user != node.user and 0 == node.is_public:
17             return Response({
18                 'detail': 'Not found.'
19             }, status=status.HTTP_404_NOT_FOUND)
20         serializer = NodeSerializer(node, context={'request':
21 request})
22         return Response(serializer.data)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `NodesDetail` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.

2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT 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 IoT dari *method* `get_object()`.

7) Baris 16-19, jika perangkat IoT yang diakses bukan milik pengguna dan perangkat IoT tersebut tidak memiliki label visibilitas publik, *web service* mengembalikan HTTP status 404.

8) Baris 20-21, *web service* mengembalikan HTTP status 200 beserta perangkat IoT yang diminta.

d. Mengubah perangkat IoT

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

```
1  ...
2
3  class NodeDetail(GenericAPIView):
4      authentication_classes = (JSONWebTokenAuthentication,)
5      permission_classes = (IsUser,)
6
7      @staticmethod
8      def get_object(pk):
9          try:
10             return Nodes.objects.get(pk=pk)
11         except Exception:
12             raise Http404
13
14     def put(self, request, pk, format=None):
15         node = self.get_object(pk)
16         if request.user != node.user:
17             return Response({
18                 'detail': 'You can not update another person node.'
19             }, status=status.HTTP_403_FORBIDDEN)
20         serializer = NodeSerializer(node, data=request.data,
21 context={'request': request}, partial=True)
22         if serializer.is_valid():
23             serializer.save()
24             return Response(serializer.data,
25 status=status.HTTP_200_OK)
26         return Response(serializer.errors,
27 status=status.HTTP_400_BAD_REQUEST)
```

Penjelasan dari potongan kode program sebelumnya adalah:

1) Baris 3, deklarasi kelas `NodesDetail` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.
2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT 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 IoT dari *method* `get_object()`.
7) Baris 16-19, jika perangkat IoT yang diakses bukan milik pengguna, *web service* mengembalikan HTTP status 403.
8) Baris 20-21, deklarasi variabel `serializer` yang menangani validasi *object* perangkat IoT sebelum disimpan ke dalam basis data.
9) Baris 22-25, jika validasi berhasil, *object* perangkat IoT 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.

e. Menghapus perangkat IoT

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

```
1   ...
2
3   class NodeDetail(GenericAPIView):
4       authentication_classes = (JSONWebTokenAuthentication,)
5       permission_classes = (IsUser,)
6
7       @staticmethod
8       def get_object(pk):
9           try:
10              return Nodes.objects.get(pk=pk)
11          except Exception:
12              raise Http404
13
14      def delete(self, request, pk, format=None):
15          node = self.get_object(pk)
```

```
16          if request.user != node.user:
17              return Response({
18                  'detail': 'You can not delete another person node.'
19              }, status=status.HTTP_403_FORBIDDEN)
20          node.delete()
21          return Response(status=status.HTTP_204_NO_CONTENT)
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3, deklarasi kelas `NodesDetail` yang mewarisi atribut dan *method* dari kelas `ListAPIView`.
2) Baris 4, deklarasi atribut kelas yang mendeskripsikan bahwa autentikasi pada kriteria ini menggunakan token akses JWT. Kelas yang menangani autentikasi 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 IoT 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 IoT dari *method* `get_object()`.
7) Baris 16-19, jika perangkat IoT yang diakses bukan milik pengguna, *web service* mengembalikan HTTP status 403.
8) Baris 20, menghapsu *object* perangkat IoT di basis data.
9) Baris 21, *web service* akan mengembalikan HTTP status 204 menandakan perangkat IoT berhasil dihapus.

5.4 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. Komponen *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.

5.4.1 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": "web-console ",
  "version": "1.0.0",
  "description": "Agri Hub Web Client",
  "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:

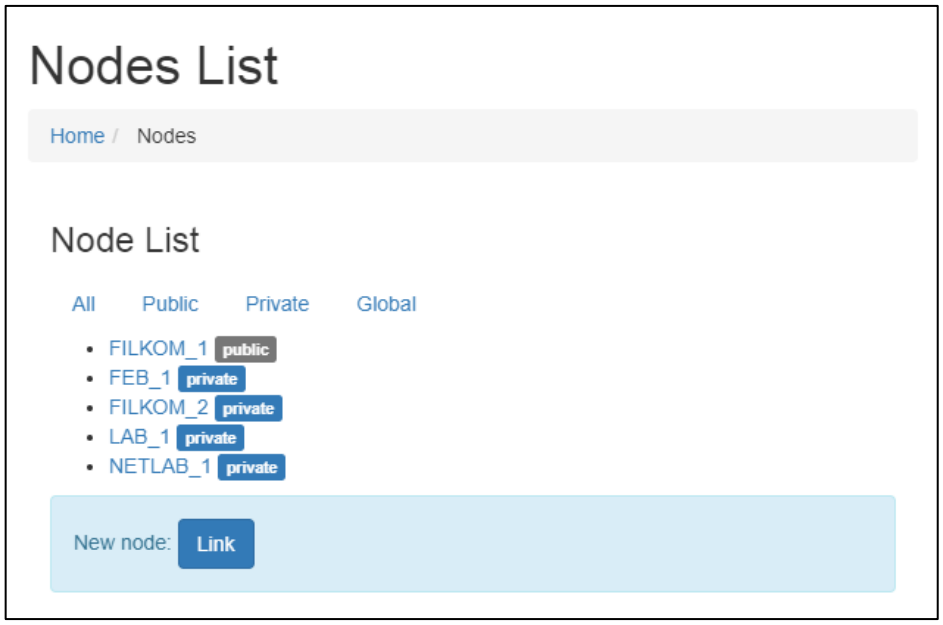
```
$ npm install
```

5.4.2 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 IoT, lihat perangkat IoT, buat perangkat IoT, ubah perangkat IoT dan lihat data sensor.

a. Lihat Semua Perangkat IoT

   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 IoT dapat dilihat pada Gambar 5.2.



Gambar 5.2 Implementasi aplikasi web *console*: lihat semua perangkat IoT

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

```
...

@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 {
```

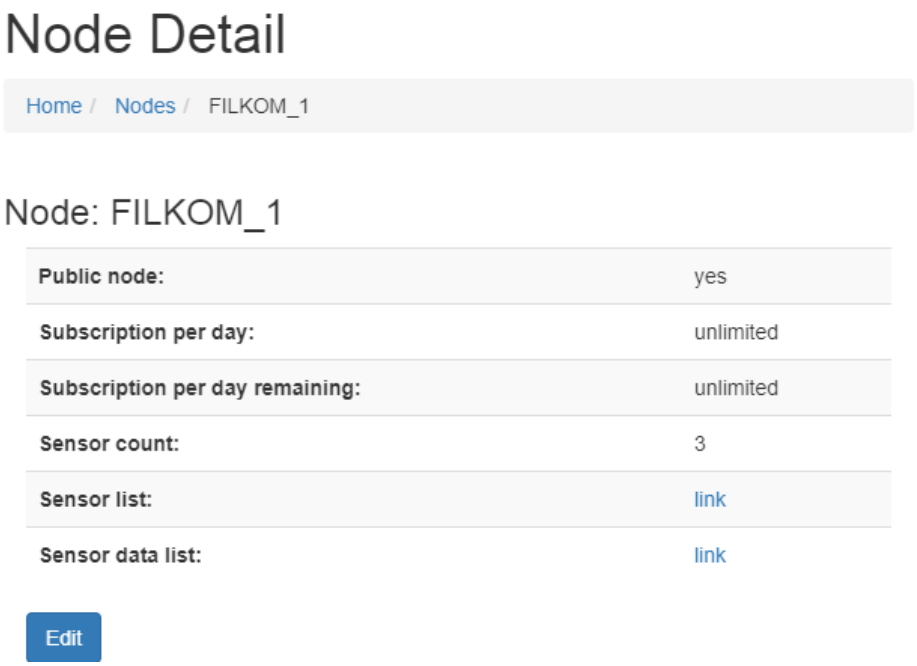
```
34
35  this.router.navigateByUrl(`/nodes?visibility=${$event.nextId}`);
36          this.getNodes($event.nextId);
37      }
38  }
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3-6, deklarasi *component* `NodeComponent` yang mengimplementasi *interface* `OnInit`.
2) Baris 8, deklarasi variabel `nodes` untuk menampung *array* dari *object* perangkat IoT yang nantinya didapatkan dari web service.
3) Baris 9, deklarasi variabel `links` untuk 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 IoT 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 IoT yang ditampilkan.

b. Lihat Perangkat IoT

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



Gambar 5.3 Implementasi aplikasi web *console*: lihat perangkat IoT

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

```
...

@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]);
    }

```

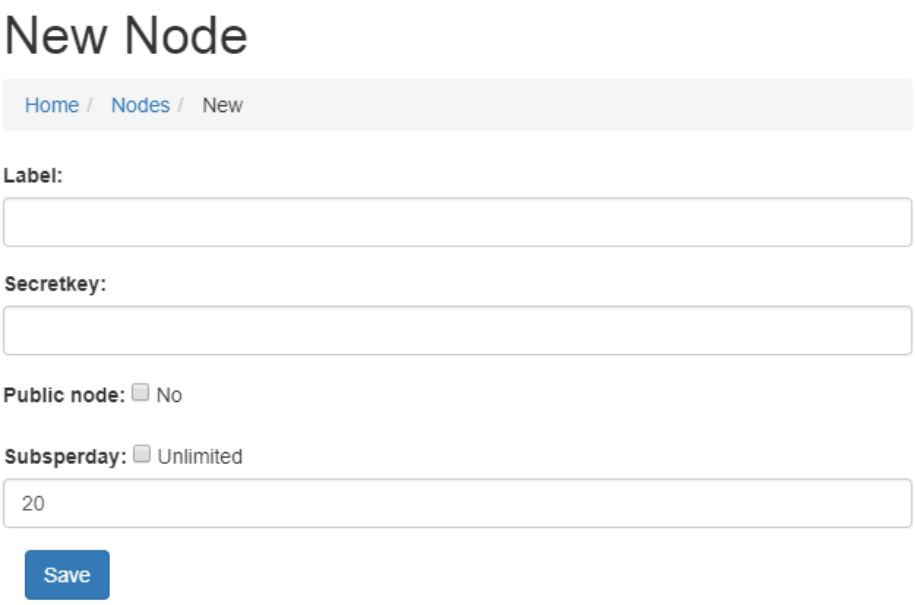
```
32      private setUpNode(node: Node): void {
33          this.node = node;
34          this.is_mine = (node.user ==
35  this.credentialsService.getUser().username) ?
36              true: false;
37          this.links = [
38              { label: "Home", url: "/" },
39              { label: "Nodes", url: "/nodes/" },
40              { label: this.node.label, url: "/" , is_active: true}
41          ];
42      }
43  }
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3-6, deklarasi component `NodeDetailComponent` yang mengimplementasi interface `OnInit`.
2) Baris 7, deklarasi variabel `node` untuk menampung *object* perangkat IoT yang nantinya didapatkan dari web service.
3) Baris 8, deklarasi variabel `links` untuk menampung navigasi URL yang tersedia.
4) Baris 9, deklarasi variabel `is_mine` bertipe data *Boolean*. Nilai dari variabel ini akan menentukan ditampilkan atau tidaknya *button* Edit. Jika perangkat IoT tersebut merupakan milik pengguna yang sedang terautentikasi, 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 IoT dari *web service*.
8) Baris 23, *object* perangkat IoT yang diterima dijadikan parameter untuk method `setUpNode()` .
9) Baris 32-42, deklarasi *method* `setUpNode()` untuk memberikan nilai terhadap variabel `node`, `links`, dan `is_mine`.

c. Buat Perangkat IoT

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



Gambar 5.4 Implementasi aplikasi web *console*: buat perangkat IoT

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

```
1   ...
2
3   @Component({
4       ...
5   })
6   export class NodeNewComponent {
7       ...
8       links: any[];
9       node: Node = new Node;
10      unlimited: boolean;
11
12      errors: Array<{ field: string, message: string}>;
13
14      constructor(
15          private nodeService: NodeService,
16          private router: Router
17      ) {}
18
19      ngOnInit() {
20          this.links = [
21              { label: "Home", url: "/" },
22              { label: "Nodes", url: "/nodes/" },
23              { label: "New", is_active: true }
24          ];
25      }
26
27      unlimitedStateChange(): void {
28          if (this.unlimited) {
29              this.node.pubsperday = -1;
30          } else {
31              this.node.pubsperday = 0;
32          }
33      }
```

```
34
35      save(): void {
36          this.node.is_public = this.node.is_public ? 1 : 0;
37          this.nodeService.save(this.node)
38              .subscribe(
39                  node => this.router.navigate(['/nodes']),
40                  error => this.extractErrors(error)
41              );
42      }
43
44      private extractErrors(err: any): void {
45          let errorsParse = JSON.parse(err._body);
46          this.errors = [];
47          for(let index in errorsParse) {
48              if(errorsParse.hasOwnProperty(index)) {
49                  this.errors.push({
50                      field: index,
51                      message: errorsParse[index]
52                  })
53              }
54          }
55      }
56
57      public closeAlert(alert: any) {
58          const index: number = this.errors.indexOf(alert);
59          this.errors.splice(index, 1);
60      }
61  }
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3-6, deklarasi component `NodeNewComponent` yang mengimplementasi interface `OnInit`.
2) Baris 8, deklarasi variabel `links` untuk menampung navigasi URL yang tersedia.
3) Baris 9, deklarasi variabel `node` untuk menampung data dari form input.
4) Baris 10, deklarasi variabel `unlimited` yang digunakan oleh form input bertipe *checkbox*. Nilai dari variabel ini nantinya akan dikonversi ke dalam bilangan *integer* yang menentukan pembatasan pengiriman dari perangkat IoT yang dibuat.
5) Baris 12, deklarasi variabel `errors` untuk menampung pesan kegagalan validasi dalam pembuatan perangkat IoT.
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 IoT baru ke *web service* untuk disimpan.

13) Baris 39, jika pembuatan perangkat IoT berhasil, pengguna akan diarahkan ke halaman lihat semua perangkat IoT.

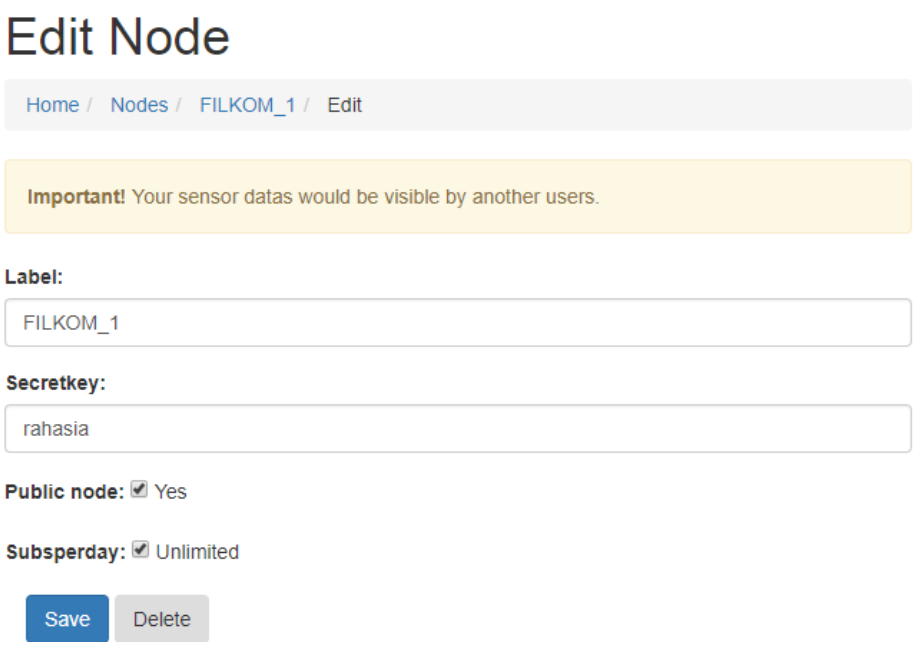
14) Baris 40, jika pembuatan perangkat IoT tidak berhasil, pesan error yang diterima dijadikan parameter untuk method `extractErrors()`.

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.

d. Uban Perangkat IoT

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



Gambar 5.5 Implementasi aplikasi web *console*: ubah perangkat IoT

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

```
1   ...
2
3   @Component({
4       ...
5   })
6   export class NodeEditComponent {
7       links: any[];
8       node: Node;
9       unlimited: boolean;
10      _initial_subsperday: number;
11
12      errors: Error[];
13
14      constructor(
15          private nodeService: NodeService,
16          private route: ActivatedRoute,
17          private credentialsService: CredentialsService,
18          private router: Router
19      ) {}
20
21      ngOnInit() {
22          this.route.params
23              .switchMap((params: Params) =>
24  this.nodeService.getNode(params['id']))
25              .subscribe(
26                  node => this.setUpNode(node),
27                  error => console.log(error)
28              );
29          this.node = new Node;
30      }
31
32      private setUpNode(node: Node): void {
```

```
33          // raise 403 when node is not owned by this auth user
34          if (node.user !=
35  this.credentialsService.getUser().username) {
36              this.router.navigateByUrl('/403', { skipLocationChange:
37  true });
38          }
39          this.node = node;
40          this.unlimited = (-1 == node.subsperday);
41          this._initial_pubsperday = node.subsperday;
42          this.links = [
43              { label: "Home", url: "/" },
44              { label: "Nodes", url: "/nodes/" },
45              { label: this.node.label, url: `/nodes/view/${node.id}`
46  },
47              { label: "Edit", is_active: true }
48          ];
49      }
50
51      unlimitedStateChange(): void {
52          if (this.unlimited) {
53              this.node.pubsperday = -1;
54          } else {
55              this.node.pubsperday = (-1 == this._initial_subsperday)
56  ? 0 :
57                  this._initial_subsperday;
58          }
59      }
60
61      save(): void {
62          this.node.is_public = this.node.is_public ? 1 : 0;
63          this.nodeService.save(this.node)
64              .subscribe(
65                  node => this.router.navigate(['/nodes/view',
66  node.id]),
67                  error => this.extractErrors(error)
68              );
69      }
70
71      delete(): void {
72          if(confirm("Are you sure?")) {
73              this.nodeService.delete(this.node.url)
74                  .subscribe(
75                      () => this.router.navigate(['/nodes/']),
76                      error => this.extractErrors(error)
77                  );
78          }
79      }
80
81      private extractErrors(err: any): void {
82          let errorsParse = JSON.parse(err._body);
83          this.errors = [];
84          for(let index in errorsParse) {
85              if(errorsParse.hasOwnProperty(index)) {
86                  this.errors.push({
87                      field: index,
88                      message: errorsParse[index]
89                  })
90              }
91          }
92      }
```

```
93
94      public closeAlert(alert: any) {
95          const index: number = this.errors.indexOf(alert);
96          this.errors.splice(index, 1);
97      }
98  }
```

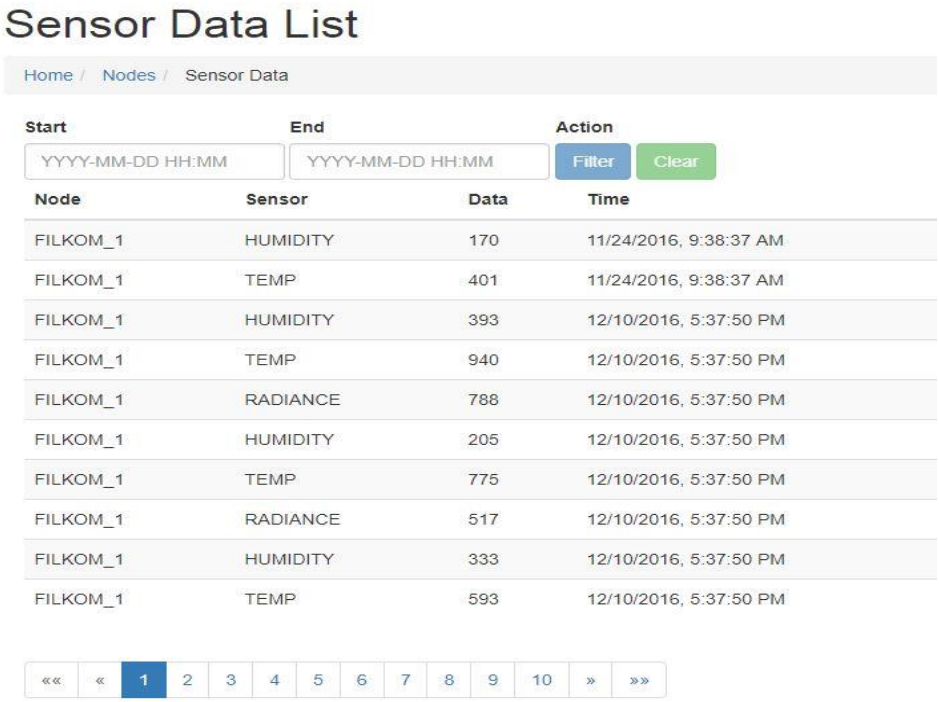
Penjelasan dari potongan kode program di atas adalah:

1) Baris 3-6, deklarasi component `NodeEditComponent` yang mengimplementasi interface `OnInit`.
2) Baris 7, deklarasi variabel `node` untuk menampung data dari form input.
3) Baris 8, deklarasi variabel `links` untuk menampung navigasi URL yang tersedia.
4) Baris 9, deklarasi variabel `unlimited` yang digunakan oleh form input bertipe *checkbox*. Nilai dari variabel ini nantinya akan dikonversi ke dalam bilangan *integer* yang menentukan pembatasan pengiriman dari perangkat IoT yang dibuat.
5) Baris 10, deklarasi variabel `_initial_pubsperday` untuk menyimpan nilai pembatasan pengiriman dari perangkat IoT sebelum dilakukan perubahan.
6) Baris 12, deklarasi variabel `errors` untuk menampung pesan kegagalan validasi dalam pembuatan perangkat IoT.
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 IoT dari *web service*.
10) Baris 26, *object* perangkat IoT yang diterima dijadikan parameter untuk method `setUpNode()` .
11) Baris 27, memberikan nilai awal dari variabel `node` dengan *object* perangkat IoT. 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 IoT tersebut bukan merupakan milik pengguna yang sedang terautentikasi, 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 IoT baru ke *web service* untuk disimpan.
18) Baris 65-66, jika pembuatan perangkat IoT berhasil, pengguna akan diarahkan ke halaman detail dari perangkat IoT yang baru dibuat tersebut.
19) Baris 67, jika pembuatan perangkat IoT 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 IoT 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.

e. Lihat Data Sensor

Halaman ini berisi sensor data yang dapat di-filter berdasarkan rentang waktu tertentu. Hasil akhir dari implementasi lihat data sensor 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`.

```
...

@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 }
        ];
```

```
30      }
31
32      getSensorData(): void {
33          this.sensorDataService.getSensorDataByUser(this.page,
34  this.date_start, this.date_end)
35              .subscribe(
36                  sensordatas => {
37                      this.collectionSize = sensordatas.count;
38                      this.sensordatas = sensordatas.results as
39  SensorData[];
40                  },
41                  error => console.log(error)
42              );
43      }
44
45      pageChange(): void {
46          this.router.navigateByUrl(`sensordata?page=${this.page}`);
47          this.getSensorData();
48      }
49
50      filter(): void {
51          this.page = 1;
52          this.pageChange();
53      }
54
55      clearFilter() {
56          this.page = 1;
57          this.date_start = "";
58          this.date_end = "";
59          this.pageChange();
60      }
61  }
```

Penjelasan dari potongan kode program di atas adalah:

1) Baris 3-6, deklarasi component `SensorDataComponent` yang mengimplementasi interface `OnInit`.
2) Baris 7, deklarasi variabel `sensordatas` untuk menampung *object* data sensor yang nantinya didapatkan dari web service.
3) Baris 8, deklarasi variabel `links` untuk 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.

BAB 6 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.

6.1 Perancangan Pengujian

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

6.1.1 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 IoT.	1. Perangkat IoT mengirimkan request HTTP POST dengan JSON payload: user, label, dan secretkey. 2. Sistem akan mengembalikan HTTP status 200 beserta token akses JWT untuk perangkat IoT.
PF_002	Pengujian membuat token akses JWT bagi *web client*.	1. *Web client* mengirimkan request HTTP POST dengan JSON payload: username dan password. 2. Sistem akan mengembalikan HTTP status 200 beserta token akses JWT untuk *web client.*
PF_003	Pengujian registrasi pengguna.	1. *Web 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 IoT mengirimkan data sensor ke IoT cloud platform untuk disimpan.	1. Perangkat IoT 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.

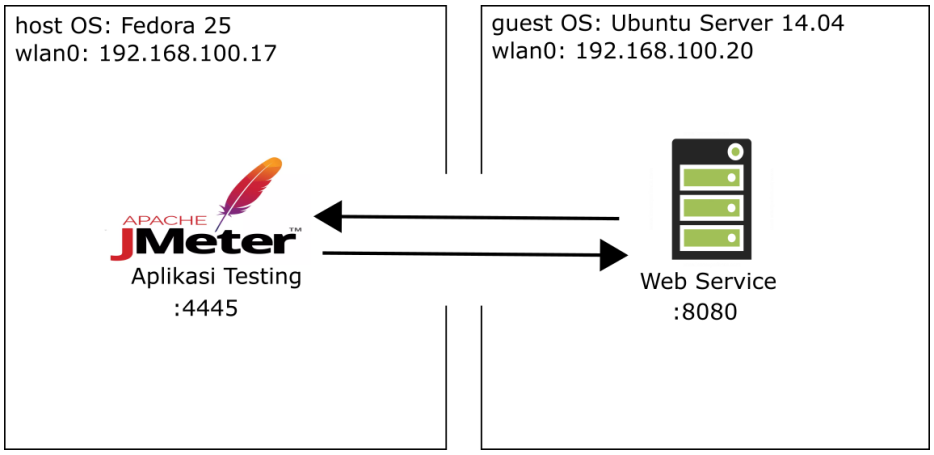
PF_005	Pengujian mengakses data sensor.	a) Menguji kriteria berdasarkan pengguna 1. *Web client* mengirimkan request HTTP GET. 2. Sistem akan mengembalikan HTTP status 200 beserta data sensor. b) Menguji kriteria berdasarkan perangkat IoT 1. *Web client* mengirimkan request HTTP GET. 2. Sistem akan mengembalikan HTTP status 200 beserta data sensor. c) Menguji kriteria berdasarkan sensor 3. *Web client* mengirimkan request HTTP GET. 1. Sistem akan mengembalikan HTTP status 200 beserta data sensor.
PF_006	Pengujian membuat perangkat IoT.	1. *Web client* mengirimkan request HTTP POST dengan JSON payload: user, label, is_public, secretkey dan pubsperday. 2. Sistem akan menyimpan data perangkat IoT ke dalam basis data. 3. Sistem akan mengembalikan HTTP status 200.
PF_007	Pengujian melihat perangkat IoT.	1. *Web client* mengirimkan request HTTP GET. 2. Sistem akan mengembalikan HTTP status 200 beserta semua perangkat IoT.
PF_008	Pengujian melihat perangkat IoT berdasarkan id.	1. *Web client* mengirimkan request HTTP GET. 2. Sistem akan mengembalikan HTTP status 200 beserta perangkat IoT.

PF_009	Pengujian mengubah data perangkat IoT.	1. *Web client* mengirimkan request HTTP POST dengan JSON payload berisi hanya field yang ingin diubah. 2. Sistem akan menyimpan perubahan data perangkat IoT ke dalam basis data. 3. Sistem akan mengembalikan HTTP status 200.
PF_010	Pengujian menghapus perangkat IoT.	1. *Web client* mengirimkan request HTTP DELETE. 2. Sistem akan menghapus perangkat IoT di dalam basis data. 3. Sistem akan mengembalikan HTTP status 204.
PF_011	Pengujian Autentikasi	a) Pengujian perangkat IoT mengirimkan data sensor tanpa token akses JWT 1. Perangkat IoT mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish ke alamat URL /sensordatas. 2. Sistem akan mengembalikan HTTP status 401 beserta pesan "Authentication credentials were not provided." b) Pengujian *web client* mengakses data sensor tanpa token akses JWT 3. *Web client* mengirimkan request HTTP GET ke alamat URL /sensordatas/user/<user-username>. 1. Sistem akan mengembalikan HTTP status 401 beserta pesan "Authentication credentials were not provided."
PF_012	Pengujian Otorisasi	a) Pengujian perangkat IoT mengirimkan data sensor menggunakan token akses JWT milik *web client*

		1. Perangkat IoT mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish ke alamat URL /sensordatas. 2. Sistem akan mengembalikan HTTP status 403 beserta pesan "You do not have permission to perform this action." b) Pengujian *web client* mengakses data sensor menggunakan token akses JWT milik perangkat IoT 1. *Web client* mengirimkan request HTTP GET ke alamat URL /sensordatas/user/<user-username>. 2. Sistem akan mengembalikan HTTP status 403 beserta pesan "You do not have permission to perform this action." c) Pengujian mengirim data sensor oleh perangkat IoT yang telah habis batas pengiriman per harinya 1. Perangkat IoT mengirimkan request HTTP POST dengan JSON payload: user, node, dan publish ke alamat URL /sensordatas. 2. Sistem akan mengembalikan HTTP status 403 beserta pesan "Publish is limit."

6.1.2 Perancangan Pengujian Performa Sistem

Pengujian performa sistem dilakukan untuk mengetahui kinerja dari IoT *cloud platform* yang diajukan. Kinerja tersebut berkaitan dengan protokol komunikasi yang digunakan, yaitu HTTP. Pengujian dilakukan dengan mengamati proses komunikasi antar entitas menggunakan beberapa nilai parameter yang ditentukan seperti skalabilitas, *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 performa 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 performa 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 lokal 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, di mana gateway dari jaringan tersebut berada pada alamat 192.168.100.1. Fitur yang dilakukan pengujian performa 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 IoT 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 IoT 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 *web client* dengan menggunakan token akses JWT milik pengguna dengan username subali. 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 GET ke alamat `/sensordatas/user/subali/`. 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 performa sistem dilakukan proses analisis dengan membandingkan beberapa parameter uji terkait performa sistem yaitu skalabilitas, *latency* dan *error rate*.

6.1.3 Perancangan Pengujian *Overhead*

Pengujian *overhead* dilakukan untuk mengetahui kerugian yang ditimbulkan dari proses autentikasi dan otorisasi menggunakan JSON Web Token. Kerugian yang dimaksud adalah besaran *Bytes* tambahan pada HTTP *header* yang diperlukan untuk mengirimkan token. Pengujian dilakukan terhadap dua metode HTTP yaitu GET dan POST. Di mana, masing-masing metode tersebut dilakukan uji *overhead* menggunakan dua skenario: dengan dan tanpa token. Pengujian pada metode HTTP GET menggunakan fitur mengakses data sensor dan metode HTTP POST menggunakan fitur mengirimkan data sensor.

Spesifikasi pesan HTTP pada pengujian *overhead* pada metode HTTP GET adalah:

- Skenario dengan JSON Web Token

```
GET /sensordatas/user/subali/ HTTP/1.1
Host: 127.0.0.1:8080
Authorization: JWT <subali_token>
```

- Skenario tanpa JSON Web Token

```
GET /sensordatas/user/subali/ HTTP/1.1
Host: 127.0.0.1:8080
```

Spesifikasi pesan HTTP pada pengujian *overhead* pada metode HTTP POST adalah:

- Skenario dengan JSON Web Token

```
POST /sensordatas/ HTTP/1.1
Host: 127.0.0.1:8080
Authorization: JWT <FILKOM_1 token>
Content-Type: application/json

{
  "publish": [
      {
        "sensor": "HUMIDITY",
        "data": "209",
        "timestamp": ""
      },
      {
        "sensor": "TEMP",
        "data": "48"
      }
  ]
}
```

- Skenario tanpa JSON Web Token

```
POST /sensordatas/ HTTP/1.1
Host: 127.0.0.1:8080
Content-Type: application/json

{
  "publish": [
      {
        "sensor": "HUMIDITY",
        "data": "209",
        "timestamp": ""
      },
      {
        "sensor": "TEMP",
        "data": "48"
      }
  ]
}
```

6.2 Hasil dan Analisis Pengujian

Pengujian terhadap sistem dilakukan berdasarkan skenario-skenario dari perancangan pengujian di atas. Hasil dari pengujian tersebut kemudian diolah dan dianalisis untuk mengetahui apakah telah sesuai dengan tahap kebutuhan dan

perancangan sebelumnya, memiliki otorisasi dan autentikasi yang sesuai, dan performa yang diharapkan.

6.2.1 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 IoT.	• Sistem mengembalikan HTTP status 200 beserta token akses JWT untuk perangkat IoT.	valid
PF_002	Pengujian membuat token akses JWT bagi *web client*.	• Sistem akan mengembalikan HTTP status 200 beserta token akses JWT untuk *web 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 IoT 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 IoT.	• Sistem menyimpan data perangkat IoT ke dalam basis data • Sistem mengembalikan HTTP status 200	valid
PF_007	Pengujian melihat perangkat IoT.	• Sistem mengembalikan HTTP status 200 beserta semua perangkat IoT	valid
PF_008	Pengujian melihat perangkat IoT berdasarkan id.	• Sistem mengembalikan HTTP status 200 beserta perangkat IoT	valid
PF_009	Pengujian mengubah data perangkat IoT.	• Sistem menyimpan perubahan data perangkat IoT ke dalam basis data • Sistem mengembalikan HTTP status 200	valid
PF_010	Pengujian menghapus perangkat IoT.	• Sistem menghapus perangkat IoT di dalam basis data • Sistem mengembalikan HTTP status 204	valid
PF_011	Pengujian autentikasi.	• 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 di atas 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 IoT dan *web 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 autentikasi 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 IoT dan *web client* terhadap *web service* dapat terhubung dan saling berkomunikasi dengan baik melalui protokol komunikasi HTTP.

6.2.2 Hasil dan Analisis Pengujian Performa Sistem

Bagian ini membahas mengenai hasil dan analisis dari pengujian performa sistem. Pengujian dilakukan berdasarkan rancangan topologi dan skenario uji yang telah ditentukan pada sub bab 6.1.2. Masing-masing skenario dijalankan sebanyak 3 kali dengan menggunakan aplikasi pengujian JMeter. Fitur yang dilakukan pengujian performa 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 IoT 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* sama dengan satu pengguna) 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

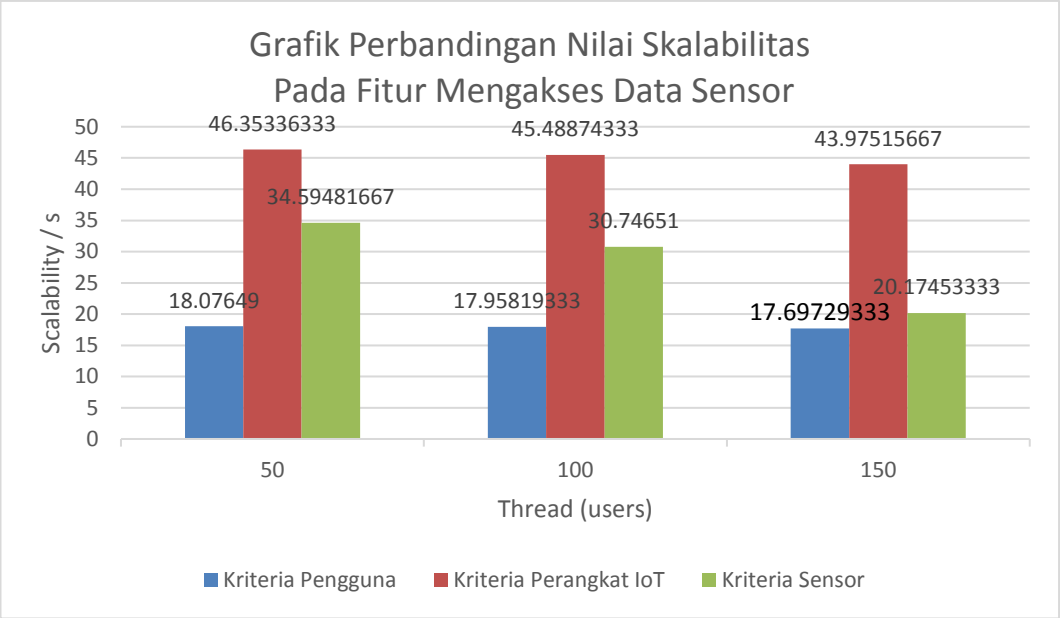
Thread (users)	Expected	Actual	Success rate %	Request Rejected rate %	Scalability / s	Latency (ms)
50	150	150	100%	0%	18.07649	1777.753333
100	300	300	100%	0%	17.95819333	1800.44
150	450	441	97.92%	2.08%	17.69729333	1861.106667

Tabel 6.4 Hasil pengujian performa dalam mengakses data sensor berdasarkan perangkat IoT

Thread (users)	Expected	Actual	Success rate %	Request Rejected rate %	Scalability / 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

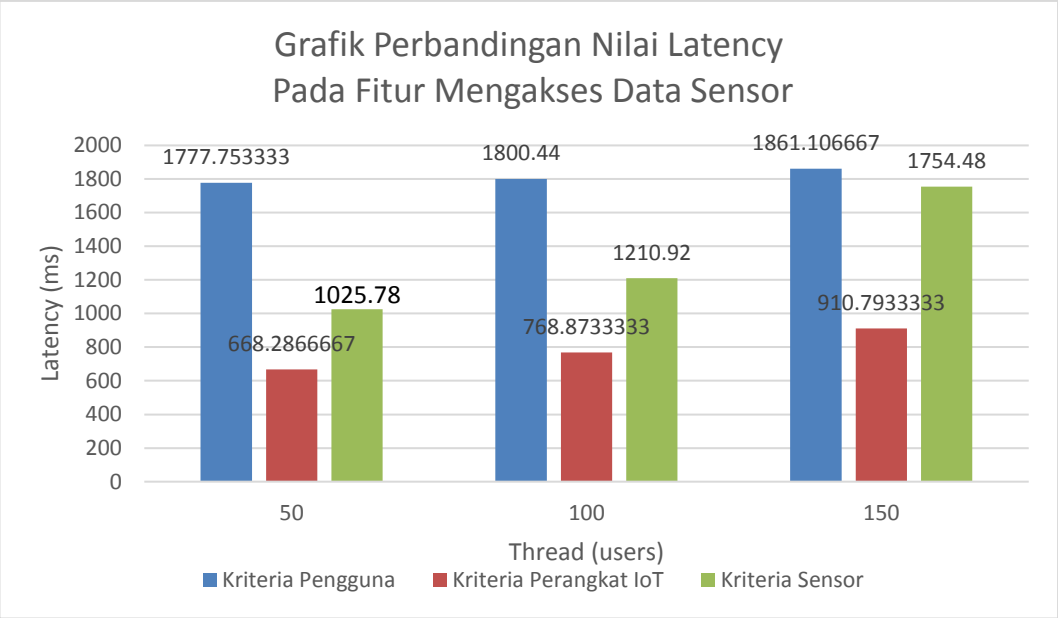
Tabel 6.5 Hasil pengujian performa dalam mengakses data sensor berdasarkan sensor

Thread (users)	Expected	Actual	Success rate %	Request Rejected rate %	Scalability / 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 *skalabilitas* pada fitur mengakses data sensor

Perbandingan nilai skalabilitas diantara ketiga kriteria dalam mengakses data sensor dapat memberitahukan berapa banyak *request* yang dapat ditangani sistem setiap detiknya. Berdasarkan grafik perbandingan nilai skalabilitas pada Gambar 6.2 di atas, terlihat bahwa mengakses data sensor menggunakan kriteria berdasarkan sensor menghasilkan skalabilitas terendah dan kriteria berdasarkan perangkat IoT menghasilkan skalabilitas 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 IoT yang dimaksud dan kueri untuk melakukan filter data sensor berdasarkan *list* sensor. Sedangkan untuk kriteria berdasarkan perangkat IoT hanya diperlukan satu kueri standar, yaitu melakukan filter data sensor berdasarkan id perangkat IoT yang terdapat dalam URL. Hal tersebut menjadikan eksekusi kueri untuk kriteria berdasarkan perangkat IoT menjadi jauh lebih cepat.



Gambar 6.3 Grafik perbandingan nilai *latency* pada fitur mengakses data sensor

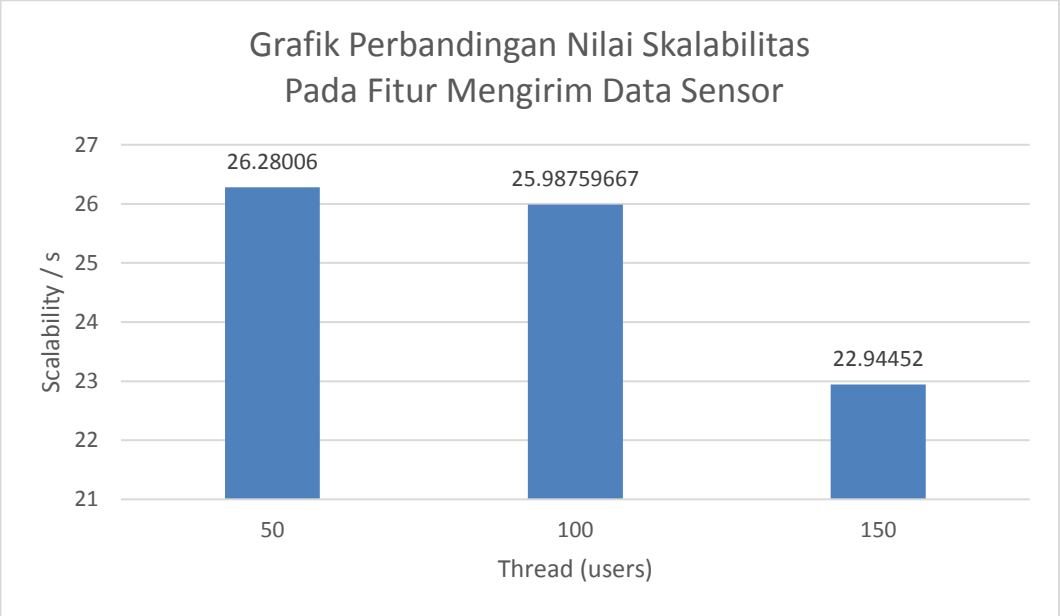
Latency merupakan nilai jeda yang dibutuhkan untuk mengirimkan pesan dari pengirim ke penerima dalam jaringan komputer. Pada kasus ini *latency* yang dihitung adalah jeda waktu dari *web service* ke aplikasi pengujian JMeter ketika memberikan *response* berisi data sensor yang diminta. Perbandingan nilai *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 IoT, 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* meningkat cukup signifikan menjadi 1861.10/ms, 910.79/ms dan 1754.48/ms.

2. Pengujian Fitur Mengirim Data Sensor

Pengujian performa 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 pengguna) berjumlah 50, 100 dan 150 secara bersamaan, sistem dapat menangani permintaan tersebut dengan keberhasilan 100%.

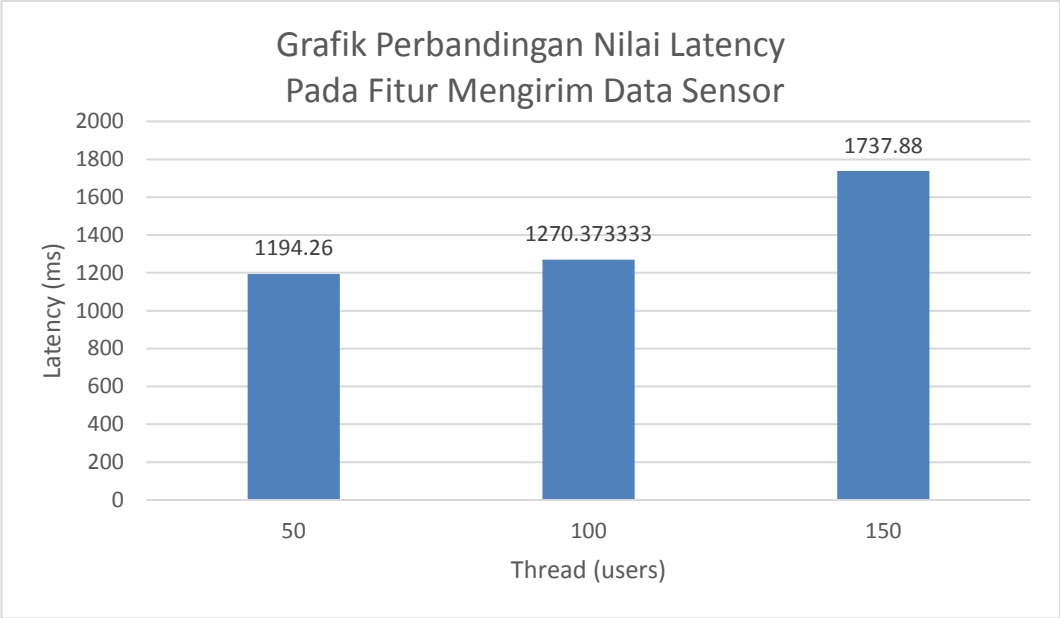
Tabel 6.6 Hasil pengujian performa dalam mengirim data sensor

Thread (users)	Expected	Actual	Success rate %	Request Rejected rate %	Scalability / 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 skalabilitas pada fitur mengirim data sensor

Perbandingan nilai skalabilitas dari fitur mengirim data sensor dapat dilihat pada Gambar 6.4. Perbandingan tersebut dapat memberitahu berapa banyak pengguna yang dapat ditangani sistem setiap detiknya. Berdasarkan gambar tersebut, didapati bahwa ketika *request* yang dikirimkan oleh 50 pengguna secara bersamaan, nilai skalabilitas yang didapat adalah 26.28/detik. Ketika pengguna berjumlah 100, skalabilitas menurun menjadi 25.98/detik karena jumlah *request* yang dikirimkan lebih besar dari sebelumnya. Begitu juga hasil yang didapat dari jumlah pengguna sebanyak 150, nilai skalabilitas 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 pengguna berjumlah 50 secara bersamaan, *latency* yang didapat adalah 1194.26/ms. Ketika *request* yang dikirimkan berjumlah 100 secara bersamaan, skalabilitas yang didapat meningkat menjadi 1270.37/ms dikarenakan semakin banyak *request* yang dikirimkan semakin lama pula jeda waktu yang dibutuhkan. Ketika *request* yang dikirim secara bersamaan berjumlah 150, nilai *latency* menurun cukup signifikan menjadi 1737.88/ms.

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

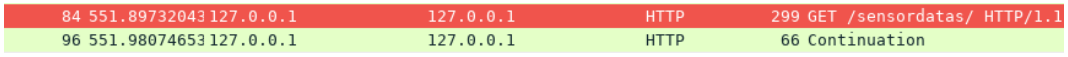
6.2.3 Hasil dan Analisis Pengujian *Overhead*

Bagian ini membahas mengenai hasil dan analisis dari pengujian *overhead* pada ukuran *Bytes* pesan HTTP karena penggunaan JSON Web Token. Pengujian dilakukan berdasarkan rancangan spesifikasi pesan HTTP yang telah ditentukan pada sub bab 6.1.3. Metode HTTP yang dilakukan pengujian *overhead* adalah metode HTTP GET dan HTTP POST yang masing-masing berafiliasi terhadap fitur mengakses data sensor dan mengirimkan data sensor.

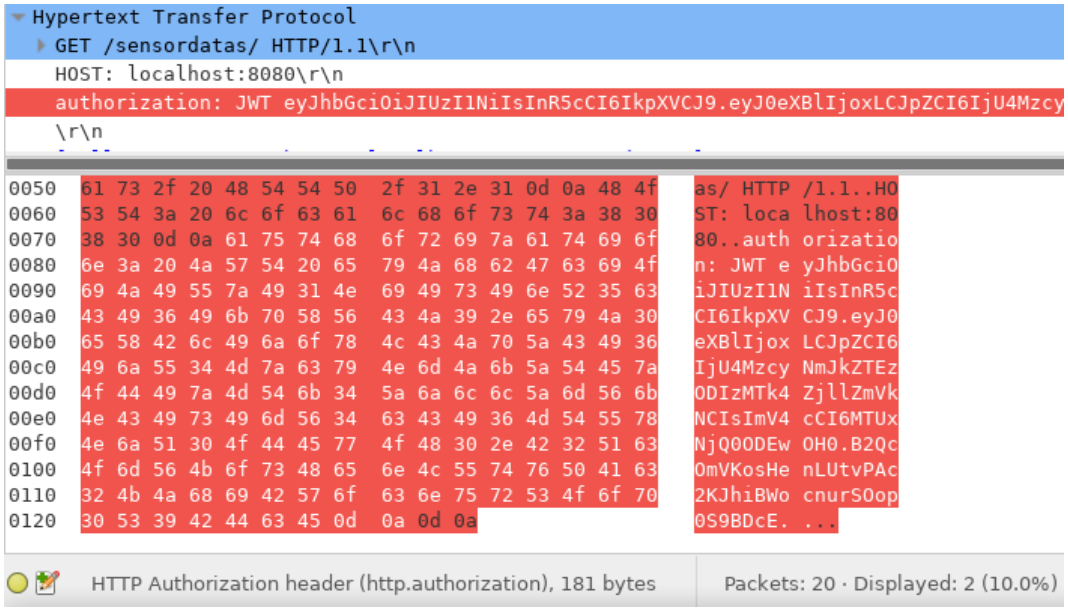
1. Pengujian metode HTTP GET
   - Dengan JSON Web Token

Dengan mengirimkan *request* HTTP GET menggunakan spesifikasi pesan pada sub bab perancangan pengujian sebelumnya, didapati ukuran total pesan HTTP dengan JSON Web Token adalah sebesar 299 *Bytes* seperti pada Gambar 6.6.

Sedangkan, ukuran tambahan *Bytes* pada *header* HTTP untuk JSON Web Token adalah sebesar 181 *Bytes* seperti pada gambar Gambar 6.7.



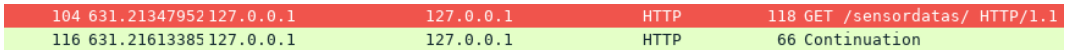
Gambar 6.6 Ukuran total pesan HTTP GET dengan JWT



Gambar 6.7 Ukuran tambahan header pada pesan HTTP GET

- Tanpa JSON Web Token

  Dengan mengirimkan request HTTP GET menggunakan spesifikasi pesan pada sub bab perancangan pengujian sebelumnya, didapati ukuran total pesan HTTP tanpa JWT adalah sebesar 118 *Bytes* seperti pada Gambar 6.8. Ukuran tersebut lebih kecil sebesar 181 *Bytes* jika dibandingkan dengan HTTP GET dengan JWT yang sebesar 299 *Bytes*.

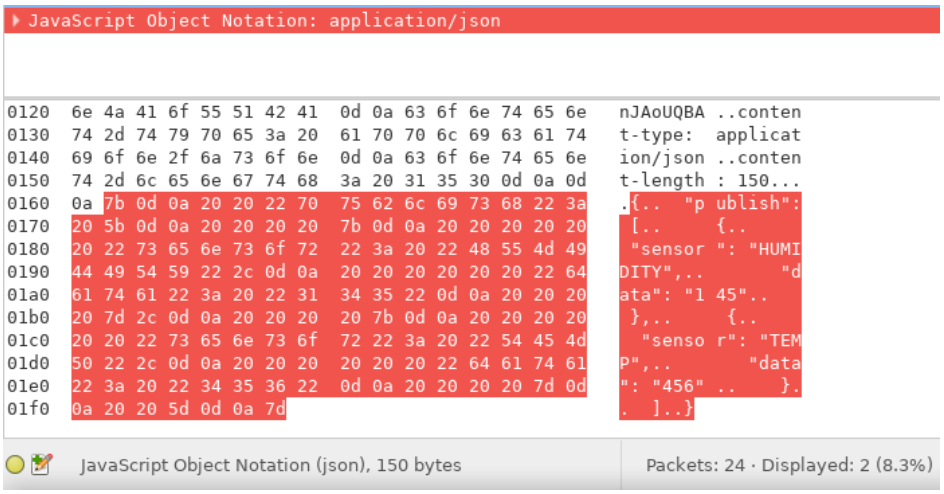


Gambar 6.8 Ukuran total pesan HTTP GET tanpa JWT

Dari hasil pengujian ukuran total *Bytes* pada pesan HTTP GET diatas, didapati bahwa ada kerugian sebesar 181 *Bytes* untuk mengakomodir metode autentikasi dan otorisasi menggunakan JSON Web Token. Bahkan, ukuran *Bytes* tambahan untuk JWT pada Gambar 6.7 sebesar 181 *Bytes* lebih besar dari ukuran pesan HTTP GET pada Gambar 6.8 yaitu skenario tanpa JWT yang hanya sebesar 118 *Bytes*.

2. Pengujian metode HTTP POST

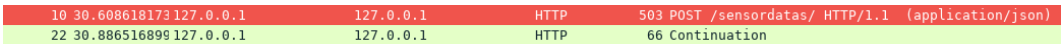
Pada pengujian ini digunakan data *payload* yang sudah didefinisikan sebelumnya berukuran 150 *Bytes* seperti pada Gambar 6.9.



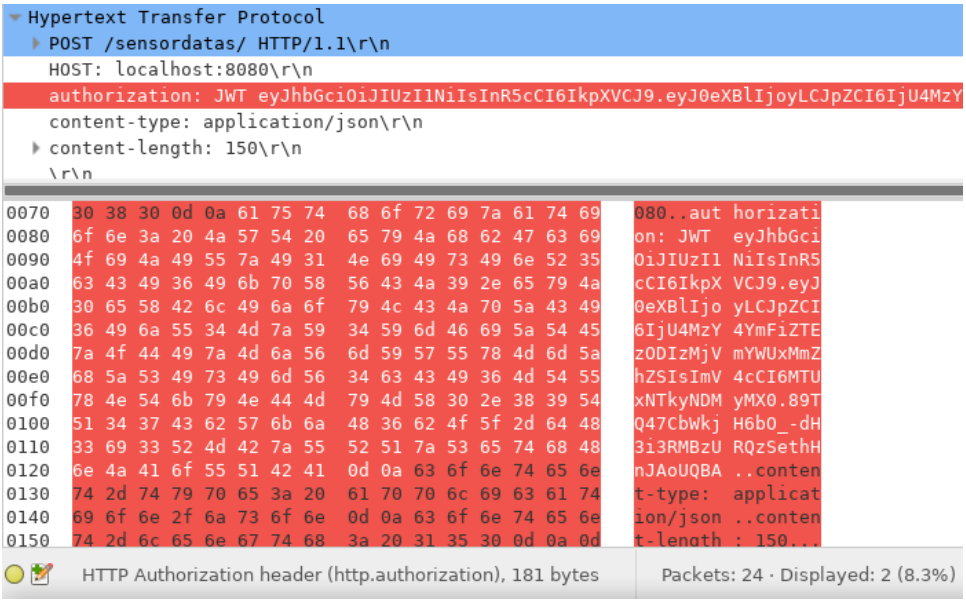
Gambar 6.9 Ukuran data *payload* HTTP POST

- Dengan JSON Web Token

Dengan mengirimkan *request* HTTP POST menggunakan spesifikasi pesan pada sub bab perancangan pengujian sebelumnya, didapati ukuran total pesan HTTP dengan JWT adalah sebesar 503 *Bytes* (termasuk data *payload*) seperti pada Gambar 6.10. Sedangkan, ukuran tambahan *Bytes* pada *header* HTTP untuk JWT adalah sebesar 181 *Bytes* seperti pada gambar Gambar 6.11.



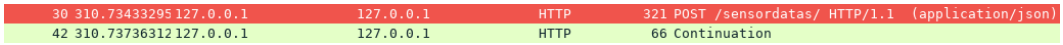
Gambar 6.10 Ukuran total pesan HTTP POST dengan JWT



Gambar 6.11 Ukuran tambahan header pada pesan HTTP POST

- Tanpa JSON Web Token

  Dengan mengirimkan request HTTP POST menggunakan spesifikasi pesan pada sub bab perancangan pengujian sebelumnya, didapati ukuran total pesan HTTP tanpa JWT adalah sebesar 321 *Bytes* (termasuk data *payload*) seperti pada Gambar 6.12. Ukuran tersebut lebih kecil sebesar 181 *Bytes* jika dibandingkan dengan HTTP POST dengan JWT yang sebesar 503 *Bytes*.



Gambar 6.12 Ukuran total pesan HTTP POST tanpa JWT

Kesimpulan dari pengujian *overhead* ini adalah terdapat kerugian header HTTP tambahan sebesar 181 *Bytes* untuk mengakomodir mekanisme autentikasi dan otorisasi menggunakan JWT. Kesimpulan dari pengujian pada metode HTTP GET dan HTTP POST, didapati bahwa ukuran header tambahan tersebut lebih besar daripada ukuran HTTP GET tanpa JWT dan ukuran data *payload* pada HTTP POST. Sehingga diperlukan mekanisme autentikasi dan otorisasi menggunakan teknologi lain yang lebih efisien untuk menekan atau menghapus tambahan *Bytes* pada *header* HTTP.

BAB 7 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.

7.1 Kesimpulan

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

1. 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.
2. Pengiriman data antara perangkat IoT dan *web client* terhadap IoT *cloud platform* harus melalui RESTful *web service* yang menyediakan aturan baku untuk menjalankan fitur-fitur yang disediakan.
3. 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 IoT yang bertindak mengirimkan data sensor dan alplikasi *client* yang bertindak mengakses data sensor dan memanajemen perangkat.
4. Berdasarkan hasil pengujian performa sistem didapati bahwa sistem dapat menangani 100 pengguna secara bersamaan dengan keberhasilan 100%.

7.2 Saran

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

1. Penelitian selanjutnya dapat mengembangkan perangkat *relay* antara perangkat IoT dengan IoT *cloud platform*.
2. IoT *cloud platform* ini dapat dikembangkan lebih lanjut dengan menambahkan komputasi cerdas untuk mengolah data sensor untuk keperluan yang lebih luas.
3. Mekanisme autentikasi dan otorisasi dapat diganti dengan teknologi lain yang lebih efisien dalam hal ukuran *header* HTTP tambahan.
4. 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.

Litoiu, M. (2002). Migrating to Web services-latency and scalability. *In Web Site Evolution*, 13-20.

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.