**Chapter 1. Introduction**

Technical Analysis

The application is responsible for performing various network related functionalities which includes General Info(displays the host name and IP address), Mac Address(Outputs the hardware address of a device), Subnet Calculator(calculates subnet), Decimal to Binary Conversion for IPv4, IPv6 validation(Checks if the given IPv6 is valid or not) and IPv6 expander(Expands the compressed address).

Goals

Certain functionalities are widely used in networking which serves very important purposes for which a good understanding of the related matter is required. With the increasing number of connected devices and jobs in this sector it’s hard to meet the requirement and having an application providing certain functionalities helps to bridge the gap and equips a low skilled employee such as help desk assistants with stronger set of skills. Many small human errors have led to various problems such as a wrong subnet calculation, wrong IP assignment, failing to validate IPv6 but with the help of the application these errors can be minimised.

Approach

To meet the objectives of the project finding the most frequently used and error occurring functionalities were to be identified after going through an extensive research on stack overflow, reddit and conducting surveys on social media the functionalities were selected. A simple and intuitive UI was needed.

Document Structure

----

**Chapter 2. Background**

Literature Review

There are many networking functionalities used on a daily basis with a wide range of options from high end features such as SSH to a remote host and as simple as converting a decimal IP into binary IP. However, considering the factor about frequent human errors and their level of impact certain functionalities are selected. The functionalities included on the project are listed below.

Introduction:

**Chapter 3. Methodology**

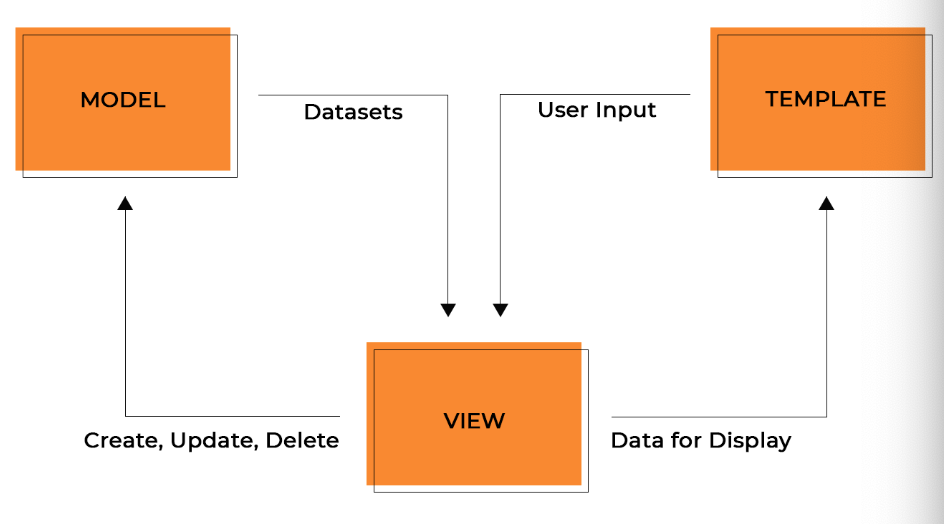
**Technologies**

Python

On a vast pool of programming languages Python is considered as a best tool for network programming. It provides two levels of access to network services. At a low level, we can access the basic socket support in the underlying operating system, which allows us to implement clients and servers for both connection-oriented and connectionless protocols. It has libraries that provides higher-level access to specific application -level network protocols, such as FTP, HTTP and so on. Libraries such as ipaddress can be a big help and the interest and resources on this part of python is ever increasing.  
<https://www.tutorialspoint.com/python/python_networking.htm>

Django

Django follows the Don’t Repeat Yourself (DRY) principle, making this framework time-efficient. It is well suited for high load systems and can decrease development time thanks to lots of helper objects. Django uses Model-View-Controller architecture which allows developers to change the visual part of an app and the logic part separately without affecting one another. The architecture has three layers Model, View and Template which are responsible for different things and can be used independently.



Models contain information about the data and are represented by attributes(fields). It holds custom methods, properties and other things related to data manipulation also models allows us to create, read, update and delete object in the original database.

The view executes three tasks: it accepts HTTP requests, applies logic provided by Python classes and methods, and provides HTTP responses to client’s requests. The view fetches data from a model and either gives each template access to specific data from a model and either gives each template access to specific data to be displayed or processes data beforehand.

Templates are files with HTML code that’s used to render data. The contents of these files can be static or dynamic. It presents the data.

<https://steelkiwi.com/blog/why-django-best-web-framework-your-project/>

Virtual Environment

They are isolated container containing all the software dependencies for a given project. To solve the problem of dependency conflicts among different versions of Python and Django.

All the dependencies can be listed in a file and get then all installed by using pip install -r requirements-test.txt

IPv4

It is the fourth version of the Internet Protocol. It uses 32-bit addresses which can be expressed in binary and decimal format. The addresses are categorised into five classes. Class A 0 – 127, Class B 128 – 191, Class C 192 – 223, Class D 224 – 239, Class E 240 – 255.

IPv6

It is the most recent version of Internet Protocol. It uses 128-bit addresses. Uses flow label field to identify packet flow for Quality of service handling. Devices are assigned a unique IP address for identification and each device can be routable.

Atom

It was the best suited Integrated Development Environment (IDE) for this project, getting a terminal access from the IDE itself and allowing to open multiple terminal windows was a great help. Atom is integrated with GitHub which worked as an amazing source of backup.

**Chapter 4**

**System Design and Specification**

The functionality included in the application are described in an abstract format here.

**General Info**

It is a simple yet very useful function which gets the IP and the hostname configured on the network. To get access to the information required socket library is used which retrieves the hostname and Ip address being configured to the device. The function get\_general\_info holds two variable host\_name and ip\_address which gets the host name and ip address using the function socket which returns the IP address and host name as a dictionary with key ‘host\_name’, ‘ip\_address’ and values host\_name and ip\_address

**Mac Address**

The module uuid and regex is implemented. UUID stands for Universal Unique Identifier which with the help of regex extracts the MAC address of the device. Function get\_mac\_address uses regex and function uuid.getnode() which extracts the mac address of the device.

**Subnet Calculator**

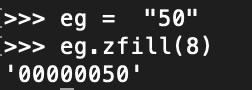
Class SubnetForm takes forms.Form as an argument where ip\_address and subnet\_mask is defined as a CharField with their respective labels and maximum length allowed.After taking the input the IP address goes through a function clean\_ip\_address. Django form validates all the data it receives and to access the cleaned data self.cleaned\_data is being used. Now, the address is split into dots (.) and stored as 4 octets, it goes through an if statements where the length of ip\_octets is checked which should be equals to 4 as ipv4 addresses are 32-bit long and the first octet of the ip address should be <= 1 and <= 223, to be valid IP the first octet can’t be = 127 as it is the loopback address, first octet or second octet can’t be 169,254 as it is APIPA reserved address and the other octets should be in a valid range of 0 to 255. After going through all the conditions an IP address is returned if the conditions aren’t met a ValidationError is raised.

Now, once the IP is cleaned the mask is cleaned, which is achieved under the function clean\_subnet\_mask. A list of all the valid masks are stored on the list masks which includes [255,254,248,224,292,198,0]. The clean subnet mask is retrieved and mask\_octets stores the split subnet mask using dot(.) as a separator. Once the octets are separated it goes through various if statements. First the length of the mask octets must be 4, the first octet is 255, second, third and fourth octet must be in the pre-defined list mask and the first mask octet should be >= second octet which should be >= third octet and the third octet should be >= fourth octet in this format mask\_octets[0] >= mask\_octets[1] >=, mask\_octets[2] >= and mask\_octets[3] >= 4. After meeting the above-mentioned requirements, the mask is considered valid and a clean, valid mask is returned. If the mentioned conditions aren’t met a ValidationError is raised.

After validation of the IP address and subnet mask we move to calculate\_subnet function where we clean the ip\_octets and mask\_octets and store the split version using dot as a separator. An empty list mask\_octets\_binary is made to store the binary version of mask but it isn’t initialised at this stage. To get all the elements in the mask\_octets we run through a loop where all the elements are converted to binary using bin(int(octet)).lstrip(‘0b’) which takes away the starting 0b on the left side as we get 0b after converting a decimal to binary.



We append the binary\_octet to mask\_octets\_binary using .zfill(8) which appends 0 until the variable is 8 chars long



After having 8 bits long binary mask we make a variable binary\_mask which contains all the values stored in mask\_octets\_binary using .join

Now, we have to count the host bits which is donated by 0 for which we use no\_of\_zeroes = binary\_mask.count(“0”) and to calculate the no of subnet bits since the address is 32 bits long we use no\_of\_ones = 32 – no\_of\_zeroes. To find the number of legal host we have to calculate the 2 ^ no of zeros and subtract 2, no\_of\_hosts = abs(2 \*\* no\_of\_zeroes -2). To return the absolute value of the number abs is used.

Wildcard masks are used to specify a range of network addresses. They are commonly used with routing protocols like OSPF and access lists. Wildcard mask is 32bits long. It acts as an inverted subnet masks, but with wildcard mask, the zero bits indicate that the corresponding bit position must match the same bit position in the IP address. The ones bit indicates the corresponding bit position doesn’t have to match the bit position in the IP address.

To obtain the wildcard mask we define an empty list wildcard\_octets. Now, we run through a loop for all the elements in mask\_octets and we store the value of 255 – int(octet) on it and the value is appended as a string in the list wildcard\_octets. Now, we store all the wildcard\_octets in a variable wildcard\_mask. At this stage all the requirements for subnet mask and wild card mask is achieved.

Now, the IP address needs to be converted into binary format which follows identical approach as subnet mask, we define a empty list ip\_octets\_binary and we run through a loop in ip\_octets which contains the IP and using the bin method we convert the integer octet and use lstrip(‘0b’), the output is appended to the empty list using .zfill method to make it 8 characters long if not and a binary\_ip is defined to store the binary IP address.

To get the network address and broadcast address from the binary strings we define two variables network\_address\_binary and broadcast\_address\_binary. For network address we take the binary IP find the no of ones on it and multiply char 0 with the number of zeroes present in the IP.

E.g. 192.168.4.3 255.255.255.0 Network address = 192.168.4.0 as 255 represents 11111111 in binary so all the 1 bits are considered for network and the 0 bits are considered for hosts as 0 in binary for a IP format is 00000000.  
network\_address\_binary = binary\_ip[:(no\_of\_ones)] + "0" \* no\_of\_zeros

For broadcast address, the remaining bits after network address is filled in the 0’s with 1 which then becomes the broadcast address

E.g. all the 0 bits in above address is filled with 1’s 11111111 which converted to decimal is 255 192.168.4.255 becomes the broadcast address in this case.

broadcast\_address\_binary = binary\_ip[:(no\_of\_ones)] + "1" \* no\_of\_zeros

Once, the binary conversion is completed the addresses are converted back into decimal format. For this we need to be in a range of 0 to 32 with step of 8 as the octets are 8 bits long. For each IP address bit in the range returned by the range function, we are going to slice the binary network address starting from that bit. Let’s see the first one which is positioned at index 0 up to but not including that index plus 8 this means that using this slice right here we are extracting the first octet which contains the first 8 bits of the IP address indexes 0 through 7 then the second octet which contains next eight bits of the address meaning indexes 8 to 15 and so on. So, we will end up with four slices of the binary IP address 0 to 8, 8 to 16, 16 to 24 and 24 to 31 these being the indexes, each slice goes up to but not including the index on the right side of the colon. Finally, we are appending each slice to the empty list we created.

We define an empty list net\_ip\_address and use for loop to iterate over the list which makes the conversions from binary to decimal to string and appends each octet to the list. Int function helps us to convert binary to decimal and converts the decimal to string

To get the network address we join the element together this way we obtain the network address in decimal format the same procedure is applied for broadcast address. After obtaining the decimal format of IP and mask. All the values are returned as a dictionary with the variable name being the key and output being the value.

Importing the Modules

**Converter**

Class ConvertForm takes in forms.Form as an argument where ip\_address is defined as a CharField with label “Enter an IP address” of max\_length 15 to take the user input. The IP address received from user input is cleaned and split using dot(.) as a separator. The IP is formatted and converted using {0:08b}.{1:08b}.{2:08b}.{3:08b} for the respective index where 0 is added until the characters are 8 bits long.

E.g.



**Expand IPv6**

The functionality of ipaddress library is used here which provides the capabilities to create, manipulate and operate on IPv4 and IPv6 addresses. Class ExpandForm takes in forms.Form as an argument where ip\_address is defined as a CharField with label “Enter an IP address” of max\_length 15 to take the user input. After the input is taken. IPv6 addresses can be compresses using three rules as mentioned below,   
  
1) IPv6 Zero Compression where we can represent groups of zeroes by one double colon(::) but we can perform this only once in our address e.g.  
  
Original IPv6 address:  
2001:1265:0000:0000:0AE4:0000:005B:06B0  
IPv6 address with zero compression:  
2001:1265::0AE4:0000:005B:06B0

2) IPv6 Leading Zero Compression here we can eliminate the starting zeros from any block if we have all zero in a bock we can represent with one 0 e.g.   
  
Original IPv6 address:  
2001:1265:0000:0000:0AE4:0000:005B:06B0  
IPv6 address with Leading zero compression:  
2001:1265:0:0:AE4:0:5B:6B0

3) Both Zero and Leading zero compression we can use both zero and leading compression together following the rules of these compression techniques.

Original IPv6 address:  
2001:1265:0000:0000:0AE4:0000:005B:06B0  
Both compression:  
2001:1265::AE4:0:5B:6B0

Inside function expand\_ipv6 the ip\_address is cleaned and we go inside the try block where we check if the given IP is an IPv6 address or not using ipaddress.IPv6Address function and if true the exploded version on the address is returned alongside the IP address provided by the user else if there is an error an exception AddressValueError is raised with the given IP and error message to user.

**Valid IPv6**

Class ValidIPv6Form takes forms.Form as an argument where ip\_address is defined as a type character with a label “Enter an IPv6 address” with a max length of 39 characters. To process the input function valid\_ipv6 is used. The cleaned format of IP address is retrieved. Two variables addr and ip is defined and inside the try block the ip is checked if it belongs to multicast using the function provided by ipaddress library and so on it is checked if it is private, global, link local, Unique local, loopback, ipv4 mapped, IPv6 RFC 3056, and IPv6 RFC 4380 addresses which all are the different types of IPv6 address which serves following purposes.

Private IPv6 Address: These addresses are reserved for local use and are not public address space. Packets with these addresses in the source or destination fields are not intended to be routed on the public Internet but are intended to be routed with the local network.

Global Ipv6 Address: These are publicly routable addresses. Global addresses start at 2000::/3

Link-Local Address: These addresses are used on a single link or a non-routed common acess network, such as an Ethernet LAN. Link-Local Address starts with FE80::/10

Unique local Address: These addresses are intended for non routing purposes over the Internet. They were designed to replace site-local addresses. They allow communication throughout a site while being routable to multiple networks. Unique local address starts with FC00::/7

Ipv6 Loopback Address: This address when used will loop the packets back on the same interface without even leaving the interface. It is generally used for development and testing of network applications. The prefix for loopback address is ::1/128

IPv4-Mapped Address: Theses addresses are used to embed IPv4 addresses in an IPv6 address. One use for this is in a dual stack transition scenario where IPv4 address can be mapped into an IPv6 address. E.g. ::ffff:192.0.2.47

IPv6 RFC 3056 Address: Used with 6 to 4 tunnelling, which is an IPv4 to Ipv6 transition system. The structure allows IPv6 packets to be transmitted over an IPv4 network without the need to configure explicit tunnels.

IPv6 RFC 4380 Address: This is also called Teredo. It is a mapped address allowing IPv6 tunnelling through IPv4 NATs. The address is formed using the Teredo prefix, the server’s unique IPv4 address, flags describing the type of NAT, the obfuscated client port and the client IPv4 address.

**Flow diagrams**

**SUBNET**

Importing the Modules

Valid

Checking Subnet Mask validity

Checking IP address validity

Convert mask to binary string

Convert IP to binary strings

Counting host bits in the mask and calculating number of hosts/subnet

Obtaining the wildcard mask

Obtain the network and broadcast addresses from the binary strings

Printing the results for selected IP/mask

General Info

Import library socket

Functions of socket is used to extract the host name and IP address

The output is displayed to user

Mac Address

Import library uuid, re

Library uuid function extracts the MAC address and uses regex to match a pattern

The output is displayed to user

IP converter

Importing the modules

Check the IP

Valid

Not valid Display an error message

Convert the IP address to binary

The output is displayed to user

Expand ipv6

Import the modules

Not valid Display an error message

Valid the Ip

Expand the IP

The output is displayed to user

Valid ipv6

Import the modules

Valid the Ip

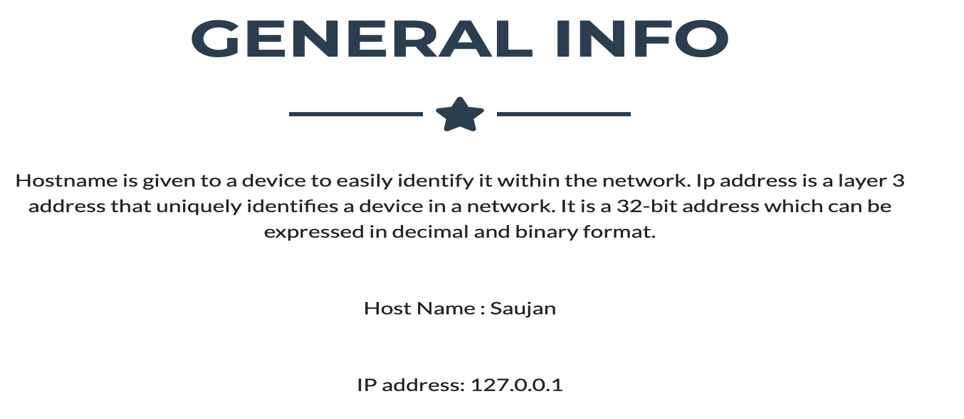
Not valid Display an error message

Compare the IP with different functions

The output is displayed to user

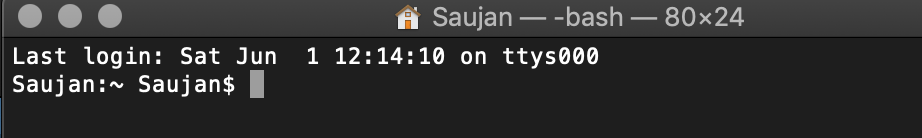
Test cases

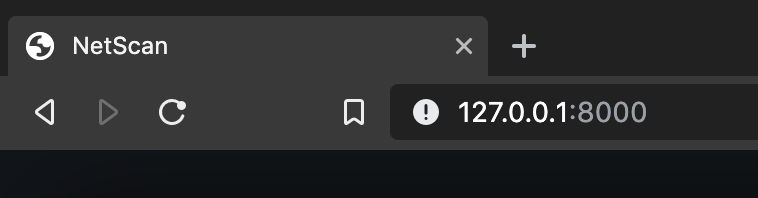
While using the functionality on the application the output displayed is



Checking it manually

The host name

  
The Ip in use

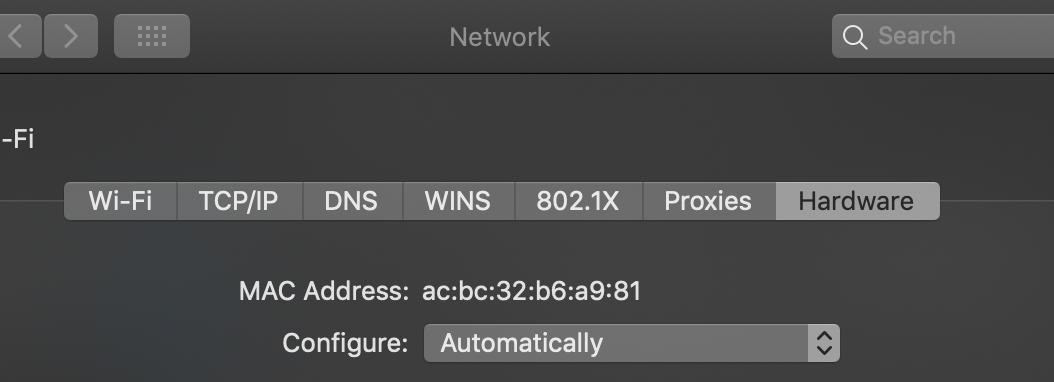


MAC address

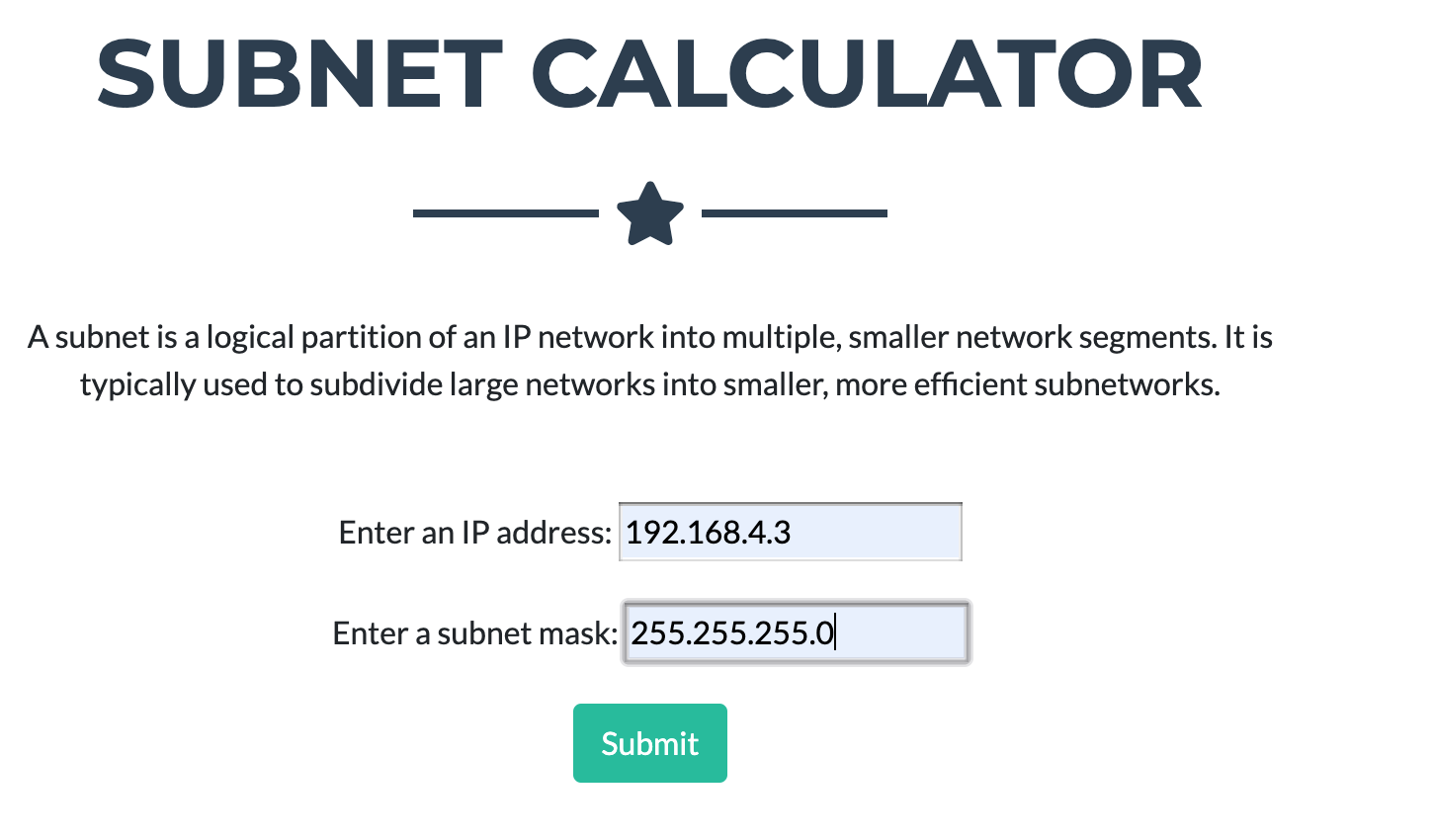
Output from the application

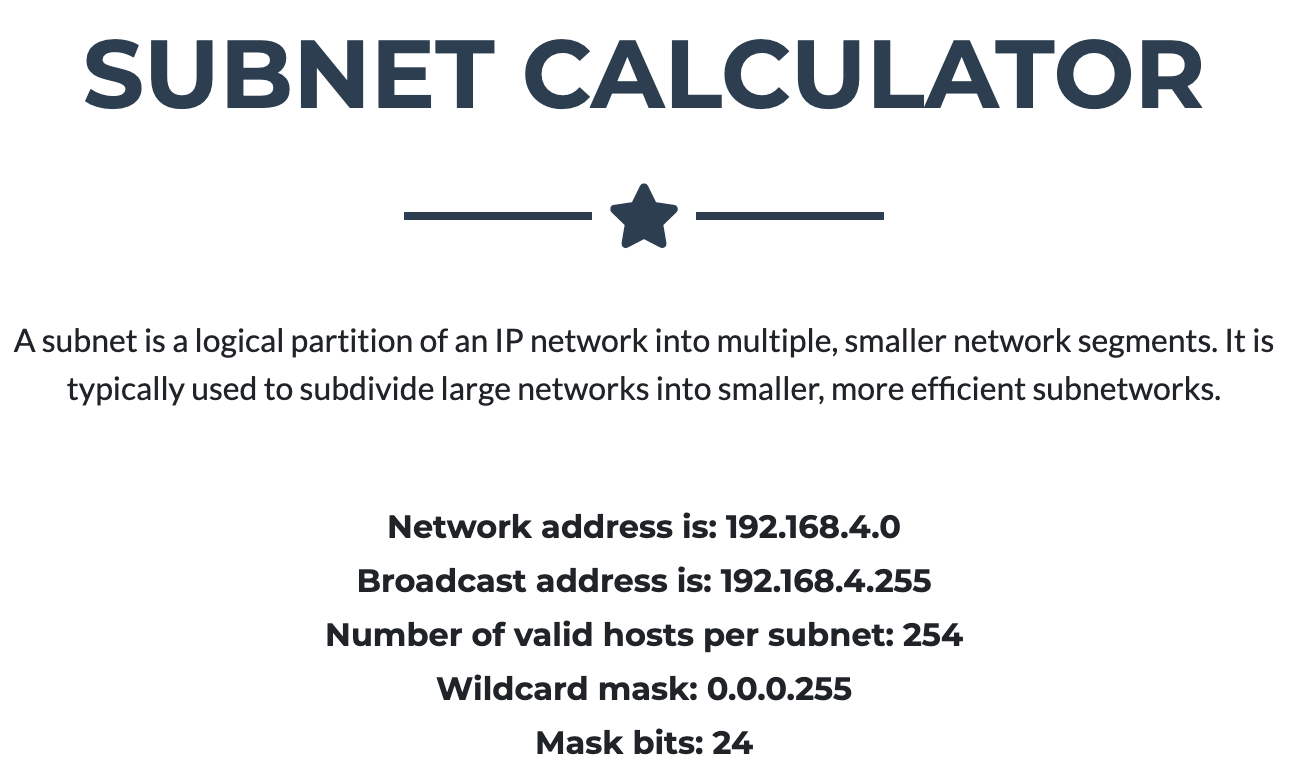


Checking it manually

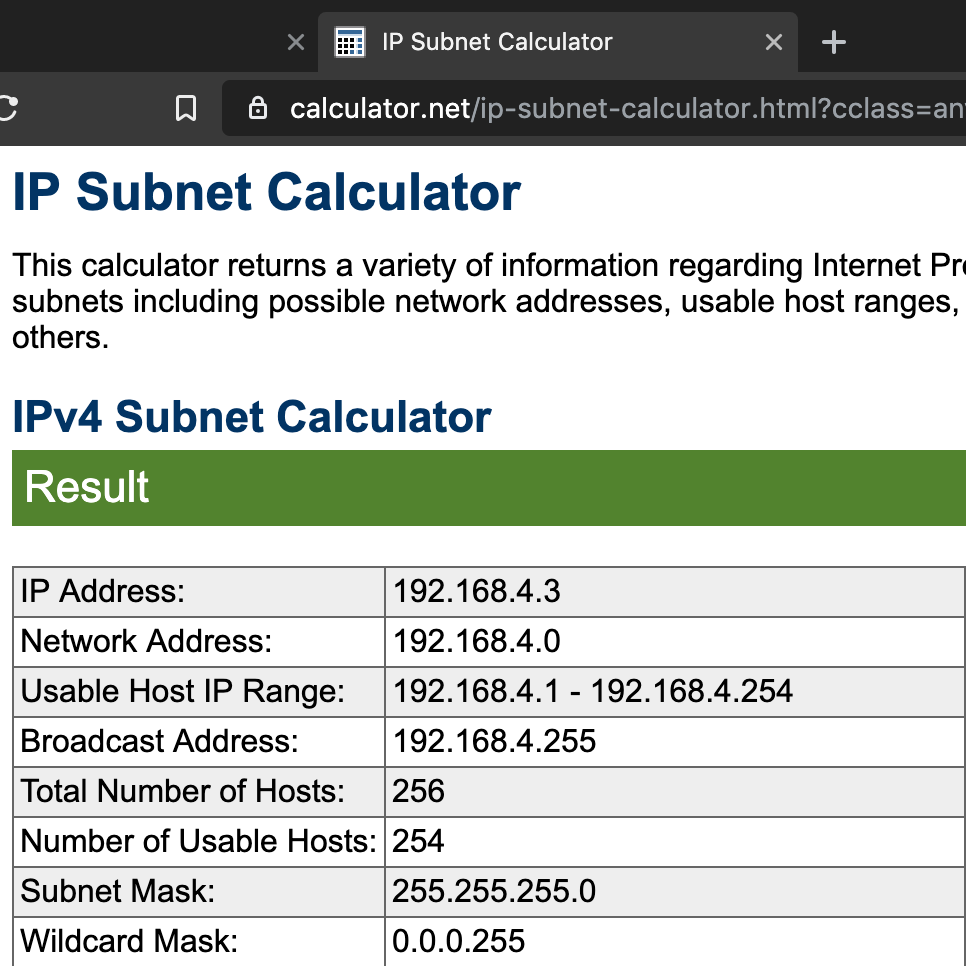


Subnet Calculator



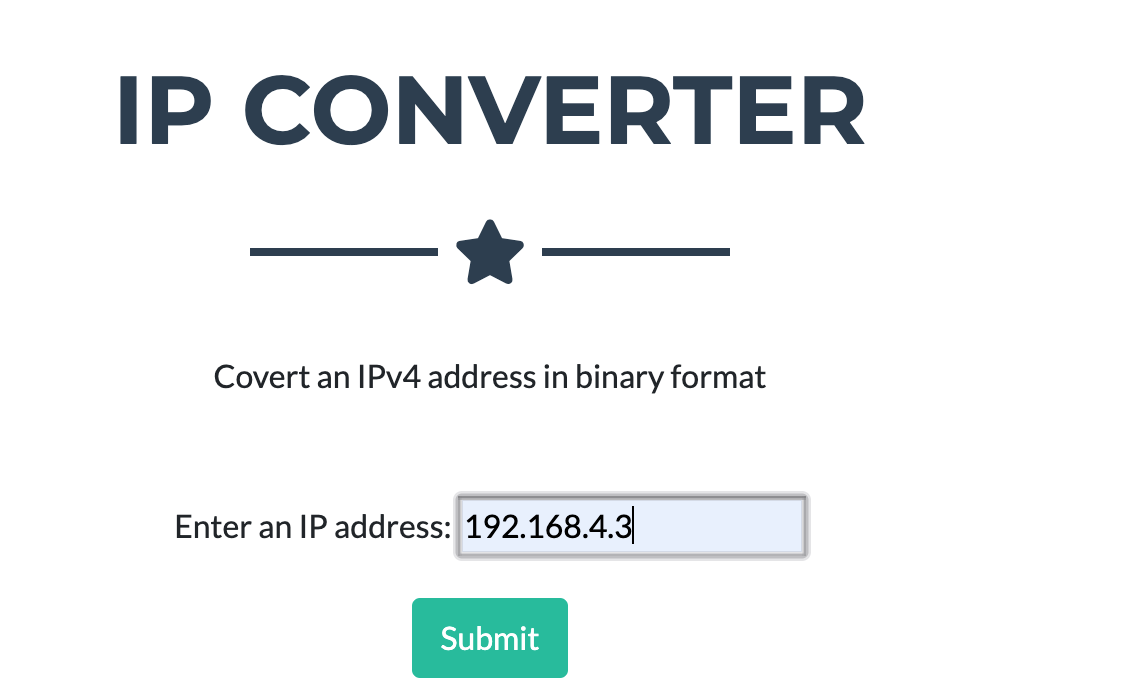


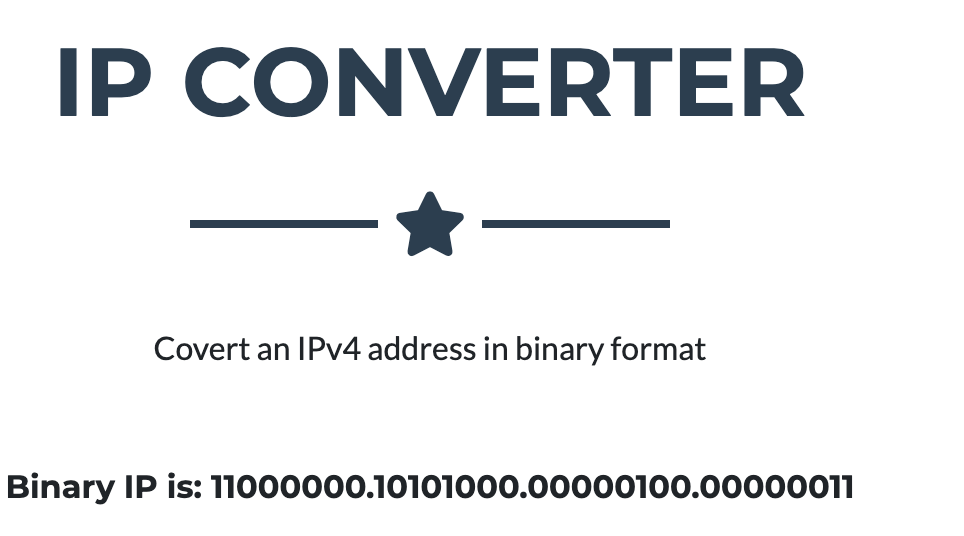
Checking it in a different web application



<https://www.calculator.net/ip-subnet-calculator.html>

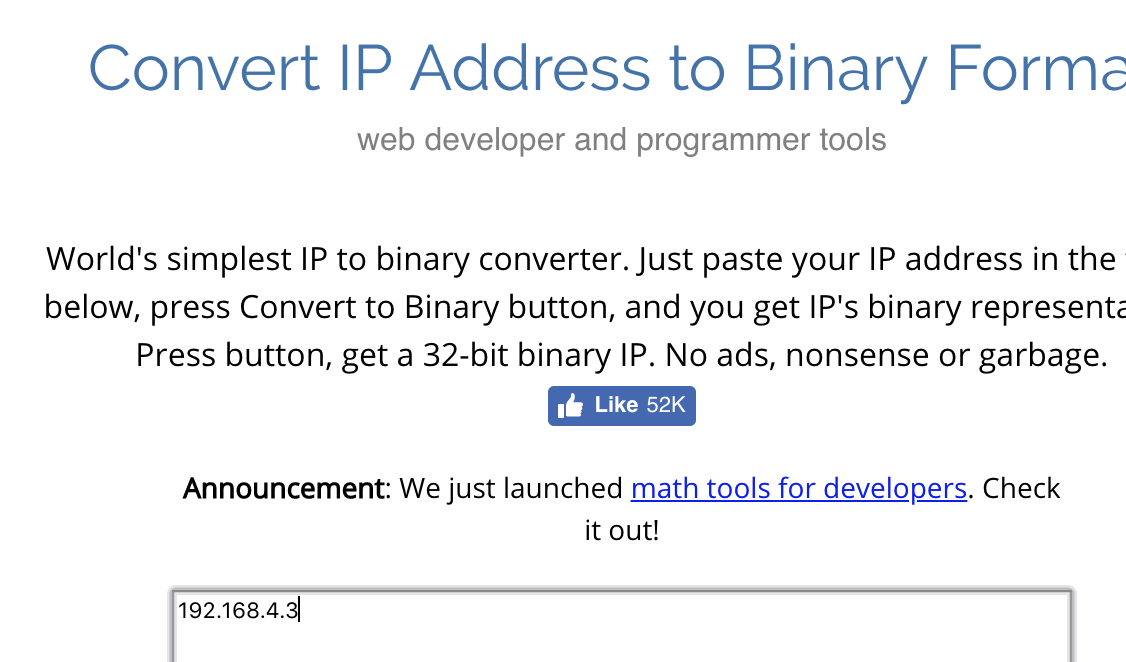
IP converter

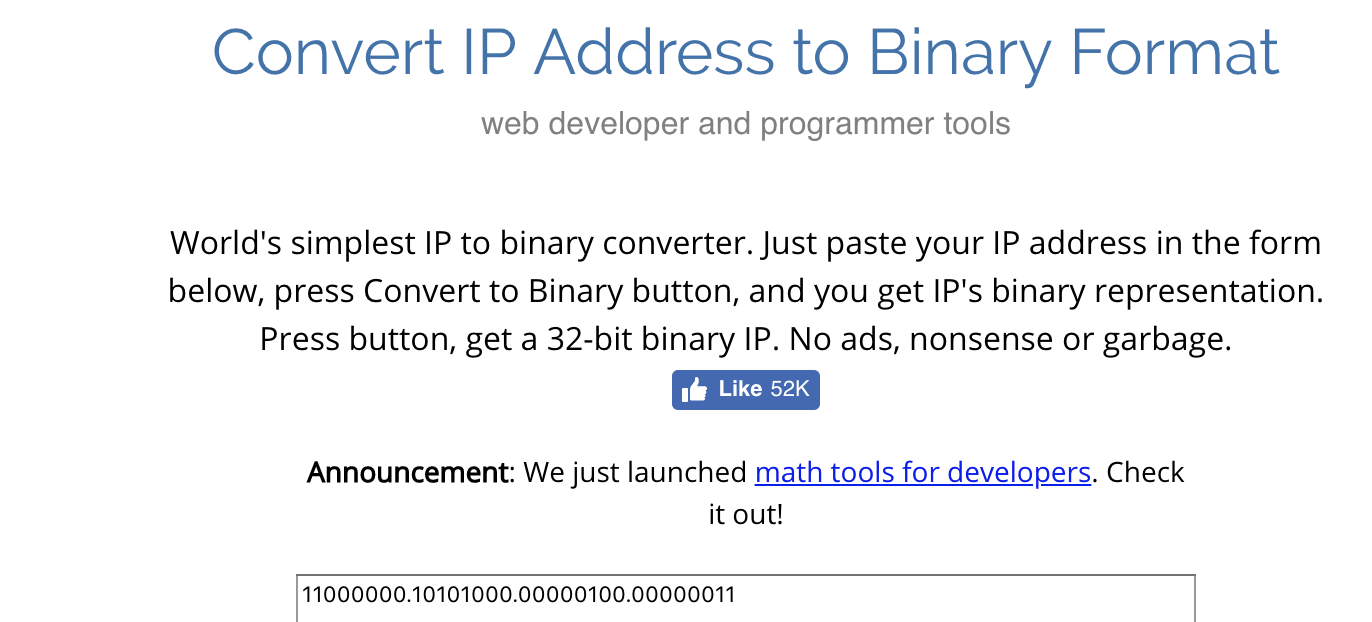




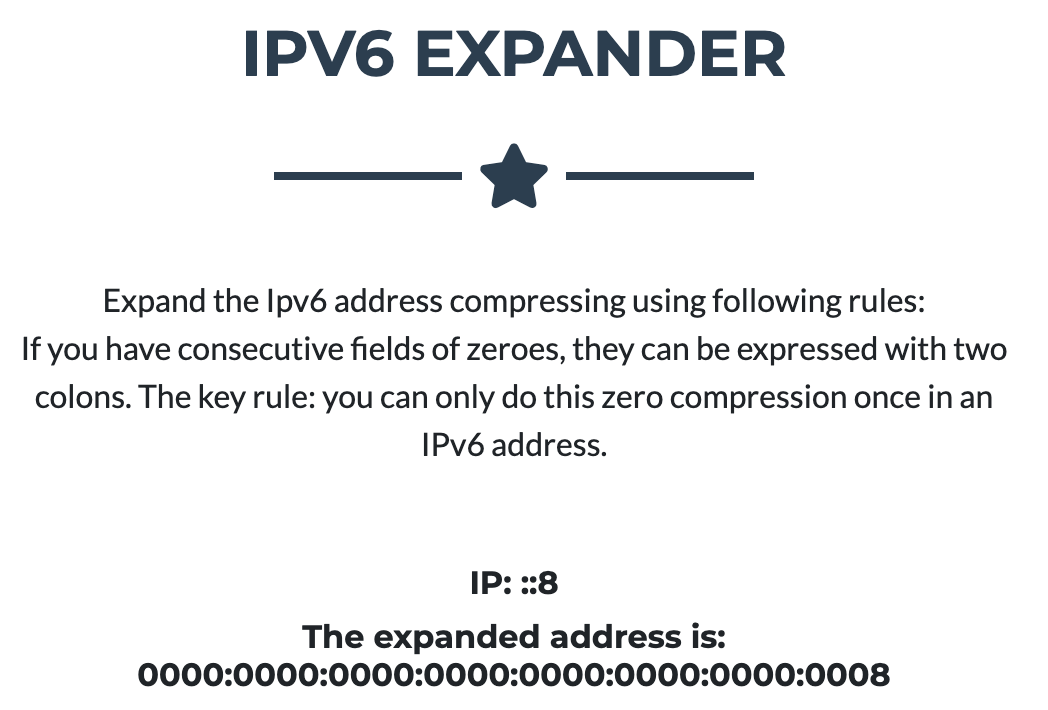
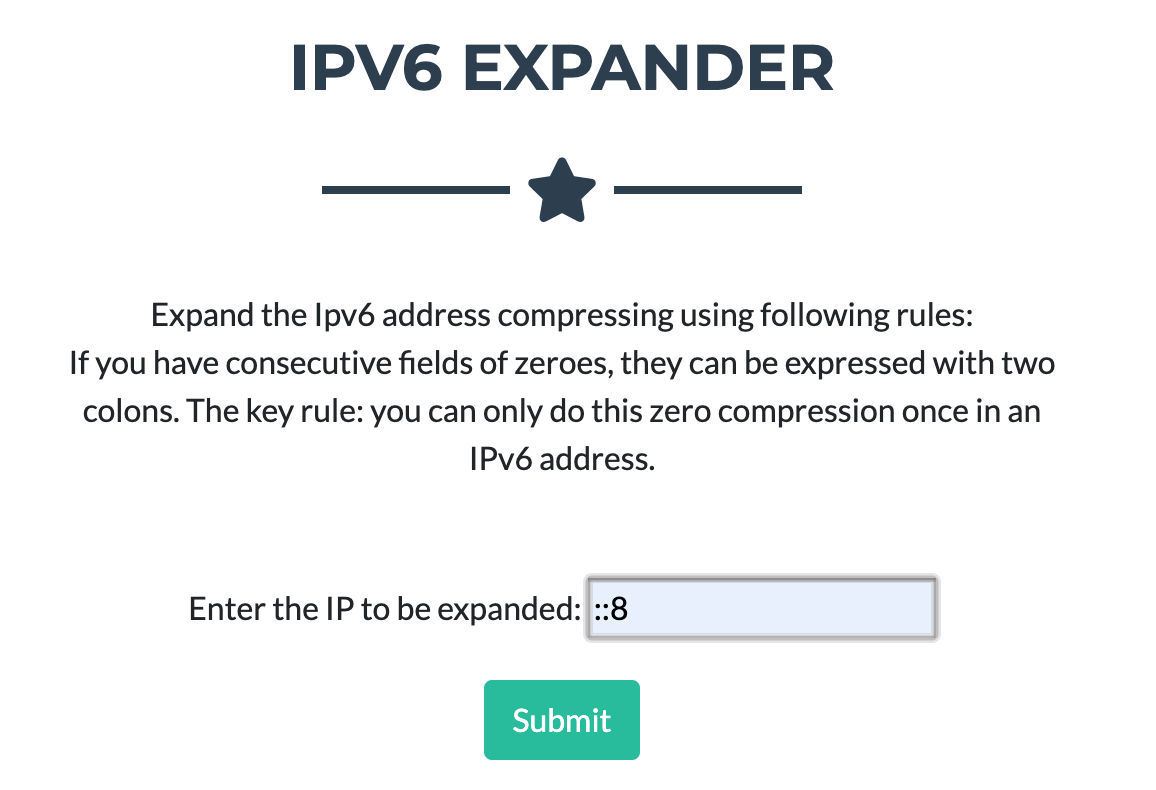
Checking it in a different site

<https://www.browserling.com/tools/ip-to-bin>





Expand IPv6



Valid Ipv6

