

TUGAS AKHIR - KI141502

**RANCANG BANGUN PERANGKAT LUNAK *INTERNET*  
ACCESS MANAGEMENT BERBASIS KONTAINER**

FOURIR AKBAR  
NRP 05111440000115

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, 2018

*(Halaman ini sengaja dikosongkan)*



TUGAS AKHIR - KI141502

**RANCANG BANGUN PERANGKAT LUNAK *INTERNET*  
ACCESS MANAGEMENT BERBASIS KONTAINER**

FOURIR AKBAR  
NRP 05111440000115

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, 2018

*(Halaman ini sengaja dikosongkan)*

UNDERGRADUATE THESIS - KI141502

**DESIGN AND IMPLEMENTATION OF INTERNET ACCESS  
MANAGEMENT SOFTWARE USING CONTAINER**

FOURIR AKBAR  
NRP 05111440000115

Supervisor I  
Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD

Supervisor II  
Bagus Jati Santoso, S.Kom., Ph.D

Department of INFORMATICS  
Faculty of Information Technology and Communication  
Institut Teknologi Sepuluh Nopember  
Surabaya, 2018

*(Halaman ini sengaja dikosongkan)*

## **LEMBAR PENGESAHAN**

### **RANCANG BANGUN PERANGKAT LUNAK *INTERNET ACCESS MANAGEMENT* BERBASIS KONTAINER**

#### **TUGAS AKHIR**

Diajukan Guna 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 :

**FOURIR AKBAR**  
**NRP: 05111440000115**

Disetujui oleh Dosen Pembimbing Tugas Akhir :

Royyana Muslim Ijtihadie, S.Kom, M.Kom, PhD .....  
NIP: 197708242006041001 (Pembimbing 1)

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

**SURABAYA**  
**Juni 2018**

*(Halaman ini sengaja dikosongkan)*



## **RANCANG BANGUN PERANGKAT LUNAK *INTERNET* ACCESS MANAGEMENT BERBASIS KONTAINER**

**Nama** : **FOURIR AKBAR**  
**NRP** : **05111440000115**  
**Jurusan** : **Informatika FTIK**  
**Pembimbing I** : **Royyana Muslim Ijtihadie, S.Kom,  
M.Kom, PhD**  
**Pembimbing II** : **Bagus Jati Santoso, S.Kom., Ph.D**

### **Abstrak**

*Internet turut memiliki peranan penting dalam dunia pendidikan, karena dengan adanya internet dapat menambah ilmu pengetahuan dan menambah motivasi belajar siswa ataupun mahasiswa. Dalam dunia pendidikan, internet telah menjadi platform penting, misalnya adalah proses belajar mengajar yang dilakukan secara online dengan e-learning, ataupun ketika pengajar memberikan nilai kepada siswanya dilakukan secara online, dan lain sebagainya. Keamanan menjaga data-data dalam dunia pendidikan, melindungi terhadap penggunaan malware, menerapkan kepatuhan akses internet, menyederhanakan manajemen jaringan menjadi tantangan utama untuk manajemen TI. Maka dari itu dibutuhkan sebuah online yang digunakan sebagai internet access management atau untuk melakukan manajemen akses terhadap user yang menggunakan jaringannya. Namun akan terjadi permasalahan ketika banyak user yang mengakses internet dengan menggunakan server yang digunakan sebagai internet access management. Dalam pembacaan log history dari setiap user akan tercampur karena hanya melewati satu server saja.*

*Internet Access Management Berbasis Kontainer memungkinkan untuk mencatat setiap log history dari setiap user*

*yang mengakses internet secara detail. Rancangan sistem pada server akan menggunakan kontainer docker. Kontainer docker merupakan operating-system-level virtualization untuk menjalankan beberapa sistem linux yang terisolasi (kontainer) pada sebuah host. Kontainer berfungsi untuk mengisolasi aplikasi atau servis dan dependensinya. Untuk setiap servis atau aplikasi yang terisolasi dibutuhkan satu kontainer pada server host yang ada dan setiap kontainer akan menggunakan sumber daya yang ada pada server host selama kontainer tersebut menyala. Pembuatan Internet Access Management Berbasis Kontainer dapat mengetahui cara bagaimana client dapat melakukan autentifikasi, untuk mengarahkan traffic dari client ke kontainer docker yang sesuai. Dapat juga digunakan untuk mempermudah pencatatan aktivitas dari masing-masing client yang mengakses internet dan juga dapat meringankan beban dari penggunaan server.*

***Kata-Kunci:*** Docker, Internet Access Management, Kontainer

# **DESIGN AND IMPLEMENTATION OF INTERNET ACCESS MANAGEMENT SOFTWARE USING CONTAINER**

**Name : FOURIR AKBAR**  
**NRP : 05111440000115**  
**Major : Informatics FTIK**  
**Supervisor I : Royyana Muslim Ijtihadie, S.Kom,  
M.Kom, PhD**  
**Supervisor II : Bagus Jati Santoso, S.Kom., Ph.D**

## **Abstract**

*Internet has an important role in the education, because we can explore knowledge and increase student motivation or student learning. In education, the Internet has become an important platform, for example is the process of teaching and learning that is done online with e-learning, or when teachers give score to their students are done online, and so on. The security of keeping the data in education, protecting against malware, using internet access compliance, simplifying network management becomes a major challenge for IT management. Therefore we need an online system that is used as internet access management or to perform management access to users who use the network. But there will be problems when many users access the internet with a server that is used as internet access management. The process of reading log history of each user will be mixed because it only passes one server only.*

*Container Based Internet Access Management makes it possible to log every log history of every user accessing the Internet in detail. The system design on the server will use docker containers. Docker containers are operating-system-level virtualization to run some isolated linux systems (containers) on*

*a host. Containers serve to isolate applications or services and their dependencies. For each isolated service or application it takes one container on an existing host server and each container will use the resources on the host server as long as the container is on. Creation of Container Based Internet Access Management can find out how the client can authenticate, to redirect traffic from the client to the appropriate docker container. It can also be used to facilitate the recording of activity from each client accessing the internet and can also lighten the the server load usage.*

**Keywords:** *Container, Docker, Internet Access Management*

## KATA PENGANTAR

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

Alhamdulillahirabbil'alam, segala puji bagi Allah SWT, yang telah melimpahkan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan Tugas Akhir yang berjudul **Rancang Bangun Perangkat Lunak *Internet Acces Management* Berbasis Kontainer**. Pengerjaan Tugas Akhir ini merupakan suatu kesempatan yang sangat baik bagi penulis. Dengan pengerjaan Tugas Akhir ini, penulis bisa belajar lebih banyak untuk memperdalam dan meningkatkan apa yang telah didapatkan penulis selama menempuh perkuliahan di Teknik Informatika ITS. Dengan Tugas Akhir ini penulis juga dapat menghasilkan suatu implementasi dari apa yang telah penulis pelajari. Selesaiannya Tugas Akhir ini tidak lepas dari bantuan dan dukungan beberapa pihak. Sehingga pada kesempatan ini penulis mengucapkan syukur dan terima kasih kepada:

1. Allah SWT atas anugerahnya yang tidak terkira kepada penulis dan Nabi Muhammad SAW.
2. Bapak, Mama, dan keluarga Penulis yang selalu memberikan perhatian, dorongan dan kasih sayang yang menjadi semangat utama bagi diri Penulis sendiri baik selama penulis menempuh masa perkuliahan maupun pengerjaan Tugas Akhir ini.
3. Bapak Royyana Muslim Ijtihadie, S.Kom., M.Kom., PhD. selaku Dosen Pembimbing yang telah banyak meluangkan waktu untuk memberikan ilmu, nasihat, motivasi, pandangan dan bimbingan kepada Penulis baik selama Penulis menempuh masa kuliah maupun selama pengerjaan Tugas Akhir ini.
4. Bagus Jati Santoso, S.Kom., PhD. selaku dosen pembimbing yang telah memberikan ilmu, dan masukan kepada Penulis.

5. Seluruh tenaga pengajar dan karyawan Jurusan Teknik Informatika ITS yang telah memberikan ilmu dan waktunya demi berlangsungnya kegiatan belajar mengajar di Jurusan Teknik Informatika ITS.
6. Seluruh teman Penulis di Jurusan Teknik Informatika ITS yang telah memberikan dukungan dan semangat kepada Penulis selama Penulis menyelesaikan Tugas Akhir ini.
7. Teman-teman, Kakak-kakak dan Adik-adik *administrator* Laboratorium Arsitektur dan Jaringan Komputer yang selalu menjadi teman untuk berbagi ilmu.

Penulis menyadari bahwa Tugas Akhir ini masih memiliki banyak kekurangan. Sehingga dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depannya.

Surabaya, Juni 2018

Fourir Akbar

# DAFTAR ISI

<b>ABSTRAK</b>	<b>vii</b>
<b>ABSTRACT</b>	<b>ix</b>
<b>Kata Pengantar</b>	<b>xi</b>
<b>DAFTAR ISI</b>	<b>xiii</b>
<b>DAFTAR TABEL</b>	<b>xix</b>
<b>DAFTAR GAMBAR</b>	<b>xxi</b>
<b>DAFTAR KODE SUMBER</b>	<b>xxiii</b>
<b>BAB I PENDAHULUAN</b>	<b>1</b>
1.1 Latar Belakang . . . . .	1
1.2 Rumusan Masalah . . . . .	2
1.3 Batasan Masalah . . . . .	3
1.4 Tujuan . . . . .	3
1.5 Manfaat . . . . .	4
1.6 Metodologi . . . . .	4
1.6.1 Studi literatur . . . . .	4
1.6.2 Desain dan Perancangan Sistem . . . . .	5
1.6.3 Implementasi Sistem . . . . .	5
1.6.4 Uji Coba dan Evaluasi . . . . .	5
1.7 Sistematika Laporan . . . . .	5
<b>BAB II TINJAUAN PUSTAKA</b>	<b>9</b>
2.1 Python . . . . .	9
2.2 Flask . . . . .	9
2.3 Nginx . . . . .	10
2.4 Iptables . . . . .	10
2.5 MySQL . . . . .	11
2.6 Mitmproxy . . . . .	13

2.7	VirtualBox . . . . .	13
2.8	Docker . . . . .	14
2.8.1	<i>Docker Container</i> . . . . .	14
2.8.2	<i>Docker Images</i> . . . . .	15
2.8.3	<i>Docker Registry</i> . . . . .	15
<b>BAB III DESAIN DAN PERANCANGAN</b>		<b>17</b>
3.1	Deskripsi Umum Sistem . . . . .	17
3.2	Kasus Penggunaan . . . . .	17
3.3	Arsitektur Sistem . . . . .	20
3.3.1	Desain Umum Sistem . . . . .	21
3.3.2	Pembuatan Halaman <i>Login</i> dari Sebuah Sistem. . . . .	23
3.3.2.1	Desain Basis Data . . . . .	24
3.3.2.2	Desain <i>Web Service</i> . . . . .	25
3.3.3	Perancangan Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> dari Sistem . . . . .	26
3.3.4	Pembuatan <i>Middleware</i> untuk Menerima Permintaan dari <i>Client</i> . . . . .	27
3.3.4.1	Desain Basis Data . . . . .	28
3.3.4.2	Desain <i>Web Service</i> . . . . .	29
3.3.5	Perancangan Pemasangan Kontainer pada <i>Docker Host</i> . . . . .	29
3.3.6	Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Kontainer <i>Docker</i> dari Tiap-Tiap <i>Client</i> . . . . .	31
3.3.7	Pembuatan Halaman <i>Administrator</i> untuk Membaca <i>Log File</i> dari <i>Client</i> . . . . .	32
3.3.8	Pembuatan <i>Schedule</i> pada <i>Docker Host</i> . . . . .	34
<b>BAB IV IMPLEMENTASI</b>		<b>35</b>
4.1	Lingkungan Implementasi . . . . .	35
4.1.1	Perangkat Keras . . . . .	35



4.1.2	Perangkat Lunak . . . . .	35
4.2	Implementasi Pembuatan Halaman <i>Login</i> dari Sebuah Sistem . . . . .	36
4.2.1	Implementasi <i>Web Service</i> pada Halaman <i>Login</i> . . . . .	36
4.2.1.1	Implementasi Tampilan Antarmuka Halaman <i>Login</i> . . .	37
4.2.1.2	Rute <i>Web Service</i> pada Halaman <i>Login</i> . . . . .	37
4.2.1.3	<i>Pseudocode Web Service</i> pada Halaman <i>Login</i> . . . . .	39
4.2.2	Implementasi Basis Data pada Halaman <i>Login</i> . . . . .	40
4.3	Implementasi Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> dari Sistem . . . . .	41
4.4	Implementasi Pembuatan <i>Middleware</i> . . . . .	41
4.4.1	Implementasi <i>Web Service</i> pada <i>Middleware</i> . . . . .	42
4.4.1.1	Rute <i>Web Service</i> pada <i>Middleware</i> . . . . .	42
4.4.1.2	<i>Pseudocode Web Service</i> pada <i>Middleware</i> . . . . .	43
4.4.2	Implementasi Basis Data pada <i>Middleware</i> . . . . .	45
4.5	Implementasi Pemasangan Kontainer Docker pada <i>Docker Host</i> . . . . .	45
4.5.1	Menambahkan dan Memperbarui Kontainer Docker yang Berisikan Mitmproxy . . . . .	45
4.5.2	Menggunakan <i>Image</i> Kontainer Docker yang Sudah Dibuat . . . . .	47
4.6	Implementasi Pembuatan Aturan untuk Mengarahkan <i>Traffic Client</i> ke Kontainer Docker dari Tiap-Tiap <i>Client</i> . . . . .	48

4.7	Implementasi Pembuatan Halaman <i>Administrator</i>	48
4.7.1	Rute <i>Web Service</i> pada Halaman <i>Administrator</i> . . . . .	49
4.7.2	Implementasi Pembacaan <i>Log File</i> dari <i>Client</i> . . . . .	51
4.7.3	Implementasi Antarmuka Halaman <i>Administrator</i> . . . . .	52
<b>BAB V</b>	<b>PENGUJIAN DAN EVALUASI</b>	<b>55</b>
5.1	Lingkungan Uji Coba . . . . .	55
5.2	Skenario Uji Coba . . . . .	60
5.2.1	Skenario Uji Coba Fungsionalitas . . . . .	61
5.2.1.1	Uji <i>client</i> dapat Mengakses Internet . . . . .	61
5.2.1.1.1	Uji <i>Client</i> dapat <i>Login</i> ke Dalam Sistem . . . . .	62
5.2.1.1.2	Uji <i>Client</i> dapat Mengirimkan Permintaan Penyediaan Kontainer <i>Docker</i> ke <i>Docker Host</i> . . . . .	63
5.2.1.1.3	Uji <i>Docker Host</i> dapat Menerima Permintaan Penyediaan Kontainer <i>Docker</i> . . . . .	64
5.2.1.2	Uji Fungsionalitas Menu Aplikasi Halaman <i>Administrator</i> . . . . .	65
5.2.1.3	Uji Fungsionalitas <i>Schedule</i> pada <i>Docker Host</i> . . . . .	67
5.2.2	Skenario Uji Coba Performa . . . . .	68

5.2.2.1	Uji Performa Penggunaan <i>Memory</i> . . . . .	68
5.2.2.2	Uji Performa Penggunaan <i>CPU</i> . . . . .	69
5.2.2.3	Uji Performa Kecepatan Menangani <i>Request</i> . . . . .	69
5.2.2.4	Uji Performa Keberhasilan <i>Request</i> . . . . .	69
5.3	Hasil Uji Coba dan Evaluasi . . . . .	69
5.3.1	Uji Fungsionalitas . . . . .	69
5.3.1.1	Uji <i>Client</i> dapat Mengakses Internet . . . . .	70
5.3.1.1.1	Uji <i>Client</i> dapat <i>Login</i> ke Dalam Sistem . . . . .	70
5.3.1.1.2	Uji <i>Client</i> dapat Mengirimkan Permintaan Penyediaan Kontainer <i>Docker</i> ke <i>Docker Host</i> . . . . .	71
5.3.1.1.3	Uji <i>Docker Host</i> dapat Menerima Permintaan Penyediaan Kontainer <i>Docker</i> . . . . .	72
5.3.1.2	Uji Fungsionalitas Menu Aplikasi Halaman <i>Administrator</i> . . . . .	72
5.3.1.3	Uji Fungsionalitas <i>Schedule</i> pada <i>Docker Host</i> . . . . .	73
5.3.2	Hasil Uji Performa . . . . .	74
5.3.2.1	Penggunaan <i>Memory</i> . . . . .	74
5.3.2.2	Penggunaan <i>CPU</i> . . . . .	74
5.3.2.3	Kecepatan Menangani <i>Request</i> . . . . .	75
5.3.2.4	Keberhasilan <i>Request</i> . . . . .	75

5.4	Hasil Uji Coba dan Evaluasi . . . . .	75
5.4.1	Hasil Uji Performa . . . . .	75
5.4.1.1	Kecepatan Menangani <i>Request</i> .	75
5.4.1.2	Penggunaan CPU . . . . .	78
5.4.1.3	Penggunaan <i>Memory</i> . . . . .	80
5.4.1.4	Keberhasilan <i>Request</i> . . . . .	81
<b>BAB VI</b>	<b>PENUTUP</b>	<b>83</b>
6.1	Kesimpulan . . . . .	83
6.2	Saran . . . . .	83
<b>DAFTAR</b>	<b>PUSTAKA</b>	<b>85</b>
<b>BAB A</b>	<b>INSTALASI PERANGKAT LUNAK</b>	<b>87</b>
<b>BAB B</b>	<b>Konfigurasi</b>	<b>91</b>
<b>BIODATA</b>	<b>PENULIS</b>	<b>95</b>

## DAFTAR TABEL

3.1	Daftar Kode Kasus Penggunaan . . . . .	19
3.1	Daftar Kode Kasus Penggunaan . . . . .	20
3.2	Atribut basis data nrp-mahasiswa . . . . .	25
3.3	Atribut basis data kontainer . . . . .	28
3.3	Atribut basis data kontainer . . . . .	29
4.1	Daftar Rute <i>Web Service</i> . . . . .	38
4.2	Daftar Rute <i>Web Service</i> . . . . .	43
4.3	Daftar Rute <i>Web Service</i> pada Halaman <i>Administrator</i> . . . . .	49
5.1	<i>Server</i> Untuk <i>Docker Host</i> . . . . .	56
5.2	<i>Server</i> Untuk Halaman Login . . . . .	56
5.3	Komputer Penguji 1 . . . . .	57
5.4	Komputer Penguji 2 . . . . .	57
5.5	Komputer Penguji 3 . . . . .	58
5.6	Komputer Penguji 4 . . . . .	58
5.7	Komputer Penguji 5 . . . . .	59
5.8	Komputer Penguji 6 . . . . .	59
5.9	Skenario Uji <i>Client</i> dapat Mengakses Internet . . .	62
5.10	Skenario Uji <i>Client</i> dapat <i>Login</i> ke Dalam Sistem	62
5.11	Skenario Uji <i>Client</i> dapat <i>Login</i> Mengirimkan Permintaan Penyediaan Kontainer <i>Docker</i> . . . .	63
5.12	Skenario Uji <i>Docker Host</i> dapat Menerima Permintaan Penyediaan Kontainer <i>Docker</i> . . . .	65
5.13	Skenario Uji Fungsionalitas Aplikasi Halaman <i>Administrator</i> . . . . .	66
5.14	Skenario Uji <i>Schedule</i> pada <i>Docker Host</i> . . . . .	68
5.15	Hasil Uji Coba <i>Client</i> dapat Mengakses Internet .	70
5.16	Hasil Uji Coba <i>Client</i> dapat <i>Login</i> ke Dalam Sistem	71
5.17	Hasil Uji Coba <i>Client</i> dapat Mengirimkan Permintaan Penyediaan Kontainer <i>Docker</i> ke <i>Docker Host</i> . . . . .	71

5.18 Hasil Uji Coba <i>Docker Host</i> dapat Menerima Permintaan Penyediaan Kontainer <i>Docker</i> . . . . .	72
5.19 Hasil Uji Fungsionalitas Aplikasi Halaman <i>Administrator</i> . . . . .	73
5.20 Skenario Uji <i>Schedule</i> pada <i>Docker Host</i> . . . . .	74
5.21 Kecepatan Menangani <i>Request</i> Unduh dan <i>Upload</i> Menggunakan <i>Internet Access Management</i> Berbasis Kontainer . . . . .	76
5.22 Kecepatan Menangani <i>Request</i> Unduh dan <i>Upload</i> Menggunakan <i>Internet Access Management</i> Konvensional . . . . .	77
5.23 <i>Penggunaan CPU</i> . . . . .	79
5.24 <i>Penggunaan Memory</i> . . . . .	80
5.25 <i>Success Ratio Request</i> . . . . .	81

## DAFTAR GAMBAR

3.1	Digram Kasus Penggunaan . . . . .	18
3.2	Arsitektur Komponen Sistem . . . . .	23
3.3	Desain Halaman <i>Login</i> . . . . .	26
3.4	Desain Mengarahkan <i>Traffic Client</i> ke Halaman <i>Login</i> . . . . .	27
3.5	Alur kerja dari <i>mitmproxy transparent</i> HTTP . . .	30
3.6	Alur kerja dari <i>mitmproxy transparent</i> HTTPS . .	31
3.7	Desain pembuatan aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i> . . . . .	32
3.8	Desain halaman dashboard <i>administrator traffic</i> <i>client</i> ke kontainer <i>docker</i> . . . . .	33
3.9	Desain pembuatan aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i> . . . . .	33
4.1	Halaman <i>Login</i> . . . . .	37
4.2	<i>Query</i> untuk membuat tabel kontainer . . . . .	40
4.3	Command untuk mengarahkan <i>client</i> ke halaman <i>login</i> . . . . .	41
4.4	Perintah memasukkan <i>user</i> ke grup <i>docker</i> . . . .	45
4.5	Perintah untuk Pemasangan Mitmrproxy . . . . .	46
4.6	Command untuk mengarahkan <i>client</i> ke halaman <i>login</i> . . . . .	48
4.7	Halaman <i>Administrator Menu User List</i> . . . . .	52
4.8	Halaman <i>Administrator Menu History</i> . . . . .	53
5.1	Arsitektur dari Setiap Komponen Uji Coba . . . . .	60
5.2	Grafik Kecepatan Menangani <i>Request</i> . . . . .	76
5.3	Grafik Kecepatan Menangani <i>Request</i> . . . . .	77
5.4	Grafik Penggunaan <i>Memory</i> . . . . .	79
5.5	Grafik Penggunaan <i>Memory</i> . . . . .	81

*(Halaman ini sengaja dikosongkan)*



## DAFTAR KODE SUMBER

4.1	Pseudocode Web Service . . . . .	40
4.2	Pseudocode Web Service . . . . .	44
2.1	Isi Berkas app.conf . . . . .	91
2.2	Command untuk Reload Supervisor . . . . .	92
2.3	Command untuk mengaktifkan konfigurasi Nginx . . . . .	92
2.4	Command untuk merestart Nginx . . . . .	92
2.5	Isi Berkas app . . . . .	92

*(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

Seiring dengan perkembangan zaman yang sangat pesat, negara-negara sudah mempunyai teknologi yang sangat maju. Teknologi mempunyai peranan yang sangat penting dalam kehidupan manusia, karena dengan adanya teknologi, manusia bisa saling berhubungan dengan mudah. Sekarang teknologi sudah semakin canggih. Teknologi yang paling populer sekarang ini adalah internet karena dengan adanya internet, banyak informasi-informasi yang dapat kita ambil dengan mudah. Internet merupakan suatu perpustakaan besar yang di dalamnya terdapat sangat banyak informasi yang berupa teks dalam bentuk media elektronik. Selain itu internet dikenal sebagai dunia maya, karena hampir seluruh aspek kehidupan di dunia nyata ada di internet, seperti olah raga, politik, hiburan, akademik, bisnis, dan lain sebagainya. Internet juga mempunyai peranan yang sangat penting dalam dunia pendidikan, karena dengan adanya internet bisa menambah ilmu pengetahuan kita dan dapat menambah motivasi belajar siswa ataupun mahasiswa. Dengan dimanfaatkan internet dalam dunia pendidikan agar siswa atau mahasiswa dapat memiliki komitmen untuk belajar secara aktif dan memiliki teknis kemampuan khususnya di bidang pendidikan. Oleh karena itu, internet dapat mempermudah proses belajar mengajar dengan baik.

Dalam dunia pendidikan, internet telah menjadi *platform* penting, misalnya adalah proses belajar mengajar yang dilakukan secara *online* dengan e-learning, ataupun ketika pengajar

memberikan nilai kepada siswanya dilakukan secara *online*, dan lain sebagainya. Keamanan menjaga data-data dalam dunia pendidikan, melindungi terhadap penggunaan malware, menerapkan kepatuhan akses internet, menyederhanakan manajemen jaringan menjadi tantangan utama untuk manajemen TI. Maka dari itu dibutuhkan sebuah *online* yang digunakan sebagai *internet access management* atau untuk melakukan manajemen akses terhadap *user* yang menggunakan jaringannya.

Namun akan terjadi permasalahan ketika banyak *user* yang mengakses internet dengan menggunakan *server* yang digunakan sebagai *internet access management*. Dalam pembacaan *log history* dari setiap *user* akan tercampur karena hanya melewati satu *server* saja.

Dalam tugas akhir ini akan dibuat sebuah rancangan sistem pada *server* yang akan dijadikan sebagai *internet access management*, yang memungkinkan untuk mencatat setiap *log history* dari setiap *user* yang mengakses internet secara detail. Rancangan sistem pada *server* akan menggunakan kontainer *docker*. Kontainer *docker* merupakan *operating-system-level virtualization* untuk menjalankan beberapa sistem linux yang terisolasi (kontainer) pada sebuah host. Kontainer berfungsi untuk mengisolasi aplikasi atau servis dan dependensinya. Untuk setiap servis atau aplikasi yang terisolasi dibutuhkan satu kontainer pada *server host* yang ada dan setiap kontainer akan menggunakan sumber daya yang ada pada *server host* selama kontainer tersebut menyala.

## 1.2 Rumusan Masalah

Berikut beberapa hal yang menjadi rumusan masalah dalam tugas akhir ini:

1. Bagaimana cara *client* untuk melakukan *autentifikasi*?
2. Bagaimana cara membuat sebuah kontainer *docker* secara

otomatis ketika terdapat *client* yang akan mengakses internet?

3. Bagaimana cara mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai?
4. Bagaimana cara mencatat aktivitas dari *client*?
5. Bagaimana perbandingan performa antara IAM konvensional dengan IAM berbasis kontainer?
6. Bagaimana mengevaluasi penggunaan sumber daya dan skalabilitas pada *docker host*?

### 1.3 Batasan Masalah

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

1. Satu *client* yang berhasil *login* akan disediakan satu kontainer.
2. Kontainer yang digunakan adalah *docker*.
3. Parameter untuk mengetahui apa saja yang diakses oleh *client* adalah *access log* dari *client* tersebut.
4. Setiap *client* mendapatkan IP *private*.
5. Performa yang diukur adalah *response time*.
6. Bahasa pemrograman yang digunakan adalah *Python*.

### 1.4 Tujuan

Tugas akhir dibuat dengan beberapa tujuan. Berikut beberapa tujuan dari pembuatan tugas akhir:

1. Mengetahui cara bagaimana *client* dapat melakukan *autentifikasi*.
2. Mengimplementasikan metode untuk membuat sebuah kontainer terhadap *client* yang telah berhasil *login* ke jaringan ITS.

3. Mengetahui cara untuk mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai.
4. Mengetahui bagaimana cara mencatat aktivitas *client*.
5. Mengetahui bagaimana perbandingan performa antara IAM konvensional dengan IAM berbasis kontainer.
6. Mengetahui penggunaan sumber daya dan skalabilitas pada *docker host*.

## 1.5 Manfaat

Tugas akhir dibuat dengan beberapa manfaat. Berikut beberapa manfaat dari pembuatan tugas akhir:

1. Mengetahui cara bagaimana *client* dapat melakukan *autentifikasi*.
2. Mengetahui cara untuk mengarahkan *traffic* dari *client* ke kontainer *docker* yang sesuai.
3. Mempermudah pencatatan aktivitas dari masing-masing *client* yang mengakses internet.
4. Meringankan beban dari penggunaan *server* di ITS karena penggunaan kontainer *docker* lebih ringan.

## 1.6 Metodologi

Metodologi yang digunakan pada pengerjaan Tugas Akhir ini adalah sebagai berikut:

### 1.6.1 Studi literatur

Studi literatur merupakan langkah yang dilakukan untuk mendukung dan memastikan setiap tahap pengerjaan tugas akhir sesuai dengan standar dan konsep yang berlaku. Pada tahap studi literatur ini, akan dilakukan studi mendalam mengenai kontainer *docker*, *flask*, *mitmproxy*, dan pembuatan aturan dengan menggunakan *iptables*. Adapun literatur yang dijadikan sumber

berasal dari paper, buku, materi perkuliahan, forum serta artikel dari internet.

### **1.6.2 Desain dan Perancangan Sistem**

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep. Tahap ini merupakan tahap yang paling penting dimana bentuk awal aplikasi yang akan diimplementasikan didefinisikan. Pada tahapan ini dibuat kasus penggunaan yang ada pada sistem, arsitektur sistem, serta perencanaan implementasi pada sistem.

### **1.6.3 Implementasi Sistem**

Implementasi merupakan tahap membangun implementasi rancangan sistem yang telah dibuat. Pada tahapan ini merealisasikan apa yang telah didesain dan dirancang pada tahapan sebelumnya, sehingga menjadi sebuah sistem yang sesuai dengan apa yang telah direncanakan.

### **1.6.4 Uji Coba dan Evaluasi**

Pada tahapan ini dilakukan uji coba terhadap sistem yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat kesesuaian dengan perencanaan. Selain itu, tahap ini juga akan melakukan uji performa sistem dan melakukan perbandingan dengan metode lain untuk mengetahui efisiensi penggunaan sumber daya serta evaluasi berdasarkan hasil uji performa tersebut.

## **1.7 Sistematika Laporan**

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna bagi pembaca yang berminat melakukan pengembangan

lebih lanjut. Secara garis besar, buku tugas akhir ini terdiri atas beberapa bagian seperti berikut:

1. **Bab I Pendahuluan**

Bab yang berisi latar belakang, tujuan, manfaat, permasalahan, batasan masalah, metodologi yang digunakan dan sistematika laporan.

2. **Bab II Dasar Teori**

Bab ini berisi penjelasan secara detail mengenai dasar-dasar penunjang dan teori-teori yang digunakan dalam pembuatan tugas akhir ini.

3. **Bab III Desain dan Perancangan**

Bab ini berisi tentang analisis dan perancangan sistem yang dibuat, termasuk di dalamnya mengenai analisis kasus penggunaan, desain arsitektur sistem, dan perancangan implementasi sistem.

4. **Bab IV Implementasi**

Bab ini membahas implementasi dari desain yang telah dibuat pada bab sebelumnya. Penjelasan berupa pemasangan alat dan kode program yang digunakan untuk mengimplementasikan sistem.

5. **Bab V Uji Coba dan Evaluasi**

Bab ini membahas tahap-tahap uji coba serta melakukan evaluasi terhadap sistem yang dibuat.

6. **Bab VI Kesimpulan dan Saran**

Bab ini merupakan bab terakhir yang memberikan kesimpulan dari hasil percobaan dan evaluasi yang telah dilakukan. Pada bab ini juga terdapat saran bagi pembaca yang berminat untuk melakukan pengembangan lebih



lanjut.

*(Halaman ini sengaja dikosongkan)*

## **BAB II**

### **TINJAUAN PUSTAKA**

#### **2.1 Python**

Python adalah bahasa pemrograman interpretatif multiguna dengan prinsip agar sumber kode yang dihasilkan memiliki tingkat keterbacaan yang baik. Python diklaim sebagai bahasa yang menggabungkan kapabilitas, kemampuan, dengan sintaksis kode yang sangat jelas, dan dilengkapi dengan fungsionalitas pustaka standar yang besar serta komprehensif. Python mendukung beragam paradigma pemrograman, seperti pemrograman berorientasi objek, pemrograman imperatif, dan pemrograman fungsional. Python dapat digunakan untuk berbagai keperluan pengembangan perangkat lunak dan dapat berjalan di berbagai platform sistem operasi. [1]

#### **2.2 Flask**

Flask adalah sebuah kerangka kerja web. Artinya, Flask menyediakan perangkat, pustaka, dan teknologi yang memungkinkan seorang pengembang untuk membangun aplikasi berbasis web. Aplikasi web yang bisa dibangun bisa berupa sebuah halaman web, blog, wiki, bahkan untuk web komersial. Flask dibangun berbasiskan pada Werkzeug, Jinja 2, dan MarkupSafe yang mana menggunakan bahasa pemrograman Python sebagai basisnya. Flask sendiri pertama kali dikembangkan pada tahun 2010 dan didistribusikan dengan lisensi BSD. [2]

Flask termasuk sebagai perangkat kerja mikro karena tidak membutuhkan banyak perangkat atau pustaka tertentu agar bisa bekerja. Flask tidak menyediakan fungsi untuk melakukan interaksi dengan basis data, tidak mempunyai validasi *form* atau fungsi lain yang umumnya bisa digunakan dan disediakan pada sebuah kerangka kerja web. Meskipun memiliki kemampuan

yang minim, tapi Flask mendukung dan memberikan kemudahan bagi pengembang untuk menambahkan pustaka sendiri untuk mendukung aplikasinya. Berbagai pustaka seperti validasi *form*, mengunggah file, berbagai macam teknologi autentifikasi bisa digunakan dan tersedia untuk Flask. Bahkan pustaka-pustaka pendukung tersebut lebih sering diperbarui dibandingkan dengan Flasknya sendiri.

### 2.3 Nginx

Nginx adalah sebuah perangkat lunak yang bisa digunakan untuk *web server*, *load balancer*, dan *reverse proxy*. Nginx terkenal karena stabil, memiliki tingkat performa tinggi dan konsumsi sumber daya yang minim. Pada kasus saat terjadi koneksi dalam jumlah yang banyak secara bersamaan, penggunaan *memory*, CPU, dan sumber daya sistem yang lain sangat kecil dan stabil. [5]

Nginx bisa digunakan untuk menyajikan konten HTTP yang dinamis menggunakan FastCGI, SCGI untuk menangani scripts, aplikasi WSGI , dan bisa juga digunakan sebagai sebuah *load balancer*. Nginx menggunakan *asynchronous event-driven* untuk menangani permintaan. Dengan menggunakan model ini bisa, pengembang bisa melakukan prediksi kinerja Nginx saat terjadi jumlah permintaan yang banyak.

### 2.4 Iptables

*Firewall* merupakan sebuah mekanisme wajib *access* kontrol antar jaringan ataupun antar sistem. *Firewall* ini sangat penting karena bertujuan untuk memastikan keamanan dari sebuah jaringan. *Firewall* dapat menjadi *filter* yang sangat sederhana dan mudah digunakan, tetapi *firewall* juga dapat menjadi *filter* yang sangat penting bagi sebuah jalan keluar suatu jaringan.

Prinsip dari penggunaan *firewall* tetaplah sama, dimana penggunaannya untuk *monitoring* dan *filtering* semua pertukaran informasi di jaringan *internal* dan juga di jaringan *external*.

*Netfilter* / *Iptables* merupakan sebuah sistem *firewall* berbasis linux yang mempunyai fungsi yang sangat berguna. *Netfilter* / *iptables kernel* menggunakan sebuah mekanisme baru, bernama *Iptables*. *Iptables* sendiri merupakan sebuah perangkat lunak atau alat yang dapat melakukan manajemen *filter* dari sebuah paket yang ada pada suatu *kernel*. *Iptables* mempunyai *table* dan juga *chain* dari masing-masing *table*. *Table* pada *Iptables* terdiri dari tiga, atau juga bisa disebut *Iptables* memiliki tiga fungsi utama, antara lain menjadi penyaring paket, mentranslasikan suatu alamat, dan melakukan penghalusan paket seperti TTL, TOS, dan MARK. [6]

*Filter table* merupakan sebuah konfigurasi *default* dari *Iptables*, dimana pada *filter table* terdapat tiga *chain*, antara lain *chain* INPUT, FORWARD, dan OUTPUT. *NAT table* berfungsi untuk merubah tujuan dari sumber dari sebuah paket. Pada *NAT table* terdapat dua *chain*, antara lain *chain* PREROUTING dan POSTROUTING. *Mangle table* berfungsi untuk menghaluskan paket atau juga dapat mengubah isi dari sebuah data kecuali IP *address* dan *port address*. Pada *mangle table* terdapat dua *chain*, antara lain POSTROUTING dan OUTPUT.

## 2.5 MySQL

MySQL adalah sebuah perangkat lunak terbuka untuk melakukan manajemen basis data SQL atau DBMS. MySQL ditulis dalam bahasa pemrograman C dan C++. MySQL merupakan salah satu perangkat lunak terbuka yang banyak disukai oleh pengembang dan digunakan dalam banyak aplikasi web. Parser SQL yang digunakan ditulis dalam bahasa pemrograman yacc. MySQL bekerja pada banyak *platform*,

seperti FreeBSD, HP-UX, Linux, macOS, Microsoft Windows, NetBSD, OpenBSD, OpenSolaris, Oracle Solaris, dan SunOS. MySQL tersedia sebagai perangkat lunak gratis di bawah lesensi *GNU General Public License* (GPL), tetapi juga tersedia lisensi komersial untuk kasus-kasus dimana penggunaanya tidak cocok dengan penggunaan GPL. [7]

Setiap pengguna dapat secara bebas menggunakan MySQL, namun dengan batasan perangkat lunak tersebut tidak boleh dijadikan produk turunan yang bersifat komersial. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam basis data yang telah ada sebelumnya, yaitu SQL (*Structured Query Language*). SQL adalah sebuah konsep pengoperasian basis data, terutama untuk proses pemilihan atau seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah.

Kehandalan suatu sistem basis data dapat diketahui dari cara kerja pengoptimasiannya dalam melakukan proses perintah-perintah SQL yang dibuat oleh pengguna maupun program-program aplikasi yang memanfaatkannya. Sebagai *server* basis data, MySQL mendukung operasi basis data transaksional maupun operasi basis data non-transaksional. Pada modus operasi non-transaksional, MySQL dapat dikatakan handal dalam hal unjuk kerja dibandingkan *server* basis data kompetitor lainnya. Namun pada modus non-transaksional tidak ada jaminan atas reliabilitas terhadap data yang tersimpan, karenanya modus non-transaksional hanya cocok untuk jenis aplikasi yang tidak membutuhkan reliabilitas data seperti aplikasi blogging berbasis web (wordpress), CMS, dan sejenisnya. Untuk kebutuhan sistem yang ditujukan untuk bisnis sangat disarankan untuk menggunakan modus basis data transaksional, hanya saja sebagai konsekuensinya unjuk kerja MySQL pada modus transaksional tidak secepat unjuk kerja pada modus non-transaksional.

## 2.6 Mitmproxy

Mitmproxy adalah sebuah *interception proxy* untuk HTTP dengan antarmuka pengguna *console* yang ditulis dengan bahasa *Python*. Mitmproxy merupakan sebuah perangkat lunak yang interaktif dimana Mitmproxy memungkinkan dapat memotong dan memodifikasi HTTP *requests* atau *response* dengan sangat cepat.

Mitmproxy adalah sebuah *proxy* berkemampuan SSL yang berfungsi sebagai *man-in-the-middle* untuk komunikasi HTTP dan HTTPS. Untuk dapat mengetahui atau memodifikasi komunikasi HTTPS, Mitmproxy berupura-pura menjadi *server* ke *client* dan *client* ke server, sementara itu Mitmproxy diposisikan di tengah-tengah berfungsi untuk menerjemahkan lalu lintas dari keduanya. Mitmproxy menghasilkan sertifikat *on-the-fly* untuk mengetahui *client* agar percaya bahwa mereka berkomunikasi dengan *server*. [8]

Pertama kali Mitmproxy dimulai, maka akan menghasilkan sertifikat SSL yang berada pada `/.mitmproxy/cert.pem`. Sertifikat ini akan digunakan untuk *browser-side*. Karena tidak akan cocok dengan *domain* yang *client* kunjungi, dan tidak akan memverifikasi terhadap otoritas sertifikasi, *client* harus menambahkan pengecualian untuk setiap situs yang *client* kunjungi. Permintaan SSL dicegat dengan hanya mengamsumsikan bahwa semua permintaan `CONNECT` adalah HTTPS. Sambungan dari *browser* dibungkus SSL, dan kita membaca permintaan dengan berpura-pura menjadi *server* yang menghubungkan.

## 2.7 VirtualBox

*VirtualBox* merupakan salah satu produk perangkat lunak yang sekarang dikembangkan oleh Oracle. Aplikasi ini pertama

kali dikembangkan oleh perusahaan Jerman, Innotek GmbH. Februari 2008, Innotek GmbH diakuisisi oleh Sun Microsystems. Sun Microsystems kemudian juga diakuisisi oleh Oracle. VirtualBox berfungsi untuk melakukan virtualisasi sistem operasi. VirtualBox juga dapat digunakan untuk membuat virtualisasi jaringan komputer sederhana. Penggunaan VirtualBox ditargetkan untuk *server*, desktop, dan penggunaan *embedded*.

Berdasarkan jenis VMM yang ada, VirtualBox merupakan jenis *hypervisor type 2*. VirtualBox sendiri memiliki berbagai macam kegunaan, diantaranya VirtualBox dapat memainkan semua sistem operasi baik itu menggunakan windows, linux, atau turunan linux lainnya. VirtualBox juga dapat dipergunakan untuk mengujicoba OS baru. VirtualBox juga dapat digunakan sebagai media untuk membuat simulasi jaringan.

## 2.8 Docker

Docker adalah sebuah aplikasi yang bersifat *open source* yang berfungsi sebagai wadah untuk memasukkan sebuah perangkat lunak secara lengkap beserta semua hal yang dibutuhkan oleh perangkat lunak tersebut agar dapat berfungsi sebagaimana mestinya. Docker dapat dijalankan di berbagai sistem operasi, pengembang dapat dengan mudah menggunakan layanan Docker melalui <https://hub.docker.com> untuk mengunduh *images* ataupun membuat *images* yang diinginkan. [9]

### 2.8.1 Docker Container

*Docker container* atau kontainer Docker bisa dikatakan sebagai sebuah wadah atau tempat, dimana kontainer Docker ini dibuat dengan menggunakan *docker image*. [10] Saat kontainer Docker dijalankan, maka akan terbentuk sebuah *layer* di atas



*docker image*. Contohnya saat menggunakan *image* Ubuntu, kemudian membuat sebuah kontainer Docker dari *image* Ubuntu tersebut dengan nama *mitmproxy-ubuntu*. Setelah itu dilakukan pemasangan sebuah perangkat lunak, misalnya Mitmproxy, maka secara otomatis kontainer Docker *mitmproxy-ubuntu* akan berada di atas *layer image* Ubuntu, dan di atasnya lagi merupakan *layer mitmproxy* berada. *Docker Kontainer* atau Kontainer Docker ke depannya dapat digunakan untuk menghasilkan sebuah *docker images*. *Docker images* yang dihasilkan dari kontainer Docker itu sendiri nantinya dapat digunakan kembali untuk membuat kontainer Docker yang lainnya.

### 2.8.2 *Docker Images*

*Docker images* adalah sebuah *blueprint* atau rancangan dasar dari sebuah perangkat lunak berbasis Docker yang bersifat *read-only*. *Blueprint* ini sendiri merupakan sebuah sistem operasi atau sistem operasi yang telah dipasang berbagai perangkat lunak dan pustaka pendukung. *Docker iamges* berfungsi untuk membuat kontainer Docker, dimana dengan menggunakan satu *docker iamge* dapat dibuat lebih dari satu kontainer Docker. *Docker image* sendiri dapat menyelesaikan permasalahan yang dikenal dengan "*dependency hell*", dimana sulitnya untuk melengkapi dependensi sebuah perangkat lunak. Permasalahan tersebut dapat diselesaikan karena semua kebutuhan perangkat lunak sudah berada di dalamnya.

### 2.8.3 *Docker Registry*

*Docker Registry* adalah kumpulan dari berbagai macam *docker image* yang bersifat tertutup maupun terbuka yang dapat diakses di <https://hub.docker.com/> atau dapat diakses pada *server* sendiri. Dengan menggunakan *docker registry*, seseorang dapat menggunakan *docker image* yang telah dibuat oleh orang

lainnya. Hal seperti ini dapat mempermudah seseorang untuk melakukan pengembangan dan juga transfer aplikasi. [11]

## BAB III

### DESAIN DAN PERANCANGAN

Pada bab ini dibahas mengenai analisis dan perancangan dari sistem.

#### 3.1 Deskripsi Umum Sistem

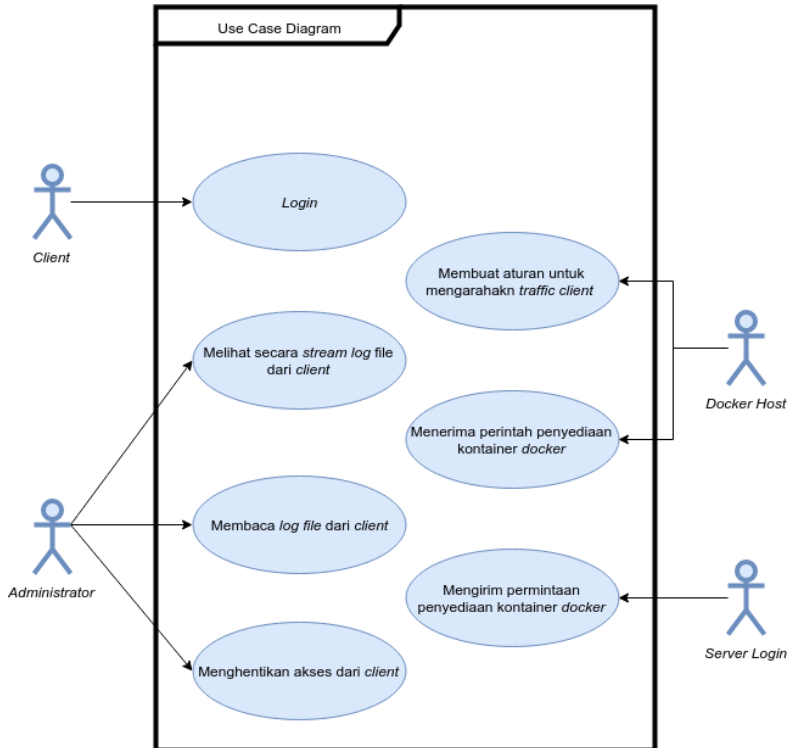
Sistem yang akan dibuat adalah sebuah sistem yang dapat membuat sebuah kontainer *docker* secara otomatis untuk setiap satu *client* yang telah *login* ke dalam sistem. Saat *client* belum *login* ke dalam sistem, maka *client* tersebut akan diarahkan ke halaman *login* dari sistem. Saat *client* mencoba untuk *login* ke dalam sistem, maka sistem akan melakukan pengecekan di dalam basis data apakah *username* dan *password* yang diinputkan sudah benar atau salah.

Setelah *client* berhasil *login* ke dalam sistem, sistem akan mengirimkan perintah untuk membuat kontainer *docker* yang berisikan Mitmproxy ke *docker host*. Setelah berhasil membuat kontainer *docker* untuk *client* tersebut, maka *traffic* internet dari *client* tersebut akan diarahkan ke kontainer *docker* berisikan Mitmproxy yang baru saja dibuat. Setelah itu *client* dapat mengakses internet.

#### 3.2 Kasus Penggunaan

Terdapat empat aktor dalam sistem yang akan dibuat yaitu *Client*, *Server Login*, *Administrator*, dan *Docker Host*. *Client* adalah aktor yang melakukan proses *login* ke dalam sistem, *server login* adalah aktor yang melakukan proses permintaan penyediaan kontainer *docker*, *administrator* adalah aktor yang melakukan monitoring kontainer *docker* yang sedang berjalan, sedangkan *docker host* adalah aktor yang akan menjadi tempat penyedia kontainer dan menerima perintah penyediaan kontainer. Diagram kasus penggunaan menggambarkan kebutuhan -

kebutuhan yang harus dipenuhi sistem. Diagram kasus penggunaan digambarkan pada Gambar 3.1.



**Gambar 3.1:** Digram Kasus Penggunaan

Digram kasus penggunaan pada Gambar 3.1 dideskripsikan masing-masing pada Tabel 3.1.

**Tabel 3.1:** Daftar Kode Kasus Penggunaan

<b>Kode Kasus Penggunaan</b>	<b>Nama Kasus Penggunaan</b>	<b>Keterangan</b>
UC-0001	<i>Login</i>	<i>Client</i> dapat <i>login</i> ke dalam sistem.
UC-0002	Mengirim Permintaan Penyediaan Kontainer <i>Docker</i>	<i>Server login</i> dapat mengirimkan permintaan penyediaan kontainer <i>docker</i> pada <i>docker host</i> .
UC-0003	Menerima Perintah Penyediaan Kontainer <i>Docker</i>	Proses dimana <i>docker host</i> akan menerima perintah dari sistem, untuk menyediakan kontainer secara otomatis.
UC-0004	Membuat Aturan untuk Mengarahkan <i>Traffic Client</i>	Proses dimana <i>docker host</i> akan membuat aturan untuk mengarahkan <i>traffic client</i> ke halaman <i>login</i> dari sistem atau untuk membuat aturan untuk mengarahkan <i>traffic client</i> ke kontainer <i>docker</i> dari tiap-tiap <i>client</i> .

**Tabel 3.1:** Daftar Kode Kasus Penggunaan

<b>Kode Kasus Penggunaan</b>	<b>Nama Kasus Penggunaan</b>	<b>Keterangan</b>
UC-0005	Membaca <i>Log File</i> dari <i>Client</i>	Proses dimana <i>administrator</i> dari sebuah jaringan dapat membaca <i>log file</i> dari <i>client</i> sampai pada <i>client</i> terakhir mengakses internet.
UC-0006	Melihat Secara <i>Stream Log File</i> dari <i>Client</i>	Proses dimana <i>administrator</i> dari sebuah jaringan dapat melihat <i>log file</i> dari <i>client</i> secara langsung atau <i>live</i> .
UC-0007	Menghentikan akses dari <i>client</i> .	Proses dimana <i>administrator</i> dari sebuah jaringan dapat menghapus akses dari <i>client</i> untuk mengakses internet, dengan cara menghapus kontainer <i>docker</i> dan <i>rules</i> sesuai dengan IP dari <i>client</i> tersebut.

### 3.3 Arsitektur Sistem

Pada sub-bab ini, dibahas mengenai tahap analisis arsitektur, analisis teknologi dan desain sistem yang akan dibangun.

### 3.3.1 Desain Umum Sistem

Berdasarkan deskripsi umum sistem yang telah ditulis diatas, dapat diperoleh kebutuhan sistem ini, diantaranya :

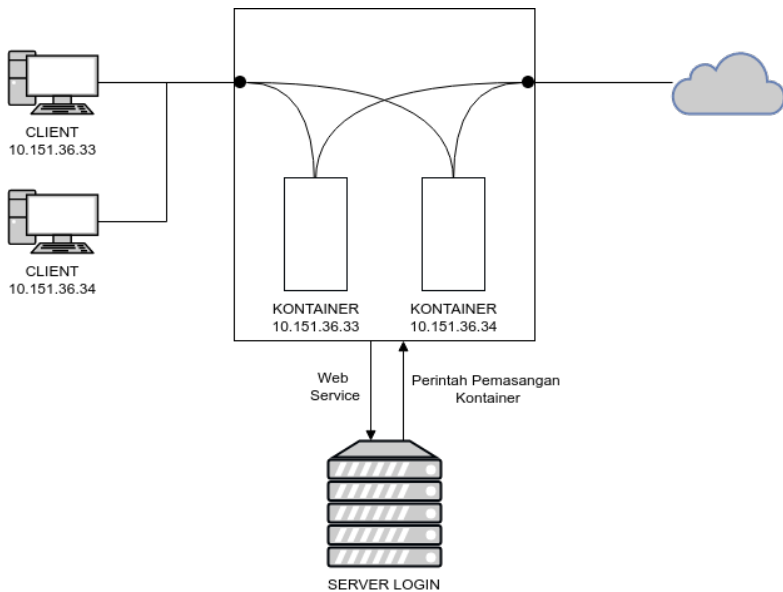
1. Pembuatan halaman *login* dari sebuah sistem.
2. Pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem.
3. Pembuatan *middleware* untuk menerima permintaan dari *client*.
4. Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client*.
5. Pemasangan kontainer pada *docker host*.
6. Pembuatan halaman *administrator* untuk membaca *log file* dari *client*.
7. Pembuatan *schedule* pada *docker host*.

Untuk memenuhi kebutuhan sistem tersebut, penulis membagi sistem menjadi beberapa komponen. Komponen yang akan dibangun antara lain:

1. Pembuatan halaman *login* dari sebuah sistem.  
Berfungsi sebagai tampilan antarmuka dari halaman *login* sebuah sistem untuk *client*. Selain itu juga berfungsi untuk mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*.
2. Pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem.  
Berfungsi untuk mengarahkan tiap *client* yang belum *login* ke dalam sistem, ke halaman *login* dari sistem. Hal ini dilakukan dengan menjalankan sebuah *script* dengan menggunakan *Iptables* pada *docker host*.
3. Pembuatan *middleware* untuk menerima permintaan dari *client*.  
Berfungsi untuk menerima permintaan pembuatan kontainer *docker* dari *client*. Selain itu juga berfungsi untuk membuat kontainer *docker* secara otomatis.

4. Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client*.  
Berfungsi untuk mengarahkan *traffic* dari tiap *client* yang telah berhasil *login*, ke kontainer *docker* dari tiap-tiap *client*. Hal ini dilakukan dengan menjalankan sebuah *script* dengan menggunakan *Iptables* pada *docker host*.
5. Pemasangan kontainer pada *docker host*.  
Berfungsi untuk memasang kontainer *docker* pada *docker host* secara otomatis. Hal ini dilakukan dengan menjalankan sebuah perintah penyediaan kontainer pada *docker host*.
6. Pembuatan halaman *administrator* untuk membaca *log file* dari *client*.  
Berfungsi untuk melihat apa saja yang telah diakses oleh *client*. *Log* yang tersimpan terdapat *log* HTTP maupun *log* HTTPS. Hal ini dilakukan dengan menjalankan sebuah perintah untuk melihat *log file* dari suatu *client*.
7. Pembuatan *schedule* pada *docker host*.  
Berfungsi untuk menghentikan akses internet dari *client* dengan cara menghapus kontainer *docker* dan *rules* *Iptables* sesuai dengan IP *client* tersebut.





**Gambar 3.2:** Arsitektur Komponen Sistem

Pada pada Gambar 3.2 ditunjukkan arsitektur sistem secara umum dengan detail-detail dari komponen yang terdapat didalamnya. Setiap komponen tersebut akan diimplementasikan dengan teknologi pendukung yang dibutuhkan.

Nantinya tiap *client* akan mempunyai satu kontainer *docker* dan satu *port* secara pribadi. *Traffic* dari *client* tersebut akan diarahkan menuju ke kontainer *dockernya* dari tiap-tiap *client*, setelah itu *client* baru dapat mengakses internet.

### 3.3.2 Pembuatan Halaman *Login* dari Sebuah Sistem.

Pembuatan halaman *login* dari sebuah sistem adalah komponen yang bertugas untuk menyediakan tampilan antarmuka dari halaman *login* untuk *client*. Awalnya semua *traffic* diarahkan menuju ke halaman *login* dari sebuah sistem,

karena diasumsikan bahwa semua *client* diasumsikan belum *login* ke dalam sistem. Supaya *client* dapat mengakses internet, maka *client* harus *login* ke dalam sistem terlebih dahulu dengan memasukkan *username* dan *password* dari *client* tersebut.

Dikarenakan ada beberapa kebutuhan yang harus dipenuhi, komponen pada pembuatan halaman *login* dari sebuah sistem dibagi lagi menjadi dua sub komponen, yaitu:

1. Basis Data

Basis data pada komponen pembuatan halaman *login* dari sebuah sistem berfungsi sebagai tempat penyimpanan data *username* dan *password* yang digunakan untuk *login* ke dalam sistem. Basis data juga berfungsi sebagai tempat penyimpanan data kontainer *docker* yang sudah dibuat.

2. *Web Service*

*Web service* berfungsi sebagai antarmuka untuk *client* ketika *client* akan *login* ke dalam sistem. Selain itu *web service* juga berfungsi untuk mengirimkan permintaan penyediaan kontainer *docker* ke *docker host* ketika terdapat *client* yang telah berhasil *login* ke dalam sistem.

Pada tugas akhir ini, bahasa Python dipilih sebagai bahasa pemrograman dan Flask dipilih sebagai kerangka kerja untuk bahasa pemrograman Python yang digunakan untuk mengimplementasikannya. Lalu, pada bagian penyimpanan data atau basis data, MySQL dipilih sebagai RDBMS untuk tugas akhir ini.

### 3.3.2.1 Desain Basis Data

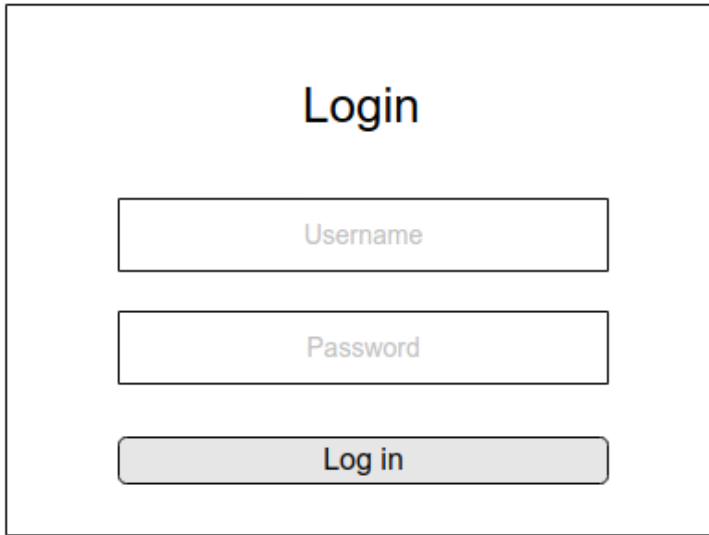
Komponen basis data berfungsi sebagai tempat penyimpanan data *username* dan *password* yang digunakan untuk *login* ke dalam sistem. Dalam basis data ini terdapat satu entitas dan empat atribut, ditunjukkan pada Tabel 3.2

**Tabel 3.2:** Atribut basis data nrp-mahasiswa

No	Kolom	Tipe	Keterangan
1	id	int(11)	Sebagai primary key pada tabel, nilai awal adalah <code>AUTO_INCREMENT</code> .
2	username	varchar(50)	Menunjukkan NRP dari mahasiswa yang telah terdaftar.
3	password	varchar(50)	Menunjukkan password dari NRP mahasiswa yang telah terdaftar.
4	created-at	timestamp	Menunjukkan waktu kapan <i>username</i> tersebut didaftarkan.
5	updated-at	timestamp	Menunjukkan waktu kapan <i>username</i> tersebut dilakukan pengubahan atau <i>update</i> .
5	deleted-at	timestamp	Menunjukkan waktu kapan <i>username</i> tersebut dihapus dari basis data.

### 3.3.2.2 Desain *Web Service*

Komponen *web service* berfungsi untuk menyediakan antar muka halaman *login* untuk *client* dan untuk mengirimkan permintaan pembuatan kontainer *docker* secara otomatis pada *docker host* setelah terdapat *client* yang berhasil *login* ke dalam sistem. Halaman *login* akan menggunakan Material UI untuk mendapatkan tampilan yang sederhana dan nyaman untuk digunakan. Desain *web service* untuk halaman *login* dari sistem dapat dilihat pada Gambar 3.3.

A login form design within a rectangular border. At the top center is the word "Login" in a large, bold, black font. Below it are two input fields: the first is labeled "Username" in a light gray font, and the second is labeled "Password" in a light gray font. At the bottom is a button labeled "Log in" in a black font, which has a light gray background and a black border.

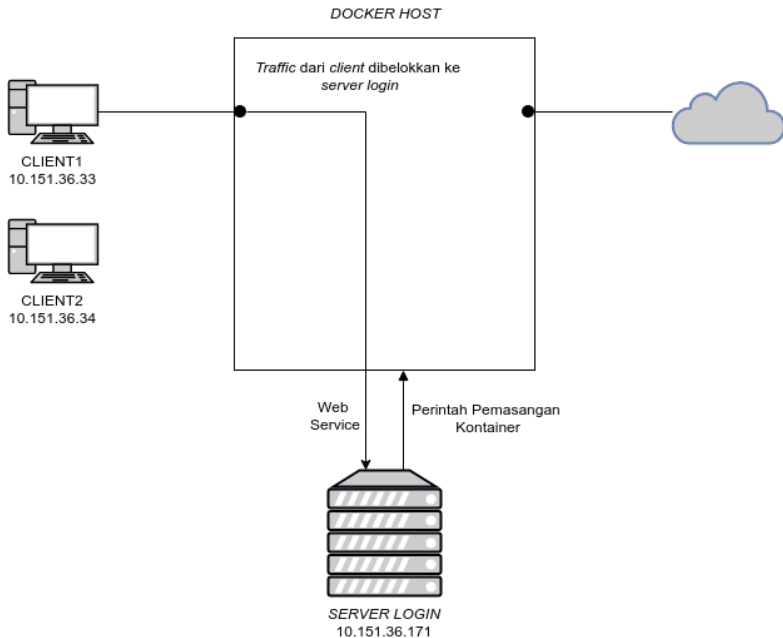
**Gambar 3.3:** Desain Halaman *Login*

Lalu untuk desain *backend* dari *web serve* untuk halaman *login* akan menggunakan bahasa pemrograman Python dengan kerangka kerja Flask yang dijalankan pada *port* 4000.

### **3.3.3 Perancangan Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Halaman *Login* dari Sistem**

Pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem adalah komponen yang bertugas untuk membelokkan *traffic* dari *client* yang akan menuju ke internet. Awalnya semua *traffic* dari satu *subnet client* tersebut akan diarahkan ke halaman *login* dari sistem dengan membuat sebuah aturan menggunakan *Iptables*, karena asumsinya adalah belum ada *client* yang berhasil *login* ke dalam sistem. Desain perancangan pembuatan aturan untuk mengarahkan *traffic client*

ke halaman *login* dapat dilihat pada Gambar 3.4 .



**Gambar 3.4:** Desain Mengarahkan *Traffic Client* ke Halaman *Login*

### 3.3.4 Pembuatan *Middleware* untuk Menerima Permintaan dari *Client*

Pembuatan *middleware* untuk menerima permintaan dari *client* adalah komponen yang bertugas untuk menerima permintaan dari *client* yang telah berhasil *login* ke dalam sistem. Permintaan yang dikirimkan oleh *client* adalah permintaan untuk membuat kontainer *docker* secara otomatis. Nantinya setiap satu *client* yang berhasil *login* ke dalam sistem akan dibuatkan satu kontainer *docker*.

Dikarenakan ada beberapa kebutuhan yang harus dipenuhi, komponen pada pembuatan *middleware* untuk menerima

permintaan dari *client* dibagi lagi menjadi dua buah komponen, yaitu:

1. Basis Data

Basis data berfungsi sebagai tempat penyimpanan data kontainer *docker* yang sudah dibuat.

2. *Web Service*

*Web service* berfungsi sebagai penerima permintaan dari *client*, yang nantinya akan membuat sebuah kontainer *docker* secara otomatis pada *docker host*.

Sama seperti komponen pembuatan halaman *login* dari sebuah sistem, pada tugas akhir ini, bahasa Python dipilih sebagai bahasa pemrograman yang digunakan untuk mengimplementasikannya. Lalu, pada bagian penyimpanan data atau basis data, MySQL dipilih sebagai RDBMS untuk tugas akhir ini.

### 3.3.4.1 Desain Basis Data

Komponen basis data berfungsi sebagai tempat penyimpanan data kontainer *docker* yang sudah dibuat. Dalam basis data ini terdapat satu entitas dan empat atribut, ditunjukkan pada Tabel 3.3.

**Tabel 3.3:** Atribut basis data kontainer

No	Kolom	Tipe	Keterangan
1	id	int(11)	Sebagai primary key pada tabel, nilai awal adalah <code>AUTO_INCREMENT</code> .
2	username	varchar(50)	Menunjukkan NRP dari mahasiswa yang telah berhasil dibuatkan satu kontainer <i>docker</i> .

**Tabel 3.3:** Atribut basis data kontainer

No	Kolom	Tipe	Keterangan
3	ip	varchar(50)	Menunjukkan IP dari <i>client</i> yang telah berhasil dibuatkan satu kontainer <i>docker</i> .
4	port	varchar(50)	Menunjukkan <i>port</i> dari <i>client</i> yang telah berhasil dibuatkan satu kontainer <i>docker</i> .
5	created-at	timestamp	Menunjukkan waktu pertama kali kontainer <i>docker</i> tersebut dibuat.

### 3.3.4.2 Desain *Web Service*

Komponen *web service* berfungsi untuk menerima permintaan dari *client* untuk membuat satu kontainer *docker* pada *docker host*. Kontainer *docker* yang akan dibuat pada *docker host* akan dibuat secara otomatis oleh sistem, dan kontainer *docker* yang dibuat akan memiliki nama sesuai dengan IP dari *client* yang telah berhasil *login* ke dalam sistem. Setelah menerima permintaan dari *client*, maka sistem akan mengirimkan perintah untuk membuat kontainer *docker* khusus untuk satu *client*.

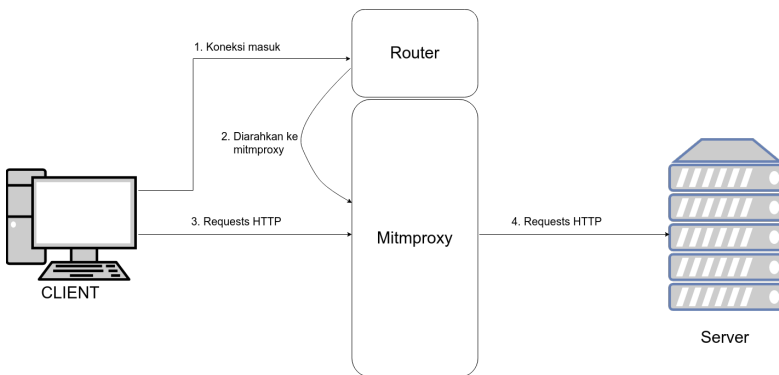
### 3.3.5 Perancangan Pemasangan Kontainer pada *Docker Host*

Pemasangan kontainer adalah komponen yang berfungsi untuk memasang kontainer *docker* yang berisi Mitmproxy pada *docker host* setelah ada permintaan dari *client* yang telah berhasil *login* ke dalam sistem. Proses ini dilakukan secara otomatis, dan nama

dari kontainer *docker* tersebut akan sesuai dengan IP dari *client* yang telah berhasil *login* ke dalam sistem.

Saat kontainer *docker* telah berhasil dibuat, maka kontainer *docker* tersebut akan mempunyai satu *port* khusus yang sama dengan *client* yang baru saja *login*. *Port* khusus nantinya akan digunakan untuk mengarahkan *traffic* dari *client* yang akan mengakses internet.

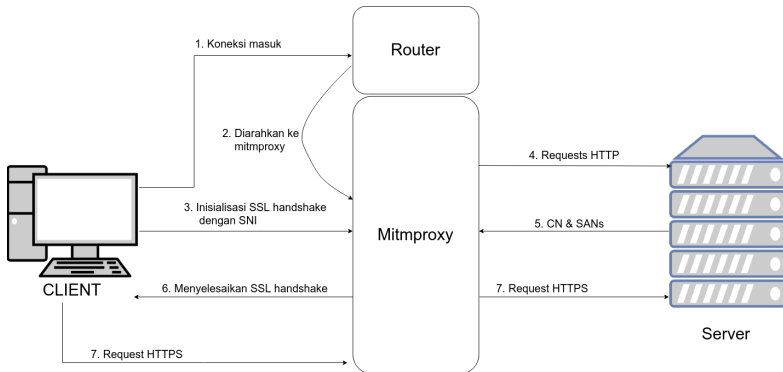
*Image* Mitmproxy dipilih sebagai *image* pada kontainer *docker* karena Mitmproxy merupakan sebuah perangkat lunak yang interaktif dimana Mitmproxy memungkinkan dapat memotong dan memodifikasi HTTP *requests* atau *response* dengan sangat cepat. Mitmproxy sendiri juga dapat berjalan dengan *transparent mode* sehingga *client* tidak mengetahui jika *traffic* dari *client* tersebut ternyata melalui Mitmproxy. Gambar alur kerja dari Mitmproxy dengan *transparent mode* untuk HTTP dapat dilihat pada Gambar 3.5.



**Gambar 3.5:** Alur kerja dari *mitmproxy transparent* HTTP

Sedangkan gambar alur kerja dari Mitmproxy dengan *transparent mode* untuk HTTPS dapat dilihat pada Gambar 3.6.

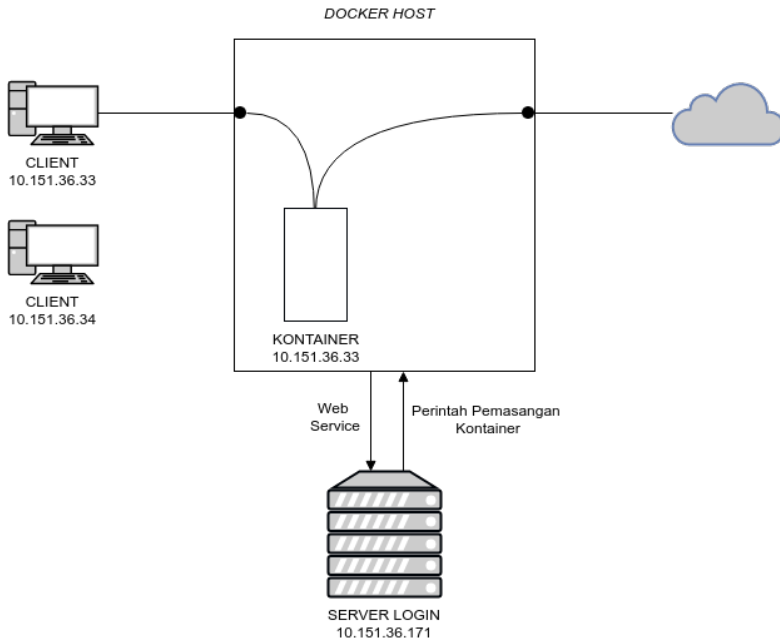




**Gambar 3.6:** Alur kerja dari *mitmproxy* transparent HTTPS

### 3.3.6 Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Kontainer *Docker* dari Tiap-Tiap *Client*

Pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client* adalah komponen yang bertugas untuk mengarahkan satu client ke satu kontainer *docker* yang sesuai. Setelah *client* berhasil *login* ke dalam sistem, maka aturan ini akan dibuat menggunakan *Iptables*. Lalu *client* juga akan diberikan sebuah aturan dengan menggunakan *Iptables* yang memperbolehkan *client* tersebut mengakses internet. Desain dari pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap *client* dapat dilihat pada Gambar 3.7.

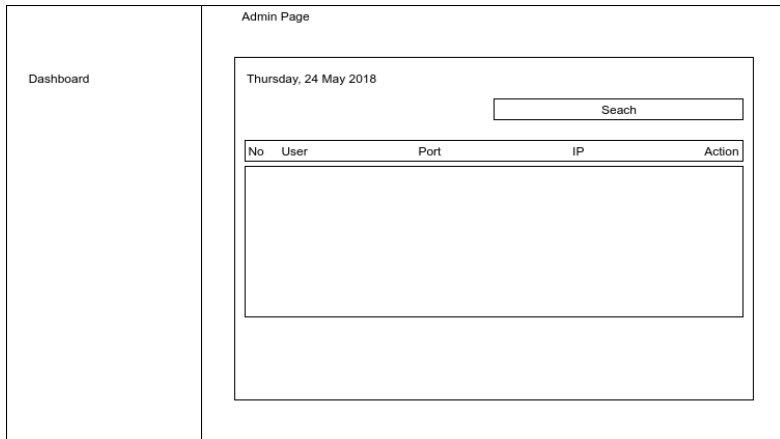


**Gambar 3.7:** Desain pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker*

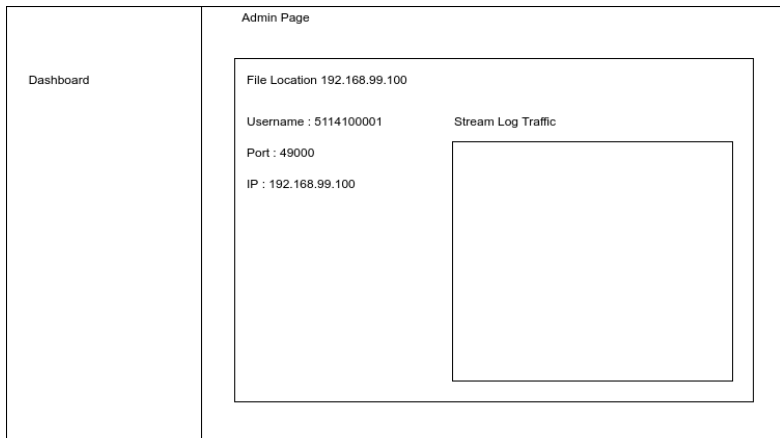
### 3.3.7 Pembuatan Halaman *Administrator* untuk Membaca *Log File* dari *Client*

Pembuatan halaman *administrator* untuk membaca *log file* dari *client* adalah komponen yang bertugas untuk mengunduh *log file* dari *client* dan juga untuk membaca *log file* dari *client* secara langsung atau *live* maupun hanya pada saat *client* terakhir mengakses internet.

Halaman *administrator* akan menggunakan Bootstrap 4 untuk mendapatkan tampilan yang sederhana dan nyaman untuk digunakan. Desain dari halaman *administrator* dapat dilihat pada Gambar 3.8 dan Gambar 3.9 .



**Gambar 3.8:** Desain halaman dashboard *administrator traffic client* ke kontainer *docker*



**Gambar 3.9:** Desain pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker*

### 3.3.8 Pembuatan *Schedule* pada *Docker Host*

Pembuatan *schedule* pada *docker host* adalah komponen yang bertugas untuk melakukan *reboot* sistem secara keseluruhan pada *docker host*. Dalam hal ini, seluruh kontainer *docker* maupun *rules* yang sudah dibuat akan dihapuskan. *Schedule* ini akan dijalankan setiap hari pada pukul 23.59. Jika terdapat *client* yang telah berhasil *login* ke dalam sistem sebelum pukul 23.59, lalu *client* tersebut menggunakan akses internet lebih dari pukul 23.59 atau telah berganti keesokan harinya, maka akses menggunakan internet dari *client* tersebut akan terputus. Jika *client* tersebut ingin mengakses internet lagi, maka *client* tersebut harus melakukan *login* kembali ke dalam sistem.

## **BAB IV**

### **IMPLEMENTASI**

Setelah melewati proses perancangan mengenai sistem yang akan dibuat, maka akan dilakukan implementasi dari sistem tersebut. Bab ini akan membahas mengenai implementasi dari sistem yang meliputi proses pembuatan setiap komponen sehingga sistem dapat berjalan dengan baik. Masing-masing proses pembuat komponen akan dilengkapi dengan *pseudocode* atau konfigurasi dari sistem.

#### **4.1 Lingkungan Implementasi**

Dalam mengimplementasikan sistem, digunakan beberapa perangkat pendukung sebagai berikut.

##### **4.1.1 Perangkat Keras**

Perangkat keras yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Komputer dengan *processor* Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz dan RAM 8GB
2. Komputer dengan *processor* Intel(R) Core(TM)2 Duo CPU E7200 @ 2.53GHz dan RAM 1GB

##### **4.1.2 Perangkat Lunak**

Perangkat lunak yang digunakan dalam pengembangan sistem adalah sebagai berikut:

1. Sistem Operasi Linux Mint 18.03 64 Bit sebagai *docker host*.
2. Sistem Operasi Ubuntu 16.04 LTS 64 Bit sebagai *client*.
3. Sistem Operasi Debian 8.6.0 64 Bit sebagai *server login*.
4. Python versi 2.7.12 untuk pengembangan web service.
5. Flask versi 1.0.2 sebagai kerangka kerja Python.0
6. Nginx versi 1.10.3

7. Mitmproxy versi 3.0.4 untuk mencatat semua *traffic* dari *client*.
8. MySQL versi 5.7.18 untuk Sistem Manajemen Basis Data.
9. Docker versi 1.13.1 sebagai kontainer yang akan di pasangkan pada *server*.
10. Iptables versi 1.6.0 untuk membuat aturan terhadap *client*.

## 4.2 Implementasi Pembuatan Halaman *Login* dari Sebuah Sistem

Halaman *login* dibangun pada sebuah *server* dengan IP 10.151.36.171. Pada implementasi pembuatan halaman *login* dari sebuah sistem menggunakan perangkat lunak antara lain:

1. Python versi 3.5.2.
2. Flask versi 1.0.2.

Lalu sistem operasi yang digunakan adalah sistem operasi Debian 8.6.0 64 Bit, yang akan dipasang pada *virtual machine* dengan menggunakan VirtualBox. Python akan berfungsi sebagai komponen dasar pembangunan sistem yang akan dibangun dengan menggunakan kerangka kerja Flask dengan IP 10.151.36.171 dan *port* 4000. Implementasi pembuatan halaman *login* dari sebuah sistem akan terbagi menjadi implementasi *web service* dan implementasi basis data.

### 4.2.1 Implementasi *Web Service* pada Halaman *Login*

Diperlukan beberapa tahap, antara lain pemasangan perangkat lunak dan tahap konfigurasi. Tahap pemasangan perangkat lunak dan tahap konfigurasi pada *server* untuk halaman *login* dijelaskan pada Lampiran A.

Pada sub-bab implementasi *web service* pada halaman *login* akan dibagi lagi menjadi tiga bagian, antara lain implementasi tampilan antarmuka halaman *login*, rute *web service* pada halaman *login* dan *pseduocode web service* pada halaman *login*.

#### 4.2.1.1 Implementasi Tampilan Antarmuka Halaman *Login*

Halaman *login* merupakan halaman utama yang menampilkan sebuah *form input* untuk *client*. Pada halaman ini terdapat dua *form input*, yaitu *form input* untuk `Username` atau NRP dari *client* dan juga *form input* untuk `Password` dari *client*. Implementasi antarmuka halaman *login* dapat dilihat pada Gambar 4.1.

ITS  
Institut Teknologi Sepuluh Nopember

myITS  
Sistem Informasi Terintegrasi - ITS

Username

Password

Login

Lupa password?

1. Tidak password Anda akan berlaku. Cetak password minimal 8 karakter (huruf kapital, huruf kecil, dan angka).

2. Jangan lupa password yang baru untuk mencegah risiko yang tidak terduga terhadap akun yang tidak diinginkan pada data Anda.

3. Jangan menggunakan tanggal lahir atau hal-hal umum lainnya sebagai password.

Gambar 4.1: Halaman *Login*

#### 4.2.1.2 Rute *Web Service* pada Halaman *Login*

Pada halaman *login* diperlukan adanya rute-rute yang bisa diakses untuk melayani *client*, supaya *client* dapat membuka tampilan antar muka dari halaman *login* dan juga supaya *client* dapat mengirimkan permintaan untuk membuat kontainer *docker* pada *docker host*. Daftar rute yang disediakan oleh halaman *login* tertera pada Tabel 4.1.

Tabel 4.1: Daftar Rute *Web Service*

HTTP Method	Rute	Deskripsi
GET	/	Berfungsi untuk mengarahkan <i>redirect</i> ke rute <i>login</i> dengan <i>method</i> GET.
GET	/login	Berfungsi untuk menampilkan tampilan grafis antar muka halaman <i>login</i> ketika <i>client</i> belum <i>login</i> ke dalam sistem dan untuk menampilkan tampilan grafis antar muka halaman sukses <i>login</i> ketika <i>client</i> telah berhasil <i>login</i> ke dalam sistem.
POST	/login	Berfungsi untuk menyimpan data hasil <i>input</i> dari <i>client</i> dan mengirimkan perintah untuk membuat kontainer <i>docker</i> yang berisikan Mitmproxy secara otomatis pada <i>docker host</i> .



**Tabel 4.1:** Daftar Rute *Web Service*

<b>HTTP Method</b>	<b>Rute</b>	<b>Deskripsi</b>
GET	/logout	Berfungsi untuk keluar dari sistem dan kontainer <i>docker</i> yang sudah dibuat untuk <i>client</i> yang telah berhasil <i>login</i> dengan menggunakan <i>username</i> tersebut akan di- <i>destroy</i> . Lalu aturan-aturan yang sudah dibuat dengan menggunakan <i>Iptables</i> untuk <i>client</i> tersebut juga dihapuskan.

#### 4.2.1.3 *Pseudocode Web Service* pada Halaman *Login*

Ketika *client* belum *login* ke dalam sistem, maka akan diarahkan ke tampilan grafis antar muka dari halaman *login*. Lalu setelah *client* berhasil *login* ke dalam sistem, maka akan diarahkan ke tampilan grafis antar muka halaman sukses *login*. Pada Kode Sumber 4.1 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

**Kode Sumber 4.1: Pseudocode Web Service**

```
1   Check whether the client is already login or
    not yet
2
3   if session.get login
4       open welcome page
5
6   else
7       open login page
8
9       if login success
10          session.get login = True
11          send request to docker host
12
13  return
```

#### 4.2.2 Implementasi Basis Data pada Halaman *Login*

Berdasarkan hasil desain dan perancangan basis data pada bab 3 terdapat satu entitas yang diimplementasikan menjadi suatu tabel pada basis data MySQL, yaitu entitas *nrp-mahasiswa*. Detail implementasi *query* untuk membuat basis data dengan entitas *nrp-mahasiswa* seperti pada Gambar 4.2.

```
CREATE TABLE nrp-mahasiswa (
  id int(11) PRIMARY KEY AUTO_INCREMENT,
  nrp VARCHAR(50)
  password VARCHAR(50)
  isLogin int(11)
);
```

**Gambar 4.2:** *Query* untuk membuat tabel kontainer

### 4.3 Implementasi Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Halaman *Login* dari Sistem

Pada implementasi pembuatan aturan untuk mengarahkan *traffic client* ke halaman *login* dari sistem diasumsikan bahwa belum ada *client* yang telah *login* ke dalam sistem. Karena diasumsikan bahwa belum ada *client* yang telah berhasil *login* ke dalam sistem, maka semua *client* tidak diperbolehkan untuk mengakses internet. Kemudian untuk mengarahkan *traffic* dari *client* dibuatkan beberapa *rules* dengan menggunakan *iptables* pada *Docker Host* dengan IP 10.151.36.134. seperti Gambar 4.3.

```
iptables -I FORWARD 1 -s 192.168.99.0/24 -j REJECT
iptables -I FORWARD 1 -s 192.168.99.0/24 -p tcp d 10.151.36.173 --
    dport 4000 -j ACCEPT
iptables -t nat -I PREROUTING 1 -p tcp -s 192.168.99.0/24 --dport
    80 -j DNAT --to 10.151.36.173:4000
```

**Gambar 4.3:** Command untuk mengarahkan *client* ke halaman *login*

*Rules* pertama berfungsi untuk melarang semua *client* untuk melewati *router*. *Rules* kedua berfungsi untuk mengizinkan semua *client* membuka halaman *login*. Sedangkan *rules* ketiga berfungsi untuk mengarahkan semua *traffic client* ke halaman *login*.

### 4.4 Implementasi Pembuatan *Middleware*

*Middleware* dibangun pada *Docker Host* dengan IP 10.151.36.134 dengan menggunakan sistem operasi Linux Mint 18.03 64 Bit. *Middleware* merupakan komponen yang akan menerima permintaan dari *client*, mengirimkan perintah untuk membuat kontainer *docker* secara otomatis pada *docker host*, dan menentukan rute *traffic* dari *client* menuju ke internet sesuai kontainer *docker* masing-masing user. Implementasi *middleware*

akan terbagi menjadi implementasi basis data dan implementasi *web service*.

Pada implementasi pembuatan *middleware* menggunakan perangkat lunak antara lain:

1. Python versi 2.7.12.
2. Flask versi 1.0.2.
3. Docker versi 1.13.1.

Python akan berfungsi sebagai komponen dasar pembangunan sistem, salah satunya adalah sebagai komponen dasar pembuatan *middleware*, sedangkan Flask akan berfungsi sebagai kerangka kerja untuk pembuatan *middleware*. Implementasi pembuatan *middleware* akan terbagi menjadi implementasi *web service* dan implementasi basis data dan.

#### **4.4.1 Implementasi Web Service pada Middleware**

Diperlukan beberapa tahap, antara lain pemasangan perangkat lunak dan tahap konfigurasi. Tahap pemasangan perangkat lunak dan tahap konfigurasi pada *middleware* di *Docker Host* dijelaskan pada Lampiran A.

Pada sub-bab implementasi *web service* pada *middleware* akan dibagi lagi menjadi dua bagian, antara lain rute *web service* pada *middleware* dan juga *pseudocode web service* pada *middleware*.

##### **4.4.1.1 Rute Web Service pada Middleware**

*Middleware* tidak memiliki antar muka grafis. Namun tetap diperlukan adanya rute-rute yang bisa diakses untuk melayani permintaan penyediaan kontainer *docker* dari *client*. Daftar rute yang disediakan oleh *middleware* tertera pada Tabel 4.2.

**Tabel 4.2:** Daftar Rute *Web Service*

<b>HTTP Method</b>	<b>Rute</b>	<b>Deskripsi</b>
POST	/test/endpoint/	Berfungsi untuk menyimpan data hasil <i>input</i> dari <i>client</i> dan mengirimkan perintah untuk membuat kontainer <i>docker</i> yang berisikan <i>mitmproxy</i> secara otomatis pada <i>docker host</i> .
POST	/logout/endpoint/	Berfungsi untuk menerima permintaan penghentian akses internet dari <i>client</i> sendiri dan juga untuk menjalankan perintah penghapusan kontainer <i>docker</i> dan <i>rules</i> dari <i>client</i> tersebut. Setelah pada basis data kontainer akan dilakukan pembaruan <i>status</i> dari kontainer yang telah digunakan oleh <i>client</i> tersebut menjadi 0.

#### 4.4.1.2 *Pseudocode Web Service pada Middleware*

Saat *client* telah memasukkan *input* ke sistem, sistem akan mencocokkan terlebih dahulu dengan basis data kontainer. Jika benar, maka sistem akan mengirimkan data *input* dari *client* ke *middleware*. Lalu *middleware* akan menyimpan data *input*

dari *client* ke dalam sebuah *file*. Setelah itu *middleware* akan mengirimkan perintah untuk membuat sebuah kontainer *docker* yang berisikan Mitmproxy pada *docker host*.

Saat *middleware* menyimpan data *input* dari *client* ke dalam sebuah *file*, yang disimpan adalah *username* atau NRP, *IP Address*, dan *port*. Nantinya *port* tersebut akan menjadi *port* khusus untuk kontainer *docker* yang berisikan *mitmproxy* untuk *client* tersebut.

Saat kontainer *docker* yang berisikan *mitmproxy* akan dibuat pada *docker host*, sistem akan membuat kontainer *docker* dengan *mode network=host*, nama sesuai *IP Address* dari *client* tersebut, dan *port* kontainer *docker* sesuai dengan *port* yang sudah disimpan pada *file*.

Setelah kontainer *docker* yang berisikan *mitmproxy* berhasil dibuat, maka sistem akan membuat *rules* yang berfungsi untuk mengarahkan *traffic* dari *client* menuju ke kontainer *docker* milik *client* tersebut, dan memperbolehkan *client* untuk mengakses internet. Pada Kode Sumber 4.2 diperlihatkan bagaimana implementasinya dalam bentuk *pseudocode*.

**Kode Sumber 4.2:** Pseudocode Web Service

```
1  Check whether the client is already login or  
   not yet  
2  
3  if session.get login  
4      create container  
5      add new rules to container  
6  
7  else  
8      add new rules to page login  
9  
10 return
```

#### 4.4.2 Implementasi Basis Data pada *Middleware*

Berdasarkan hasil perancangan basis data pada bab 3 terdapat 2 entitas yang diimplementasikan menjadi suatu tabel pada basis data MySQL, yaitu entitas `kontainer`. Detail implementasi entitas `kontainer` tertera pada Gambar ??.

#### 4.5 Implementasi Pemasangan Kontainer Docker pada *Docker Host*

Setelah berhasil melakukan pemasangan *docker* versi 1.13.1 pada *docker host* dengan IP 10.151.36.134, sekarang lakukan konfigurasi supaya *docker* tidak hanya dapat digunakan oleh *root user* dari sebuah sistem. Hal ini dapat dilakukan dengan menjalankan perintah pada Gambar 4.4.

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

**Gambar 4.4:** Perintah memasukkan *user* ke grup *docker*

##### 4.5.1 Menambahkan dan Memperbarui Kontainer Docker yang Berisikan Mitmproxy

Setelah berhasil melakukan pemasangan *docker* pada *docker host* dan melakukan konfigurasi supaya *docker* tidak hanya dapat digunakan oleh *root user* dari sebuah sistem, selanjutnya dapat mencoba membuat sebuah kontainer *docker* yang berisi aplikasi Mitmproxy. Untuk membuat sebuah kontainer *docker* yang berisi Mitmproxy, penulis melakukannya dengan sistem operasi Ubuntu dalam format *docker* yang disediakan oleh *Docker Hub*. Untuk melakukan unduh, jalankan perintah `docker pull ubuntu`.

Setelah berhasil diunduh, selanjutnya jalankan sistem operasi Ubuntu dengan menggunakan perintah `docker run --name`

```
testmitmproxy --privileged=True --network=host  
ubuntu.
```

Parameter `--name` berguna untuk memberikan nama pada kontainer *docker* agar mudah dikenali dimana lokasi aplikasi saat dijalankan. Pada kasus ini kontainer *docker* diberi nama dengan `testmitmproxy`. Parameter `--privileged=True` berguna untuk memberikan kendali hak akses penuh kepada kontainer *docker* tersebut, sama seperti dengan *root user*. Parameter `--network=host` berguna untuk mendefinisikan jaringan yang akan digunakan oleh kontainer *docker* tersebut. Setelah menjalankannya, kontainer *docker* yang terbentuk dapat digunakan lebih lanjut, misalnya dengan mengubah data yang ada didalamnya, menambahkan fitur baru, atau hanya sekedar mengganti nama dari aplikasi.

Dalam kasus ini penulis menambahkan fitur baru, yaitu menambah Mitmproxy. Untuk menambah atau memasang Mitmproxy pada kontainer *docker* yang baru saja dibuat, jalankan perintah berikut pada Gambar 4.5.

```
sudo apt-get update  
sudo apt-get install python3 python3-dev python3-pip  
sudo pip3 install cryptography  
sudo pip3 install mitmproxy
```

**Gambar 4.5:** Perintah untuk Pemasangan Mitmrproxy

Mitmproxy versi 3.0.4 membutuhkan Python minimal versi 3.5, maka dari itu penulis memasang Python versi 3.5.2. Mitmrproxy juga membutuhkan modul *cryptography* yang berguna untuk melakukan enkripsi maupun dekripsi ketika Mitmproxy sedang berjalan. Lalu aktifkan `ipv4.forwarding` dengan menjalankan perintah `sudo sysctl -w net.ipv4.ip-forward=1`.

Setelah berhasil melakukan pemasangan Mitmproxy pada kontainer *docker*, jika ingin membuat *images* baru dari kontainer



*docker* tersebut, maka hal pertama yang harus dilakukan adalah menghentikan kontainer *docker* yang sedang berjalan dengan menggunakan perintah `docker stop [nama container]`.

Nama *container* ini tergantung dari nama kontainer *docker* yang sudah dibuat. Untuk kasus yang digunakan oleh penulis, penulis menggunakan perintah `docker stop testmitmproxy`. Setelah itu lakukan *commit* dengan menjalankan perintah `docker commit [nama container] [nama repository]`.

Nama *container* ini tergantung dari nama kontainer *docker* yang sudah dibuat. Sedangkan nama *repository* ini tergantung dari nama *repository* yang telah dibuat di Docker Hub. Untuk kasus yang digunakan oleh penulis, penulis menggunakan perintah `docker commit testmitmproxy fourirakbar/mitmproxy-oing:version1`. Pada bagian nama *repository* ini memiliki tiga bagian dengan pola seperti `[URL]/[nama]:[versi]`. Artinya membuat *image* dengan URL *repository* pada Docker Hub dengan nama `fourirakbar`. Kemudian nama dari *image*-nya sendiri adalah `mitmproxy-oing` dan versinya adalah `version1`. Setelah melakukan *commit*, maka *image* baru akan terbentuk. Langkah terakhir adalah melakukan *push image* ke Docker Hub dengan menggunakan perintah `docker push [nama container] [nama repository]`.

#### 4.5.2 Menggunakan *Image* Kontainer Docker yang Sudah Dibuat

Setelah berhasil menambahkan dan memperbarui kontainer *docker* yang berisi Mitmproxy, penulis tidak perlu melakukannya lagi. Penulis hanya perlu memanggil kontainer *docker* dengan menjalankan perintah `docker pull fourirakbar/mitmproxy-oing:version1`.

Lalu untuk menjalankan kontainer *docker* yang sudah di *pull*, jalankan perintah `docker run --name [IP CLEINT]`

```
--privileged=True --network=host
fourirakbar/mitmproxy-oing:version1.
```

#### 4.6 Implementasi Pembuatan Aturan untuk Mengarahkan *Traffic Client* ke Kontainer Docker dari Tiap-Tiap *Client*

Pada implementasi pembuatan aturan untuk mengarahkan *traffic client* ke kontainer *docker* dari tiap-tiap client dibuat ketika terdapat *client* yang telah berhasil *login* ke dalam sistem. Setelah *client* berhasil *login* ke dalam sistem, maka akan dibuatkan beberapa *rules* dengan menggunakan Iptables seperti Gambar 4.6.

```
iptables -I FORWARD 1 -s [IP_CLIENT] -j ACCEPT
iptables -t nat -I PREROUTING 1 -s [IP_CLIENT] -p tcp --dport 80 -j
  REDIRECT --to-ports [PORTS_CLIENT]
iptables -t nat -I PREROUTING 1 -s [IP_CLIENT] -p tcp --dport 443 -
  j REDIRECT --to-ports [PORTS_CLIENT]
iptables -t nat -I POSTROUTING 1 -o wlp3s0 -j MASQUERADE -s [
  IP_CLIENT]
```

**Gambar 4.6:** Command untuk mengarahkan *client* ke halaman *login*

Pada *rules* pertama berfungsi untuk mengizinkan atau memperbolehkan *traffic* dari *client* melewati *router*. Lalu *rules* kedua dan ketiga berfungsi untuk mengarahkan *traffic client* ke kontainer *docker* yang sudah dibuat dengan satu port khusus untuk *client* tersebut. Lalu *rules* keempat berfungsi untuk mengizinkan atau memperbolehkan *client* untuk mengakses internet.

#### 4.7 Implementasi Pembuatan Halaman *Administrator*

Halaman *administrator* dibangun pada *Docker Host* dengan IP 10.151.36.134 dengan port 5001. Fungsi dari halaman *administrator* adalah untuk melihat siapa saja *client* yang

berhasil *login* ke dalam sistem dan megakses internet, juga untuk melihat rekap *client* yang telah berhasil *login* dan mengakses internet. Selain itu juga dapat menghentikan akses penggunaan internet dari *client*. Pada sub-bab ini akan dibagi lagi menjadi beberapa bagian, antara lain rute *web service* pada halaman *administrator*, implementasi pembacaan *log file* dari *client* dan implementasi antarmuka.

#### 4.7.1 Rute *Web Service* pada Halaman *Administrator*

Pada halaman *administrator* diperlukan adanya rute-rute yang bisa diakses untuk melayani permintaan dari *user* yang sedang membuka hallaman *administrator*, yaitu untuk melihat *log* dari *client*. Daftar rute yang disediakan tertera pada Tabel 4.3.

**Tabel 4.3:** Daftar Rute *Web Service* pada Halaman *Administrator*

HTTP Method	Rute	Parameter	Deskripsi
GET	/table	-	Berfungsi untuk menampilkan halaman <i>dashboard</i> yang menunjukkan siapa saja <i>client</i> yang telah berhasil <i>login</i> ke dalam sistem dan sedang aktif mengakses internet.

**Tabel 4.3:** Daftar Rute *Web Service*

<b>HTTP Method</b>	<b>Rute</b>	<b>Parameter</b>	<b>Deskripsi</b>
GET	/stream/	id, ip, user, port	Berfungsi untuk melihat <i>log</i> dari <i>client</i> secara langsung atau <i>live</i> dengan memasukkan parameter berupa id, ip, user, dan port.
GET	/lihat/	id, ip, user, port	Berfungsi untuk melihat <i>log</i> dari <i>client</i> sampai dengan terakhir <i>client</i> mengakses internet dengan memasukkan parameter berupa id, ip, user, dan port.
GET	/history	-	Berfungsi untuk menampilkan rekap atau daftar <i>client</i> yang telah berhasil <i>login</i> ke dalam sistem.

Tabel 4.3: Daftar Rute *Web Service*

HTTP Method	Rute	Parameter	Deskripsi
GET	/stop/	id, ip, user, port	Berfungsi untuk menghentikan akses penggunaan internet dari sebuah <i>client</i> dengan memasukkan parameter berupa id, ip, user, dan port.

#### 4.7.2 Implementasi Pembacaan *Log File* dari *Client*

Pada implementasi pembacaan *log file* dari *client* dilakukan dengan membaca *file* hasil *output* dari *mitmproxy*. *File* hasil *output* dari *mitmproxy* berbentuk *binary* dimana yang bisa membacanya hanya komputer saja. Maka dari itu perlu dilakukan pembacaan lagi *file* yang berisi *binary* tersebut dengan menjalankan *command* `mitmdump -nr [NAMA-FILE] --set flow-detail=2 --showhost > [NAMA-FILE]`.

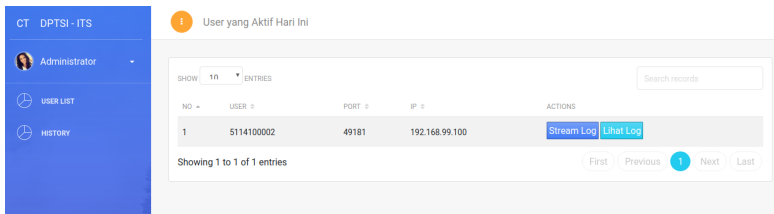
Sedangkan untuk membaca *log file* dari *client* secara langsung atau *live* dapat dilakukan dengan menjalankan *command* `tail -f -c +0 [NAMA FILE] | mitmdump -n -r - --set flow detail=1 --showhost`.

Parameter `-nr` berfungsi untuk tidak menjalankan *proxy server* dari *mitmproxy* sendiri, dan juga berfungsi untuk melakukan analisa dari *file output mitmproxy* yang berbentuk *binary*. Sedangkan parameter `--set flow detail` berfungsi untuk menampilkan detail dari analisa yang dilakukan oleh *mitmproxy*. Terdapat tingkat satu sampai dengan tiga, semakin tinggi tingkat yang diberikan maka semakin jelas detail dari *log*

yang dianalisa oleh *mitmproxy*. Lalu parameter `--showhost` berfungsi untuk menampilkan *header* dari URL yang telah diakses oleh *client*.

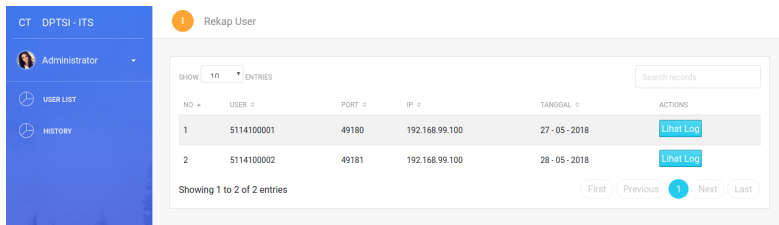
### 4.7.3 Implementasi Antarmuka Halaman *Administrator*

Halaman *administrator* diperuntukkan bagi *User* yang mempunyai akses ke *Docker Host*. Halaman ini berguna sebagai *dashboard* dari *administrator*. Pada halaman ini menampilkan siapa saja *Client* yang telah berhasil *login* ke dalam sistem. Terdapat dua *button*, yaitu *Stream Log* yang berfungsi untuk melihat secara langsung atau *live log* dari *client*. Lalu terdapat *button* Lihat Log yang berfungsi untuk melihat *log* dari *client* sampai terakhir *client* tersebut mengakses internet.



**Gambar 4.7:** Halaman *Administrator* Menu *User List*

Pada bagian *sidebar* terdapat dua menu yaitu *User List* dan *History*. Menu *User List* berguna untuk melihat *client* yang telah berhasil *login* ke dalam sistem. Sedangkan menu *history* berguna untuk melihat rekam *client* yang telah berhasil *login* dan mengakses internet pada hari-hari sebelumnya. Implementasi antarmuka halaman *administrator* pada menu *User List* dapat dilihat pada Gambar 4.7. Dan implementasi antarmuka halaman *administrator* oada nenu *History* dapat dilihat pada Gambar 4.8



CT DPTSI - ITS

Administrator

USER LIST

HISTORY

Rekap User

SHOW 10 ENTRIES

Search records

NO	USER	PORT	IP	TANGGAL	ACTIONS
1	5114100001	49180	192.168.99.100	27-05-2018	<a href="#">Lihat Log</a>
2	5114100002	49181	192.168.99.100	28-05-2018	<a href="#">Lihat Log</a>

Showing 1 to 2 of 2 entries

First Previous 1 Next Last

**Gambar 4.8:** Halaman *Administrator* Menu *History*

*(Halaman ini sengaja dikosongkan)*



## BAB V

### PENGUJIAN DAN EVALUASI

Pada bab ini akan dibahas uji coba dan evaluasi dari sistem yang telah dibuat. Sistem akan diuji coba fungsionalitas dan performanya dengan menjalankan skenario uji coba yang sudah ditentukan. Uji coba dilakukan untuk mengetahui hasil dari sistem ini sehingga dapat menjawab rumusan masalah pada tugas akhir ini.

#### 5.1 Lingkungan Uji Coba

Lingkungan pengujian menggunakan komponen-komponen yang terdiri dari: satu *server docker host*, satu *server login*, dan enam komputer pengujian. Semua komputer pengujian menggunakan enam buah desktop dengan sistem operasi Ubuntu 16.04 . Pengujian dilakukan di Laboratorium Arsitektur dan Jaringan Komputer Jurusan Teknik Informatika ITS.

Spesifikasi untuk setiap komponen yang digunakan ditunjukkan pada Tabel 5.1 untuk *docker host*, Tabel 5.2 untuk *server* halaman login, Tabel 5.3 untuk Komputer Pengujian 1, Tabel 5.4 untuk Komputer Pengujian 2, Tabel 5.5 untuk Komputer Pengujian 3, Tabel 5.6 untuk Komputer Pengujian 4, Tabel 5.7 untuk Komputer Pengujian 5, dan Tabel 5.8 untuk Komputer Pengujian 6.

## 1. *Server Untuk Docker Host*

**Tabel 5.1:** *Server Untuk Docker Host*

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz
	RAM 8GB
	Hard disk 500GB
<b>Perangkat Lunak</b>	Linux Mint 18.03 64 bit
	Docker-CE versi 1.13.1.
	MySQL versi 5.7.18.
	Python versi 2.7.12.
	Flask versi 1.0.2.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.134
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : X450LD

## 2. *Server Untuk Halaman Login*

**Tabel 5.2:** *Server Untuk Halaman Login*

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 1GB
	Hard disk 20GB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Python versi 2.7.12.
	Flask versi 1.0.2.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.171
	Netmask : 255.255.255.0
	Gateway : 10.151.36.1
	Hostname : SERVERLOGIN

### 3. Komputer Penguji

#### (a) Komputer Penguji 1

**Tabel 5.3:** Komputer Penguji 1

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM) i5-2120 CPU @ 3.30GHz
	RAM 8GB
	Hard disk 500GB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Firefox Quantum versi 60.0.1.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.33
	Netmask : 255.255.255.0
	Gateway : 10.151.36.134
	Hostname : DRONA

#### (b) Komputer Penguji 2

**Tabel 5.4:** Komputer Penguji 2

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
	RAM 6GB
	Hard disk 1TB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Firefox Quantum versi 60.0.1.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.34
	Netmask : 255.255.255.0
	Gateway : 10.151.36.134
	Hostname : BHISMA

**(c) Komputer Penguji 3****Tabel 5.5:** Komputer Penguji 3

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
	RAM 4GB
	Hard disk 250GB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Firefox Quantum versi 60.0.1.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.33
	Netmask : 255.255.255.0
	Gateway : 10.151.36.134
	Hostname : ARJUNA

**(d) Komputer Penguji 4****Tabel 5.6:** Komputer Penguji 4

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM) i3-2120 CPU @ 3.30GHz
	RAM 8GB
	Hard disk 1TB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Firefox Quantum versi 60.0.1.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.38
	Netmask : 255.255.255.0
	Gateway : 10.151.36.134
	Hostname : KRESNA

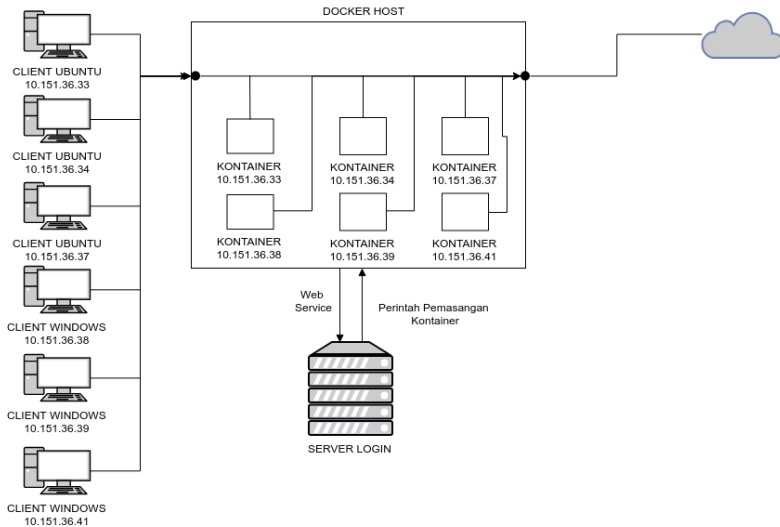
(e) **Komputer Penguji 5****Tabel 5.7:** Komputer Penguji 5

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 2GB
	Hard disk 120GB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Firefox Quantum versi 60.0.1.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.39
	Netmask : 255.255.255.0
	Gateway : 10.151.36.134
	Hostname : NARASOMA

(f) **Komputer Penguji 6****Tabel 5.8:** Komputer Penguji 6

<b>Perangkat Keras</b>	Processor Intel(R) Core(TM)2Duo CPU E7200 @ 2.53GHz
	RAM 2GB
	Hard disk 120GB
<b>Perangkat Lunak</b>	Ubuntu 16.04 64 bit
	Firefox Quantum versi 60.0.1.
<b>Konfigurasi Jaringan</b>	IP address : 10.151.36.41
	Netmask : 255.255.255.0
	Gateway : 10.151.36.134
	Hostname : ANGGADA

Untuk gambar arsitektur dari setiap komponen yang digunakan dapat dilihat pada Gambar 5.1.



**Gambar 5.1:** Arsitektur dari Setiap Komponen Uji Coba

## 5.2 Skenario Uji Coba

Uji coba akan dilakukan untuk mengetahui keberhasilan sistem yang telah dibangun. Skenario pengujian dibedakan menjadi 2 bagian, yaitu:

- **Uji Fungsionalitas**

Pengujian ini didasarkan pada fungsionalitas yang disajikan sistem.

- **Uji Performa**

Pengujian ini untuk menguji ketahanan sistem terhadap sejumlah permintaan ke aplikasi secara bersamaan. Pengujian dilakukan dengan melakukan *benchmark* pada sistem.

### 5.2.1 Skenario Uji Coba Fungsionalitas

Uji coba fungsionalitas dilakukan dengan cara menjalankan sistem yang telah dibuat, dan melakukan pengujian terhadap fitur yang telah dibuat. Uji coba fungsionalitas akan berfungsi untuk memastikan sistem sudah memenuhi kebutuhan yang tertera pada Bab 3, yaitu meliputi:

1. Pengujian *client* dapat mengakses internet.
2. Pengujian fungsionalitas menu aplikasi halaman administrator.
3. Pengujian schedule pada *docker host*.

#### 5.2.1.1 Uji *client* dapat Mengakses Internet

Pengujian ini dilakukan untuk mengetahui apakah *client* dapat mengakses internet atau tidak. Pada uji *client* dapat mengakses internet akan dibagi lagi menjadi beberapa bagian, antara lain:

1. Pengujian *client* dapat *login* ke dalam sistem.
2. Pengujian *client* dapat mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*.
3. Pengujian *docker host* dapat menerima permintaan penyediaan kontainer *docker*.

Pengujian menggunakan enam buah komputer penguji. Pengujian ini dapat dilakukan dengan membuka *browser* dan membuka *website* HTTP ataupun juga HTTPS. Daftar uji fungsionalitas *client* dapat mengakses internet dijelaskan pada Tabel 5.9.

**Tabel 5.9:** Skenario Uji *Client* dapat Mengakses Internet

No	Uji Coba	Hasil Harapan
1	<i>Client</i> membuka website HTTP maupun HTTPS dengan menggunakan <i>browser</i> .	<i>Client</i> dapat membuka website HTTP maupun HTTPS dengan menggunakan <i>browser</i> yang ada.

#### 5.2.1.1.1 Uji *Client* dapat *Login* ke Dalam Sistem

Pengujian ini dilakukan untuk mengetahui apakah *client* sudah bisa *login* ke dalam sistem saat *client* akan mengakses internet. Pengujian menggunakan satu buah *server* yang berperan sebagai *server login* untuk *client* dan menggunakan enam buah Komputer Penguji yang dijalankan dengan VirtualBox berperan sebagai *client*. Pengujian dilakukan oleh *client* dalam keadaan belum *login* ke dalam sistem. Ketika *client* mencoba membuka sebuah web, maka *client* akan diarahkan ke *server login* terlebih dahulu.

Alamat / IP address dari *server login* yang digunakan adalah 10.151.36.173. Setelah *client* diarahkan ke *server login*, selanjutnya adalah *client* harus memasukkan *input username* dan *password*. Daftar uji fungsionalitas *client* dapat *login* ke dalam sistem dijelaskan pada Tabel 5.10

**Tabel 5.10:** Skenario Uji *Client* dapat *Login* ke Dalam Sistem

No	Uji Coba	Hasil Harapan
1	<i>Client</i> membuka sebuah website ketika belum <i>login</i> ke dalam sistem.	<i>Traffic</i> dari <i>client</i> akan diarahkan ke <i>server login</i> dan <i>client</i> dapat membuka halaman <i>login</i> secara otomatis.



**Tabel 5.10:** Skenario Uji Mengelola Aplikasi Berbasis Docker

No	Uji Coba	Hasil Harapan
2	<i>Client</i> melakukan <i>login</i> ke <i>server login</i> .	<i>Client</i> berhasil melakukan <i>login</i> dengan menggunakan <i>username</i> dan <i>password</i> yang sudah ditentukan.

#### 5.2.1.1.2 Uji *Client* dapat Mengirimkan Permintaan Penyediaan Kontainer *Docker* ke *Docker Host*

Pengujian ini dilakukan untuk memberikan perintah kepada *docker host* untuk menyediakan kontainer *docker* ke *client* yang baru saja berhasil *login* ke dalam sistem. Pengujian menggunakan satu buah *server* yang berperan sebagai *server login* yang akan mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*.

Pengujian ini dapat dilakukan setelah *client* berhasil memasukkan *username* dan *password* ke sistem, dan berhasil *login* ke dalam sistem. Setelah itu sistem akan mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*. Daftar uji fungsionalitas *client* dapat mengirimkan permintaan penyediaan kontainer *docker* ke *docker host* dijelaskan pada Tabel 5.11

**Tabel 5.11:** Skenario Uji *Client* dapat *Login* Mengirimkan Permintaan Penyediaan Kontainer *Docker*

No	Uji Coba	Hasil Harapan
1	<i>Client</i> mengirimkan permintaan penyediaan kontainer <i>docker</i> kepada <i>docker host</i> .	<i>Client</i> berhasil mengirimkan permintaan penyediaan kontainer <i>docker</i> kepada <i>docker host</i> .

### 5.2.1.1.3 Uji *Docker Host* dapat Menerima Permintaan Penyediaan Kontainer *Docker*

Pengujian ini dilakukan untuk menerima perintah permintaan penyediaan kontainer *docker* pada *docker host*. Pengujian menggunakan satu buah *server* yang berperan sebagai *docker host* yang akan menerima permintaan penyediaan kontainer *docker*.

Alamat dari *server* yang berperan sebagai *docker host* adalah 10.151.36.134. Setelah berhasil menerima permintaan penyediaan kontainer *docker*, selanjutnya adalah sistem akan menuliskan *username*, *IP address*, dan *port* dari *client* yang telah mengirimkan permintaan penyediaan kontainer *docker* ke basis data yang sudah tersedia.

Setelah selesai menuliskan *username*, *IP address*, dan *port* dari *client* yang telah mengirimkan permintaan penyediaan kontainer *docker* ke basis data yang sudah tersedia, selanjutnya adalah membuat satu buah kontainer *docker* dengan nama kontainer [IP-Username-Port], dimana IP adalah *IP address* dari *client*, *Username* adalah *username* ketika *client* memasukkan *inputan* kepada sistem saat akan *login*, dan *Port* adalah sebuah *port* khusus untuk *client* tersebut.

Terakhir, pengujian yang dilakukan adalah membuat sebuah *directory* pada *docker host* yang berfungsi untuk menyimpan data *log traffic* dari *client*. *Directory* ini akan dibuat pada /container-data/[Tanggal]/[IP-USERNAME-PORT], dimana *Tanggal* akan sesuai dengan tanggal ketika *client* berhasil *login* ke dalam sistem, dan [IP-USERNAME-PORT] sesuai dengan yang sudah dituliskan pada basis data.

Daftar uji fungsionalitas *docker host* dapat menerima permintaan penyediaan kontainer *docker* dijelaskan pada Tabel 5.12.

**Tabel 5.12:** Skenario Uji *Docker Host* dapat Menerima Permintaan Penyediaan Kontainer *Docker*

No	Uji Coba	Hasil Harapan
1	Sistem menerima permintaan penyediaan kontainer <i>docker</i> dari <i>client</i> pada <i>docker host</i> .	Sistem berhasil menerima permintaan penyediaan kontainer <i>docker</i> dari <i>client</i> dan menuliskannya pada basis data pada <i>docker host</i> .
2	Sistem membuat satu buah kontainer <i>docker</i> untuk <i>client</i> pada <i>docker host</i> .	Sistem berhasil membuat satu buah kontainer <i>docker</i> dengan nama sesuai yang ditulis pada basis data untuk <i>client</i> yang berisi Mitmproxy pada <i>docker host</i> .
3	Sistem membuat sebuah <i>directory</i> pada <i>docker host</i> sesuai dengan tanggal dan informasi dari <i>client</i> .	Sistem berhasil membuat sebuah <i>directory</i> pada <i>docker host</i> sesuai dengan tanggal dan informasi dari <i>client</i> yang sudah dituliskan pada basis data.

### 5.2.1.2 Uji Fungsionalitas Menu Aplikasi Halaman *Administrator*

Aplikasi halaman *administrator* digunakan untuk membaca *log traffic* dari *client* yang sedang mengakses internet dan juga untuk melihat rekap *client* yang telah menggunakan internet pada hari-hari sebelumnya. Aplikasi halaman *administraotr* terdiri dari dua bagian utama, yaitu halaman *user list*, dan *history*.

Rancangan pengujian dan hasil yang diharapkan ditunjukkan dengan Tabel 5.13.

**Tabel 5.13:** Skenario Uji Fungsionalitas Aplikasi Halaman *Administrator*

No	Menu	Uji Coba	Hasil Harapan
1	User List	Menampilkan halaman <i>user list</i> pada web.	Halaman <i>user list</i> dapat tampil pada web.
		Melihat secara <i>live</i> atau langsung <i>log traffic</i> dari <i>client</i> .	<i>User</i> yang mempunyai akses membuka halaman <i>administrator</i> dapat melihat secara <i>live</i> atau langsung <i>log traffic</i> dari <i>client</i> .
		Melihat <i>log traffic</i> terakhir dari <i>client</i> .	<i>User</i> yang mempunyai akses membuka halaman <i>administrator</i> dapat melihat <i>log traffic</i> terakhir dari <i>client</i> .

**Tabel 5.13:** Skenario Uji Fungsionalitas Aplikasi Dasbor

No	Menu	Uji Coba	Hasil Harapan
		Menghentikan akses internet dari <i>client</i> .	<i>User</i> yang mempunyai akses membuka halaman <i>administrator</i> dapat menghentikan akses internet dari sebuah <i>client</i> .
2	History	Melihat <i>log traffic</i> dari <i>client</i> pada hari-hari sebelumnya.	<i>User</i> yang mempunyai akses membuka halaman <i>administrator</i> dapat melihat <i>log traffic</i> dari <i>client</i> pada hari-hari sebelumnya.

### 5.2.1.3 Uji Fungsionalitas *Schedule* pada *Docker Host*

Pengujian ini dilakukan untuk mengetahui apakah sistem dapat melakukan *reboot* untuk menghapus semua kontainer *docker* dan *rules* yang sedang aktif berjalan setiap hari pada pukul 23.59.

Rancangan pengujian dan hasil yang diharapkan ditunjukkan dengan Tabel 5.14

**Tabel 5.14:** Skenario Uji *Schedule* pada *Docker Host*

No	Uji Coba	Hasil Harapan
1	Sistem menjalankan sebuah <i>script</i> yang telah disediakan, dilakukan setiap hari pada pukul 23.59 secara otomatis.	Sistem dapat menjalankan <i>script</i> yang telah disediakan, dilakukan setiap hari pada pukul 23.59 secara otomatis. <i>Script</i> yang telah disediakan berfungsi untuk melakukan penghapusan kontainer <i>docker</i> dan <i>rules</i> yang sedang aktif.

## 5.2.2 Skenario Uji Coba Performa

Uji performa dilakukan dengan menggunakan enam buah desktop yang berperan sebagai *client* untuk melakukan akses ke internet secara bersama-sama. *Client* akan mencoba mengakses internet dengan membuka website HTTP maupun HTTPS.

Percobaan dilakukan dengan dua skenario, yaitu mengakses website HTTP dan mengakses website HTTPS

### 5.2.2.1 Uji Performa Penggunaan *Memory*

Pengujian dilakukan dengan menghitung penggunaan *memory* yang terjadi pada *docker host*. Penggunaan *memory* di sini adalah penggunaan dari kontainer aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan *memory* dari masing-masing kontainer selama proses pengujian dilakukan.

#### **5.2.2.2 Uji Performa Penggunaan CPU**

Pengujian dilakukan dengan menghitung penggunaan CPU yang terjadi pada *docker host*. Penggunaan CPU di sini adalah penggunaan dari kontainer aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan CPU dari masing-masing kontainer selama proses pengujian dilakukan.

#### **5.2.2.3 Uji Performa Kecepatan Menangani Request**

Pengujian dilakukan dengan mengukur kecepatan unduh dan juga upload dari *client*. Kecepatan unduh dan upload yang diukur dengan menggunakan *tools speedtest* yang tersedia pada masing-masing *client*.

#### **5.2.2.4 Uji Performa Keberhasilan Request**

Pengujian dilakukan dengan menghitung jumlah *request* atau jumlah permintaan penyediaan kontainer *docker* yang gagal dilakukan selama skenario dijalankan. Dari semua jumlah *request* yang dikirimkan selama pengujian, akan didapatkan persen *request* yang gagal dilakukan.

### **5.3 Hasil Uji Coba dan Evaluasi**

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

#### **5.3.1 Uji Fungsionalitas**

Berikut dijelaskan hasil pengujian fungsionalitas pada sistem yang dibangun.

### 5.3.1.1 Uji *Client* dapat Mengakses Internet

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1 dan pada Tabel 5.9. Pada hasil uji *client* dapat mengakses internet akan dibagi lagi menjadi beberapa bagian, antara lain

1. Pengujian *client* dapat *login* ke dalam sistem.
2. Pengujian *client* dapat mengirimkan permintaan penyediaan kontainer *docker* ke *docker host*.
3. Pengujian *docker host* dapat menerima permintaan penyediaan kontainer *docker*.

Setelah melalui hasil uji yang telah disebutkan diatas, hasil pengujian *client* dapat mengakses internet seperti tertera pada Tabel 5.15

**Tabel 5.15:** Hasil Uji Coba *Client* dapat Mengakses Internet

No	Uji Coba	Hasil
1	<i>Client</i> membuka sebuah website HTTP maupun HTTPS dengan menggunakan <i>browser</i> .	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.9, hasil uji coba menunjukkan semua skenario berhasil ditangani.

#### 5.3.1.1.1 Uji *Client* dapat *Login* ke Dalam Sistem

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1.1 dan pada Tabel 5.10. Hasil pengujian seperti tertera pada Tabel 5.16.



**Tabel 5.16:** Hasil Uji Coba *Client* dapat *Login* ke Dalam Sistem

No	Uji Coba	Hasil
1	<i>Client</i> membuka sebuah website ketika belum <i>login</i> ke dalam sistem.	OK.
2	<i>Client</i> melakukan <i>login</i> ke <i>server</i> <i>login</i> .	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.10, hasil uji coba menunjukkan semua skenario berhasil ditangani.

#### **5.3.1.1.2 Uji *Client* dapat Mengirimkan Permintaan Penyediaan Kontainer *Docker* ke *Docker Host***

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1.2 dan pada Tabel 5.11. Hasil pengujian seperti tertera pada Tabel 5.17.

**Tabel 5.17:** Hasil Uji Coba *Client* dapat Mengirimkan Permintaan Penyediaan Kontainer *Docker* ke *Docker Host*

No	Uji Coba	Hasil
1	<i>Client</i> mengirimkan permintaan penyediaan kontainer <i>docker</i> kepada <i>docker host</i> .	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.11, hasil uji coba menunjukkan semua skenario berhasil ditangani.

### 5.3.1.1.3 Uji *Docker Host* dapat Menerima Permintaan Penyediaan Kontainer *Docker*

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.1.3 dan pada Tabel 5.12. Hasil pengujian seperti tertera pada Tabel 5.18.

**Tabel 5.18:** Hasil Uji Coba *Docker Host* dapat Menerima Permintaan Penyediaan Kontainer *Docker*

No	Uji Coba	Hasil
1	Sistem menerima permintaan penyediaan kontainer <i>docker</i> dari <i>client</i> pada <i>docker host</i> .	OK.
2	Sistem membuat satu buah kontainer <i>docker</i> untuk <i>client</i> pada <i>docker host</i> .	OK.
3	Sistem membuat sebuah <i>directory</i> pada <i>docker host</i> sesuai dengan tanggal dan informasi dari <i>client</i> .	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.12, hasil uji coba menunjukkan semua skenario berhasil ditangani.

### 5.3.1.2 Uji Fungsionalitas Menu Aplikasi Halaman *Administrator*

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.2 dan pada Tabel 5.13. Hasil pengujian seperti tertera pada Tabel 5.19

**Tabel 5.19:** Hasil Uji Fungsionalitas Aplikasi Halaman *Administrator*

No	Menu	Uji Coba	Hasil
1	User List	Menampilkan halaman <i>user list</i> pada web.	OK.
		Melihat secara <i>live</i> atau langsung <i>log traffic</i> dari <i>client</i> .	OK.
		Melihat <i>log traffic</i> terakhir dari <i>client</i> .	OK.
		Menghentikan akses internet dari <i>client</i> .	OK.
2	History	Melihat <i>log traffic</i> dari <i>client</i> pada hari-hari sebelumnya.	OK.

Sesuai dengan skenario uji coba yang diberikan pada Tabel 5.13, hasil uji coba menunjukkan semua skenario berhasil ditangani.

### 5.3.1.3 Uji Fungsionalitas *Schedule* pada *Docker Host*

Pengujian dilakukan sesuai dengan skenario yang dijelaskan pada subbab 5.2.1.3 dan pada Tabel 5.14. Hasil pengujian seperti tertera pada Tabel 5.20

**Tabel 5.20:** Skenario Uji *Schedule* pada *Docker Host*

No	Uji Coba	Hasil Harapan
1	Sistem menjalankan sebuah <i>script</i> yang telah disediakan, dilakukan setiap hari pada pukul 23.59 secara otomatis.	OK.

### 5.3.2 Hasil Uji Performa

Seperti yang sudah dijelaskan pada subbab 5.2 pengujian performa dilakukan dengan menggunakan enam buah *desktop* yang berperan sebagai *client* untuk melakukan akses ke internet secara bergantian. *Client* akan mencoba mengakses internet dengan membuka *website* HTTP maupun HTTPS.

#### 5.3.2.1 Penggunaan *Memory*

Pengujian dilakukan dengan menghitung penggunaan *memory* yang terjadi pada *docker host*. Penggunaan *memory* di sini adalah penggunaan dari kontainer aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan *memory* dari masing-masing kontainer selama proses pengujian dilakukan.

#### 5.3.2.2 Penggunaan CPU

Pengujian dilakukan dengan menghitung penggunaan CPU yang terjadi pada *docker host*. Penggunaan CPU di sini adalah penggunaan dari kontainer aplikasi yang sedang berjalan. Perhitungan dilakukan dengan mengambil nilai rata-rata penggunaan CPU dari masing-masing kontainer selama proses pengujian dilakukan.

### 5.3.2.3 Kecepatan Menangani *Request*

Pengujian dilakukan dengan mengukur kecepatan unduh dan juga upload dari *client*. Kecepatan unduh dan upload yang diukur dengan menggunakan *tools speedtest* yang tersedia pada masing-masing *client*.

### 5.3.2.4 Keberhasilan *Request*

Pengujian dilakukan dengan menghitung jumlah *request* atau jumlah permintaan penyediaan kontainer *docker* yang gagal dilakukan selama skenario dijalankan. Dari semua jumlah *request* yang dikirimkan selama pengujian, akan didapatkan persen *request* yang gagal dilakukan.

## 5.4 Hasil Uji Coba dan Evaluasi

Berikut dijelaskan hasil uji coba dan evaluasi berdasarkan skenario yang telah dijelaskan pada subbab 5.2.

### 5.4.1 Hasil Uji Performa

Seperti yang sudah dijelaskan pada subbab 5.2 pengujian performa dilakukan dengan menggunakan enam buah desktop yang berperan sebagai *client* untuk melakukan akses ke internet secara bergantian. *Client* akan mencoba mengakses internet dengan membuka website HTTP maupun HTTPS.

#### 5.4.1.1 Kecepatan Menangani *Request*

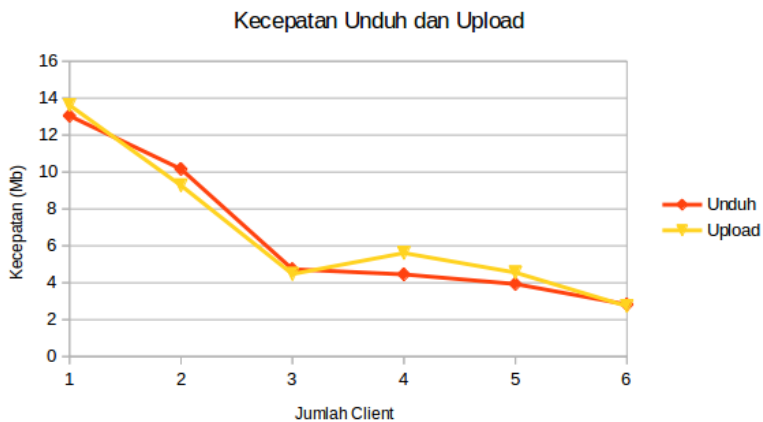
Dari hasil uji coba kecepatan menangani *request*, yaitu kecepatan unduh dan *upload* dari *client*. Uji coba dilakukan dengan melakukan cek kecepatan pada website <http://www.speedtest.net/id>. Uji coba dilakukan dengan menggunakan satu komputer penguji sampai dengan enam

komputer penguji secara bersamaan. Hasil dari uji coba menangani *request* dapat dilihat pada Tabel 5.21.

**Tabel 5.21:** Kecepatan Menangani *Request* Unduh dan *Upload* Menggunakan *Internet Access Management* Berbasis Kontainer

Jumlah Client	Kecepatan unduh	Kecepatan <i>Upload</i>
1	± 13,04 Mbps	± 13,62 Mbps
2	± 10,16 Mbps	± 9,27 Mbps
3	± 4,73 Mbps	± 4,47 Mbps
4	± 4,45 Mbps	± 5,61 Mbps
5	± 3,93 Mbps	± 4,55 Mbps
6	± 2,82 Mbps	± 2,73 Mbps

Hasil uji coba kecepatan menangani *request* dengan menggunakan *Internet Access Management* berbasis kontainer ditunjukkan dalam grafik pada Gambar 5.2.



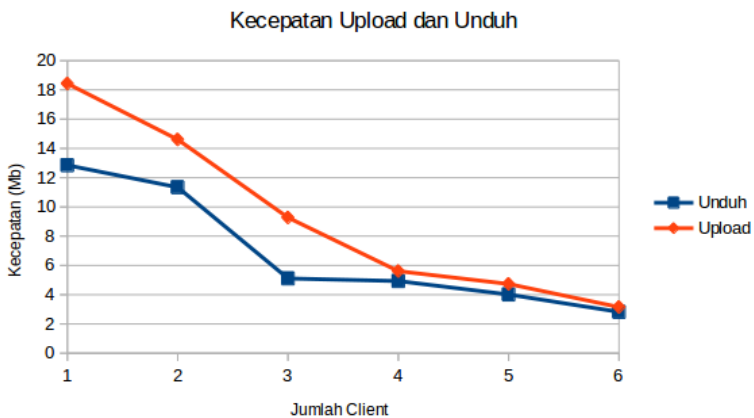
**Gambar 5.2:** Grafik Kecepatan Menangani *Request*

Lalu untuk uji coba dilakukan dengan enam komputer penguji dan dengan menggunakan *internet access management* konvensional dapat dilihat pada Tabel 5.22

**Tabel 5.22:** Kecepatan Menangani *Request* Unduh dan *Upload* Menggunakan *Internet Access Management* Konvensional

Jumlah Client	Kecepatan unduh	Kecepatan <i>Upload</i>
1	$\pm 12,85$ Mbps	$\pm 18,43$ Mbps
2	$\pm 11,34$ Mbps	$\pm 14,61$ Mbps
3	$\pm 5,11$ Mbps	$\pm 9,27$ Mbps
4	$\pm 4,93$ Mbps	$\pm 5,61$ Mbps
5	$\pm 4,01$ Mbps	$\pm 4,73$ Mbps
6	$\pm 2,82$ Mbps	$\pm 3,15$ Mbps

Hasil uji coba kecepatan menangani *request* dengan menggunakan *Internet Access Management* konvensional ditunjukkan dalam grafik pada Gambar 5.3.



**Gambar 5.3:** Grafik Kecepatan Menangani *Request*

Dari Tabel 5.21 dan Tabel 5.22 dapat dilihat bahwa semakin banyak jumlah *client*, maka semakin berkurang kecepatan unduh maupun *upload* dari masing-masing *client*. Hal tersebut dikarenakan *bandwith* dari *server* yang digunakan sebagai *docker host* juga terbatas dan menggunakan Kabel UTP Cat 5 100Mb/s. Jika semakin besar *bandwith* yang dapat diterima oleh *server* yang akan digunakan sebagai *docker host*, maka semakin cepat pula kecepatan unduh maupun *upload* dari masing-masing *client*. Hal tersebut sangat mempengaruhi kecepatan saat uji coba berlangsung.

#### 5.4.1.2 Penggunaan CPU

Dari hasil uji coba penggunaan CPU dilakukan ketika *client* tidak melakukan aktivitas sama sekali (*idle*), sampai ketika *client* melakukan aktivitas dengan mencoba membuka website HTTP maupun HTTPS. Perhitungan penggunaan CPU adalah jumlah penggunaan CPU dari masing-masing kontainer *docker* dari *client*, dibagi dengan jumlah *cores* yang tersedia pada *docker host*. Uji coba dilakukan dengan menggunakan salah satu komputer penguji untuk mencoba membuka website HTTP maupun HTTPS. Dari hasil uji coba ini, dapat dilihat pada Tabel 5.23



**Tabel 5.23:** *Penggunaan CPU*

<b>Waktu Detik</b>	<b>Jumlah CPU yang Digunakan</b>
15	0.30%
30	0.59%
45	0.73%
60	0.9%
75	0.95%
90	1.54%
105	2.65%
120	3.80%
135	5%

Dari Tabel 5.23 dapat dilihat bahwa ketika *client* sedang tidak melakukan aktivitas (*idle*), maka penggunaan CPU akan kecil. Sedangkan ketika *client* melakukan aktivitas dengan mencoba membuka website HTTP maupun HTTPS, maka penggunaan CPU akan mengalami peningkatan.

Hasil uji coba penggunaan CPU ditunjukkan dalam grafik pada Gambar 5.4.

**Gambar 5.4:** Grafik Penggunaan *Memory*

### 5.4.1.3 Penggunaan *Memory*

Dari hasil uji coba penggunaan *memory*, semakin banyak *request* yang diterima, semakin banyak *memory* yang diperlukan. Perhitungan penggunaan *memory* adalah jumlah penggunaan dari masing-masing kontainer *docker* dari *client*. Dari hasil uji coba ini, dapat dilihat pada Tabel 5.24.

**Tabel 5.24:** *Penggunaan Memory*

<b>Jumlah <i>Client</i></b>	<b>Jumlah <i>Memory</i> yang Digunakan</b>
1	145 MB
2	213 MB
3	254 MB
4	421 MB
5	571 MB
6	786 MB

Dari Tabel 5.24 dapat dilihat bahwa semakin banyak jumlah kontainer *docker* yang sedang berjalan, maka semakin meningkat pula penggunaan *memory* dari *docker host*. Rata-rata penggunaan *memory* dari kontainer *docker* yang sudah dibuat sebesar  $\pm 131$  MB. Jika semakin besar *memory* yang tersedia pada *docker host*, maka semakin banyak pula kontainer *docker* yang dapat dibuat.

Selain itu dapat pula melakukan pembatasan sumber daya pada kontainer *docker* yang sudah dibuat jika hanya memiliki *memory* yang terbatas pada *docker host*. Pembatasan sumber daya dapat dilakukan ketika kontainer *docker* akan dibuat dengan menambahkan parameter `-m` atau `--memory=`.

Hasil uji coba penggunaan *memroy* ditunjukkan dalam grafik pada Gambar 5.5.



**Gambar 5.5:** Grafik Penggunaan *Memory*

#### 5.4.1.4 Keberhasilan *Request*

Pada uji coba ini, dilakukan perhitungan seberapa besar jumlah *request* yang berhasil dilakukan. Untuk jumlah *client*, dapat dilihat pada Tabel 5.25.

**Tabel 5.25:** *Success Ratio Request*

<i>Client</i>	Persen Sukses
<i>Client 1</i>	100%
<i>Client 2</i>	100%
<i>Client 3</i>	100%
<i>Client 4</i>	100%
<i>Client 5</i>	100%
<i>Client 6</i>	100%

Dari Tabel 5.25 dapat dilihat bahwa semua *request* dari *client* sukses dijalankan dan tidak ada *error* sama sekali. Hasil uji coba tersebut berdasarkan keberhasilan *client* untuk mengakses

internet, mulai dari *client* login ke dalam sistem sampai dengan sistem menyediakan sebuah kontainer *docker* untuk *client tersebut* dan berdasarkan keberhasilan *client* untuk mengakses website HTTP maupun HTTPS.

## 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 perancangan, implementasi dan pengujian terhadap sistem, dapat diambil beberapa kesimpulan berikut:

1. Sistem dapat mengarahkan *client* ke halaman *login* dari sistem supaya *client* dapat melakukan *autentifikasi*.
2. Sistem dapat membuat kontainer *docker* yang berisi *mitmproxy* secara otomatis ketika terdapat *client* yang berhasil *login* ke dalam sistem.
3. Sistem dapat mengarahkan *traffic* dari *client* ke kontainer *docker* yang sudah dibuat dan digunakan sebagai *internet access management* bagi *client* dan memperbolehkan atau mengijinkan *client* tersebut untuk mengakses internet.
4. Sistem dapat mencatat semua aktivitas dari *client* ketika *client* mengakses website HTTP maupun HTTPS.
5. Didapatkan data penggunaan sumber daya pada *server* yang digunakan sebagai *docker host*, dan didapatkan data perbandingan performa kecepatan antara *Internet Access Management* berbasis kontainer dengan *Internet Access Management* konvensional.

#### 6.2 Saran

Berikut beberapa saran yang diberikan untuk pengembangan lebih lanjut:

1. Sistem dapat dikembangkan dengan menggunakan *server* lebih dari satu untuk meringankan beban kerja dari *server* itu sendiri.
2. Sistem dapat dikembangkan dengan menentukan beban dari setiap *server*, dengan menambahkan kriteria-kriteria yang sesuai dengan lingkungan sistem yang ada, seperti jarak antara *docker host* dengan *middleware* atau kecepatan bandwidth dari setiap *docker host* merupakan kriteria yang baik.

## DAFTAR PUSTAKA

- [1] “Welcome to Python.org,” 29 Mei 2018. [Daring]. Tersedia pada: <https://www.python.org/>. [Diakses: 29 Mei 2018].
- [2] “Welcome | Flask (A Python Microframework),” 29 Mei 2018. [Daring]. Tersedia pada: <http://flask.pocoo.org/>. [Diakses: 29 Mei 2018].
- [3] D. I. Fernandez., J. M. Atencia., O. Q. Gamboa., dan O. E. H. Bedoya., “Design and Implementation of an ”Web API” for the Automatic Translation Colombia’s Language Pairs: Spanish-Wayuunaiki Case,” *IEEE Transactions on Cloud Computing*, Jul. 2013.
- [4] “Supervisor: A Process Control System,” 29 Mei 2018. [Daring]. Tersedia pada: <https://http://supervisord.org/>. [Diakses: 29 Mei 2018].
- [5] X. Chi, B. Liu, Q. Niu, dan Q. Wu, “Web Load Balance and Cache Optimization Design Based Nginx under High-Concurrency Environment,” in *2012 Third International Conference on Digital Manufacturing Automation*, Jul. 2012, hal. 1029–1032.
- [6] L. fei Xuan. dan P. fei Wu., “The Optimization and Implementation of Iptables Rules Set on Linux,” *IEEE Transactions on Cloud Computing*, Jun. 2015.
- [7] Y. Ping, H. Hong-Wei, dan Z. Nan, “Design and implementation of a MySQL database backup and recovery system,” in *Proceeding of the 11th World Congress on Intelligent Control and Automation*, Jun. 2014, hal. 5410–5415.
- [8] “Welcome to Mitmproxy.org,” 29 Mei 2018. [Daring]. Tersedia pada: <https://mitmproxy.org/>. [Diakses: 29 Mei 2018].

- [9] “What is Docker?” 2016, 29 Mei 2018. [Daring]. Tersedia pada: <https://www.docker.com/what-docker>. [Diakses: 29 Mei 2018].
- [10] C. Boettiger, “An introduction to Docker for reproducible research, with examples from the R environment,” *ACM SIGOPS Operating Systems Review*, vol. 49, no. 1, hal. 71–79, Jan. 2015, arXiv: 1410.0846.
- [11] “Docker Registry,” Jun. 2017, 29 Mei 2018. [Daring]. Tersedia pada: <https://docs.docker.com/registry/>. [Diakses: 29 Mei 2018].



## LAMPIRAN A

### INSTALASI PERANGKAT LUNAK

#### Instalasi Pustaka Python

Dalam pengembangan sistem ini, digunakan berbagai pustaka pendukung. Pustaka pendukung yang digunakan merupakan pustaka untuk bahasa pemrograman Python. Berikut adalah daftar pustaka yang digunakan dan cara pemasangannya:

- Python Pip  
`$ sudo apt-get install python-pip`
- Python Dev  
`$ sudo apt-get install python-dev`
- Setuptools  
`$ sudo apt-get install python-setuptools`
- MySQLd  
`$ sudo apt-get install python-mysqldb`

#### Pemasangan kerangka kerja Flask

Dalam pengembangan sistem ini, digunakan Flask karena Flask merupakan kerangka kerja yang menggunakan bahasa pemrograman Python. Untuk memasang Flask, jalankan perintah `sudo pip3 install flask`.

#### Pemasangan perangkat lunak Unicorn

Dalam pengembangan sistem ini, digunakan Unicorn untuk menangani servis dari halaman *login*. Untuk memasang Unicorn, jalankan perintah `sudo pip3 install unicorn`.

#### Pemasangan perangkat lunak Supervisor

Dalam pengembangan sistem ini, digunakan Supervisor supaya servis pada halaman *login* dapat langsung berjalan ketika

*server* dinyalakan. Untuk memasang Supervisor, jalankan perintah `sudo apt-get install supervisor`.

## Pemasangan perangkat lunak Nginx

Dalam pengembangan sistem ini, digunakan Nginx sebagai *web server* untuk halaman *login*. Untuk memasang Nginx, jalankan perintah `sudo apt-get install nginx`.

## Instalasi Lingkungan Docker

Proses pemasangan Docker dapat dilakukan sesuai tahap berikut:

- Menambahkan repository Docker

Langkah ini dilakukan untuk menambahkan *repository* Docker ke dalam paket `apt` agar dapat di unduh oleh Ubuntu. Untuk melakukannya, jalankan perintah 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
```

- Mengunduh Docker  
Docker dikembangkan dalam dua versi, yaitu CE (*Community Edition*) dan EE (*Enterprise Edition*). Dalam pengembangan sistem ini, digunakan Docker CE karena merupakan versi Docker yang gratis. Untuk mengunduh Docker CE, jalankan perintah `sudo apt-get -y install docker-ce`.
- Mencoba menjalankan Docker  
Untuk melakukan tes apakah Docker sudah terpasang dengan benar, gunakan perintah `sudo docker run hello-world`.

*(Halaman ini sengaja dikosongkan)*

## LAMPIRAN B

### KONFIGURASI

#### Konfigurasi Supervisor

Supaya servis pada halaman *login* dapat langsung berjalan, penulis menambahkan konfigurasi perangkat lunak Supervisor pada `/etc/supervisor/conf.d/app.conf` seperti pada Kode Sumber 2.1.

```
command = /home/fourirakbar/flask-loginpage
         /flask-loginpageenv/bin/python /home/
         fourirakbar/flask-loginpage/flask-
         loginpageenv/bin/gunicorn -b
         0.0.0.0:4000 app:app

directory = /home/fourirakbar/flask-
            loginpage

user = fourirakbar

stdout_logfile = /home/fourirakbar/flask-
                 loginpage/logs/app_stdout.log

stderr_logfile = /home/fourirakbar/flask-
                 loginpage/logs/app_stderr.log

redirect_stderr = True
autostart = True
enviroment = PATH = "/home/fourirakbar /
               flask-loginpage/flask-loginpageenv/bin",
PRODUCTION=1
```

**Kode Sumber 2.1:** Isi Berkas app.conf

Perlu diperhatikan ketika menambahkan atau mengubah

konfigurasi Supervisor pada `/etc/supervisor/conf.d/` yang telah di *server* untuk halaman *login*, perlu dilakukan *reload Supervisor* dengan menjalankan *command* pada terminal seperti pada Kode Sumber 2.2.

```
sudo supervisorctl reread
sudo supervisorctl reload
sudo supervisorctl status
```

**Kode Sumber 2.2:** Command untuk Reload Supervisor

Perlu diperhatikan pula ketika menambahkan atau mengubah konfigurasi Nginx pada `/etc/nginx/sites-available/` yang telah di *server* untuk halaman *login*, perlu dilakukan aktivasi konfigurasi Nginx dengan menjalankan *command* pada terminal seperti pada Kode Sumber 2.3.

```
sudo ln -s /etc/nginx/sites-available/app
/etc/nginx/sites-enabled/app
```

**Kode Sumber 2.3:** Command untuk mengaktifkan konfigurasi Nginx

Setelah itu, jalankan Kode Sumber 2.4 supaya konfigurasi yang baru saja diaktifkan dapat digunakan.

```
sudo service nginx restart
```

**Kode Sumber 2.4:** Command untuk merestart Nginx

## Konfigurasi Nginx

Supaya web *server* untuk halaman *login* dapat dijalankan, penulis menambahkan konfigurasi perangkat lunak Nginx pada `/etc/nginx/sites-available/app` seperti pada Kode Sumber 2.5.

```
server {
    listen 80;
```

```

server_name 10.151.36.173;

add_header X-Frame-Options SAMEORIGIN;
add_header X-Content-Type-Options nosniff
;
add_header X-XSS-Protection "1; mode=
    block";

access_log /home/fourirakbar/flask-
    loginpage/logs/app_access.log;
location / {
    proxy_pass http://127.0.0.1:4000;
    proxy_set_header Host $host;
    proxy_set_header X-Real-IP
        $remote_addr;
    proxy_set_header X-Forwarded-For
        $proxy_add_x_forwarded_for;
}
location ^~ /static/ {
    include /etc/nginx/mime.types;
    alias /home/fourirakbar/flask-
        loginpage/static/;
}
}

```

**Kode Sumber 2.5:** Isi Berkas app

*(Halaman ini sengaja dikosongkan)*



## BIODATA PENULIS



**Fourir Akbar**, akrab dipanggil Oing, lahir pada tanggal 25 April 1996 di Surabaya. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Departemen Informatika Institut Teknologi Sepuluh Nopember. Memiliki beberapa hobi antara lain futsal dan DOTA. Pernah menjadi asisten dosen pada mata kuliah sistem operasi dan mata kuliah jaringan komputer pada semester 2016/2017 dan 2017/2018. Lalu juga pernah menjadi asisten dosen pada mata kuliah sistem terdistribusi pada tahun ajaran 2017/2018. Penulis juga pernah menjadi asisten dosen pendidikan informatika dan komputer terapan (PIKTI) ITS pada tahun ajaran 2016/2017 dan 2017/2018. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, antara lain sebagai Staff Departemen Hubungan Luar Himpunan Mahasiswa Teknik Computer-Informatika pada tahun ajaran 2015/2016. Penulis juga aktif dalam kepanitiaan Schematics, antara lain sebagai Staff Biro Revolutionary Entertainment and Expo with Various Arts pada tahun ajaran 2015/2016 dan menjadi Badan Pengurus Harian (BPH) Biro Perlengkapan dan Transportasi pada tahun 2016/2017. Penulis juga merupakan salah satu administrator aktif pada Laboratorium Arsitektur dan jaringan Komputer di Departemen Informatika ITS.