AGH

AGH UNIVERSITY OF SCIENCE AND TECHNOLOGY

FACULTY OF INFORMATION TECHNOLOGY, ELECTRONICS AND TELECOMMUNICATIONS

TELECOMMUNICATIONS DEPARTMENT

ENGINEER THESIS

DEVELOPMENT OF INTERNET APPLICATION TO STEER AND MANAGE LAN NETWORKS

Opracowanie aplikacji internetowej do kontroli i zarządzania sieciami LAN.

Author: **Radosław Skałbania**

Field of study: Electronics and Telecommunication

Type of studies: Stationary

Work supervisor: D.Sc. Jerzy Domżał

Cracow, 2020

OŚWIADCZENIE STUDENTA

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (tj. Dz.U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym (tj. Dz.U. z 2018 r. poz. 1668, z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz.U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

(*czytelny podpis studenta)*

Table of Contents

1. Introduction

As we all probably know, in the times we live there is a need for constant, stable and fast connection between all devices around the globe.

If we are day to day citizens we will be mostly using an Internet network which is managed by multiple internet service providers and depends on the application service we are using - multiple companies. However, if we are employees in those, we probably will not be using the public Internet as it is dangerous and sometimes unpredictable, instead such businesses are creating local area networks which are connected with each other by secured virtual private network connections. As different firms handle different services in those local area networks, they for sure have to be stable to keep incomes on the same level and to not let companies go bankrupt in case of some network outages.

Those companies are hiring multiple people called network engineers to support and maintain networks. In an old-fashioned way, if such an engineer will hear about a network incident, he will firstly try to locate the network device (assuming he works in a huge company which delivers IT services to multiple places around the world) by using multiple network diagrams (which are static and sometimes not up to date), then obtain IP address and finally log in to network device and proceed with troubleshooting. Same process goes to configuration of such devices. In a new-fashioned way, people came up with an idea to centralize the whole network management into a single point of view - commonly called *Network Controller*[1]. Network Controller is a software based infrastructure solution, dedicated for managing and configuring network devices. All interactions between end network devices and network engineers must go via this controller. Basic architecture splits up into two main parts - *Northbound API*[2] and *Southbound API*[3]. First one is responsible for handling interactions with the user, it can be written as a *REST API*[4] or simple web based View[5]. Second one is responsible for communicating with end network devices via multiple network protocols such as SSH, Telnet, NETCONF, RESTCONF or SNMP. This project will be focused on implementing mentioned Network Controller.

Now, when we have already understood the purpose of this project, let's move on to the next chapter.

2. Tools and Frameworks

In this chapter we will focus on understanding tools, protocols, frameworks and python modules, which were used to develop web based Network Controller. First, let's talk about the *Python*[6] programming language.

2.1 Python

Python is an interpreted, dynamically typed, high-level, multiple purpose programming language developed by Dutch programmer Guido van Rossum in 1991. The reason why this project was mainly developed with help of this tool is that it emphasizes code readability as in the code syntax there is no place for curly brackets while defining classes, functions or even simple boolean expressions. Instead, indentation levels are used. As per PEP 8 - official guide for Python code each indentation block is built from four spaces. Next reason why this tool has been chosen is that it has numerous open source modules. Based on the data from *Python Package Index*[7] (11.12.2020) there are 277136 open source projects.

This language constructs an object-oriented approach to help developers write clear, logical code for small and large-scale applications. We can write simple Python scripts which can be used to consume REST API, configure network devices, sniff packets from the network and many many more... or, we can use it to deploy a web server and create our own web application. Based on the software company TIOBE which published a list of most popular programming languages, Python has made third place as in December 2020. Python splits up into two main versions - Python 2 and Python 3. There are few main differences between those versions which can be found in a syntax or Python standard library. As the use of both versions changed dramatically, support for the 2.x versions ended in January 2020. For the record, this project has been written in Python 3.8.1.

As in next chapters when we talk about implementation and project structure there for sure will be some code snippets, so lets firstly understand the basics of Python syntax.

2.1.1 Python Syntax

The word syntax is a set of rules, principles and processes that govern the structure of sentences. It applies as well to the programming world. Python syntax was developed with a thought to enhance code readability and named after Monty Python (the British comedy troupe). The easiest way to understand Python is by reading it’s core philosophy - The Zen of Python.

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

We already spoke about indentations, let's make clear some basic statements and flow control techniques:

- The *if* statement which executes a block of code after evaluating the condition and checking if it's true, along with *elif* and *else.*

```
>>> x = 5
>>> if x > 5:
...     print('x is greater than 5')
... elif x < 5:
...     print('x is smaller than 5')
... else:
...     print('x equals to 5')
x equals to 5
```

- The *for* statement which iterates through a given iterable object and assigns it to local temporary variable, used in specified block.

```
>>> for i in range(5):
...     print(f'i is now {i}')
i is now 0
i is now 1
i is now 2
i is now 3
i is now 4
```

- The *while* statement is similar both to *if* and *for* statements as in every iteration it evaluates the condition and iterates further only if it's true.

```
>>> x =  5
>>> while x > 0:
...     print(f'x is now {x}')
...     x -= 1
x is now 5
x is now 4
x is now 3
x is now 2
x is now 1
```

- The *try* statement which allows code flow to be handled with possible exceptions caught by *except* statement. The *else* block executes only if no exceptions were raised. The *finally* statement executes no matter what happened before.

```
>>> try:
...     print(1 / 0)
... except Exception as exception:
...     print(exception)
... else:
...     print('This block will not be executed.')
... finally:
...     print('Tried to divide by 0 but got exception.')

division by zero
Tried to divide by 0 but got an exception.
```

- The *def* statement which defines a function or a method.

```
>>> def add(x, y):
...     return x + y

>>> add(10, 20)
```

```
30
```

- The *class* statement is essential Python syntax for defining a class. It executes a block of code and attaches local namespaces to a class. To define a constructor we use a (*dunder*) __*init*__ method.

```
>>> class Device:
...     def __init__(self, hostname, ip_address):
...         self.hostname = hostname
...         self.ip_address = ip_address

>>> router = Device('EdgeRouter1', '192.168.8.1')

>>> router
<__main__.Device object at 0x1072495e0>

>>> router.hostname
'EdgeRouter1'

>>> router.ip_address
'192.168.8.1'
```

- The *return* statement is responsible for returning output when a given function or method is called.

- The *import* statement is used to import modules whose functions, classes and variables can be used in the current program.

  Now, when we understood basic concepts of Python - how it looks like and works, lets move to the next chapter.

2.2 Django framework

Django is a high level Python Web framework which is a core tool in this project. To better understand how Django works lets firstly talk about *MVC*[8] (model view controller) design pattern.

2.2.1 MVC

Model View Controller design pattern consist from three main parts:

- **Model** - This part is responsible for designing databases and communicating with Controller as well as retrieving and manipulating data.
- **View** - This part is responsible for presenting the frontend logic (rendering HTML templates/data) to the end user. Based on the user input it communicates with the Controller.
- **Controller** - This part is responsible for handling all the backend logic / requests and based on the user input, it communicates with the correct Model and manages data, then it sends back return output to the View part.

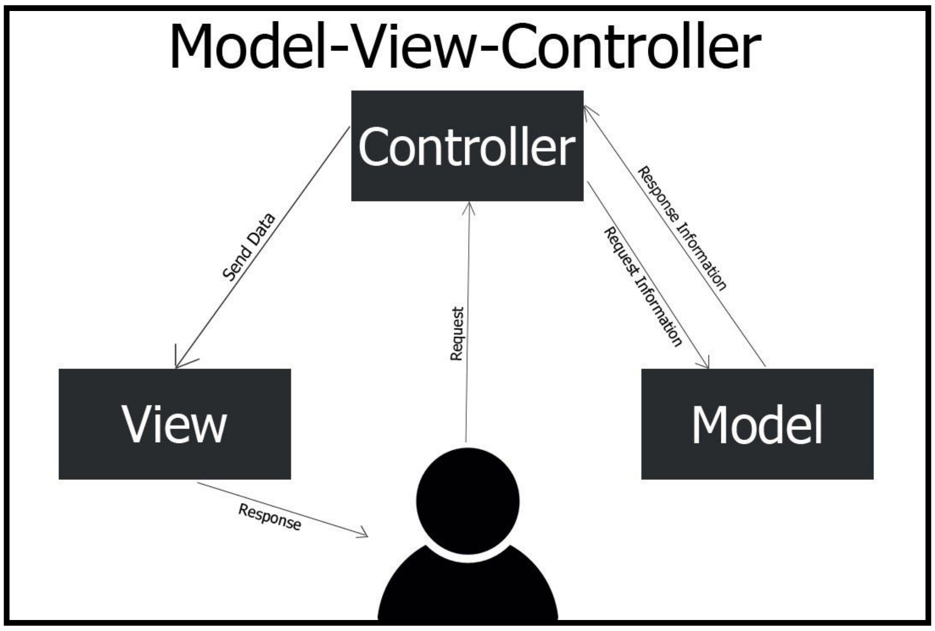


Figure 1 *- Model View Controller.*

Now, when we have a basic knowledge of what MVC is, let's quickly look into HTTP and its methods.

2.2.2 HTTP

HTTP[9] stands for hypertext transfer protocol and it is the most commonly used application layer protocol for transferring data between web servers and clients (in this case web browsers). It implements few methods which user and server can interact with but in this case we will talk only about two of them:

- GET - it requests from the server a specified in the URL resource. URL stands for Uniform Resource Locator. Those types of requests can be with or without additional parameters. Simple GET requests issued by a client from web browser to an AGH server:
  http://www.iet.agh.edu.pl/pl/
  If all worked correctly the server respond to the client with HTML data and status code 200. Finally the web browser is rendering a given HTML template and returns back requested web page.

- POST - these requests are used when the client is asking server to perform some changes on it such as sending a form with user data when creating an account. If all worked correctly, server creates a new entry in it’s database and returns status code 200.

Now, when we know what HTTP is and which methods it can use, we can move on to explaining how Django works itself.

2.2.3 Django

When the user issue a HTTP request with specified URL to the Django server, it firstly goes to the Controller (Django backend) and matches given URL with those which were implemented and if it finds one, it routes the request to the View function and depends of the given HTTP method it communicates back with Controller which is retrieving and/or manipulating in the Model (Django database). Then, data goes back to the View block via Controller and renders it out to the user.

Now, when we have met how Django works on a basic level lets move to more advanced topics such as web sockets and asynchronous tasks. Both were used in this project and will be described in next chapters.

2.2.4 Web Sockets - Django Channels

This module extends capabilities of basic Django framework with handling not only HTTP traffic but also protocols, which require long-running connections such as WebSockets, MQTT or chatbots. However in this project, we will focus only on WebSockets as we will see further it will be used to establish a web terminal SSH connection between Network Controller and network device. To better understand Django Channels[10], let's firstly explain what WebSocket is.

WebSocket[11] is a connection-oriented protocol as it works with support of TCP (Transmission Control Protocol). The first thing it performs is establishing a full-duplex bidirectional connection by a 3 way handshake. In this way both client and server agree to use a WebSocket protocol and become event-triggered asynchronous connection. Now we cleared the basic concepts of WebSockets, lets move on to explaining Django Channels itself.

Django Channels implements WebSockets protocol from server perspective, meaning, it handles the bidirectional asynchronous connection with the client which, in this project is WebSocket implemented with JavaScript as we will see further. Initially to open this type of connection client has to make a HTTP request to the WebSocket endpoint exposed by developer, then an asynchronous consumer starts on the server and tries to establish mentioned TCP connection with the client. While we warmed up the asynchronous topic, let's proceed to asynchronous tasks with Django Celery.

2.2.5 Asynchronous Tasks - Django Celery and RabbitMQ

While this project is mostly focused on monitoring local area networks with our Network Controller we do not want to run all tasks synchronously, on a single thread as it is not user friendly. When Django controller receives a HTTP request from the user, it goes to the specific view function/class (if given URL matched routes specified by developer) and then executes queries or process data. While the user waits for response from the server she/he is unable to perform different operations. To resolve such an issue asynchronous tasks come to play.

In this project, asynchronous tasks were handled by using worker - Django Celery[12] and message broker - RabbitMQ[13]. Initially Django sends a task to Django Celery which can be executed on different threads, periodically or on demand - all in hands of the developer. However, between Django and Django Celery there is also mentioned message broker RabbitMQ, which acts as a task queue that can take tasks from many different technologies and utilize them into one standardized technology such as Django Celery. So to make things clear, user creates a task which is firstly sent to message broker, then formatted and properly utilized. It finally ends up in the worker queue which executes a given task on a different thread than main Django server. As we will see later in next chapters, it indeed does the job.

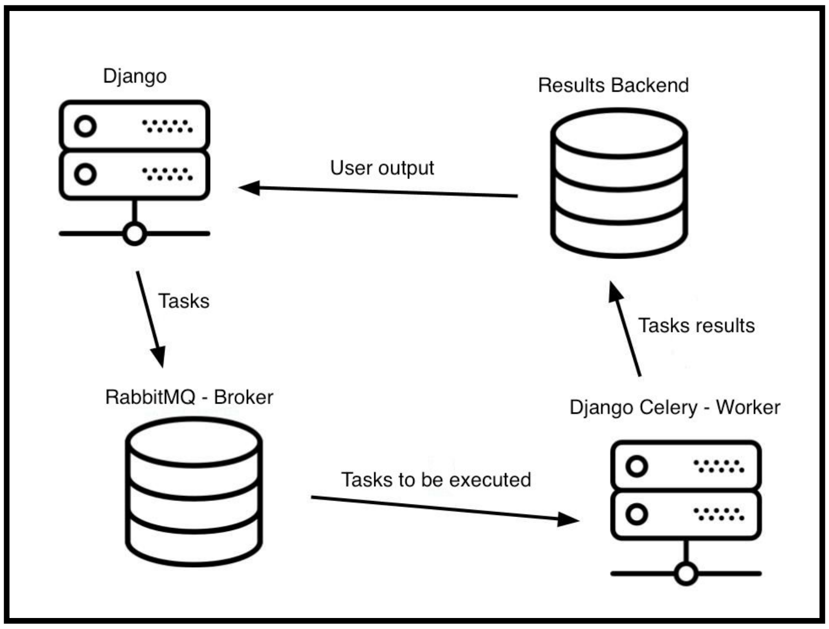


Figure 2 *- Handling Tasks Asynchronously.*

2.3 Secure Shell - SSH

This protocol ensures fully encrypted TCP connection, and is widely used in establishing virtual terminal sessions with end devices. To allow such communication between devices they both need to have installed SSH[14] server or client on a local machine as it works on Application layer with default TCP port 22. SSH handles traffic encryption with AES (Advanced Encryption Standard). It uses RSA cryptography algorithm to generate both public and private keys. Users can authenticate to remote devices by using simple password or key-pairs. In this chapter, we will focus on two Python libraries which are using SSH to programmatically access network devices previously mentioned as a Southbound API.

2.3.1 Netmiko

Netmiko is a Python open-source module, created by Kirk Byers which allows a developer to create programs for network automation. It can connect to devices using Telnet or SSH, however in this project SSH has been used as it is a more secure option. As we will see in the further chapter, this tool has been used to establish a web based SSH terminal session.

To understand it more clearly, let’s look at the example below, which presents us retrieving information about all interfaces from the network device.

```
# importing Connect Handler class from netmiko module.
>>> from netmiko import ConnectHandler

# creating a dictionary with device parameters and it’s access  details.
>>> cisco_881 = {
...      'device_type': 'cisco_ios',
...      'host':   '10.10.10.10',
...      'username': 'test',
...      'password': 'password',
...     'secret' : 'secret_password'
... }

# applying all parameters to ConnectHandler constructor.
>>> connection = ConnectHandler(**cisco_881)

# executing ‘sh ip interface brief’ command remotely on a network device.
>>> output = connection.send_command('sh ip interface brief')

# printing retrieved data.
>>> print(output)
   Interface                  IP-Address      OK? Method Status
   Protocol
   FastEthernet0              unassigned      YES unset  down
   down
   FastEthernet1              unassigned      YES unset  down
   down
   FastEthernet2              unassigned      YES unset  down
   down
   FastEthernet3              unassigned      YES unset  down
   down
   FastEthernet4              10.10.10.10     YES manual up
   up
```

2.3.2 Napalm

Napalm is very similar to netmiko framework however it consists of some additional functions such as network device configuration management. As we will see in next chapters, this tool was a key to automate the configuration process for all network devices discovered in LAN. Let’s look at a basic example of adding and removing loopback interface on a network device.

```
# importing napalm module.
>>> import napalm

# creating a multiline string representing loopback configuration.
>> add_new_interface = """
... int lo2727
... ip address 172.16.0.10 255.255.255.255
... no shutdown """

# initializing napalm driver object with network device OS.
>>> driver = napalm.get_network_driver(‘ios’)

# passing device hostname and access parameters.
>>> device = driver(
... hostname=’192.168.8.153’,
... username=’rskalban’,
... password=’cisco’,
... optional_args={‘secret’:’cisco’})

# openning connection with a network device.
>> device.open()
```

```
# loading loopback configuration to the device object merge buffer.
>> device.load_merge_candidate(config=add_new_interface)

# applying configuration on device.
>> device.commit_config()

# executing show ip interface brief command to make sure if new
configuration was added.
>> interfaces = device.cli(['show ip int brief'])

# printing the output
>> print(interfaces['show ip int brief'])

Interface                  IP-Address      OK? Method Status
Protocol
GigabitEthernet0/0         192.168.8.153   YES DHCP   up
up
GigabitEthernet0/1         10.0.0.254      YES NVRAM  up
up
GigabitEthernet0/2         unassigned      YES NVRAM  up
up
GigabitEthernet0/3         unassigned      YES NVRAM  administratively down
down
Loopback0                  unassigned      YES unset  up
up
Loopback10                 10.10.10.10     YES NVRAM  administratively down
down
Loopback2727               172.16.0.10     YES TFTP   up
up
```

As we can see a new Loopback2727 interface with IP address 172.16.0.10 has been added successfully. Now lets use rollback method to remove it.

```
>> device.rollback()
>> interfaces = device.cli(['show ip int brief'])
>> print(interfaces['show ip int brief'])

Interface                  IP-Address      OK? Method Status
Protocol
GigabitEthernet0/0         192.168.8.153   YES DHCP   up
up
GigabitEthernet0/1         10.0.0.254      YES NVRAM  up
up
GigabitEthernet0/2         unassigned      YES NVRAM  up
up
GigabitEthernet0/3         unassigned      YES NVRAM  administratively down
down
Loopback0                  unassigned      YES unset  up
up
Loopback10                 10.10.10.10     YES NVRAM  administratively down
down
```

Now we see the previously added *Loopback2727* is no longer available on device. As we could imagine such a tool would be very powerful in managing large scale networks. While we are in the "manage" topic, let's move onto the next supervising tool implemented in my Network Controller.

2.4 Simple Network Management Protocol - SNMP

SNMP[15] is a Southbound API of the Network Controller which in this project, mainly focuses on retrieving data from devices such us: device operating system, device image, number of interfaces, name of interfaces, MAC and IP addresses bound to them and more informations owned by network devices. Also, it is responsible for sending and receiving network notifications such as: "Interface GigabitEthernet0/1 changed state to administratively down" or "OSPF neighbor 10.1.1.1 changed state to DOWN" which are very useful in maintaining and troubleshooting network issues. But before we will move on to examples, let's precisely understand how this protocol works, as the name of it is not truly relevant.

SNMP is an application layer protocol used for monitoring and configuring network devices. Initially it was developed in the 1980s. Over the years as networks become larger and more complicated, several of SNMP versions have been released.

- SNMP version 1 (SNMPv1)
- SNMP version 2 (SNMPv2)
- SNMP version 2c (SNMPv2c)
- SNMP version 3 (SNMPv3)

Both versions 1 and 2 are rarely used today as they are forwarding the traffic in a clear text which means that they are very insecure and SNMP data can be easily exploited by hackers. Version 3 comes with more secured approach, meaning features like authentication, encryption, and message integrity. For the record, in this project SNMPv3 was used. While this protocol was developed both for monitoring and configuring capabilities, only the first one are used these days.

There are three main components used in this protocol:

- Monitoring devices
- SNMP Agent
- SNMP Manager

SNMP Agent is the device which listens on UDP port 161 for requests from SNMP Manager. It can also send notifications to the SNMP Manager which listens for them on UDP port 162. As already mentioned, those can be both not important as well as critical network events. To understand requests which SNMP Managers send to SNMP Agents we need to firstly clarify what MIBs and OIDs are.

2.4.1 Management Information Base - MIB

MIB[16] is a collection of information used for managing devices in the network. Network devices have a database called 'MIB' or 'MIB table' or 'MIB Tree' with the set of 'objects'. These objects store valuable information like CPU utilization, interface status, etc. within the network device.

2.4.2 Object identifier - OID

As we already know MIB is not only a device peripherals and configuration database but it is as well a very general object - it consists of multiple OIDs[17] in a tree like structure, which every one of them represents a single, or a set of device objects. For instance, if you want to retrieve information on a managed device such as it's system up time (sysUpTime), your MIB request will be: OID = (1.3.6.1.2.1.1.3.0). When an SNMP Manager sends a request for the value of (1.3.6.1.2.1.1.3.0), to the SNMP Agent, it basically reads the requested OID from the Agent's MIB, giving back the sysUpTime object. Now when OIDs and MIBs have been clarified, let's move on to the communication model.

2.4.3 Requests, Responses and Traps

We distinguish several messages used to communicate between SNMP Agents and SNMP Managers.

- **GetRequest:** This type of message is used by SNMP Manager to retrieve the value of a variable or list of variables. The SNMP Agent returns a response with requested details.
- **SetRequest:** This type of message is also used by SNMP Manager to change the value of a variable or set of variables. The SNMP Agent returns a response with new values for the variables.
- **Response:** The SNMP Agent generates this message, which contains all information requested by the SNMP Manager.
- **Trap:** The SNMP Agent generates network events, syslogs or notifications and automatically directs them to the SNMP Manager.
- **InformRequest:** While SNMP runs over UDP transport protocol (which is connectionless) it somehow needs to acknowledge received information. InformRequest comes with resolution of this issue.

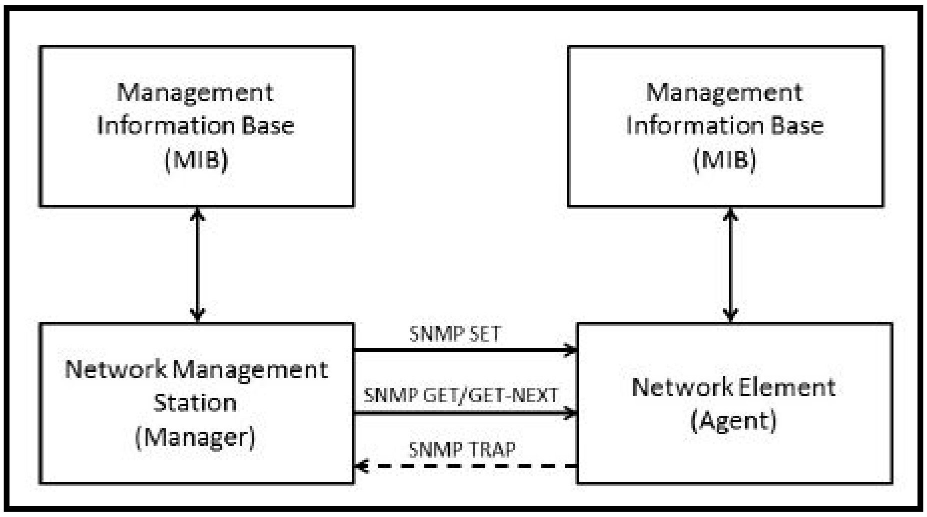

Management Information Base (MIB)
Management Information Base (MIB)
Network Management Station (Manager)
SNMP SET
SNMP GET/GET-NEXT
SNMP TRAP
Network Element (Agent)

Figure 3 *- SNMP Communication Process.*

2.4.4 EasySNMP

EasySNMP is a Python SNMP implementation. In this project, EasySNMP has been used to retrieve network device details which acts as a SNMP Manager. Let's look at the code snippet example below to understand how things were done in this Network Controller.

```
# importing Session class from easysnmp Python module.
>>> from easysnmp import Session

# passing all necessary parameters to create a SNMPv3 connection.
>>> session = Session(
... hostname='192.168.8.153',
... version=3,
... security_level='auth_with_privacy',
... security_username='test1',
... privacy_protocol='AES128',  # using AES128 encryption
algorithm.
... privacy_password='cisco54321',    # encryption key.
... auth_protocol='MD5',              # using MD5 hash function.
... auth_password='cisco12345')

# retrieving hostname from network device ('sysName' and 0 values
corresponds to OID).
>>> print(session.get(('sysName',0)).value)
EdgeRouter.mydomain.com

# retrieving interface name.
>>> print(session.get(('ifDescr', 1)).value)
GigabitEthernet0/0
```

2.4.5 PySNMP

PySNMP is a very similar module to EasySNMP however it brings us additional functionality (which EasySNMP does not implement), meaning handling network Traps from SNMP Agents. Here it as well acts as a SNMP Manager. Let's look at a code snippet example taken out from this project below.

```
# importing all pysnmp items.
from pysnmp.proto import api
from pyasn1.codec.ber import decoder
from pysnmp.carrier.asyncore.dgram import udp
from pysnmp.carrier.asyncore.dispatch import AsyncoreDispatcher

def _initialize_engine(self):
   # initialize asynchronous task.
   self.transport_dispatcher = AsyncoreDispatcher()

# register a receiver function.
self.transport_dispatcher.registerRecvCbFun(self._receive_and_save)

# register transport details such as IP address of SNMP Manager and UDP
port.

self.transport_dispatcher.registerTransport(
      self.udp_domain_name,
      self.udp_socket_transport().openServerMode((
      self.snmp_host, self.snmp_host_port)))
```

```
# start the asynchronous task.
self.transport_dispatcher.jobStarted(1)

# This function listens on the UDP port for incoming SNMP Traps. When it
receives one, it decodes the message into multiple parts and saves it into
the Django DeviceTrapModel database.

def _receive_and_save(self, transportDispatcher, transportDomain,
transportAddress, wholeMsg):

   while wholeMsg:

       msgVer = int(api.decodeMessageVersion(wholeMsg))
       if msgVer in api.protoModules:
           pMod = api.protoModules[msgVer]

       else:
           logging.warning('Unsupported SNMP version %s' % msgVer)
           return

       reqMsg, wholeMsg = decoder.decode(
           wholeMsg, asn1Spec=pMod.Message(),
       )

      logging.warning('########## RECEIVED NEW TRAP ########## ')
      logging.warning(f'Notification message from
      {transportDomain}:{transportAddress}')

      reqPDU = pMod.apiMessage.getPDU(reqMsg)

       if reqPDU.isSameTypeWith(pMod.TrapPDU()):
        trap_date = datetime.now()
        trap_date = trap_date.replace(hour=datetime.now().hour + 1)
        trap_date = trap_date.strftime("%m/%d/%Y, %H:%M:%S")

        trap_domain = transportDomain
        trap_address = transportAddress[0]
        trap_port = transportAddress[1]

        logging.warning(f'Datetime: {trap_date}')
        logging.warning(f'Trap Domain: {trap_domain}')
        logging.warning(f'Agent Address{trap_address}:{trap_port}')

         full_system_name = self._get_system_name(trap_address)

         Device_model=DeviceModel.objects.get(full_system_name=full_system_
         name)

         self.trap_model_parameters = dict(
                           device_model=device_model,
                           trap_domain=trap_domain,
                           trap_address=trap_address,
                           trap_port=trap_port,
                           trap_date=trap_date
                       )

         varBinds = pMod.apiTrapPDU.getVarBinds(reqPDU)

           if not self._database_validator():
                trap_model = DeviceTrapModel(**self.trap_model_parameters)
                trap_model.save()

               for trap_oid, trap_data in varBinds:
                   var_bids_parameters = {
                       'trap_model': trap_model,
                       'trap_oid': trap_oid,
                       'trap_data': trap_data
                   }

                      var_bids_model = VarBindModel(**var_bids_parameters)
                   var_bids_model.save()

       else:
           varBinds = pMod.apiPDU.getVarBinds(reqPDU)

       logging.warning('Var-binds:')
       for oid, val in varBinds:
           logging.warning('%s = %s' % (oid, val))

   return wholeMsg
```

Now when we know how SNMP trap receiver have been implemented, let's generate some Traps from a network device.

Logs from Cisco router:

```
EdgeRouter(config)#int lo10
EdgeRouter(config-if)#no shutdown
*Dec 14 14:02:29.673: %LINK-3-UPDOWN: Interface Loopback10, changed state to up
*Dec 14 14:02:29.674: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback10,
changed state to up
EdgeRouter(config-if)#shut
*Dec 14 14:10:58.232: %LINK-5-CHANGED: Interface Loopback10, changed state to
administratively down
*Dec 14 14:10:58.233: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback10,
changed state to down
```

Logs from Network Controller:

```
[2020-12-14 14:02:31,983: WARNING] ########## RECEIVED NEW TRAP ##########
[2020-12-14 14:02:31,983:WARNING] Notification message from (1, 3, 6, 1, 6, 1,
1):('192.168.8.153', 59619)
[2020-12-14 14:02:31,983: WARNING] Datetime: 12/14/2020, 15:02:31
[2020-12-14 14:02:31,984: WARNING] Trap Domain: (1, 3, 6, 1, 6, 1, 1)
```

```
[2020-12-14 14:02:31,984: WARNING] Agent Address: 192.168.8.153:59619
[2020-12-14 14:02:32,057: WARNING] Var-binds:
[2020-12-14 14:02:32,057: WARNING] 1.3.6.1.2.1.1.3.0 = 1965381
[2020-12-14 14:02:32,057: WARNING/ForkPoolWorker-6] 1.3.6.1.6.3.1.1.4.1.0 =
1.3.6.1.4.1.9.9.41.2.0.1
[2020-12-14 14:02:32,057: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.2.51 = LINK
[2020-12-14 14:02:32,057: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.3.51 = 4
[2020-12-14 14:02:32,058: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.4.51 = UPDOWN
[2020-12-14 14:02:32,058: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.5.51 = Interface
Loopback10, changed state to up
[2020-12-14 14:02:32,058: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.6.51 = 1965381

[2020-12-14 14:11:00,026: WARNING] ########## RECEIVED NEW TRAP ##########
[2020-12-14 14:11:00,026: WARNING] Notification message from (1, 3, 6, 1, 6, 1,
1):('192.168.8.153', 59619)
[2020-12-14 14:11:00,027: WARNING] Datetime: 12/14/2020, 15:11:00
[2020-12-14 14:11:00,027: WARNING] Trap Domain: (1, 3, 6, 1, 6, 1, 1)
[2020-12-14 14:11:00,027:WARNING] Agent Address: 192.168.8.153:59619
[2020-12-14 14:11:00,107: WARNING] Var-binds:
[2020-12-14 14:11:00,107: WARNING] 1.3.6.1.2.1.1.3.0 = 2016037
[2020-12-14 14:11:00,107: WARNING] 1.3.6.1.6.3.1.1.4.1.0 =
1.3.6.1.4.1.9.9.41.2.0.1
[2020-12-14 14:11:00,107: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.2.54 = LINEPROTO
[2020-12-14 14:11:00,108: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.3.54 = 6
[2020-12-14 14:11:00,108: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.4.54 = UPDOWN
[2020-12-14 14:11:00,108: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.5.54 = Line protocol
on Interface Loopback10, changed state to down
[2020-12-14 14:11:00,108: WARNING] 1.3.6.1.4.1.9.9.41.1.2.3.1.6.54 = 2016037
```

As you can see in the above example all SNMP Traps sent by SNMP Agent have been decoded and saved into the Network Controller database.

2.5 Link Layer Discovery Protocol - LLDP

LLDP[18] is a multi vendor compatible link layer protocol used by network devices to advertise their details such as connected neighbors, local and remote ports used to link them together. It gathers as well details such as Vlan ID, management IP address or system description. This protocol has been used as a core element to discover and build network topologies dynamically by a Network Controller. Let’s look at the example below to understand it’s simplicity and strength

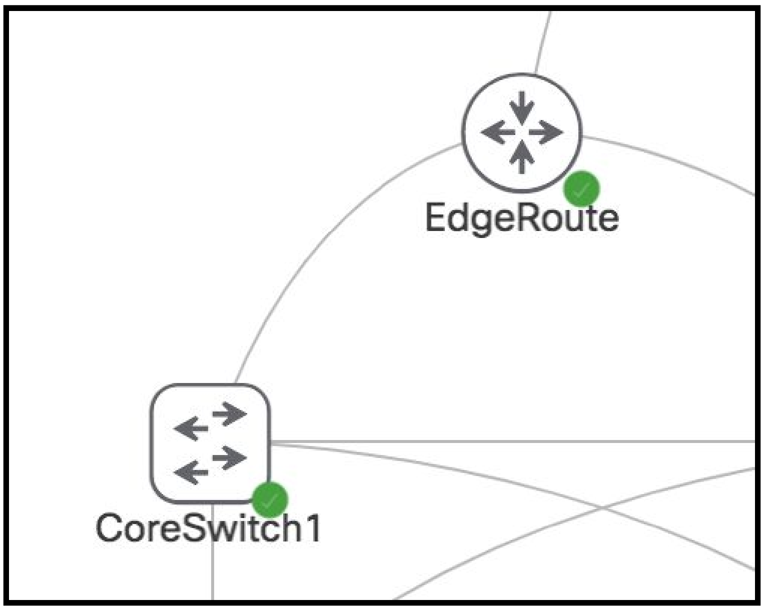


Figure 4 *- Part of LAN topology.*

On both devices *CoreSwitch1* and *EdgeRouter* LLDP has been activated globally. After issuing "show lldp neighbors" we can see that both devices discovered their connections.

```
CoreSwitch1#sh lldp neighbors
Capability codes:
    (R) Router, (B) Bridge, (T) Telephone, (C) DOCSIS Cable Device
    (W) WLAN Access Point, (P) Repeater, (S) Station, (O) Other

Device ID           Local Intf     Hold-time  Capability      Port ID
EdgeRouter.mydomain.Gi0/3          120        R               Gi0/2

Total entries displayed: 1

EdgeRouter#sh lldp neighbors
Capability codes:
    (R) Router, (B) Bridge, (T) Telephone, (C) DOCSIS Cable Device
    (W) WLAN Access Point, (P) Repeater, (S) Station, (O) Other

Device ID           Local Intf     Hold-time  Capability      Port ID
CoreSwitch1.mydomainGi0/2          120        R               Gi0/3

Total entries displayed: 1
```

We can as well use "sh lldp neighbors detail" to check some more information.

```
EdgeRouter#sh lldp neighbors detail
------------------------------------------------
Local Intf: Gi0/2
Chassis id: 5254.0005.c737
Port id: Gi0/3
Port Description: GigabitEthernet0/3
System Name: CoreSwitch1.mydomain.com
System Description:
Cisco IOS Software, vios_l2 Software (vios_l2-ADVENTERPRISEK9-M), Version
15.2(CML_NIGHTLY_20190423)FLO_DSGS7, EARLY DEPLOYMENT DEVELOPMENT BUILD,
synced to  V152_6_0_81_E
Technical Support: http://www.cisco.com/techsupport
Copyright (c) 1986-2019 by Cisc

Time remaining: 112 seconds
System Capabilities: B,R
Enabled Capabilities: R
Management Addresses:
    IP: 10.0.0.1
Auto Negotiation - not supported
Physical media capabilities - not advertised
Media Attachment Unit type - not advertised
Vlan ID: - not advertised


Total entries displayed: 1
```

2.5 JavaScript

Javascript[19] is a high-level, often just-in-time compiled and multi-paradigm programming language. Alongside with HTML and CSS, JavaScript is one of the core technologies of the World Wide Web. It has a curly-bracket syntax and dynamic typing. It is widely used in interactive web pages as it changes content of them dynamically, without the need of full web page reload. As I have mentioned earlier, handling asynchronous traffic from server perspective is done by using Django Channels but from client or web browser perspective, it is made by JavaScript's WebSocket.

Also, it is applied to generate dynamic network topology for all discovered network devices by a Network Controller.

2.5.1 jQuery

jQuery[20] is the most popular JavaScript library. It was created to simplify HTML tree traversal and manipulation, as well as handling events, CSS animation, and Ajax. Statistics from May 2019 says that this JavaScript library is used by 7.3 millions of the most popular websites in the world. In this project jQuery has been used to grab and create HTML objects as well as dynamically change, add or delete their properties. Also, it is used to generate asynchronous requests from the client's backend.

2.5.2 Extensible Markup Language - XML

XML is an Extensible Markup Language meaning, it defines a set of rules for encoding documents which is both readable for humans and machines. It is commonly used for exchanging data between different devices or applications.

Example of XML data format representing a book object:

```
<?xml version="1.0"?>
  <book>
      <id>0512</id>
      <author>Gambardella, Matthew</author>
      <title>XML Developer's Guide</title>
      <genre>Computer</genre>
      <price>44.95</price>
      <publish_date>2000-10-01</publish_date>
      <description>An in-depth look at creating applications
      with XML.</description>
   </book>
```

2.5.3 JavaScript Object Notation - JSON

Unlike XML, it is an open standard file format, and data interchange format, that uses human-readable text to store and exchange data objects consisting of key–value pairs and array data types (or any other serializable value). It is a very common data format, with a diverse range of applications, such as serving as a replacement for XML in AJAX systems. It consists of both curly and square brackets. Example of JSON data format representing a book object given below.

```
{
 "book":{
      "id":"0512",
      "Authors":[
            "Gambardella",
            "Matthew",]
      "Title":"XML Developer's Guide",
      "Genre":"Computer",
      "Price":"44.95",
      "Publish Date":"2000-10-01",
      "Description":"An in-depth look at creating applications
      with XML"
}
```

2.5.4 Asynchronous JavaScript and XML - AJAX

Ajax stands for Asynchronous JavaScript and XML. Ajax uses both XML and JSON to asynchronously communicate with the backend of a server. In this project, AJAX has been used to dynamically make requests to the Django server, without reloading the page. Ajax is not a new technology or programming language rather than that, it is a new approach to old technologies. We will see exact use cases in the next chapters.

2.6 Cascading Style Sheets - CSS

This project has been implemented with both backend and frontend logic. Second one is what the user sees while using the Network Controller. As this application is web-based, the frontend has been built by using both HTML and CSS[21]. Probably everyone reading this document is somehow familiar with HTML, I will only remind you that it stands for Hyper Text Markup Language and is used by web browsers as a core item for rendering web pages. The CSS styling is used to basically make the frontend more clear and user friendly. It enables the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility, provide more flexibility and control in the specification of presentation characteristics. For the record, CSS styling has been borrowed and modified from open source project - Django Dashboard Black[22]

2.6.1 Bootstrap

Bootstrap is an open source CSS framework, mobile-first front-end web development, which implements both CSS styling as well as JavaScript (that's optional). It gives frontend developers some tools which help with better designing typography, forms, buttons, navigation, and other interface components.

2.7 Network Virtualization

To provide an example of use for Network Controller, there is of course need for a network with some devices like routers or switches, which are both compatible with SNMP, LLDP and SSH. Basically they should run Cisco IOS (operating system designed for network devices by Cisco Systems). As I was not able to access multiple enterprise network devices, I came up with an idea to virtualize it. With the resolution of this issue CML^2 comes to play.

2.7.1 Cisco Modeling Labs 2 - CML^2

CML^2 **[23]** is a powerful platform used for network virtualization. It was designed to be a flexible, all-in-one virtual networking lab. With Cisco CML and virtual instances for network operation system (Cisco IOS), it is possible to build entire network simulations with availability to create an Internet gateway. Cisco CML provides multiple interfaces by which users can interact with: CLI interface, a powerful web based interface and REST API for automation and integration. In this project I have built a virtual network by using a web based interface. We will see example in further chapter.

2.8 Tools Versions

- Python - 3.8.1
- Django - 3.1.1
- Django Channels - 3.0.2
- Django Celery - 4.4.7
- RabbitMQ - 3.8.9
- Netmiko - 3.3.0
- Napalm - 3.2.0
- SNMP - 3.0
- EasySNMP - 4.4.12
- PySNMP - 0.2.5
- JavaScript - Depends on the browser.
- jQuery - 3.3.1
- CSS - 2.1
- Bootstrap - 4.3.1
- CML^2 - 2.0.0-b13

3. Project Structure

3.1 Main App

3.1.1 Description

This part of the project is a core element of it as it possesses all configuration files and route specifications.

3.1.2 Implementation

```
main_app
│
├── asgi.py
├── backend
│   └── helpers.py
├── celery.py
├── mixins
│   └── JSONResponseMixin.py
├── routing.py
├── settings.py
└── urls.py
```

- ***asgi.py*** - it exposes ASGI (Asynchronous Server Gateway Interface) functionality for Django server, which means we are able to handle asynchronous traffic from the server's perspective.
- ***backend/****helpers.py* - it is used to verify if access and SNMP configurations were properly applied by the user.
  Functions:
  - *check_initial_configurations_applied()*
  - *check_if_properly_configured()*

- ***celery.py*** - is a configuration file for embedding Celery (Worker) into Django server.
- ***mixins/****JSONResponseMixin.py* - is responsible for generating HTTP responses in JSON format.
  Class:
  - *JSONResponseMixin(object)*
    Methods:
    - render_to_response(self,context,**response_kwargs
)
    - convert_context_to_json(self, context)

- ***routing.py*** - it exposes a web socket endpoint to which the client is pointing in order to establish a WebSocket connection.
- ***settings.py*** - this file is a core configuration file for Django server.
- ***urls.py*** - this file exposes all main routes handled by Network Controller.

3.2 Logging App

3.2.1 Description

This app is responsible for handling authentication functions.

3.2.2 Implementation

```
logging_app
|
├── templates
│   └── login.html
└── views.py
```

- ***templates/****login.html* - this HTML file consists of proper tags which represents the logging page.
- ***views.py*** - in this file there are two class based views responsible for handling both synchronous GET and POST HTTP requests referred to login page.

  Classes:

  - *LoginView(ListView)*

    Methods:

    - get(self, request, *args, **kwargs)
    - post(self, request, *args, **kwargs)

  - LogoutView(ListView)

    Methods:

    - get(self, request, *args, **kwargs)

3.3 Registration App

3.3.1 Description

This app delivers to users a registration form, by which users can create new accounts.

3.3.2 Implementation

```
registration_app
|
├── templates
│   └── register.html
└── views.py
```

- ***templates/****register.html* - this HTML template serves registration form.

- ***views.py*** – this file handles all GET and POST HTTP requests specified for the registration section.

  Class:
  ```
  - RegistrationView(ListView)
  ```
  Methods:
  ```
      - get(self, request, *args, **kwargs)
      - post(self, request, *args, **kwargs)
  ```

3.4 Dashboard App

3.4.1 Description

Dashboard app merge up together config, manage, visualize and logout sections into one nicely designed navigation panel.

3.4.2 Implementation

```
dashboard_app
│
├── templates
│   └── dashboard.html
├── urls.py
└── views.py
```

- **templates/**`dashboard.html` – this HTML file brings up all needed tags to create navigation panel and containers.
- **urls.py** – is responsible for exposing all available dashboard, config, manage and visualize urls.
- **views.py** – this file serves all synchronous HTTP GET requests.

  Class:
  ```
  - DashboardView(ListView)
  ```
  Method:
  ```
      - get(self, request, *args, **kwargs)
  ```

3.5 Configuration App

3.5.1 Description

This part of the project is responsible for creating and applying access and SNMP configuration to all discovered devices in the network. When a user creates access configuration, there is a need to specify network IP address for the given LAN network. After specifying that details, Network Controller is trying to reach all devices in the given network, and returns back only those which responded. In access configuration, the user as well specifies authentication details which are configured on network devices as well as devices operating system. Then SNMP custom configuration needs to be added. After both configurations are saved in the Django database, Network Controller is logging to all available devices discovered initially by using access configuration and then applies SNMP and LLDP configuration to end network devices for further usage with help of NAPALM.

3.5.2 Implementation

```
config_app
|
├── backend
│   ├── ConfigManager.py
│   ├── helpers.py
│   ├── parsers.py
│   └── static.py
├── forms.py
├── models.py
└── templates
    └── config_network.html
```

- **backend/**`ConfigManager.py` - this file is responsible for connecting to multiple devices discovered in the network and configuring them with a given SNMP configuration. Each session is being established on a separate thread.

  Class:

  - `ConfigManager`

  Methods:

  ```
  -  __init__(self, initial_config_data,
     login_params, available_hosts)

  -  __connect_and_change_single(self, host,
  config_commands, type_of_change='configure')

  - connect_and_configure_multiple(self,
  config_commands=None,type_of_change='configure')

  - get_command_output(self)
  ```

- **backend/**`helpers.py` - this file contains functions which help config app with simple operations.

  Functions:

  ```
  - convert_mask(config)
  - ping_ip(current_ip_address)
  - ping_all(config)
  - get_thread_output(que_object, threads_list)
  - connect_and_get_output(device)
  ```

- **backend/**`parsers.py` - this file includes parsers which are responsible for converting user data into NAPALM and Netmiko required format.

  Functions:

  ```
  - parse_initial_config(object_id)
  - parse_snmp_config(object_id)
  - remove_snmp_config(object_id)
  ```

- **backend/**`static.py` – this file consists of static data used in config app.
- **`forms.py`** – this file implements access config and SNMP config forms, by which users can add entry to ConfigParametrs and SNMPConfigParameters tables.

  Classes:

  - `ConfigParametersForm(forms.ModelForm)`
  - `SNMPConfigParametersForm(forms.ModelForm)`

- **`models.py`** – this file creates database tables needed for storing config app data.

  Classes:

  - `ConfigParameters(models.Model)`
  - `AvailableDevices(models.Model)`
  - `SNMPConfigParameters(models.Model)`

- **templates/**`config_network.html` – this HTML file is responsible for presenting config app to the user.

3.6 Manage App

3.6.1 Description

Manage app is a core element in this project as it exploits previously configured SNMP, SSH and LLDP protocols, by which it is building a network devices database. It is also responsible for running Trap Engine and establishing web based SSH terminal session. By using EasySNMP and multi threading technique, manage app spawns multiple threads correlated to number of discovered devices in the network and then, it establishes SNMPv3 sessions with all agents and retrieves device details specified by developer. Users can as well start a TrapEngine which sends a SNMP trap receiver function (which opens UDP port 162 and listens for notifications) as a task to RabbitMQ, which immediately sends to Django Celery Worker and then executes as no periodic features have been implemented. While this functionality is running asynchronously, users can as well start a web based SSH terminal session. Manage app exposes a WebSocket endpoint to which users are directed after starting the terminal session and opens it on the server side. Meantime Network Controller is establishing SSH session with chosen network device by using Netmiko functionality. Finally JavaScript's WebSocket is being opened and connection between client and server is established after a 3 way handshake.

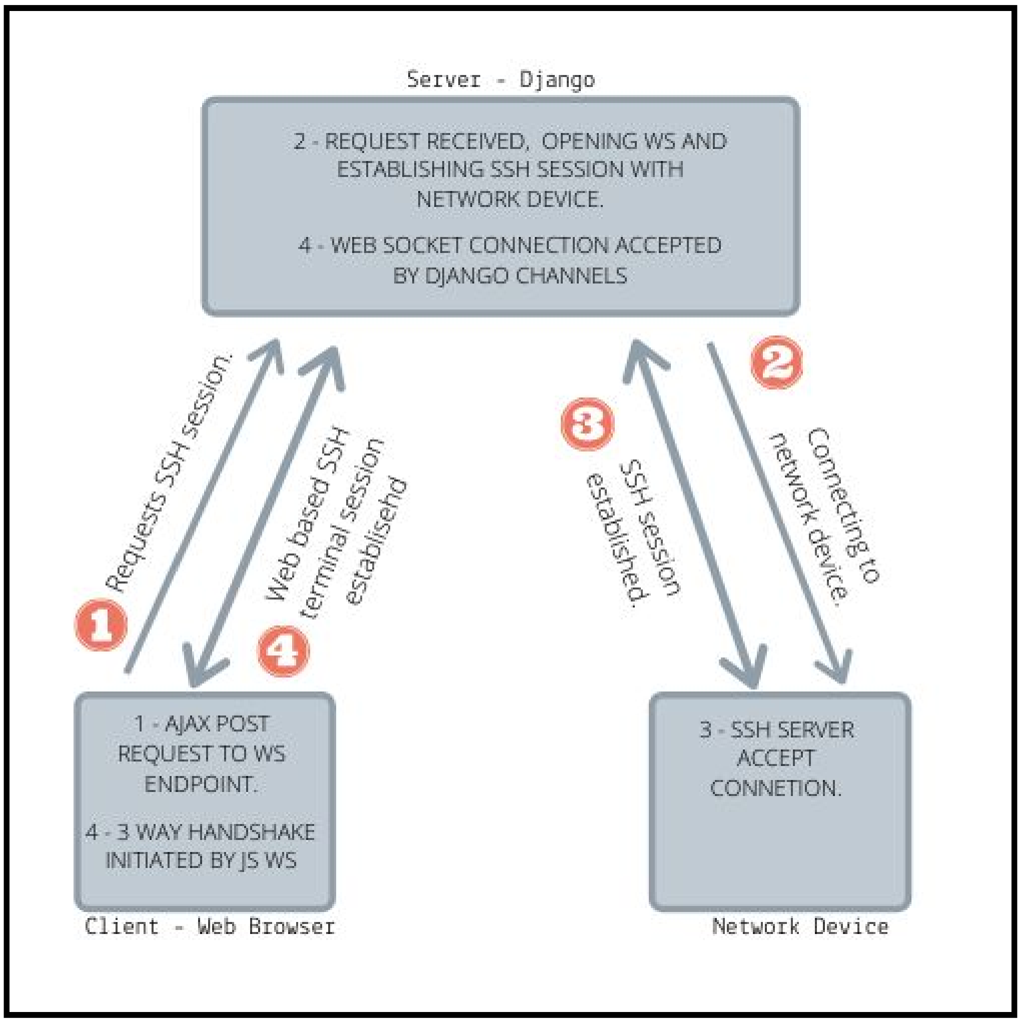

Server - Django
2 - REQUEST RECEIVED, OPENING WS AND ESTABLISHING SSH SESSION WITH NETWORK DEVICE.
4 - WEB SOCKET CONNECTION ACCEPTED BY DJANGO CHANNELS
1
Requests SSH session.
4
Web based SSH terminal session establisehd
3
SSH session established.
2
Connecting to network device.
1 - AJAX POST REQUEST TO WS ENDPOINT.
4 - 3 WAY HANDSHAKE INITIATED BY JS WS
Client - Web Browser
3 - SSH SERVER ACCEPT CONNETION.
Network Device

Figure 5 *- Web based SSH transmission.*

3.6.2 Implementation

```
manage_app
|
├── backend
|    ├── ConnectionHandler.py
|    ├── DeviceManager.py
|    ├── TrapEngine.py
|    ├── mixins
|    |    ├── AjaxSSHSessionMixin.py
|    |    └── AjaxTrapEngineMixin.py
|    ├── parse_model.py
|    ├── static.py
|    └── tasks.py
├── consumers.py
├── models.py
├── templates
|    └── manage_network.html
└── views.py
```

- **backend/**`ConnectionHandler.py` - this file is responsible for establishing an asynchronous SSH session between Network Controller (Django server) and specific network device (SSH server). Further used to exchange data between user's WebSocket and server's Django Channel session.

  Class:
  - `SSHConnectionHandler`

    Methods:
    - `initialize_connection(cls,device_id,conf_access_id)`
    - `write_to_connection(self, command_string)`
    - `read_from_connection(self)`

- **backend/**`DeviceManager.py` - this file contains key elements by which devices database is being created. It uses EasySNMP module to generate SNMP GetRequests towards SNMP agents and retrieve all needed data specified by developer.

  Classes:
  - `DeviceManager`

    Methods:
    - `__init__(self, user, available_hosts, snmp_config_id)`
    - `__get_single_device_details(self, hostname)`
    - `get_multiple_device_details(self)`
  - `DeviceSystem_`

    Methods:
    - `__init__(self, hostname, session)`
  - `DeviceInterface_`

    Methods:
    - `__init__(self, number, session)`
  - `Device`

    Methods:
    - `__init__(self, hostname, session)`
    - `__get_lldp_entries(self)`

- **backend/**`TrapEngine.py` - in this file SNMP trap receiver has been implemented. As mentioned earlier, it is used by RabbitMQ broker and Django Celery worker to run this type of tasks asynchronously.

  Class:
  - `TrapEngine`

    Methods:
    - `__init__(self, snmp_host, snmp_host_port, session_parameters)`
    - `_receive_and_save(self, transportDispatcher, transportDomain, transportAddress, wholeMsg)`
    - `_database_validator(self)`
    - `_get_system_name(self, trap_address)`
    - `_initialize_engine(self)`
    - `run_engine(self)`

- `close_engine(self)`

- **backend/mixins/**`AjaxSSHSessionMixin.py` - this file handles asynchronous HTTP POST requests by which users start a web based SSH terminal session.

  Class:
  - `AjaxSSHSessionView(JSONResponseMixin, View)`

    Method:
    - `post(self, request, *args, **kwargs)`

- **backend/mixins/**`AjaxTrapEngineMixin.py` - this file handles asynchronous HTTP POST requests, by which users start a Trap Engine and are able to receive notifications from SNMP agents.

  Class:
  - `AjaxTrapEngineView(JSONResponseMixin, View)`

    Methods:
    - `post(self, request, *args, **kwargs)`
    - `_post_start_trap_engine(self)`
    - `_post_stop_trap_engine(self)`

- **backend/**`parse_model.py` - this file includes parsers which are responsible for performing some data formatting.

  Functions:
  - `parse_to_session_parameters(snmp_config_id)`
  - `parse_mac_address(mac_address)`
  - `parse_up_time(system_ticks)`
  - `save_to_database_lldp_data(lldp_neighbor_details)`
  - `format_lldp_data(devices)`
  - `parse_and_save_to_database(devices, user)`
  - `parse_trap_model(device_trap_models, trap_data)`

- **backend/**`static.py` - this file consists of static data used in manage app.
- **backend/**`tasks.py` - this file specifies exact Trap Engine task which is further used by Django Celery.

  Function:
  - `run_trap_engine(snmp_host, snmp_config)`

- **consumers.py** - this file implements Django Channels functionality, in other words it handles all asynchronous traffic initiated by JavaScript’s WebSocket, and acts as a connector between the user's frontend SSH session and the network device one.

  Class:
  - `SSHConsumer(AsyncConsumer)`

    Methods:
    - `websocket_connect(self, event)`
    - `websocket_receive(self, event)`
    - `websocket_disconnect(self, event)`

- **`models.py`** - in this file all database tables referred to manage app are defined.

  Classes:

  ```
  - DeviceModel(models.Model)
  - DeviceInterface(models.Model)
  - DeviceTrapModel(models.Model)
  - VarBindModel(models.Model)
  ```

- **`templates/`**`manage_network.html` - this file contains a HTML template with all proper tags mandatory to render manage app frontend.

- **`views.py`** - this file handles both synchronous and asynchronous HTTP GET and POST methods directed to manage app.

  Classes:

  - `ManageNetworkView(ListView)`

    Methods:

    ```
    - _get_device_details(self, request)
    - _get_traps_pages(self)
    - _refresh_device_list(self, request)
    - _get_config_details(self, request)
    - get(self, request, *args, **kwargs)
    ```

  - `AjaxTrapView(JSONResponseMixin, View)`

    Methods:

    ```
    - get(self, request, *args, **kwargs)
    ```

3.7 Visualize App

3.7.1 Description

Most computer networks are likely to be huge and complicated. In order to perform successful troubleshooting in those, network engineers must have access to the latest network diagrams. However, as we can probably imagine there are always some kind of changes or improvements. To keep an eye on the most recent topologies, visualize app comes into play. When users successfully go through config and manage app, they will see dynamically generated network diagram with all neighbor connections discovered by LLDP in the visualize section. If a user clicks on a specific device icon, the new card with neighbor connection details will pop up. LLDP data is being gathered in manage app while building the devices database. When users go to visualize section, on the backend NetworkMapper kicks in and generates graph data in JSON format, which next is used by JavaScript to render out dynamic topology.

3.7.2 Implementation

```
visualize_app
|
├── backend
│   ├── NetworkMapper.py
│   ├── mixins
│   │   └── AjaxDeviceNeighbors.py
│   └── static.py
├── templates
│   └── visualize.html
└── views.py
```

- **`backend/`**`NetworkMapper.py` – like mentioned earlier, this file takes neighbor connection device data from Django database and converts it into JSON data file.

  Class:
  - `NetworkMapper`

    Methods:
    - `__init__(self)`
    - `__hostname_parser(hostname)`
    - `__node_validator(self, device_name)`
    - `__object_validator(self, id, container, object_id)`
    - `clear_graph_data(self)`
    - `generate_graph_data(self)`
    - `generate_nodes_graph_data(self)`
    - `generate_links_graph_data(self)`

- **`backend/mixins/`**`AjaxDeviceNeighbors.py` – this file is responsible for handling asynchronous HTTP GET requests which are created by the user while clicking on the device icon for getting neighbor details. Server returns those details to JavaScript which injects new card.

  Class:
  - `AjaxDeviceNeighborsView(JSONResponseMixin, View)`

    Method:
    - `get(self, request, *args, **kwargs)`

- **`backend/`**`static.py` – contains static data for visualize app.
- **`templates/`**`visualize.html` – contains HTML file with all tags needed to render visualize app.
- **`views.py`** – this file handles all synchronous HTTP GET requests issued to visualize app.

  Class:
  - `VisualizeNetworkView(ListView)`

    Method:
    - `get(self, request, *args, **kwargs)`

4. Example Of Use

In this last but not least chapter, I will present an example of use for designed Network Controller. LAN Network will be simulated in the already mentioned and described CML^2 virtual environment.

4.1 Virtualized network

Virtualized LAN network shown below consists of four Cisco switches and one Cisco router, which is a gateway to the external world - home LAN and going further - WAN. On all devices OSPF routing protocol is running in area 0. Edge Router redistributes to its neighbor a default route, by which all devices are able to reach Network Controller (located in home LAN running on my macOS) as well as Internet.

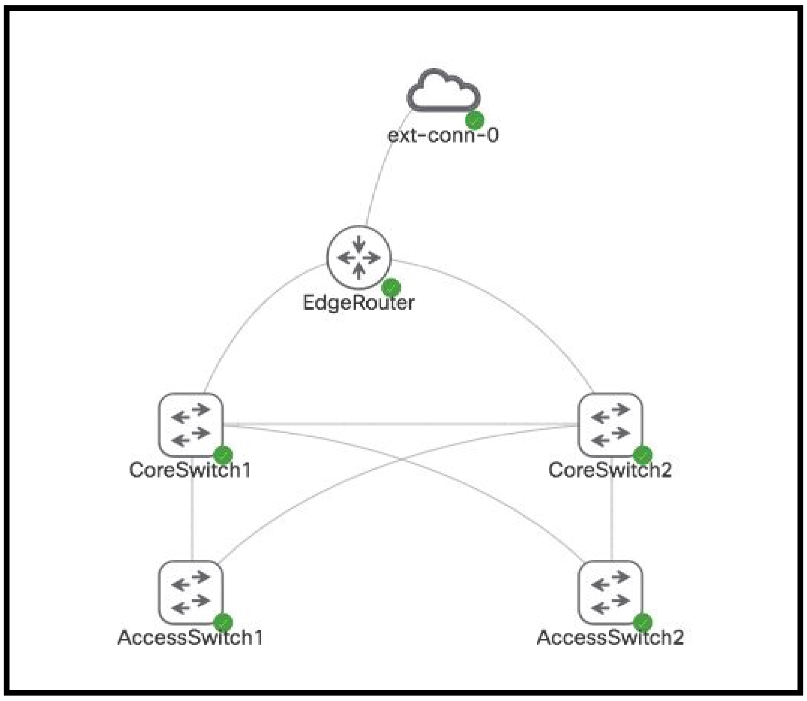


Figure 6 *- Virtualized LAN Topology.*

Mandatory configuration needed on every device for Network Controller to function properly.

```
### ACCESS CONFIGURATION ###
configure terminal
      enable 15
      enable secret enable_password
      ip domain-name mydomain.com
      crypto key generate rsa 1024
      ip ssh version 2

      line vty 0 4
            transport input telnet ssh
            login local

      username rskalban privilege 15 password cisco
```

Above configuration enables SSH server on device and sets up login credentials

```
### NAPALM CONFIGURATION ###
ip scp server enable
aaa new-model
aaa authentication login default local
aaa authorization exec default local none
```

Above configuration enables AAA and SCP services in order to make the configuration manager (NAPALM engine) work properly.

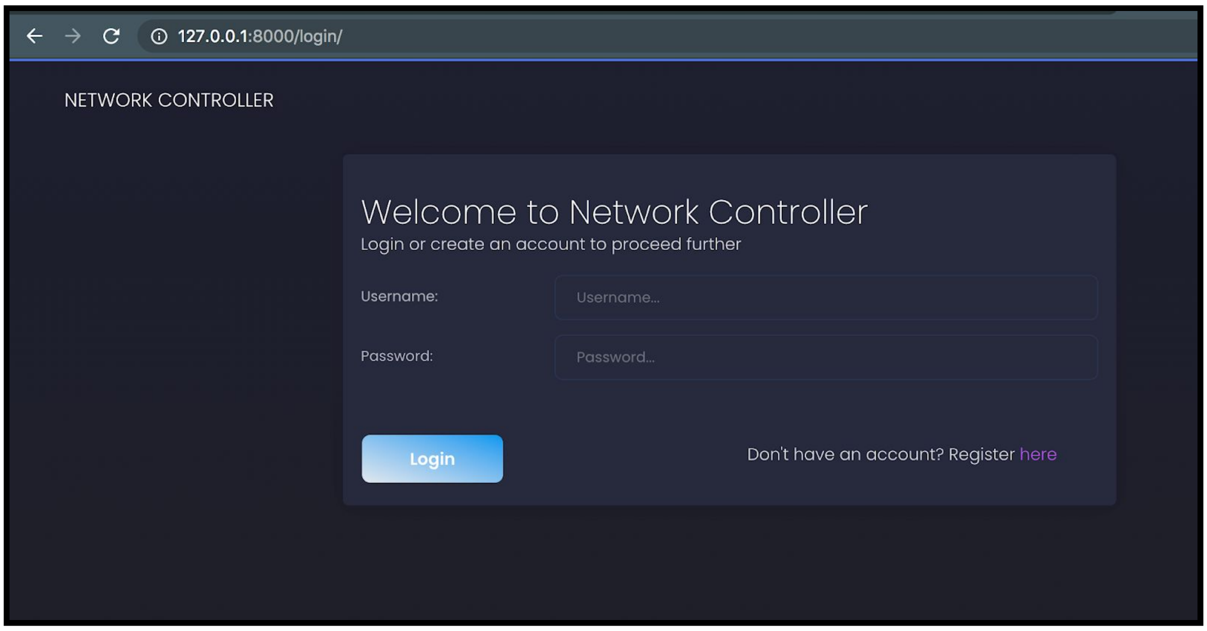
The next step for making sure that Network Controller will run properly is that we have to configure all network devices in such a way that they will be able to reach it. As mentioned earlier in this example EdgeRouter is redistributing default route towards external network in which, Network Controller is under 192.168.8.106 IPv4 address. Lets see if all devices can access it.

```
AccessSwitch1#ping 192.168.8.106
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.8.106, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/4/7 ms

AccessSwitch2#ping 192.168.8.106
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.8.106, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/5/7 ms

CoreSwitch1#ping 192.168.8.106
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.8.106, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 5/6/7 ms

CoreSwitch2#ping 192.168.8.106
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.8.106, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/3/7 ms

EdgeRouter#ping 192.168.8.106
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.8.106, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/3 ms
```

Now that we confirmed all devices can access Network Controller, let's finally get to it.

4.2 Network Controller

This infrastructure can be used by multiple users, managing different networks at the same time so let's begin with registration of a new account.

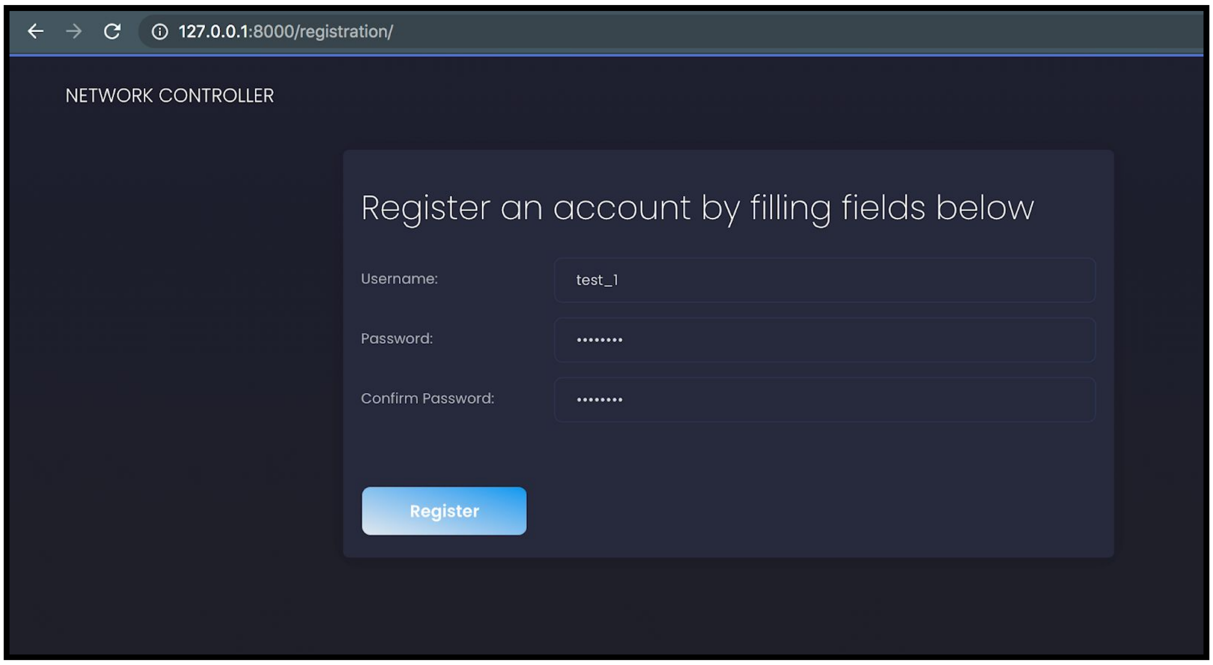


Welcome to Network Controller
Login or create an account to proceed further

Don't have an account? Register here

***Figure** 7 - Login View.*

The first page we will get when trying to access Network Controller is the login page (created in login app). As we have not created a new account yet, let's do that by clicking on *here* link.



Register an account by filling fields below

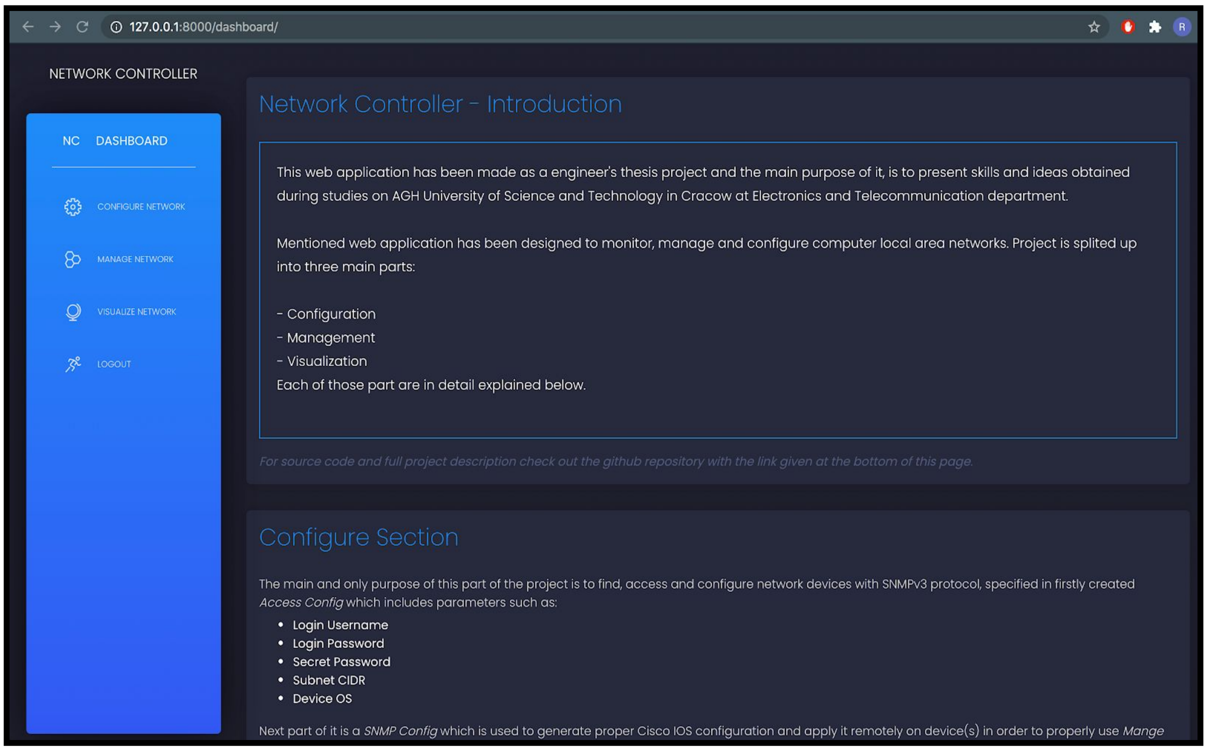
Username: test_1

Password: ••••••••

Confirm Password: ••••••••

***Figure 8** - Registration View.*

After successful registration (handled by registration app) we will be directed to a dashboard, which acts as a bridge for all parts of the project.


127.0.0.1:8000/dashboard/

NETWORK CONTROLLER

NC DASHBOARD

CONFIGURE NETWORK

MANAGE NETWORK

VISUALIZE NETWORK

LOGOUT

Network Controller - Introduction

This web application has been made as a engineer's thesis project and the main purpose of it, is to present skills and ideas obtained during studies on AGH University of Science and Technology in Cracow at Electronics and Telecommunication department.

Mentioned web application has been designed to monitor, manage and configure computer local area networks. Project is splited up into three main parts:

- Configuration
- Management
- Visualization

Each of those part are in detail explained below.

For source code and full project description check out the github repository with the link given at the bottom of this page.

Configure Section

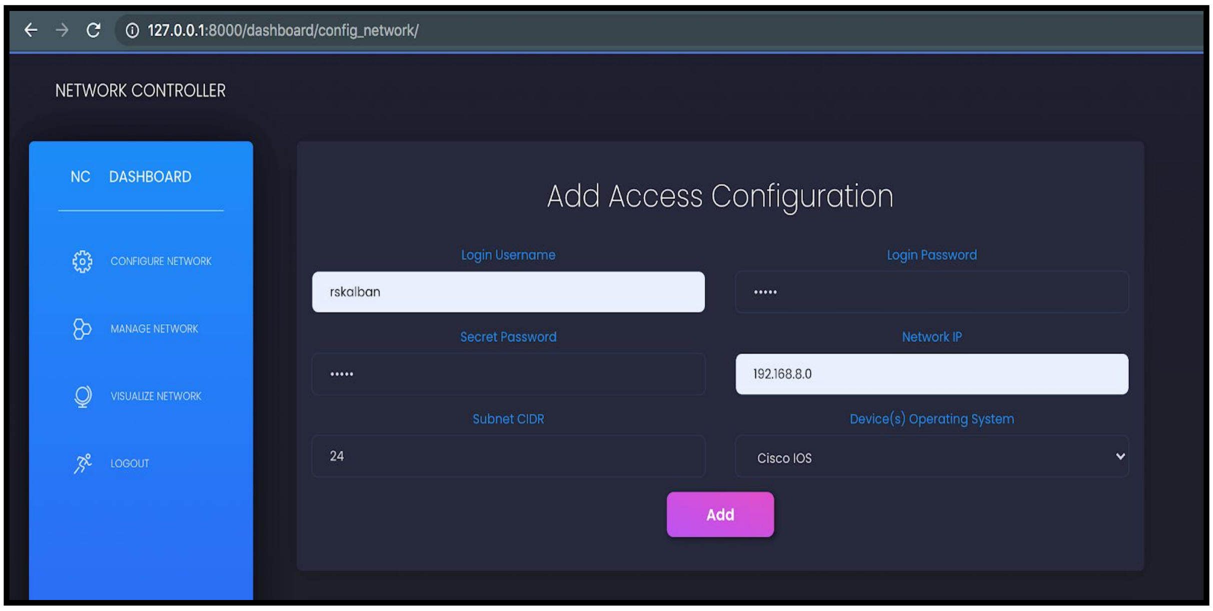
The main and only purpose of this part of the project is to find, access and configure network devices with SNMPv3 protocol, specified in firstly created *Access Config* which includes parameters such as:

- Login Username
- Login Password
- Secret Password
- Subnet CIDR
- Device OS

Next part of it is a *SNMP Config* which is used to generate proper Cisco IOS configuration and apply it remotely on device(s) in order to properly use *Mange*

Figure 9 *- Dashboard View.*

From the dashboard page we will be directed to the configuration section (managed by configuration app). In this part we will add both access and SNMP configuration. Access configuration must be the same as specified on network devices (login credentials) and network IP must be the same subnet as for LAN.


127.0.0.1:8000/dashboard/config_network/

NETWORK CONTROLLER

NC DASHBOARD

CONFIGURE NETWORK

MANAGE NETWORK

VISUALIZE NETWORK

LOGOUT

Add Access Configuration

Login Username: rskalban

Login Password: •••••

Secret Password: •••••

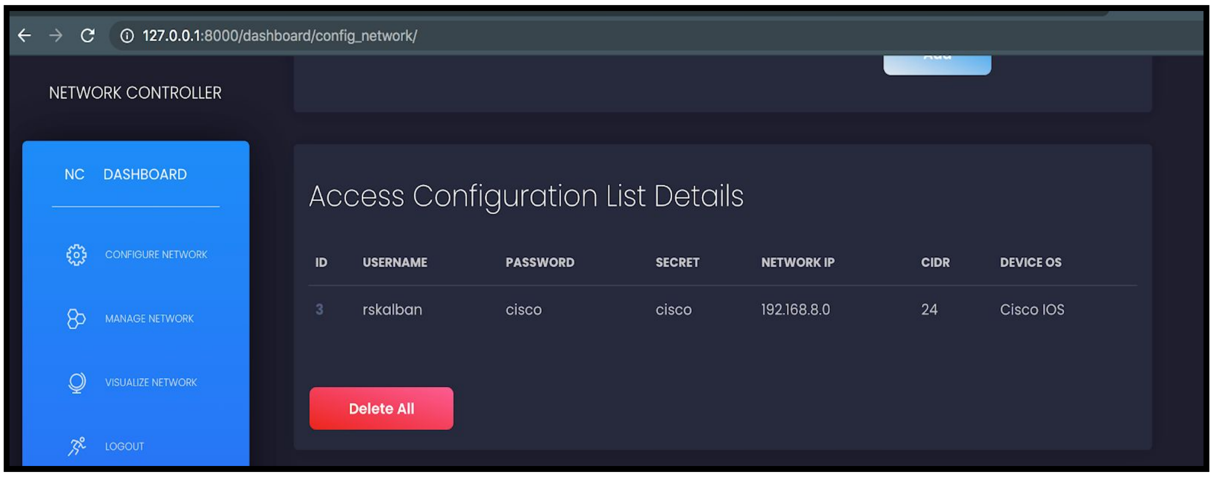
Network IP: 192.168.8.0

Subnet CIDR: 24

Device(s) Operating System: Cisco IOS

Add

Figure 10 *- Configuration View - Adding Access Configuration.*


127.0.0.1:8000/dashboard/config_network/

NETWORK CONTROLLER

NC DASHBOARD

CONFIGURE NETWORK
MANAGE NETWORK
VISUALIZE NETWORK
LOGOUT

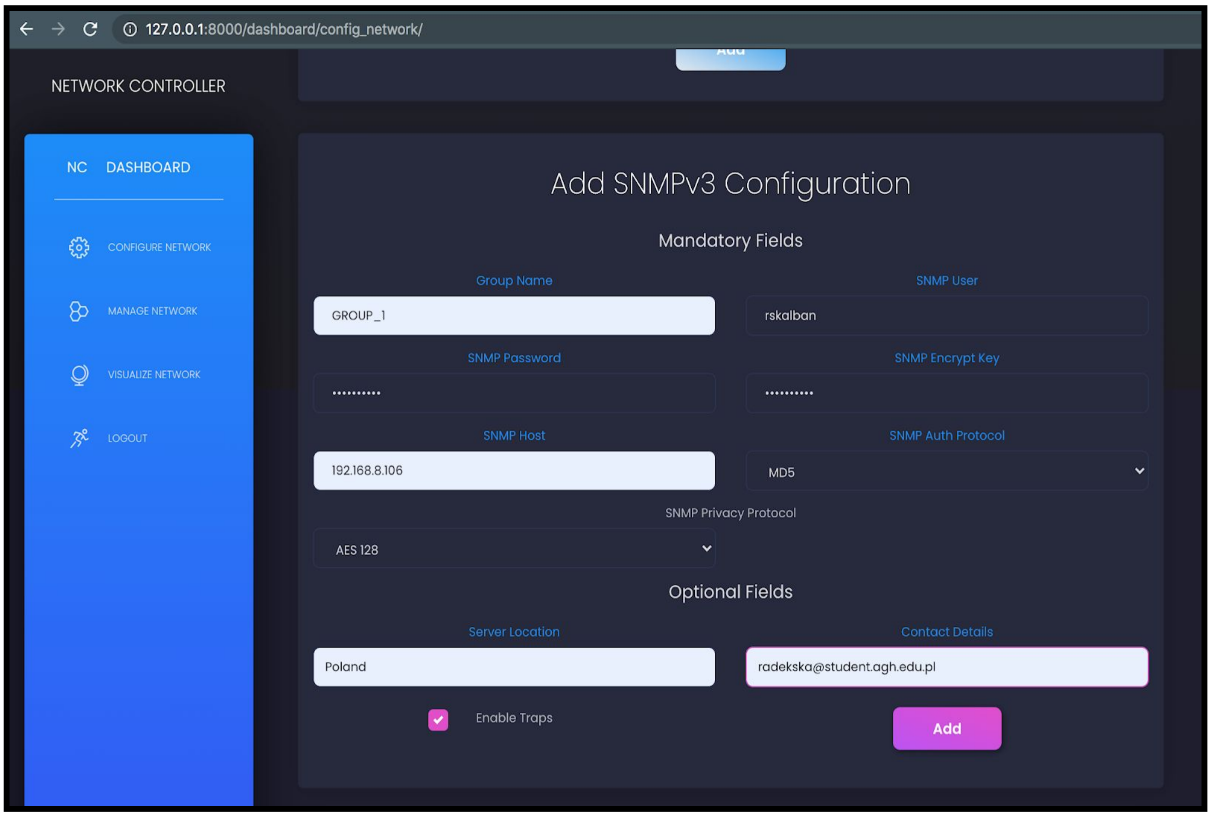
Access Configuration List Details

ID	USERNAME	PASSWORD	SECRET	NETWORK IP	CIDR	DEVICE OS
3	rskalban	cisco	cisco	192.168.8.0	24	Cisco IOS

Delete All

Figure 11 *- Configuration View - Available Access Configurations.*

In the next step we will add our custom SNMPv3 configuration. All parameters can be freely chosen except SNMP Host, as it has to be the IP address of the Network Controller. This way SNMP Agents will know to which address send network notifications.


127.0.0.1:8000/dashboard/config_network/

NETWORK CONTROLLER

NC DASHBOARD

CONFIGURE NETWORK
MANAGE NETWORK
VISUALIZE NETWORK
LOGOUT

Add SNMPv3 Configuration

Mandatory Fields

Group Name: GROUP_1
SNMP User: rskalban
SNMP Password: ••••••••••
SNMP Encrypt Key: ••••••••••
SNMP Host: 192.168.8.106
SNMP Auth Protocol: MD5
SNMP Privacy Protocol: AES 128

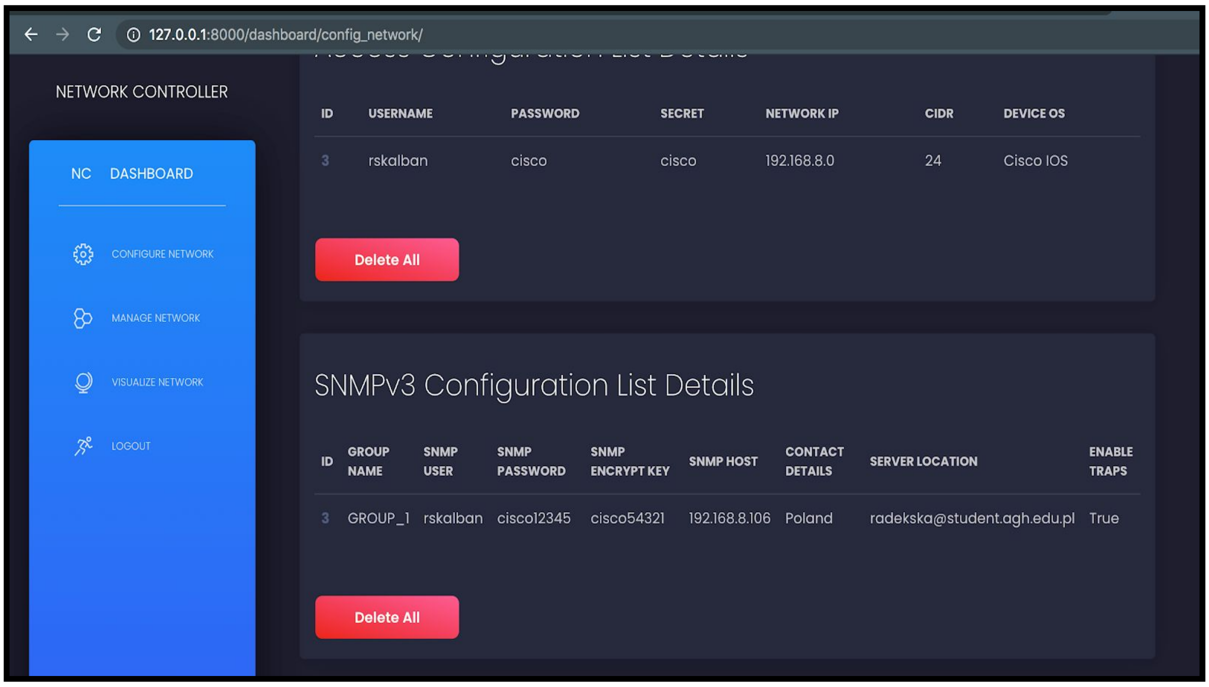
Optional Fields

Server Location: Poland
Contact Details: radekska@student.agh.edu.pl

✓ Enable Traps

Add

Figure 12 *- Configuration View - Adding SNMPv3 Configuration.*


127.0.0.1:8000/dashboard/config_network/

NETWORK CONTROLLER

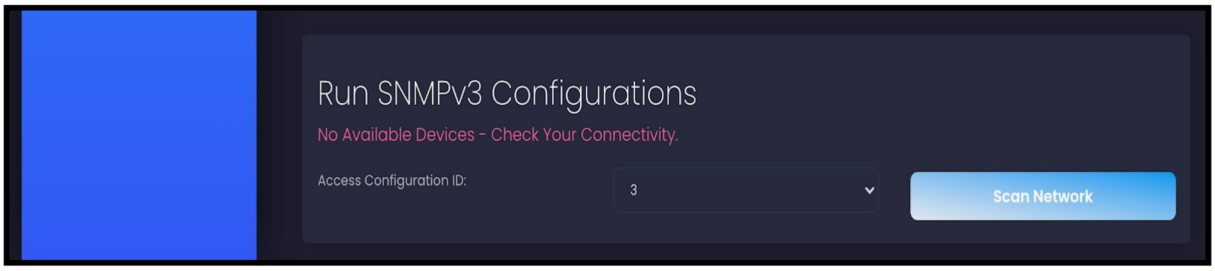
ID	USERNAME	PASSWORD	SECRET	NETWORK IP	CIDR	DEVICE OS
3	rskalban	cisco	cisco	192.168.8.0	24	Cisco IOS

SNMPv3 Configuration List Details

ID	GROUP NAME	SNMP USER	SNMP PASSWORD	SNMP ENCRYPT KEY	SNMP HOST	CONTACT DETAILS	SERVER LOCATION	ENABLE TRAPS
3	GROUP_1	rskalban	cisco12345	cisco54321	192.168.8.106	Poland	radekska@student.agh.edu.pl	True

Figure 13 *- Configuration View - Available SNMPv3 Configurations.*

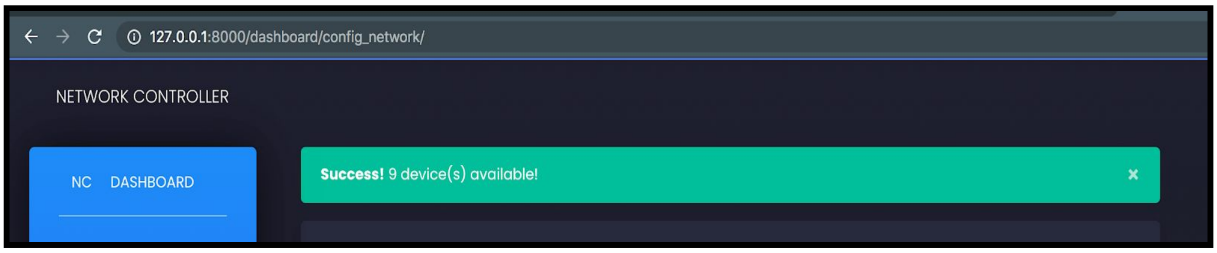
After adding both SNMPv3 and access configuration we are ready to scan the network for available devices.


Run SNMPv3 Configurations

No Available Devices - Check Your Connectivity.

Access Configuration ID: 3

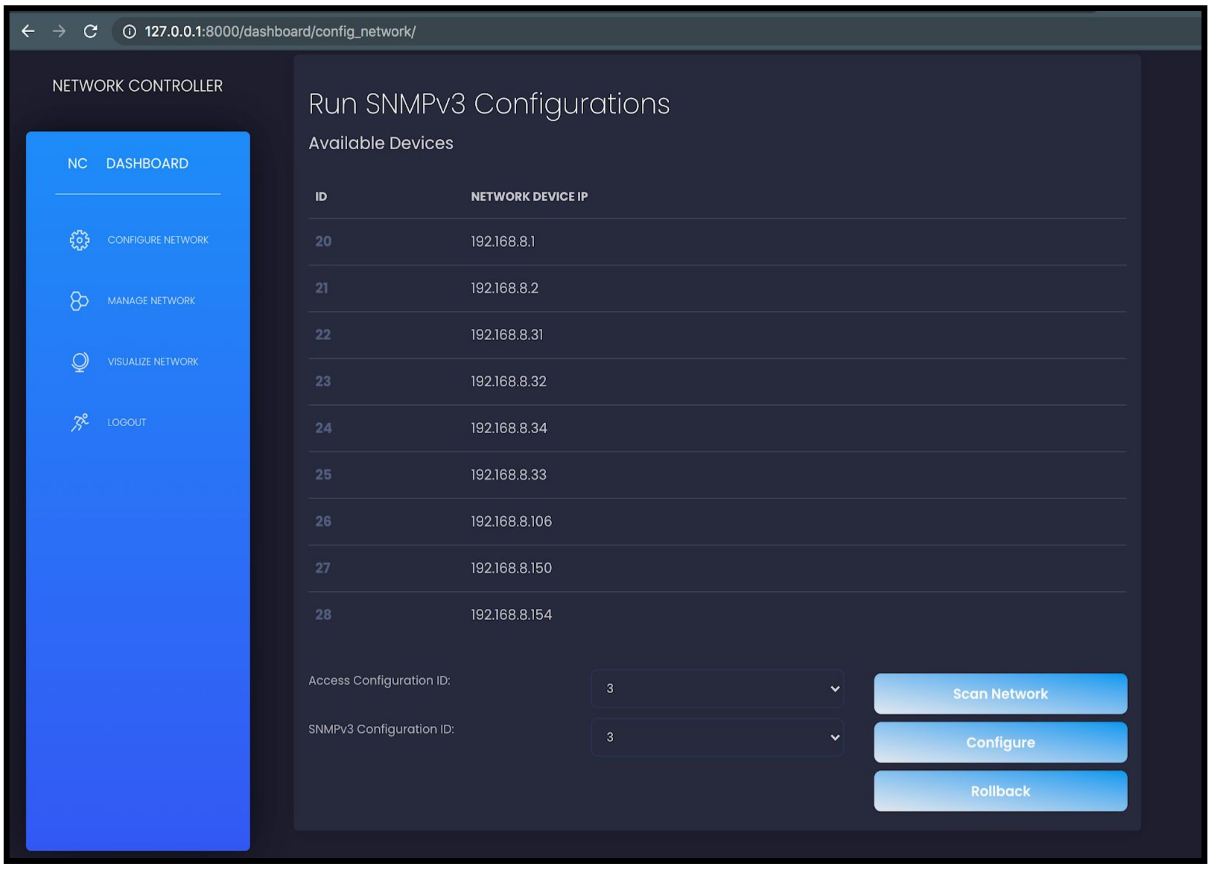
Figure 14 *- Configuration View - Scan Network Button.*


127.0.0.1:8000/dashboard/config_network/

NETWORK CONTROLLER

Success! 9 device(s) available!

Figure 15 *- Configuration View - Scan Network Output.*



Run SNMPv3 Configurations

Available Devices

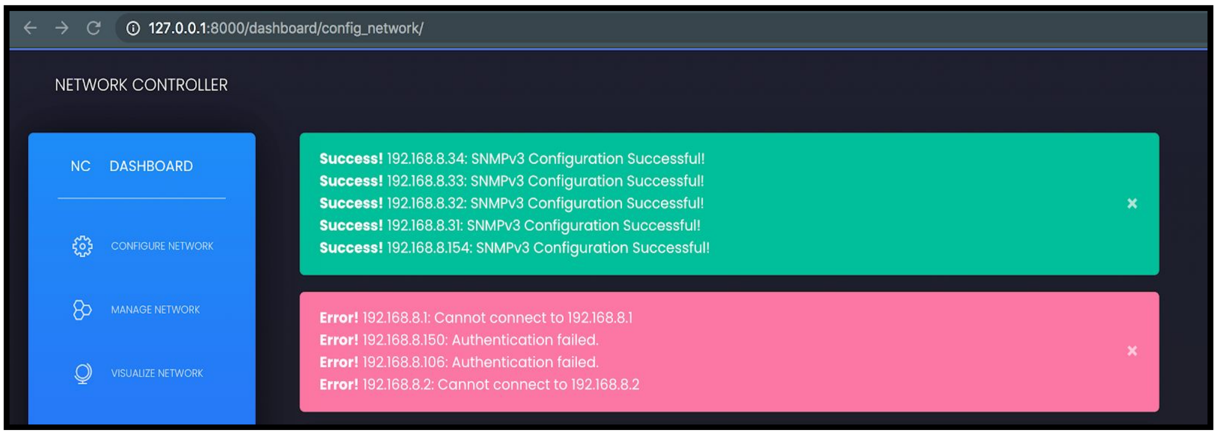
ID	NETWORK DEVICE IP
20	192.168.8.1
21	192.168.8.2
22	192.168.8.31
23	192.168.8.32
24	192.168.8.34
25	192.168.8.33
26	192.168.8.106
27	192.168.8.150
28	192.168.8.154

Access Configuration ID: 3

SNMPv3 Configuration ID: 3

Figure 16 *- Configuration View - List of Available Devices.*

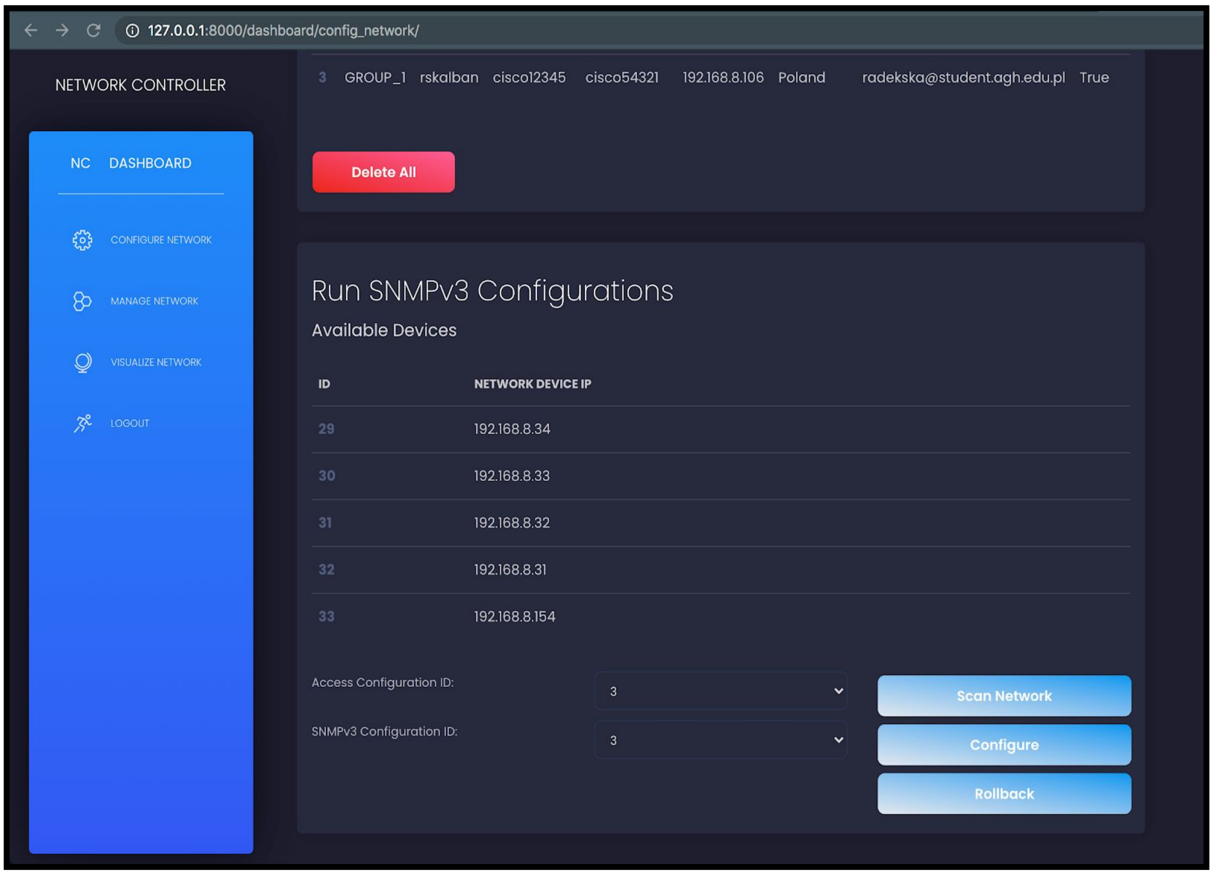
After we discovered all devices in the specified network, let's hit the *Configure* button to apply SNMPv3 configuration.



Success! 192.168.8.34: SNMPv3 Configuration Successful!
Success! 192.168.8.33: SNMPv3 Configuration Successful!
Success! 192.168.8.32: SNMPv3 Configuration Successful!
Success! 192.168.8.31: SNMPv3 Configuration Successful!
Success! 192.168.8.154: SNMPv3 Configuration Successful!

Error! 192.168.8.1: Cannot connect to 192.168.8.1
Error! 192.168.8.150: Authentication failed.
Error! 192.168.8.106: Authentication failed.
Error! 192.168.8.2: Cannot connect to 192.168.8.2

Figure 17 *- Configuration View - SNMPv3 Configuration Output.*


127.0.0.1:8000/dashboard/config_network/

NETWORK CONTROLLER

NC DASHBOARD

CONFIGURE NETWORK

MANAGE NETWORK

VISUALIZE NETWORK

LOGOUT

3	GROUP_1	rskalban	cisco12345	cisco54321	192.168.8.106	Poland	radekska@student.agh.edu.pl	True

Delete All

Run SNMPv3 Configurations

Available Devices

ID	NETWORK DEVICE IP
29	192.168.8.34
30	192.168.8.33
31	192.168.8.32
32	192.168.8.31
33	192.168.8.154

Access Configuration ID: 3

SNMPv3 Configuration ID: 3

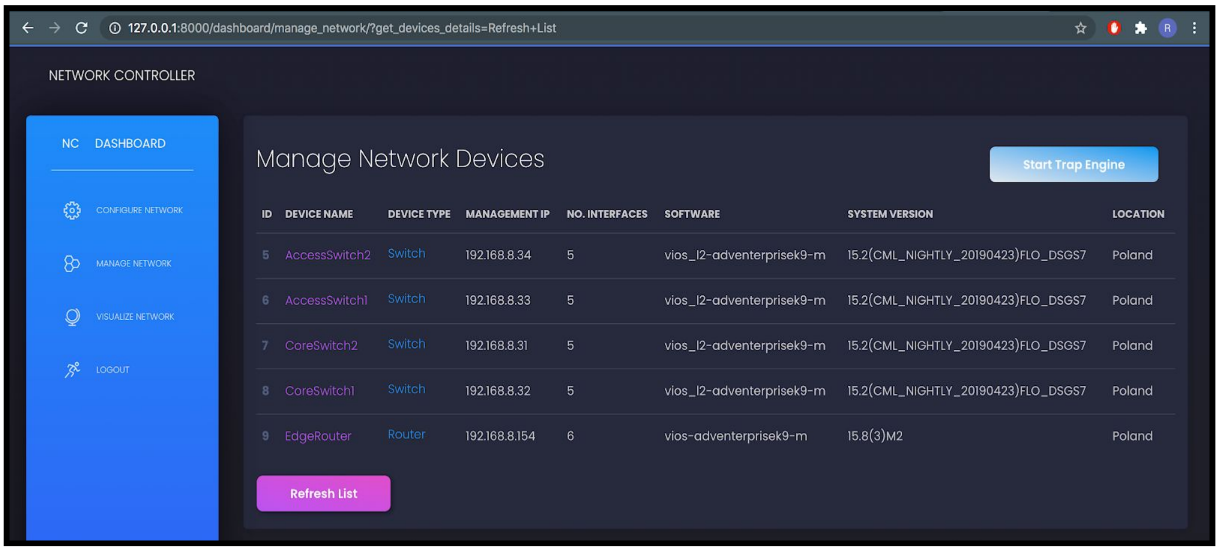
Scan Network

Configure

Rollback

Figure 18 - *Configuration View - Updated List of Available Devices.*

After successfully adding both SNMPv3 and access configuration, we can move on to the manage section. First thing to do there is to hit the *Refresh List* button to run *DeviceManager* which will collect all device data specified by the developer.


127.0.0.1:8000/dashboard/manage_network/?get_devices_details=Refresh+List

NETWORK CONTROLLER

NC DASHBOARD

CONFIGURE NETWORK

MANAGE NETWORK

VISUALIZE NETWORK

LOGOUT

Manage Network Devices

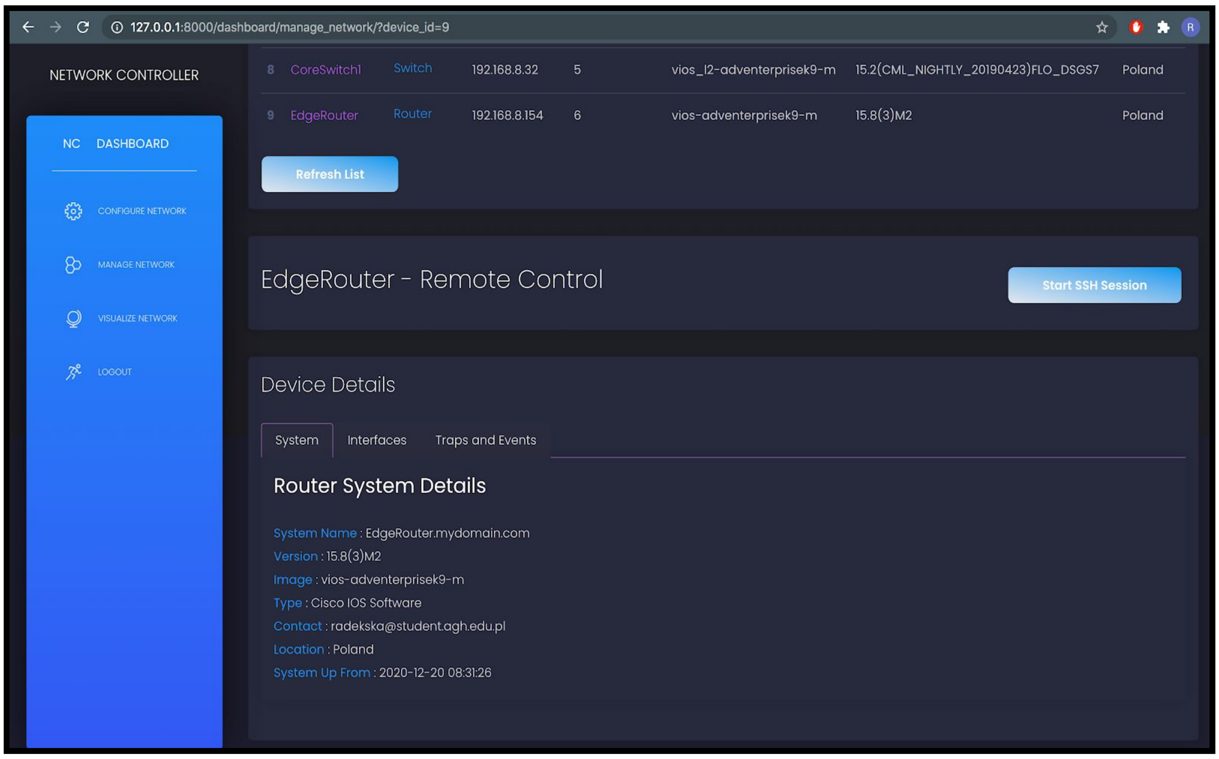
Start Trap Engine

ID	DEVICE NAME	DEVICE TYPE	MANAGEMENT IP	NO. INTERFACES	SOFTWARE	SYSTEM VERSION	LOCATION
5	AccessSwitch2	Switch	192.168.8.34	5	vios_l2-adventerprisek9-m	15.2(CML_NIGHTLY_20190423)FLO_DSGS7	Poland
6	AccessSwitch1	Switch	192.168.8.33	5	vios_l2-adventerprisek9-m	15.2(CML_NIGHTLY_20190423)FLO_DSGS7	Poland
7	CoreSwitch2	Switch	192.168.8.31	5	vios_l2-adventerprisek9-m	15.2(CML_NIGHTLY_20190423)FLO_DSGS7	Poland
8	CoreSwitch1	Switch	192.168.8.32	5	vios_l2-adventerprisek9-m	15.2(CML_NIGHTLY_20190423)FLO_DSGS7	Poland
9	EdgeRouter	Router	192.168.8.154	6	vios-adventerprisek9-m	15.8(3)M2	Poland

Refresh List

Figure 19 - *Manage View - List of Discovered Network Devices.*

When user clicks on the network device name, some new cards will pop up. The first one is for web based SSH terminal session and second one contains additional device information such as system, interface or traps and events details.



8	CoreSwitch1	Switch	192.168.8.32	5	vios_l2-adventerprisek9-m	15.2(CML_NIGHTLY_20190423)FLO_DSGS7	Poland
9	EdgeRouter	Router	192.168.8.154	6	vios-adventerprisek9-m	15.8(3)M2	Poland

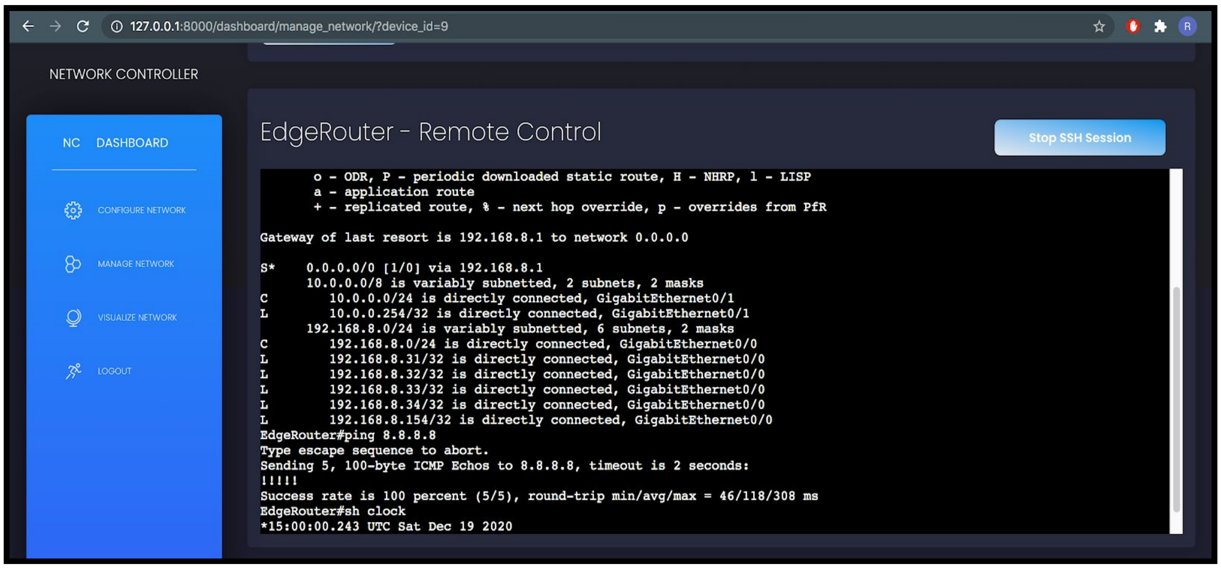
EdgeRouter - Remote Control

Device Details

Router System Details

System Name : EdgeRouter.mydomain.com
Version : 15.8(3)M2
Image : vios-adventerprisek9-m
Type : Cisco IOS Software
Contact : radekska@student.agh.edu.pl
Location : Poland
System Up From : 2020-12-20 08:31:26

Figure 20 *- Manage View - Device Details.*

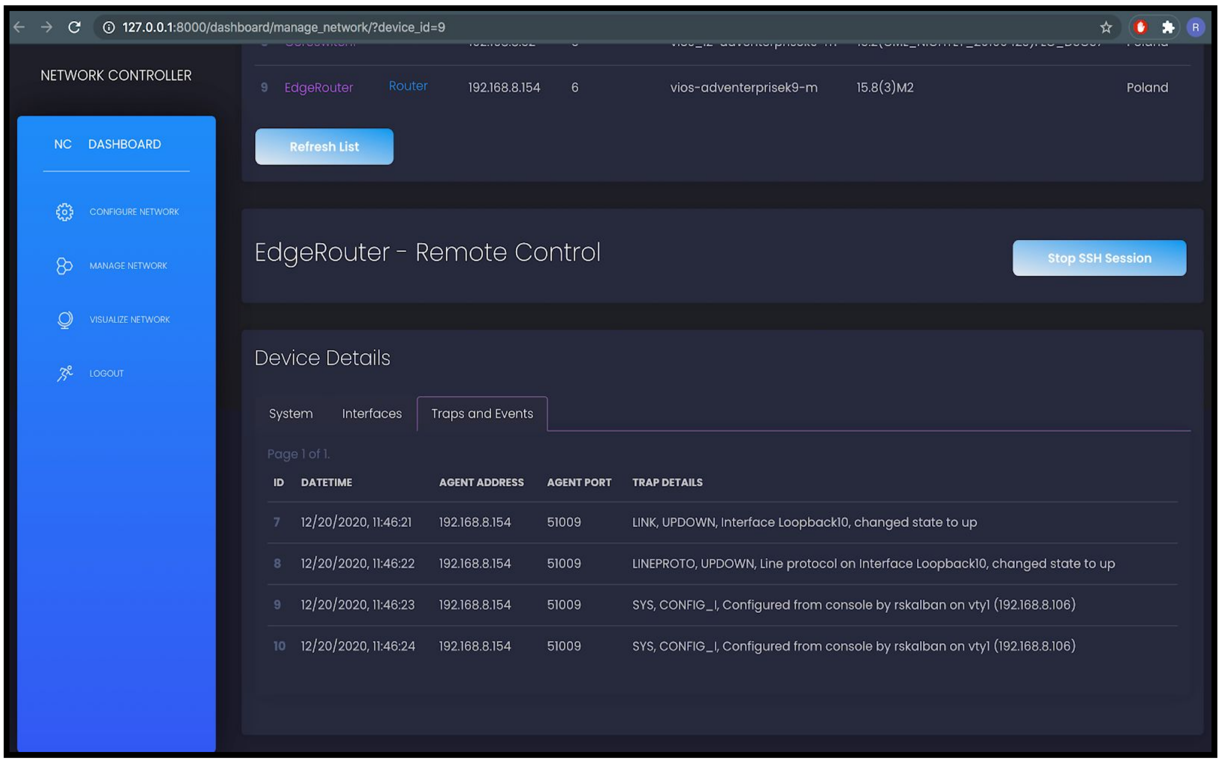


EdgeRouter - Remote Control

```
       o - ODR, P - periodic downloaded static route, H - NHRP, l - LISP
       a - application route
       + - replicated route, % - next hop override, p - overrides from PfR

Gateway of last resort is 192.168.8.1 to network 0.0.0.0

S*    0.0.0.0/0 [1/0] via 192.168.8.1
      10.0.0.0/8 is variably subnetted, 2 subnets, 2 masks
C        10.0.0.0/24 is directly connected, GigabitEthernet0/1
L        10.0.0.254/32 is directly connected, GigabitEthernet0/1
      192.168.8.0/24 is variably subnetted, 6 subnets, 2 masks
C        192.168.8.0/24 is directly connected, GigabitEthernet0/0
L        192.168.8.31/32 is directly connected, GigabitEthernet0/0
L        192.168.8.32/32 is directly connected, GigabitEthernet0/0
L        192.168.8.33/32 is directly connected, GigabitEthernet0/0
L        192.168.8.34/32 is directly connected, GigabitEthernet0/0
L        192.168.8.154/32 is directly connected, GigabitEthernet0/0
EdgeRouter#ping 8.8.8.8
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 8.8.8.8, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 46/118/308 ms
EdgeRouter#sh clock
*15:00:00.243 UTC Sat Dec 19 2020
```

Figure 21 *- Manage View - Web Based SSH Session.*

To enable network notifications, user have to click on *Start Trap Engine* button in top right corner of the manage view. After that, new updates will pop up in *Traps and Events* section.



9	EdgeRouter	Router	192.168.8.154	6	vios-adventerprisek9-m	15.8(3)M2	Poland

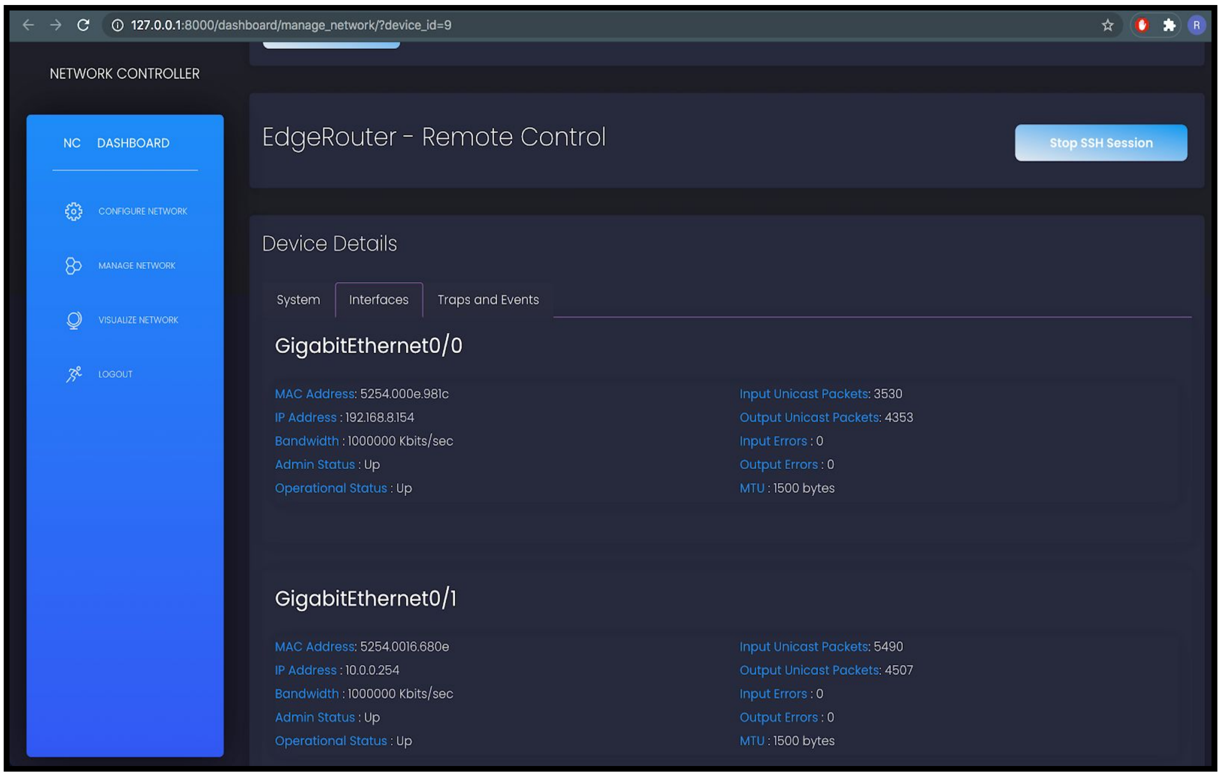
EdgeRouter - Remote Control

Device Details

Page 1 of 1.

ID	DATETIME	AGENT ADDRESS	AGENT PORT	TRAP DETAILS
7	12/20/2020, 11:46:21	192.168.8.154	51009	LINK, UPDOWN, Interface Loopback10, changed state to up
8	12/20/2020, 11:46:22	192.168.8.154	51009	LINEPROTO, UPDOWN, Line protocol on Interface Loopback10, changed state to up
9	12/20/2020, 11:46:23	192.168.8.154	51009	SYS, CONFIG_I, Configured from console by rskalban on vty1 (192.168.8.106)
10	12/20/2020, 11:46:24	192.168.8.154	51009	SYS, CONFIG_I, Configured from console by rskalban on vty1 (192.168.8.106)

Figure 22 *- Manage View - Traps and Events.*



EdgeRouter - Remote Control

Device Details

GigabitEthernet0/0

MAC Address: 5254.000e.981c
IP Address : 192.168.8.154
Bandwidth : 1000000 Kbits/sec
Admin Status : Up
Operational Status : Up
Input Unicast Packets: 3530
Output Unicast Packets: 4353
Input Errors : 0
Output Errors : 0
MTU : 1500 bytes

GigabitEthernet0/1

MAC Address: 5254.0016.680e
IP Address : 10.0.0.254
Bandwidth : 1000000 Kbits/sec
Admin Status : Up
Operational Status : Up
Input Unicast Packets: 5490
Output Unicast Packets: 4507
Input Errors : 0
Output Errors : 0
MTU : 1500 bytes

Figure 23 *- Manage View - Device Interfaces.*

Now, when we have found out what the manage section has to offer, we can move on to the last part of this project - visualize app.

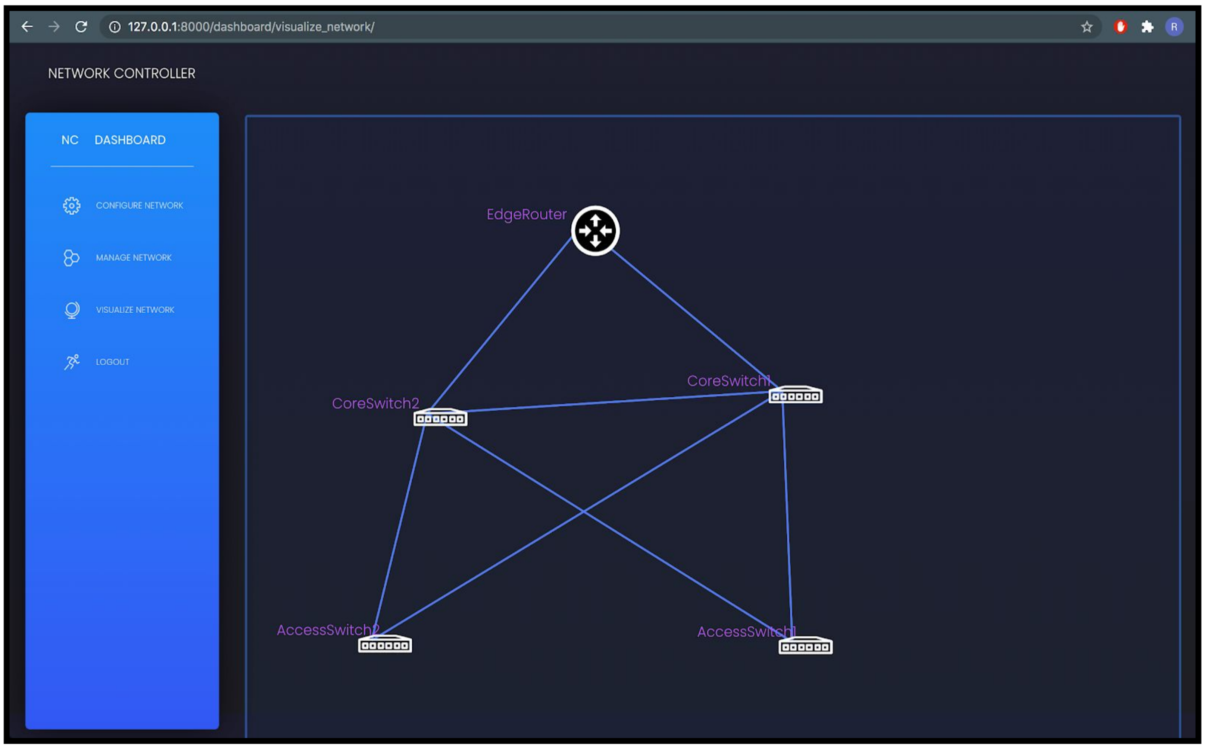
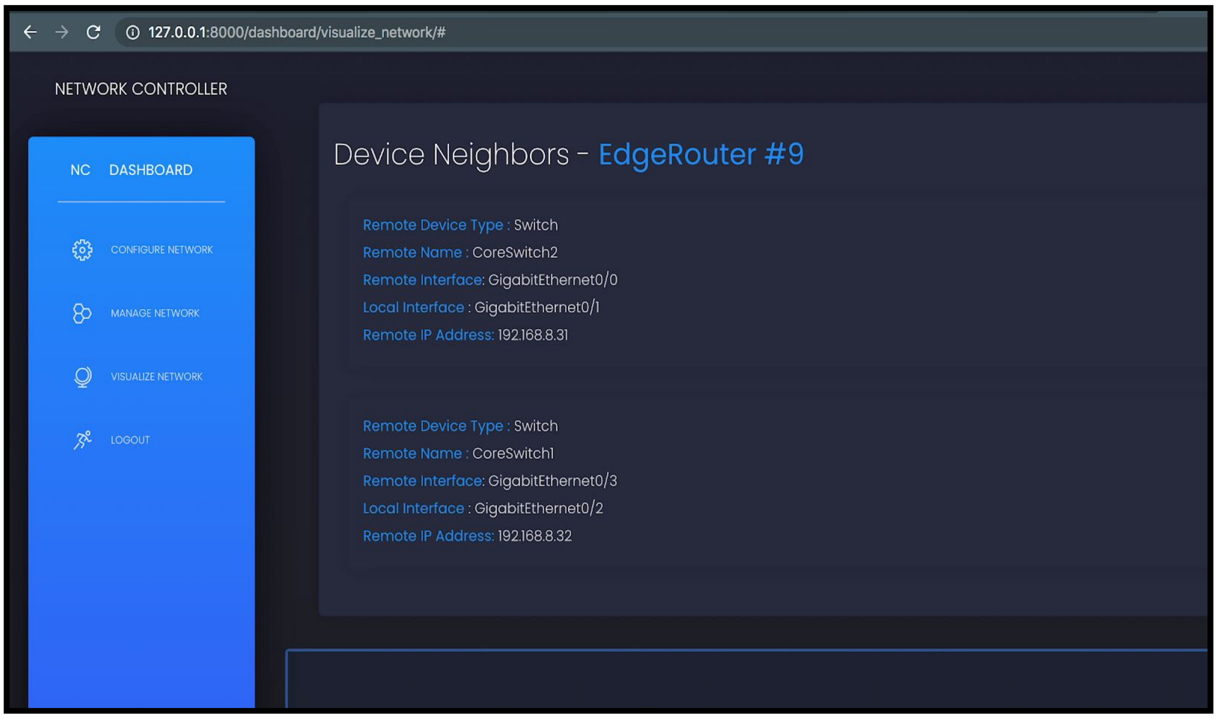


Figure 23 *- Visualize View - Dynamic Network Diagram.*

As we can see above, Network Controller discovered and rendered out the same diagram which we saw in CML^2.



Device Neighbors - EdgeRouter #9

Remote Device Type : Switch
Remote Name : CoreSwitch2
Remote Interface: GigabitEthernet0/0
Local Interface : GigabitEthernet0/1
Remote IP Address: 192.168.8.31

Remote Device Type : Switch
Remote Name : CoreSwitch1
Remote Interface: GigabitEthernet0/3
Local Interface : GigabitEthernet0/2
Remote IP Address: 192.168.8.32

Figure 24 *- Visualize View - Device Neighbors Details.*

5. Closure

Nowadays, Network Controllers are becoming more popular and powerful as they provide unified managing and controlling user friendly interfaces for our networks.

Modern IT industry is heading towards centralized solutions, rather than distributed ones as they are more efficient and easier for network incidents troubleshooting. Best way for providing such platforms are cloud-based solutions as they are more secure and stable. Also important is that clients which decide to use such products are free from designing and deploying servers, which might be highly time and resource consuming. In the near future, I would like to deploy my Network Controller as an open source cloud-based solution, to make it available for every interested network administrator.

The inspiration for this project came out not only from passion for computer networks and Python programming but also from a great desire to learn new technologies and develop my engineering skills. I would also like to thank my university for stimulating this will in me, it was a great journey.

6. Bibliography

[1] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: Infrastructure Automation.*

[2] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: Infrastructure Automation.*

[3] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: Infrastructure Automation.*

[4] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 7: RESTful API Requests and Responses.*

[5] *Django Documentation* - https://docs.djangoproject.com/en/3.1/topics/http/views/

[6] Ramalho Luciano 2015. *Zaawansowany Python. Jasne, zwięzłe i efektywne programowanie.*

[7] *Python Package Index* - https://pypi.org/

[8] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 2: Software Development and Design.*

[9] *Wikipedia HTTP*- https://pl.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[10] *Django Channels Documentation* - channels.readthedocs.io/en/stable/

[11] *MDN Web Docs* - developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/Writing_WebSocket_servers

[12] *Django Celery Docs* - docs.celeryproject.org/en/stable/django/first-steps-with-django.html

[13] *Wikipedia RabbitMQ* - en.wikipedia.org/wiki/RabbitMQ

[14] *Wikipedia SSH* - pl.wikipedia.org/wiki/Secure_Shell

[15] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: IP Services*

[16] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: IP Services*

[17] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: IP Services*

[18] *Wikipedia LLDP* - en.wikipedia.org/wiki/Link_Layer_Discovery_Protocol

[19] *Wikipedia JavaScript* - en.wikipedia.org/wiki/JavaScript

[20] *Wikipedia jQuery* - pl.wikipedia.org/wiki/JQuery

[21] *Wikipedia CSS* - pl.wikipedia.org/wiki/Kaskadowe_arkusze_styl

[22] Django Dashboard Source Code - github.com/app-generator/django-dashboard-black

[23] Chris Jackson, Jason Gooley, Adrian Iliesiu, Ashutosh Malegaonkar 2020. *Cisco Certified DevNet Associate DEVASC 200-901 Official Cert Guide - Chapter 15: Infrastructure Automation.*

Project repository - github.com/radekska/NetworkController