TUGAS AKHIR - IF184802

IMPLEMENTASI *HEADLESS BROWSER* UNTUK *LOAD TESTING* BERBASIS *WEB SERVICE* MENGGUNAKAN INFRASTRUKTUR *DOCKER*

CAHYA PUTRA HIKMAWAN
NRP 05111540000119

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

(Halaman ini sengaja dikosongkan)

TUGAS AKHIR - IF184802

IMPLEMENTASI *HEADLESS BROWSER* UNTUK *LOAD TESTING* BERBASIS *WEB SERVICE* MENGGUNAKAN INFRASTRUKTUR *DOCKER*

CAHYA PUTRA HIKMAWAN
NRP 05111540000119

Dosen Pembimbing I
Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.

Dosen Pembimbing II
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya, 2019

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS - IF184802

HEADLESS BROWSER IMPLEMENTATION FOR LOAD TESTING BASED ON WEB SERVICE USING DOCKER INFRASTRUCTURE

CAHYA PUTRA HIKMAWAN
NRP 05111540000119

First Advisor
Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.

Second Advisor
Bagus Jati Santoso, S.Kom., Ph.D.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology and Communication
Sepuluh Nopember Institute of Technology
Surabaya, 2019

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI *HEADLESS BROWSER* UNTUK *LOAD TESTING* BERBASIS *WEB SERVICE* MENGGUNAKAN INFRASTRUKTUR *DOCKER*

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Arsitektur dan Jaringan Komputer
Program Studi S1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh :
CAHYA PUTRA HIKMAWAN
NRP: 05111540000119

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD. .....
NIP: 197708242003041001 (Pembimbing 1)

Bagus Jati Santoso, S.Kom., Ph.D. .....
NIP: 198611252018031001 (Pembimbing 2)

SURABAYA
Juni 2019

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI *HEADLESS BROWSER* UNTUK *LOAD TESTING* BERBASIS *WEB SERVICE* MENGGUNAKAN INFRASTRUKTUR *DOCKER*

Nama	:	**CAHYA PUTRA HIKMAWAN**
NRP	:	**05111540000119**
Departemen	:	**Informatika FTIK-ITS**
Pembimbing I	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.**
Pembimbing II	:	**Bagus Jati Santoso, S.Kom., Ph.D.**

Abstrak

Saat ini pengembangan aplikasi berbasis web sangat banyak dilakukan. Dalam pengembangan web, akan diperlukan uji beban yang dilakukan pada web yang dikembangkan sebelum diluncurkan ke tahap produksi. Dilakukannya uji beban tentu saja untuk menghindari terjadinya kegagalan muat ketika sudah diluncurkan.

Dalam melakukan uji beban, pengembang akan membutuhkan resource user untuk mengakses web dan waktu yang cukup lama karena dilakukan secara manual. Beberapa contoh tools yang dapat mengatasi hal tersebut untuk melakukan uji beban yaitu JMeter, k6 dan sebagainya. Namun tools tersebut menggunakan thread untuk resource user-nya, bila dilihat kondisi lapangan yaitu pengguna mengakses web menggunakan browser dan memiliki IP masing-masing, thread kurang bisa mengatasi hal tersebut. Oleh karena itu, dibutuhkan tools yang dapat menangani hal tersebut, salah satunya yaitu pengujian secara headless yang memanfaatkan infrastruktur Docker dan melakukan otomasi untuk pengambilan data uji beban. Docker akan menjadi resource user yang mengakses melalui headless browser untuk uji beban.

Pada tugas akhir ini, dibuat sistem yang mampu melayani uji beban pada suatu web. Dengan menggunakan Docker Container sebagai load generator yang akan bisa melakukan akses ke web yang diuji melalui Headless Chrome dan akan dilakukan otomasi pengambilan data uji beban menggunakan API Puppeteer. Selain itu, sistem juga dilengkapi task scheduler untuk melayani permintaan uji beban dari multiuser. Hasil uji yang didapatkan pada sistem ini adalah beberapa performance metrics, error console pada browser dan tangkapan layar terhadap interface web yang diuji.

Hasil uji coba menunjukkan bahwa Docker Container dapat dijadikan sebagai resource user karena sumberdaya yang dibutuhkan hanya 1,3MB setiap Docker Container dalam keadaan sleep, namun untuk ketika Puppeteer mengakses web yang diuji melalui Headless Chrome membutuhkan sumberdaya CPU dan RAM yang cukup tinggi jika dijalankan secara bersama-sama. Maka dari itu, dibutuhkan pembatasan atau manajemen proses untuk Headless Chrome. Sedangkan untuk data uji beban yang dihasilkan dari Headless Chrome sangat memuaskan.

***Kata-Kunci**: headless browser, headless chrome, load test, puppeteer, docker*

HEADLESS BROWSER IMPLEMENTATION FOR LOAD TESTING BASED ON WEB SERVICE USING DOCKER INFRASTRUCTURE

Name	:	**CAHYA PUTRA HIKMAWAN**
NRP	:	**05111540000119**
Department	:	**Informatics FTIK-ITS**
First Advisor	:	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD.**
Second Advisor	:	**Bagus Jati Santoso, S.Kom., Ph.D.**

Abstract

Nowadays web-based application development is very much done. In web development, a load test will be needed on the web that was developed before being launched into the production stage. The load test is carried out of course to avoid loading failure when it is launched.

In carrying out load tests, developers will need a resource user to access the web and a considerable amount of time because it is done manually. Some examples of tools that can overcome this are to carry out load tests namely JMeter, k6 and so on. However, these tools use threads for their resource users, if you see field conditions, that is, users access the web using a browser and have their own IP, the thread is less able to overcome this. Therefore, we need tools that can handle this, one of which is headless testing that utilizes Docker infrastructure and automates the load test data collection. Docker will be a resource user who accesses through the headless browser for load testing.

In this final project, a system that is capable of serving load tests on a web is made. By using Docker Container as a load generator that will be able to access the web tested through

Headless Chrome and automation of load test data retrieval will be carried out using the Puppeteer API. In addition, the system also features a task scheduler to serve multiuser load test requests. The test results obtained in this system are several performance metrics, browser console errors and screenshots of the tested web interface.

The trial results show that Docker Container can be used as a resource user because the resources needed are only 1.3MB per Docker Container in a sleep state, but for when Puppeteer accesses the web tested through Headless Chrome it requires high CPU and RAM resources if run together -same. Therefore, restrictions or process management are needed for Headless Chrome. Whereas for load test data generated from Headless Chrome is very satisfying.

***Keywords**: headless browser, headless chrome, load test, puppeteer, docker*

KATA PENGANTAR

Bismillahirrahmanirrahim,

Alhamdulillahirabbil'alamin, segala puji bagi Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul "**Implementasi *Headless Browser* untuk *Load Testing* Berbasis *Web Service* Menggunakan Infrastruktur *Docker***". Pengerjaan Tugas Akhir ini merupakan kesempatan yang baik bagi penulis untuk belajar lebih banyak, serta memperdalam apa yang telah didapatkan penulis selama menempuh perkuliahan di Informatika ITS. Dengan adanya Tugas Akhir ini, penulis juga dapat menghasilkan suatu implementasi dari apa yang penulis pelajari. Sehingga penulis mengharapkan implementasi Tugas Akhir ini dapat memberikan kontribusi dan manfaat. Selesainya Tugas Akhir ini tidak lepas dari bantuan dan dukungan dari beberapa pihak baik langsung maupun tidak langsung. Sehingga pada kesempatan ini penulis mengucapkan terima kasih kepada:

1. Allah SWT atas anugerah-Nya yang tidak terkira dan Nabi Muhammad SAW.
2. Ibu dan Ayah yang selalu memberikan doa yang tak terhingga, dukungan moral dan material selama penulis hidup. Serta selalu memberi semangat dan dorongan untuk menyelesaikan Tugas Akhir ini.
3. Bapak Royyana Muslim Ijtihadie, S.Kom, M.Kom, Ph.D., selaku pembimbing I dan Bapak Bagus Jati Santoso, S.Kom., Ph.D., selaku dosen pembimbing II yang telah membantu, membimbing dan memotivasi penulis mulai dari pengerjaan proposal hingga terselesaikannya Tugas Akhir ini.
4. Bapak Darlis Herumurti, S.Kom., M.Kom., selaku Kepala Jurusan Teknik Informatika ITS saat ini, Bapak Radityo Anggoro, S.Kom., M.Sc., selaku koordinator TA dan segenap dosen dan karyawan Teknik Informatika yang telah memberikan ilmu dan pengalamannya, serta memberikan fasilitas yang aman dan nyaman, sehingga

penulis dapat menempuh studi di Informatika ITS.

5. Seluruh teman-teman Laboratorium Arsitektur dan Jaringan Komputer (AJK): Putol, Sinyo, Penyok, Satria, Nahda, Hana, Daus, Aguel, Raldo, Tamtam, Khawari, Haura, Lia, Sulton, Fawwaz, Yoga dan Mail yang bersedia direpotkan, merepotkan dan menemani penulis dalam mengerjakan Tugas Akhir.
6. Alumni-alumni AJK, Mas Daniel, Mas Uul, Mas Wicak, Asbun, Toni, Pak Lek, Mas Fatih, Mbak Nindy dan Mbak Zaza yang selalu membuat saya termotivasi untuk lulus dan memberikan arahan dalam pengerjaan Tugas Akhir ini.
7. Rozana Firdausi yang selalu merepotkan, direpotkan, menemani dan memberikan semangat secara langsung dan tidak langsung, sehingga penulis dapat mengerjakan Tugas Akhir ini dengan baik.
8. Tim Proyek Arya Fajar Production, Pak Arya Yudhi Wijaya, S.Kom, M.Kom., Pak Fajar Baskoro, S.Kom., MT., Pak Ary Mazharuddin S., S.Kom., M.Comp.Sc., Mas Jumali, Tayar, Sinyo, Fajri, Adit, Irsa, Tamtam, Fasma, Jonathan, Nila, Andhika dan Fawwaz yang telah merepotkan dan memberikan pengalaman yang berharga bagi penulis dalam hal lapangan pekerjaan secara nyata.
9. Grup Sayang: Sinyo, Yoza, Tayar, Fajri, Yayan, Alvin, Bas, Putol, Chasni, Raca dan reza yang telah memberikan secuil motivasi dan memberikan dampak yang baik secara jasmani maupun secara rohani penulis.
10. Teman-teman satu bimbingan, Hero, Didin, Satria dan Nahda yang telah sama-sama berjuang dalam mengerjakan dan mengimplementasikan Tugas Akhir tentang Sistem Terdistribusi dan Komputasi Awan.
11. Teman-teman angkatan 2015 yang menemani dari awal hingga akhir serta semua pihak yang turut membantu penulis dalam menyelesaikan Tugas Akhir ini.

Penulis sangat menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga penulis memohon maaf secara tulus kepada pembaca. Dengan kerendahan hati, penulis mengharapkan kritik dan saran yang membangun dari pembaca sebagai pembelajaran dan perbaikan untuk kedepannya.

Surabaya, Juni 2019

Cahya Putra Hikmawan

(Halaman ini sengaja dikosongkan)

DAFTAR ISI

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

DAFTAR GAMBAR

(Halaman ini sengaja dikosongkan)

DAFTAR KODE SUMBER

(Halaman ini sengaja dikosongkan)

BAB I

PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1 Latar Belakang

Saat ini pengembangan aplikasi berbasis web sangat banyak dilakukan. Dalam pengembangan web, akan diperlukan uji beban yang dilakukan pada web yang dikembangkan sebelum diluncurkan ke tahap produksi. Salah satu teknik uji beban adalah *Headless Testing*, salah satu *browser* yang menggunakan teknik ini adalah *Headless Chrome*. Teknik ini menyediakan akses kontrol seperti *browser* pada umumnya untuk mendapatkan data dokumen uji beban, hanya saja teknik ini berjalan secara *headless* atau tanpa menampilkan antarmuka pengguna. Dikarenakan berjalan secara *headless*, maka untuk melakukan pengambilan data dokumen uji beban dilakukan menggunakan baris perintah atau *CLI(Command Line Interface)*.

Pengembang aplikasi web tentu saja membutuhkan teknik untuk uji beban, namun dalam melakukan uji beban dibutuhkan *resource user* untuk mengakses web dan waktu yang cukup lama karena dilakukan secara manual. Beberapa contoh *tools* yang dapat mengatasi hal tersebut untuk melakukan uji beban yaitu *JMeter*, *k6* dan sebagainya. Namun *tools* tersebut menggunakan *thread* untuk *resource user*-nya, bila dilihat kondisi lapangan yaitu pengguna mengakses web menggunakan *browser* dan memiliki *IP* masing-masing, *thread* kurang bisa mengatasi hal tersebut. Oleh karena itu, dibutuhkan *tools* yang dapat menangani hal tersebut, salah satunya yaitu pengujian secara *headless* yang memanfaatkan infrastruktur *Docker* dan melakukan otomasi untuk pengambilan data uji beban. *Docker*

akan menjadi *resource user* yang mengakses melalui *headless browser* untuk uji beban.

Pada tugas akhir ini, akan dibangun sebuah sistem uji beban menggunakan teknik secara *headless* yang akan memanfaatkan *Headless Chrome* sebagai *tester* headless browsernya dan membuat *load generator* yang memanfaatkan infrastruktur *Docker* sebagai *resource user*, serta otomasi pengambilan data dari *tester* menggunakan pustaka *Node* yaitu *Puppeteer*[3].

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut :

1. Bagaimana cara mengimplementasikan *Headless Chrome* sebagai *tester* untuk *load generator*?
2. Bagaimana cara mengimplementasikan *Docker* bisa menjadi *load generator* untuk uji beban?
3. Bagaimana cara menghasilkan laporan uji beban pada web?
4. Bagaimana mengelola layanan pengujian untuk *multiuser* dalam bentuk antrian?

1.3 Batasan Masalah

Dari permasalahan yang telah dipaparkan di atas, terdapat beberapa batasan masalah pada tugas akhir ini, yaitu:

1. *Headless Browser* yang digunakan adalah *Headless Chrome*.
2. Kontainer yang digunakan adalah *Docker*.
3. Distribusi kontainer menggunakan *Docker Swarm*.
4. Aplikasi yang akan diuji berupa aplikasi web.
5. Uji coba aplikasi akan menggunakan *Node library yang menyediakan API (Application Programming Interface)* untuk mengontrol *Headless Chrome* yaitu *Puppeteer*.

1.4 Tujuan

Tujuan pembuatan tugas akhir ini antara lain:

1. Membuat sistem manajemen pengujian aplikasi secara *headless* menggunakan *Headless Chrome*.
2. Membuat sistem agar *Docker* bisa menjadi *load generator* untuk pengujian.
3. Membuat sistem untuk menampilkan laporan uji beban pada web.
4. Membuat sistem yang dapat mengelola permintaan layanan uji beban dari *multiuser* berupa antrian.

1.5 Manfaat

Manfaat dari pembuatan tugas akhir ini yaitu:

1. Mempelajari penggunaan *Headless Browser* untuk pengujian suatu aplikasi yaitu *Headless Chrome*
2. Mempelajadi penggunaan *Docker* sebagai *load generator* untuk melakukan uji beban.
3. Mengetahui performa uji beban suatu aplikasi web.
4. Meminimalisir adanya kegagalan web saat dimuat ketika sudah diluncurkan.

1.6 Metodologi

Metodologi yang digunakan untuk pembuatan Tugas Akhir ini adalah sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi tentang deskripsi pendahuluan dari tugas akhir yang akan dibuat. Pendahuluan ini terdiri dari hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah untuk tugas

akhir, tujuan dari pembuatan tugas akhir, dan manfaat dari hasil pembuatan tugas akhir. Selain itu dijabarkan pula tinjauan pustaka yang digunakan sebagai referensi pendukung pembuatan tugas akhir. Sub bab metodologi berisi penjelasan mengenai tahapan penyusunan tugas akhir mulai dari penyusunan proposal hingga penyusunan buku tugas akhir. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

1.6.2 Studi Literatur

Pada tahap ini dilakukan pencarian informasi dan referensi mengenai *Headless Chrome*, *Puppeteer* dan *Docker* untuk mendukung dan memastikan setiap tahap pembuatan tugas akhir sesuai dengan standar dan konsep yang berlaku, serta dapat diimplementasikan. Sumber informasi dan referensi bisa didapatkan dari buku, jurnal dan internet.

1.6.3 Analisis dan Desain Perangkat Lunak

Pada tahap ini dilakukan analisis dan perancangan terhadap arsitektur sistem yang akan dibuat. Tahap ini merupakan tahap yang paling penting dimana segala bentuk implementasi bisa bekerja dengan baik ketika arsitektur sistem yang baik pula.

1.6.4 Implementasi Perangkat Lunak

Pada tahap ini dilakukan implementasi atau realisasi dari hasil analisis dan perancangan arsitektur yang sudah dibuat sebelumnya, sehingga menjadi sebuah infrastruktur yang sesuai dengan apa yang direncanakan.

1.6.5 Pengujian dan Evaluasi

Pada tahap ini dilakukan pengujian untuk mengukur performa web dan kegagalan saat dimuat menggunakan

arsitektur sistem yang sudah dibuat menggunakan infrastruktur *Docker*. Beberapa performa yang diukur pada pengujian antara lain, *Load Event End*, *Response End*, *First Meaningful Paint*, *CSS Tracing End* dan *Dom Content Loaded* dalam satuan *ms(millisecond)* serta *error console*. Setelah dilakukan uji coba, maka dilakukan evaluasi terhadap kinerja arsitektur sistem yang telah diimplementasikan dengan harapan bisa diperbaiki ketika ada pengembangan ke depannya.

1.6.6 Penyusunan Buku Tugas Akhir

Pada tahap ini dilakukan penyusunan buku tugas akhir yang berisikan dokumentasi yang mencakup teori, konsep, implementasi dan hasil pengerjaan tugas akhir.

1.7 Sistematika Penulisan

Sistematika penulisan laporan tugas akhir secara garis besar adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan masalah, tujuan, manfaat, metodologi dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan metode, algoritme, *library* dan *tools* yang digunakan dalam penyusunan tugas akhir ini. Kajian teori yang dimaksud berisi tentang penjelasan singkat mengenai *Headless Browser*, *Headless Chrome*, *Puppeteer*, *NodeJS*, *Docker*, *Docker Swarm*, *Laravel* dan *Python*.
3. Bab III. Desain dan Perancangan
   Bab ini berisi mengenai analisis dan perancangan arsitektur sistem yang akan diimplementasikan pada tugas akhir ini.

4. Bab IV. Implementasi
   Bab ini berisi bahasan tentang implementasi dari arsitektur sistem yang dibuat pada bab sebelumnya. Penjelasan berupa kode program yang digunakan untuk mengimplementasikan sistem.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisi bahasan tentang tahapan uji coba terhadap performa web dan evaluasi terhadap sistem yang dibuat.
6. Bab VI. Penutup
   Bab ini merupakan bab terakhir yang memaparkan kesimpulan dari hasil pengujian dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran yang ditujukan bagi pembaca yang berminat untuk melakukan pengembangan lebih lanjut.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan.

BAB II

TINJAUAN PUSTAKA

2.1 *Headless Browser*

Headless Browser merupakan jenis perangkat lunak yang dapat mengakses halaman web, namun tidak menampilkan antarmuka pengguna. Seperti peramban pada umumnya, *Headless Browser* juga memiliki kemampuan yang sama, hanya saja menyisakan mesin dan lingkungan *javascript*. Oleh karena itu *Headless Browser* hanya bisa diakses menggunakan baris perintah atau *CLI(Command Line Interface)*[4]. Beberapa *Headless Browser* yaitu *Headless Chrome*, *Selenium WebDriver* dan *Firefox Headless Mode*. Salah satu *Headless Browser* yang akan digunakan pada Tugas Akhir ini adalah *Headless Chrome*, karena *Headless Chrome* memiliki fitur khusus yang bisa ditemukan pada bagian *performance*. *Headless Chrome* juga memiliki fitur umum seperti *Headless Browser* yang lain. Beberapa fitur umumnya yaitu[5]:

1. Memudahkan untuk melakukan pengujian secara otomatis.
2. Bisa dijalankan di server, mode *headless* tidak membutuhkan antarmuka pengguna.
3. Membuat dokumen atau file seperti PDF dan Screenshot.
4. *Debugging*.

Dalam tugas akhir ini, *Headless Browser* akan digunakan sebagai browser untuk menguji web yang akan diuji performanya, sedangkan jenis yang digunakan adalah *Headless Chrome*. *Headless Chrome* saat ini dikembangkan oleh *Google Developer* dan memiliki lisensi dari *Apache 2.0 License*.

2.2 *Node.js*

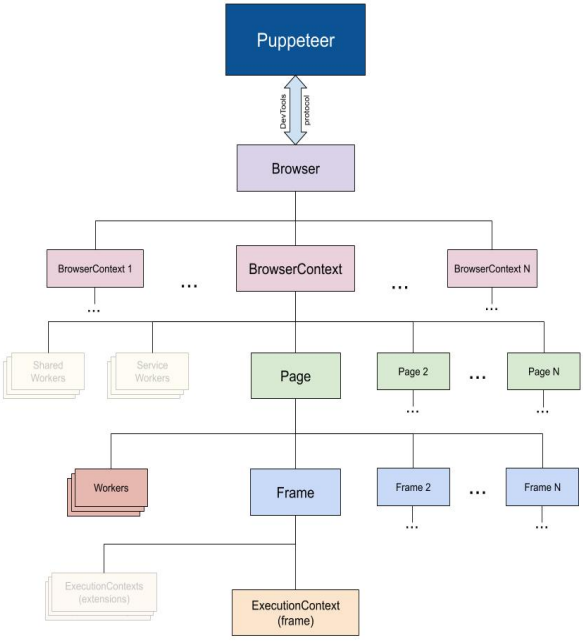
Node.js atau *node* adalah sebuah platform dengan lingkungan *JavaScript* sisi server. *Node.js* berbasis pada *Chrome's*

JavaScript Runtime yang menggunakan teknologi V8 dan berfokus pada performa maupun konsumsi memori rendah. Tapi V8 juga mendukung proses server yang berjalan lama. Tidak seperti kebanyakan platform modern yang lain dengan mengandalkan *multithreading*. *Node.js* menggunakan penjadwalan I/O secara asinkron. Proses pada *Node.js* dibayangkan sebagai proses *single-threaded daemon*. Hal ini berbeda dengan kebanyakan sistem penjadwalan dalam bahasa pemrograman lain yang berbentuk *library*. *Node.js* seringkali digunakan pengembang sebagai web server atau layanan *API*. Selain itu *Node.js* juga mendukung *event callback* untuk setiap penggunaan fungsi, memungkinkan ketika fungsi tersebut dipanggil akan terjadi *sleep* ketika tidak ada hasil apapun.[6][7].

Salah satu pustaka *Node.js* yang menyediakan layanan *API* adalah *Puppeteer*. *Node.js* akan digunakan sebagai bahasa pemrograman untuk mengimplementasikan *Puppeteer*. *Puppeteer* adalah sebuah pustaka dari *Node* yang memiliki kemampuan yang mumpuni untuk memberikan layanan *API* yang berfungsi untuk mengontrol layanan dari *Chrome* atau *Chromium*. Selain itu, kemampuan kontrol *Puppeteer* sangat memungkinkan untuk akses pada protokol *Devtools Protocol* yang saat ini dikembangkan oleh tim *Google Developer*, dimana protokol tersebut memiliki kemampuan yang cukup berguna yaitu sebagai *tools instrument*, *inspect*, *debug*, dan *profile chrome*[3].

Dibandingkan dengan *PhantomJS* yang sudah tidak dikembangkan lagi, *Puppeteer* masih dikembangkan secara berkala. Begitupun fitur-fitur yang disediakan *Puppeteer* sangat mumpuni untuk melakukan beberapa pengujian terhadap web yaitu melakukan pengambilan gambar ataupun pdf, otomasi, pengujian antarmuka pengguna, *keyboard input*, *timeline trace* untuk mendapatkan performa dan ekstensi pada *Chrome*. Untuk lebih jelasnya struktur diagram *Puppeteer* ditunjukkan pada

Gambar 2.1.



Gambar 2.1: Struktur diagram *Puppeteer*

Pada tugas akhir ini, *Puppeteer* akan digunakan sebagai alat untuk mengontrol *Headless Browser* yang akan diimplementasikan pada sistem perangkat lunak yang akan dibangun, karena lebih mumpuni dan memiliki beragam fitur *API* untuk mengakses *Headless Chrome* dibandingkan dengan pustaka *Node* yang lain.

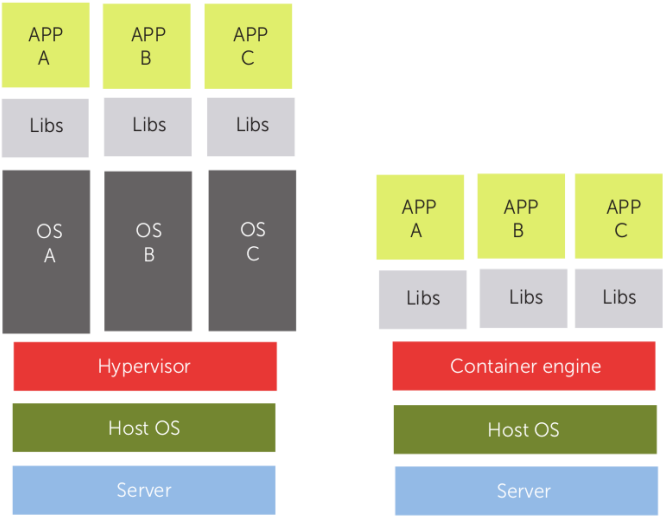
2.3 *Docker*

Docker adalah sebuah platform terbuka yang berfungsi sebagai wadah untuk membangun, membungkus, dan menjalankan aplikasi supaya dapat berfungsi sebagaimana mestinya. *Docker* memungkinkan untuk memisahkankan aplikasi dari infrastruktur supaya *software* dapat di jalankan dengan lebih cepat. *Docker* pada dasarnya memperluas *LXC(Linux Containers)* menggunakan kernel dan *API* pada level aplikasi yang akan dijalankan secara bersamaan pada isolasi *CPU*, memori, I/O, jaringan dan yang lainnya. *Docker* juga menggunakan *namespaces* untuk mengisolasi segala tampilan pada aplikasi yang mendasari lingkungan operasinya, termasuk *process tree*, jaringan, ID pengguna, dan file sistem.

Docker Container dibuat oleh sebuah *Docker Images*. *Docker Images* hanya mencakup dasar dari operasi sistem atau hanya memuat set dari *prebuilt* aplikasi yang sudah siap dijalankan. Ketika membuat *Docker Images*, bisa menjalankan perintah (yaitu apt-get install) membentuk lapisan baru diatas lapisan sebelumnya. Perintah tersebut bisa dijalankan manual satu-persatu atau secara otomatis menggunakan *Dockerfile*.

Setiap *Dockerfile* adalah kombinasi beberapa perintah yang dibuat menjadi menjadi satu atau *script* yang bisa dijalankan secara otomatis sebagai *Docker Images* utama atau untuk membuat *Docker Images* yang baru[8][1]. *Docker* juga menyediakan layanan untuk mengunduh dan mengupload *Docker images* melalui `https://hub.docker.com/`. Untuk melihat perbedaan antara kontainer dan *VM(Virtual Machine)* dapat dilihat pada Gambar 2.2.

Pada tugas akhir ini, *Docker Container* akan digunakan sebagai pengguna untuk mengakses web yang akan diuji, dan bisa diumpamakan sebagai pengguna asli untuk otomasi pengujiannya.



Gambar 2.2: Perbandingan kontainer dan *Virtual Machine*[1]

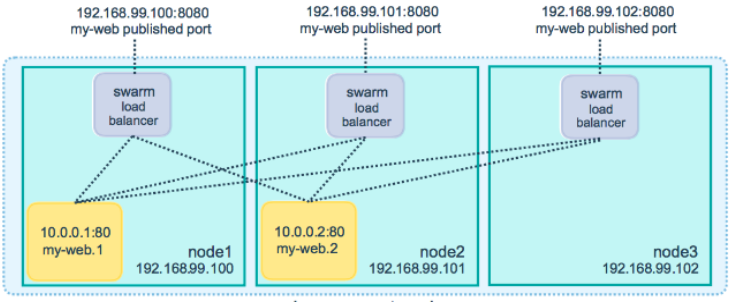
2.3.1 *Docker Swarm*

Docker Swarm - disebut juga *Swarm* adalah mode pada *Docker* yang memiliki fitur yang tertanam pada mesinnya untuk Manajemen *Cluster* atau *Orkestrasi*. Pada mode *Swarm* akan terdapat lebih dari satu *host* dimana *host* tersebut bisa berfungsi sebagai manager, worker, atau bisa juga keduanya. Konsep pada *Swarm* antara lain adalah *Nodes*, *Services*, *Tasks* dan *Load Balancing*. Mode *Swarm* juga memudahkan untuk mengatur bagian replikasi, jaringan, penyimpanan, port dan sebagainya.

Dibandingkan dengan kontainer yang berdiri sendiri, *Swarm* lebih mudah untuk mengubah konfigurasi servis, termasuk penyimpanan dan jaringannya tanpa harus menyalakan kembali kontainer secara manual. *Docker* akan otomatis memperbarui konfigurasi dengan cara menghentikan *service task* yang

memiliki konfigurasi lama, kemudian akan membuat kembali *service task* menggunakan konfigurasi yang sudah diperbarui. *Swarm* juga bisa menggunakan *Docker Compose* untuk mendefinisikan dan menjalankan kontainer, *Docker Compose* menggunakan *YAML file* sebagai konfigurasinya.[9].

Pada saat mode *Swarm*, *node manager* akan mengimplementasikan *Raft Consensus Algorithm* untuk memanajemen *cluster*. *Consensus* memungkinan manajer untuk mengatur dan menjadwal *tasks* pada setiap *cluster* dan memastikan status tetap konsisten, dimana ketika ada salah satu *nodes* yang gagal dalam mejalankan servis, manajer bisa mengembalikan servis menjadi stabil kembali[10]. Untuk melihat rute diagram ketika mode *Swarm* dapat dilihat pada Gambar 2.3.



Gambar 2.3: Rute diagram ketika mode *Swarm*[2]

Pada tugas akhir ini, *Docker Swarm* akan digunakan sebagai manager atau okestrasi yang mengatur segala servis maupun aktivitas *Docker Container* dan sebagai *Load Balancer* untuk pembagian beban kontainer pada setiap *nodes* yang merupakan instansi dari *Docker Swarm* tersebut.

2.4 *Laravel*

Laravel adalah salah satu kerangka kerja yang berbahasa *PHP* dan dibuat untuk memudahkan pengembang untuk mengembangan dan mendesain sebuah web yang menekankan kesederhanaan dan fleksibitas. Kerangka kerja ini mendukung metode *MVC(Model-View-Controller)*. dimana *MVC* digunakan untuk mengembangkan sebuah aplikasi yang memisahkan data(*Model*) dari tampilan(*View*) dan juga dari logika dari aplikasi tersebut(*Controller*)[11].

Model digunakan untuk memanipulasi data dari basis data, *View* berhubungan dengan antarmuka web seperti *HTML*, *CSS* dan *JS* sebagai data pada pengguna. *Controller* berhubungan dengan segala urusan logika pada servis web tersebut atau juga bisa disebut otaknya. *Controller* juga berfungsi sebagai jembatan antara *View* dan *Model*[11].

Pada tugas akhir ini, kerangka kerja *Laravel* akan digunakan untuk mengimplementasikan aplikasi web yang dibangun pada tugas akhir ini, dimana kerangka kerja ini sangat banyak digunakan oleh pengembang, memiliki dokumentasi resmi yang sangat baik, serta forum yang cukup baik. *Laravel* yang akan digunakan adalah versi 5.8.

2.5 *Python*

Python adalah bahasa pemrograman tingkat tinggi yang didukung oleh struktur data *built-in* semantik dinamis, selain itu Python mendukung pemrograman *procedural*, *object-oriented* dan *functional*. *Python* merupakan bahasa pemrograman *interpreted*, oleh sebab itu, *Python* tidak memakan biaya untuk kompilasi, sehingga proses pengembangan, pengujian dan *debug* menjadi lebih cepat.

Kelebihan bahasa pemrograman ini adalah memiliki modul

dan *package*, serta memiliki banyak standar pustaka yang didistribusikan secara bebas dan gratis. Selain itu *Python* mudah dibaca karena memiliki sintaksis yang sederhana, sehingga dapat mengurangi biaya *maintenance*. *Debug* pada *Python* juga mudah karena tidak akan terjadi *segmentation fault*, namun akan memberi umpan balik berupa *exception* apabila terdapat kesalahan atau *error*[12].

Pada tugas akhir ini, Bahasa pemrograman *Python* akan digunakan untuk mengimplementasikan algoritme *task queue* pada sistem yang akan dibangun. *Python* yang digunakan adalah versi 3.6.8.

BAB III

DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan sistem.

3.1 Deskripsi Umum Sistem

Sistem yang akan dibangun pada tugas akhir ini adalah sebuah sistem yang dapat melakukan otomasi uji beban terhadap suatu web. Uji beban pada sistem akan berjalan secara *headless* menggunakan sebuah *tester* yaitu *Headless Chrome*. *Headless Chrome* akan mendapatkan data uji beban ketika mengakses web yang diuji, sedangkan yang digunakan untuk mengambil data uji beban adalah sebuah pustaka *Node* yaitu *Puppeteer*. Sistem juga akan menggunakan *Docker* sebagai infrastruktur, sehingga *Docker* dapat digunakan sebagai *load generator* untuk melakukan uji beban yang bisa disebut kontainer.

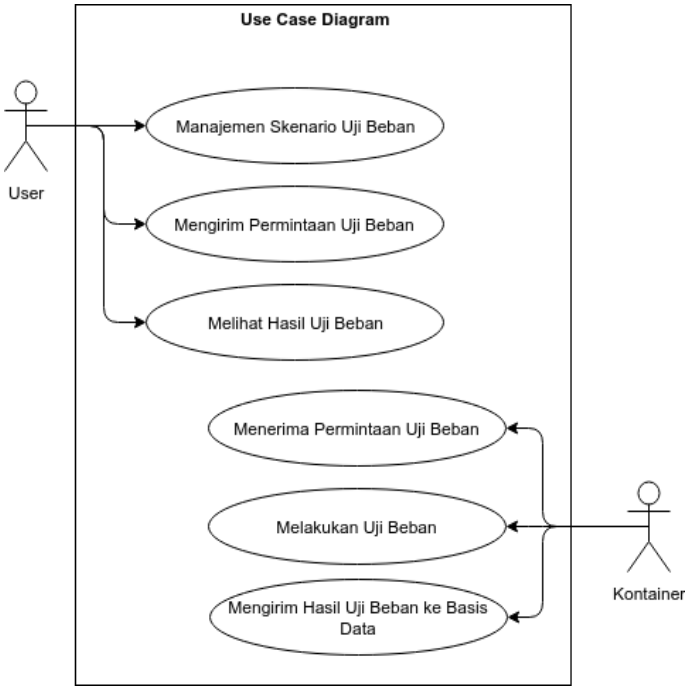
Kontainer yang akan dipasang pada sistem membutuhkan sebuah alat orkestrasi untuk memanajemen kontainer secara otomatis, alat orkestrasi yang digunakan adalah *Docker Swarm*. *Docker Swarm* akan melibatkan 3 *node host* yang akan dibagi menjadi 1 *node host* sebagai *swarm manager* dan 2 *node host* sebagai *worker*. *Docker Swarm* akan bertanggung jawab dalam mendistribusikan kontainer ke masing-masing *swarm node* yang tergabung pada lingkungan *swarm* atau bisa disebut sebagai *load balancer*.

Proses uji beban akan diproses ketika pengguna melakukan *request* skenario uji beban pada *web service* yang disediakan sistem. Kemudian *controller* pada *web service* akan mengirimkan skenario uji pada kontainer terpilih untuk melakukan pengujian. Setiap kontainer akan terinstall *Headless Chrome* dan *Puppeteer*. *Headless Chrome* akan mendapatkan data uji beban dan otomasi pengambilan data uji beban dilakukan oleh *Puppeteer*. *Puppeteer* akan melakukan ekstraksi data uji

beban menjadi satuan *millisecond(ms)*.

Sistem akan menyediakan basis data untuk menyimpan data yang diperlukan sistem. Basis data akan dipasang diluar lingkungan swarm dan lingkungan *web service*. Sistem juga menyediakan antarmuka pengguna berupa web yang akan digunakan untuk melihat laporan hasil uji beban. Sedangkan untuk mengatasi *multiuser*, sistem akan menggunakan *task scheduler(crontab)* dan *queue*(antrian).

3.2 Kasus Penggunaan



Gambar 3.1: Diagram kasus penggunaan

Terdapat dua aktor dalam sistem yang akan dibuat yaitu *User* dan Kontainer. *User* merupakan aktor(pengguna) yang bisa melakukan manajemen pada skenario yang ingin diuji dan melihat hasilnya, sedangkan Kontainer merupakan aktor yang akan digunakan sebagai *load generator* untuk melakukan uji beban. Diagram kasus penggunaan digambarkan pada Gambar 3.1 dan dijelaskan masing-masing pada Table 3.1.

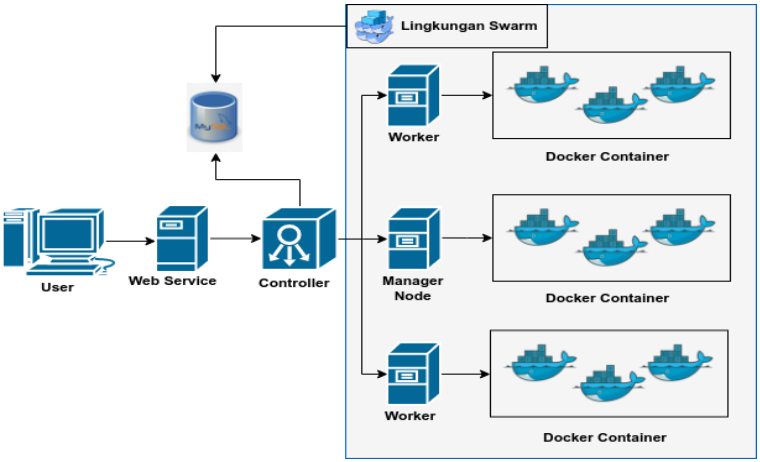
Tabel 3.1: Daftar kode kasus penggunaan

Kode Kasus Penggunaan	**Nama Kasus Penggunaan**	**Keterangan**
UC-0001	Manajemen Skenario Uji Beban	*User* dapat menambah, melihat dan menghapus skenario uji beban
UC-0002	Mengirim Permintaan Uji Beban	*User* dapat mengirimkan permintaan uji beban ke sistem melalui *web service* yang disediakan
UC-0003	Melihat Hasil Uji Beban	Ketika proses uji beban selesai, *user* dapat melihat hasilnya di antarmuka pengguna *web service* yang disediakan
UC-0004	Menerima Permintaan Uji Beban	Proses dimana kontainer akan menerima permintaan uji beban dari *User*

Tabel 3.1: Daftar kode kasus penggunaan

Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
UC-0005	Melakukan Uji Beban	Proses dimana kontainer akan melakukan uji beban sesuai skenario yang dikirim
UC-0006	Mengirim Hasil Uji Beban ke Basis Data	Ketika kontainer telah selesai melakukan pengujian, data yang didapatkan akan dikirim ke basis data *MySQL*

3.3 Arsitektur Sistem



Gambar 3.2: Desain arsitektur sistem

Pada sub-bab ini, akan dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun. Arsitektur sistem secara umum ditunjukkan pada Gambar 3.2.

3.3.1 Desain Umum Sistem

Berdasarkan yang dijelaskan pada deskripsi umum sistem, dapat diperoleh beberapa kebutuhan sistem antara lain:

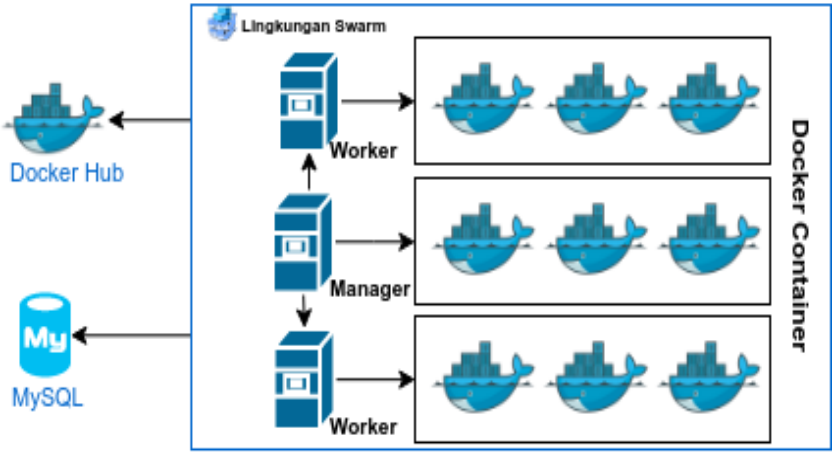
1. *Load generator* untuk melakukan uji beban.
2. *Tester* yang bisa mengambil data uji beban.
3. *Web service* sebagai antarmuka pengguna.
4. Basis data untuk menyimpan data sistem.
5. *Task queue* untuk menangani kasus *request* lebih dari satu user.

Untuk memenuhi kebutuhan sistem yang dijelaskan sebelumnya, penulis membagi menjadi beberapa komponen sistem yang akan digunakan pada tugas akhir ini.

1. *Load generator*
   Berfungsi sebagai pengganti user yang akan melakukan akses web melalui *browser*.
2. Pengambil data uji beban
   Berfungsi untuk mengambil data uji beban ketika *load generator* mengakses web dari *browser*.
3. *Service Controller*
   Berfungsi sebagai pengatur sistem uji beban yang terdiri :
   - *Web Service*
     Berfungsi sebagai tampilan antarmuka pengguna untuk menggunakan sistem.
   - Basis Data
     Berfungsi untuk menyimpan data yang digunakan untuk menyimpan segala data yang dibutuhkan oleh sistem.
   - *Task Queue*

Berfungsi untuk membuat *task scheduler* dan antrian untuk menangani kasus *request* lebih dari satu user.

3.3.2 Perancangan *Load Generator*



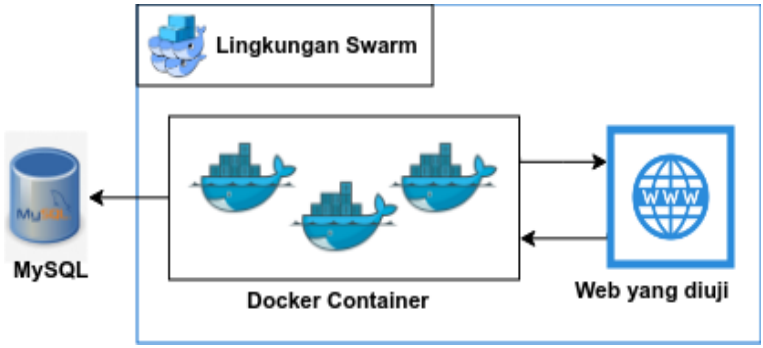
Gambar 3.3: Desain perancangan *load generator*

Komponen *load generator* akan difungsikan sebagai pengganti pengguna yang mengakses ke suatu web melalui *browser*. Komponen yang akan digunakan sebagai *load generator* adalah *Docker* atau bisa disebut kontainer. Ketika melakukan pemasangan kontainer sebuah *Docker Image*, untuk memenuhi hal tersebut, pada tugas akhir ini penulis akan membuat sebuah *Docker Image* yang akan diunggah ke *Docker Hub*. Sehingga ketika akan memasang kontainer pada *node host* yang baru, node host tersebut hanya perlu mengunduh *Docker Image* yang telah diunggah sebelumnya. Seluruh kontainer akan dibangun di dalam lingkungan *swarm* untuk memudahkan dalam mengatur atau memanajemen kontainer ke semua *node host* yang tergabung di dalam lingkungan *swarm*, sedangkan untuk

memudahkan akses ke setiap kontainer, maka data dari kontainer akan disimpan di dalam basis data *MySQL*. *Load generator* akan terdiri dari 3 *node host*, 1 sebagai *manager node* dan 2 lainnya sebagai *worker*, sedangkan basis data akan berada di luar lingkungan *swarm*. Desain perancangan komponen ini digambarkan pada Gambar 3.3.

3.3.3 Perancangan Pengambil Data Uji Beban

Komponen ini akan membutuhkan suatu alat yang bisa mendapatkan data uji beban terlebih dahulu. Pada tugas akhir ini, akan menggunakan *Headless Chrome* untuk mendapatkan data uji beban ketika mengakses web. Setelah mendapatkan data uji beban, diperlukan juga suatu alat yang bisa digunakan untuk mengambil data uji beban pada *Headless Chrome*, alat tersebut adalah *Puppeteer*. *Puppeteer* akan melakukan pengambilan secara otomatis ketika ada perintah yang masuk dan menyimpan data uji beban pada basis data *MySQL*. Alat-alat yang digunakan pada komponen ini akan dipasang pada masing-masing kontainer di setiap *node host*. Desain perancangan komponen ini digambarkan pada Gambar 3.4.



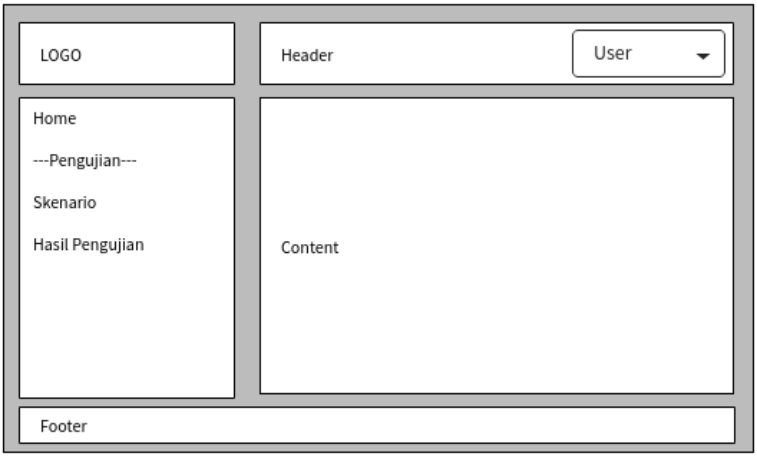
Gambar 3.4: Desain pengambil data uji beban

3.3.4 Perancangan *Service Controller*

Komponen ini akan digunakan untuk mengatur segala proses uji beban pada sistem. Pada komponen ini akan terdapat 3 buah sub-komponen yaitu *web service*, basis data dan *task queue* atau antrian.

3.3.4.1 Desain *Web Service*

Web service akan berfungsi sebagai antarmuka pengguna dan sebagai penghubung antara pengguna dengan kontainer. Antarmuka pengguna berfungsi memudahkan pengguna untuk membuat skenario yang akan dikirimkan ke *load generator* dan kemudian *load generator* akan melakukan uji beban sesuai dengan skenario yang dikirim melalui *web service*. Sedangkan untuk mengatur segala aktivitasnya dibutuhkan sebuah *controller* dan rute yang akan dipasang pada *web service*. Desain antarmuka pengguna ditunjukkan pada Gambar 3.5.



Gambar 3.5: Desain antarmuka pengguna

Selain itu, akan dirancang juga fitur-fitur pada *web service* yang akan digunakan pengguna antara lain:

1. Menambah dan menghapus skenario.
2. Menentukan jumlah *load generator* pengujian.
3. Melihat performa hasil pengujian.
4. Melihat tangkapan layar tampilan web yang diuji.
5. Melihat *console error*.
6. Melihat status antrian proses uji.

3.3.4.2 Desain Basis Data

Komponen basis data diperlukan untuk menyimpan data-data yang berkaitan dengan sistem. data yang disimpan adalah data *node host swarm*, data kontainer, data pengguna, data skenario pengujian, data antrian *request*, data hasil pengujian, data *error console*, data rata-rata hasil pengujian. Dari data-data tersebut maka dibutuhkan suatu tabel diantaranya yaitu:

- Tabel *swarms*
  Menyimpan data *nohe host* yang tergabung di dalam lingkungan *swarm*.
- Tabel *containers*
  Menyimpan data *Docker Container* yang telah dipersiapkan.
- Tabel *users*
  Menyimpan data pengguna.
- Tabel *scenarios*
  Menyimpan data skenario pengujian.
- Tabel *queues*
  Menyimpan data antrian *request* pengujian dari pengguna.
- Tabel *results*
  Menyimpan data hasil pengujian yang dilakukan setiap kontainer.

- Tabel *errors*
  Menyimpan data *console error* yang ada di *browser*.
- Tabel *summary results*
  Menyimpan data rata-rata hasil pengujian setiap skenario.

3.3.4.3 Desain Penggunaan *Task Queue*

Pada *service controller* akan ada banyak *request* dari pengguna, setiap *request* tentu saja akan terdapat proses yang akan berjalan dalam jangka waktu yang cukup lama. Jika proses tersebut berada di dalam fungsi yang dipanggil melalui protokol *HTTP*, maka akan memberikan umpan balik setelah semua proses yang ada dibaliknya selesai. Hal ini akan membuat pengguna yang melakukan *request* perlu menunggu dan tidak efisien. Untuk mengatasi hal ini, akan dirancang sebuah komponen antrian atau bisa disebut *task queue*. *Task queue* akan membuat antrian untuk setiap *request* dibelakang layar. Antrian *request* tersebut akan disimpan pada basis data *MySQL*.

BAB IV

IMPLEMENTASI

Bab ini membahas mengenai implementasi dari sistem yang sudah di desain dan dirancang pada bab sebelumnya. Pembahasan secara rinci akan dijelaskan pada setiap komponen yang ada yaitu *load generator*, pengambil data uji beban dan *service controller* yang meliputi *web service*, basis data dan *task queue*.

4.1 Lingkungan Implementasi

Dalam mengimplementasikan sistem pada tugas akhir ini, digunakan beberapa perangkat pendukung sebagai berikut.

4.1.1 Perangkat Keras

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. *Web service* dan *task queue*, *processor* AMD FX-7600P Radeon R7, 12 Compute Cores 4C+8G dan RAM 8GB.
2. *Node swarm* dengan IP 167.71.194.235, *processor* Intel(R) Xeon(R) CPU E5-2650 v4@2.20GHz dan RAM 4GB.
3. *Node swarm* dengan IP 165.22.55.82, *processor* Intel(R) Xeon(R) CPU E5-2650 v4@2.20GHz dan RAM 4GB.
4. *Node swarm* dengan IP 167.71.194.233, *processor* Intel(R) Xeon(R) CPU E5-2650 v4@2.20GHz dan RAM 4GB.
5. Basis data *MySQL* dengan IP 178.128.123.143, *processor* Intel(R) Xeon(R) Gold 6140 CPU@2.30GHz dan RAM 1GB.

4.1.2 Perangkat Lunak

Perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Sistem Operasi *Ubuntu* 18.04 LTS 64 Bit
2. *Docker* versi 18.09.6

3. *Headless Chrome*
4. *Puppeteer* versi 0.12.0
5. *NPM* versi 6.4.1
6. *Node.js* versi 8.15.1
7. *Python* versi 3.6.8
8. *MySQL* Ver 14.14 Distrib 5.7.26
9. *Shell Script*
10. *PHP* dan *Laravel* versi 5.8

4.2 Implementasi *Load Generator*

Berdasarkan perancangan dan desain, *load generator* merupakan aktor yang akan berfungsi menggantikan pengguna ketika mengakses ke web. *Load generator* yang akan digunakan adalah *Docker*, namun diperlukan beberapa tahap untuk bisa menggunakan *Docker* atau kontainer sebagai *load generator*, yaitu tahap pemasangan dan konfigurasi. Tahap pemasangan *Docker* dapat dilihat di Kode Sumber A.1, sedangkan untuk konfigurasi akan dibagi menjadi beberapa tahap yaitu:

1. Pembuatan *Docker Image*
2. Pembuatan lingkungan kontainer
3. Pemasangan *Headless Chrome* dan *Puppeteer*

4.2.1 Implementasi Pembuatan *Docker Image*

Docker Image digunakan untuk menjalankan kontainer, pada tugas akhir ini *Docker Image* akan dibuat terlebih dahulu agar bisa digunakan untuk menjalankan *Puppeteer* dan *Headless Chrome*. Namun untuk membuat *Docker Image* diperlukan beberapa tahapan yaitu konfigurasi `Dockerfile`, pemasangan *Docker Compose* dapat dilihat pada Kode Sumber A.3 dan konfigurasi `docker-compose.yml` dan unggah *Docker Image* ke *Docker Hub*.

Konfigurasi `Dockerfile` dapat dilihat pada Kode Sumber IV.1, sedangkan konfigurasi `docker-compose.yml` dapat dilihat pada Kode Sumber IV.2. Kemudian jalankan perintah *Docker Compose* pada Kode Sumber IV.3 untuk pembuatan *Docker Image*. Setelah *Docker Image* terbuat, ubah nama *Docker Image* tersebut dan melakukan *commit* agar bisa di *push*. Perintah pada Kode Sumber IV.4 yang digunakan oleh penulis ketika mengunggah *Docker Image* ke *Docker Hub*.

```
1  FROM node:8
2  RUN apt-get update
3  # for https
4  RUN apt-get install -yyq ca-certificates
5  # install libraries
6  RUN apt-get install -yyq libappindicator1
       libasound2 libatk1.0-0 libc6 libcairo2
       libcups2 libdbus-1-3 libexpat1 libfontconfig
       1 libgcc1 libgconf-2-4 libgdk-pixbuf2.0-0
       libglib2.0-0 libgtk-3-0 libnspr4 libnss3
       libpango-1.0-0 libpangocairo-1.0-0 libstdc++
       6 libx11-6 libx11-xcb1 libxcb1 libxcomposite
       1 libxcursor1 libxdamage1 libxext6 libxfixes
       3 libxi6 libxrandr2 libxrender1 libxss1
       libxtst6
7  # tools
8  RUN apt-get install -yyq gconf-service lsb-
       release wget xdg-utils
9  RUN apt-get install -yyq fonts-liberation
10 COPY code /app/code
11 COPY output /app/output
12 WORKDIR /app/code
13 RUN yarn install
```

Kode Sumber IV.1: Konfigurasi *Dockerfile*

```
version: '3'
services:
  puppeteer:
    build: .
    shm_size: '1gb'
    entrypoint: ["sh", "-c", "sleep infinity"]
```

Kode Sumber IV.2: Konfigurasi *docker-compose.yml*

```
$ docker-compose up
```

Kode Sumber IV.3: Perintah untuk menjalankan *Docker Compose*

```
$ docker tag puppeteer_puppeteer:latest
    cphikmawan/ta2019:newpupp

$ docker push cphikmawan/ta2019:newpupp
```

Kode Sumber IV.4: Perintah untuk mengunggah *Docker Image*

4.2.2 Implementasi Pembuatan Lingkungan Kontainer

Kontainer akan dipasangkan pada suatu lingkungan yang bisa mengatur segala aktivitas kontainer, lingkungan yang akan dibangun pada sistem menggunakan alat orkestrasi yaitu *Docker Swarm*. Untuk mengimplementasikan *Docker Swarm* pada sistem ini, dibutuhkan satu *node host* sebagai *manager node* dan dua lainnya sebagai *worker*. Tahap pertama yang dilakukan yaitu menginisiasi salah satu *node host* yang akan digunakan sebagai *manager node*. Perintah inisiasi *manager node* terdapat pada Kode Sumber IV.5.

```
$ docker swarm init --advertise-addr [IP NODE]
```

Kode Sumber IV.5: Perintah untuk inisiasi *manager node*

Setelah perintah pada kode sumber IV.5 dijalankan, maka *manager node* akan menghasilkan sebuah token yang digunakan oleh *node host* yang lain untuk bergabung sebagai *worker*. Perintah yang harus dijalankan pada setiap *node host* yang lain terdapat pada Kode Sumber IV.6.

```
$ docker swarm join --token [token] [IP MANAGER
    ]:2377
```

Kode Sumber IV.6: Perintah untuk bergabung ke *Swarm*

Tahap terakhir yang dilakukan yaitu memastikan semua *node host* sudah tergabung dengan *manager node*.

```
$ docker node ls
```

Kode Sumber IV.7: Perintah untuk melihat daftar *Swarm Node*

4.2.3 Implementasi Pemasangan *Headless Chrome* dan *Puppeteer*

Headless Chrome dan *Puppeteer* akan dipasang pada masing-masing kontainer menggunakan *Docker Image* yang telah dibuat sebelumnya, untuk pemasangannya akan dilakukan dilingkungan *Docker Swarm* dan dilakukan pada *manager node*. Namun untuk implementasinya dibutuhkan beberapa persiapan dan konfigurasi yang harus dilakukan terlebih dahulu yaitu konfigurasi unduh *Docker Image*, konfigurasi membuat *Docker Network*, konfigurasi *puppeteer.yml*, konfigurasi *deployment*.

Untuk mengunduh *Docker Image* dilakukan pada setiap *node host* menggunakan perintah pada Kode Sumber IV.8 dan membuat *Docker Network* pada Kode Sumber IV.9. Sedangkan konfigurasi `puppeteer.yml` dapat dilihat di Kode Sumber IV.10

```
$ docker image pull cphikmawan/ta2019:puppeteer
```

Kode Sumber IV.8: Perintah untuk mengunduh *Docker Image*

```
$ docker network create \
  --driver overlay \
  --subnet 10.0.0.0/18 \
  --attachable \
  [nama_network]
```

Kode Sumber IV.9: Perintah untuk membuat *Docker Network*

```
1  version: '3'
2  # konfigurasi service
3  services:
4    # konfigurasi service puppeteer
5    puppeteer:
6      image: cphikmawan/ta2019:newpupp
7      volumes:
8        - ./output:/app/output
9        - ./code:/app/code
10     working_dir: /app/code
11     # konfigurasi untuk jumlah kontainer dan
          handling
12     deploy:
13       replicas: 1000
14       restart_policy:
15         condition: on-failure
16     # entrypoint awal sleep
17     entrypoint: ["sh", "-c", "sleep infinity"]
18 # konfigurasi jaringan
19 networks:
20   default:
21     external:
22       name: swarm-network
```

Kode Sumber IV.10: Konfigurasi *puppeteer.yml*

Tahap selanjutnya adalah konfigurasi *deployment* dengan menjalankan perintah yang ditunjukkan pada Kode Sumber IV.11 di terminal *manager node*

```
$ docker stack deploy --compose-file=puppeteer.
    yml [nama_stack]
```

Kode Sumber IV.11: Perintah untuk pemasangan kontainer

4.3 Implementasi Pengambil Data Uji Beban

Pengambil Data Uji Beban akan dilakukan menggunakan sebuah pustaka *Node* yaitu *Puppeteer*. Pada saat dilakukan uji beban, *Puppeteer* akan mengambil data uji beban dari *Headless Chrome* menggunakan *API* dari *Puppeteer* secara otomatis. Beberapa data uji beban yang akan diambil yaitu:

1. *Response End*
   Atribut ini menunjukkan waktu setelah *user* menerima *byte* terakhir dari dokumen sebelum koneksi transportasi ditutup.
2. *CSS Tracing End*
   Atribut ini menunjukkan waktu dari ekstraksi akhir file *CSS* dimuat.
3. *DOM Content Loaded*
   Atribut ini menunjukkan waktu setelah dokumen sudah diterima oleh *user*.
4. *First Meaningful Paint*
   Atribut ini adalah atribut khusus yang ada pada *Chrome* yang menunjukkan bahwa segala konten halaman yang dimuat sudah ditampilkan di layar.
5. *Load Event End*
   Atribut ini mengembalikan waktu ketika memuat dokumen selesai.

Selain data uji beban diatas, *Puppeteer* juga digunakan untuk mengambil sebuah tangkapan layar sesuai skenario yang dikirimkan oleh pengguna dan mengambil data saat terjadi kegagalan saat memuat *assets* yang tertulis pada *console browser*. Adapun *pseudocode* untuk melakukan pengambilan data uji beban dapat dilihat pada Kode Sumber IV.12.

```
1  Variable Declaration
2  Data = Read Scenario File Configuration
3
4  TESTPAGE FUNCTION:
5    CALL HELPERS FUNCTION:
6      GET Navigation Start
7    START Trace CSS Data
8    Trying to Accessing Website
9    END Trace CSS Data
10   CALL HELPERS FUNCTION:
11     GET Extracted Performance Data
12     GET Extracted CSS Tracing
13     RETURN Extracted Data
14 END TESTPAGE FUNCTION
15
16 HELPERS FUNCTION:
17   GET Navigation Start
18     RETURN Navigation Start
19   Extracted Performance Data = Performance Data
         * 1000 - Navigation Start
20     RETURN Extracted Performance Data
21   Extracted CSS Tracing = CSS Tracing / 1000
22     RETURN Extracted CSS Tracing
23 End Helpers Function
24
25 MAIN FUNCTION:
26   START Headless Browser
```

```
27      GET Error Console
28        RETURN Data to Database
29      TRY:
30        CALL TESTPAGE FUNCTION(Data):
31          Get Data From TestPage -> Save Data Uji
                Beban
32        GET Page Screenshoot -> Save Screenshoot
33      CATCH ERROR:
34        GET Error
35        GET Page Screenshoot -> Save Screenshoot
36    END MAIN FUNCTION
```

Kode Sumber IV.12: *Pseudocode Puppeteer*

4.4 Implementasi *Service Controller*

Berdasarkan desain dan perancangan, *service controller* terdiri dari komponen *web service*, basis data dan *task queue*. Komponen tersebut akan diimplementasikan pada satu komputer milik penulis yang akan digunakan untuk *web service* dan penggunaan *task queue*, serta satu buah server untuk basis data *MySQL*.

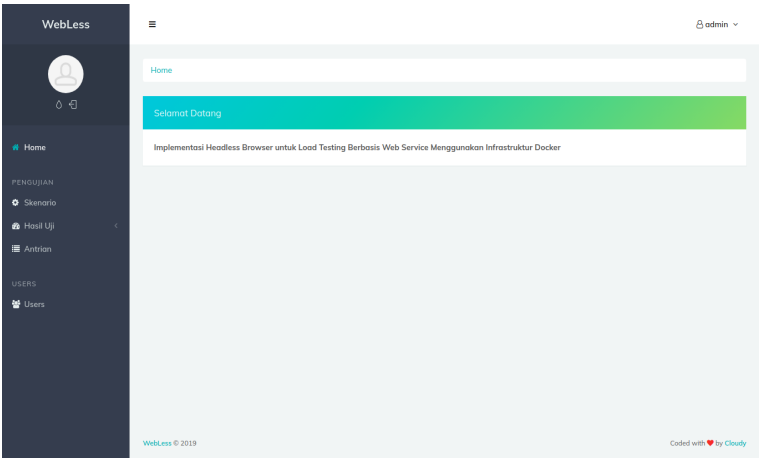
4.4.1 Implementasi *Web Service*

Pada implementasi *web service* dibutuhkan beberapa persiapan lingkungan yang perlu dilakukan, urutannya meliputi langkah-langkah berikut:

1. Instalasi *PHP*
2. Instalasi *Composer*
3. Instalasi *Laravel* versi 5.8
4. Instalasi *MySQL*

Web service akan menggunakan bahasa *PHP* dan kerangka kerja *Laravel* versi 5.8, sedangkan *Composer* berfungsi untuk

memanajemen instalasi pustaka pada *PHP* dan untuk penyimpanan data yang digunakan pada sistem akan disimpan pada basis data *MySQL*. *Web service* berfungsi untuk memudahkan pengguna melakukan uji beban pada suatu web. Tampilan antarmuka pengguna ditunjukkan pada Gambar 4.1.



Gambar 4.1: Tampilan web antarmuka pengguna

Web service memiliki beberapa rute *HTTP* yang akan digunakan oleh pengguna ketika mengakses web sistem. Rute-rute tersebut ditunjukkan pada Tabel 4.1.

Tabel 4.1: Rute *HTTP* pada *web service*

No	Rute	Metode	Aksi
1	/	GET	Mengakses halaman *home*
2	/login	GET	Mengakses halaman *login*
3	/login	POST	Melakukan *login*
4	/logout	GET	Melakukan *logout*

Tabel 4.1: Rute *HTTP* pada *web service*

No	Rute	Metode	Aksi
5	/skenario	GET	Melihat skenario
6	/skenario/create	GET	Mengakses halaman tambah skenario
7	/skenario	POST	Menambahkan skenario
8	/skenario/id	DELETE	Menghapus skenario
9	/worker	GET	Mengakses halaman *worker*
10	/worker	POST	Menambahkan jumlah *worker(load generator)* dan data antrian
11	/hasil/rata-rata	GET	Mendapatkan hasil pengujian
12	/hasil/error-console	GET	Mendapatkan *error console* web
13	/hasil/images	GET	Melihat tangkapan layar web
14	/antrian	GET	Melihat jumlah dan status antrian

4.4.2 Implementasi Skema Basis Data

Berdasarkan hasil perancangan basis data pada bab sebelumnya. Data yang dibutuhkan dan digunakan oleh sistem akan disimpan di dalam basis data *MySQL*. Data yang disimpan adalah data *node host swarm*, data kontainer, data pengguna, data skenario pengujian, data antrian *request*, data hasil pengujian,

data *error console*, data rata-rata hasil pengujian.

4.4.2.1 Tabel *Swarms*

Pada tabel *swarms* menyimpan data-data dari node host yang tergabung dilingkungan *swarm*. Berikut definisi tabel *swarms* pada Tabel 4.2.

Tabel 4.2: Tabel *swarms*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*.
2	swarm_ip	varchar(255)	Menunjukkan *IP* dari *node host*
3	swarm_username	varchar(255)	Menunjukkan *username* dari *node host*
4	swarm_password	varchar(255)	Menunjukkan *password* dari *node host*
5	is_used	smallint(6)	Menunjukkan status dari *node host*

4.4.2.2 Tabel *Containers*

Padata tabel *containers* menyimpan data-data dari *Docker Container* yang akan digunakan sebagai *load generator*. Berikut definisi tabel *containers* pada Tabel 4.3.

Tabel 4.3: Tabel *containers*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	task_id	varchar(100)	Menunjukkan *task id* dari kontainer
3	node_id	varchar(100)	Menunjukkan *node id* tempat kontainer dipasang
4	container_id	varchar(100)	Menunjukkan *id* dari kontainer
5	node_ip	varchar(100)	Menunjukkan *IP* tempat kontainer dipasang
6	node_host	varchar(100)	Menunjukkan *hostname* tempat kontainer dipasang
7	status	smallint(6)	Menunjukkan *flag* status dari *container*
8	username	varchar(100)	Menunjukkan status kontainer yang sedang digunakan *user*

4.4.2.3 Tabel *Users*

Pada tabel *users* menyimpan data-data pengguna web yang disediakan sistem. Berikut definisi tabel *users* pada Tabel 4.4. Data pengguna juga memiliki *constraint* dan disimpan pada Tabel *role_user* 4.5 dan Tabel *roles* 4.6.

Tabel 4.4: Tabel *users*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	name	varchar(255)	Menunjukkan nama pengguna
3	email	varchar(255)	Menunjukkan *email* pengguna
4	username	varchar(255)	Menunjukkan *username* pengguna
5	password	varchar(255)	Menunjukkan *password* pengguna

Tabel 4.5: Tabel *role_user*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	role_id	int(10)	Menunjukkan *id role* pengguna
3	user_id	int(10)	Menunjukkan *id* pengguna

Tabel 4.6: Tabel *roles*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*

Tabel 4.6: Tabel *roles*

No	Kolom	Tipe	Keterangan
2	name	varchar(255)	Menunjukkan nama *role*
3	description	varchar(255)	Menunjukkan deskripsi hak akses dari nama *role*

4.4.2.4 Tabel *Scenarios*

Pada tabel *scenarios* menyimpan data-data skenario yang telah dibuat oleh pengguna. Berikut definisi tabel *scenarios* pada Tabel 4.7.

Tabel 4.7: Tabel scenarios

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	scenario_id	varchar(255)	Menunjukkan skenario *id* sebagai *foreign key*
3	username	varchar(255)	Menunjukkan keterangan pengguna pembuat skenario
4	scenario_method	varchar(255)	Menunjukkan metode uji
5	scenario_link	varchar(255)	Menunjukkan *link website* yang diuji
6	scenario_worker	varchar(255)	Menunjukkan jumlah *load generator* yang diinginkan pengguna

Tabel 4.7: Tabel scenarios

No	Kolom	Tipe	Keterangan
7	scenario_status	smallint(6)	Menunjukkan status skenario, nilai awal adalah 0

4.4.2.5 Tabel *Queues*

Pada tabel *queues* menyimpan data-data antrian request yang dilakukan oleh semua pengguna. Berikut definisi tabel *queues* pada Tabel 4.8.

Tabel 4.8: Tabel *queues*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	created_at	timestamp	Menunjukkan waktu pembuatan *queue request* oleh pengguna
3	username	varchar(255)	Menunjukkan keterangan pengguna pembuat *request*
4	worker	int(11)	Menunjukkan jumlah *load generator* yang diinginkan pengguna
5	status	smallint(6)	Menunjukkan status dari *queue*, nilai awal adalah 0

4.4.2.6 Tabel *Results*

Pada tabel *results* menyimpan data-data hasil uji beban yang dilakukan oleh kontainer dan *Puppeteer*. Berikut definisi tabel *results* pada Tabel 4.9.

Tabel 4.9: Tabel *results*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	scenario_id	varchar(255)	Menunjukkan skenario *id* sebagai *foreign key*
3	link	varchar(255)	Menunjukkan *link* website yang diuji
4	method	varchar(100)	Menunjukkan metode uji
5	worker	varchar(255)	Menunjukkan jumlah *load generator* yang diinginkan pengguna
6	username	varchar(100)	Menunjukkan keterangan pengguna pembuat *request*
7	host	varchar(100)	Menunjukkan *IP node host* yang digunakan kontainer penguji
8	response_end	varchar(100)	Menunjukkan hasil uji response time dalam satuan *ms*
9	dom_content_load	varchar(100)	Menunjukkan hasil uji waktu memuat *DOM* web dalam satuan *ms*

Tabel 4.9: Tabel *results*

No	Kolom	Tipe	Keterangan
10	load_event_end	varchar(100)	Menunjukkan hasil uji *load time* dalam satuan *ms*
11	css_trace_end	varchar(100)	Menunjukkan hasil uji waktu memuat *css time* dalam satuan *ms*
12	first_meaningful	varchar(100)	Menunjukkan hasil uji waktu memuat konten utama dalam satuan *ms*
13	status	smallint(6)	Menunjukkan status apakah untuk proses rata-rata, nilai awal adalah 0

4.4.2.7 Tabel *Errors*

Pada tabel *errors* menyimpan data-data kegagalan yang terekam pada *console browser* ketika diakses didalam *Headless Chrome*. Berikut definisi tabel *errors* pada Tabel 4.10.

Tabel 4.10: Tabel *errors*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	scenario_id	varchar(255)	Menunjukkan skenario *id* sebagai *foreign key*
3	link	varchar(255)	Menunjukkan *link website* yang diuji

Tabel 4.10: Tabel *errors*

No	Kolom	Tipe	Keterangan
4	worker	varchar(255)	Menunjukkan jumlah *load generator* yang diinginkan pengguna
5	username	varchar(100)	Menunjukkan keterangan pengguna pembuat *request*
6	host	varchar(100)	Menunjukkan *IP node host* yang digunakan kontainer penguji
7	type	varchar(100)	Menunjukkan tipe *error*
8	text	varchar(255)	Menunjukkan keterangan *error*
9	args	varchar(255)	Menunjukkan argumen *error*
10	location_url	varchar(255)	Menunjukkan lokasi *url error*

4.4.2.8 Tabel *Summary Results*

Pada tabel *summary_results* menyimpan data perhitungan rata-rata dari hasil uji beban yang sudah disimpan pada tabel *results*. Data perhitungan rata-rata tersebut merupakan data *metrics performance* yang akan ditampilkan di web service dan pengguna dapat melihat hasil uji beban ini pada menu yang sudah disediakan. Berikut definisi tabel *summary_results* pada Tabel 4.11.

Tabel 4.11: Tabel *summary_results*

No	Kolom	Tipe	Keterangan
1	id	bigint(20)	Sebagai *primary key*, nilai awal adalah *auto_increment*
2	scenario_id	varchar(255)	Menunjukkan *skenario id* sebagai *foreign key*
3	link	varchar(255)	Menunjukkan *link website* yang diuji
4	method	varchar(100)	Menunjukkan metode uji
5	worker	varchar(255)	Menunjukkan jumlah *load generator* yang diinginkan pengguna
6	username	varchar(100)	Menunjukkan keterangan pengguna pembuat *request*
7	error	varchar(100)	Menunjukkan presentase kegagalan saat dilakukan uji beban
8	response_end	varchar(100)	Menunjukkan rata-rata hasil uji *response time* dalam satuan *ms*
9	dom_content_load	varchar(100)	Menunjukkan rata-rata hasil uji waktu memuat *DOM* web dalam satuan *ms*
10	load_event_end	varchar(100)	Menunjukkan rata-rata hasil uji *load time* dalam satuan *ms*

Tabel 4.11: Tabel *summary_results*

No	Kolom	Tipe	Keterangan
11	css_trace_end	varchar(100)	Menunjukkan rata-rata hasil uji waktu memuat *css time* dalam satuan *ms*
12	first_meaningful	varchar(100)	Menunjukkan rata-rata hasil uji waktu memuat konten utama dalam satuan *ms*

4.4.3 Implementasi *Task Queue*

Task queue akan digunakan untuk mengatur antrian *request* dari pengguna, pada tugas akhir ini implementasi *task queue* akan dipasang pada komputer yang sama dengan *web service*. Bahasa pemrograman yang akan digunakan untuk mengimplementasikan *task queue* adalah bahasa pemrograman *Python*, sedangkan untuk basis data akan terkoneksi dengan basis data *MySQL* yang terpasang pada server yang berbeda. Selain itu *task queue* akan dijalankan setiap 1 menit sekali pada *crontab*. Selama *task queue* berjalan algoritmenya akan selalu melakukan pengecekan apakah ada antrian yang bisa dieksekusi. Konfigurasi *crontab* yang digunakan ditunjukkan pada Kode Sumber IV.13. Sedangkan *Pseudocode* yang digunakan untuk mengimplementasikan *task queue* ditunjukkan pada Kode Sumber IV.14.

```
1       * * * * * /usr/bin/python3 queue.py >>
           output.log 2>&1
```

Kode Sumber IV.13: Konfigurasi *crontab*

```
1   Connection
2
3   Variable Declaration
4
5   DECLARE ADDITIONAL FUNCTION
6     Get Data Task Queue from MySQL <- LIMIT 1
7     RETURN data
8
9   MAIN FUNCTION
10    Data = CALL ADDITIONAL FUNCTION
11    IF Data Not NULL
12      Do Task Queue Job
13    Else
14     RETURN Null to Output File
15  END FUNCTION
```

Kode Sumber IV.14: *Pseudocode task queue*

BAB V

PENGUJIAN DAN EVALUASI

Pada bab ini akan dibahas uji coba dan evaluasi dari sistem yang dibuat. Sistem akan diuji coba fungsionalitasnya dengan menjalankan skenario pengujian performa pada web. Uji coba dilakukan untuk mengetahui kinerja sistem dengan lingkungan uji coba yang ditentukan.

5.1 Lingkungan Uji Coba

Lingkungan Uji coba sistem ini terdiri dari beberapa komponen yaitu *web service dan task queue*, server basis data, server *manager node*, dua server *worker*. Server yang digunakan sistem menggunakan layanan *Virtual Private Server* dari DigitalOcean, sedangkan *web service* dan *task queue* akan dibangun di komputer penulis. Spesifikasi untuk setiap komponen ditunjukkan pada Tabel 5.1.

Tabel 5.1: Spesifikasi komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
1	*Web Service & Task Queue*	*Processor* AMD FX-7600P Radeon R7, 4 Core, 8GB RAM, 250GB SSD	Ubuntu 18.04 LTS, Laravel 5.8, Python 3.6
2	Basis Data	1 Core Processor, 1GB RAM, 20GB SSD	Ubuntu 18.04 LTS, MySQL 5.7
3	*Manager Node*	2 *Core Processor*, 4GB RAM, 80GB SSD	Ubuntu 18.04 LTS, Python 3.6, Docker 18.09.6, Node.js 8.15, NPM 6.4.1, Chrome, Puppeteer 0.12.0, MySQL Client 5.7

Tabel 5.1: Spesifikasi komponen

No	Komponen	Perangkat Keras	Perangkat Lunak
4	*Worker* 1	2 *Core Processor*, 4GB RAM, 80GB SSD	Ubuntu 18.04 LTS, Python 3.6, Docker 18.09.6, Node.js 8.15, NPM 6.4.1, Chrome, Puppeteer 0.12.0, MySQL Client 5.7
5	*Worker* 2	2 *Core Processor*, 4GB RAM, 80GB SSD	Ubuntu 18.04 LTS, Python 3.6, Docker 18.09.6, Node.js 8.15, NPM 6.4.1, Chrome, Puppeteer 0.12.0, MySQL Client 5.7

Untuk akses ke setiap komponen, digunakan *ip* publik yang disediakan untuk masing-masing komponen. Detail ditunjukkan pada Tabel 5.2.

Tabel 5.2: *IP* dan *hostname* server

No	Komponen	IP	Hostname
1	*Web Service & Task Queue*	10.151.253.110	night
2	Basis Data	178.128.123.143	NIGHT
3	*Manager Node*	167.71.194.235	CLOUD
4	*Worker Node* 1	165.22.55.82	RAIN
5	*Worker Node* 2	167.71.194.233	STORM

5.2 Skenario Uji Coba

Uji coba ini dilakukan untuk menguji apakah fungsionalitas yang diidentifikasikan terhadap kebutuhan sistem benar-benar telah diimplementasikan dan bekerja seperti yang seharusnya. Skenario pengujian dibedakan menjadi 2 bagian yaitu:

- **Uji Fungsionalitas**
  Pengujian yang dilakukan didasarkan pada fungsionalitas yang disajikan sistem.
- **Uji Performa**
  Pengujian ini dilakukan untuk mengetahui ketersediaan atau penggunaan sumberdaya *CPU* dan *RAM* terhadap jumlah *load generator* yang dipasang pada sistem.

5.2.1 Skenario Uji Fungsionalitas

Uji fungsionalitas dibagi menjadi beberapa bagian antara lain yaitu *user* mengelola skenario melalui web, *user* mengirim request uji beban melalui web, penggunaan *task queue* terhadap *request user*, pengambil data uji beban, *user* melihat hasil uji beban melalui web.

5.2.1.1 Uji Fungsionalitas *User* Mengelola Skenario

Uji coba ini dilakukan dengan mengakses sistem melalui rute `/skenario`. Pengguna akan mengirimkan *http request* kepada *web service* yang telah disediakan. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.3.

Tabel 5.3: Skenario uji fungsionalitas *user* mengelola skenario

No	Rute	Uji Coba	Harapan
1	/skenario	Mengirimkan *request* menuju rute *web service* melalui *browser*	*Request* berhasil diterima oleh *web service*, kemudian *web service* mengirimkan umpan balik berupa data skenario dari pengguna yang ditampilkan di *browser*
2	/skenario/create	Mengirimkan *request* menuju rute *web service* melalui *browser*	*Request* berhasil diterima oleh *web service* dan halaman untuk membuat skenario ditampilkan di *browser*
3	/skenario	Mengirimkan *request* pembuatan skenario menuju rute *web service* melalui *browser* menggunakan metode *POST*	*Web service* berhasil menyimpan data skenario di basis data dan di setiap *node host* dalam bentuk file konfig

Tabel 5.3: Skenario uji fungsionalitas *user* mengelola skenario

No	Rute	Uji Coba	Harapan
4	/skenario/{id}	Mengirimkan *request* untuk menghapus skenario menuju rute *web service* melalui *browser* menggunakan metode *POST*	*Web service* berhasil menghapus data skenario di basis data dan di setiap *node host*

5.2.1.2 Uji Fungsionalitas *User* Mengirim Request Uji Beban

Uji coba ini dilakukan dengan mengakses sistem melalui rute `/worker`. Pengguna akan mengirimkan *http request* kepada *web service* yang telah disediakan. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.4.

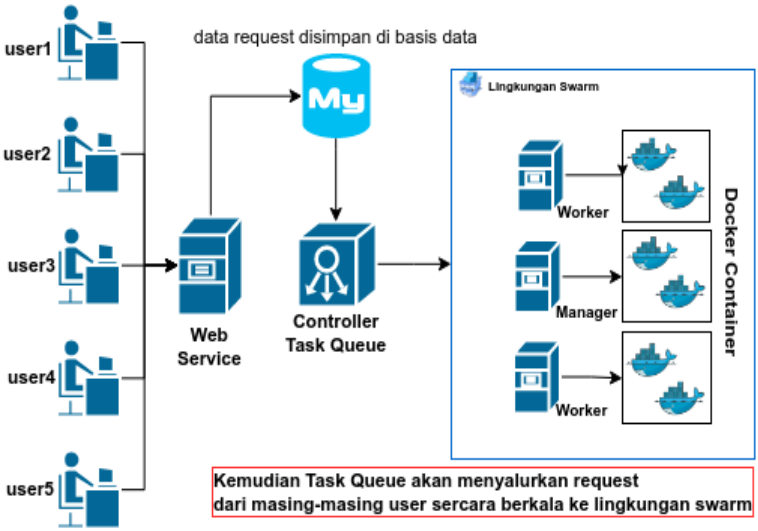
Tabel 5.4: Skenario uji fungsionalitas *user* mengirim request uji beban

No	Rute	Uji Coba	Harapan
1	/worker	Mengirimkan *request* menuju rute *web service* melalui *browser*	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman untuk mengatur jumlah *worker(load generator)* di *browser*

Tabel 5.4: Skenario uji fungsionalitas *user* mengirim request uji beban

No	Rute	Uji Coba	Harapan
2	/worker	Mengirimkan *request* menuju rute *web service* melalui *browser* menggunakan metode *POST*	*Web service* berhasil menyimpan data *worker* dan membuat antrian *request* di dalam basis data

5.2.1.3 Uji Fungsionalitas Penggunaan *Task Queue* Terhadap *Request User*



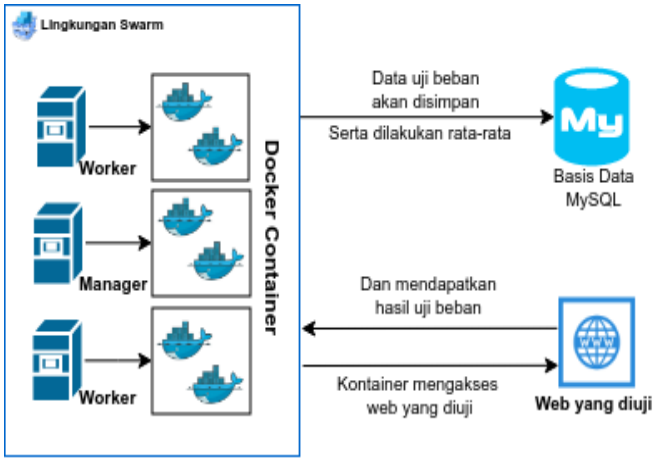
Gambar 5.1: Uji fungsionalitas penggunaan *task queue* terhadap *request user*

Uji coba ini dilakukan dengan cara menjalankan *script* yang menggunakan bahasa pemrograman *Python* pada *crontab* yang

ada di *linux* dan dijadwalkan setiap 1 menit. Setiap kali pengguna mengirim *request* uji beban melalui web, akan disimpan di basis data *MySQL*. Ketika ada daftar antrian yang ada di basis data *MySQL*, *script* akan mengeksekusi hanya satu *request* yang memiliki *timestamp* paling awal dari beberapa *request* yang lain dan akan dilakukan pengujian terhadap skenario yang dikirim. Setelah pengujian skenario selesai, status akan diubah menjadi *done* dan *script* akan mengeksekusi *request* yang lain satu-persatu. Pada pengujian ini akan dilakukan oleh 5 pengguna yang akan melakukan *request* uji beban. Gambaran pengujian ditunjukkan pada Gambar 5.1.

5.2.1.4 Uji Fungsionalitas Pengambil Data Uji Beban

Uji coba ini dilakukan dengan cara menjalankan *script puppeteer* pada setiap kontainer. *script puppeteer* akan dijalankan ketika ada *request* dari pengguna melalui *web service* yang disediakan sistem. Gambaran pengujian ditunjukkan pada Gambar 5.2.



Gambar 5.2: Uji fungsionalitas pengambil data uji beban

5.2.1.5 Uji Fungsionalitas *User* Melihat Hasil Uji Beban

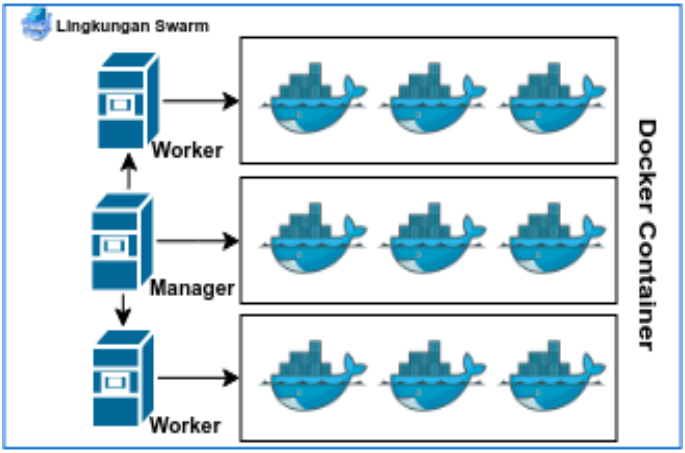
Uji coba ini dilakukan dengan mengakses sistem melalui rute `/hasil`. Pengguna akan mengirimkan *http request* kepada *web service* yang telah disediakan. Rancangan pengujian dan hasil yang diharapkan ditunjukkan pada Tabel 5.5.

Tabel 5.5: Skenario uji fungsionalitas *user* melihat hasil uji beban

No	Rute	Uji Coba	Harapan
1	/hasil/rata-rata	Mengirimkan *request* menuju rute *web service* melalui *browser*	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan hasil perhitungan rata-rata untuk hasil uji beban
2	/hasil/error-console	Mengirimkan *request* menuju rute *web service* melalui *browser*	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan daftar *error console* yang terekam pada *browser*
3	/hasil/images	Mengirimkan *request* menuju rute *web service* melalui *browser*	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan hasil tangkapan layar web yang diuji pada *browser*

5.2.2 Skenario Uji Performa

Sistem *load generator* dan pengambil data uji beban dibangun pada 3 buah *node host* yang terpasang di lingkungan *swarm*. Sebelum dilakukan uji performa, dilakukan pemasangan *load generator* yang diinisiasi pada terminal *manager node*, kemudian *manager node* akan mendistribusikan kontainer ke setiap *node host* yang tergabung. Setelah selesai, data dari setiap kontainer akan disimpan di basis data. Uji coba akan dilakukan secara bertahap untuk membuat 500 kontainer, status awal kontainer sebelum diberikan *request* akan *sleep*. Setelah itu uji coba performa dilakukan untuk menguji performa sistem terhadap jumlah *load generator* yang dikirimkan oleh pengguna. Pengujian akan dikirimkan oleh pengguna melalui *web service* yang disediakan sistem. Jumlah *load generator* yang akan diuji mulai dari 100, 200, 300, 400 dan 500. Hasil yang diharapkan dari pengujian performa sistem yaitu *CPU* dan *Memory* yang digunakan pada setiap *node host*. Arsitektur pengujian tertera pada Gambar 5.3.



Gambar 5.3: Arsitektur uji performa

5.3 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang sudah dijelaskan pada bab 5.2.

5.3.1 Uji Fungsionalitas

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang sudah dibangun.

5.3.1.1 Uji Fungsionalitas *User* Mengelola Skenario

Uji coba ini dilakukan dengan mengakses sistem melalui rute yang telah ditentukan pada Tabel 5.3. Pengguna akan mengirim http request kepada web service yang telah disediakan. Hasil uji coba seperti tertera pada Tabel 5.6.

Tabel 5.6: Skenario uji fungsionalitas *user* mengelola skenario

No	Rute	Harapan	Hasil
1	/skenario	*Request* berhasil diterima oleh *web service*, kemudian *web service* mengirimkan umpan balik berupa data skenario dari pengguna yang ditampilkan di *browser*	OK
2	/skenario/create	*Request* berhasil diterima oleh *web service* dan halaman untuk membuat skenario ditampilkan di *browser*	OK

Tabel 5.6: Skenario uji fungsionalitas *user* mengelola skenario

No	Rute	Harapan	Hasil
3	/skenario	*Web service* berhasil menyimpan data skenario di basis data dan di setiap *node host* dalam bentuk file konfig	OK
4	/skenario/{id}	*Web service* berhasil menghapus data skenario di basis data dan di setiap *node host*	OK

5.3.1.2 Uji Fungsionalitas *User* Mengirim Request Uji Beban

Uji coba ini dilakukan dengan mengakses sistem melalui rute yang telah ditentukan pada Tabel 5.4. Pengguna akan mengirim http request kepada web service yang telah disediakan. Hasil uji coba seperti tertera pada Tabel 5.7.

Tabel 5.7: Skenario uji fungsionalitas *user* mengirim request uji beban

No	Rute	Harapan	Hasil
1	/worker	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan halaman untuk mengatur jumlah *worker(load generator)* di *browser*	OK

Tabel 5.7: Skenario uji fungsionalitas *user* mengirim request uji beban

No	Rute	Harapan	Hasil
2	/worker	*Web service* berhasil menyimpan data *worker* dan membuat antrian *request* di dalam basis data	OK

5.3.1.3 Uji Fungsionalitas Penggunaan *Task Queue* Terhadap *Request User*

Uji coba ini akan dilakukan oleh 5 user yang melakukan request uji beban melalui *web service* sesuai dengan penjelasan pada bab 5.2.1.3. Keterangan hasil uji yang dilakukan bisa dilihat pada Tabel 5.8.

Tabel 5.8: Hasil penggunaan *task queue* terhadap *request user*

Username	Jumlah	Link	Antrian ke	Hasil
user1	100	https://dev.ppdbsda.net	1	OK
user2	200	https://dev.ppdbsda.net	2	OK
user3	300	https://dev.ppdbsda.net	3	OK
user4	400	https://dev.ppdbsda.net	4	OK
user5	500	https://dev.ppdbsda.net	5	OK

5.3.1.4 Uji Fungsionalitas Pengambil Data Uji Beban

Uji coba ini akan dilakukan oleh *Puppeteer* yang terpasang pada *node host* di lingkungan *swarm*. Skenario pengambil data uji beban sesuai dengan *request task queue* pada Tabel 5.8. Hasil dari pengujian ini dapat dilihat pada Tabel 5.9.

Tabel 5.9: Hasil pengambil data uji beban

Username	Response End	CSS Tracing End	DOM Content Loaded	First Meaningful Paint	Load Event End
user1	499,17	652,53	1592,35	1636,69	2001,26
user2	503,88	655,09	1589,95	1660,26	2010,12
user3	510,26	660,09	1567,26	1690,82	2084,83
user4	519,98	661,13	1598,62	1699,87	2188,10
user5	523,56	665,61	1599,40	1685,76	2190,45

Selain itu akan didapatkan perhitungan presentase *error* ketika terjadinya *timeout* saat mengakses web yang diuji atau hasil *performance metrics* pada Tabel 5.9 terdapat hasil kurang dari 0 atau minus.

5.3.1.5 Uji Fungsionalitas *User* Melihat Hasil Uji Beban

Uji coba ini dilakukan dengan mengakses sistem melalui rute yang telah ditentukan pada Tabel 5.5. Pengguna akan mengirim *http request* kepada *web service* yang telah disediakan. Hasil uji coba seperti tertera pada Tabel 5.10.

Tabel 5.10: Skenario uji fungsionalitas *user* melihat hasil uji beban

No	Rute	Uji Coba	Hasil
1	/hasil/rata-rata	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan hasil perhitungan rata-rata untuk hasil uji beban	OK

Tabel 5.10: Skenario uji fungsionalitas *user* melihat hasil uji beban

No	Rute	Uji Coba	Hasil
2	/hasil/error-console	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan daftar *error console* yang terekam pada *browser*	OK
3	/hasil/images	*Request* berhasil diterima oleh *web service*, kemudian *web service* menampilkan hasil tangkapan layar web yang diuji pada *browser*	OK

5.3.2 Uji Performa

Seperti yang dijelaskan pada bab 5.2.2 pengujian performa akan dilakukan pada 3 *node host* yang terpasang di lingkungan *swarm*. Pertama akan dilakukan pembuatan *load generator(worker)* terlebih dahulu, kemudian akan dilakukan uji performa sistem ketika menerima layanan uji beban.

5.3.2.1 Uji Pembuatan *Load Generator*

Kondisi awal ketersediaan sumberdaya sebelum dilakukan pembuatan *load generator* pada masing-masing *node host* ditunjukkan pada Tabel 5.11.

Tabel 5.11: Kondisi awal ketersediaan sumberdaya sebelum pembuatan

No	Hostname	CPU	RAM	Storage
1	CLOUD	99,70%	287M/3.9G	74/78G
2	RAIN	99,64%	256M/3.9G	74/78G

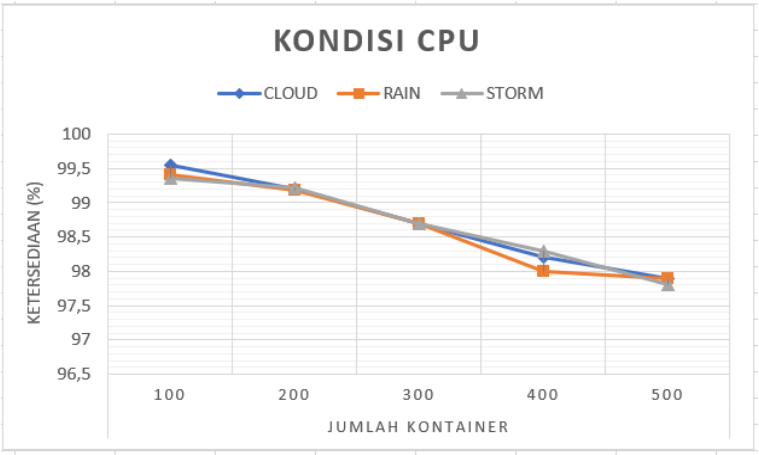
Tabel 5.11: Kondisi awal ketersediaan sumberdaya sebelum pembuatan

No	Hostname	CPU	RAM	Storage
3	STORM	99,69%	255M/3.9G	74/78G

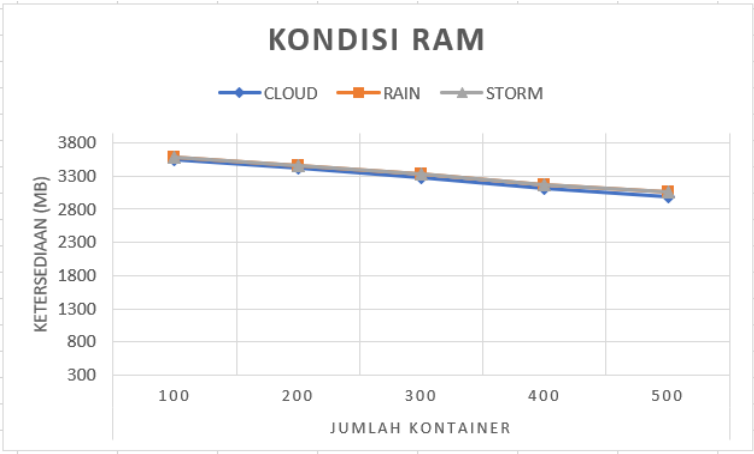
Setelah dilakukan uji coba untuk membuat kontainer yang akan digunakan sebagai *load generator*, ketersediaan sumberdaya *CPU* dan *RAM* ditunjukkan pada Tabel 5.12 serta grafik kondisi ketersediaan *CPU* setelah pemasangan pada Gambar 5.4 dan kondisi ketersediaan *RAM* setelah pemasangan pada Gambar 5.5.

Tabel 5.12: Kondisi ketersediaan sumberdaya setelah pembuatan

No	Hostname	CPU	RAM
1	CLOUD	97,9%	950M/3,9G
2	RAIN	97,9%	892M/3,9G
3	STORM	97,8%	889M/3,9G



Gambar 5.4: Kondisi ketersediaan sumberdaya *CPU* setelah pembuatan



Gambar 5.5: Kondisi ketersediaan sumberdaya *RAM* setelah pembuatan

Dari grafik tersebut dapat dilihat kebutuhan sumberdaya *CPU* dan *RAM* tidak mengalami perbedaan yang banyak dan hanya berubah secara *linear*. *CPU* tidak terlalu banyak berubah setelah pembuatan load generator dikarenakan semua *load generator* di atur untuk *sleep* sebelum ada *request* uji beban. Namun ketika pembuatan *load generator* dan pendistribusiannya, tentu saja penggunaan *CPU* bisa sampai 100% dan untuk penggunaan *RAM* adalah 1,3MB untuk setiap kontainernya. Pada tugas akhir ini, server yang digunakan hanya memiliki spesifikasi 2 *Core CPU*. Adapun masalah-masalah yang dihadapi dalam pengujian ini yaitu ketika membuat *load generator* dengan jumlah diatas 500 secara bersamaan yang akan mengakibatkan *CPU Usage* bisa melebihi kapasitasnya dan hal tersebut akan membuat server *not responding*. Namun untuk penggunaan *RAM* ketika pembuatan masih bisa dikatakan cukup.

Sedangkan hasil distribusi kontainer ke setiap *node host* setelah dilakukan pembuatan *load generator* ditunjukkan pada Tabel 5.13.

Tabel 5.13: Hasil distribusi kontainer setelah pembuatan

No	Jumlah Kontainer	CLOUD	RAIN	STORM
1	100	34	33	33
2	200	66	67	67
3	300	100	100	100
4	400	132	134	134
5	500	166	167	167

Pendistribusian pada Tabel 5.13 diatasi oleh manajer pada Docker Swarm dan didistribusikan secara rata ke setiap node host yang tergabung.

5.3.2.2 Uji Performa Sistem ketika Menerima Request Layanan Uji Beban

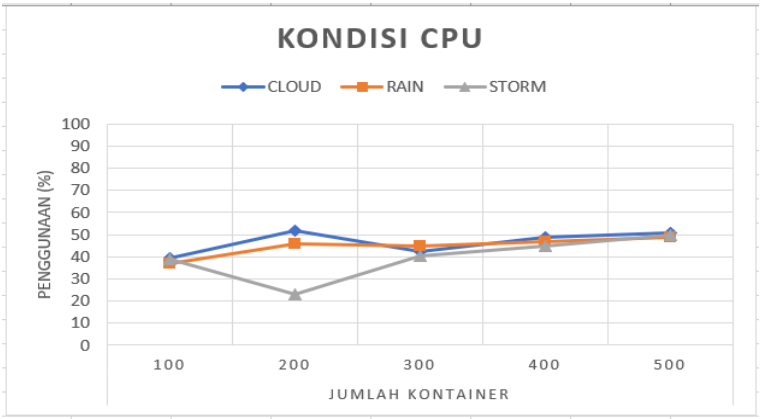
Pengujian ini dilakukan ketika sistem menerima request layanan uji beban, informasi penggunaan sumberdaya ditunjukkan pada Tabel 5.14, dari tabel tersebut didapatkan grafik penggunaan CPU pada Gambar 5.6 dan grafik penggunaan RAM pada Gambar 5.7.

Tabel 5.14: Kondisi penggunaan sumberdaya ketika ada *request*

No	Hostname	Jumlah	CPU	RAM
1	CLOUD	100	39,5%	1,1/3,9G
		200	52%	1,2/3,9G
		300	42,2%	1,3/3,9G
		400	48,7%	1,36/3,9G
		500	50,9%	1,39/3,9G
2	RAIN	100	36,7%	1,02/3,9G
		200	46%	1,18/3,9G
		300	44,9%	1,3/3,9G
		400	46,8%	1,33/3,9G
		500	49%	1,4/3,9G

Tabel 5.14: Kondisi penggunaan sumberdaya ketika ada *request*

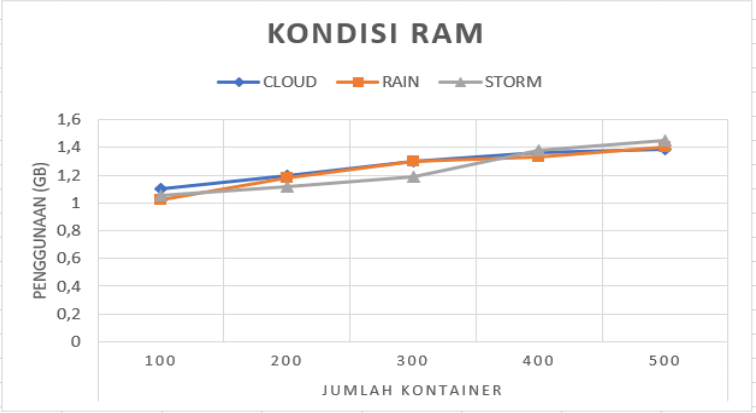
No	Hostname	Jumlah	CPU	RAM
3	STORM	100	38,9%	1,05/3,9G
		200	23%	1,12/3,9G
		300	40,2%	1,19/3,9G
		400	44,9%	1,38/3,9G
		500	49,6%	1,45/3,9G



Gambar 5.6: Kondisi penggunaan sumberdaya *CPU* ketika ada *request*

Dilihat dari grafik pada Gambar 5.6, penggunaan *CPU* mengalami perubahan secara *linear* yaitu semakin banyak *request* uji beban akan semakin tinggi, namun pada saat pengujian ketika jumlah *load generator* 200, terjadi anomali pada ketersediaan sumberdaya, hal tersebut dikarenakan urutan kontainer id yang disimpan pada basis data tidak berimbang untuk nomor urut dari 1 sampai 200, sehingga pada *node host* CLOUD dan RAIN memiliki jumlah *load generator* yang harus dijalankan jauh lebih banyak dari pada *node host* STORM, sehingga mengakibatkan tenjadinya anomali penggunaan *CPU*

pada *node host* STORM.



Gambar 5.7: Kondisi penggunaan sumberdaya *RAM* ketika ada *request*

Sedangkan untuk penggunaan *RAM* pada Gambar 5.7, perubahannya secara linear saja, dikarenakan memang adanya pembatasan berjalannya proses secara bersama dalam setiap waktunya, karena *Chrome* yang dijalankan akan memerlukan sumberdaya yang banyak. Jika tidak dibatasi, dapat mengakibatkan penggunaan *CPU* dan *RAM* akan melebihi kapasitas dan mengakibatkan server menjadi *not responding*. Dapat disimpulkan juga dalam melakukan uji beban menggunakan *Headless Chrome* akan membutuhkan banyak sumberdaya *CPU* dan *RAM*, sedangkan untuk penggunaan *Docker* tidak memerlukan sumberdaya sebanyak *Headless Chrome*.

(Halaman ini sengaja dikosongkan)

BAB VI

PENUTUP

Bab ini membahas kesimpulan yang dapat diambil dari tujuan pembuatan sistem dan hubungannya dengan hasil uji coba dan evaluasi yang telah dilakukan. Selain itu, terdapat beberapa saran yang bisa dijadikan acuan untuk melakukan pengembangan dan penelitian lebih lanjut.

6.1 Kesimpulan

Dari proses perencangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Dalam pengembangan web, uji beban dapat dilakukan menggunakan *Headless Chrome* sebagai *tester*, yang dipadukan dengan *API* pengambil uji beban yaitu *Puppeteer*. Namun ketika *Headless Chrome* dijalankan oleh *load generator*, membutuhkan sumberdaya *CPU* dan *RAM* yang cukup tinggi jika dilakukan secara bersamaan, sehingga harus dilakukan pembatasan terhadap proses yang berjalan secara bersamaan.
2. *Load generator* yang digunakan sebagai *resource user* dapat diimplementasikan oleh *Docker*. *Docker* dipasang menggunakan *Docker Image* yang sudah diinstalasi *Node.js* dan *Puppeteer* didalamnya. Berdasarkan hasil uji coba performa *Docker* sangat mumpuni untuk dijadikan *resource user* karena membutuhkan sedikit sumberdaya dan dapat melakukan akses langsung ke suatu web.
3. Data Laporan uji beban didapatkan menggunakan *Puppeteer* dan disajikan pada web dalam bentuk tabel dengan data inti yaitu *Response End*, *CSS Tracing End*, *Dom Content Loaded*, *First Meaningful Paint* dan *Load Event End*. Laporan yang disajikan juga menampilkan *error console* yang terekam pada *browser* dan tangkapan layar web yang diuji.

4. Dalam menangani layanan uji beban yang dikirimkan oleh pengembang web, sistem menggunakan sebuah *task scheduler crontab* yang menjalankan *script Python* setiap satu menit dan melakukan pengecekan secara berkala.

6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

- Melakukan optimasi terhadap penggunaan *CPU* dan *RAM* supaya hasil uji lebih baik, misalnya menggunakan *multithreading* untuk mengatur proses ketika dijalankan.
- Memadukan *Headless Chrome* dan *Puppeteer* dengan *tools load test* yang lain misalnya *K6*, *Jmeter Rest API*, *Selenium Web Driver* atau *Firefox Headless Mode* untuk mendapatkan data uji beban.
- Untuk menangani permintaan jumlah *load generator* yang tinggi, server yang digunakan harus dilakukan *scaling* secara horizontal maupun vertikal, supaya dapat menunjang sistem untuk melayani uji beban.
- Pengembang bisa mempercantik tampilan antarmuka pengguna agar lebih nyaman untuk digunakan dan mengoptimasi layanan uji beban pada metode *POST*.

DAFTAR PUSTAKA

[1] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," in *IEEE Internet Computing*, vol. 1, 2014, hal. 81–84.

[2] "Get started, part 4: Swarms," 2019, 24 Mei 2019. [Daring]. Tersedia pada: https://docs.docker.com/get-started/part4/. [Diakses: 24 Mei 2019].

[3] "Puppeteer," 2019, 09 April 2019. [Daring]. Tersedia pada: https://pptr.dev/. [Diakses: 09 April 2019].

[4] R. Roemer, *Backbone.js Testing*. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK: Packt Publishing Ltd, 7 2013, hal. 141–142.

[5] "Getting started with headless chrome," 10 Mei 2019. [Daring]. Tersedia pada: https://developers.google.com/web/updates/2017/04/headless-chrome. [Diakses: 10 Mei 2019].

[6] Joyent, "About node.js," 1 Juni 2019. [Daring]. Tersedia pada: https://nodejs.org/en/about/. [Diakses: 1 Juni 2019].

[7] S. Tilkov dan S. Vinoski, "Node.js: Using javascript to build high-performance network programs," in *IEEE Internet Computing*, vol. 14, 2010, hal. 80–83.

[8] "What is docker?" 2018, 01 Desember 2018. [Daring]. Tersedia pada: https://www.docker.com. [Diakses: 01 Desember 2018].

[9] "Swarm mode key concepts," 2019, 24 Mei 2019. [Daring]. Tersedia pada: https://docs.docker.com/engine/swarm/key-concepts/. [Diakses: 24 Mei 2019].

[10] "Raft consensus in swarm mode," 2019, 24 Mei 2019. [Daring]. Tersedia pada: https://docs.docker.com/engine/swarm/raft/. [Diakses: 24 Mei 2019].

[11] M. Arif, A. Dentha, dan H. W. S. Sindung, "Designing internship monitoring system web based with laravel framework," in *2017 IEEE International Conference on Communication, Networks and Satellite (Comnetsat)*, 2017.

[12] "What is python? executive summary," 2019, 20 Juni 2019. [Daring]. Tersedia pada: https://www.python.org/doc/essays/blurb/. [Diakses: 20 Juni 2019].

LAMPIRAN A

INSTALASI PERANGKAT LUNAK

Instalasi *Docker*

Untuk melakukan instalasi *Docker*, dilakukan seperti langkah-langkah berikut:

```
$ sudo apt-get -y install \
apt-transport-https \
ca-certificates \
curl

$ curl -fsSL https://download.docker.com/linux/
ubuntu/gpg | sudo apt-key add -

$ sudo add-apt-repository \
"deb [arch=amd64] https://download.docker.com/
   linux/ubuntu \
$(lsb_release -cs) \
stable"

$ sudo apt-get update

$ sudo apt-get install docker-ce docker-ce-cli
   containerd.io
```

Kode Sumber A.1: Perintah instalasi Docker

Setelah menjalankan perintah pada kode sumber A.1. Jalankan perintah berikut agar *Docker* bisa dijalankan sebagai *Non-Root User*.

```
$ sudo groupadd docker
$ sudo usermod -aG docker $USER
```

Kode Sumber A.2: Perintah mengubah hak User

Instalasi *Docker Compose*

Docker Compose digunakan untuk otomasi dalam menjalankan *Dockerfile* dan berperan untuk pembuatan Docker Image. Instalasi *Docker Compose* dapat dilihat pada kode sumber A.3.

```
$ sudo curl -L "https://github.com/docker/compose
    /releases/download/1.24.0/docker-compose-$(
    uname -s)-$(uname -m)" -o /usr/local/bin/
    docker-compose

$ sudo chmod +x /usr/local/bin/docker-compose

$ sudo ln -s /usr/local/bin/docker-compose /usr/
    bin/docker-compose
```

Kode Sumber A.3: Perintah instalasi Docker Compose

Instalasi *Linux Package*

Beberapa *package* yang dibutuhkan dalam pembuatan sistem.

- Node.js
  ```
  $ sudo apt install nodejs
  ```
- NPM
  ```
  $ sudo apt install npm
  ```
- MySQL
  ```
  $ sudo apt install mysql-server mysql-client
  ```
- PIP
  ```
  $ sudo apt install python3-pip
  ```

- mysql-connector-python

```
$ pip3 install mysql-connector-python
```

- Composer

```
$ php -r
"copy('https://getcomposer.org/installer',
'composer-setup.php');"

$ php -r "if (hash_file('sha384',
'composer-setup.php') ===
'48e3236262b34d30969dca3c37281b3b4bbe3221bda826ac6a9a62
echo 'Installer verified';  else  echo
'Installer corrupt';
unlink('composer-setup.php');  echo
PHP_EOL;"

$ php composer-setup.php
$ php -r "unlink('composer-setup.php');"
$ mv composer.phar /usr/local/bin/composer
```

- Laravel

```
$ composer global require laravel/installer
```

(Halaman ini sengaja dikosongkan)

LAMPIRAN B

KODE SUMBER

Kode Sumber Pengambilan Data *Metrics Performance*

Isi berkas index.js

```
1  const puppeteer = require('puppeteer');
2  const testPage = require('./testPage');
3  const fs = require('fs');
4  const db = require('../config/databases');
5  const scenario_id = process.argv[2];
6  const counter = process.argv[3];
7  const worker = process.argv[4];
8  const host = process.argv[5];
9
10 let rawdata = fs.readFileSync('/app/code/assets/
       config_' + scenario_id + '.json');
11 let config = JSON.parse(rawdata);
12 (async () => {
13   const browser = await puppeteer.launch({ args:
         ['--no-sandbox'] });
14   const page = await browser.newPage();
15   await page.on('console', msg =>
16     db.query('INSERT INTO errors (scenario_id,
           link, worker, username, host, type, text,
           location_url) VALUES (?, ?, ?, ?, ?, ?, ?,
            ?)',
17     [scenario_id, config.scenario_link, worker,
           config.username, host, msg._type, msg.
           _text, msg._location.url])
18   );
19   try {
20     let data = await testPage(page, config,
           counter);
```

```
21      db.query('INSERT INTO results (scenario_id,
            link, method, worker, username, host,
            response_end, dom_content_load,
            load_event_end, css_trace_end,
            first_meaningful) VALUES (?, ?, ?, ?, ?, ?
            , ?, ?, ?, ?, ?)',
22          [scenario_id, config.scenario_link, config.
                scenario_method, worker, config.username
                , host, data.Timing.responseEnd, data.
                Timing.domContentLoadedEventEnd, data.
                Timing.loadEventEnd, data.TraceResult.
                cssEnd, data.Metrics.
                FirstMeaningfulPaint]);
23      db.end();
24      await page.screenshot({ path: '/app/output/ss
            ' + scenario_id + '.png' });
25      await browser.close();
26    } catch (error) {
27      console.error(error);
28      db.query('INSERT INTO results (scenario_id,
            link, method, worker, username, host,
            response_end, dom_content_load,
            load_event_end, css_trace_end,
            first_meaningful) VALUES (?, ?, ?, ?, ?, ?
            , ?, ?, ?, ?, ?)',
29          [scenario_id, config.scenario_link, config.
                scenario_method, worker, config.username
                , host, -1, -1, -1, -1, -1]);
30      db.end();
31      await page.screenshot({ path: '/app/output/ss
            ' + scenario_id + '.png' });
32      await browser.close();
33    }
```

```
34 })();
```

Kode Sumber B.1: Isi berkas index.js

Isi berkas testPage.js

```
1  const {
2    getTimeFromPerformanceMetrics,
3    extractDataFromPerformanceMetrics,
4    extractDataFromPerformanceTiming,
5    extractDataFromTracing,
6  } = require('./helpers');
7
8  async function testPage(page, config, counter) {
9    const client = await page.target().
         createCDPSession();
10   await client.send('Performance.enable');
11   const navigationStart =
         getTimeFromPerformanceMetrics(
12     await client.send('Performance.getMetrics'),
13     'NavigationStart'
14   );
15
16   await page.tracing.start({ path: './trace' +
         config.scenario_id + counter + '.json' });
17
18   await page.goto(config.scenario_link);
19
20   const performanceTiming = JSON.parse(
21     await page.evaluate(() => JSON.stringify(
           window.performance.timing))
22   );
23
```

```
24    let firstMeaningfulPaint = 0;
25    while (firstMeaningfulPaint === 0) {
26      await page.waitFor(300);
27      performanceMetrics = await client.send('
          Performance.getMetrics');
28      firstMeaningfulPaint =
          getTimeFromPerformanceMetrics(
29        performanceMetrics, 'FirstMeaningfulPaint'
30      );
31    }
32
33    await page.tracing.stop();
34
35    const cssTracing = await extractDataFromTracing
        (
36      './trace' + config.scenario_id + counter + '.
          json',
37      config.scenario_link,
38    );
39
40    let TraceResult = {
41      cssEnd: cssTracing.end - navigationStart,
42    }
43
44    let Metrics = extractDataFromPerformanceMetrics
        (
45      performanceMetrics,
46      'FirstMeaningfulPaint',
47    );
48
49    let Timing = extractDataFromPerformanceTiming(
50    performanceTiming,
51      'responseEnd',
```

```
    'domContentLoadedEventEnd',
    'loadEventEnd',
  );

  return { TraceResult, Metrics, Timing };
}

module.exports = testPage;
```

Kode Sumber B.2: Isi berkas testPage.js

Isi berkas helpers.js

```
const fs = require('fs');

const getTimeFromPerformanceMetrics = (metrics, name) =>
  metrics.metrics.find(x => x.name === name).value * 1000;

const extractDataFromPerformanceMetrics = (metrics, ...dataNames) => {
  const navigationStart = getTimeFromPerformanceMetrics(
    metrics,
    'NavigationStart'
  );
  const extractedData = {};
  dataNames.forEach(name => {
    extractedData[name] = getTimeFromPerformanceMetrics(metrics, name) - navigationStart;
  });
```

```
16    return extractedData;
17  };
18
19  const extractDataFromPerformanceTiming = (timing,
        ...dataNames) => {
20    const navStart = timing.navigationStart;
21
22    const extractedData = {};
23    dataNames.forEach(name => {
24      extractedData[name] = timing[name] - navStart
          ;
25    });
26
27    return extractedData;
28  };
29
30  const extractDataFromTracing = (path, link) =>
      new Promise(resolve => {
31    const tracing = JSON.parse(fs.readFileSync(path
        , 'utf8'));
32    const resourceTracings = tracing.traceEvents.
        filter(
33      x =>
34        x.cat === 'devtools.timeline' &&
35        typeof x.args.data !== 'undefined' &&
36        typeof x.args.data.url !== 'undefined' &&
37        x.args.data.url.includes(link)
38    );
39    const resourceTracingSendRequest =
        resourceTracings.find(
40      x => x.name === 'ResourceSendRequest'
41    );
```

```
42    const resourceId = resourceTracingSendRequest.
          args.data.requestId;
43    const resourceTracingEnd = tracing.traceEvents.
          filter(
44      x =>
45        x.cat === 'devtools.timeline' &&
46        typeof x.args.data !== 'undefined' &&
47        typeof x.args.data.requestId !== 'undefined
              ' &&
48        x.args.data.requestId === resourceId
49    );
50    const resourceTracingStartTime =
          resourceTracingSendRequest.ts / 1000;
51    const resourceTracingEndTime =
52      resourceTracingEnd.find(x => x.name === '
            ResourceFinish').ts / 1000;
53
54    fs.unlink(path, () => {
55      resolve({
56        end: resourceTracingEndTime,
57      });
58    });
59  });
60
61  module.exports = {
62    getTimeFromPerformanceMetrics,
63    extractDataFromPerformanceMetrics,
64    extractDataFromPerformanceTiming,
65    extractDataFromTracing,
66  };
```

Kode Sumber B.3: Isi berkas helpers.js

Kode Sumber Basis Data *MySQL*

```
CREATE TABLE `swarms` (
  `id` bigint(20) unsigned NOT NULL
     AUTO_INCREMENT,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  `swarm_ip` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  `swarm_username` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  `swarm_password` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  `is_used` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `swarms_swarm_ip_unique` (`swarm_ip
     `)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=
    utf8mb4 COLLATE=utf8mb4_unicode_ci;

CREATE TABLE `containers` (
  `id` bigint(20) unsigned NOT NULL
     AUTO_INCREMENT,
  `task_id` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  `node_id` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  `container_id` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
  `node_ip` varchar(255) COLLATE utf8mb4
     _unicode_ci NOT NULL,
```

```
19    `node_host` varchar(255) COLLATE utf8mb4
          _unicode_ci NOT NULL,
20    `status` smallint(6) NOT NULL DEFAULT '0',
21    `username` varchar(100) COLLATE utf8mb4
          _unicode_ci NOT NULL DEFAULT '0',
22    PRIMARY KEY (`id`)
23  ) ENGINE=InnoDB AUTO_INCREMENT=101 DEFAULT
       CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
24
25  CREATE TABLE `users` (
26    `id` bigint(20) unsigned NOT NULL
          AUTO_INCREMENT,
27    `name` varchar(255) COLLATE utf8mb4_unicode_ci
          NOT NULL,
28    `email` varchar(255) COLLATE utf8mb4_unicode_ci
           NOT NULL,
29    `username` varchar(255) COLLATE utf8mb4
          _unicode_ci NOT NULL,
30    `email_verified_at` timestamp NULL DEFAULT NULL
          ,
31    `password` varchar(255) COLLATE utf8mb4
          _unicode_ci NOT NULL,
32    `remember_token` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
33    `created_at` timestamp NULL DEFAULT NULL,
34    `updated_at` timestamp NULL DEFAULT NULL,
35    `request_test` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
36    PRIMARY KEY (`id`),
37    UNIQUE KEY `users_email_unique` (`email`),
38    UNIQUE KEY `users_username_unique` (`username`)
39  ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=
       utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

```
40
41 CREATE TABLE `role_user` (
42   `id` bigint(20) unsigned NOT NULL
       AUTO_INCREMENT,
43   `created_at` timestamp NULL DEFAULT NULL,
44   `updated_at` timestamp NULL DEFAULT NULL,
45   `role_id` int(10) unsigned NOT NULL,
46   `user_id` int(10) unsigned NOT NULL,
47   PRIMARY KEY (`id`)
48 ) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=
     utf8mb4 COLLATE=utf8mb4_unicode_ci;
49
50 CREATE TABLE `roles` (
51   `id` bigint(20) unsigned NOT NULL
       AUTO_INCREMENT,
52   `created_at` timestamp NULL DEFAULT NULL,
53   `updated_at` timestamp NULL DEFAULT NULL,
54   `name` varchar(255) COLLATE utf8mb4_unicode_ci
       NOT NULL,
55   `description` varchar(255) COLLATE utf8mb4
       _unicode_ci NOT NULL,
56   PRIMARY KEY (`id`)
57 ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=
     utf8mb4 COLLATE=utf8mb4_unicode_ci;
58
59 CREATE TABLE `scenarios` (
60   `id` bigint(20) unsigned NOT NULL
       AUTO_INCREMENT,
61   `created_at` timestamp NULL DEFAULT NULL,
62   `updated_at` timestamp NULL DEFAULT NULL,
63   `scenario_id` varchar(255) COLLATE utf8mb4
       _unicode_ci NOT NULL,
```

```
64    `username` varchar(255) COLLATE utf8mb4
          _unicode_ci NOT NULL,
65    `scenario_method` varchar(255) COLLATE utf8mb4
          _unicode_ci NOT NULL,
66    `scenario_link` varchar(255) COLLATE utf8mb4
          _unicode_ci NOT NULL,
67    `scenario_worker` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
68    `scenario_status` smallint(6) NOT NULL DEFAULT
          '0',
69    `scenario_button` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
70    PRIMARY KEY (`id`)
71  ) ENGINE=InnoDB AUTO_INCREMENT=4 DEFAULT CHARSET=
        utf8mb4 COLLATE=utf8mb4_unicode_ci;
72
73  CREATE TABLE `queues` (
74    `id` bigint(20) unsigned NOT NULL
          AUTO_INCREMENT,
75    `created_at` timestamp NULL DEFAULT NULL,
76    `updated_at` timestamp NULL DEFAULT NULL,
77    `username` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
78    `worker` int(11) DEFAULT NULL,
79    `status` smallint(6) DEFAULT NULL,
80    PRIMARY KEY (`id`)
81  ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=
        utf8mb4 COLLATE=utf8mb4_unicode_ci;
82
83  CREATE TABLE `results` (
84    `id` bigint(20) unsigned NOT NULL
          AUTO_INCREMENT,
```

```
85    `scenario_id` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
86    `link` varchar(255) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
87    `method` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
88    `worker` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
89    `username` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
90    `host` varchar(100) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
91    `response_end` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
92    `dom_content_load` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
93    `load_event_end` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
94    `css_trace_end` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
95    `first_meaningful` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
96    `status` smallint(6) DEFAULT '0',
97    PRIMARY KEY (`id`)
98  ) ENGINE=InnoDB AUTO_INCREMENT=126 DEFAULT
        CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
99
100 CREATE TABLE `errors` (
101   `id` bigint(20) unsigned NOT NULL
          AUTO_INCREMENT,
102   `scenario_id` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
```

```
103   `link` varchar(255) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
104   `worker` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
105   `username` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
106   `host` varchar(100) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
107   `type` varchar(100) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
108   `text` varchar(255) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
109   `args` varchar(255) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
110   `location_url` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
111   PRIMARY KEY (`id`)
112 ) ENGINE=InnoDB AUTO_INCREMENT=126 DEFAULT
       CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
113
114 CREATE TABLE `summary_results` (
115   `id` bigint(20) unsigned NOT NULL
          AUTO_INCREMENT,
116   `scenario_id` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
117   `link` varchar(255) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
118   `method` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
119   `worker` varchar(255) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
120   `username` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
```

```
121   `error` varchar(100) COLLATE utf8mb4_unicode_ci
          DEFAULT NULL,
122   `response_end` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
123   `dom_content_load` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
124   `load_event_end` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
125   `css_trace_end` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
126   `first_meaningful` varchar(100) COLLATE utf8mb4
          _unicode_ci DEFAULT NULL,
127   PRIMARY KEY (`id`)
128 ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=
      utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

Kode Sumber B.4: Basis data MySQL

Kode Sumber *Task Queue*

```
1  import mysql.connector
2  from mysql.connector import Error
3
4  def connect():
5    connection = mysql.connector.connect(
6      host="178.128.123.143",
7      user="cloudy",
8      passwd="sembarang12",
9      database="tugas_akhir_2019"
10   )
11   return connection
```

Kode Sumber B.5: Isi berkas connection.py

```
from connection import connect
from sys import argv
import os
import subprocess
from pathlib import Path

def checkDb():
  con = connect()
  cursor = con.cursor()

  # get antrian request
  getQueueStatus = ('SELECT * FROM queues WHERE
      status < 2 ORDER BY created_at LIMIT 1')
  cursor.execute(getQueueStatus)
  result = cursor.fetchone()
  count = cursor.rowcount
  return count

def getQueue():
  # connection
  con = connect()
  cursor = con.cursor()

  # get antrian request
  getQueueStatus = ('SELECT * FROM queues WHERE
      status < 2 ORDER BY created_at LIMIT 1')
  cursor.execute(getQueueStatus)
  result = cursor.fetchone()
  id_queue = result[0]
  username = result[3]
  status = result[5]

```

```
31    # update status antrian menjadi 1 atau proses
32    if status==0:
33      updateQueueStatus = ('UPDATE queues SET
            status=1 WHERE username=%s AND id=%s')
34      cursor.execute(updateQueueStatus,(username,
            id_queue,))
35      con.commit()
36
37  def runScenario():
38    # connection
39    con = connect()
40    cursor = con.cursor()
41
42    # get queue status = 1
43    getQueueStatus = ('SELECT * FROM queues WHERE
          status = 1 ORDER BY created_at LIMIT 1')
44    cursor.execute(getQueueStatus)
45    result = cursor.fetchone()
46    count = cursor.rowcount
47    if count > 0:
48      id_queue = result[0]
49      username = result[3]
50      worker = result[4]
51
52      # update status kontainer
53      updateContainerStatus = ('UPDATE containers
            SET status = 1, username = %s WHERE status
             = 0 LIMIT %s')
54      cursor.execute(updateContainerStatus,(
            username,worker,))
55      con.commit()
56
57      # get info swarm untuk sshpass
```

```
58      getSwarmConnection = ('SELECT DISTINCT(c.
            node_ip), s.swarm_username, s.
            swarm_password, c.username,COUNT(c.node_ip
            ) cc \
59        FROM containers c JOIN swarms s \
60        WHERE s.swarm_ip = c.node_ip AND c.status =
              1 AND c.username = %s \
61        GROUP BY c.node_ip HAVING cc > 1')
62      cursor.execute(getSwarmConnection,(username,)
            )
63      swarms = cursor.fetchall()
64
65      # run sshpass untuk eksekusi skenario
66      print('process...')
67      path='/home/cloudy/tugas-akhir-2019/
            implementasi/scripts/run_scenario.py'
68      for swarm in swarms:
69        ip = swarm[0]
70        user = swarm[1]
71        passw = swarm[2]
72        command_line = ('sshpass -p %s ssh -o
              UserKnownHostsFile=/dev/null -o
              StrictHostKeyChecking=no %s@%s' % (passw
              , user, ip))
73        cmd = ('python3 %s %s %s' % (path, username
              , ip))
74        run = command_line.split()
75        run.append(cmd)
76        subprocess.Popen(run, stdout=subprocess.
              PIPE, stderr=open(os.devnull, 'w'))
77
78      # update status running
79      updateQueueStatus = ('UPDATE queues SET
```

```
        status = -1 WHERE username=%s AND id=%s')
80      cursor.execute(updateQueueStatus,(username,
            id_queue,))
81      con.commit()
82
83  def calculateResult():
84    # connection
85    con = connect()
86    cursor = con.cursor()
87
88    # get queue request is done
89    getQueueStatus = ('SELECT * FROM queues WHERE
          status = -1 ORDER BY created_at LIMIT 1')
90    cursor.execute(getQueueStatus)
91    queue = cursor.fetchone()
92    username = queue[3]
93
94    # get count result
95    getCountResult = ('SELECT r.scenario_id, s.
          scenario_worker, count(r.scenario_id) AS jml
           \
96    FROM results r JOIN scenarios s ON s.
          scenario_id = r.scenario_id \
97    WHERE r.username=%s AND r.status = 0 GROUP BY r
          .scenario_id, s.scenario_worker')
98    cursor.execute(getCountResult,(username,))
99    result = cursor.fetchone()
100   check = cursor.rowcount
101   if check > 0:
102     scena_id = result[0]
103     worker = int(result[1])
104     hasil = result[2]
105     if worker == hasil:
```

```
106         getAvgResult = ('SELECT t.scenario_id, s.
                scenario_link, s.scenario_method, s.
                scenario_worker ,t.username, \
107           (s.scenario_worker - count(t.scenario_id)
                )/s.scenario_worker*100 AS error, \
108           AVG(t.response_end) AS response_end, \
109           AVG(t.dom_content_load) AS
                dom_content_load, \
110           AVG(t.load_event_end) AS load_event_end,
                \
111           AVG(t.css_trace_end) AS css_trace_end, \
112           AVG(t.first_meaningful) AS
                first_meaningful \
113           FROM scenarios s INNER JOIN results t ON
                t.scenario_id = s.scenario_id \
114           WHERE s.username = %s AND load_event_end
                > 0 AND s.scenario_id \
115           NOT IN (SELECT scenario_id FROM
                summary_results) \
116           GROUP BY scenario_id, s.scenario_worker,
                s.scenario_link, s.scenario_method, t.
                username')
117         cursor.execute(getAvgResult,(username,))
118         results = cursor.fetchone()
119         check_res = cursor.rowcount
120         if check > 0:
121           scen_id = results[0]
122           link = results[1]
123           method = results[2]
124           scen_worker = results[3]
125           username = results[4]
126           error = results[5]
127           response = results[6]
```

```
128         dom = results[7]
129         load = results[8]
130         css = results[9]
131         fmfp = results[10]
132 
133         insertAvgResult = ('INSERT INTO
                summary_results (scenario_id, link,
                method, worker, username, error,
                response_end, dom_content_load,
                load_event_end, css_trace_end,
                first_meaningful) \
134         values (%s, %s, %s, %s, %s, %s, %s, %s, %
                s, %s, %s)')
135         cursor.execute(insertAvgResult,(scen_id,
                link, method, scen_worker, username,
                error, response, dom, load, css, fmfp,
                ))
136         con.commit()
137 
138         updateQueueStatus = ('UPDATE queues SET
                status = 2 WHERE status = -1')
139         updateResultStatus = ('UPDATE results SET
                 status = 1 WHERE scenario_id = %s')
140         updateContainerStatus = ('UPDATE
                containers SET status = 0, username =
                0 WHERE status = 1')
141         updateScenarioStatus = ('UPDATE scenarios
                 SET scenario_status = 2 WHERE
                scenario_id = %s')
142         cursor.execute(updateQueueStatus)
143         cursor.execute(updateResultStatus,(
                scena_id,))
144         cursor.execute(updateContainerStatus)
```

```
145         cursor.execute(updateScenarioStatus,(
                scena_id,))
146         con.commit()
147 
148         # get info swarm untuk sshpass
149         getConnection = ('SELECT swarm_ip,
                swarm_username, swarm_password FROM
                swarms WHERE is_used = 1')
150         cursor.execute(getConnection)
151         swarm_conn = cursor.fetchall()
152         # run sshpass untuk copy gambar
153         for swarm in swarm_conn:
154           ip = swarm[0]
155           user = swarm[1]
156           passw = swarm[2]
157           from_path='/home/cloudy/tugas-akhir-201
                9/implementasi/output/ss'+scena_id
                +'.png'
158           to_path='/home/cloudy/web-tugas-akhir-2
                019/public/images/'
159           command_line = 'sshpass -p %s scp %s@%s
                :%s %s' % (passw, user, ip,
                from_path, to_path)
160           os.system(command_line)
161 
162 def main():
163   check = checkDb()
164   if check > 0:
165     getQueue()
166     runScenario()
167     calculateResult()
168   else:
169     print('Masih Kosong!')
```

```
170
171 if __name__ == '__main__':
172   main()
```

Kode Sumber B.6: Isi berkas queue.py

```
1  from connection import connect
2  from sys import argv
3  import os
4
5  def main():
6    # variabel argv
7    username = argv[1]
8    ip = argv[2]
9
10   # connection
11   con = connect()
12   cursor = con.cursor()
13
14   # get id skenario yang akan diuji
15   query = "SELECT scenario_id, scenario_worker
         FROM scenarios WHERE username = %s AND
         scenario_status = 1"
16   cursor.execute(query,(username,))
17   scenarios = cursor.fetchall()
18
19   # get kontainer
20   getContainers = "SELECT node_ip, container_id
         FROM containers WHERE node_ip = %s AND
         status = 1"
21   cursor.execute(getContainers,(ip,))
22   containers = cursor.fetchall()
23
24   for scenario in scenarios:
```

```
25      scenid = scenario[0]
26      worker = scenario[1]
27      for container in containers:
28        host = container[0]
29        contid = container[1]
30        command_line = 'docker container exec %s
              node getdata/index.js %s %s %s %s' % (
              contid, scenid, contid, worker, host)
31        os.system(command_line)
32
33  if __name__ == '__main__':
34    main()
```

Kode Sumber B.7: Isi berkas run_scenario.py

Kode Script Penyimpan Data Kontainer

```
1  #!/bin/bash
2  set -e
3
4  SERVICE_NAME=$1;
5
6  TASK_ID=$(docker service ps --filter 'desired-
       state=running' $SERVICE_NAME -q)
7  NODE_ID=$(docker inspect --format '{{ .NodeID }}'
        $TASK_ID)
8  CONTAINER_ID=$(docker inspect --format '{{ .
       Status.ContainerStatus.ContainerID }}'
       $TASK_ID)
9  NODE_IP=$(docker inspect --format '{{ .Status.
       Addr }}' $NODE_ID)
10 NODE_HOST=$(docker inspect --format '{{ .
       Description.Hostname }}' $NODE_ID)
```

```
11
12 echo $TASK_ID | tr ' ' '\n' > 1.txt
13 echo $NODE_ID | tr ' ' '\n' > 2.txt
14 echo $CONTAINER_ID | tr ' ' '\n' > 3.txt
15 echo $NODE_IP | tr ' ' '\n' > 4.txt
16 echo $NODE_HOST | tr ' ' '\n' > 5.txt
17
18 sed 's/[^ ][^ ]*/"&"/g' 1.txt > taskid.txt
19 sed 's/[^ ][^ ]*/"&"/g' 2.txt > nodeid.txt
20 sed 's/[^ ][^ ]*/"&"/g' 3.txt > conid.txt
21 sed 's/[^ ][^ ]*/"&"/g' 4.txt > nodeip.txt
22 sed 's/[^ ][^ ]*/"&"/g' 5.txt > nodehost.txt
23
24 paste -d',' taskid.txt nodeid.txt conid.txt
       nodeip.txt nodehost.txt > data.txt
25
26 awk '{new="INSERT INTO containers (task_id,
       node_id, container_id, node_ip, node_host)
       VALUES ("$1");"; print new}' data.txt > data.
       sql
27
28 mysql -ucloudy -psembarang12 -h178.128.123.143
       tugas_akhir_2019 -e "truncate table
       tugas_akhir_2019.containers"
29 mysql -ucloudy -psembarang12 -h178.128.123.143
       tugas_akhir_2019 < data.sql
30
31 rm *.txt
32 rm data.sql
```

Kode Sumber B.8: Isi berkas save_containers.sh

BIODATA PENULIS

Cahya Putra Hikmawan, biasa disapa Awan dan lahir pada tanggal 09 September 1996 di Krembung, Kabupaten Sidoarjo. Penulis telah menempuh pendidikan formal di SD Al-Ishlah, SMP Negeri 1 Krembung, MBI Amanatul Ummah Pacet dan melanjutkan studi program sarjana di Departemen Informatika, Institut Teknologi Sepuluh Nopember. Penulis memiliki hobi antara lain membaca novel, menonton drama dan memancing. Selama menempuh pendidikan di Departemen Informatika ITS, penulis aktif dalam kegiatan akademik dan non akademik. Di bidang akademik penulis pernah menjadi asisten dosen dan asisten praktikum untuk mata kuliah Sistem Operasi (2017 dan 2019), Jaringan Komputer (2017-2018), serta menjadi Koor Asisten Praktikum untuk mata kuliah Jaringan Komputer (2017). Selain itu penulis menjadi Admin Laboratorium Arsitektur dan Jaringan Komputer (AJK) dan mengikuti kegiatan pelatihan linux di tulungagung sebagai asisten dan panitia. Sedangkan untuk kegiatan non akademik penulis aktif menjadi Staf dan Badan Pengurus Harian II NLC (National Logic Competition) Schematics 2016 dan Schematics 2017, Staf dan Staf Ahli Departemen Pengembangan Profesi Himpunan Mahasiswa Teknik Computer-Informatika (HMTC) ITS. Penulis dapat dihubungi melalui surat elektronik di `cp.hikmawan@gmail.com`.