

LAPORAN PROYEK AKHIR

**PENGEMBANGAN APLIKASI MANAJEMEN OPENFLOW TABLE BERBASIS
ANDROID PADA JARINGAN *SOFTWARE DEFINED NETWORK*
MENGUNAKAN *OPENDAYLIGHT CONTROLLER***

***OPENFLOW TABLE MANAGEMENT ANDROID APPLICATION DEVELOPMENT
ON SOFTWARE DEFINED NETWORK USING OPENDAYLIGHT CONTROLLER***



Diajukan oleh :

SWAKRESNA EDITYOMURTI

14/370252/SV/07759

**PROGRAM SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET
DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA
SEKOLAH VOKASI
UNIVERSITAS GADJAH MADA
YOGYAKARTA**

2018

**PENGEMBANGAN APLIKASI MANAJEMEN OPENFLOW TABLE BERBASIS
ANDROID PADA JARINGAN *SOFTWARE DEFINED NETWORK*
MENGUNAKAN *OPENDAYLIGHT CONTROLLER***

**Proyek Akhir
Program Studi Teknologi Rekayasa Internet**

**Diajukan kepada
Departemen Teknik Elektro dan Informatika
Fakultas Sekolah Vokasi Universitas Gadjah Mada
Sebagai syarat kelengkapan studi jenjang Sarjana Terapan (D-IV)
Dalam memperoleh derajat Sarjana Terapan Teknik
Program Studi Teknologi Rekayasa Internet**

**Oleh :
SWAKRESNA EDITYOMURTI**

14/370252/SV/07759

**PROGRAM SARJANA TERAPAN TEKNOLOGI REKAYASA INTERNET
DEPARTEMEN TEKNIK ELEKTRO DAN INFORMATIKA
SEKOLAH VOKASI
UNIVERSITAS GADJAH MADA
YOGYAKARTA
2018**

LEMBAR PENGESAHAN


Judul : PENGEMBANGAN APLIKASI MANAJEMEN OPENFLOW
TABLE BERBASIS ANDROID PADA JARINGAN *SOFTWARE
DEFINED NETWORK* MENGGUNAKAN *OPENDAYLIGHT
CONTROLLER*
Nama : Swakresna Edityomurti
Program Studi : Teknologi Rekayasa Internet
Dosen Pembimbing : Nur Rohman Rosyid, S.T., M.T., D.Eng.
Waktu Ujian : Selasa, 24 Juli 2018, Pukul 07.30 – 08.30, Ruang 230

Telah dipertanggung jawabkan dan diuji oleh tim penguji serta disetujui dan disahkan
sebagai syarat kelengkapan studi jenjang Sarjana Terapan (S.Tr)
Program Studi Teknologi Rekayasa Internet
Departemen Teknik Elektro dan Informatika
Sekolah Vokasi Universitas Gadjah Mada

Yogyakarta, 24 Juli 2018

Diterima dan disetujui oleh,

Ketua Penguji


Alif Subardono, S.T., M.Eng.
NIP. 197402102002121001

Sekretaris Penguji



Ir. Sri Lestari, M.T.
NIP. 195908281986022001

Dosen Pembimbing



Nur Rohman Rosyid, S.T., M.T., D.Eng.
NIP. 1120120075

Mengetahui,

Ketua Departemen Teknik Elektro dan
Informatika


Nur Rohman Rosyid, S.T., M.T., D.Eng.
NIP. 1120120075

Ketua Program Studi
DIV Teknologi Rekayasa Internet


Muhammad Arrofiq, S.T., M.T., P.hD.
NIP. 197311271999031001

PERNYATAAN KEASLIAN PENELITIAN

Dengan ini, saya SWAKRESNA EDITYOMURTI menyatakan bahwa proyek akhir ini adalah karya saya sendiri dan belum pernah diajukan sebagai pemenuhan persyaratan untuk memperoleh gelar kesarjanaan Diploma-Empat (D-IV) dari Universitas Gadjah Mada maupun perguruan tinggi lainnya.

Semua informasi yang dimuat dalam proyek akhir ini yang berasal dari karya orang lain, baik yang dipublikasikan maupun tidak, telah diberikan penghargaan dengan mengutip nama sumber penulis secara benar dan semua isi dari proyek akhir ini sepenuhnya menjadi tanggung jawab penulis.

Yogyakarta, 19 Juli 2018



Swakresna Edityomurti
14/370252/SV/07759

KATA PENGANTAR

Puji dan rasa syukur penulis panjatkan atas kehadiran Allah SWT, karena berkat limpahan rahmat, hidayat, dan inayah-Nya Proyek Akhir ini dapat diselesaikan dengan baik. Salam dan sholawat semoga senantiasa selalu tercurahkan pada baginda Rasulullah Muhammad S A W.

Proyek Akhir yang berjudul "*Pengembangan Aplikasi Manajemen OpenFlow Table berbasis Android pada Jaringan Software Defined Network menggunakan OpenDaylight Controller*" ini disusun sebagai syarat untuk mendapatkan gelar Sarjana Terapan (D-IV) Program Studi Teknologi Rekayasa Internet Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada.

Penulis mengucapkan rasa terimakasih yang sebesar-besarnya atas semua bantuan yang sudah diberikan kepada penulis, baik secara langsung maupun tidak langsung selama proses pembuatan dan penyusunan laporan proyek akhir ini hingga selesai. Terlebih secara khusus penulis mengucapkan rasa terimakasih kepada :

1. Bapak Nur Rohman Rosyid, S.T., M.T., D.Eng. selaku dosen pembimbing proyek akhir, karena sudah berkenan membimbing dan memberikan arahan serta dukungan selama penulis menyelesaikan proyek akhir.
2. Bapak Muhammad Arrofiq, S.T., M.T., Ph.D., selaku ketua Program Studi Diploma Teknologi Rekayasa Internet.
3. Ibu Ir. Sri Lestari, M.T. selaku dosen wali yang telah memberikan bimbingan selama penulis mengikuti masa studi di Program Studi D4 Teknologi Rekayasa Internet, Departemen Teknik Elektro dan Informatika, Sekolah Vokasi, Universitas Gadjah Mada.
4. Bapak, Ibu, serta Saudara yang selalu memberikan doa serta dukungan secara penuh kepada penulis dalam proses masa studi selama 4 tahun ini.
5. Seluruh dosen serta staff dan karyawan Program Studi D4 Teknologi Rekayasa Internet, Sekolah Vokasi, Universitas Gadjah Mada, atas ilmu, bimbingan dan bantuannya hingga penulis selesai menyusun proyek akhir ini.
6. Seluruh teman-teman D4 Teknologi Rekayasa Internet, terkhusus angkatan 2014 yang selalu kebersamai dalam setiap kesempatan dari awal hingga masa perkuliahan berakhir.

7. Hanif Sugiyanto, Ali Mustofa, Mas Sabrang, Mas Semilir Bayu, Mas Helmy Yunan, Mas Penya Adinugraha, Mas Allan, Mas Iden, Mas Niam, Mas Angga, Mas Isa, Mas Ali, serta seluruh teman-teman di CV. Extra Integer Technology atas dukungan dan bantuannya memberikan izin selama menyusun proyek akhir ini.
8. Berbagai pihak yang tidak bisa disebutkan satu persatu yang telah membantu menyelesaikan proyek akhir ini.

Penulis menyadari bahwa proyek akhir ini belum sempurna, baik dari segi materi maupun penyajiannya. Oleh karena itu penulis mengharapkan masukan berupa kritik dan saran yang membangun untuk memperbaiki setiap kekurangan yang ada sehingga kedepannya akan lebih baik lagi.

Terakhir penulis berharap, semoga proyek akhir ini dapat memberikan hal yang bermanfaat dan menambah wawasan bagi pembaca dan khususnya bagi penulis juga.

Yogyakarta. 19 Juli 2018



Penulis

DAFTAR ISI

HALAMAN SAMPUL	i
LEMBAR PENGESAHAN.....	iii
PERNYATAAN KEASLIAN PENELITIAN	iv
KATA PENGANTAR	v
DAFTAR ISI	vii
DAFTAR GAMBAR	ix
DAFTAR TABEL	xi
INTISARI.....	xii
ABSTRACT	xiii
BAB I PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah.....	3
1.3 Batasan Masalah	4
1.4 Tujuan Penelitian	4
1.5 Manfaat Penelitian	4
1.6 Sistematika Penulisan	5
BAB II TINJAUAN PUSTAKA	6
2.1 Konsep Jaringan Komputer	10
2.2 Software Defined Network	11
2.3 Flow	13
2.4 OpenFlow.....	13
2.5 OpenFlow Switch	14
2.6 OpenFlow Table	14
2.7 OpenDaylight.....	16
2.8 REST-API OpenDaylight	17
2.9 Aplikasi Peranti Bergerak	18
2.10 Android	19
2.11 Material Design.....	19
2.12 Pengembangan Aplikasi Android	20
2.13 Android Studio.....	20
2.14 <i>Unified Modeling Language (UML)</i>	21
2.15 <i>Rapid Application Development (RAD)</i>	25
2.16 <i>Black-Box Testing (Functional Testing)</i>	27
2.17 Hipotesis	27

BAB III METODOLOGI PENELITIAN.....	28
3.1 Bahan	28
3.2 Peralatan.....	28
3.3 Tahapan Penelitian.....	29
3.4 Instalasi OpenDaylight	33
3.5 Menjalankan Mininet dan Simulasi Topologi Jaringan SDN	35
3.6 Konfigurasi Open vSwitch.....	37
3.7 Identifikasi Persyaratan Aplikasi	38
3.8 Desain RAD	41
3.9 Pengujian Aplikasi	48
BAB IV HASIL PENELITIAN DAN PEMBAHASAN	55
4.1 Pengembangan Aplikasi	55
4.2 Pengujian Penerapan SDN pada Perangkat Raspberry Pi.....	59
4.3 Pengaksesan data dari OpenDaylight.....	60
4.4 Pengujian Penambahan flow berdasarkan Pencocokan Port	62
4.5 Pengujian Penambahan flow berdasarkan Pencocokan Alamat MAC	64
4.6 Pengujian Penambahan flow berdasarkan Pencocokan Alamat IP.....	65
4.7 Black-Box Testing (Functional Testing)	67
BAB V PENUTUP	69
5.1 Kesimpulan	69
5.2 Saran	69
DAFTAR PUSTAKA	70
LAMPIRAN	72

DAFTAR GAMBAR

Gambar 2.1 Aplikasi ACL Berbasis Desktop (Achmad, 2016)	9
Gambar 2.2 Antar muka OpenFlow Manager (OFM) App (CiscoDevNet, 2016)	9
Gambar 2.3 Arsitektur <i>Software Defined Network</i>	12
Gambar 2.4 Mekanisme Switch OpenFlow	14
Gambar 2.5 OpenFlow Table	15
Gambar 2.6 Flowchart aliran Packet Flow pada Switch OpenFlow	16
Gambar 2.7 Logo OpenDaylight <i>Controller</i>	17
Gambar 2.8 Tampilan Dokumentasi REST-API OpenDaylight	18
Gambar 2.9 Logo Android Studio	21
Gambar 2.10 Contoh diagram <i>use case</i> (Awaad, 2005)	22
Gambar 2.11 Contoh diagram <i>activity</i> (Awaad, 2005)	23
Gambar 2.12 Contoh <i>component diagram</i> (Awaad,2005)	24
Gambar 2.13 <i>James Martin RAD Model</i> (James, 1991)	26
Gambar 3.1 Diagram Alir Tahapan Penelitian	30
Gambar 3.2 Fase Model RAD	31
Gambar 3.3 Tampilan awal OpenDaylight pada terminal	34
Gambar 3.4 Tampilan awal OpenDaylight GUI melalui web browser	35
Gambar 3.5 Konfigurasi jaringan pada Mininet di dalam VirtualBox	35
Gambar 3.6 Tampilan simulasi topologi pada MiniEdit	36
Gambar 3.7 Diagram alir Fitur Devices	39
Gambar 3.8 Diagram alir Fitur Flow Tables	40
Gambar 3.9 Diagram alir Fitur Settings	41
Gambar 3.10 Alur proses kerja aplikasi	42
Gambar 3.11 <i>Use Case Diagram</i> Aplikasi manajemen <i>flow</i>	42
Gambar 3.12 <i>Activity Diagram</i> aplikasi	43
Gambar 3.13 Rancangan Antarmuka Daftar Menu	44
Gambar 3.14 Rancangan Antarmuka Menu Devices	45
Gambar 3.15 Rancangan Antarmuka Menu Flow Table	46
Gambar 3.16 Rancangan Antarmuka <i>Flow Details</i>	46
Gambar 3.17 Rancangan Antarmuka <i>Add Flow</i>	47
Gambar 3.18 Rancangan Antarmuka Menu <i>Settings</i>	47
Gambar 3.19 Topologi pengujian	53
Gambar 4.1 Antarmuka Daftar Menu	55
Gambar 4.2 Antarmuka Fitur <i>Devices</i>	56
Gambar 4.3 Antarmuka <i>Flow Table</i>	56
Gambar 4.4 Antarmuka <i>Flow Table</i> dengan statistik	57
Gambar 4.5 Antarmuka <i>Flow Details</i>	57
Gambar 4.6 Antarmuka <i>Select Device</i>	58
Gambar 4.7 Antarmuka <i>General Properties</i>	58
Gambar 4.8 Antarmuka <i>Match</i>	58

Gambar 4.9 Antarmuka <i>General Action</i>	58
Gambar 4.10 Antarmuka <i>Settings</i>	59
Gambar 4.11 Tampilan OpenDaylight GUI	59
Gambar 4.12 Pengambilan data topologi melalui Postman	61
Gambar 4.13 Pengambilan data Flow Table melalui Postman	61
Gambar 4.14 Penambahan data <i>flow</i> melalui Postman	62
Gambar 4.15 Penghapusan <i>flow</i> melalui Postman	62

DAFTAR TABEL

Tabel 2.1 Ringkasan Uraian Penelitian	10
Tabel 3.1 <i>Test Script</i> Pengujian Aplikasi	49
Tabel 4.1 Hasil pengujian ping sebelum penambahan <i>flow</i> skenario 1.....	63
Tabel 4.2 Konfigurasi <i>flow</i> Skenario 1	63
Tabel 4.3 Hasil pengujian ping setelah penambahan <i>flow</i> skenario 1	64
Tabel 4.4 Hasil pengujian ping sebelum penambahan <i>flow</i> skenario 2.....	64
Tabel 4.5 Konfigurasi <i>flow</i> Skenario 2	65
Tabel 4.6 Hasil pengujian ping setelah penambahan <i>flow</i> skenario 2	65
Tabel 4.7 Hasil pengujian ping sebelum penambahan <i>flow</i> skenario 3.....	66
Tabel 4.8 Konfigurasi <i>flow</i> Skenario 3	66
Tabel 4.9 Hasil pengujian ping setelah penambahan <i>flow</i> skenario 3	67
Tabel 4.10 Hasil Pengujian Fungsional	68

INTISARI

PENGEMBANGAN APLIKASI MANAJEMEN OPENFLOW TABLE BERBASIS ANDROID PADA JARINGAN *SOFTWARE DEFINED NETWORK* MENGUNAKAN *OPENDAYLIGHT CONTROLLER*

Manajemen *flow* pada setiap perangkat jaringan SDN merupakan pekerjaan penting dalam menyesuaikan kebutuhan jaringan, mengatur kinerja jaringan, dan menjaga performa jaringan. Kontroler SDN (*OpenDaylight*) memungkinkan manajemen *flow* dilakukan secara terpusat dengan memanfaatkan API untuk diakses menggunakan aplikasi pihak ketiga. Di sisi lain, pertumbuhan penggunaan seluler atau *smartphone* telah memungkinkan aplikasi seluler memberikan akses informasi secara lebih cepat, lebih mudah, akurat, dan fleksibel. Akses informasi yang lebih cepat, lebih mudah, akurat, dan fleksibel tentunya juga dibutuhkan oleh *Network Administrator* dalam melakukan pekerjaannya. Tersedianya aplikasi *smartphone* untuk melakukan manajemen jaringan SDN khususnya dalam manajemen *flow* akan memenuhi kebutuhan *Network Administrator* akan akses informasi yang lebih cepat, lebih mudah, akurat, dan fleksibel. Pada penelitian ini dilakukan pembuatan aplikasi *smartphone* berbasis Android yang berguna untuk melakukan manajemen *flow* pada jaringan SDN yang menggunakan *OpenDaylight* sebagai kontrolernya.

Kata Kunci : *Software Defined Network*, *OpenDaylight*, Manajemen *flow*, Aplikasi Android

ABSTRACT

OPENFLOW TABLE MANAGEMENT ANDROID APPLICATION DEVELOPMENT ON SOFTWARE DEFINED NETWORK USING OPENDAYLIGHT CONTROLLER

Flow management on each SDN network device is an important job for adjusting network requirements, managing network workflow, and maintaining network performance. The SDN controller (OpenDaylight) allows centralized flow management by utilizing APIs for access using third-party applications. On the other hand, mobile or smartphone usage growth has enabled mobile apps to access information more quickly, easily, accurate and flexible. Accessing information become faster, easier, accurate, and flexible also required by Network Administrator in doing its work. The availability of smartphone applications to perform SDN network management, especially in flow management will meet the needs of Network Administrator on accessing information faster, easier, accurate, and flexible. This research is about developing an Android-based smartphone application for flow management on SDN network using OpenDaylight as its controller.

Keywords: *Software Defined Network, OpenDaylight, Flow management, Android Application*

BAB I

PENDAHULUAN

1.1 Latar Belakang

Software Defined Network (SDN) merupakan sebuah konsep pendekatan baru untuk mendesain, mengelola dan mengimplementasikan arsitektur jaringan yang memisahkan antara sistem control (*control plane*) dan sistem *forwarding* (*data plane*) pada perangkat jaringan (Eueng, 2015). Arsitektur SDN memungkinkan suatu jaringan dapat secara dinamis menyesuaikan lingkungan untuk kebutuhan aplikasi maupun kebutuhan pengguna, menyederhanakan manajemen dan meningkatkan skalabilitas jaringan yang diwujudkan melalui implementasi sederhana dari penambahan komponen dan layanan jaringan (Grgurevic, 2015). SDN juga mampu memberikan solusi untuk berbagai permasalahan jaringan seperti memungkinkan sebuah jaringan dapat berkomunikasi dengan berbagai perangkat jaringan yang memiliki vendor dan standar yang berbeda (*proprietary*) tanpa harus tahu bagaimana untuk mengoperasikan perangkat karena lingkungan jaringan yang dikelola dari satu titik menggunakan protokol yang disebut *OpenFlow* (Hikam, 2017).

OpenFlow adalah standar komunikasi protokol yang bertujuan untuk mengontrol sistem *forwarding* (*data plane*), yang telah dipisahkan secara fisik dari sistem kontrol (*control plane*) menggunakan perangkat lunak pengendali (*controller*) pada sebuah server. Pada arsitektur logikal sebuah *OpenFlow Switch* terdapat *Flow Table* yang berfungsi menentukan paket yang datang masuk ke dalam suatu *flow*, dan kemudian mengambil tindakan terhadap paket tersebut. *Flow Table* inilah yang mempunyai peranan penting dalam terjadinya pengiriman dan penerimaan paket komunikasi dalam jaringan. Untuk dapat mengatur sebuah *Flow Table* diperlukan manajemen *flow* dimana pada arsitektur SDN dilakukan oleh *controller* sehingga pengaturan *Flow Table* di seluruh *OpenFlow Switch* dilakukan secara terpusat.

Dalam arsitektur *Software Defined Network*, dibutuhkan sebuah *controller*. Hal ini dikarenakan *controller*-lah yang bertugas mendefinisikan jaringan, mengatur masalah *availability*, laju *traffic data*, *routing* dan *forwarding*, serta menangani seluruh infrastruktur jaringan yang ada di bawahnya. Terdapat berbagai macam *controller* tersedia saat ini. Setiap *controller* memiliki kelebihan dan kekurangannya masing-masing. Oleh karena itu pemilihan *controller* sangat penting dalam membangun sebuah arsitektur SDN. Terdapat beberapa pertimbangan yang perlu dilakukan dalam memilih sebuah *controller* yaitu performa, kemudahan penggunaan, basis pengguna (*user base*), dukungan komunitas (*community support*), dan dukungan API (*northbound API & southbound API*).

Pada penelitian ini, *OpenDaylight* digunakan sebagai *controller* dalam arsitektur SDN. *OpenDaylight* adalah sebuah proyek *open source* kolaboratif dimana Linux Foundation merupakan pengelola utama dan didukung oleh *IBM*, *Cisco*, *Juniper*, *VMWare* dan beberapa vendor jaringan besar lainnya. *OpenDaylight* merupakan *platform controller* SDN yang diimplementasikan pada protokol *OpenFlow*. *OpenDaylight* dipilih karena memiliki beberapa keunggulan dibandingkan dengan *controller* lain, diantaranya berjalan diatas *Java Virtual Machine* sehingga dapat dioperasikan di berbagai Sistem Operasi, memiliki basis pengguna dan dukungan komunitas yang cukup besar, dan memiliki dukungan *Application Programming Interface* (API) yang mudah digunakan. *OpenDaylight* menyediakan *Representational State Transfer* (REST) API untuk melakukan manajemen jaringan yang memungkinkan aplikasi pihak ketiga berkomunikasi dengan *OpenDaylight* untuk melakukan manajemen jaringan termasuk dalam melakukan manajemen *flow*.

Di sisi lain, perkembangan penggunaan perangkat bergerak (*mobile*) khususnya *smartphone* pada beberapa tahun terakhir meningkat pesat. Hal ini disebabkan karena kebutuhan akan akses informasi yang lebih cepat, lebih mudah, akurat, dan fleksibel.

Smartphone dapat memenuhi kebutuhan tersebut melalui aplikasi-aplikasi yang bekerja untuk melakukan pekerjaan spesifik tertentu. Kebutuhan akan akses informasi yang lebih cepat, lebih mudah, akurat, dan fleksibel tentunya juga dibutuhkan oleh *Network Administrator*. Tersedianya aplikasi *smartphone* untuk melakukan manajemen jaringan SDN khususnya dalam manajemen *flow* akan menyederhanakan dan meningkatkan efisiensi *Network Administrator* dalam melakukan kinerjanya.

Berdasarkan data yang didapatkan oleh *Dell EMC* pada tahun 2017 bahwa 81 persen perusahaan-perusahaan teknologi menyatakan ketersediaan aplikasi *mobile* dalam menjalankan bisnis mereka adalah kunci untuk pertumbuhan bisnis. Kemudian berdasarkan data dari *Data Center Connectivity Survey* yang dilakukan oleh *Human Interface Inc.* pada Agustus 2016 menyatakan bahwa 97 persen *data center* telah memiliki jaringan Wi-Fi dan 81 persen memperbolehkan perangkat *mobile* berada di dalam *data center*.

Hingga saat ini belum tersedia aplikasi untuk melakukan manajemen *flow* khususnya untuk OpenDaylight *controller* pada Google Playstore maupun AppStore. Hal tersebut yang menjadi latar belakang penelitian ini. Pada penelitian ini dilakukan implementasi arsitektur jaringan SDN menggunakan Open vSwitch dalam perangkat Raspberry Pi kemudian dilakukan pengembangan aplikasi Android untuk melakukan manajemen *flow* pada jaringan tersebut.

1.2 Rumusan Masalah

Belum tersedianya aplikasi berbasis *mobile* untuk melakukan manajemen *flow* sebagai penunjang mobilitas dan ke-efektif-an pekerjaan seorang *Network Administrator* pada jaringan SDN yang menggunakan OpenDaylight sebagai *controller*-nya.

1.3 Batasan Masalah

Beberapa batasan masalah yang digunakan pada pembuatan proyek akhir ini adalah sebagai berikut :

1. Penelitian ini difokuskan kepada perancangan dan pembuatan aplikasi manajemen *flow* berbasis Android.
2. Jaringan yang digunakan untuk penelitian adalah jaringan SDN yang menggunakan OpenDaylight sebagai *Controller*.
3. OpenDaylight yang digunakan dalam penelitian adalah OpenDaylight versi Berrylium.
4. Aplikasi manajemen *flow* hanya dapat dijalankan pada perangkat *smartphone* dengan Sistem Operasi Android versi Lollipop atau setelahnya.
5. Aplikasi manajemen *flow* diuji pada jaringan lokal SDN.
6. Manajemen *flow* yang dilakukan dalam penelitian meliputi melihat, menambah, merubah, dan menghapus *flow*.

1.4 Tujuan Penelitian

Tujuan dari penelitian dalam proyek akhir ini adalah untuk menghasilkan sebuah aplikasi Android untuk melakukan manajemen *flow* pada jaringan SDN yang menggunakan OpenDaylight sebagai *controller*.

1.5 Manfaat Penelitian

Manfaat yang dapat diambil dari penelitian dan pengerjaan proyek akhir ini adalah sebagai berikut :

1. Mengembangkan sebuah aplikasi yang dapat digunakan oleh *Network Administrator* dalam melakukan manajemen *flow* pada jaringan SDN yang menggunakan OpenDaylight sebagai *controller*-nya.
2. Membantu meningkatkan produktivitas *Network Administrator*.

1.6 Sistematika Penulisan

Untuk menggambarkan secara menyeluruh mengenai masalah yang akan dibahas dalam laporan proyek akhir ini, maka dibuat sistematika penulisan yang terbagi dalam lima bab sebagai berikut :

1. BAB I PENDAHULUAN

Memuat latar belakang masalah, perumusan masalah, batasan masalah, tujuan penulisan, kegunaan penulisan dan sistematika penulisan.

2. BAB II TINJAUAN PUSTAKA

Tinjauan pustaka menurut uraian sistematis tentang informasi yang relevan dan mutakhir yang terkait dengan lingkup materi penelitian atau teknologi yang akan diterapkan. Uraian dalam tinjauan pustaka ini selanjutnya menjadi dasar teori yang digunakan oleh penulis dalam melaksanakan penelitian dan menyajikan argumentasi dalam pembahasan hasil penelitian.

3. BAB III BAHAN DAN METODE PENELITIAN

Memuat bahan, peralatan, tahapan penelitian, dan rancangan sistem serta analisis data yang ada pada penelitian ini.

4. BAB IV ANALISIS HASIL DAN PEMBAHASAN

Bagian ini memuat semua temuan ilmiah yang diperoleh sebagai data hasil penelitian, atau hasil unjuk kerja prrotipe yang dibuat. Pada bagian ini peneliti menyusun secara sistematis disertai argumentasi yang rasional tentang hasil unjuk kerja yang diperoleh dari hasil penelitian.

5. BAB V PENUTUP

Bagian ini memuat kesimpulan serta saran dari penelitian proyek akhir ini.

BAB II

TINJAUAN PUSTAKA

SDN mampu memberikan solusi untuk berbagai permasalahan jaringan seperti memungkinkan sebuah perangkat jaringan dapat berkomunikasi dengan berbagai perangkat jaringan yang memiliki vendor dan standar yang berbeda (*proprietary*) tanpa harus tahu bagaimana untuk mengoperasikan perangkat karena lingkungan jaringan yang dikelola dari satu titik menggunakan protokol yang disebut *OpenFlow* (Hikam, 2017). *OpenFlow* adalah standar komunikasi protokol yang bertujuan untuk mengontrol sistem forwarding (*data plane*), yang telah dipisahkan secara fisik dari sistem kontrol (*control plane*) menggunakan perangkat lunak pengendali (*controller*) pada sebuah server. *OpenFlow* inilah yang memungkinkan perangkat jaringan seperti *router* dan *switch* untuk dikendalikan dengan program (*programmable*) melalui antarmuka yang telah ditentukan. Selain itu, *OpenFlow* juga memberikan fleksibilitas yang tinggi dalam melakukan *forwarding* paket dan melakukan *routing* dari banyak *flow* yang dapat diadaptasikan ke lingkungan yang dapat berubah menggunakan virtualisasi dan *flow-based routing* (Pupatwibul, 2013).

Dalam arsitektur *Software Defined Network*, dibutuhkan sebuah *controller*. Hal ini dikarenakan *controller* lah yang bertugas mendefinisikan jaringan, mengatur masalah *availability*, laju *traffic data*, *routing* dan *forwarding*, serta menangani seluruh infrastruktur jaringan yang ada di bawahnya. *Controller* SDN bekerja berdasarkan protokol seperti *OpenFlow* yang memungkinkan *server* memberitahu kemana paket dikirimkan. Perangkat jaringan meneruskan paket data yang diterima berdasarkan aturan *Flows* yang ditetapkan dari *controller* yang kemudian disimpan dalam *Flow Table* (Hikam, 2017). *Flow* terbagi atas tiga bagian dalam menentukan aturannya yaitu Pencocokan (*match*), Penghitung, dan Aksi (Instruksi). Manajemen *flow* sangat penting dilakukan untuk dapat mendefinisikan kinerja

jaringan dalam menentukan harus kemana suatu paket diantarkan (*forwarding*), diabaikan (*drop*), atau dimodifikasi.

OpenDaylight adalah sebuah proyek *open source* kolaboratif dimana Linux Foundation merupakan pengelola utama dan didukung oleh *IBM, Cisco, Juniper, VMWare* dan beberapa vendor jaringan besar lainnya. *OpenDaylight* merupakan *platform controller* SDN yang diimplementasikan pada protokol *OpenFlow*. *Controller* ini menyediakan *Representational State Transfer Application Programming Interface* (REST API) yang memungkinkan aplikasi pihak ketiga melakukan perubahan terhadap jaringan melalui komunikasi dengan *OpenDaylight*. Manajemen *flow* merupakan salah satu dari beberapa API yang disediakan oleh *OpenDaylight*. API inilah yang memungkinkan aplikasi pihak ketiga untuk melakukan manajemen *flow* melalui komunikasi dengan *controller*.

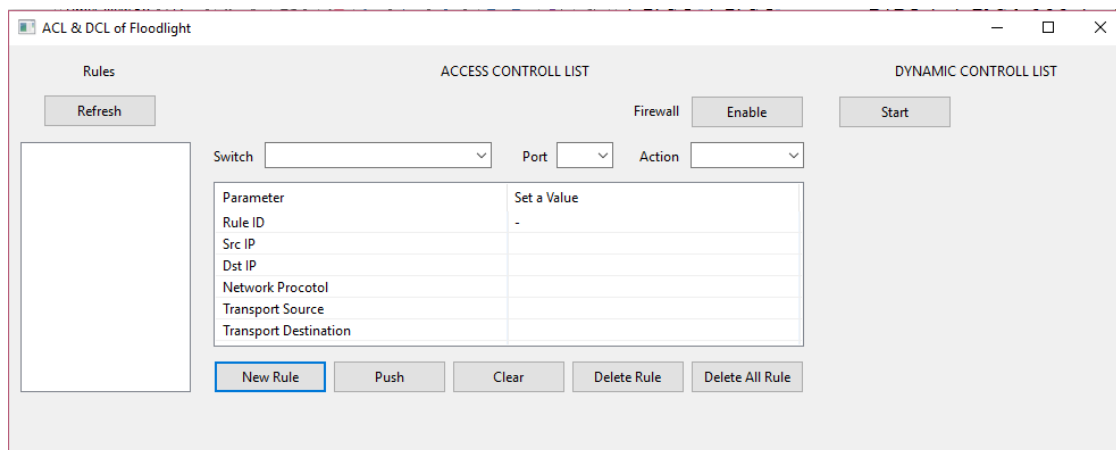
Penggunaan dan pengembangan aplikasi seluler (*Mobile Application*) merupakan sektor yang relatif baru dan berkembang sangat pesat. Aplikasi seluler berjalan pada perangkat telepon seluler genggam yang mudah digunakan dan dapat diakses dari mana saja. Pada saat ini, banyak orang menggunakan aplikasi seluler untuk keperluan sehari-hari seperti mengontak teman, menelusuri internet, membuat dokumen, hiburan, dan sebagainya. Aplikasi seluler tidak hanya berdampak pada pengguna namun juga memiliki peran penting pada bisnis dan industri (Islam, 2010). Aplikasi seluler terdiri dari program yang berjalan pada sebuah perangkat seluler dan mengerjakan pekerjaan tertentu pada pengguna. Aplikasi Seluler memberikan kelebihan dibanding dari platform lain yang diantaranya adalah pengaksesan lebih cepat, lebih mudah, akurat, dan fleksibel.

Penelitian arsitektur *Software Defined Network* untuk diterapkan ke jaringan pada umumnya masih dilakukan dengan bantuan software mininet. *Mininet* merupakan *emulator* jaringan yang mensimulasikan koleksi dari *host-end, switch, router, dan link* pada *single*

kernel Linux. Adapun penelitian yang juga menerapkan SDN pada *switch* berkapabilitas OpenFlow sebagai contoh penelitian yang berjudul “Prototipe Infrastruktur *Software Defined Network* dengan Protokol *OpenFlow* menggunakan Ubuntu sebagai Kontroler” (Katardie, 2014) mengimplementasikan *OpenFlow* pada perangkat mikrotik dan menggunakan *OpenDaylight* yang diinstall pada Ubuntu sebagai kontrolernya. Hasil penelitian ini menyatakan bahwa kinerja *switch* berkapabilitas *OpenFlow* cukup baik jika dibandingkan dengan simulator *mininet*. Hal ini dibuktikan dari selisih gap pengiriman paket ICMP baik *routing* maupun VLAN tidak jauh berbeda dengan hasil yang diperoleh simulator *mininet*.

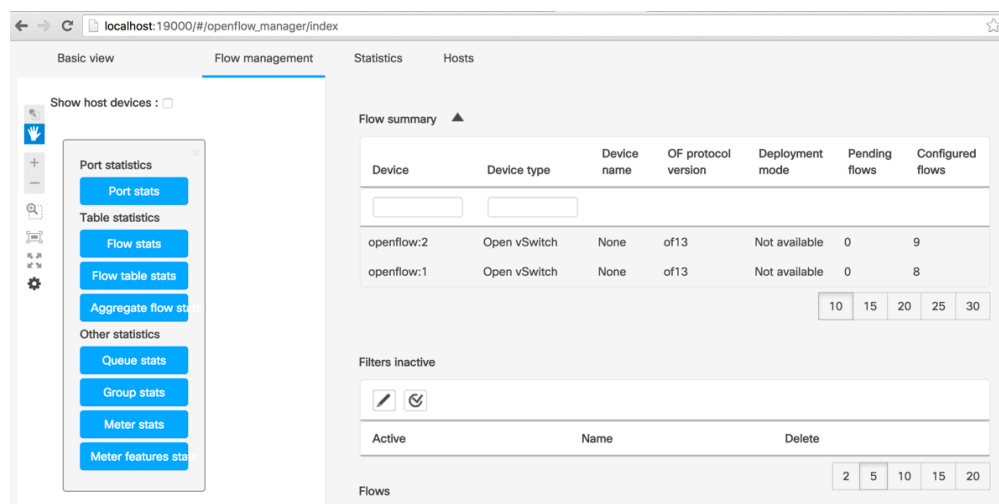
Penelitian berjudul “Implementasi dan Analisis Kinerja Arsitektur *Software Defined Network* berbasis *OpenDaylight Controller*” (Hikam, 2017) membandingkan jaringan SDN dengan *OpenDaylight* sebagai kontroler dengan jaringan tradisional dan jaringan simulasi *mininet* menyatakan bahwa Arsitektur SDN memiliki fleksibilitas dan skalabilitas yang baik, meskipun semakin besar jaringan yang dimiliki maka performansi juga semakin menurun.

Penelitian berjudul “Implementasi *Access Control List* dan *Dynamic Control List* berbasis Aplikasi Desktop pada *Software Defined Network* menggunakan *FloodLight Controller*” (Achmad, 2018) membuat sebuah aplikasi desktop yang berfungsi untuk melakukan manajemen *Access Control List* dan *Dynamic Control List*. Pada penelitian ini aplikasi desktop berbasis JAVA dibuat dengan memanfaatkan REST API dari *Floodlight Controller* untuk melakukan perubahan data pada jaringan SDN. Penelitian ini menyatakan bahwa aplikasi yang dibuat akan mempermudah administrator jaringan dalam melakukan manajemen *Access Control List* dan *Dynamic Control List*. Antarmuka aplikasi tersebut dapat dilihat pada Gambar 2.1 dibawah ini.



Gambar 2.1 Aplikasi ACL Berbasis Desktop (Achmad, 2016)

Pada tahun 2016 CiscoDevNet mengembangkan sebuah aplikasi Manajemen *OpenFlow* berbasis Web bernama “OpenDaylight OpenFlow Manager (OFM) App”. Dimana aplikasi ini dapat diintegrasikan pada *OpenDaylight Controller* yang memiliki fungsi utama untuk melihat, menambah, mengubah, dan menghapus *flow* yang terdapat pada jaringan SDN. Aplikasi ini masih dalam pengembangan dan dikembangkan secara *open source*. Antarmuka aplikasi tersebut dapat dilihat pada Gambar 2.2 dibawah ini.



Gambar 2.2 Antar muka OpenFlow Manager (OFM) App (CiscoDevNet, 2016)

Adapun ringkasan berdasarkan uraian penelitian di atas dapat dilihat pada tabel di bawah ini.

Tabel 2.1 Ringkasan Uraian Penelitian

No	Judul Penelitian	Pokok Bahasan	Tipe Penelitian	Jenis Controller	Penulis
1	Prototipe Insfrastruktur <i>Software Defined Network</i> dengan Protokol <i>OpenFlow</i> menggunakan Ubuntu sebagai Kontroler	Membuat prototipe infrastruktur <i>Software Definged Network</i> dengan protokol <i>OpenFlow</i> menggunakan Ubuntu	Implementasi	OpenDaylight	Rikie Katardie
2	Implementasi dan Analisis Kinerja Arsitektur <i>Software Defined Network</i> berbasis <i>OpenDaylight Controller</i>	Membangun arsitektur SDN pada perangkat Mikrotik, melihat kinerja yang dihasilkan, dan Membandingkan dengan arsitektur tradisional	Implementasi	OpenDaylight	Muhammad Hikam Hidayat
3	Implementasi <i>Access Control List</i> dan <i>Dynamic Control List</i> berbasis Aplikasi Desktop pada <i>Software Defined Network</i> menggunakan <i>FloodLight Controller</i>	Membuat aplikasi untuk melakukan manajemen <i>Access Control List</i> dan <i>Dynamic Control List</i> berbasis Desktop	Implementasi	FloodLight	Solaeman Achmad
4	OpenDaylight OpenFlow Manager (OFM) App	Membangun aplikasi manajemen <i>OpenFlow</i> berbasis Web	Implementasi	OpenDaylight	Jan Medved dkk. (CiscoDevNet)

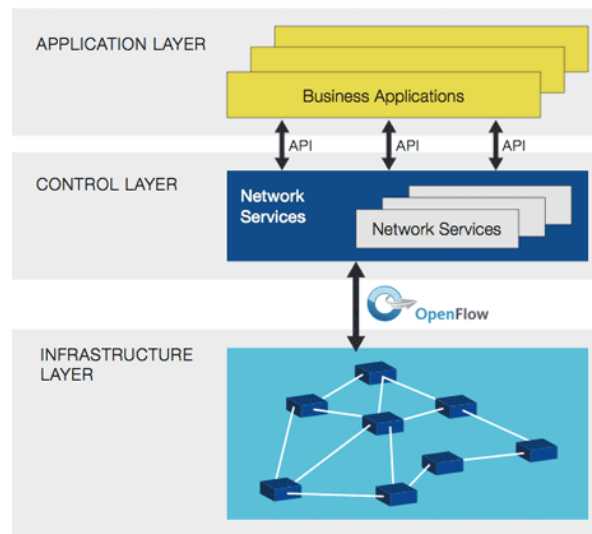
2.1 Konsep Jaringan Komputer

Jaringan komputer adalah hubungan dua simpul yang umumnya berupa komputer atau yang lebih ditujukan untuk melakukan pertukaran data atau untuk berbagi perangkat lunak, perangkat keras dan bahkan berbagi kekuatan pemrosesan (Kadir, 2003).

Jaringan komputer ini di dalamnya berisi kumpulan beberapa komputer dan perangkat lain seperti router, switch, dan sebagainya yang saling terhubung satu sama lain melalui media perantara. Media perantara ini dapat berupa media kabel ataupun tanpa kabel. Untuk saling dapat bertukar data, maka dari satu komputer ke komputer lain harus saling terhubung. Perangkat-perangkat utama yang membangun jaringan komputer tersebut antara lain, router, switch, host, firewall dan lainnya. Tujuan utama dibangunnya suatu sistem jaringan komputer adalah untuk membawa data dan informasi dari sisi pengirim menuju ke penerima secara cepat dan tepat tanpa adanya kesalahan melalui media transmisi atau media komunikasi tertentu (Oetomo, 2003).

2.2 Software Defined Network

Software Defined Network (SDN) merupakan paradigma baru dalam mendesain, mengelola, dan mengimplementasikan arsitektur jaringan yang memisahkan antara sistem kontrol (*control plane*) dan sistem *forwarding* (*data plane*) pada perangkat jaringan. *Software Defined Network* memungkinkan membangun sebuah jaringan dimana seluruh fungsionalitas perangkat jaringannya dikendalikan oleh *software*. Awal mula terciptanya teknologi *Software Defined Network* dimulai tidak lama setelah Sun Microsystems merilis Java pada tahun 1995, namun pada saat itu belum cukup membangunkan para peneliti untuk mengembangkan teknologi tersebut. Baru pada tahun 2008 *Software Defined Network* mulai dikembangkan di UC Berkeley dan Stanford University. Dan kemudian teknologi tersebut mulai dipromosikan oleh Open Networking Foundation yang didirikan pada tahun 2011 untuk memperkenalkan teknologi SDN dan *OpenFlow* (Linuwih, 2016).



Gambar 2.3 *Arsitektur Software Defined Network*

Setiap lapisan pada arsitektur *Software Defined Network* (SDN) dapat bekerja secara independen. Melalui antarmuka jaringan, perangkat fisik yang berbeda vendor akan disatukan dengan suatu perangkat lunak agar dapat dikonfigurasi (Wensong. 2010). Aspek arsitektur ini memungkinkan *administrator* jaringan untuk mengatasi beberapa tantangan dalam perkembangan jaringan komputer.

Gambaran logis mengenai arsitektur *Software-Defined Networking* (SDN) pada Gambar 4.1 menunjukkan terdapat 3 lapisan pada arsitektur *Software-Defined Networking* (SDN) (F. Hu, 2014), yaitu:

1. Lapisan Infrastruktur (*Infrastructure Layer*)

Lapisan infrastruktur terdiri dari elemen-elemen jaringan dan perangkat keras yang menjalankan fungsi *packet switching* dan *forwarding*.

2. Lapisan Kontrol (*Control Layer*)

Lapisan Kontrol menyediakan fungsionalitas kontrol secara padu yang mengawasi perilaku jaringan *forwarding* melalui *open interface*.

3. Lapisan Aplikasi (*Application Layer*)

Lapisan Aplikasi berfungsi untuk menyediakan *interface* dalam pembuatan program aplikasi yang kemudian berguna untuk mengatur serta mengoptimalkan jaringan secara baik dan fleksibel.

Pemisahan antara bagian kontrol dan *forwarding* pada SDN dapat diibaratkan dengan sistem operasi dimana bagian kontrol memberi antar muka terhadap jaringan yang dapat digunakan untuk melakukan manajemen dan menawarkan fungsionalitas baru untuk jaringan. SDN dapat diaplikasikan secara luas seperti pada lingkungan jaringan yang heterogen seperti media jaringan ataupun layanannya.

2.3 Flow

Menurut Roberts (2015), *flow* merupakan aliran data yang dibatasi oleh waktu dan ruang yang memiliki identifikasi yang sama. Aliran data dibatasi oleh ruang dikarenakan observasi data dilakukan pada antarmuka (*interface*) tertentu (misalnya pada perangkat Router), kemudian data dibatasi oleh waktu dimana data diobservasi tidak lebih dari interval tertentu, yang biasanya dalam hitungan detik. Pengidentifikasi data didapat dari parameter *IP Header* tertentu, termasuk alamat IP dan parameter lain, seperti *Ipv6 flow label*.

2.4 OpenFlow

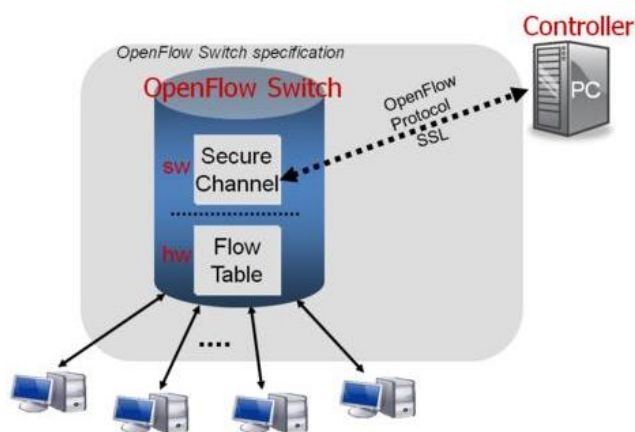
OpenFlow adalah standar antarmuka komunikasi yang menghubungkan antara lapisan *controller* dengan lapisan forwarding pada arsitektur SDN. OpenFlow mengizinkan akses langsung dan manipulasi forwarding plane dari sebuah perangkat jaringan seperti switch dan router baik fisik maupun virtual. Secara sederhana OpenFlow merupakan sebuah antarmuka antara SDN *controller* dengan perangkat switch. (Hyojoon, 2013). Komponen utama pada arsitektur jaringan dengan *controller* berbasis *OpenFlow* adalah *switch OpenFlow* dan *Controller* yang dijalankan pada sebuah *Server*.

2.5 OpenFlow Switch

OpenFlow switch dapat dibagi menjadi tiga bagian yaitu:

- Sebuah *flow table* yang mengindikasikan bahwa switch harus memproses flow yang ada di dalamnya. Daftar *flow* ini dibuat berdasarkan *actions* yang mana bersinggungan langsung dengan setiap *flow*.
- Sebuah saluran yang aman dibutuhkan untuk menghubungkan *switch* dengan *controller*. Melalui saluran ini, *OpenFlow* menyediakan jalur komunikasi antara *switch* dan *controller* melalui protokol yang disebut *OpenFlow protocol*.
- Komponen yang terakhir adalah protokol *OpenFlow*. Protokol ini menyediakan sebuah standar dan komunikasi terbuka antara *controller* dan *switch*. *OpenFlow protocol* menentukan ke *interface* manakah *flow* akan diterapkan dari *flow table*.

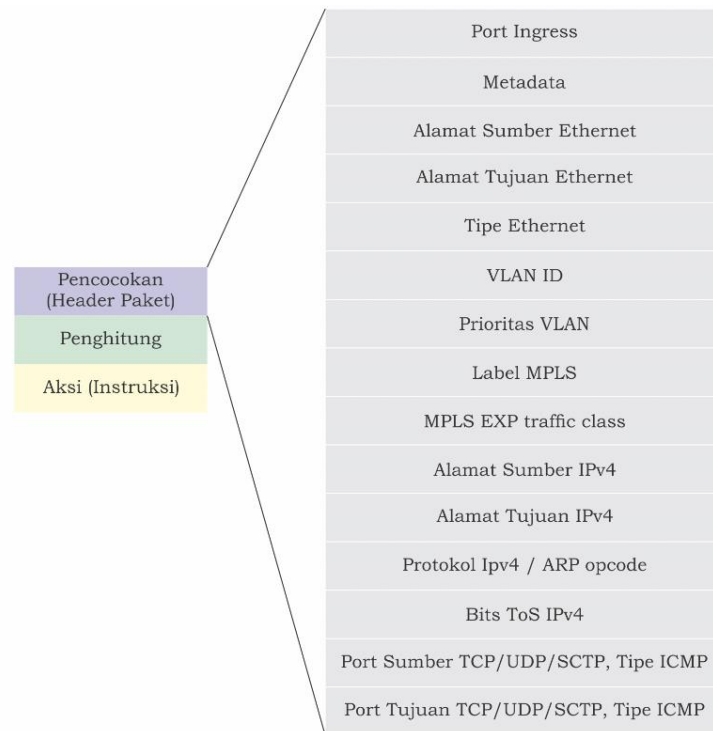
Gambaran mengenai konsep dari *OpenFlow switch* dapat dilihat pada Gambar 2.4 (McKeownd dkk, 2008) dibawah ini.



Gambar 2.4 Mekanisme Switch OpenFlow

2.6 OpenFlow Table

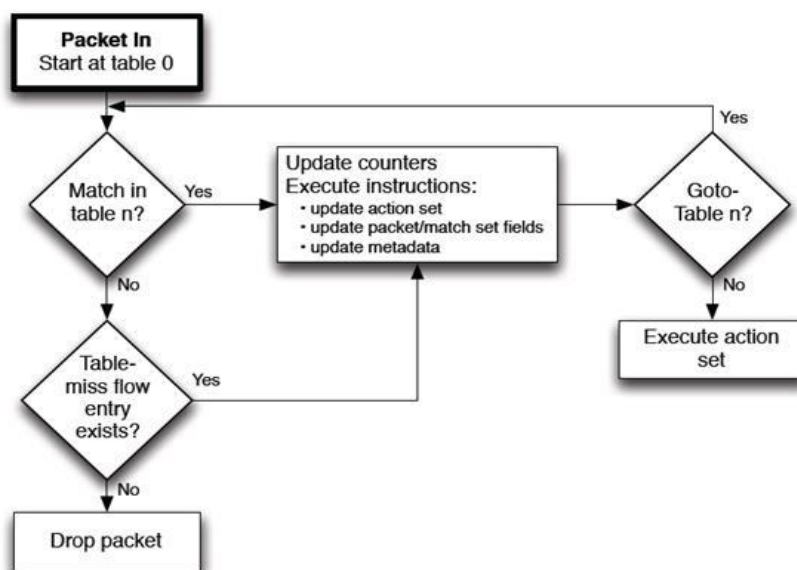
Pada *switch OpenFlow* terdapat tabel yang berisi dari tiga bagian yaitu : Pencocokan (*match*), Penghitung, dan Aksi seperti yang terdapat pada Gambar 2.5.



Gambar 2.5 OpenFlow Table

- **Pencocokan** : Digunakan untuk mencocokkan informasi terbaru dalam *flow table* terhadap paket data yang datang.
- **Penghitung** : Digunakan untuk memperbarui statistik dari paket setelah pencocokan dilakukan dan untuk menjaga jumlah paket dan *byte* untuk setiap *flow*.
- **Aksi (Instruksi)** : Digunakan untuk melakukan tindakan tertentu yang diatur oleh *controller* untuk memproses paket yang masuk setelah pencocokan telah dilakukan.

Ketika suatu paket dikirimkan ke *switch*, *switch OpenFlow* akan memproses paket seperti yang ditunjukkan pada Gambar 2.6 dibawah ini.



Gambar 2.6 Flowchart aliran Packet Flow pada Switch OpenFlow

Switch OpenFlow menerima paket, dan akan mencocokkan *ingress port* dan *headers* dari paket dengan *match fields* pada *flow entries*. Jika pencocokan sesuai, maka aksi tertentu akan dilakukan oleh *SDN controller* dan *counter* mendapatkan update. Jika *switch* menerima paket tanpa aturan pencocokan dengan *flow entries* pada *flow table*, maka secara default paket akan dikirimkan pada *controller*. (Banjar dkk, 2014)

2.7 OpenDaylight

Controller SDN adalah aplikasi SDN yang mengelola *flow control* untuk mengaktifkan intelligence networking. *Controller SDN* bekerja berdasarkan protokol seperti OpenFlow yang memungkinkan *server* memberitahu kemana paket dikirimkan. Perangkat meneruskan paket data yang diterima berdasarkan aturan yang ditetapkan dari *controller* (Anggara, 2015). Jika perangkat tidak memiliki aturan untuk paket data yang telah tiba, perangkat meneruskan paket ke *controller* untuk ulasan. *Controller* menentukan apa yang harus dilakukan dengan paket dan, jika perlu, mengirimkan aturan baru untuk perangkat sehingga dapat menangani paket data di masa depan dengan cara yang sama (Hu, 2014). Terdapat

banyak jenis *controller* yang tersedia secara *open source* salah satunya adalah OpenDaylight.

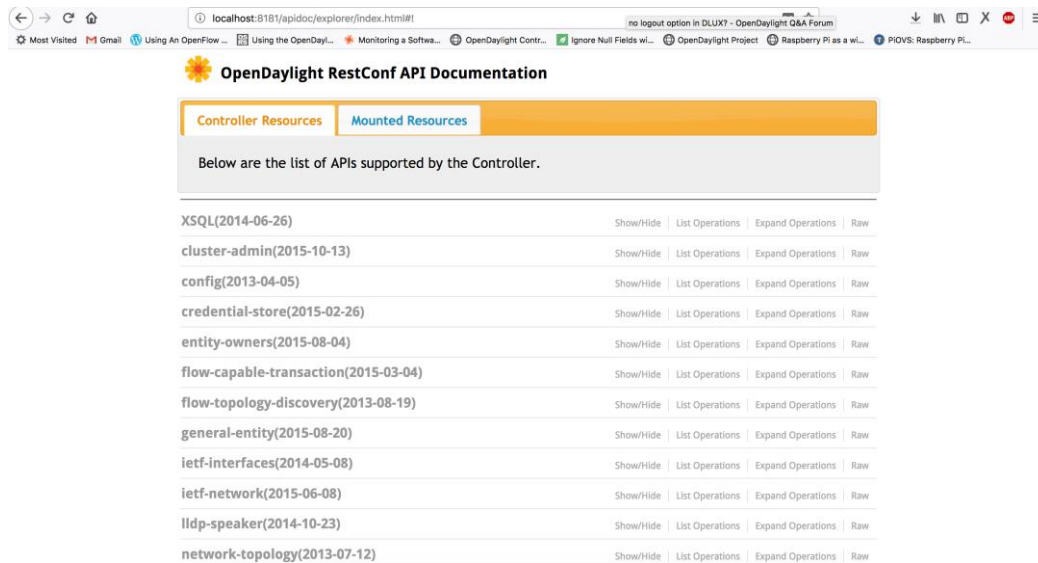
OpenDaylight merupakan sebuah infrastruktur *controller* yang memiliki ketersediaan tinggi, modular, *extensible*, *scalable* dan multi-protokol, dibangun untuk penyebaran SDN di jaringan heterogen, *multi-vendor*, serta modern. OpenDaylight Project adalah sebuah proyek *open source* kolaboratif di Linux Foundation yang bertujuan untuk mempercepat adopsi Software-Defined Network (SDN) dan menciptakan pondasi yang solid untuk Network Function Virtualization (NFV) dengan pendekatan yang lebih transparan untuk mendorong inovasi baru dan mengurangi risiko (Riswanto, 2015). Linux Foundation Collaborative Projects sendiri merupakan proyek perangkat lunak didanai secara independen yang bertujuan untuk memanfaatkan kekuatan pengembangan kolaboratif untuk bahan bakar inovasi di industri dan ekosistem. Logo OpenDaylight dapat dilihat pada Gambar 2.7.



Gambar 2.7 Logo OpenDaylight Controller

2.8 REST-API OpenDaylight

OpenDaylight menyediakan dokumentasi ketersediaan REST-API yang dapat diakses melalui alamat url <http://localhost:8181/apidoc/explorer/index.html> dimana pada dokumentasi ini terdapat daftar *url* apa saja yang dapat dimanfaatkan untuk melakukan komunikasi secara REST. Tampilan dokumentasi REST tersebut tampak seperti pada Gambar 2.8.



The screenshot shows the OpenDaylight RestConf API Documentation page. It features a navigation bar with 'Controller Resources' and 'Mounted Resources' tabs. Below the tabs, a message states: 'Below are the list of APIs supported by the Controller.' A table lists various APIs with their dates and associated actions.

API Name	Show/Hide	List Operations	Expand Operations	Raw
XSQL(2014-06-26)	Show/Hide	List Operations	Expand Operations	Raw
cluster-admin(2015-10-13)	Show/Hide	List Operations	Expand Operations	Raw
config(2013-04-05)	Show/Hide	List Operations	Expand Operations	Raw
credential-store(2015-02-26)	Show/Hide	List Operations	Expand Operations	Raw
entity-owners(2015-08-04)	Show/Hide	List Operations	Expand Operations	Raw
flow-capable-transaction(2015-03-04)	Show/Hide	List Operations	Expand Operations	Raw
flow-topology-discovery(2013-08-19)	Show/Hide	List Operations	Expand Operations	Raw
general-entity(2015-08-20)	Show/Hide	List Operations	Expand Operations	Raw
ietf-interfaces(2014-05-08)	Show/Hide	List Operations	Expand Operations	Raw
ietf-network(2015-06-08)	Show/Hide	List Operations	Expand Operations	Raw
lldp-speaker(2014-10-23)	Show/Hide	List Operations	Expand Operations	Raw
network-topology(2013-07-12)	Show/Hide	List Operations	Expand Operations	Raw

Gambar 2.8 Tampilan Dokumentasi REST-API OpenDaylight

2.9 Aplikasi Peranti Bergerak

Menurut Technopedia, aplikasi perangkat bergerak merupakan sebuah jenis perangkat lunak aplikasi yang didesain untuk dapat dijalankan pada peranti bergerak dikembangkan untuk membuat layanan yang diakses melalui *Personal Computer* (PC) dapat diakses melalui peranti bergerak oleh pengguna. Aplikasi pada peranti bergerak berukuran relatif kecil dan memiliki fungsi yang terbatas. Pada awalnya, aplikasi peranti bergerak dipopulerkan oleh Apple Incorporation melalui produk-produknya berupa iPhone, iPad, dan iPod Touch.

Aplikasi peranti bergerak terdiri dari kumpulan program set yang berisikan tugas-tugas tertentu yang dibutuhkan oleh pengguna dan dapat dijalankan pada peranti bergerak. Aplikasi peranti bergerak adalah sebuah segmen pengembangan baru dan cepat dari teknologi informasi dan komunikasi secara global. Aplikasi peranti bergerak mudah, ramah pengguna, tidak memerlukan biaya yang tinggi (tidak mahal), dapat diunduh, dan dapat dijalankan pada kebanyakan peranti bergerak seperti ponsel pintar dan komputer tablet termasuk dari peranti yang masuk ke dalam kategori *entry level*. Aplikasi peranti bergerak

berjalan pada area yang luas dan bervariasi dalam fungsinya, seperti untuk telepon, mengirim pesan, menjajah dunia internet, *chatting*, komunikasi jaringan sosial, dan hiburan seperti musik dan permainan.

2.10 Android

Android merupakan sebuah sistem operasi yang dikembangkan oleh Google. Sistem operasi ini berbasis pada *kernel* Linux dan didesain secara khusus untuk sebuah peranti bergerak yang memiliki layar sentuh seperti ponsel pintar dan komputer tablet. Tampilan pengguna yang dimiliki oleh sistem operasi Android secara garis besar didasarkan pada manipulasi secara langsung menggunakan gerakan sentuhan pada layar sentuh seperti menggeser, mengetuk, dan menahan pada layar sentuh. Sistem operasi Android saat ini juga dikembangkan dan digunakan pada perangkat lain seperti Android TV untuk televisi, Android Auto untuk mobil, dan Android Wear untuk *smart watch*, yang tampilan khusus setiap jenis sistem operasi Android yang digunakan.

Berdasarkan data yang diperoleh oleh International Data Corporation (IDC) pada kuartal pertama tahun 2017, sistem operasi Android menempati urutan pertama dengan persentase 85.0% dari seluruh sistem operasi peranti bergerak yang terjual di pasar, diikuti dengan sistem operasi milik Apple Inc yaitu iOS sebesar 14.7%, Windows Phone 0.1%, dan sistem operasi lain sebesar 0.1%. Data tersebut menunjukkan bahwa sistem operasi Android sangat jauh diminati untuk digunakan pada peranti bergerak.

2.11 Material Design

Material Design merupakan sebuah konsep desain yang dikembangkan oleh Google pada tahun 2014. Material Design merupakan konsep desain yang didasarkan pada desain elemen cetak seperti tipografi, *grids*, *space*, skala, warna dan perbandingan untuk menciptakan hierarki, makna, dan fokus yang membawa pengguna ke dalam sebuah

pengalaman. Material Design mengadopsi peralatan dari area desain cetak, seperti garis utama klise struktural, mendorong konsistensi dengan mengulang elemen visual, *grids* yang terstruktur, dan jarak pada *platform* layar dengan ukuran yang berbeda-beda. Skala pada tata letak dapat menyesuaikan dengan berbagai ukuran layar untuk mendukung sebuah aplikasi yang dapat diperluas. Saat ini, Material Design digunakan sebagai salah satu panduan perancangan dan pengembangan antar muka web dan menjadi panduan utama dalam pengembangan antarmuka aplikasi Android (Techcrunch, 2014).

2.12 Pengembangan Aplikasi Android

Aplikasi Android merupakan perangkat lunak yang bekerja pada perangkat seluler dengan sistem operasi Android dimana digunakan untuk melakukan pekerjaan tertentu. Aplikasi Android dapat dikembangkan dengan berbagai macam bahasa, *tools*, dan *framework*. Pada awal perkembangannya kebanyakan pengembangan aplikasi android dilakukan menggunakan Bahasa Java dengan IDE (IDE) Eclipse, kemudian berkembang bahasa dan *tools* lain seperti React Native, Cordova, jQuery Mobile, dan lain sebagainya. Pada tahun 2017, Google menambahkan bahasa Kotlin sebagai bahasa pemrograman pengembangan Android yang resmi serta Android Studio sebagai IDE-nya. Pengembangan Aplikasi Android memiliki langkah-langkah pengembangan seperti rekayasa perangkat lunak pada umumnya yaitu memiliki tahapan seperti Pengumpulan kebutuhan *software*, Perancangan Antarmuka, Implementasi kode, Uji coba, *Bug fixing*, dan *Deployment*.

2.13 Android Studio

Android Studio merupakan sebuah *integrated development environment (IDE)* khusus untuk membangun aplikasi yang berjalan pada *platform* Android. Android Studio ini berbasis pada IntelliJ IDEA, sebuah *IDE* untuk bahasa pemrograman Kotlin. Bahasa Kotlin bersifat *case sensitive*, sehingga harus memperhatikan penggunaan huruf besar dan kecil. Sedangkan untuk membuat tampilan atau *layout*, digunakan bahasa XML. Android Studio

juga terintegrasi dengan *Android software development kit* (SDK) untuk *deploy* ke perangkat Android. Gambar 2.9 merupakan logo dari Android Studio.



Gambar 2.9 Logo Android Studio

2.14 Unified Modeling Language (UML)

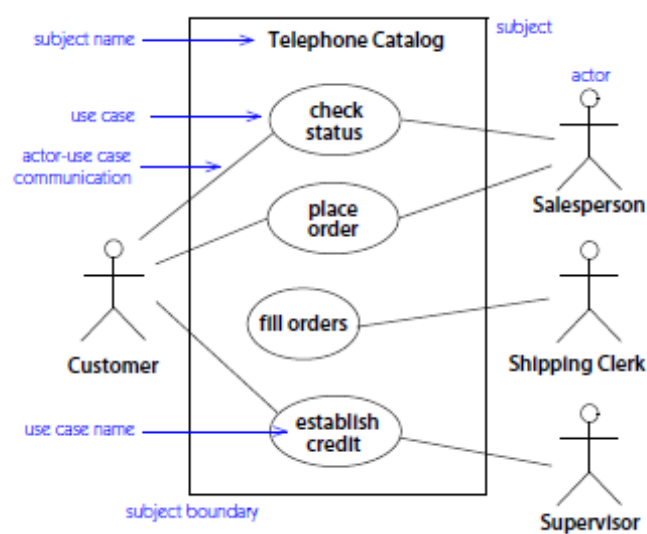
UML merupakan suatu bahasa pemodelan visual yang berguna untuk menentukan, menggambarkan, membangun, dan mendokumentasikan sistem perangkat lunak (Awaad, 2005). Penggunaan UML dimaksudkan untuk menyatukan berbagai teknik pemodelan dalam suatu standar. UML dibuat dengan melakukan analisa terhadap sistem yang akan dibuat. Berbagai komponen yang ada pada proses pengembangan perangkat lunak divisualisasikan agar mempermudah pemahaman serta membantu dalam proses iterasi pengembangan.

Dalam pengembangan UML terdapat beberapa paradigma atau yang biasa disebut dengan *view*. Secara sederhana suatu *view* merupakan bagian dari UML yang merepresentasikan aspek dari suatu sistem. Setiap *view* dideskripsikan dengan diagram dengan tujuan untuk mempermudah pemahaman terhadap *view* tersebut. Beberapa *view* yang ada pada UML diantaranya adalah *use case view* dengan diagram *use case*, *activity view* dengan diagram *activity*, dan *design view* dengan diagram komponen. Ketiga jenis *view* tersebut dijelaskan sebagai berikut (Awaad, 2005):

1. Use Case View

User Case View memodelkan fungsionalitas subyek (seperti suatu sistem) seperti yang dirasakan oleh agen luar yang disebut aktor, yang berinteraksi dengan subyek

dari sudut pandang tertentu. Sebuah *use case* merupakan sebuah unit dari fungsionalitas yang diekspresikan sebagai sebuah transaksi antara aktor dan subyek. Tujuan dari *use case view* adalah untuk membuat daftar aktor dan *uses case*, serta untuk memperlihatkan aktor mana yang berpartisipasi untuk setiap *use case*. Aktor dapat berupa manusia, sistem komputer dan proses. Aktor dapat berpartisipasi dalam suatu atau lebih *use case*. Contoh diagram *use case* dapat dilihat pada Gambar 2.10.



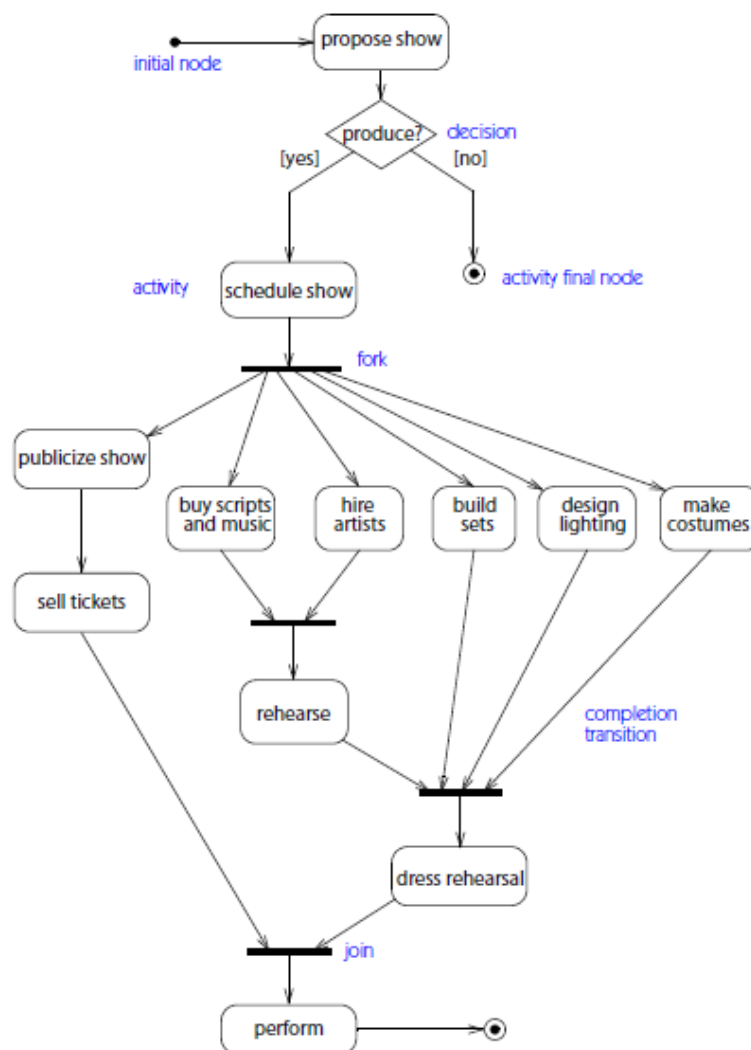
Gambar 2.10 Contoh diagram *use case* (Awaad, 2005)

2. Activity View

Sebuah *activity view* menunjukkan alur pengendalian dari aktivitas-aktivitas komputasi termasuk melakukan perhitungan atau alur kerja. Sebuah aksi merupakan langkah paling sederhana dari suatu komputasi. Sebuah *activity node* adalah sekumpulan dari beberapa aksi atau sub aktivitas. Sebuah aktivitas mendeskripsikan antara komputasi *sequential* dan komputasi *concurrent*. Aktivitas dari suatu sistem dapat dilihat dalam sebuah diagram aktivitas.

Activity didefinisikan dengan memuat *activity node* didalamnya. *Activity node* menggambarkan eksekusi dari suatu *statement* dalam sebuah prosedur atau performa dari tahap eksekusi dalam aliran kerja. Setiap *nodes* dapat dihubungkan dengan garis

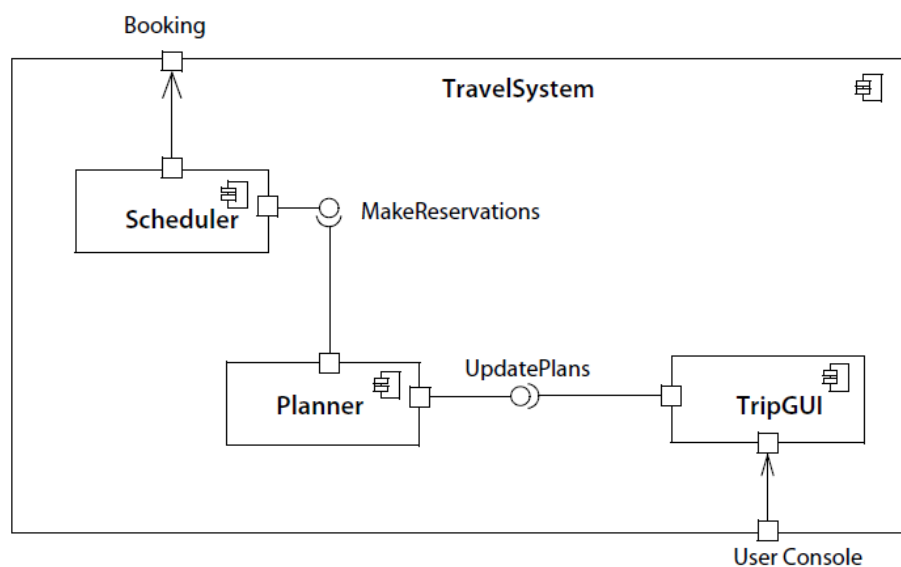
berupa *control flow* atau *data flow*. *Control flow* menggambarkan aliran kendali dari program yang dieksekusi sedangkan *data flow* menggambarkan aliran pertukaran data yang terjadi antar *node*. Setiap *node* dapat bekerja dengan menunggu selesainya komputasi *node* yang sebelumnya. Saat komputasi selesai dilakukan, eksekusi komputasi dilanjutkan ke ujung keluaran *node* dari *node* yang sudah selesai tersebut. Suatu *activity* dapat dipisahkan dengan menggunakan *partition*. Hal ini dimaksudkan untuk mengelola *activity* sesuai tanggung jawabnya. Contoh diagram aktivitas dapat dilihat pada Gambar 2.11.



Gambar 2.11 Contoh diagram *activity* (Awaad, 2005)

3. Design View

Design View menunjukkan organisasi logis dari komponen yang dapat digunakan kembali ke dalam unit yang dapat diandalkan, yang disebut komponen. Sebuah komponen memiliki seperangkat antarmuka eksternal dan implementasi internal yang tersembunyi. Komponen berinteraksi melalui antarmuka sehingga ketergantungan pada komponen lainnya dihindari. Selama implementasi, komponen apa pun yang mendukung antarmuka dapat digantikan, hal ini memungkinkan berbagai bagian sistem dikembangkan tanpa ketergantungan pada implementasi internal. Komponen merupakan modular suatu sistem logis atau fisik yang dapat dilihat secara eksternal jauh lebih ringkas daripada implementasinya. Komponen tidak bergantung langsung pada komponen lainnya, tetapi pada antarmuka yang komponen pendukungnya. Komponen dalam model dapat digantikan oleh komponen lain yang mendukung antarmuka yang tepat, dan contoh komponen dalam konfigurasi sistem dapat digantikan oleh sebuah *instance* dari komponen yang mendukung antarmuka yang sama. Gambar 2.12 merupakan contoh dari *design view* berupa *component diagram*.

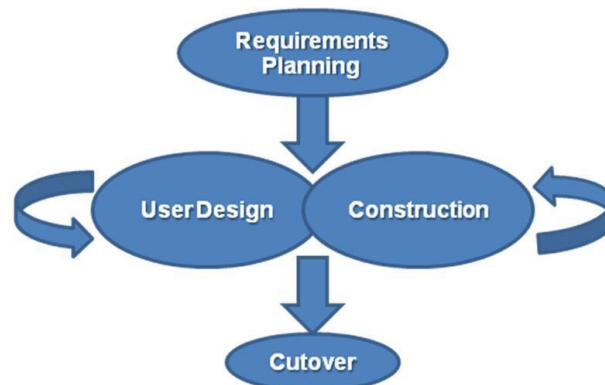


Gambar 2.12 Contoh *component diagram* (Awaad,2005)

2.15 *Rapid Application Development (RAD)*

Rapid Application Development (RAD) merupakan salah satu metode dalam pengembangan perangkat lunak yang diperkenalkan pertama kali oleh Barry Boehm sebagai pengembangan metode *waterfall* dengan nama *spiral model* (Boehm, 1987). Berdasarkan ide yang sudah dikembangkan oleh Boehm, James Martin mengembangkan RAD. Secara umum, RAD melakukan pendekatan terhadap proses pengembangan perangkat lunak dengan mengurangi penekanan pada tahap perencanaan dan meningkatkan penekanan pada tahap pengembangan perangkat lunak. Berbeda dengan metode *waterfall* yang memerlukan spesifikasi perangkat lunak yang ditetapkan secara ketat sebelum mulai memasuki proses pengembangan perangkat lunak. Pendekatan yang dilakukan dengan menggunakan metode RAD adalah dengan penekanan terhadap adaptasi dan perlunya menyesuaikan kebutuhan dalam menanggapi pengetahuan yang diperoleh seiring dengan kemajuan proyek. Purwarupa aplikasi sering digunakan untuk menentukan spesifikasi dari aplikasi yang dikembangkan. RAD sangat sesuai untuk pengembangan perangkat lunak yang didorong oleh persyaratan antarmuka pengguna meskipun tidak hanya terbatas pada antarmuka pengguna (James, 1991).

Berdasarkan pada metode RAD yang dikembangkan oleh James Martin, terdapat empat buah fase yaitu *requirement planning phase*, *user design phase*, *construction phase*, dan *cutover phase*. Gambar 2.13 merupakan model RAD yang dikembangkan oleh James Martin.



Gambar 2.13 James Martin RAD Model (James, 1991)

1. *Requirements Planning Phase*

Pada tahap ini dilakukan perancangan persyaratan yang dibutuhkan dalam pengembangan perangkat lunak. Perancangan tersebut berisi ketika *stakeholder* setuju dengan perencanaan yang telah dibuat.

2. *User Design Phase*

Pada tahap ini dilakukan desain terhadap perangkat lunak oleh pengembangan dengan melibatkan pengguna. Tahap ini memungkinkan pengguna untuk memahami, memodifikasi, dan menyetujui desain yang telah dibuat. Tahap ini melalui proses perulangan hingga pengguna merasa sudah memenuhi kebutuhannya.

3. *Construction Phase*

Pada tahap ini dilakukan konstruksi perangkat lunak yang dilakukan berdasarkan tahap *user design*. Pada tahap ini pengguna masih dilibatkan untuk memantau proses pengembangan perangkat lunak. Hal ini dimaksudkan agar pengguna dapat memberi masukan ketika terjadi ketidaksesuaian antara konstruksi dengan kebutuhan pengguna atau meminta peningkatan fitur terhadap perangkat lunak.

4. *Cutover Phase*

Tahap ini merupakan tahap terakhir dalam RAD yang dapat dilakukan setelah pengguna menyetujui desain dan konstruksi perangkat lunak. Pada tahap ini dilakukan implementasi sistem serta pengujian. *Cutover* juga memungkinkan

migrasi sistem serta pelatihan terhadap pengguna agar pengguna dapat memaksimalkan perangkat lunak yang sudah dikembangkan.

2.16 Black-Box Testing (Functional Testing)

Black-Box Testing merupakan sebuah metode pengujian perangkat lunak dengan tujuan untuk menguji fungsi dari sebuah perangkat lunak tanpa memperhatikan struktur internal dari perangkat yang diuji. Metode ini dapat digunakan dalam setiap tahapan pengujian seperti *unit testing*, *integration testing*, *system testing*, dan *acceptance testing* (Patton, 2001).

Dalam melakukan pengujian dengan pengujian menggunakan metode ini, pengetahuan terhadap kode-kode di dalam aplikasi, struktur aplikasi, dan pengetahuan pemrograman tidak dibutuhkan. Penguji diharuskan untuk sadar tentang apa saja yang harus dapat dilakukan oleh perangkat lunak yang dikembangkan, bukan bagaimana perangkat lunak tersebut melakukan tugas-tugasnya (Patton, 2001).

2.17 Hipotesis

Mengacu pada uraian-uraian sub bab sebelumnya, seperti perumusan masalah, tinjauan pustaka, dan landasan teori, maka dapat dikemukakan hipotesis sebagai berikut:

1. Pengembangan aplikasi manajemen *flow* dapat dilakukan.
2. Aplikasi manajemen *flow* dapat diimplementasikan ke dalam Jaringan *Software Defined Network*.

BAB III

BAHAN DAN METODE PENELITIAN

Pada bab ini dijelaskan mengenai kebutuhan alat dan bahan serta perangkat lunak pendukung untuk melakukan pengembangan aplikasi manajemen *flow* berbasis Android pada *Software Defined Network*. Selanjutnya dijelaskan juga mengenai metode penelitian dan skenario sistem pengujian.

3.1 Bahan

Perangkat lunak *opensource* dan *freeware* gunakan diantaranya adalah

1. Mininet, sebagai alat simulasi jaringan SDN.
2. Oracle VM VirtualBox, sebagai *virtual machine* untuk menjalankan Mininet.
3. Wireshark, sebagai alat untuk menganalisa trafik komunikasi paket data pada jaringan.
4. Android Studio, digunakan untuk mengembangkan aplikasi berbasis android.
5. Postman, digunakan untuk melakukan *request url* API.
6. OpenDaylight versi Berrylium, digunakan sebagai SDN *controller*.

3.2 Peralatan

Adapun peralatan yang digunakan dalam penelitian di antaranya adalah

1. Laptop sebagai *Controller* dengan spesifikasi:
 - CPU : 3.1GHz dual-core Intel Core i5
 - Harddisk : 128GB SSD
 - RAM : 8GB LPDDR3
 - OS : MacOS High Sierra
2. PC sebagai Host sebanyak 4 unit dengan spesifikasi:
 - CPU : Intel Celeron
 - RAM : 10 GB

- OS : Windows 10

3. Android Smartphone sebanyak 1 unit dengan spesifikasi:

- CPU : Qualcomm Snapdragon 625
- RAM : 64GB
- OS : Android Oreo 8.0

4. Raspberry Pi 3 Model B sebanyak 7 unit dengan spesifikasi:

- CPU : 1.2GHz quad-core ARM
- Memory : 1 GB LPDDR2-900 SDRAM
- USB Port : 4
- Network : 10/1000 Mbps Ethernet, 802.11n Wireless LAN

5. USB to Ethernet Converter sebanyak 16 unit

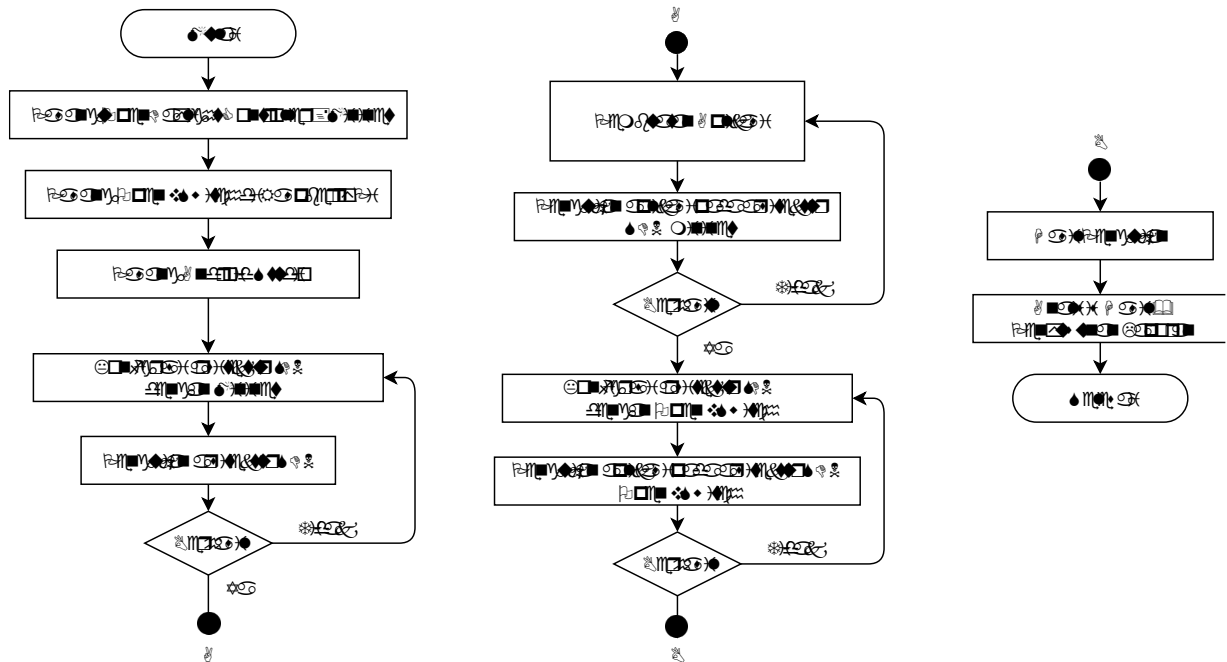
6. Switch sebanyak 1 unit

7. Access Point sebanyak 1 unit

8. Kabel LAN sebanyak 18 unit

3.3 Tahapan Penelitian

Metode penelitian yang dilakukan penulis dalam penelitian dapat dilihat pada *flow chart* berikut:



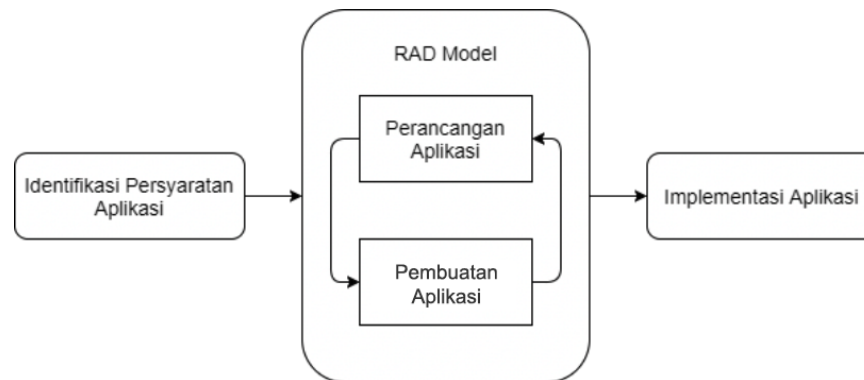
Gambar 3.1 Diagram Alir Tahapan Penelitian

1. Tahap Instalasi.

Tahap pertama yang dilakukan adalah menginstal semua *software* yang dibutuhkan. Adapun *software* yang diinstall pada tahap ini adalah OpenDaylight *controller*, Mininet VM, dan Open vSwitch yang akan dipasang pada Raspberry Pi.

2. Tahap Pengembangan Aplikasi.

Pada tahap ini dilakukan identifikasi kebutuhan aplikasi, perancangan aplikasi, kemudian pembuatan aplikasi. Pengembangan dilakukan dengan menggunakan metode RAD. RAD memungkinkan pengembangan aplikasi yang cepat, mudah, dan dapat mencapai kepuasan pengguna yang baik. Pada Gambar 3.2 dapat dilihat fase yang dijalankan dalam metode RAD.



Gambar 3.2 Fase Model RAD

Berdasarkan pada Gambar 3.2 tersebut fase yang dilalui selama proses pengembangan aplikasi adalah sebagai berikut:

a. Identifikasi Persyaratan Aplikasi

Proses identifikasi persyaratan aplikasi memiliki tujuan untuk mendapatkan kebutuhan yang diperlukan dalam pengembangan aplikasi. Identifikasi persyaratan dilakukan dengan cara berdiskusi dengan dosen pembimbing untuk memperoleh tujuan pembuatan aplikasi, kebutuhan aplikasi (fungsional dan non-fungsional), fitur yang akan dibangun, dan desain antarmuka dari aplikasi yang akan dikembangkan. Identifikasi diperlukan untuk menjadi dasar dalam perancangan desain RAD.

b. Perancangan Aplikasi

Perancangan aplikasi didasarkan pada hasil identifikasi persyaratan aplikasi. Proses perancangan aplikasi dilakukan dengan melihat kebutuhan non-fungsional yang dimiliki oleh aplikasi agar mendapatkan rancangan yang baik. Jika terdapat kekurangan dalam rancangan, maka dilakukan perbaikan pada aplikasi. Hal ini dilakukan hingga rancangan dirasa sesuai. Perancangan desain RAD yang dilakukan terdiri atas perancangan model dan perancangan antarmuka aplikasi. Perancangan basis data tidak dibutuhkan karena di dalam aplikasi, sistem basis

data yang digunakan hanya bersifat penyimpanan konfigurasi penghubungan aplikasi menuju OpenDaylight dan bersifat tempat penyimpanan sementara.

- Perancangan model

Perancangan model ditunjang dengan beberapa model diagram UMLM. UML memuat tipe diagram yang sudah terstandarisasi untuk digunakan mendeskripsikan dan memetakan perangkat lunak atau desain sistem dan struktur basis data. Diagram UML yang digunakan adalah *use case diagram* dan *activity diagram*.

- Perancangan antarmuka

Perancangan antarmuka dilakukan untuk membentuk sebuah panduan antarmuka aplikasi yang akan dikembangkan. Perancangan antarmuka dilakukan dengan menggunakan aplikasi fluidui secara online.

c. Pembuatan Aplikasi

Pada tahap ini dilakukan proses konstruksi aplikasi berdasarkan persyaratan aplikasi dan rancangan yang telah dibangun sebelumnya. Pada tahap ini, proses translasi fitur yang terdapat dari hasil identifikasi menjadi kode yang kemudian membentuk sebuah aplikasi. Proses pengembangan aplikasi sepenuhnya berdasarkan pada rancangan yang telah dibangun.

d. Implementasi Aplikasi

Pada saat aplikasi sudah mencakup keseluruhan dari fitur yang dibutuhkan dan desain antarmuka yang dibangun, selanjutnya dilakukan pemasangan aplikasi pada perangkat yang disiapkan untuk selanjutnya digunakan dan dilakukan pengujian.

3. Tahap Persiapan Pengujian Aplikasi.

Pada tahap ini dilakukan persiapan pengujian aplikasi. Persiapan dilakukan dengan membuat arsitektur SDN pada perangkat Raspberry Pi. Dimana pada arsitektur perangkat tersebut, aplikasi dilakukan pengujian.

4. Tahap Pengujian Aplikasi.

Pengujian aplikasi didasarkan pada *black-box testing* atau *functional testing*. Pengujian dilakukan dengan tujuan untuk mengetahui apakah aplikasi yang dibangun sudah sesuai dengan fitur yang ada pada hasil identifikasi persyaratan atau belum. Pengujian algoritma dan arsitektur aplikasi tidak dilakukan karena bukan merupakan batasan dalam melakukan *functional testing*.

5. Tahap Analisis.

Analisis merupakan tahapan paling akhir dalam penelitian ini. Pada tahap ini hasil pengujian yang telah dilakukan pada tahap sebelumnya akan dikumpulkan, diurai, dibedakan dan dipilah untuk selanjutnya digolongkan dan dikelompokkan berdasarkan kriteria tertentu kemudian dibuat suatu kesimpulan dari hasil yang didapat.

3.4 Instalasi OpenDaylight

Instalasi OpenDaylight dilakukan dengan cara melakukan *download* OpenDaylight dari *official website*-nya melalui perintah

```
# wget https://nexus.opendaylight.org/content/groups/public/org/
opendaylight/integration/distribution-karaf/0.4.0-Beryllium/
distribution-karaf-0.4.0-Beryllium.tar.gz
```

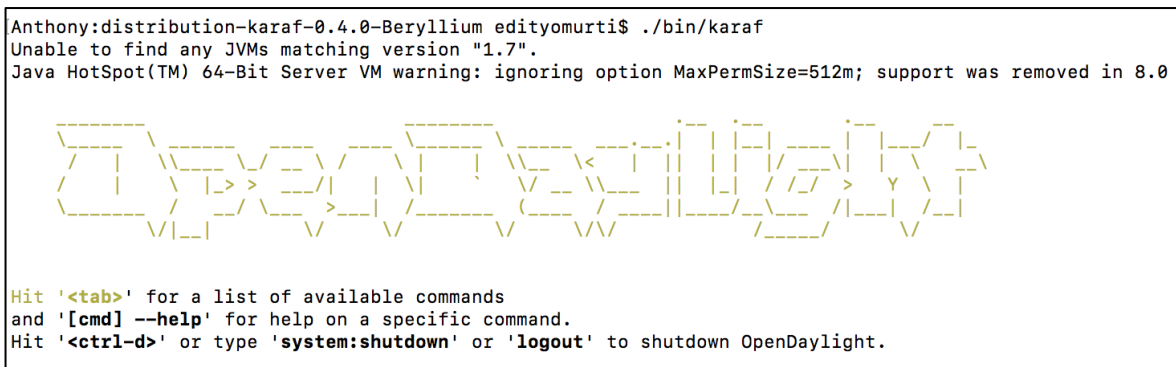
Setelah *download* selesai dilakukan, langkah selanjutnya adalah melakukan ekstrak file tersebut dengan perintah berikut

```
# tar -xvf distribution-karaf-0.4.0-Beryllium.tar.gz
```

Setelah ekstraksi selesai, maka selanjutnya adalah membuka *folder* tersebut kemudian menjalankan OpenDaylight dengan perintah berikut

```
# cd distribution-karaf-0.4.0-Beryllium.tar.gz
# ./bin/karaf
```

Jika OpenDaylight berhasil dijalankan maka terminal akan menampilkan tampilan seperti pada Gambar 3.3.



```
Anthony:distribution-karaf-0.4.0-Beryllium edityomurti$ ./bin/karaf
Unable to find any JVMs matching version "1.7".
Java HotSpot(TM) 64-Bit Server VM warning: ignoring option MaxPermSize=512m; support was removed in 8.0

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or type 'system:shutdown' or 'logout' to shutdown OpenDaylight.
```

Gambar 3.3 Tampilan awal OpenDaylight pada terminal

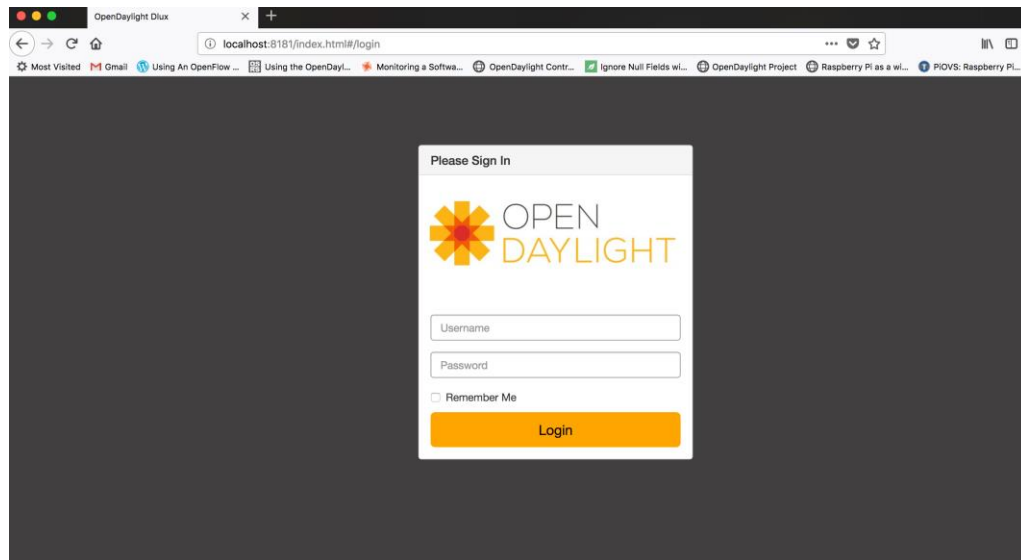
Setelah OpenDaylight berhasil dijalankan maka selanjutnya menginstall fitur-fitur pendukung agar OpenDaylight dapat berjalan baik. Fitur-fitur tersebut antara lain:

- odl-restconf : Memberikan akses kepada RESTCONF API
- odl-l2switch-switch : Memberikan fungsionalitas jaringan seperti Ethernet switch
- odl-mdsal-apidocs : Memberikan akses ke Yang API
- odl-dlux-all : Mengaktifkan GUI OpenDaylight

Instalasi fitur-fitur tersebut dilakukan dengan perintah sebagai berikut

```
opendaylight-user@root> feature:install odl-restconf odl-l2switch-all
odl-mdsal-apidocs odl-dlux-all
```

Setelah fitur-fitur tersebut selesai diinstall maka GUI Opendaylight dapat diakses melalui *web browser* dengan alamat `http://localhost:8181/index.html/`. Secara *default*, *username* dan *password* untuk masuk ke dalam GUI OpenDaylight adalah “admin”.

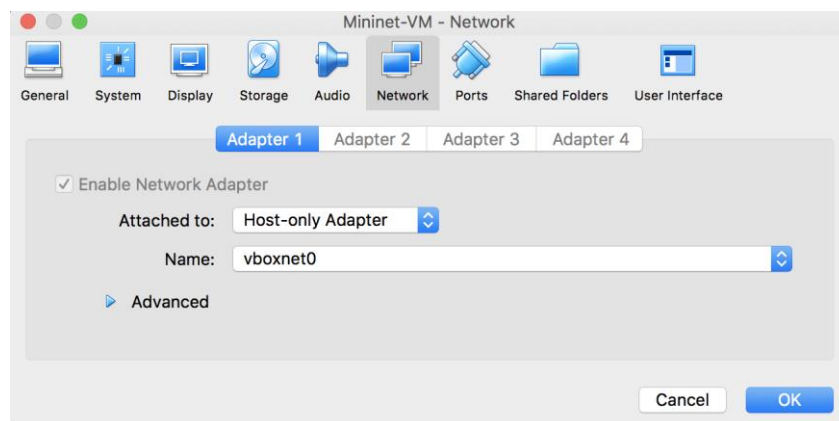


Gambar 3.4 Tampilan awal OpenDaylight GUI melalui web browser.

3.5 Menjalankan Mininet dan Simulasi Topologi Jaringan SDN

3.5.1. Menjalankan Mininet pada VirtualBox

Mininet yang digunakan adalah Mininet VM dimana mininet tersebut berjalan pada Ubuntu Server 14.04 LTS yang terinstall di dalam VirtualBox dengan MAC OS sebagai host. Konfigurasi adapter jaringan pada Mininet di dalam VirtualBox dikonfigurasi dengan *Host-only Adapter* seperti pada Gambar 3.5.



Gambar 3.5 Konfigurasi jaringan pada Mininet di dalam VirtualBox

Pada konfigurasi jaringan tersebut MAC OS memiliki alamat IP pada interface *vboxnet0* yaitu 192.168.56.1 sedangkan Mininet-VM memiliki alamat IP 192.168.56.101. Secara *default*, *username* dan *password* untuk dapat masuk ke dalam Mininet adalah “mininet”.

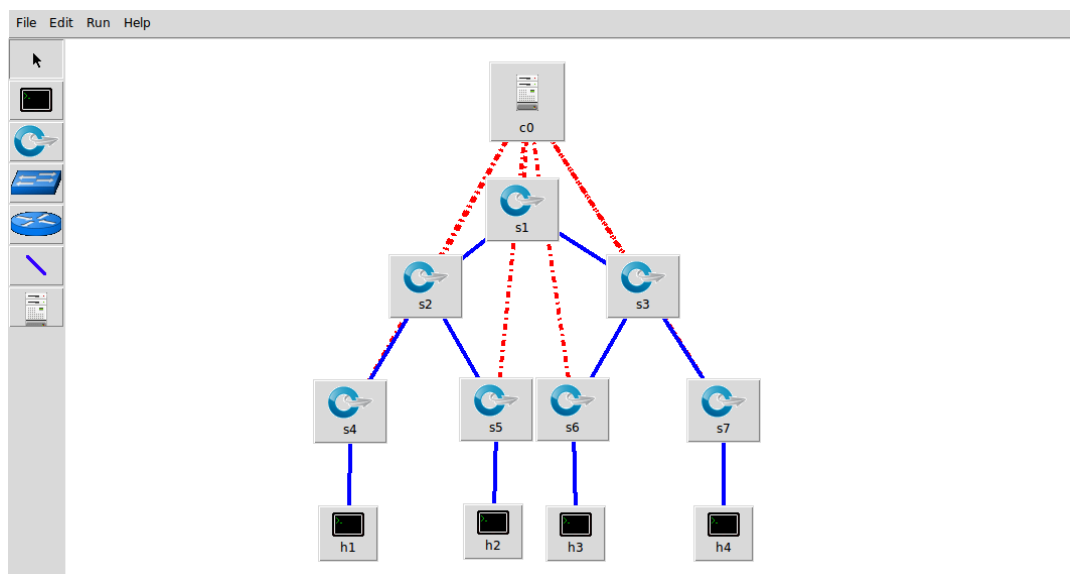
3.5.2. Membuat topologi dengan MiniEdit

MiniEdit merupakan tools yang terdapat pada Mininet yang berfungsi untuk membuat dan mengkonfigurasi topologi secara GUI. Untuk menggunakan *tools* ini dari Mininet-VM diperlukan akses SSH dari *host* MAC OS menuju Mininet-VM dan mengaktifkan X11 *forwarding*.

```
edityomurti$ ssh -X mininet@192.168.1.101
```

Selanjutnya membuka MiniEdit dan kemudian dilakukan konfigurasi topologi sesuai dengan skenario topologi.

```
root@mininet-vm:/home/mininet# mininet/mininet/examples/miniedit.py
```



Gambar 3.6 Tampilan simulasi topologi pada MiniEdit

3.6 Konfigurasi Open vSwitch

Sebelum melakukan konfigurasi OVS (Open vSwitch) pada perangkat Raspberry Pi 3, terlebih dahulu perlu melakukan installasi paket OVS pada Ubuntu 16.04 yang telah diinstall pada Raspberry Pi dengan menggunakan perintah :

```
# sudo apt install openvswitch-switch openvswitch-common bridge-  
utils
```

Setelah paket OVS telah terinstall pada Ubuntu, langkah berikutnya adalah melakukan konfigurasi OVS dengan menggunakan perintah :

```
# ovs-vsctl add-br <bridge_name>  
# ovs-vsctl add-port <bridge_name> eth1  
# ovs-vsctl add-port <bridge_name> eth2  
# ovs-vsctl add-port <bridge_name> eth3  
# ovs-vsctl add-port <bridge_name> eth4  
# ifconfig eth1 0 up  
# ifconfig eth2 0 up  
# ifconfig eth3 0 up  
# ifconfig eth4 0 up
```

Perintah tersebut digunakan untuk membuat *ovs bridge* dengan nama *bridge* tersebut menyesuaikan dengan *switch* yang digunakan. Kemudian menambahkan *port* eth1 sampai dengan eth4 pada *bridge* dan menghilangkan konfigurasi ip pada setiap *port*. Selanjutnya melakukan konfigurasi pada *OpenFlow* yang digunakan dan *socket* yang digunakan oleh *controller* serta konfigurasi tambahan lainnya dengan menggunakan perintah :

```
# ovs-vsctl set bridge <bridge_name> stp_enable=true  
# ovs-vsctl set bridge <bridge_name> other-config:disable-in-  
band=true  
# ovs-vsctl set bridge <bridge_name> protocol=OpenFlow13  
# ovs-vsctl set-controller <bridge_name> tcp:192.168.1.100:6653
```

Perintah tersebut digunakan untuk konfigurasi pada OVS agar mengaktifkan STP pada *openvswitch* dan menonaktifkan jaringan *in-band* pada *switch*. Kemudian OVS

dikonfigurasi agar menggunakan protokol *OpenFlow* versi 1.3 dan *controller* yang digunakan menggunakan IP 192.168.1.100 dan *port* 6653.

3.7 Identifikasi Persyaratan Aplikasi

Identifikasi persyaratan aplikasi dilakukan dengan menghasilkan kebutuhan fungsional dan kebutuhan non-fungsional sebagai berikut

1. Kebutuhan Fungsional

Kebutuhan fungsional mencakup proses atau layanan yang harus disediakan oleh sistem. Berikut adalah kebutuhan fungsional pada sistem:

- Sistem dapat menampilkan daftar perangkat yang terhubung pada OpenDaylight.
- Sistem dapat menampilkan daftar *flow*.
- Sistem dapat menampilkan informasi detail suatu *flow*.
- Sistem dapat melakukan penambahan *flow* dengan beberapa parameter berdasarkan pencocokan Port, alamat MAC, dan alamat IP.
- Sistem dapat melakukan pengubahan parameter pada suatu *flow*.
- Sistem dapat melakukan penghapusan *flow*.

2. Kebutuhan Non-Fungsional

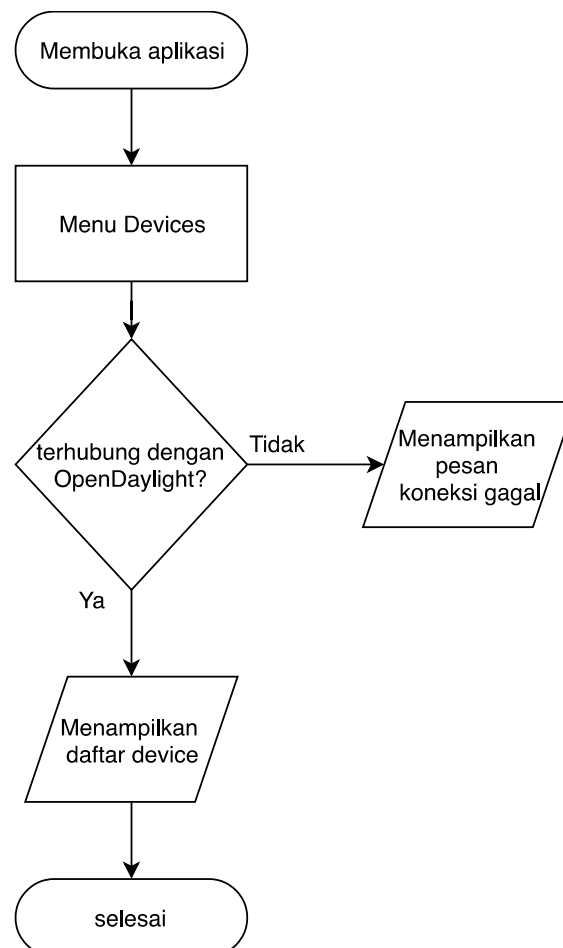
Kebutuhan non-fungsional mencakup properti perilaku yang harus disediakan oleh sistem. Berikut adalah kebutuhan non-fungsional pada sistem:

- Sistem memiliki antarmuka yang baik sehingga memudahkan pengguna untuk menggunakan aplikasi.
- Sistem menggunakan bahasa dan gambar yang mudah dimengerti oleh pengguna.
- Sistem dapat dijalankan pada Android versi Lollipop atau setelahnya.

Identifikasi persyaratan aplikasi dilakukan dan menghasilkan tiga buah alur fitur pada aplikasi yang terbagi menjadi tiga buah menu yaitu *Devices*, *Flow Table*, dan *Settings*.

1. Fitur *Devices*

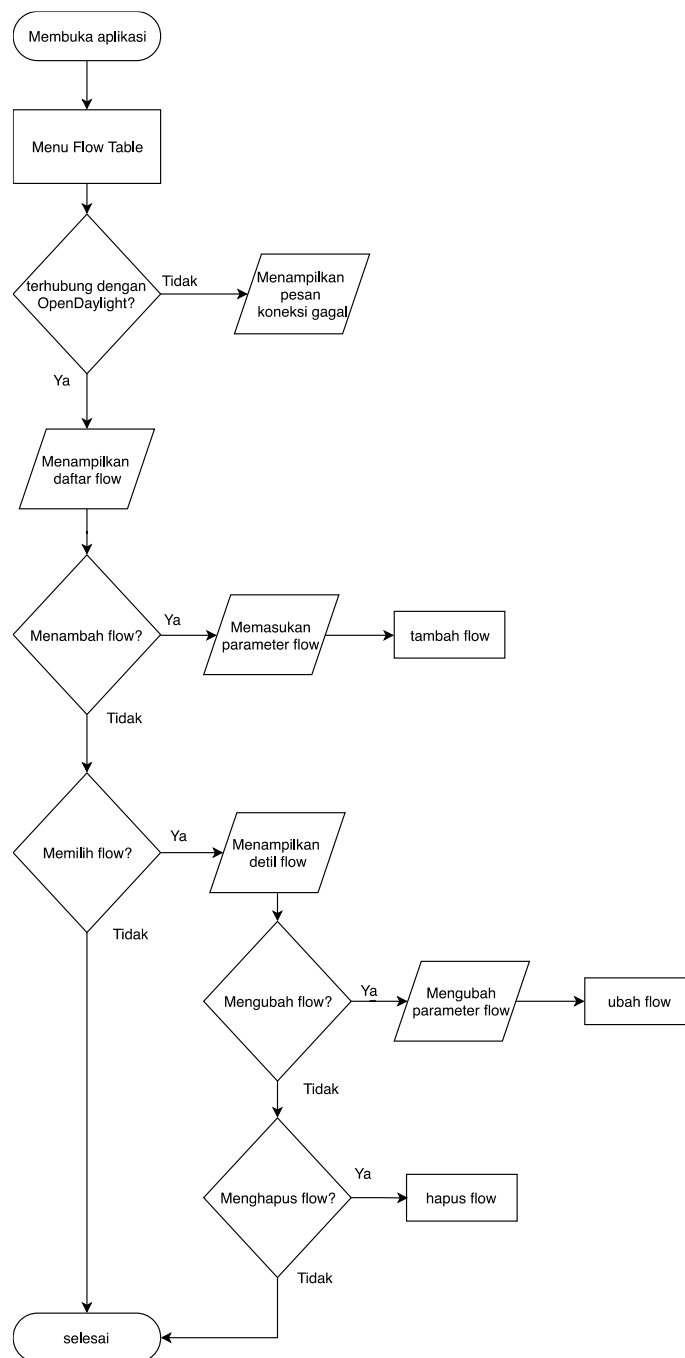
Pada menu ini terdapat daftar *device* (*switch* dan *host*) yang terhubung pada *controller* yang ada pada jaringan. Terdapat informasi-informasi *device* berupa nama *device*, jumlah *node* yang terhubung, dan rincian *node* (nama *switch* dan/atau MAC Address *host*). Pada sisi kanan atas (*Action Menu*) juga terdapat *button refresh* yang berfungsi untuk melakukan *update* kondisi data terbaru.



Gambar 3.7 Diagram alir Fitur Devices

2. Fitur *Flow Table*

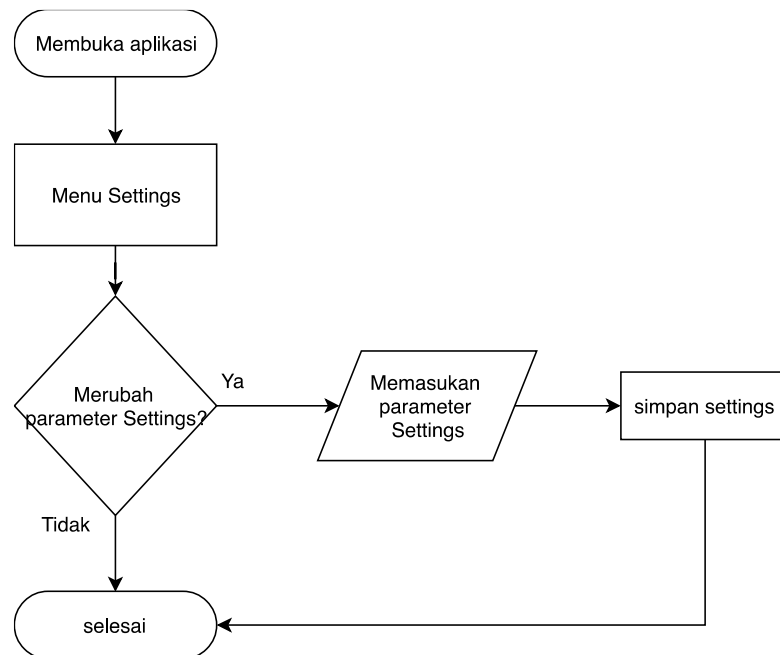
Pada menu ini terdapat daftar *flow* pada setiap *switch (device)* yang ada dalam jaringan. Terdapat 3 *action menu* yang berupa *button* penambahan *flow*, penampilan statistik *flow*, dan juga *refresh*. Pada setiap *item* dalam daftar *flow* terdapat informasi-informasi *flow* berupa nama *device*, *flow ID*, nilai prioritas, dan tipe *flow*.



Gambar 3.8 Diagram alir Fitur Flow Tables

3. Fitur *Settings*

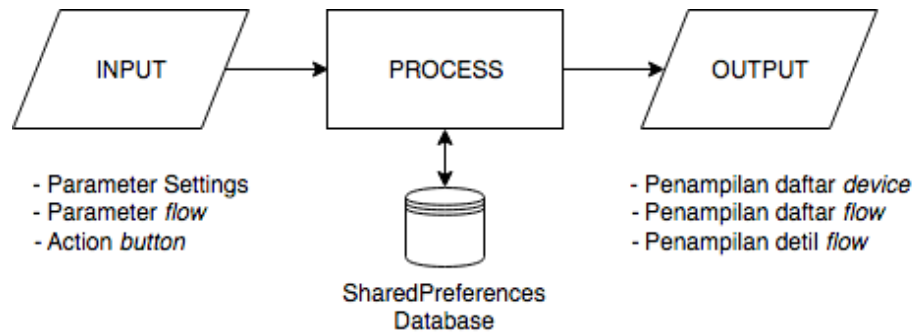
Pada menu ini terdapat konfigurasi aplikasi untuk dapat berkomunikasi dengan OpenDaylight *controller* berupa input alamat IP, *port*, *username*, dan *password*. Data konfigurasi ini sangat penting agar aplikasi dapat terhubung dengan *controller*.



Gambar 3.9 Diagram alir Fitur Settings

3.8 Desain RAD

Aplikasi bekerja dengan tiga tahapan utama yaitu masukan (*input*), proses, dan keluaran (*output*). Masukan dapat berupa pemilihan menu, pemilihan action pada *button*, dan juga data yang dimasukkan oleh pengguna seperti pada data parameter *flow* dan parameter *settings*. Aplikasi mengolah data masukan untuk disimpan dan digunakan dalam proses HTTP *request* menuju API OpenDaylight. *Output* dari aplikasi adalah berupa penampilan informasi dari *action* yang dilakukan pengguna. Alur proses pada aplikasi dapat dilihat pada Gambar 3.10.



Gambar 3.10 Alur proses kerja aplikasi

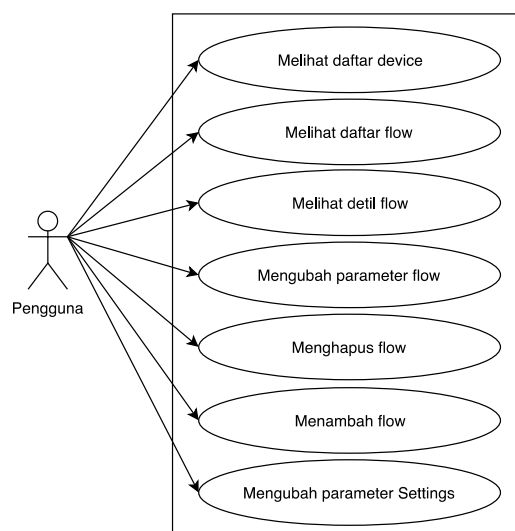
Dalam pengembangan aplikasi, dilakukan dua tahap perancangan yaitu perancangan model dan perancangan antarmuka.

3.8.1. Perancangan Model

Perancangan model dilakukan dengan menggunakan diagram UML Diagram UML yang digunakan adalah *use case diagram*, dan *activity diagram*.

1. Use Case Diagram

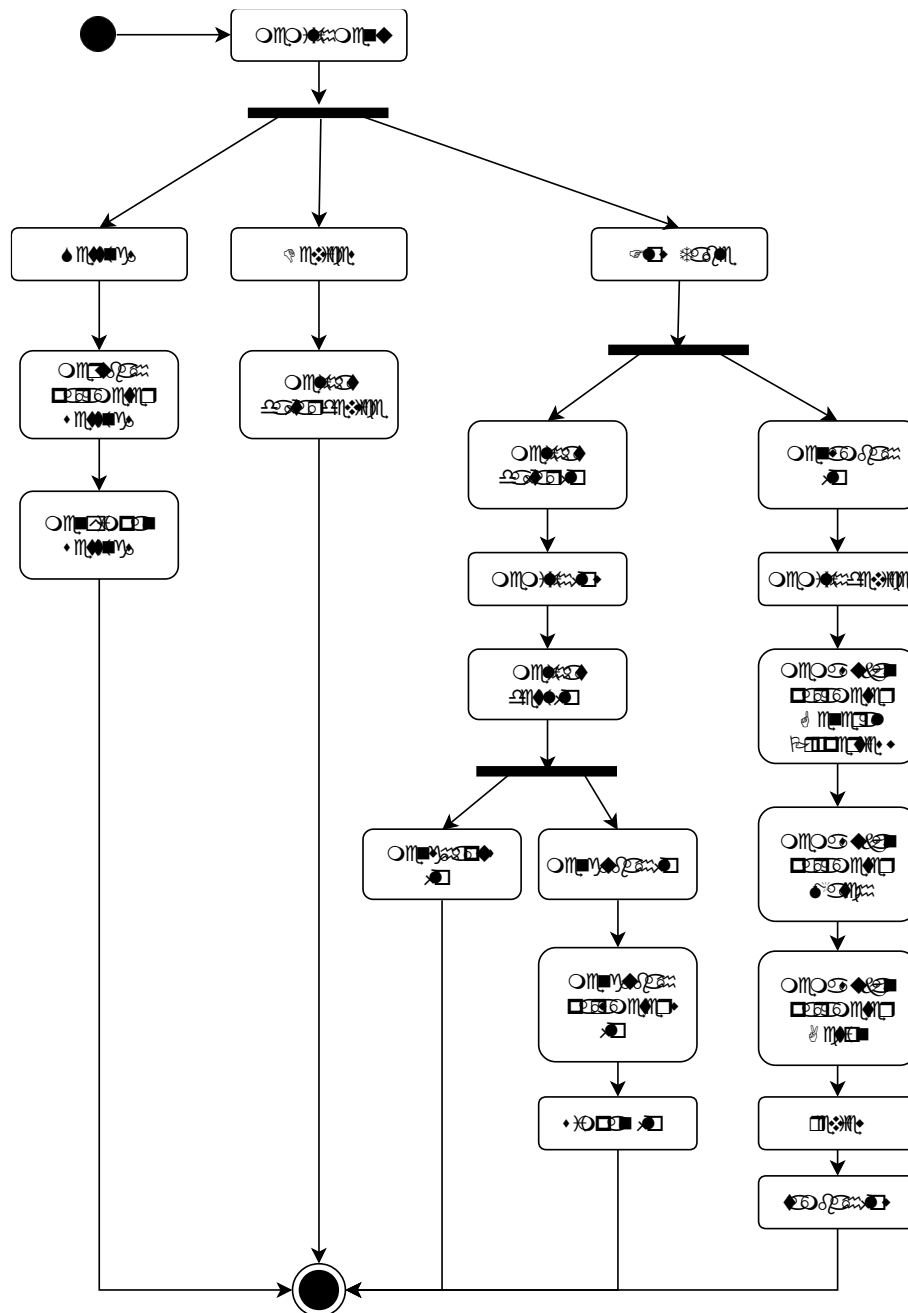
Diagram ini merupakan diagram UML yang digunakan untuk menggambarkan skenario fungsi dari aplikasi dikembangkan. Tujuan utama penggunaan *use case diagram* adalah untuk membuat visualisasi dari fungsi yang dibuat dalam suatu aplikasi. Gambar 3.11 merupakan *use case diagram* dari aplikasi yang dikembangkan.



Gambar 3.11 Use Case Diagram Aplikasi manajemen flow

2. Activity Diagram

Activity Diagram digunakan untuk merepresentasikan aliran kerja dan langkah-langkah yang harus ditempuh. Dalam perancangan *activity diagram*, keseluruhan *activity* yang terdapat di dalam aplikasi ditampilkan sesuai dengan tiga buah fitur yang terdapat pada aplikasi. Pada Gambar 3.12 terlihat *activity diagram* aplikasi.



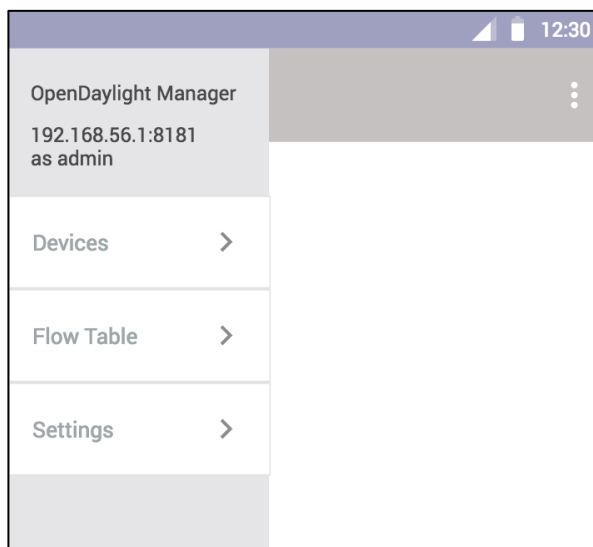
Gambar 3.12 Activity Diagram aplikasi

3.8.2. Perancangan Antarmuka

Perancangan antarmuka untuk aplikasi OpenDaylight Manager Mobile (aplikasi manajemen *flow* berbasis Android pada jaringan SDN menggunakan OpenDaylight *Controller*). Antarmuka yang dirancang merupakan bentuk kasar dari antarmuka yang nantinya digunakan di dalam aplikasi. Rancangan ini hanya untuk menunjukkan kebutuhan apa saja yang harus ditampilkan di dalam tampilan aplikasi.

1. Daftar Menu

Aplikasi manajemen *flow* memiliki 3 menu utama yaitu Devices, Flow Table, dan Settings. Daftar menu-menu tersebut terletak di menu samping (*Navigation Drawer*). Pada tampilan ini juga terdapat informasi *username* dan alamat IP yang sedang terhubung dengan aplikasi. Rancangan tampilan dari daftar menu tersebut tampak seperti pada Gambar 3.13.

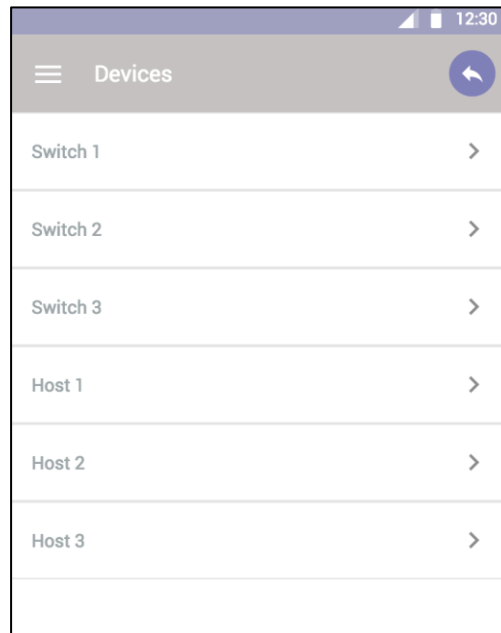


Gambar 3.13 Rancangan Antarmuka Daftar Menu

2. Menu Devices

Pada menu ini terdapat daftar *device* (*switch* dan *host*) yang terhubung pada *controller* yang ada pada jaringan. Terdapat informasi-informasi *device* berupa nama *device*, jumlah *node* yang terhubung, dan rincian *node* (nama

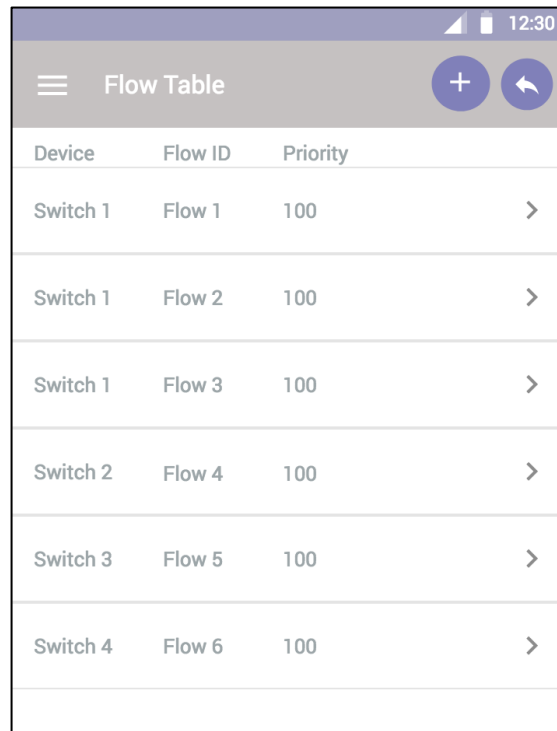
switch dan/atau MAC Address *host*). Pada sisi kanan atas (*Action Menu*) juga terdapat *button refresh* yang berfungsi untuk melakukan *update* kondisi data terbaru.



Gambar 3.14 Rancangan Antarmuka Menu Devices

3. Menu Flow Table

Pada menu ini terdapat daftar *flow* pada setiap *switch (device)* yang ada dalam jaringan. Terdapat 3 *action menu* yang berupa *button* penambahan *flow*, penampilan statistik *flow*, dan juga *refresh*. Pada setiap *item* dalam daftar *flow* terdapat informasi-informasi *flow* berupa nama *device*, *flow ID*, nilai prioritas, dan tipe *flow*.

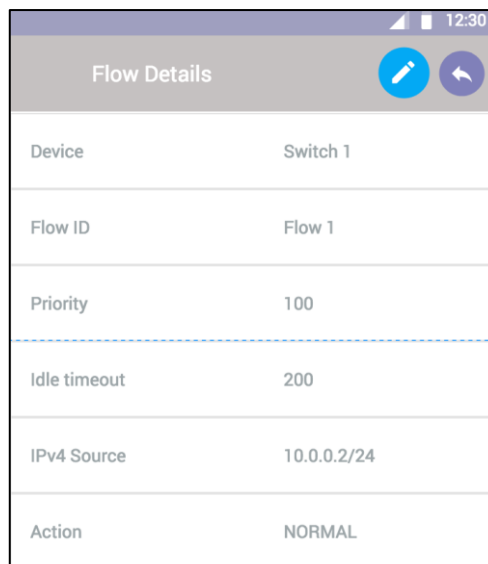


Device	Flow ID	Priority	
Switch 1	Flow 1	100	>
Switch 1	Flow 2	100	>
Switch 1	Flow 3	100	>
Switch 2	Flow 4	100	>
Switch 3	Flow 5	100	>
Switch 4	Flow 6	100	>

Gambar 3.15 Rancangan Antarmuka Menu Flow Table

4. Flow Details

Pada tampilan ini terdapat informasi rincian *flow* dan juga terdapat *action menu* berupa penghapusan *flow*.

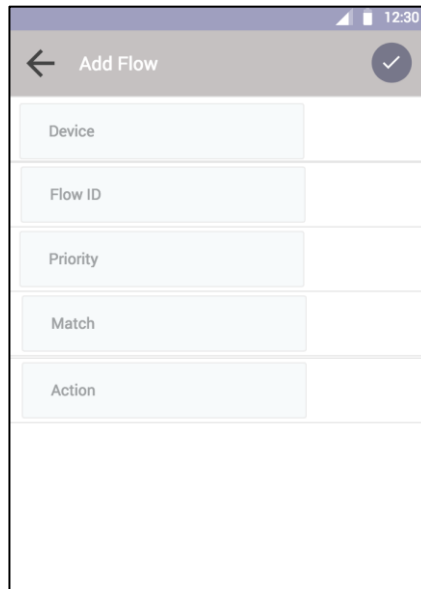


Flow Details	
Device	Switch 1
Flow ID	Flow 1
Priority	100
Idle timeout	200
IPv4 Source	10.0.0.2/24
Action	NORMAL

Gambar 3.16 Rancangan Antarmuka Flow Details

5. Add Flow

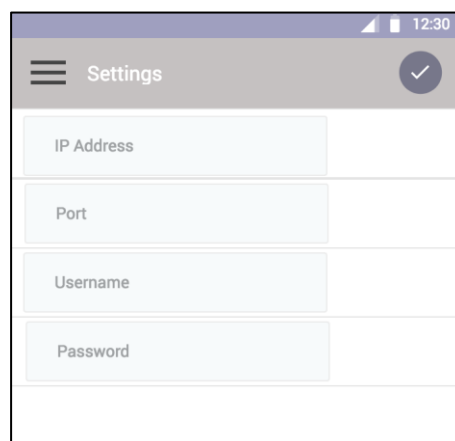
Penambahan *flow* terbagi menjadi 4 tampilan bertahap yaitu tampilan pemilihan device (*Select Device*), input properti umum (*General Properties*), pencocokan (*Match*), aksi (*Action*), dan Review.



Gambar 3.17 Rancangan Antarmuka *Add Flow*

6. Settings

Pada tampilan ini terdapat konfigurasi aplikasi untuk dapat berkomunikasi dengan *controller* berupa input alamat IP, *port*, *username*, dan *password*. Data konfigurasi ini sangat penting agar aplikasi dapat terhubung dengan *controller*.



Gambar 3.18 Rancangan Antarmuka Menu *Settings*

3.9 Pengujian Aplikasi

3.9.1. *Black-Box Testing (Functional Testing)*

Pengujian aplikasi pada penelitian ini menggunakan metode *Black-Box Testing*. Metode ini menitik beratkan pada pengujian fungsi-fungsi yang dikembangkan menjadi fitur pada aplikasi. Dengan menggunakan metode ini, proses pengujian tidak memperhatikan proses yang terjadi di dalam aplikasi. Proses pengujian aplikasi dimulai dengan menentukan fitur yang akan diuji untuk selanjutnya dilakukan pengujian dengan alat (*smartphone*) dan pada jaringan SDN yang telah disiapkan. Proses pengujian dilakukan dengan membuat *test script* sebagai acuan pengujian. *Test Script* berisi fungsi apa saja yang akan diuji dan langkah untuk mengujinya. *Test Script* dapat dilihat pada Tabel 3.1.

Tabel 3.1 Test Script Pengujian Aplikasi

No.	Skenario	Kasus Pengujian	Kebutuhan	Langkah	
1	Fitur <i>Settings</i>	Mengubah parameter <i>Settings</i>	Aplikasi telah dibuka	1	Memilih menu <i>Settings</i>
				2	Memasukan alamat IP
				3	Memasukan nomor Port
				4	Memasukan <i>username</i>
				5	Memasukan <i>password</i>
				6	Menekan tombol <i>save</i>
2	Fitur <i>Devices</i>	Melihat daftar <i>device</i>	Aplikasi telah dibuka	1	Memilih menu <i>Devices</i>
			Parameter <i>Settings</i> telah ditentukan dengan benar	2	Memeriksa apakah daftar <i>device</i> telah sesuai
3	Fitur <i>Device</i>	Memuat ulang daftar <i>device</i>	Menu <i>Devices</i> telah dipilih	1	Memeriksa apakah daftar <i>devices</i> telah terlihat
				2	Menekan tombol <i>refresh</i>
				3	Memeriksa apakah Daftar <i>device</i> telah dimuat ulang
4	Fitur <i>Flow Table</i>	Menambah <i>flow</i> dengan pencocokan berdasarkan Port	Aplikasi telah dibuka	1	Memilih menu <i>Flow Table</i>
			Parameter <i>Settings</i> telah ditentukan dengan benar	2	Menekan tombol <i>tambah</i>
				3	Menentukan <i>device</i>
			<i>Device</i> telah ditentukan	4	Menekan tombol <i>next</i>
				5	Memasukan Flow ID
				6	Memasukan <i>priority</i>
			Parameter <i>general properties</i> telah ditentukan	7	Menekan tombol <i>next</i>
				8	Menekan tombol <i>Add Match</i>
				9	Memilih item <i>in-port</i>

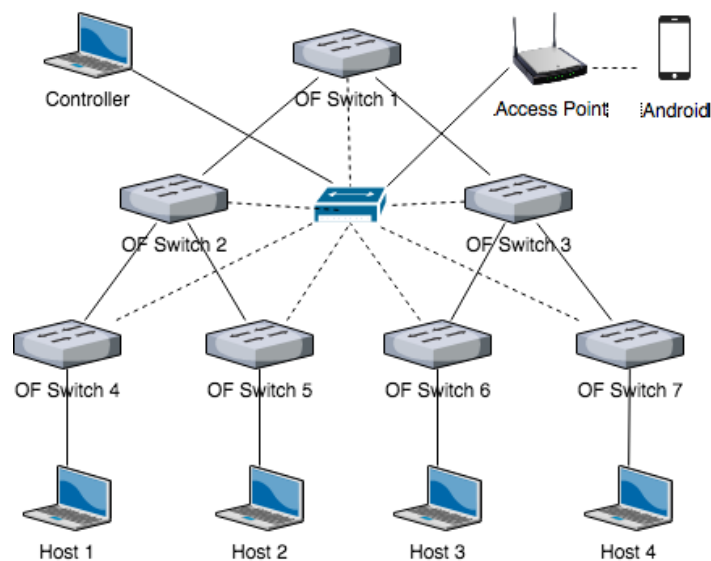
				10	Menentukan <i>in-port</i>
			Parameter <i>match</i> telah ditentukan	11	Menekan tombol <i>next</i>
				12	Menekan tombol <i>Add Action</i>
				13	Memilih item <i>normal</i>
			Parameter <i>action</i> telah ditentukan	14	Menekan tombol <i>review</i>
				15	Menekan tombol <i>finish</i>
5	Fitur <i>Flow Table</i>	Menambah <i>flow</i> dengan pencocokan berdasarkan alamat MAC	Aplikasi telah dibuka	1	Memilih menu <i>Flow Table</i>
			Parameter <i>Settings</i> telah ditentukan dengan benar	2	Menekan tombol <i>tambah</i>
				3	Menentukan <i>device</i>
			<i>Device</i> telah ditentukan	4	Menekan tombol <i>next</i>
				5	Memasukan Flow ID
				6	Memasukan <i>priority</i>
			Parameter <i>general properties</i> telah ditentukan	7	Menekan tombol <i>next</i>
				8	Menekan tombol <i>Add Match</i>
				9	Memilih item <i>MAC Address source</i>
				10	Memasukan alamat MAC asal
				11	Menekan tombol <i>Add Match</i>
				12	Memilih item <i>MAC Address Destination</i>
				13	Memasukan alamat MAC tujuan
			Parameter <i>match</i> telah ditentukan	14	Menekan tombol <i>next</i>
				15	Menekan tombol <i>Add Action</i>
				16	Memilih item <i>flood</i>
			Parameter <i>action</i> telah ditentukan	17	Menekan tombol <i>review</i>
				18	Menekan tombol <i>finish</i>
6	Fitur <i>Flow Table</i>		Aplikasi telah dibuka	1	Memilih menu <i>Flow Table</i>

		Menambah <i>flow</i> dengan pencocokan berdasarkan alamat IP	Parameter <i>Settings</i> telah ditentukan dengan benar	2	Menekan tombol <i>tambah</i>
				3	Menentukan <i>device</i>
			<i>Device</i> telah ditentukan	4	Menekan tombol <i>next</i>
				5	Memasukan Flow ID
				6	Memasukan <i>priority</i>
			Parameter <i>general properties</i> telah ditentukan	7	Menekan tombol <i>next</i>
				8	Menekan tombol <i>Add Match</i>
				9	Memilih item <i>IPv4 Source</i>
				10	Memasukan alamat IP asal
				11	Menekan tombol <i>Add Match</i>
				12	Memilih item <i>IPv4 Destination</i>
				13	Memasukan alamat IP tujuan
				14	Menekan tombol <i>Add Match</i>
				15	Memilih item <i>EtherType</i>
				16	Memasukan ethertype “0x800”
			Parameter <i>match</i> telah ditentukan	17	Menekan tombol <i>next</i>
				18	Menekan tombol <i>Add Action</i>
				19	Memilih item <i>flood</i>
			Parameter <i>action</i> telah ditentukan	20	Menekan tombol <i>review</i>
				21	Menekan tombol <i>finish</i>
7	Fitur <i>Flow Table</i>	Melihat daftar <i>flow</i>	Aplikasi telah dibuka	1	Memilih menu <i>Flow Table</i>
			Parameter <i>Settings</i> telah ditentukan dengan benar	2	Memeriksa apakah daftar <i>flow</i> telah sesuai
8	Fitur <i>Flow Table</i>	Melihat detil <i>flow</i>	Menu <i>Flow Table</i> telah dipilih	1	Memilih salah satu <i>flow</i> dari daftar <i>flow</i>

				2	Memeriksa apakah detil <i>flow</i> telah sesuai
9	Fitur <i>Flow Table</i>	Mengubah <i>flow</i>	<i>Details Flow</i> telah dibuka	1	Menekan tombol <i>edit</i>
				2	Mengubah parameter <i>flow</i>
			Telah dilakukan perubahan paramter <i>flow</i>	3	Menekan tombol <i>next</i>
				4	Menekan tombol <i>finish</i>
				5	Memeriksa apakah parameter <i>flow</i> telah sesuai dengan perubahan yang dilakukan
10	Fitur <i>Flow Table</i>	Menghapus <i>flow</i>	<i>Details Flow</i> telah dibuka	1	Menekan tombol <i>delete</i>
				2	Menekan teombol <i>yes</i> pada kotak <i>dialog</i> konfirmasi
				3	Memeriksa apakah <i>flow</i> telah hilang dari daftar <i>flow</i>

3.9.2. Skenario Uji Coba Penambahan Flow

Skenario dibuat dengan tujuan melakukan uji coba fungsionalitas utama aplikasi dalam melakukan manajemen *flow* dimana fungsionalitas tersebut mencakup penambahan *flow*, pengubahan *flow*, dan penghapusan *flow*. Uji coba keberhasilan penambahan *flow* dilakukan dengan beberapa skenario berdasarkan pencocokan paket (*match*) di *layer* yang berbeda yaitu berdasarkan Port, alamat MAC, dan Alamat IP. Pada setiap skenario dilakukan pembuatan *flow* yang berbeda-beda kemudian dilakukan pengujian apakah *flow* berhasil diterapkan pada topologi jaringan dengan tujuan dapat mengirimkan paket ICMP antar dua *host* yang sebelumnya tidak dapat dilakukan. Pengubahan dan penghapusan *flow* juga dilakukan pada setiap skenario. Gambaran dari topologi yang digunakan pada yang digunakan dapat dilihat pada Gambar 3.19.



Gambar 3.19 Topologi pengujian

Berikut adalah skenario yang berbeda-beda untuk menguji keberhasilan pembuatan *flow*:

1. Skenario Pembuatan *flow* berdasarkan pencocokan Port

Pada skenario ini dilakukan pembuatan *flow* dengan pencocokan (*match*) berdasarkan port pada *switch*. Action yang diterapkan pada skenario ini adalah action berupa OUTPUT PORT yang mengirimkan paket menuju *port* lain.

2. Skenario Pencocokan *flow* berdasarkan alamat MAC

Pada skenario ini dilakukan pembuatan *flow* dengan pencocokan (*match*) berdasarkan alamat MAC asal dan alamat MAC tujuan. Action yang diterapkan pada skenario ini adalah FLOOD yang mengirim paket ke *port* selain *port* paket berasal. Pada skenario ini juga ditambahkan *flow* ARP (*Address Resolution Protocol*).

3. Skenario Pencocokan *flow* berdasarkan alamat IP

Pada skenario ini dilakukan pembuatan *flow* dengan pencocokan (*match*) berdasarkan alamat IP asal dan alamat IP tujuan. Action yang diterapkan pada skenario ini adalah NORMAL yang memproses paket ke menggunakan pemrosesan *switch* tradisional. Pada skenario ini juga ditambahkan *flow* ARP.

BAB IV

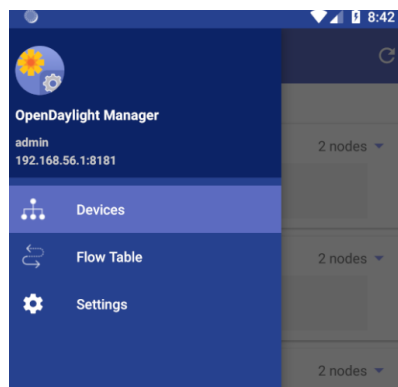
HASIL PENELITIAN DAN PEMBAHASAN

4.1 Pengembangan Aplikasi

Aplikasi yang dikembangkan pada penelitian ini merupakan sebuah aplikasi yang dikembangkan untuk sistem operasi Android. Pengembangan dilakukan menggunakan bahasa Kotlin dan XML pada IDE Android Studio sesuai dengan standar Google dalam melakukan pengembangan aplikasi Android. Pengembangan antarmuka aplikasi juga mengikuti aturan Google menggunakan pola *material design*.

4.1.1. Daftar Menu

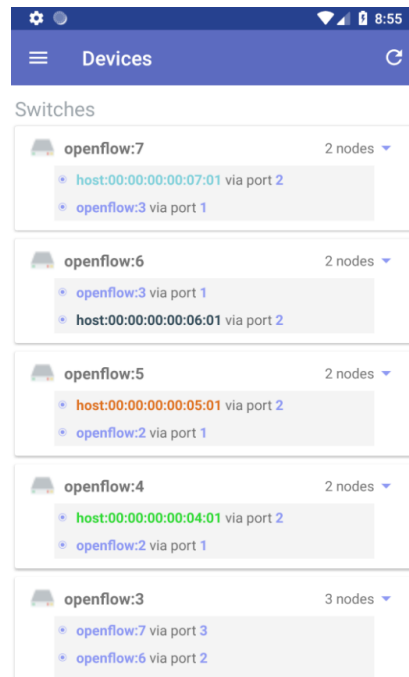
Aplikasi manajemen *flow* memiliki 3 menu utama yaitu Devices, Flow Table, dan Settings. Daftar menu-menu tersebut terletak di menu samping (*Navigation Drawer*). Pada tampilan ini juga terdapat informasi *username* dan alamat IP yang sedang terhubung dengan aplikasi. Tampilan dari menu utama tersebut tampak seperti pada Gambar 4.1



Gambar 4.1 Antarmuka Daftar Menu

4.1.2. Fitur *Devices*

Pada fitur ini terdapat informasi daftar *device* yang terhubung dengan OpenDaylight. *Device* terbagi menjadi 2 jenis yaitu *Switches* dan *Host*. Informasi mengenai *device* yang ditampilkan pada setiap *item Switches* adalah berupa jumlah *nodes*, nama *device* lain yang terhubung pada *node*, dan port yang menghubungkan *Switches* dengan *device* lain.

Gambar 4.2 Antarmuka Fitur *Devices*

4.1.3. Fitur *Flow Table*

Pada fitur ini terdapat informasi daftar *flow* yang ada pada setiap *Switches*. Informasi yang ditampilkan pada setiap *flow* dalam daftar ini berupa nama *flow*, ID *flow*, prioritas *flow*, dan tipe *flow*. Terdapat *action menu* pada bagian kanan atas berupa aksi untuk melakukan penambahan *flow*, penampilan statistik *flow*, dan muat ulang daftar *flow*.

Device	Flow ID	Priority	Type
openflow:3	#UFTABLE*0-600	2	operational
openflow:3	#UFTABLE*0-566	0	operational
openflow:2	#UFTABLE*0-590	100	operational
openflow:2	#UFTABLE*0-608	2	operational
openflow:2	#UFTABLE*0-607	2	operational
openflow:2	#UFTABLE*0-609	2	operational
openflow:2	#UFTABLE*0-591	0	operational
openflow:1	#UFTABLE*0-594	100	operational
openflow:1	#UFTABLE*0-611	2	operational
openflow:1	#UFTABLE*0-610	2	operational
openflow:1	#UFTABLE*0-595	0	operational

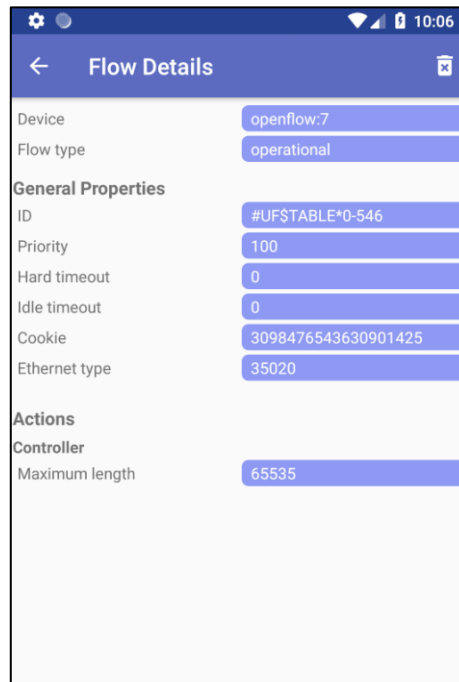
Gambar 4.3 Antarmuka *Flow Table*

Device	Flow ID	Priority	Type	packet count	byte count
openflow:3	#UFTABLE*0-600	2	operational	0	0
openflow:3	#UFTABLE*0-566	0	operational	0	0
openflow:2	#UFTABLE*0-590	100	operational	6	510
openflow:2	#UFTABLE*0-608	2	operational	0	0
openflow:2	#UFTABLE*0-607	2	operational	2	196
openflow:2	#UFTABLE*0-609	2	operational	0	0
openflow:2	#UFTABLE*0-591	0	operational	0	0
openflow:1	#UFTABLE*0-594	100	operational	4	340

Gambar 4.4 Antarmuka *Flow Table* dengan statistik

4.1.4. *Flow Details*

Pada tampilan ini terdapat informasi rincian *flow* dan juga terdapat *action menu* berupa penghapusan *flow*.

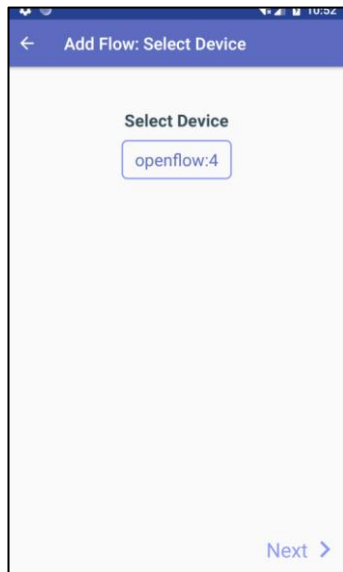


Flow Details	
Device	openflow:7
Flow type	operational
General Properties	
ID	#UFS\$TABLE*0-546
Priority	100
Hard timeout	0
Idle timeout	0
Cookie	3098476543630901425
Ethernet type	35020
Actions	
Controller	
Maximum length	65535

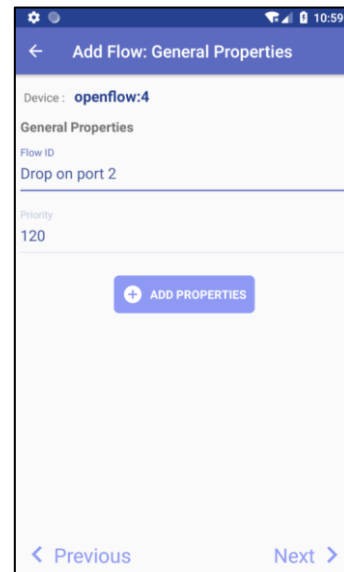
Gambar 4.5 Antarmuka *Flow Details*

4.1.5. *Add Flow*

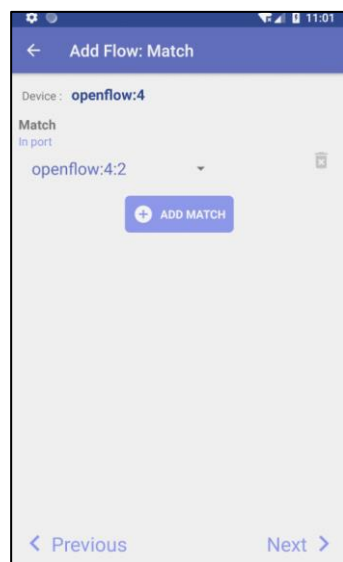
Penambahan *flow* terbagi menjadi 4 tampilan bertahap yaitu tampilan pemilihan *device* (*Select Device*), input properti umum (*General Properties*), pencocokan (*Match*), aksi (*Action*), dan Review.



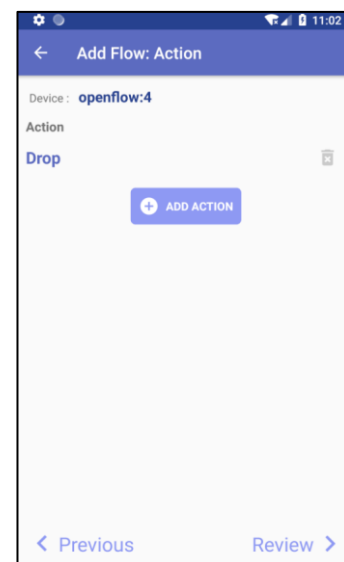
Gambar 4.6. Antarmuka *Select Device*



Gambar 4.7 Antarmuka *General Properties*



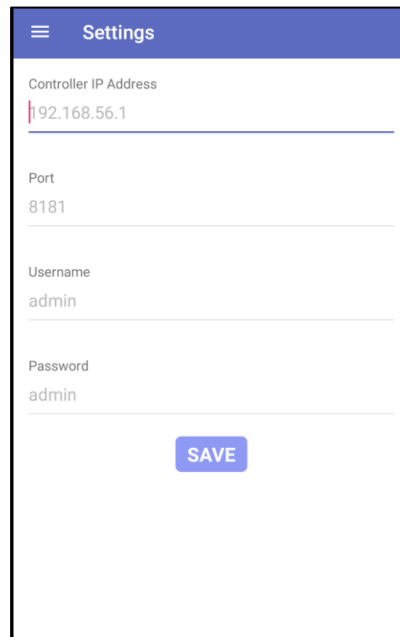
Gambar 4.8. Antarmuka *Match*



Gambar 4.9 Antarmuka *Action*

4.1.6. Fitur *Settings*

Pada fitur ini dilakukan pemasukan data parameter konfigurasi aplikasi agar dapat terhubung dengan OpenDaylight. Parameter pada konfigurasi ini berupa alamat IP, nomor *Port*, *username*, dan *password*.

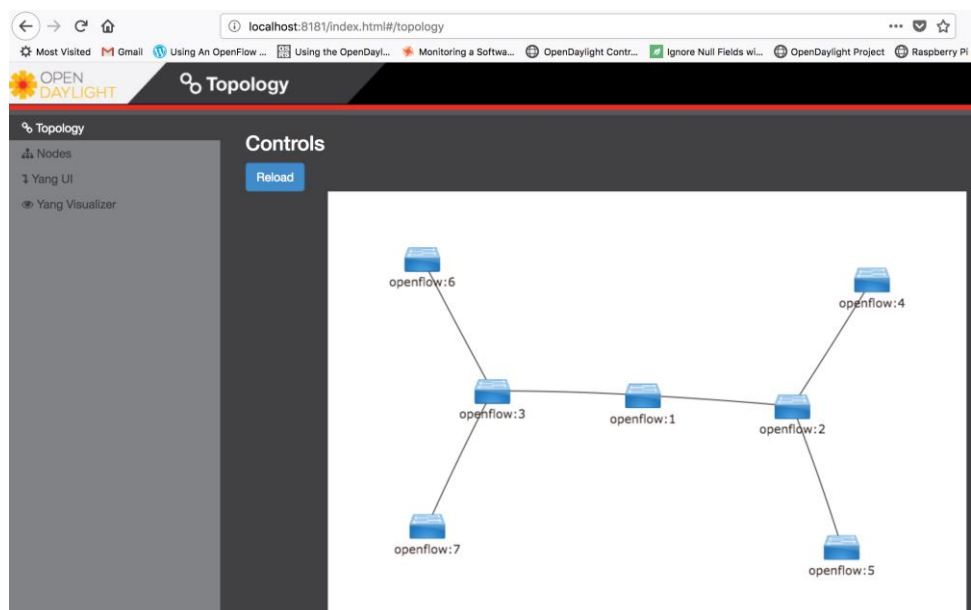


Gambar 4.10 Antarmuka *Settings*

4.2 Pengujian Penerapan SDN pada Perangkat Raspberry Pi

Sebelum melakukan pengujian berdasarkan skenario yang sudah dibuat, terlebih dahulu dilakukan pengujian penerapan SDN pada perangkat Raspberry Pi. Pengujian ini dilakukan dengan melihat kesesuaian penampilan informasi topologi *switch* pada OpenDaylight GUI.

Fitur OpenDaylight GUI menampilkan informasi seperti pada Gambar 4.11.



Gambar 4.11 Tampilan OpenDaylight GUI

4.3 Pengaksesan data dari OpenDaylight

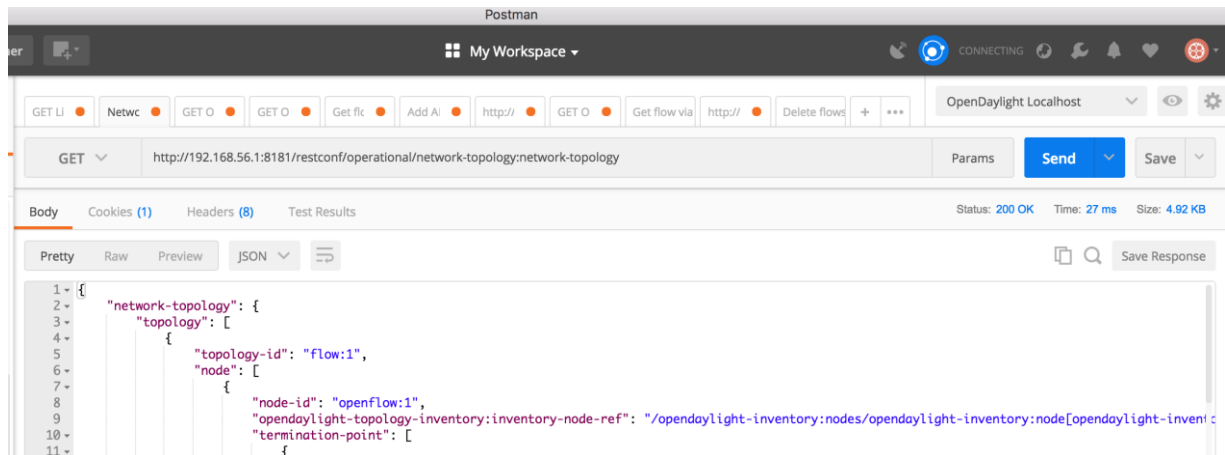
Pengaksesan data dilakukan melalui komunikasi HTTP dengan melakukan HTTP *request* menuju *endpoint* REST-API OpenDaylight dan mendapatkan *response* berupa file dengan tipe JSON kemudian dikonversi menjadi objek kelas pada file Kotlin. Komunikasi data untuk melakukan *request* dan mendapatkan *response* dilakukan menggunakan *library android* bernama Retrofit versi 2.0. Dalam setiap *request* HTTP memerlukan autentikasi dengan tipe *BasicAuth* dengan parameter *username* dan *password* yang telah dimasukkan di menu Settings.

Setiap *request* dilakukan dengan mengakses file *RestAdapter.kt* dimana pada file ini diinisialisasikan variabel nilai paramater *BasicAuth* dan basis alamat URL yaitu `http://<IP_CONTROLLER:PORT>/restconf/` kemudian diikuti alamat *endpoint* pada setiap data.

4.3.1. Data Topologi

Untuk melakukan uji coba pengambilan data berupa HTTP *request* dilakukan menggunakan Postman dalam memasukkan parameter-parameter seperti HTTP methods, *BasicAuth*, alamat *endpoint*, serta parameter *endpoint*. Apabila pengambilan data telah sesuai maka HTTP *request* diimplementasikan ke dalam aplikasi.

Pengaksesan data topologi dilakukan melalui HTTP *request* menuju endpoint “operational/network-topology:network-topology” dengan HTTP *methods* GET.

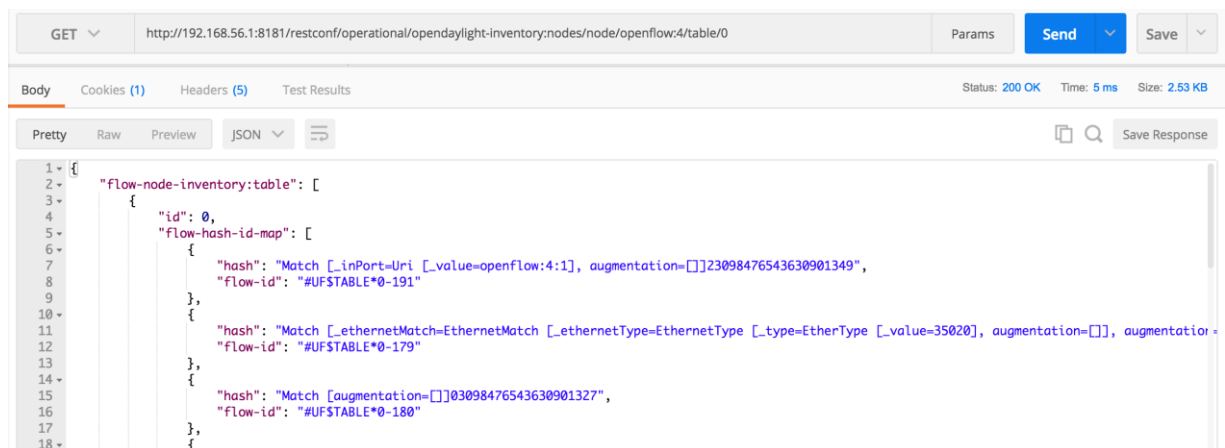


Gambar 4.12 Pengambilan data topologi melalui Postman

4.3.2. Data Flow Table

Pengaksesan data Flow Table dilakukan melalui HTTP *request* menuju beberapa *endpoint* yaitu:

- Inventory Nodes : “operational/opendaylight-inventory:nodes/”
- Flow Operational : “operational/opendaylight-inventory:nodes/node/{id}/table/{table_id}”
- Flow Config : "config/opendaylight-inventory:nodes/node/{id}/table/{table_id}"

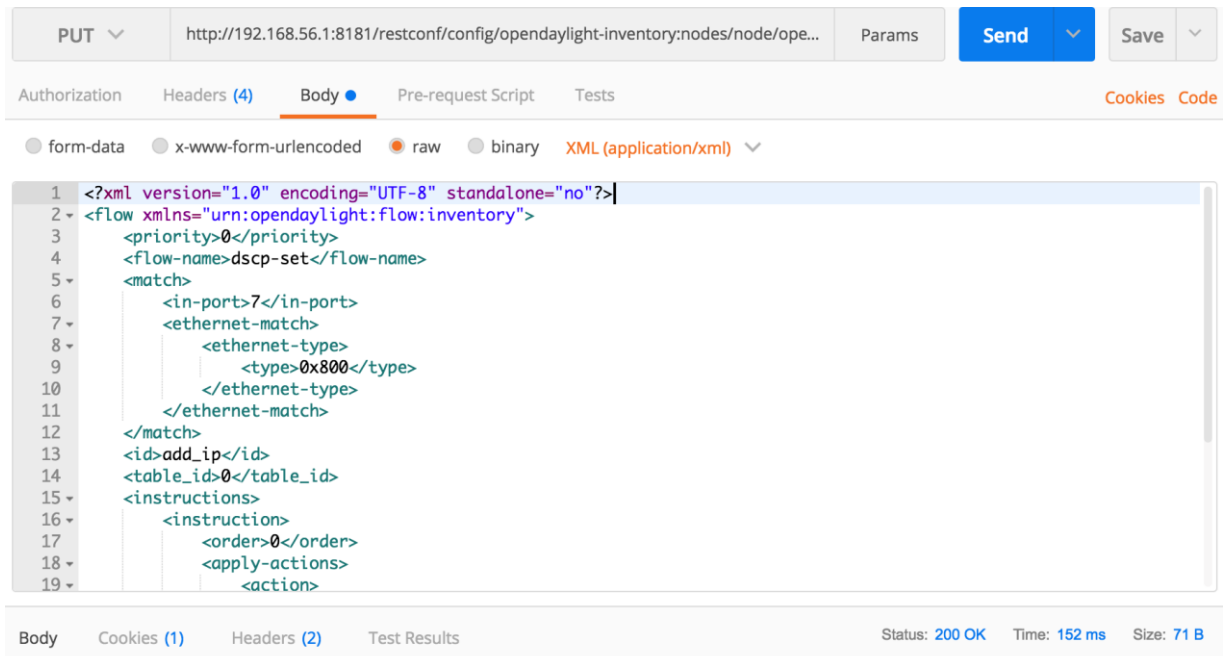


Gambar 4.13 Pengambilan data Flow Table melalui Postman

4.3.3. Penambahan data *flow*

Penambahan data *flow* dilakukan melalui HTTP *request* menuju *endpoint* “config/opendaylight-inventory:nodes/node/{node_id}/table/0/flow/{flow_id}” dengan HTTP

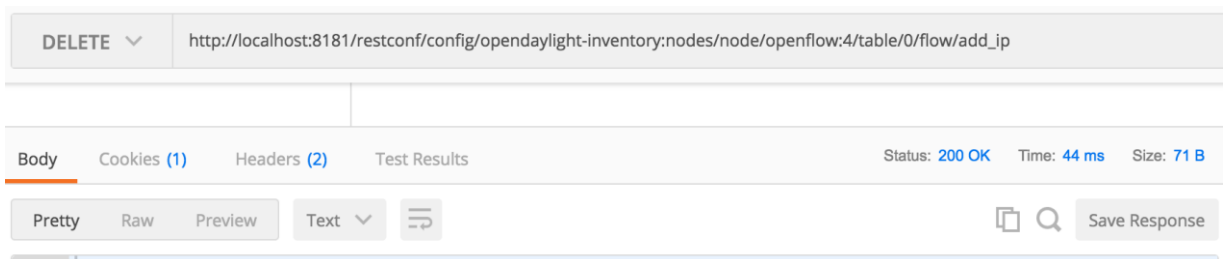
methods PUT. Pada *request* ini menggunakan HTTP *body* berupa file JSON dari kelas Flow yang akan ditambahkan.



Gambar 4.14 Penambahan data *flow* melalui Postman

4.3.4. Penghapusan data *flow*

Penghapusan data *flow* dilakukan melalui HTTP *request* menuju *endpoint* "config/opendaylight-inventory:nodes/node/{node_id}/table/0/flow/{flow_id}" dengan HTTP *methods* DELETE.



Gambar 4.15 Penghapusan *flow* melalui Postman

4.4 Pengujian Penambahan flow berdasarkan Pencocokan Port

Dalam tahap ini dilakukan pengujian pengiriman paket ICMP (ping) antara *host* H1 dengan H2 yang dilakukan sebelum dan setelah penambahan *flow*.

Hasil ping antara *host* H1 dan H2 sebelum dilakukan penambahan *flow* ditunjukkan pada Tabel 4.1. Pada hasil tersebut menunjukkan bahwa *host* H1 tidak dapat melakukan komunikasi ICMP.

Tabel 4.1 Hasil pengujian ping sebelum penambahan *flow* skenario 1

Pegujian Ping	H1 (10.0.0.1/24)	H2 (10.0.0.2/24)
H2 (10.0.0.2/24)	X	X
H1 (10.0.0.1/24)	X	X

Kemudian penambahan *flow* dari aplikasi dilakukan dengan konfigurasi *flow* seperti pada Tabel 4.2.

Tabel 4.2 Konfigurasi *flow* Skenario 1

No.	Node (Device)	ID	Priority	In-port	Normal
1	openflow:4	flow1	100	2	v
2	openflow:4	flow2	100	1	v
3	openflow:2	flow3	100	2	v
4	openflow:2	flow4	100	1	v
5	openflow:5	flow5	100	2	v
6	openflow:5	flow5	100	1	v

Pengujian ping antara *host* H1 dan H2 setelah dilakukan penambahan *flow* dari aplikasi ditunjukkan pada Tabel 4.3. Hasil ini menunjukkan bahwa *host* H1 dapat melakukan pengiriman paket ICMP menuju H2 dan menunjukkan bahwa penambahan *flow* telah berhasil dilakukan.

Tabel 4.3 Hasil pengujian ping setelah penambahan *flow* skenario 1

Pegujian Ping	H1 (10.0.0.1/24)	H2 (10.0.0.2/24)
H2 (10.0.0.2/24)	v	v
H1 (10.0.0.1/24)	v	v

Kemudian dilakukan pengujian penghapusan *flow* dari aplikasi pada setiap *flow* yang telah dibuat.

4.5 Pengujian Penambahan *flow* berdasarkan Pencocokan Alamat MAC

Dalam tahap ini dilakukan pengujian pengiriman paket ICMP (ping) antara *host* H1 dengan H2 yang dilakukan sebelum dan setelah penambahan *flow*.

Hasil ping antara *host* H1 dan H2 sebelum dilakukan penambahan *flow* ditunjukkan pada Tabel 4.4 . Pada hasil tersebut menunjukkan bahwa *host* H1 tidak dapat melakukan komunikasi ICMP.

Tabel 4.4 Hasil pengujian ping sebelum penambahan *flow* skenario 2

Pegujian Ping	H1 (10.0.0.1/24)	H2 (10.0.0.2/24)
H2 (10.0.0.2/24)	X	X
H1 (10.0.0.1/24)	X	X

Kemudian penambahan *flow* dari aplikasi dilakukan dengan konfigurasi *flow* seperti pada Tabel 4.5.

Tabel 4.5 Konfigurasi *flow* Skenario 2

No.	Node (Device)	ID	Priority	mac-source	mac-dest	Ether-type	flood	output- port
1	openflow:4	flow1	100	00:00:00:00:04:01	00:00:00:00:05:01	-	v	-
2	openflow:4	flow2	100	00:00:00:00:05:01	00:00:00:00:04:01	-	v	-
3	openflow:4	flow3	200	-	-	0x806	v	-
4	openflow:2	flow4	100	00:00:00:00:04:01	00:00:00:00:05:01	-	-	3
5	openflow:2	flow5	100	00:00:00:00:05:01	00:00:00:00:04:01	-	-	2
6	openflow:2	flow6	200	-	-	0x806	v	-
7	openflow:5	flow7	100	00:00:00:00:04:01	00:00:00:00:05:01	-	v	-
8	openflow:5	flow8	100	00:00:00:00:05:01	00:00:00:00:04:01	-	v	-
9	openflow:5	flow9	200	-	-	0x806	v	-

Pengujian ping antara *host* H1 dan H2 setelah dilakukan penambahan *flow* dari aplikasi ditunjukkan pada Tabel 4.6. Hasil ini menunjukkan bahwa *host* H1 dapat melakukan pengiriman paket ICMP menuju H2 dan menunjukkan bahwa penambahan *flow* telah berhasil dilakukan.

Tabel 4.6 Hasil pengujian ping setelah penambahan *flow* skenario 2

Pegujian Ping	H1 (10.0.0.1/24)	H2 (10.0.0.2/24)
H2 (10.0.0.2/24)	v	v
H1 (10.0.0.1/24)	v	v

Kemudian dilakukan pengujian penghapusan *flow* dari aplikasi pada setiap *flow* yang telah dibuat.

4.6 Pengujian Penambahan *flow* berdasarkan Pencocokan Alamat IP

Dalam tahap ini dilakukan pengujian pengiriman paket ICMP (ping) antara *host* H1 dengan H2 yang dilakukan sebelum dan setelah penambahan *flow*.

Hasil ping antara *host* H1 dan H2 sebelum dilakukan penambahan *flow* ditunjukkan pada Tabel 4.7 . Pada hasil tersebut menunjukkan bahwa *host* H1 tidak dapat melakukan komunikasi ICMP.

Tabel 4.7 Hasil pengujian ping sebelum penambahan *flow* skenario 3

Pegujian Ping	H1 (10.0.0.1/24)	H2 (10.0.0.2/24)
H2 (10.0.0.2/24)	X	X
H1 (10.0.0.1/24)	X	X

Kemudian penambahan *flow* dari aplikasi dilakukan dengan konfigurasi *flow* seperti pada Tabel 4.8.

Tabel 4.8 Konfigurasi *flow* Skenario 3

No.	Node (Device)	ID	Priority	ip-source	ip-dest	Ether-type	flood	output- port
1	openflow:4	flow1	100	10.0.0.1	10.0.0.2	0x800	v	-
2	openflow:4	flow2	100	10.0.0.2	10.0.0.1	0x800	v	-
3	openflow:4	flow3	200	-	-	0x806	v	-
4	openflow:2	flow4	100	10.0.0.1	10.0.0.2	0x800	-	3
5	openflow:2	flow5	100	10.0.0.2	10.0.0.1	0x800	-	2
6	openflow:2	flow6	200	-	-	0x806	v	-
7	openflow:5	flow7	100	10.0.0.1	10.0.0.2	0x800	v	-
8	openflow:5	flow8	100	10.0.0.2	10.0.0.1	0x800	v	-
9	openflow:5	flow9	200	-	-	0x806	v	-

Pengujian ping antara *host* H1 dan H2 setelah dilakukan penambahan *flow* dari aplikasi ditunjukkan pada Tabel 4.9. Hasil ini menunjukkan bahwa *host* H1 dapat melakukan pengiriman paket ICMP menuju H2 dan menunjukkan bahwa penambahan *flow* telah berhasil dilakukan.

Tabel 4.9 Tabel pengujian ping setelah penambahan *flow* skenario 3

Pegujian Ping	H1 (10.0.0.1/24)	H2 (10.0.0.2/24)
H2 (10.0.0.2/24)	v	v
H1 (10.0.0.1/24)	v	v

4.7 Black-Box Testing (*Functional Testing*)

Pengujian dengan menggunakan metode *black-box* didasarkan pada fitur-fitur yang dimiliki oleh aplikasi. Pengujian yang dilakukan disesuaikan dengan *test case* yang sudah dibuat oleh peneliti. Berdasarkan *test case* pada bab 3, didapatkan hasil pengujian bahwa semua fitur dan *case* dapat dijalankan sebagaimana mestinya. Hal ini menunjukkan bahwa fungsi yang dikembangkan berhasil berfungsi dengan baik. Hasil pengujian dapat dilihat pada Tabel 4.10.

Tabel 4.10 Hasil Pengujian Fungsional

No.	Skenario	Kasus Pengujian	Hasil
1	Fitur <i>Settings</i>	Mengubah parameter <i>Settings</i>	Berhasil
2	Fitur <i>Devices</i>	Melihat daftar <i>device</i>	Berhasil
3		Memuat ulang daftar <i>device</i>	Berhasil
4	Fitur <i>Flow Table</i>	Menambah <i>flow</i> dengan pencocokan berdasarkan Port	Berhasil
5		Menambah <i>flow</i> dengan pencocokan berdasarkan alamat MAC	Berhasil
6		Menambah <i>flow</i> dengan pencocokan berdasarkan alamat IP	Berhasil
7		Melihat daftar <i>flow</i>	Berhasil
8		Melihat detail <i>flow</i>	Berhasil
9		Mengubah <i>flow</i>	Berhasil
10		Menghapus <i>flow</i>	Berhasil

BAB V

PENUTUP

5.1 Kesimpulan

Berdasarkan data yang diperoleh serta analisis yang sudah dilakukan, maka dapat ditarik beberapa kesimpulan dari penelitian tersebut, diantaranya :

1. Telah dikembangkan aplikasi manajemen *flow* berbasis Android dengan memanfaatkan REST-API yang disediakan oleh *OpenDaylight Controller* .
2. Pengembangan aplikasi menghasilkan tiga buah fitur yaitu *Devices*, *Flow Table*, dan *Settings*. Dimana pada fitur-fitur tersebut dapat dilakukan fungsi utama aplikasi dalam melakukan manajemen *flow* yaitu melihat daftar *flow*, menambah *flow*, mengubah *flow*, dan menghapus *flow*.
3. Pengujian aplikasi menggunakan metode *Black-box testing* menunjukkan bahwa semua fungsi dan fitur pada aplikasi yang dikembangkan pada penelitian ini berfungsi sepenuhnya.

5.2 Saran

Pada pengembangan selanjutnya untuk memperbaiki proyek akhir ini dapat dilakukan beberapa pertimbangan yaitu :

1. Melakukan uji coba fungsionalitas aplikasi dari faktor lain, seperti diuji cobakan pada topologi lain, diuji cobakan pada Android dengan versi *operating system* yang lain, diuji cobakan pada versi OpenDaylight yang lain, diuji cobakan pada versi Open vSwitch dan OpenFlow yang lain.
2. Melakukan uji coba tingkat efisiensi dan keamanan kode aplikasi seperti menggunakan *Unit Testing* dan *Penetration Test*.
3. Menambahkan parameter Match dan Intruksi lain yang telah disediakan oleh REST-API OpenDaylight.
4. Menambahkan fungsi manajemen OpenDaylight dengan mengeksplorasi ketersediaan *endpoint* REST-API OpenDaylight yang lain.
5. Mengembangkan aplikasi manajemen *flow* yang berbasis versi *operating system* yang lain seperti iOS.

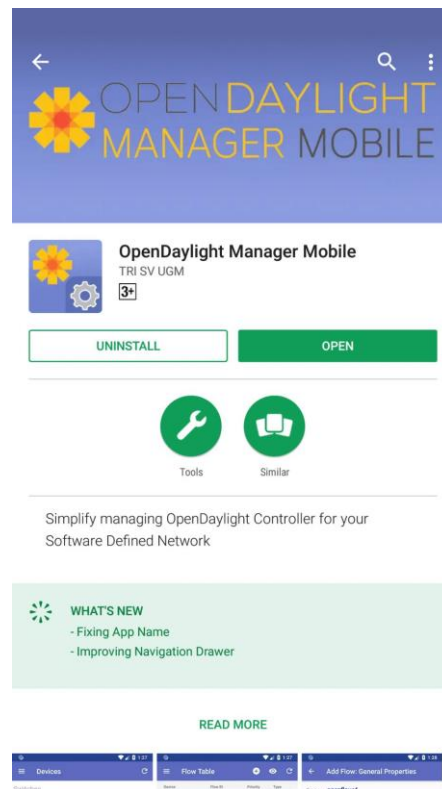
DAFTAR PUSTAKA

- Anam, Khoerul. 2017. *Analisis Performa Jaringan Software Defined Network Berdasarkan Penggunaan Cost Pada Protokol Ruting Open Shortest Path First*. CITEE 2017.
- Awaad, M.H, H, Krauss, H. D. Schmatz. 2005. *Advanced Praise for The Unified Modeling Language Reference Manual, Second Edition*, vol. 240, no. 3.
- Banjar, A., P. Papatwibul, R. Braun, B. Moulton. 2014. *Analysing the Performance of the OpenFlow Standard for Software-Defined Networking Using the OMNet++ Network Simulator*. *Computer Aided System Engineering (APCASE)*. pp. 31-37
- Boehm, B.W, H.W. Boehm et al. 1987. *A Spiral Model of Software Development and Enhancement*. *Computer* (Long. Beach. Calif)., vol. 21, no. May, pp. 61– 72
- Grgurevic, Ivan, Zvonko K, Anthony P. 2015. *Simulation Analysis of Characteristics and Application of Software-Defined Networks*. *Jurnal*. University of Zagreb, Zagreb, Croatia.
- Hikam, Muhammad. 2017. *Implementasi dan Analisis Kinerja Arsitektur Software Defined Network berbasis OpenDayLight Controller*. CITEE 2017.
- Hyojoon, Kim, Nick Feamster. 2013. *Improving Network Management with Software Defined Networking*. *Jurnal*: Georgia Institute of Technology, Georgia.
- IDC. 2017. *Smartphone OS Market Share, 2017 Q1*. Diakses pada 30 Juni 2018 halaman situs <http://www.idc.com/promo/smartphone-market-share/os>.
- Islam, Rashedul et al. 2010. *Mobile Application and Its Global Impact*. *International Journal of Engineering & Technology* vol. 10, no.6., pp. 72-78.
- Katardie, Rikie. 2014. *Prototipe Insfrastruktur Software Defined Network dengan Protokol OpenFlow menggunakan Ubuntu sebagai Kontroler*. *Jurnal DASI* vol. 15 No. 1 Maret 2014.

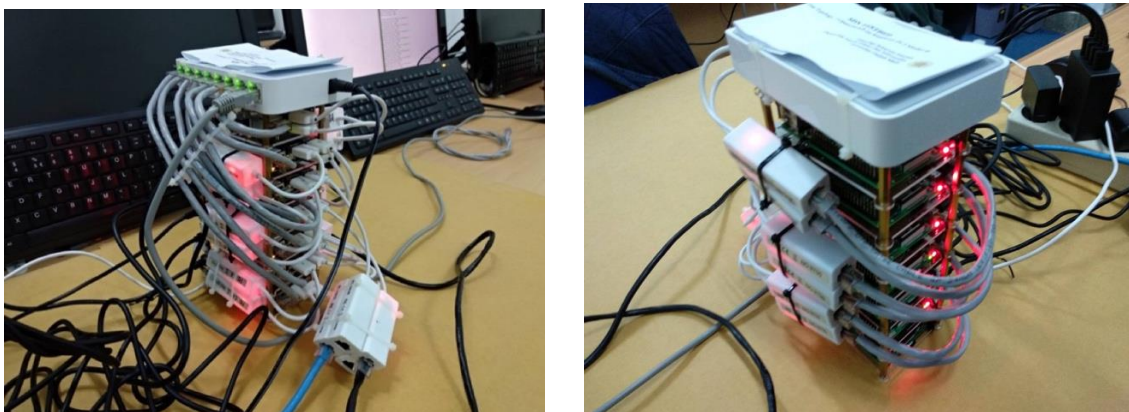
- Linuwih, Brayan Anggita. 2016. *Perancangan dan Analisis Software Defined Network pada Jaringan LAN: Penerapan dan Analisis Metode Penjaluran Path Calculating menggunakan Alogritma Dijkstra*. E-Proceeding of Engineering Vol.3, No.1 April 2016.
- Malhotra, Manoj. 2017. *OpenManage Mobile with iDrack Quick Sync 2*. DellEMC
- Mulyana, Eueng. 2015. *Buku Komunitas SDN-RG*. Bandung : Gitbook.
- Martin, James. 1991. *Rapid Application Development*. Macmillan Publishing Co., Inc.,
- Patton, Ron. 2001. *Software Testing 2nd Edition*.
- Phonearena. 2011. *Google's Android OS: Past, Present, and Future*. Diakses pada 30 Juni 2018 halaman situs https://www.phonearena.com/news/Googles-Android-OS-Past-Present-and-Future_id21273.
- Premoni, Shiddiq Jati. 2017. *Pengembangan Aplikasi Context-Aware pada Ponsel Pintar Berbasis Lokasi dan Aktivitas*. Yogyakarta: Program Studi Teknologi Informasi Universitas Gadjah Mada.
- Roberts, James. 2015. *Flow-Aware Networking*. Switzerland: Springer International Publishing.
- Techcrunch. 2014. *Exclusive: Quantum Paper And Google's Upcoming Effort To Make Consistent UI Simple*. Diakses pada 30 Juni 2018 halaman situs <http://www.androidpolice.com/2014/06/11/exclusive-quantum-paper-and-googles-upcoming-effort-to-make-consistent-ui-simple/>.
- Technopedia. _____. *Mobile Application (Mobile App)*. Diakses pada 30 Juni 2018 halaman situs <https://www.techopedia.com/definition/2953/mobile-application-mobile-app>

LAMPIRAN

LAMPIRAN – 1. Aplikasi manajemen *flow* dengan nama “OpenDaylight Manager Mobile di Google Playstore



LAMPIRAN – 2. Implementasi topologi jaringan SDN pada perangkat Raspberry Pi



LAMPIRAN – 3. Struktur *source code* aplikasi manajemen *flow*

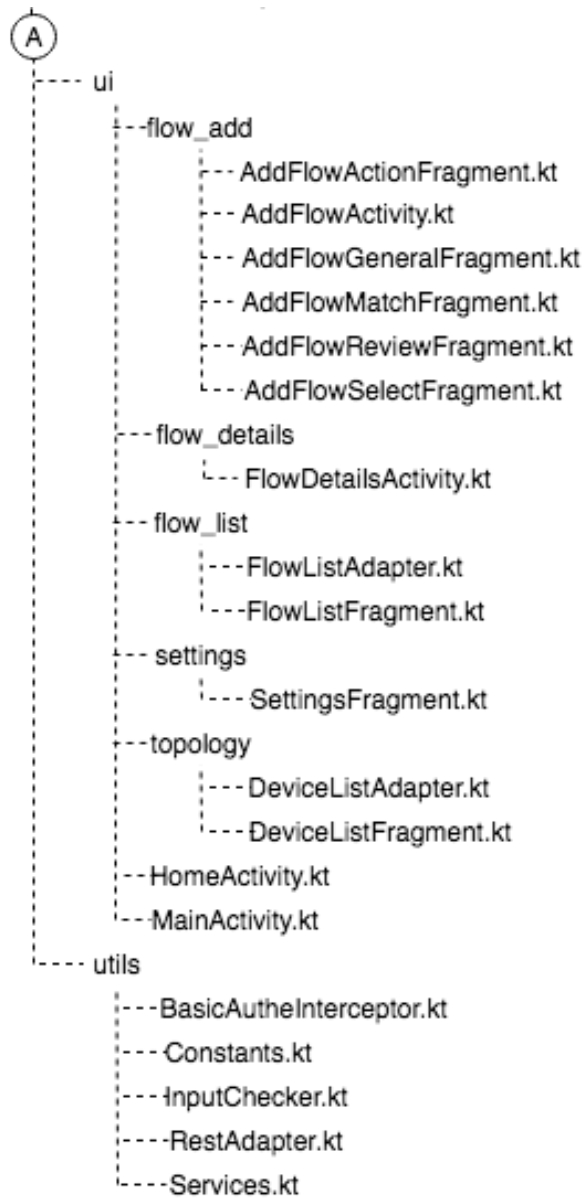
com.edityomurti.openflowmanagerapp (main package)

```

--- models
    --- flowtable
        --- flow
            -- Action.kt
            -- ApplyAction.kt
            -- Duration.kt
            -- EthernetMatch.kt
            -- Flow.kt
            -- FlowDataSent.kt
            -- FlowStatistics.kt
            -- Input.kt
            -- InputData.kt
            -- Instruction.kt
            -- InstructionData.kt
            -- Match.kt
            -- OutputAction.kt
        --- FlowHashMap.kt
        --- FlowTable.kt
        --- FlowTableData.kt
    --- topology
        --- AddressTrackerAddresses.kt
        --- Device.kt
        --- Link.kt
        --- NetworkTopology.kt
        --- Node.kt
        --- NodeConnector.kt
        --- NodeData.kt
        --- NodeDataSerializable.kt
        --- Nodes.kt
        --- Topology.kt
        --- TopologyData.kt
    --- FlowProperties.kt

```

A



LAMPIRAN – 4. Publikasi *source code* aplikasi manajemen flow pada Github

Link url : <https://github.com/edityomurti/OpenDaylight-Manager-for-Android>


The screenshot shows the GitHub repository page for 'edityomurti / OpenDaylight-Manager-for-Android'. The repository is described as 'a Flow Management App for OpenDaylight SDN Controller'. It has 26 commits, 1 branch, 0 releases, and 1 contributor. The repository is licensed under the MIT license. The repository is tagged with 'android', 'android-application', 'software-defined-network', 'sdn', 'sdn-controller', 'software-defined-networking', 'openflow', 'opendaylight', 'sdn-network', 'sdn-apps', and 'floodlight'. The repository is also tagged with 'Manage topics'.

The repository is currently on the 'master' branch. The latest commit is 'Update README.md' by edityomurti, committed 10 days ago. The commit history shows several updates to the README.md file and initial commits for various files.

The README.md file contains the following information:

OpenDaylight Manager Mobile

OpenDaylight Manager Mobile enables an IT Administrator to securely configure, monitor, and troubleshoot OpenDaylight SDN Controller directly from mobile devices.



Download

You can download the app on Google Playstore - <https://play.google.com/store/apps/details?id=com.edityomurti.openflowmanagerapp>

Screenshoot

The screenshots show the app's interface. The first screenshot shows the 'Devices' screen with a list of switches. The second screenshot shows the 'Flow Table' screen with a table of flow entries.

Device	Flow ID	Priority	Type
openflow:7	#UFSTABLE*0-422	100	operational
openflow:7	#UFSTABLE*0-451	2	operational

LAMPIRAN – 5. Source Code aplikasi manajemen flow

1. AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.edityomurti.openflowmanagerapp">

    <uses-permission android:name="android.permission.INTERNET" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher_odl_man"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_odl_man_round"
        android:supportRtl="true"
        android:theme="@style/AppTheme">
        <activity
            android:name=".ui.MainActivity"
            android:screenOrientation="portrait"></activity>
        <activity
            android:name=".ui.HomeActivity"
            android:screenOrientation="portrait"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".ui.flow_details.FlowDetailsActivity"
            android:screenOrientation="portrait"></activity>
        <activity android:name=".ui.flow_add.AddFlowActivity"
            android:screenOrientation="portrait"></activity>
    </application>

</manifest>
```

2. Action.kt

```
data class Action(
    @SerializedName("order")
    var order: Int?,
    @SerializedName("output-action")
    var outputAction: OutputAction?,
    @SerializedName("drop-action")
    var dropAction: DropAction?
): Serializable

class DropAction: Serializable
```

3. ApplyActions.kt

```
data class ApplyActions(
    @SerializedName("action")
    var actionData: MutableList<Action>
): Serializable
```

4. Duration.kt

```

data class Duration(
    @SerializedName("nanosecond")
    var nanoSecond: Int?,
    @SerializedName("second")
    var second: Int?
): Serializable

```

5. EthernetMatch.kt

```

data class EthernetMatch(
    @SerializedName("ethernet-source")
    var ethernetSource: EthernetSource?,
    @SerializedName("ethernet-destination")
    var ethernetDestination: EthernetDestination?,
    @SerializedName("ethernet-type")
    var ethernetType: EthernetType?
): Serializable

class EthernetType(
    @SerializedName("type")
    var type: String?
): Serializable

class EthernetSource(
    @SerializedName("address")
    var address: String?
): Serializable

class EthernetDestination(
    @SerializedName("address")
    var address: String?
): Serializable

```

6. Flow.kt

```

data class Flow(
    @Transient
    var flowType: String? = null,
    var nodeId: String? = null,
    @SerializedName("id")
    var id: String? = null,
    @SerializedName("cookie")
    var cookie: BigInteger? = null,
    @SerializedName("flow-name")
    var flowName: String? = null,
    @SerializedName("instructions")
    var instructions: InstructionData? = null,
    @SerializedName("flags")
    var flags: String? = null,
    @SerializedName("match")
    var match: Match? = null,
    @SerializedName("hard-timeout")
    var hardTimeOut: Int? = null,
    @SerializedName("idle-timeout")
    var idleTimeOut: Int? = null,
    @SerializedName("priority")
    var priority: Int? = null,
    @SerializedName("table_id")
    var tableId: Int? = null,
    @SerializedName("opendaylight-flow-statistics:flow-statistics")

```

```

        var flowStatistics: FlowStatistics? = null
    ): Serializable

```

7. FlowDataSent.kt

```

data class FlowDataSent(
    @SerializedName("flow")
    var flowData: MutableList<Flow>?
)

```

8. FlowStatistics.kt

```

data class FlowStatistics(
    @SerializedName("packet-count")
    var packetCount: Int?,
    @SerializedName("byte-count")
    var byteCount: Int?,
    @SerializedName("duration")
    var duration: Duration
): Serializable

```

9. Input.kt

```

data class Input(
    @SerializedName("match")
    var match: Match?,
    @SerializedName("table_id")
    var tableId: Int?,
    @SerializedName("priority")
    var priority: Int?,
    @SerializedName("node")
    var node: String?
)

```

10. InputData.kt

```

data class InputData(
    @SerializedName("input")
    var input: Input
)

```

11. Instruction.kt

```

data class Instruction(
    @SerializedName("order")
    var order: Int?,
    @SerializedName("apply-actions")
    var applyActions: ApplyActions?
): Serializable

```

12. InstructionData.kt

```

data class InstructionData(
    @SerializedName("instruction")
    var instruction: MutableList<Instruction>?
): Serializable

```

13. Match.kt

```

data class Match(
    @SerializedName("ethernet-match")
    var ethernetMatch: EthernetMatch?,
    @SerializedName("in-port")
    var inPort: String?,
    @SerializedName("ipv4-source")
    var ipv4source: String?,
    @SerializedName("ipv4-destination")
    var ipv4destination: String?
): Serializable

```

14. OutputAction.kt

```

data class OutputAction(
    @SerializedName("max-length")
    var maxLength: Int?,
    @SerializedName("output-node-connector")
    var outputNodeConnector: String?
): Serializable

```

15. FlowHashIdMap.kt

```

data class FlowHashIdMap(
    @SerializedName("hash")
    var hash: String?,
    @SerializedName("flow-id")
    var flowId: String?
)

```

16. FlowTable.kt

```

data class FlowTable(
    @SerializedName("id")
    var id: Int?,
    @SerializedName("flow-hash-id-map")
    var flowHashIdMapData: MutableList<FlowHashIdMap>?,
    @SerializedName("flow")
    var flowData: MutableList<Flow>?
)

```

17. FlowTableData.kt

```

data class FlowTableData(
    @SerializedName("flow-node-inventory:table")
    var table: MutableList<FlowTable>
)

```

18. AddressTrackerAdresses.kt

```

data class AddressTrackerAdresses(
    @SerializedName("id")

```

```

var id: String?,
@SerializedName("mac")
var mac: String?,
@SerializedName("ip")
var ip: String?
)

```

19. Device.kt

```

data class Device(
    var deviceName: String?,
    var deviceDesc: String?,
    var deviceType: String?,
    var linkDevice: MutableList<Link>
)

```

20. Link.kt

```

data class Link(
    @SerializedName("ink-id")
    var linkId: String?,
    @SerializedName("source")
    var source: Source?,
    @SerializedName("destination")
    var destination: Destination?
)

data class Source(
    @SerializedName("source-tp")
    var sourceTp: String?,
    @SerializedName("source-node")
    var sourceNode: String?
)

data class Destination(
    @SerializedName("dest-node")
    var destNode: String?,
    @SerializedName("dest-tp")
    var destTp: String?
)

```

21. NetworkTopology.kt

```

data class NetworkTopology(
    @SerializedName("network-topology")
    var topologyData: TopologyData?
)

```

22. Node.kt

```

data class Node(
    @SerializedName("node-id")
    var nodeId: String?,
    @SerializedName("id")
    var id: String?,
    @SerializedName("flow-node-inventory:ip-address")
    var ipAddress: String?,
    @SerializedName("node-connector")
    var nodeConnector: MutableList<NodeConnector>?,
    @SerializedName("termination-point")
)

```

```

    var terminationPointData: MutableList<TerminationPoint>,
    @SerializedName("host-tracker-service:attachment-points")
    var hostTrackerAttachmentPointData: MutableList<HostTrackerAttachmentPoint>,
    @SerializedName("host-tracker-service:addresses")
    var hostTrackerAddressesData: MutableList<HostTrackerAddress>,
    @SerializedName("host-tracker-service:id")
    var hostTrackerId: String?
)

data class TerminationPoint(
    @SerializedName("tp-id")
    var tpId: String?
): Serializable

data class HostTrackerAttachmentPoint(
    @SerializedName("tp-id")
    var tpId: String?,
    @SerializedName("corresponding-tp")
    var correspondingTp: String?,
    @SerializedName("active")
    var active: Boolean?
)

data class HostTrackerAddress(
    @SerializedName("id")
    var id: Int?,
    @SerializedName("mac")
    var mac: String?,
    @SerializedName("ip")
    var ip: String?
)

```

23. NodeConnector.kt

```

data class NodeConnector(
    @SerializedName("id")
    var id: String?,
    @SerializedName("flow-node-inventory:hardware-address")
    var hardwareAddress: String?,
    @SerializedName("flow-node-inventory:name")
    var name: String?,
    @SerializedName("address-tracker:addresses")
    var addresses: MutableList<AddressTrackerAddresses>?
)

```

24. NodeData.kt

```

data class NodeData(
    @SerializedName("node")
    var node: MutableList<Node>
)

```

25. NodeDataSerializable.kt

```

data class NodeDataSerializable(
    var nodeSerializableData: MutableList<NodeSerializable>
): Serializable

data class NodeSerializable(

```

```

    var nodeId: String,
    var nodeConnector: ArrayList<String>
): Serializable

```

26. Nodes.kt

```

data class Nodes(
    @SerializedName("nodes")
    var nodeData: NodeData?
)

```

27. Topology.kt

```

data class Topology(
    @SerializedName("topology-id")
    var topologyName: String?,
    @SerializedName("node")
    var nodeData: MutableList<Node>?,
    @SerializedName("link")
    var linkData: MutableList<Link>?
)

```

28. FlowTable.kt

```

data class TopologyData(
    @SerializedName("topology")
    var topology: MutableList<Topology>
)

```

29. FlowProperties.kt

```

data class FlowProperties(
    var propId : String,
    var propName: String,
    var layoutView: Int
)

```

30. AddFlowActionFragment.kt

```

class AddFlowActionFragment : Fragment() {

    lateinit var mView: View

    var actionData: MutableList<FlowProperties> = ArrayList()
    var selectedActionData: MutableList<FlowProperties> = ArrayList()

    val tag_action_drop = "action_drop"
    val tag_action_flood = "action_flood"
    val tag_action_flood_all = "action_flood_all"
    val tag_action_controller = "action_controller"
    val tag_action_normal = "action_normal"
    val tag_action_output_port = "action_output_port"
    val tag_action_output_port_2 = "action_output_port_2"
}

```

```

var nodeData: NodeDataSerializable? = null
var newFlow: Flow? = null

var arrayAdapterPort: ArrayAdapter<String>? = null

val VALUE_MAX = 65535
val VALUE_CANT_BE_BLANK = "Cannot be blank"
val VALUE_ERROR = "Value must be between 0 - $VALUE_MAX"

var modeAdd = true
var inflater: LayoutInflater? = null

companion object {
    fun newInstance(nodeData: NodeDataSerializable): AddFlowActionFragment{
        val args = Bundle()
        args.putSerializable(Constants.NODE_DATA, nodeData)

        var fragment = AddFlowActionFragment()
        fragment.arguments = args
        return fragment
    }
}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?): View? {
    mView = inflater.inflate(R.layout.fragment_add_flow_action, container, false)
    this.inflater = inflater

    newFlow = (activity as AddFlowActivity).getFlow()

    if(arguments != null){
        var nodeList: ArrayList<String>? = null

        nodeData = arguments!!.getSerializable(Constants.NODE_DATA) as NodeDataSerializable
        for(i in nodeData!!.nodeSerializableData.indices){
            if(nodeData!!.nodeSerializableData.get(i).nodeId == newFlow?.nodeId){
                nodeList = nodeData!!.nodeSerializableData.get(i).nodeConnector
            }
        }
        arrayAdapterPort = ArrayAdapter(context, R.layout.item_spinner, nodeList)
    }

    setDefaultData()
    setData()

    var alertDialog = AlertDialog.Builder(context)
    var arrayAdapter = ArrayAdapter<String>(context, android.R.layout.simple_list_item_1)
    for (i in actionData.indices){
        arrayAdapter.add(actionData[i].propName)
    }

    alertDialog.setAdapter(arrayAdapter) { dialog, which ->
        val view = inflater.inflate(actionData[which].layoutView, null, false)
        val btnDelete = view.findViewById<ImageView>(R.id.iv_delete)
        val action = actionData[which]

        if(action.propId == tag_action_output_port){
            val spinnerPort = view.findViewById<Spinner>(R.id.spinner_output_port)
            spinnerPort.adapter = arrayAdapterPort
        } else if(action.propId == tag_action_output_port_2){
            val spinnerPort = view.findViewById<Spinner>(R.id.spinner_output_port_2)
            spinnerPort.adapter = arrayAdapterPort
        }
    }
}

```



```

btnDelete.setOnClickListener {
    (view.parent as LinearLayout).removeView(view)
    selectedActionData.remove(action)
    actionData.add(action)
    arrayAdapter.add(action.propName)
    arrayAdapter.notifyDataSetChanged()
    if(arrayAdapter.count == 1){
        mView.btn_add_action.visibility = View.VISIBLE
    }
}

selectedActionData.add(action)
arrayAdapter.remove(action.propName)
arrayAdapter.notifyDataSetChanged()
actionData.remove(action)
mView.ll_action.addView(view)
dialog.dismiss()

if(arrayAdapter.count == 0){
    mView.btn_add_action.visibility = View.GONE
} else {
    mView.btn_add_action.visibility = View.VISIBLE
}
}

mView.btn_add_action.setOnClickListener {
    alertDialog.show()
}

return mView
}

fun setData(){
    this.modeAdd = (activity as AddFlowActivity).getMode()

    (activity as AddFlowActivity).setStatus(false)
    mView.tv_device.text = newFlow!!.nodeId

    if(modeAdd){
    } else {
        mView.btn_add_action.visibility = View.GONE

        var nodeId = newFlow?.nodeId
        var actionControllerMaxLength: Int? = null
        var actionDrop = false
        var actionNormalMaxLength: Int? = null
        var actionFlood = false
        var actionFloodAll = false
        var outputPortData: MutableList<OutputPort> = ArrayList()

        if(newFlow?.instructions != null && newFlow?.instructions?.instruction!![0].applyActions
        != null){
            var mActionData = newFlow?.instructions?.instruction!![0].applyActions?.actionData
            for(i in mActionData?.indices!!){
                if(mActionData[i].outputAction?.outputNodeConnector.equals("CONTROLLER")){
                    actionControllerMaxLength = mActionData[i].outputAction?.maxLength
                } else if (mActionData[i].outputAction?.outputNodeConnector.equals("NORMAL")){
                    actionNormalMaxLength = mActionData[i].outputAction?.maxLength
                } else if (mActionData[i].outputAction?.outputNodeConnector.equals("FLOOD")){
                    actionFlood = true
                } else if (mActionData[i].outputAction?.outputNodeConnector.equals("FLOOD_ALL")){
                    actionFloodAll = true
                }
            }
        }
    }
}

```

```

    } else if(mActionData[i].dropAction != null){
        actionDrop = true
    } else {
//      var outputPort = inPort?.substring(0, inPort.lastIndexOf(":")+1) +
actionData[i].outputAction?.outputNodeConnector
        var outputPort = nodeId + ":" + mActionData[i].outputAction?.outputNodeConnector
        var outputPortMaxLength = mActionData[i].outputAction?.maxLength

        outputPortData.add(OutputPort(outputPort, outputPortMaxLength))
    }
}
}
if(actionDrop){
    try{
        mView.tv_action_drop.visibility = View.VISIBLE
    } catch (e: Exception){
        addView(0)

        mView.tv_action_drop.visibility = View.VISIBLE
    }
}

if(actionFlood){
    try{
        mView.tv_action_flood.visibility = View.VISIBLE
    } catch (e: Exception){
        addView(1)

        mView.tv_action_flood.visibility = View.VISIBLE
    }
}

if(actionFloodAll){
    try{
        mView.tv_action_flood_all.visibility = View.VISIBLE
    } catch (e: Exception){
        addView(2)

        mView.tv_action_flood_all.visibility = View.VISIBLE
    }
}

if(actionControllerMaxLength != null){
    try{
        mView.et_controller_max_length.visibility = View.VISIBLE
        mView.et_controller_max_length.setText(actionControllerMaxLength.toString())
    } catch (e: Exception){
        addView(3)

        mView.et_controller_max_length.visibility = View.VISIBLE
        mView.et_controller_max_length.setText(actionControllerMaxLength.toString())
    }
}

if(actionNormalMaxLength != null){
    try{
        mView.et_normal_max_length.visibility = View.VISIBLE
        mView.et_normal_max_length.setText(actionNormalMaxLength.toString())
    } catch (e: Exception){
        addView(4)

        mView.et_normal_max_length.visibility = View.VISIBLE
        mView.et_normal_max_length.setText(actionNormalMaxLength.toString())
    }
}

```

```

    }

    println("AddFlowMatch, outputPortData.size : ${outputPortData.size}")
    if(outputPortData.size != 0){
        try{
            var outputPort1 = outputPortData[0].outputPort
            mView.spinner_output_port.visibility = View.VISIBLE
            var spinnerPos = arrayAdapterPort?.getPosition(outputPort1)
            mView.spinner_output_port.setSelection(spinnerPos!!)

mView.et_output_port_maxlength.setText(outputPortData[0].outputMaxLength!!.toString())
        } catch (e: Exception){
            addView(5)

            var outputPort1 = outputPortData[0].outputPort
            mView.spinner_output_port.visibility = View.VISIBLE
            var spinnerPos = arrayAdapterPort?.getPosition(outputPort1)
            mView.spinner_output_port.setSelection(spinnerPos!!)

mView.et_output_port_maxlength.setText(outputPortData[0].outputMaxLength!!.toString())
        }

        if(outputPortData.size > 1){
            try{
                var outputPort2 = outputPortData[1].outputPort
                mView.spinner_output_port_2.visibility = View.VISIBLE
                var spinnerPos = arrayAdapterPort?.getPosition(outputPort2)
                mView.spinner_output_port_2.setSelection(spinnerPos!!)

mView.et_output_port_2_max_length.setText(outputPortData[1].outputMaxLength!!.toString())
            } catch (e: Exception){
                addView(6)

                var outputPort2 = outputPortData[1].outputPort
                mView.spinner_output_port_2.visibility = View.VISIBLE
                var spinnerPos = arrayAdapterPort?.getPosition(outputPort2)
                mView.spinner_output_port_2.setSelection(spinnerPos!!)

mView.et_output_port_2_max_length.setText(outputPortData[1].outputMaxLength!!.toString())
            }
        }
    }
}

fun addView(which: Int){
    val view = inflater?.inflate(actionData[which].layoutView, null, false)
    val btnDelete = view?.findViewById<ImageView>(R.id.iv_delete)
    val action = actionData[which]

    btnDelete?.visibility = View.GONE

    if(action.propId == tag_action_output_port){
        val spinnerPort = view?.findViewById<Spinner>(R.id.spinner_output_port)
        spinnerPort?.adapter = arrayAdapterPort
    } else if(action.propId == tag_action_output_port_2){
        val spinnerPort = view?.findViewById<Spinner>(R.id.spinner_output_port_2)
        spinnerPort?.adapter = arrayAdapterPort
    }

    selectedActionData.add(action)

    mView.ll_action.addView(view)
}

```

```

fun setDefaultData(){
    actionData.clear()
    selectedActionData.clear()

    var drop = FlowProperties(tag_action_drop, "Drop",
        R.layout.layout_action_drop)
    actionData.add(drop)

    var flood = FlowProperties(tag_action_flood, "Flood",
        R.layout.layout_action_flood)
    actionData.add(flood)

    var floodAll = FlowProperties(tag_action_flood_all, "Flood All",
        R.layout.layout_action_flood_all)
    actionData.add(floodAll)

    var controller = FlowProperties(tag_action_controller, "Controller",
        R.layout.layout_action_controller)
    actionData.add(controller)

    var normal = FlowProperties(tag_action_normal, "Normal",
        R.layout.layout_action_normal)
    actionData.add(normal)

    var outputPort = FlowProperties(tag_action_output_port, "Output port",
        R.layout.layout_action_output_port)
    actionData.add(outputPort)

    var outputPort2 = FlowProperties(tag_action_output_port_2, "Output port 2",
        R.layout.layout_action_output_port_2)
    actionData.add(outputPort2)
}

fun setFlow(): Flow {
    var isCompleted = true
    println("setFlow :::")
    newFlow = (activity as AddFlowActivity).getFlow()

    var actionData: MutableList<Action> = ArrayList()

    for (i in selectedActionData.indices){
        when(selectedActionData[i].propId){
            tag_action_drop -> {
                println("setFlow :::: setDrop pos : $i")
                var action = Action(i, null, DropAction())
                actionData.add(action)
            }
            tag_action_flood -> {
                var outputAction = OutputAction(0, "FLOOD")
                val action = Action(i, outputAction, null)
                actionData.add(action)
            }
            tag_action_flood_all -> {
                var outputAction = OutputAction(0, "FLOOD_ALL")
                val action = Action(i, outputAction, null)
                actionData.add(action)
            }
            tag_action_controller -> {
                if (!mView.et_controller_max_length.text.isEmpty()) ||
!mView.et_controller_max_length.text.isEmpty()){
                    var maxLength = mView.et_controller_max_length.text.toString().toInt()
                    if (maxLength in 0..VALUE_MAX){
                        mView.et_controller_max_length.error = null

```

```

        var outputAction = OutputAction(maxLength, "CONTROLLER")
        val action = Action(i, outputAction, null)
        actionData.add(action)
    } else {
        mView.et_controller_max_length.error = VALUE_ERROR
        isCompleted = false
    }
} else {
    mView.et_controller_max_length.error = VALUE_CANT_BE_BLANK
    isCompleted = false
}
}

tag_action_normal -> {
    if(!mView.et_normal_max_length.text.isNullOrEmpty() ||
!mView.et_normal_max_length.text.isNullOrBlank()){
        var maxLength = mView.et_normal_max_length.text.toString().toInt()
        if(maxLength in 0..VALUE_MAX){
            mView.et_normal_max_length.error = null
            var outputAction = OutputAction(maxLength, "NORMAL")
            val action = Action(i, outputAction, null)
            actionData.add(action)
        } else {
            mView.et_normal_max_length.error = VALUE_ERROR
            isCompleted = false
        }
    } else {
        mView.et_normal_max_length.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}

tag_action_output_port -> {
    if(!mView.et_output_port_maxlength.text.isNullOrEmpty() ||
!mView.et_output_port_maxlength.text.isNullOrBlank()){
        var maxLength = mView.et_output_port_maxlength.text.toString().toInt()
        if(maxLength in 0..VALUE_MAX){
            mView.et_output_port_maxlength.error = null
            var node = mView.spinner_output_port.selectedItem.toString()
            var nodeConnector = node.substring(node.lastIndexOf(":")+1, node.length)
            var outputAction = OutputAction(maxLength, nodeConnector)
            val action = Action(i, outputAction, null)
            actionData.add(action)
        } else {
            mView.et_output_port_maxlength.error = VALUE_ERROR
            isCompleted = false
        }
    } else {
        mView.et_output_port_maxlength.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}

tag_action_output_port_2 -> {
    if(!mView.et_output_port_2_max_length.text.isNullOrEmpty() ||
!mView.et_output_port_2_max_length.text.isNullOrBlank()){
        var maxLength = mView.et_output_port_2_max_length.text.toString().toInt()
        if(maxLength in 0..VALUE_MAX){
            mView.et_output_port_2_max_length.error = null
            var node = mView.spinner_output_port_2.selectedItem.toString()
            var nodeConnector = node.substring(node.lastIndexOf(":")+1, node.length)
            var outputAction = OutputAction(maxLength, nodeConnector)
            val action = Action(i, outputAction, null)
            actionData.add(action)
        } else {
            mView.et_output_port_2_max_length.error = VALUE_ERROR

```

```

        isCompleted = false
    }
    } else {
        mView.et_output_port_2_max_length.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
    }
}

if(actionData.size != 0){
    var applyAction = ApplyActions(actionData)
    var instruction = Instruction(0, applyAction)

    var instructionData: MutableList<Instruction> = ArrayList()
    instructionData.add(instruction)

    var instructions = InstructionData(instructionData)

    newFlow?.instructions = instructions
}

if(isCompleted){
    (activity as AddFlowActivity).setDataStatus(true)
} else {
    (activity as AddFlowActivity).setDataStatus(false)
}

return newFlow!!
}

data class OutputPort(
    var outputPort: String?,
    var outputMaxLength: Int?
)
}

```

31. AddFlowActivity.kt

```

class AddFlowActivity : AppCompatActivity() {
    val TAG_SELECT_DEVICE_FRAGMENT = "TAG_SELECT_DEVICE_FRAGMENT"
    val TAG_GENERAL_PROPERTIES_FRAGMENT = "TAG_GENERAL_PROPERTIES"
    val TAG_MATCH_FRAGMENT = "TAG_MATCH_FRAGMENT"
    val TAG_ACTION_FRAGMENT = "TAG_ACTION_FRAGMENT"
    val TAG_REVIEW_FRAGMENT = "TAG_REVIEW_FRAGMENT"

    lateinit var selectDeviceFragment: AddFlowSelectDeviceFragment
    lateinit var generalPropertiesFragment: AddFlowGeneralFragment
    lateinit var matchFragment: AddFlowMatchFragment
    lateinit var actionFragment: AddFlowActionFragment
    lateinit var reviewFragment: AddFlowReviewFragment

    lateinit var nodeList: ArrayList<String>
    var nodeData: NodeDataSerializable? = null

    private var currentPosition = 1
    private val MAX_POSITION = 6

    var newFlow = Flow()

    var dataStatus: Boolean? = false

    var restAdapter: RestAdapter? = null
}

```

```

var addMode: String? = null

var titleMode: String = "Add Flow"

var modeAdd: Boolean = true

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_add_flow)

    title = titleMode
    supportActionBar?.setDisplayHomeAsUpEnabled(true)

    addMode = intent.getStringExtra(Constants.ADD_MODE)
    nodeList = intent.getStringArrayListExtra(Constants.NODE_LIST)
    nodeData = intent.getSerializableExtra(Constants.NODE_DATA) as NodeDataSerializable

    modeAdd = addMode == Constants.MODE_ADD

    if(modeAdd){
        currentPosition = 1
        titleMode = "Add Flow"
    } else {
        currentPosition = 2
        titleMode = "Edit Flow"
        newFlow = intent.extras.getSerializable(Constants.OBJECT_FLOW) as Flow
    }
    selectDeviceFragment = AddFlowSelectDeviceFragment.newInstance(nodeList)
    generalPropertiesFragment = AddFlowGeneralFragment()
    matchFragment = AddFlowMatchFragment.newInstance(nodeData!!)
    actionFragment = AddFlowActionFragment.newInstance(nodeData!!)
    reviewFragment = AddFlowReviewFragment()
    restAdapter = RestAdapter(this)

    setNavigation()

    btn_next.setOnClickListener{
        if(currentPosition != MAX_POSITION){
            currentPosition += 1
            onClickNavigation()
        }
    }

    btn_previous.setOnClickListener{
        currentPosition -= 1
        onClickNavigation()
    }
}

fun setNavigation(){
    when(currentPosition){
        1 -> {
            btn_previous.visibility = View.INVISIBLE
            tv_next.visibility = View.VISIBLE
            title = "$titleMode: Select Device"
            showSelectDevice()
        }
        2 -> {
            println("AddFlow, setNav current == 2")
            if(modeAdd){
                this.newFlow = selectDeviceFragment.setFlow()
            } else {
                dataStatus = true
            }
        }
    }
}

```

```

        if(dataStatus!!){
            if(addMode == Constants.MODE_ADD){
                btn_previous.visibility = View.VISIBLE
                btn_next.visibility = View.VISIBLE
            } else {
                btn_previous.visibility = View.INVISIBLE
                btn_next.visibility = View.VISIBLE
            }
            tv_next.text = "Next"
            title = "$titleMode: General Properties"
            showGeneralProp()
        } else {
            currentPosition -= 1
        }
    }
}
3 -> {
    this.newFlow = generalPropertiesFragment.setFlow()

    if(dataStatus!!){
        btn_previous.visibility = View.VISIBLE
        btn_next.visibility = View.VISIBLE
        showMatch()
        title = "$titleMode: Match"
        tv_next.text = "Next"
    } else {
        currentPosition -= 1
    }
}
4 -> {
    this.newFlow = matchFragment.setFlow()
    if(dataStatus!!){
        btn_previous.visibility = View.VISIBLE
        btn_next.visibility = View.VISIBLE
        tv_next.text = "Review"
        showAction()
        title = "$titleMode: Action"
    } else {
        currentPosition -= 1
    }
}
5 -> {
    this.newFlow = actionFragment.setFlow()
    if(dataStatus!!){
        btn_previous.visibility = View.VISIBLE
        btn_next.visibility = View.VISIBLE
        tv_next.text = "Finish"
        title = "$titleMode: Review"
        showReview()
    } else {
        currentPosition -= 1
        Toast.makeText(this, "Something wrong", Toast.LENGTH_SHORT).show()
    }
}
6 -> {
    sendFlow()
}
}
val focus = this.currentFocus
if(focus != null){
    val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
    imm.hideSoftInputFromWindow(focus.windowToken, 0)
}
}
}

```



```

fun showSelectDevice(){
    var transaction = supportFragmentManager.beginTransaction()
    // transaction.setCustomAnimations(R.anim.slide_in_left, R.anim.slide_out_right)
    if(supportFragmentManager.findFragmentByTag(TAG_GENERAL_PROPERTIES_FRAGMENT)
    != null){

transaction.hide(supportFragmentManager.findFragmentByTag(TAG_GENERAL_PROPERTIES_FRAGMENT))
    }
    if(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT) != null){
        transaction.hide(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT))
    }

    if(supportFragmentManager.findFragmentByTag(TAG_SELECT_DEVICE_FRAGMENT) != null){

transaction.show(supportFragmentManager.findFragmentByTag(TAG_SELECT_DEVICE_FRAGMENT
))
        } else {
            transaction.add(R.id.fragment_container, selectDeviceFragment,
TAG_SELECT_DEVICE_FRAGMENT)
        }
        transaction.commitAllowingStateLoss()
    }

    fun showGeneralProp(){
        var transaction = supportFragmentManager.beginTransaction()
        // transaction.setCustomAnimations(R.anim.slide_in_left, R.anim.slide_out_right)
        if(supportFragmentManager.findFragmentByTag(TAG_SELECT_DEVICE_FRAGMENT) != null){

transaction.hide(supportFragmentManager.findFragmentByTag(TAG_SELECT_DEVICE_FRAGMENT
))
        }
        if(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT) != null){
            transaction.hide(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT))
        }

        if(supportFragmentManager.findFragmentByTag(TAG_GENERAL_PROPERTIES_FRAGMENT)
        != null){

transaction.show(supportFragmentManager.findFragmentByTag(TAG_GENERAL_PROPERTIES_FRAGMENT
))
            generalPropertiesFragment.setData()
        } else {
            transaction.add(R.id.fragment_container, generalPropertiesFragment,
TAG_GENERAL_PROPERTIES_FRAGMENT)
            val focus = this.currentFocus
            if(focus != null){
                val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
                imm.hideSoftInputFromWindow(focus.windowToken, 0)
            }
        }
        transaction.commitAllowingStateLoss()
    }

    fun showMatch(){
        var transaction = supportFragmentManager.beginTransaction()
        if(supportFragmentManager.findFragmentByTag(TAG_SELECT_DEVICE_FRAGMENT) != null){

transaction.hide(supportFragmentManager.findFragmentByTag(TAG_SELECT_DEVICE_FRAGMENT
))
        }
        if(supportFragmentManager.findFragmentByTag(TAG_GENERAL_PROPERTIES_FRAGMENT)
        != null){

```

```

transaction.hide(supportFragmentManager.findFragmentByTag(TAG_GENERAL_PROPERTIES_FRAGMENT))
    }
    if(supportFragmentManager.findFragmentByTag(TAG_ACTION_FRAGMENT) != null){
        transaction.hide(supportFragmentManager.findFragmentByTag(TAG_ACTION_FRAGMENT))
    }

    if(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT) != null){
        transaction.show(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT))
        matchFragment.setData()
    } else {
        transaction.add(R.id.fragment_container, matchFragment, TAG_MATCH_FRAGMENT)
    }
    transaction.commitAllowingStateLoss()
}

fun showAction(){
    var transaction = supportFragmentManager.beginTransaction()
    if(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT) != null){
        transaction.hide(supportFragmentManager.findFragmentByTag(TAG_MATCH_FRAGMENT))
    }
    if(supportFragmentManager.findFragmentByTag(TAG_REVIEW_FRAGMENT) != null){
        transaction.hide(supportFragmentManager.findFragmentByTag(TAG_REVIEW_FRAGMENT))
    }

    if(supportFragmentManager.findFragmentByTag(TAG_ACTION_FRAGMENT) != null){
        transaction.show(supportFragmentManager.findFragmentByTag(TAG_ACTION_FRAGMENT))
        actionFragment.setData()
    } else {
        transaction.add(R.id.fragment_container, actionFragment, TAG_ACTION_FRAGMENT)
    }
    transaction.commitAllowingStateLoss()
}

fun showReview(){
    var transaction = supportFragmentManager.beginTransaction()
    if(supportFragmentManager.findFragmentByTag(TAG_ACTION_FRAGMENT) != null){
        transaction.hide(supportFragmentManager.findFragmentByTag(TAG_ACTION_FRAGMENT))
    }

    if(supportFragmentManager.findFragmentByTag(TAG_REVIEW_FRAGMENT) != null){
        transaction.show(supportFragmentManager.findFragmentByTag(TAG_REVIEW_FRAGMENT))
        reviewFragment.setData()
    } else {
        transaction.add(R.id.fragment_container, reviewFragment, TAG_REVIEW_FRAGMENT)
    }
    transaction.commitAllowingStateLoss()
}

fun getDataStatus() : Boolean{
    return this.dataStatus!!
}

fun setDataStatus(isCompleted: Boolean){
    this.dataStatus = isCompleted
}

fun getFlow(): Flow{
    return this.newFlow
}

fun setFlow(newFlow: Flow){
    this.newFlow = newFlow
}

```

```

}

fun getMode(): Boolean{
    return this.modeAdd
}

fun onClickNavigation(){
    setNavigation()
}

fun sendFlow(){
    var progressDialog = ProgressDialog(this)
    progressDialog.setMessage("Applying Flow ..")
    progressDialog.setCancelable(false)
    progressDialog.setCanceledOnTouchOutside(false)
    progressDialog.show()

    var nodeId = newFlow.nodeId
    newFlow.nodeId = null

    if(!modeAdd){
        nodeId = URLEncoder.encode(nodeId, "utf-8")
        println("ENCODING FLOW NODE ID to : ${nodeId}")
    }

    var flowData: MutableList<Flow> = ArrayList()
    flowData.add(newFlow)
    val flowDataSent = FlowDataSent(flowData)

    restAdapter?.getEndPoint()?.
        postFlow(nodeId!!, newFlow.id!!, flowDataSent)!!.
        enqueue(object : retrofit2.Callback<ResponseBody>{
            override fun onResponse(call: Call<ResponseBody>?, response:
Response<ResponseBody>?) {
                if(response!!.isSuccessful){
                    if(modeAdd){
                        Toast.makeText(this@AddFlowActivity, "Flow Added!",
Toast.LENGTH_SHORT).show()
                    } else {
                        Toast.makeText(this@AddFlowActivity, "Flow Updated!",
Toast.LENGTH_SHORT).show()
                    }
                    progressDialog.dismiss()
                    setResult(Activity.RESULT_OK)
                    finish()
                } else {
                    Toast.makeText(this@AddFlowActivity, "Failed Adding Flow",
Toast.LENGTH_SHORT).show()
                    newFlow.nodeId = nodeId
                    progressDialog.dismiss()
                }
            }

            override fun onFailure(call: Call<ResponseBody>?, t: Throwable?) {
                Toast.makeText(this@AddFlowActivity, "Failed Adding Flow",
Toast.LENGTH_SHORT).show()
                newFlow.nodeId = nodeId
                progressDialog.dismiss()
            }
        })
    })

    override fun onOptionsItemSelected(item: MenuItem?): Boolean {
        when(item?.itemId){

```

```

        android.R.id.home -> onBackPressed()
    }
    return super.onOptionsItemSelected(item)
}
}

```

32. AddFlowGeneralFragment.kt

```

class AddFlowGeneralFragment : Fragment() {
    lateinit var mView: View

    var propData: MutableList<FlowProperties> = ArrayList()

    var selectedPropData: MutableList<FlowProperties> = ArrayList()

    val tag_prop_hardtimeout = "prop_hardtimeout"
    val tag_prop_idletimeout = "prop_idletimeout"
    val tag_prop_cookie = "prop_cookie"

    val VALUE_MAX = 65535
    val VALUE_CANT_BE_BLANK = "Cannot be blank"
    val VALUE_ERROR = "Value must between 0 - $VALUE_MAX"

    var newFlow: Flow? = null

    var modeAdd = true

    var inflater: LayoutInflater? = null

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        mView = inflater.inflate(R.layout.fragment_add_flow_general, container, false)
        this.inflater = inflater

        setDefaultData()
        setData()

        var arrayAdapter = ArrayAdapter<String>(context, android.R.layout.simple_list_item_1)
        var alertDialog = AlertDialog.Builder(context)
        for(i in propData.indices){
            arrayAdapter.add(propData[i].propName)
        }

        alertDialog.setAdapter(arrayAdapter) { dialog, which ->
            val view = inflater.inflate(propData[which].layoutView, null, false)
            val btnDelete = view.findViewById<ImageView>(R.id.iv_delete)
            val prop = propData[which]

            btnDelete.setOnClickListener {
                (view.parent as? LinearLayout)?.removeView(view)
                selectedPropData.remove(prop)
                propData.add(prop)
                arrayAdapter.add(prop.propName)
                arrayAdapter.notifyDataSetChanged()
                if(arrayAdapter.count == 1){
                    mView.btn_add_properties.visibility = View.VISIBLE
                }
            }

            selectedPropData.add(prop)
            arrayAdapter.remove(prop.propName)
            arrayAdapter.notifyDataSetChanged()
            propData.remove(prop)

```

```

        mView.ll_properties.addView(view)
        dialog.dismiss()

        if(arrayAdapter.count == 0){
            mView.btn_add_properties.visibility = View.GONE
        } else {
            mView.btn_add_properties.visibility = View.VISIBLE
        }
    }

    mView.btn_add_properties.setOnClickListener {
        alertDialog.show()
    }

    return mView
}

fun setData(){
    this.modeAdd = (activity as AddFlowActivity).getMode()

    newFlow = (activity as AddFlowActivity).getFlow()

    (activity as AddFlowActivity).setStatus(false)
    mView.tv_device.text = newFlow?.nodeId

    if(modeAdd){
        activity?.title = "Add Flow: General Properties"
    } else {
        activity?.title = "Edit Flow: General Properties"
        mView.btn_add_properties.visibility = View.GONE

        val flowId = newFlow?.id
        val priority = newFlow?.priority
        val hardTimeOut = newFlow?.hardTimeOut
        val idleTimeOut = newFlow?.idleTimeOut
        val cookie = newFlow?.cookie

        // IF ELSE KONDISI PENAMPILAN DATA
        if(flowId != null){
            mView.et_id_flow.setText(flowId)
            mView.et_id_flow.isEnabled = false
        }

        if(priority != null){
            mView.et_priority.setText(priority.toString())
            mView.et_priority.isEnabled = false
        }

        if(hardTimeOut != null){
            println("AddView, adding 0")
            try {
                mView.et_hard_timeout.visibility = View.VISIBLE
                mView.et_hard_timeout.setText(hardTimeOut.toString())
            } catch (e: Exception){
                addView(0)

                mView.et_hard_timeout.visibility = View.VISIBLE
                mView.et_hard_timeout.setText(hardTimeOut.toString())
            }
        }

        if(idleTimeOut != null){
            try{

```

```

        mView.et_idle_timeout.visibility = View.VISIBLE
        mView.et_idle_timeout.setText(idleTimeOut.toString())
    } catch (e: Exception){
        println("AddView, adding 1")
        addView(1)

        mView.et_idle_timeout.visibility = View.VISIBLE
        mView.et_idle_timeout.setText(idleTimeOut.toString())
    }
}

if(cookie != null){
    try {
        mView.et_cookie.visibility = View.VISIBLE
        mView.et_cookie.setText(cookie.toString())
    } catch (e: Exception){
        println("AddView, adding 2")
        addView(2)

        mView.et_cookie.visibility = View.VISIBLE
        mView.et_cookie.setText(cookie.toString())
    }
}
}

fun addView(which: Int){
    val view = inflater?.inflate(propData[which].layoutView, null, false)
    val btnDelete = view?.findViewById<ImageView>(R.id.iv_delete)
    val prop = propData[which]

    // btnDelete?.setOnClickListener {
    //     (view.parent as LinearLayout).removeView(view)
    //     selectedPropData.remove(prop)
    //     propData.add(prop)
    //     arrayAdapter.add(prop.propName)
    //     arrayAdapter.notifyDataSetChanged()
    //     if(arrayAdapter.count == 1){
    //         mView.btn_add_properties.visibility = View.VISIBLE
    //     }
    // }

    btnDelete?.visibility = View.GONE

    selectedPropData.add(prop)
    // arrayAdapter.remove(prop.propName)
    // arrayAdapter.notifyDataSetChanged()
    // propData.remove(prop)

    mView.ll_properties.addView(view)
    // if(arrayAdapter.count == 0){
    //     mView.btn_add_properties.visibility = View.GONE
    // } else {
    //     mView.btn_add_properties.visibility = View.VISIBLE
    // }
}

fun setDefaultData(){
    propData.clear()
    selectedPropData.clear()

    newFlow = (activity as AddFlowActivity).getFlow()

    var hardTimeoutProp = FlowProperties(tag_prop_hardtimeout, "Hard timeout",

```

```

R.layout.layout_prop_hard_timeout)
propData.add(hardTimeoutProp)

var idleTimeoutProp = FlowProperties(tag_prop_idletimeout, "Idle timeout",
R.layout.layout_prop_idle_timeout)
propData.add(idleTimeoutProp)

var cookieProp = FlowProperties(tag_prop_cookie, "Cookie", R.layout.layout_prop_cookie)
propData.add(cookieProp)
}

fun setFlow(): Flow {
var isCompleted = true

newFlow = (activity as AddFlowActivity).getFlow()

newFlow?.tableId = 0

if (!mView.et_id_flow.text.isNullOrEmpty() && !mView.et_id_flow.text.isEmpty()){
val id = mView.et_id_flow.text.toString()
mView.et_id_flow.error = null
newFlow?.id = id
} else {
mView.et_id_flow.error = VALUE_CANT_BE_BLANK
isCompleted = false
}

if (!mView.et_priority.text.isEmpty() && !mView.et_priority.text.isNullOrEmpty()){
try{
val priority = mView.et_priority.text.toString().toInt()
if(priority in 0..VALUE_MAX){
mView.et_priority.error = null
newFlow?.priority = priority
} else {
mView.et_priority.error = VALUE_ERROR
isCompleted = false
}
} catch (e: Exception){
mView.et_priority.error = VALUE_ERROR
isCompleted = false
}
} else {
mView.et_priority.error = VALUE_CANT_BE_BLANK
isCompleted = false
}

for(i in selectedPropData.indices){
when(selectedPropData[i].propId){
tag_prop_hardtimeout -> {
if(!mView.et_hard_timeout.text.isNullOrEmpty() &&
!mView.et_hard_timeout.text.isBlank()){
try{
val hardTimeout = mView.et_hard_timeout.text.toString().toInt()
if(hardTimeout in 0..VALUE_MAX){
mView.et_hard_timeout.error = null
newFlow?.hardTimeOut = hardTimeout
} else {
mView.et_hard_timeout.error = VALUE_ERROR
isCompleted = false
}
} catch (e: Exception){
mView.et_hard_timeout.error = VALUE_ERROR
isCompleted = false
}
}
}
}
}

```

```

    } else {
        mView.et_hard_timeout.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}
tag_prop_idletimeout -> {
    if(!mView.et_idle_timeout.text.isNullOrEmpty() &&
!mView.et_idle_timeout.text.isNullOrBlank()){
        try{
            val idleTimeOut = mView.et_idle_timeout.text.toString().toInt()
            if(idleTimeOut in 0..VALUE_MAX){
                mView.et_idle_timeout.error = null
                newFlow?.idleTimeOut = idleTimeOut
            } else {
                mView.et_idle_timeout.error = VALUE_ERROR
                isCompleted = false
            }
        } catch (e: Exception){
            mView.et_idle_timeout.error = VALUE_ERROR
            isCompleted = false
        }
    } else {
        mView.et_idle_timeout.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}
tag_prop_cookie -> {
    if(!mView.et_cookie.text.isNullOrEmpty() && !mView.et_cookie.text.isNullOrBlank()){
        val cookie = mView.et_cookie.text.toString().toBigInteger()
        mView.et_cookie.error = null
        newFlow?.cookie = cookie
    } else {
        mView.et_cookie.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}
}
}

if(isCompleted){
    (activity as AddFlowActivity).setDataStatus(true)
} else {
    (activity as AddFlowActivity).setDataStatus(false)
}

return newFlow!!
}
}

```

33. AddFlowMatchFragment.kt

```

class AddFlowMatchFragment : Fragment() {

    lateinit var mView: View

    var matchData: MutableList<FlowProperties> = ArrayList()
    var selectedMatchData: MutableList<FlowProperties> = ArrayList()

    val tag_match_inport = "match_inport"
    val tag_match_ehternettype = "match_ehternettype"
    val tag_match_macsource = "match_macsource"
    val tag_match_macdest = "match_macdest"
    val tag_match_ipv4source = "match_ipv4source"
    val tag_match_ipv4dest = "match_ipv4dest"

```



```

var nodeData: NodeDataSerializable? = null
var newFlow: Flow? = null

var arrayAdapterInport: ArrayAdapter<String>? = null

val VALUE_MAX = 65535
val VALUE_CANT_BE_BLANK = "Cannot be blank"
val VALUE_ERROR = "Value must between 0 - $VALUE_MAX"
val VALUE_MAC_ADDR_ERROR = "INVALID MAC ADDRESS"
val VALUE_IP_ADDR_ERROR = "Should have valid IP with netmask \"\^\" separated"

var modeAdd = true
var inflater: LayoutInflater? = null

companion object {
    fun newInstance(nodeData: NodeDataSerializable): AddFlowMatchFragment{
        val args = Bundle()
        args.putSerializable(Constants.NODE_DATA, nodeData)

        var fragment = AddFlowMatchFragment()
        fragment.arguments = args
        return fragment
    }
}

override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
    savedInstanceState: Bundle?): View? {
    mView = inflater.inflate(R.layout.fragment_add_flow_match, container, false)
    this.inflater = inflater

    newFlow = (activity as AddFlowActivity).getFlow()

    if(arguments != null){
        var nodeList: ArrayList<String>? = null

        nodeData = arguments!!.getSerializable(Constants.NODE_DATA) as NodeDataSerializable
        for(i in nodeData!!.nodeSerializableData.indices){
            if(nodeData!!.nodeSerializableData.get(i).nodeId == newFlow?.nodeId){
                nodeList = nodeData!!.nodeSerializableData.get(i).nodeConnector
            }
        }
        arrayAdapterInport = ArrayAdapter(context, R.layout.item_spinner, nodeList)
    }

    setDefaultData()
    setData()

    var alertDialog = AlertDialog.Builder(context)
    var arrayAdapter = ArrayAdapter<String>(context, android.R.layout.simple_list_item_1)
    for(i in matchData.indices){
        arrayAdapter.add(matchData[i].propName)
    }

    alertDialog.setAdapter(arrayAdapter) { dialog, which ->
        val view = inflater.inflate(matchData[which].layoutView, null, false)
        val btnDelete = view.findViewById<ImageView>(R.id.iv_delete)
        val match = matchData[which]

        if(match.propId == tag_match_inport){
            val spinnerInport = view.findViewById<Spinner>(R.id.spinner_in_port)
            spinnerInport.adapter = arrayAdapterInport
        }
        btnDelete.setOnClickListener {
            (view.parent as LinearLayout).removeView(view)
        }
    }
}

```

```

        selectedMatchData.remove(match)
        matchData.add(match)
        arrayAdapter.add(match.propName)
        arrayAdapter.notifyDataSetChanged()
        if(arrayAdapter.count == 1){
            mView.btn_add_match.visibility = View.VISIBLE
        }
    }

    selectedMatchData.add(match)
    arrayAdapter.remove(match.propName)
    arrayAdapter.notifyDataSetChanged()
    matchData.remove(match)

    mView.ll_match.addView(view)
    dialog.dismiss()

    if(arrayAdapter.count == 0){
        mView.btn_add_match.visibility = View.GONE
    } else {
        mView.btn_add_match.visibility = View.VISIBLE
    }
}

mView.btn_add_match.setOnClickListener {
    alertDialog.show()
}

return mView
}

fun setData(){
    this.modeAdd = (activity as AddFlowActivity).getMode()

    (activity as AddFlowActivity).setStatus(false)
    mView.tv_device.text = newFlow!!.nodeId

    val inPortMatch = newFlow?.match?.inPort
    val ethernetType = newFlow?.match?.ethernetMatch?.ethernetType
    val macSource = newFlow?.match?.ethernetMatch?.ethernetSource
    val macDest = newFlow?.match?.ethernetMatch?.ethernetDestination
    val ipv4Source = newFlow?.match?.ipv4source
    val ipv4Dest = newFlow?.match?.ipv4destination

    if(modeAdd){
    } else {
        mView.btn_add_match.visibility = View.GONE
        if(inPortMatch != null){
            try {
                mView.spinner_in_port.visibility = View.VISIBLE
                var spinnerPos = arrayAdapterInport?.getPosition(inPortMatch)
                mView.spinner_in_port.setSelection(spinnerPos!!)
            } catch (e: Exception){
                addView(0)

                mView.spinner_in_port.visibility = View.VISIBLE
                println("inPortMatch : $inPortMatch")
                var spinnerPos = arrayAdapterInport?.getPosition(inPortMatch)
                println("inPortMatch spinner pos : $spinnerPos")
                mView.spinner_in_port.setSelection(spinnerPos!!)
            }
        }
    }
}

```

```

if(ethernetType != null){
    try{
        mView.et_ethernet_type.visibility = View.VISIBLE
        mView.et_ethernet_type.setText(ethernetType.type.toString())
    } catch (e: Exception){
        addView(1)
        mView.et_ethernet_type.visibility = View.VISIBLE
        mView.et_ethernet_type.setText(ethernetType.type.toString())
    }
}

if(macSource != null){
    try {
        mView.et_mac_source.visibility = View.VISIBLE
        mView.et_mac_source.setText(macSource.address)
    } catch (e: Exception){
        addView(2)
        mView.et_mac_source.visibility = View.VISIBLE
        mView.et_mac_source.setText(macSource.address)
    }
}

if(macDest != null){
    try{
        mView.et_mac_dest.visibility = View.VISIBLE
        mView.et_mac_dest.setText(macDest.address)
    } catch (e: Exception){
        addView(3)
        mView.et_mac_dest.visibility = View.VISIBLE
        mView.et_mac_dest.setText(macDest.address)
    }
}

if(ipv4Source != null){
    try {
        mView.et_ipv4_source.visibility = View.VISIBLE
        mView.et_ipv4_source.setText(ipv4Source)
    } catch (e: Exception){
        addView(4)
        mView.et_ipv4_source.visibility = View.VISIBLE
        mView.et_ipv4_source.setText(ipv4Source)
    }
}

if(ipv4Dest != null){
    try{
        mView.et_ipv4_dest.visibility = View.VISIBLE
        mView.et_ipv4_dest.setText(ipv4Dest)
    } catch (e: Exception){
        addView(5)
        mView.et_ipv4_dest.visibility = View.VISIBLE
        mView.et_ipv4_dest.setText(ipv4Dest)
    }
}
}
}

fun addView(which: Int){
    val view = inflater?.inflate(matchData[which].layoutView, null, false)
    val btnDelete = view?.findViewById<ImageView>(R.id.iv_delete)
    val match = matchData[which]

    btnDelete?.visibility = View.GONE

```

```

selectedMatchData.add(match)

if(match.propId == tag_match_inport){
    val spinnerInport = view?.findViewById<Spinner>(R.id.spinner_in_port)
    spinnerInport?.adapter = arrayAdapterInport
}

mView.ll_match.addView(view)
}

fun setDefaultData(){
    matchData.clear()
    selectedMatchData.clear()

    var inPortMatch = FlowProperties(tag_match_inport, "In port", R.layout.layout_match_in_port)
    matchData.add(inPortMatch)

    var ethernetType = FlowProperties(tag_match_etherntype, "Ethernet type",
R.layout.layout_match_etherntype)
    matchData.add(ethernetType)

    var macSource = FlowProperties(tag_match_macsource, "Source MAC",
R.layout.layout_match_mac_source)
    matchData.add(macSource)

    var macDest = FlowProperties(tag_match_macdest, "Destination MAC",
R.layout.layout_match_mac_dest)
    matchData.add(macDest)

    var ipv4Source = FlowProperties(tag_match_ipv4source, "IPv4 Source",
R.layout.layout_match_ipv4_source)
    matchData.add(ipv4Source)

    var ipv4Dest = FlowProperties(tag_match_ipv4dest, "IPv4 Destination",
R.layout.layout_match_ipv4_dest)
    matchData.add(ipv4Dest)
}

fun setFlow(): Flow {
    var isCompleted = true

    newFlow = (activity as AddFlowActivity).getFlow()

    var ethernetMatch: EthernetMatch? = null
    var match = Match(null, null, null, null)

    for(i in selectedMatchData.indices){
        when(selectedMatchData[i].propId){
            tag_match_inport -> {
                match.inPort = mView.spinner_in_port.selectedItem.toString()
            }
            tag_match_etherntype -> {
                if(!mView.et_etherntype.text.isNullOrEmpty() &&
!mView.et_etherntype.text.isNullOrBlank()){
                    try{
                        val ethernetTypeValue = mView.et_etherntype.text.toString()
                        mView.et_etherntype.error = null
                        var ethernetType = EthernetType(ethernetTypeValue)
                        if(ethernetMatch != null){
                            ethernetMatch?.ethernetType = ethernetType
                        } else {
                            ethernetMatch = EthernetMatch(null, null, ethernetType)
                        }
                    } catch (e: Exception){

```

```

        e.printStackTrace()
        mView.et_ethernet_type.error = VALUE_ERROR
        isCompleted = false
    }
} else {
    mView.et_ethernet_type.error = VALUE_CANT_BE_BLANK
    isCompleted = false
}

}

tag_match_macsource -> {
    if(!mView.et_mac_source.text.isNullOrEmpty() &&
!mView.et_mac_source.text.isNullOrEmpty()){
        val macSource = mView.et_mac_source.text.toString()
        val colonCount = macSource.length - macSource.replace(":", "").length
        if(macSource.length == 17 && colonCount == 5) {
            mView.et_mac_source.error = null
            var ethernetSource = EthernetSource(mView.et_mac_source.text.toString())
            if (ethernetMatch != null) {
                ethernetMatch?.ethernetSource = ethernetSource
            } else {
                ethernetMatch = EthernetMatch(ethernetSource, null, null)
            }
        } else {
            mView.et_mac_source.error = VALUE_MAC_ADDR_ERROR
            isCompleted = false
        }
    } else {
        mView.et_mac_source.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}

tag_match_macdest -> {
    if(!mView.et_mac_dest.text.isNullOrEmpty() &&
!mView.et_mac_dest.text.isNullOrEmpty()){
        val macDest = mView.et_mac_dest.text.toString()
        val colonCount = macDest.length - macDest.replace(":", "").length
        if(macDest.length == 17 && colonCount == 5) {
            mView.et_mac_dest.error = null
            var ethernetDestination = EthernetDestination(mView.et_mac_dest.text.toString())
            if(ethernetMatch != null){
                ethernetMatch?.ethernetDestination = ethernetDestination
            } else {
                ethernetMatch = EthernetMatch(null, ethernetDestination, null)
            }
        } else {
            mView.et_mac_dest.error = VALUE_MAC_ADDR_ERROR
            isCompleted = false
        }
    } else {
        mView.et_mac_dest.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}

tag_match_ipv4source -> {
    if(!mView.et_ipv4_source.text.isNullOrEmpty() ||
!mView.et_ipv4_source.text.isNullOrEmpty()){
        val ip = mView.et_ipv4_source.text.toString().trim()
        if(ip.contains("/")){
            val ipAddr = ip.substring(0, ip.lastIndexOf("/"))
            val netmask = ip.substring(ip.lastIndexOf("/") + 1)
            try{
                if(validIP(ipAddr) && netmask.toInt() in 1..32){
                    mView.et_ipv4_source.error = null
                }
            }
        }
    }
}

```

```

        match.ipv4source = ip
    } else {
        mView.et_ipv4_source.error = VALUE_IP_ADDR_ERROR
        isCompleted = false
    }
} catch (e: Exception){
    mView.et_ipv4_source.error = VALUE_IP_ADDR_ERROR
    isCompleted = false
}
} else {
    mView.et_ipv4_source.error = VALUE_IP_ADDR_ERROR
    isCompleted = false
}
} else {
    mView.et_ipv4_source.error = VALUE_CANT_BE_BLANK
    isCompleted = false
}
}
}
tag_match_ipv4dest -> {
    if(!mView.et_ipv4_dest.text.isNullOrEmpty() || !mView.et_ipv4_dest.text.isNullOrBlank()){
        val ip = mView.et_ipv4_dest.text.toString().trim()
        if(ip.contains("/")){
            val ipAddr = ip.substring(0, ip.lastIndexOf("/"))
            val netmask = ip.substring(ip.lastIndexOf("/") + 1)
            try{
                if(validIP(ipAddr) && netmask.toInt() in 1..32){
                    mView.et_ipv4_dest.error = null
                    match.ipv4destination = ip
                } else {
                    mView.et_ipv4_dest.error = VALUE_IP_ADDR_ERROR
                    isCompleted = false
                }
            } catch (e: Exception){
                mView.et_ipv4_dest.error = VALUE_IP_ADDR_ERROR
                isCompleted = false
            }
        } else {
            mView.et_ipv4_dest.error = VALUE_IP_ADDR_ERROR
            isCompleted = false
        }
    } else {
        mView.et_ipv4_dest.error = VALUE_CANT_BE_BLANK
        isCompleted = false
    }
}
}
}

match.ethernetMatch = ethernetMatch
newFlow?.match = match

if(isCompleted){
    (activity as AddFlowActivity).setDataStatus(true)
} else {
    (activity as AddFlowActivity).setDataStatus(false)
}

return newFlow!!
}

fun validIP(ip: String?): Boolean {

```

```

try {
    if (ip == null || ip.isEmpty()) {
        return false
    }

    val parts = ip.split("\\.".toRegex()).dropLastWhile { it.isEmpty() }.toTypedArray()
    if (parts.size != 4) {
        return false
    }

    for (s in parts) {
        val i = Integer.parseInt(s)
        if (i < 0 || i > 255) {
            return false
        }
    }
    if (ip.endsWith(".")) {
        return false
    }

    return true
} catch (nfe: NumberFormatException) {
    return false
}
}
}

```

34. AddFlowReviewFragment.kt

```

class AddFlowReviewFragment : Fragment() {

    lateinit var mView: View

    lateinit var jsonInString: String
    lateinit var newFlow: Flow

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        mView = inflater.inflate(R.layout.fragment_add_flow_review, container, false)
        newFlow = (activity as AddFlowActivity).getFlow()

        setData()

        return mView
    }

    fun setData(){
        val nodeId = newFlow.nodeId
        newFlow.nodeId = null

        var flowData: MutableList<Flow> = ArrayList()
        flowData.add(newFlow)
        val flowDataSent = FlowDataSent(flowData)

        jsonInString = Gson().toJson(flowDataSent)

        mView.tv_json.text = jsonInString
        mView.rv_json.bindJson(jsonInString)

        val px12 = Math.round(TypedValue.applyDimension(
            TypedValue.COMPLEX_UNIT_DIP, 12f, resources.getDisplayMetrics()))

        mView.rv_json.setTextSize(px12)

        println("setFlow review in port data : ${newFlow?.match?.inPort}")
    }
}

```

```

        newFlow.nodeId = nodeId
    }
}

```

35. AddFlowSelectDeviceFragment.kt

```

class AddFlowSelectDeviceFragment : Fragment() {
    lateinit var mView: View

    companion object {
        fun newInstance(nodeList: ArrayList<String>): AddFlowSelectDeviceFragment {
            val args = Bundle()
            args.putSerializable(Constants.NODE_LIST, nodeList)
            val fragment = AddFlowSelectDeviceFragment()
            fragment.arguments = args
            return fragment
        }
    }

    override fun onCreateView(inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?): View? {
        mView = inflater.inflate(R.layout.fragment_add_flow_select_device, container, false)
        activity?.title = "Add Flow: Select Device"

        if (arguments != null) {
            val nodeList = arguments!!.getSerializable(Constants.NODE_LIST) as ArrayList<String>
            val adapter = ArrayAdapter<String>(context, R.layout.item_spinner, nodeList)
            mView.spinner_select_device.adapter = adapter
        }

        return mView
    }

    fun setFlow(): Flow {
        val name = mView.spinner_select_device.selectedItem.toString()
        val newFlow = (activity as AddFlowActivity).getFlow()
        newFlow.nodeId = name
        (activity as AddFlowActivity).setStatus(true)
        return newFlow
    }
}

```

36. FlowDetailsActivity.kt

```

class FlowDetailsActivity : AppCompatActivity() {

    lateinit var flow: Flow
    var flowType: String? = null
    lateinit var bundle: Bundle
    lateinit var restAdapter: RestAdapter

    lateinit var nodeList: ArrayList<String>
    var nodeData: NodeDataSerializable? = null

    var position: Int? = null

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_flow_details)
        title = "Flow Details"
        bundle = intent.extras

        restAdapter = RestAdapter(this)

        supportActionBar?.setDisplayHomeAsUpEnabled(true)
    }
}

```



```

    position = intent.getIntExtra(Constants.FLOW_POSITION, 999)

    flow = bundle.getSerializable(Constants.OBJECT_FLOW) as Flow
    showData()
}

fun showData(){
    // General Properties
    var nodeId = flow.nodeId
    flowType = flow.flowType
    println("flowType = $flowType")
    if(flowType.isNullOrEmpty()){
        flowType = Constants.DATA_TYPE_OPERATIONAL
    }
    val table = flow.tableId
    val flowId = flow.id
    val flowName = flow.flowName
    val priority = flow.priority
    val hardTimeout = flow.hardTimeout
    val idleTimeout = flow.idleTimeout
    val cookie = flow.cookie

    println("isAvailableInConfig flowtype = $flowType")

    // Match
    val ethernetType = flow.match?.ethernetMatch?.ethernetType?.type
    val inPort = flow.match?.inPort
    val sourceMac = flow.match?.ethernetMatch?.ethernetSource?.address
    val destMac = flow.match?.ethernetMatch?.ethernetDestination?.address
    val ipSource = flow.match?.ipv4source
    val ipDestination = flow.match?.ipv4destination
    val outputPortData: MutableList<OutputPort> = ArrayList()

    // Action
    var controllerMaxLength: Int? = null
    var actionDrop = false
    var actionNormalMaxLength: Int? = null
    var actionFlood = false
    var actionFloodAll = false

    var actionData: MutableList<Action>? = ArrayList()
    if(flow.instructions != null && flow.instructions?.instruction!![0].applyActions != null){
        actionData = flow.instructions?.instruction!![0].applyActions?.actionData
        for(i in actionData?.indices!!){
            if(actionData[i].outputAction?.outputNodeConnector.equals("CONTROLLER")){
                controllerMaxLength = actionData[i].outputAction?.maxLength
            } else if(actionData[i].outputAction?.outputNodeConnector.equals("NORMAL")){
                actionNormalMaxLength = actionData[i].outputAction?.maxLength
            } else if(actionData[i].outputAction?.outputNodeConnector.equals("FLOOD")){
                actionFlood = true
            } else if(actionData[i].outputAction?.outputNodeConnector.equals("FLOOD_ALL")){
                actionFloodAll = true
            } else if(actionData[i].dropAction != null){
                actionDrop = true
            } else {
                // var outputPort = inPort?.substring(0, inPort.lastIndexOf(":")+1) +
                actionData[i].outputAction?.outputNodeConnector
                var outputPort = nodeId + ":" + actionData[i].outputAction?.outputNodeConnector
                var outputPortMaxLength = actionData[i].outputAction?.maxLength

                outputPortData.add(OutputPort(outputPort, outputPortMaxLength))
            }
        }
    }
}

```

```

    } else {
        ll_actions.visibility = View.GONE
    }

    tv_device.text = modelId

    if(flowType != null){
        tv_flow_type.text = flowType
    } else {
        ll_flow_type.visibility = View.GONE
    }

    if(table != null){
        tv_table.text = table.toString()
    } else {
        ll_table.visibility = View.GONE
    }

    if(flowId != null){
        tv_id_flow.text = flowId
    } else {
        ll_id.visibility = View.GONE
    }

    if(flowName != null){
        tv_flow_name.text = flowName
    } else {
        ll_flow_name.visibility = View.GONE
    }

    if(priority != null){
        tv_priority.text = priority.toString()
    } else {
        ll_priority.visibility = View.GONE
    }

    if(hardTimeout != null){
        tv_hard_timeout.text = hardTimeout.toString()
    } else {
        ll_hard_timeout.visibility = View.GONE
    }

    if(idleTimeout != null){
        tv_idle_timeout.text = idleTimeout.toString()
    } else {
        ll_idle_timeout.visibility = View.GONE
    }

    if(cookie != null){
        tv_cookie.text = cookie.toString()
    } else {
        ll_cookie.visibility = View.GONE
    }

    if(ethernetType != null){
        tv_ethernet_type.text = ethernetType.toString()
    } else {
        ll_ethernet_type.visibility = View.GONE
    }

    if(inPort != null){
        tv_in_port.text = inPort.toString()
    } else {
        ll_in_port.visibility = View.GONE
    }

```

```

}

if(sourceMac != null){
    tv_source_mac.text = sourceMac
} else {
    ll_source_mac.visibility = View.GONE
}

if(destMac != null){
    tv_dest_mac.text = destMac
} else {
    ll_dest_mac.visibility = View.GONE
}

if(ipSource != null){
    tv_ip_source.text = ipSource
} else {
    ll_ip_source.visibility = View.GONE
}

if(ipDestination != null){
    tv_ip_dest.text = ipDestination
} else {
    ll_ip_dest.visibility = View.GONE
}

if(controllerMaxLength != null){
    tv_controller_maximum_length.text = controllerMaxLength.toString()
} else {
    ll_controller.visibility = View.GONE
}

if(outputPortData.size != 0){
    tv_output_port.text = outputPortData[0].outputPort
    tv_output_port_maximum_length.text = outputPortData[0].outputMaxLength.toString()

    if(outputPortData.size > 1){
        tv_output_port_2.text = outputPortData[1].outputPort
        tv_output_port_2_maximum_length.text = outputPortData[1].outputMaxLength.toString()
    } else {
        ll_output_port_2.visibility = View.GONE
    }
} else {
    ll_output_port.visibility = View.GONE
    ll_output_port_2.visibility = View.GONE
}

if(actionDrop){
    ll_action_drop.visibility = View.VISIBLE
} else {
    ll_action_drop.visibility = View.GONE
}

if(actionNormalMaxLength != null){
    tv_normal_maximum_length.text = actionNormalMaxLength.toString()
} else {
    ll_action_normal.visibility = View.GONE
}

if(actionFlood){
    ll_action_flood.visibility = View.VISIBLE
} else {
    ll_action_flood.visibility = View.GONE
}

```

```

        if(actionFloodAll){
            ll_action_flood_all.visibility = View.VISIBLE
        } else {
            ll_action_flood_all.visibility = View.GONE
        }
    }

    fun checkInConfig(){
        var isAvailableInConfig = false
        var progressDialog = ProgressDialog(this)
        progressDialog.setMessage("Please wait ..")
        progressDialog.setCancelable(false)
        progressDialog.setCanceledOnTouchOutside(false)
        progressDialog.show()
        restAdapter.getEndPoint().getFlowsConfig(flow.nodeId.toString(), "0")
            .enqueue(object : Callback<FlowTableData>{
                override fun onResponse(call: Call<FlowTableData>?, response:
                Response<FlowTableData>?) {
                    if(response?.isSuccessful!!){
                        if(response.body()?.table?.get(0) != null &&
                        response.body()?.table?.get(0)?.flowData?.size != 0 &&
                        response.body()?.table?.get(0)?.flowData?.size != null){
                            var flowDataList = response.body()?.table?.get(0)?.flowData
                            for (i in flowDataList?.indices!!){
                                if (flowDataList[i].id == flow.id){
                                    println("isAvailableInConfig == TRUE")
                                    isAvailableInConfig = true
                                    break
                                }
                            }
                        }
                    }
                    if(isAvailableInConfig){
                        flow.flowType = Constants.DATA_TYPE_CONFIG
                        println("isAvailableInConfig CONFIG")
                    } else {
                        println("isAvailableInConfig != TRUE")
                    }
                    progressDialog.dismiss()
                    isSafeToDelete()
                } else {
                    Log.e("Failed checkInConfig, ", "response unsuccessful")
                    progressDialog.dismiss()
                    Toast.makeText(this@FlowDetailsActivity, "Error", Toast.LENGTH_SHORT).show()
                }
            })

        override fun onFailure(call: Call<FlowTableData>?, t: Throwable?) {
            Log.e("Failed checkInConfig, ", t?.message)
            progressDialog.dismiss()
            Toast.makeText(this@FlowDetailsActivity, "Error", Toast.LENGTH_SHORT).show()
        }
    })

    fun isSafeToDelete(){
        if(flow.flowType == Constants.DATA_TYPE_OPERATIONAL && isMatchNull()){
            var alertDialog = AlertDialog.Builder(this)
            alertDialog.setMessage("This flow doesn't has any match, it will delete all operational flow
            on ${flow.nodeId}. Continue?")
            alertDialog.setPositiveButton("Continue") { dialog, which ->
                deleteFlow()
                dialog.dismiss()
            }
        }
    }

```

```

        alertDialog.setNegativeButton("Cancel") { dialog, _ ->
            dialog.dismiss()
        }
        alertDialog.show()
    } else {
        deleteFlow()
    }
}

fun isMatchNull(): Boolean{
    var isMatchNull = true
    var flowMatch = flow.match
    if(flowMatch?.ethernetMatch != null){
        isMatchNull = false
    }
    if(flowMatch?.inPort != null){
        isMatchNull = false
    }
    if(flowMatch?.ipv4source != null){
        isMatchNull = false
    }
    if(flowMatch?.ipv4destination != null){
        isMatchNull = false
    }
    return isMatchNull
}

fun deleteFlow(){
    var progressDialog = ProgressDialog(this)
    progressDialog.setMessage("Deleting ..")
    progressDialog.setCancelable(false)
    progressDialog.setCanceledOnTouchOutside(false)
    progressDialog.show()
    if(flow.flowType == Constants.DATA_TYPE_CONFIG){
        restAdapter.getEndPoint()
            .deleteFlowConfig(flow.nodeId!!, flow.id!!)
            .enqueue(object : retrofit2.Callback<ResponseBody>{
                override fun onResponse(call: Call<ResponseBody>?, response:
                Response<ResponseBody>?) {
                    progressDialog.dismiss()
                    if(response!!.isSuccessful){
                        Toast.makeText(this@FlowDetailsActivity, "Flow deleted",
                        Toast.LENGTH_SHORT).show()
                    } else {
                        Toast.makeText(this@FlowDetailsActivity, "Delete failed",
                        Toast.LENGTH_SHORT).show()
                    }
                    println("onActivityResult, delete setResult OK")
                    setResult(Activity.RESULT_OK)
                    finish()
                }

                override fun onFailure(call: Call<ResponseBody>?, t: Throwable?) {
                    progressDialog.dismiss()
                    Toast.makeText(this@FlowDetailsActivity, "Delete failed",
                    Toast.LENGTH_SHORT).show()
                }
            })
    } else {
        var input = Input(flow.match, 0, flow.priority, "/opendaylight-
        inventory:nodes/opendaylight-inventory:node[opendaylight-inventory:id='${flow.nodeId}']")
        var inputData = InputData(input)
        restAdapter.getEndPoint()
            .deleteFlowOperational(inputData)
    }
}

```

```

        .enqueue(object : Callback<ResponseBody>{
            override fun onResponse(call: Call<ResponseBody>?, response:
Response<ResponseBody>?) {
                progressDialog.dismiss()
                if(response!!.isSuccessful){
                    Toast.makeText(this@FlowDetailsActivity, "Flow deleted",
Toast.LENGTH_SHORT).show()
                    println("onActivityResult, delete setResult OK")
                    setResult(Activity.RESULT_OK)
                    finish()
                } else {
                    Toast.makeText(this@FlowDetailsActivity, "Delete failed",
Toast.LENGTH_SHORT).show()
                }
            }

            override fun onFailure(call: Call<ResponseBody>?, t: Throwable?) {
                progressDialog.dismiss()
                Toast.makeText(this@FlowDetailsActivity, "Delete failed",
Toast.LENGTH_SHORT).show()
            }
        })
    }
}

override fun onOptionsItemSelected(item: MenuItem?): Boolean {
    when(item?.itemId){
        android.R.id.home -> onBackPressed()
        R.id.action_delete -> {
            var alertDialog = AlertDialog.Builder(this)
            alertDialog.setMessage("Are you sure to delete flow ${flow.id} ?")
            alertDialog.setPositiveButton("Yes") { dialog, which ->
                checkInConfig()
                dialog.dismiss()
            }
            alertDialog.setNegativeButton("Cancel") { dialog, _ ->
                dialog.dismiss()
            }
            alertDialog.show()
        }
        R.id.action_edit -> openEditFlowActivity()
    }
    return true
}

fun openEditFlowActivity(){
    nodeList = intent.getStringArrayListExtra(Constants.NODE_LIST)
    nodeData = intent.getSerializableExtra(Constants.NODE_DATA) as NodeDataSerializable

    if(nodeList.size > 0){
        var extras = Bundle()
        extras.putSerializable(Constants.NODE_DATA, nodeData)
        extras.putSerializable(Constants.OBJECT_FLOW, flow)

        val intent = Intent(this@FlowDetailsActivity, AddFlowActivity::class.java)
        intent.putStringArrayListExtra(Constants.NODE_LIST, nodeList)
        intent.putExtras(extras)
        intent.putExtra(Constants.ADD_MODE, Constants.MODE_EDIT)
        startActivity(intent)
    } else {
        Toast.makeText(this, "An error has occurred", Toast.LENGTH_SHORT).show()
    }
}

```

```

data class OutputPort(
    var outputPort: String?,
    var outputMaxLength: Int?
)

override fun onCreateOptionsMenu(menu: Menu?): Boolean {
    menuInflater?.inflate(R.menu.menu_flow_details, menu)
    if(flowType == Constants.DATA_TYPE_OPERATIONAL){
        menu?.findItem(R.id.action_edit)?.setVisible(false)
        println("isAvailableInConfig onCreateOptionsMenu == DATA_TYPE_OPERATIONAL")
    } else {
        println("isAvailableInConfig onCreateOptionsMenu != DATA_TYPE_OPERATIONAL")
    }
    return super.onCreateOptionsMenu(menu)
}

override fun onBackPressed() {
    setResult(Activity.RESULT_CANCELED)
    super.onBackPressed()
}
}

```

37. FlowListAdapter.kt

```

class FlowListAdapter(private var context: Context,
    private var fragment: Fragment,
    private var dataList: MutableList<Flow>,
    private var nodeList: ArrayList<String>,
    private var nodeData: NodeDataSerializable?):
    RecyclerView.Adapter<FlowListAdapter.FlowListViewHolder>(){

    var isStatisticShown = false

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): FlowListViewHolder {
        val view = LayoutInflater.from(parent.context).inflate(R.layout.item_flow, parent, false)

        return FlowListViewHolder(view)
    }

    override fun getItemCount(): Int {
        return dataList.size
    }

    override fun onBindViewHolder(holder: FlowListViewHolder, position: Int) {
        println("isStatisticShown : $isStatisticShown")
        if(isStatisticShown){
            holder.itemView.layout_statistic.visibility = View.VISIBLE
        } else {
            holder.itemView.layout_statistic.visibility = View.GONE
        }

        holder.itemView.tv_id_node.text = dataList[position].nodeId.toString()
        holder.itemView.tv_id_flow.text = dataList[position].id.toString()
        holder.itemView.tv_flow_type.text = dataList[position].flowType
        holder.itemView.tv_priority.text = dataList[position].priority.toString()
        holder.itemView.tv_packet_count.text =
            dataList[position].flowStatistics?.packetCount.toString()
        holder.itemView.tv_byte_count.text = dataList[position].flowStatistics?.byteCount.toString()

        holder.itemView.card_flow_item.setOnClickListener { openDetailsFlow(dataList[position],
            position) }
    }
}

```

```

inner class FlowListViewHolder(view: View): RecyclerView.ViewHolder(view)

fun openDetailsFlow(flow: Flow, position: Int){
    val bundle = Bundle()
    bundle.putSerializable(Constants.OBJECT_FLOW, flow)
    val intent = Intent(context, FlowDetailsActivity::class.java)
    println("flowType openDetailsFlow = ${flow.flowType} ")
    if(flow.flowType == Constants.DATA_TYPE_CONFIG){
        var extras = Bundle()
        extras.putSerializable(Constants.NODE_DATA, nodeData)
        extras.putInt(Constants.FLOW_POSITION, position)
        intent.putExtras(extras)
        intent.putStringArrayListExtra(Constants.NODE_LIST, nodeList)
    } else {

    }
    intent.putExtras(bundle)
    println("onActivityResult, startActivityForResult FlowListAdapter ")
    fragment.startActivityForResult(intent, Constants.RequestCode.OPEN_DETAILS_ACTIVITY)
}

fun setData(dataList: MutableList<Flow>){
    this.dataList.clear()
    this.dataList.addAll(dataList)
    this.dataList.sortWith(compareBy({it.nodeId}))
    notifyDataSetChanged()
}

fun addData(dataList: MutableList<Flow>,
            nodeList: ArrayList<String>,
            nodeData: NodeDataSerializable?){
    this.nodeList = nodeList
    this.nodeData = nodeData
    this.dataList.addAll(dataList)

    this.dataList.sortWith(compareBy({it.nodeId}, {it.priority}))
    this.dataList.reverse()
    this.dataList.sortWith(compareByDescending { it.flowType })
    notifyDataSetChanged()
}

fun showStatistic(isShown: Boolean){
    this.isStatisticShown = isShown
    notifyDataSetChanged()
}

fun clearData(){
    this.dataList.clear()
}
}

```


LAMPIRAN – 6. Bukti publikasi hasil proyek akhir

[Simet] Pernyataan Naskah Inbox x

Tri Listyorini <no-reply.jurnal@umk.ac.id> to me 2:21 PM (1 minute ago)

Indonesian > English [Translate message](#) [Turn off for: Indonesian x](#)

Edityo Swakresna Edityomurti:

Terima kasih untuk menyerahkan manuskrip, "PENGEMBANGAN APLIKASI MANAJEMEN OPENFLOW TABLE BERBASIS ANDROID PADA SOFTWARE DEFINED NETWORK MENGGUNAKAN OPENDAYLIGHT CONTROLLER" untuk Simetris: Jurnal Teknik Mesin, Elektro dan Ilmu Komputer. Dengan sistem manajemen jurnal online yang kami gunakan, Anda akan bisa melacak kemajuan naskah dalam proses editorial dengan login ke web site jurnal:

URL Manuskrip:
<http://jurnal.umk.ac.id/index.php/simet/author/submission/2422>
 Username: edityomurti

Jika Anda mempunyai pertanyaan, silakan hubungi saya. Terima kasih untuk mempertimbangkan jurnal ini sebagai tempat untuk karya Anda.


SIMETRIS Jurnal Teknik Mesin, Elektro dan Ilmu Komputer

ISSN 2252-4983 (Print)
 ISSN 2549-3108 (Online)

[Beranda](#) [Tentang Kami](#) [Beranda Pengguna](#) [Cari](#) [Terkini](#) [Arsip](#) [Informasi](#) [Tim Editorial](#) [Register](#)

Beranda > Pengguna > Penulis > Naskah > Penyerahan Aktif

Penyerahan Aktif

Penyerahan naskah sukses. Terima kasih atas partisipasi Anda untuk bergabung bersama Simetris: Jurnal Teknik Mesin, Elektro dan Ilmu Komputer.

- **Penyerahan Aktif**

KONFERENSI KAMI




SIMETRIS Jurnal Teknik Mesin, Elektro dan Ilmu Komputer

ISSN 2252-4983 (Print)
 ISSN 2549-3108 (Online)

[Beranda](#) [Tentang Kami](#) [Beranda Pengguna](#) [Cari](#) [Terkini](#) [Arsip](#) [Informasi](#) [Tim Editorial](#) [Register](#)

Beranda > Pengguna > Penulis > Penyerahan Aktif

Penyerahan Aktif

AKTIF ARSIP

ID	MM-DD PENGAJUAN	BAGIAN	PENULIS	JUDUL	STATUS
2422	07-26	INF	Edityomurti, Rosyid	PENGEMBANGAN APLIKASI MANAJEMEN OPENFLOW TABLE BERBASIS...	Menunggu Penugasan

KONFERENSI KAMI

