Ch1: INTRODUCTION:

In this documentation we will talk about NCP (network control program) which is a technical solution for companies that need to control internet access in their computers and even at home if parents want to control children access to the internet. NCP mainly depends on how the network manipulate the requests and how devices in the same network communicates to each other. So, we will discuss in this section many topics that reader should be aware of before start in this documentation. First, we will talk about what is NCP and what it does, how local network deals with requests, we will discuss the main logic of the program and its basic notations, what is web API and how to deal with, what is NodeJS. What is the difference between client and server?

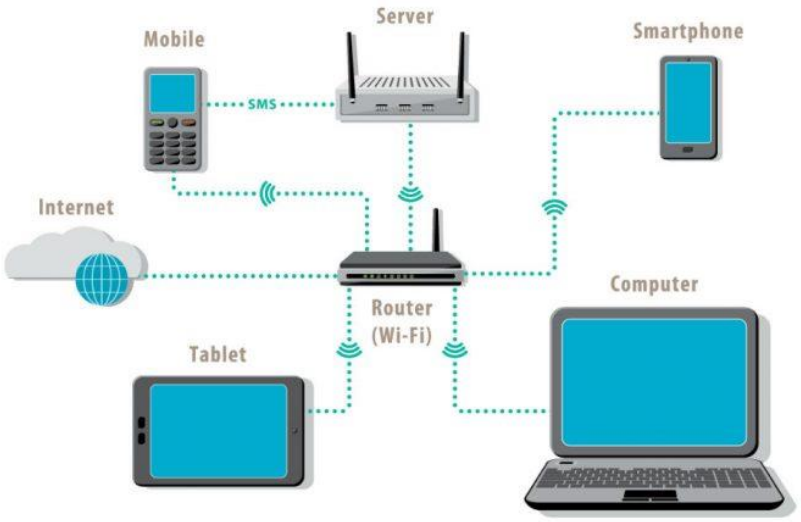
1.1- What is NCP:

NCP is web API that controls the computer via custom requests sent by a web application Installed in any android device. The main befit of this project is to control the internet access via firewall services. NCP depends mainly on a NodeJS server running on port 5050 which is listening to any request sent to it and keeps the connection alive between admin and client all time. The most brilliant feature of NCP is its simplicity to use, you can easily install, use and control every device in the network with just a finger click. The mobile app is developed with ionic framework which is a framework for angular to deal with native apps to run on different kernels like android and IOS. But this version is specified for android for developing and debugging.

With NCP you can easily send and receive messages from computer user, stream your voice to the computer, send files from mobile to computer and control the internet access. To do so it consists of two parts. First part is the desktop program that allows the connectivity and control for the program. It is a simple express server running over HTTP server that programmed with MVC technology to be modular and scalable. The second part of the program is an APK file contains an android app called (NETWORK CONTROLLER) which simplify the communication between admin and express server running on computers. The communication between them must be over specific routs known only by them in order not to be hacked with anyone. NCP must run as administrator to be able to control OS function and admin approvement for this program should be always verified to perform its tasks properly without any issues.

1.2- Local Network:

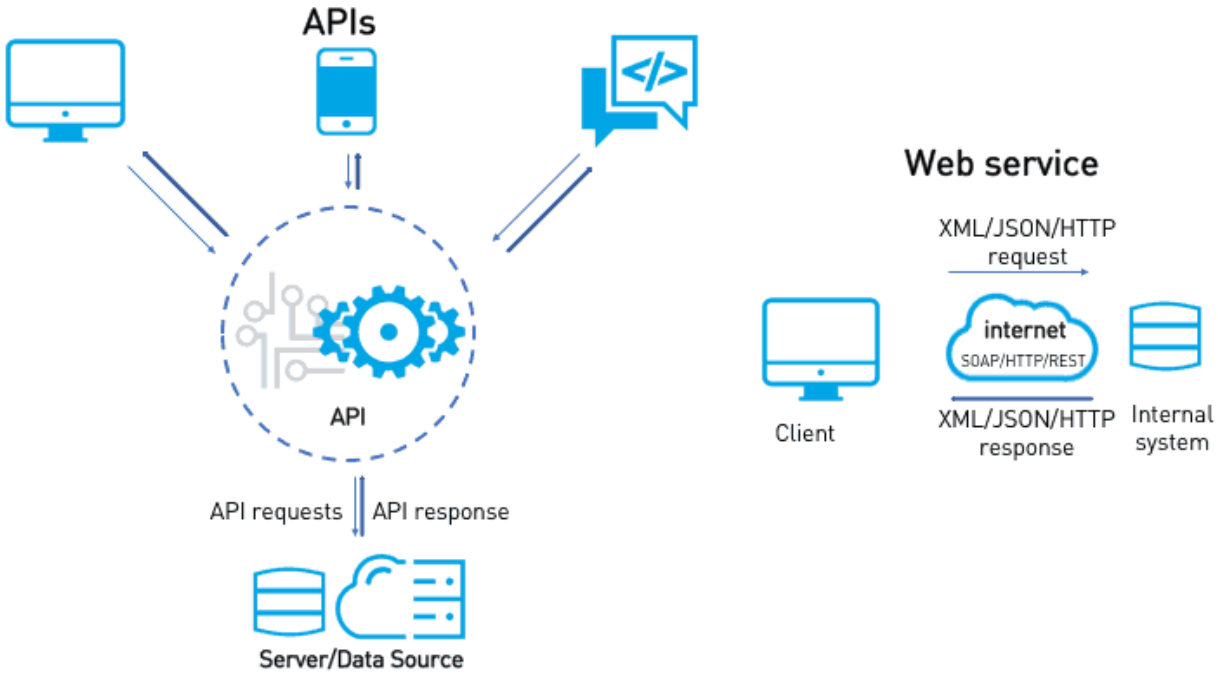
A local-area network (LAN) is a computer network that spans a relatively small area. Most often, a LAN is confined to a single room, building or group of buildings, however, one LAN can be connected to other LANs over any distance via telephone lines and radio waves. A system of LANs connected in this way is called a wide-area network (WAN). The difference between a LAN and WAN is that the wide-area network spans a relatively large geographical area. Typically, a WAN consists of two or more local-area networks (LANs) and are often connected through public networks Most LANs connect workstations and personal computers. Each node (individual computer) in a LAN has its own CPU with which it executes programs, but it also is able to access data and devices anywhere on the LAN. This means that many users can share expensive devices, such as laser printers, as well as data. Users can also use the LAN to communicate with each other, by sending email or engaging in chat sessions. LANs are capable of transmitting data at very fast rates, much faster than data can be transmitted over a telephone line; but the distances are limited and there is also a limit on the number of computers that can be attached to a single LAN.



LANs are typically confined within a small area usually one building, but that's not a firm requirement. That area might be your home or small business, and it could contain just a few devices. It might also be a much larger area, like an entire office building that contains hundreds or thousands of devices. But regardless of size, the single defining characteristic of a LAN is that it connects devices that are in a single, limited area. The advantages of using a LAN are the same advantages as having any devices networked together. Those devices can share a single internet connection, share files with one another, print to shared printers, and so on. On bigger LANs, you'll also find dedicated servers that host services like global user directories, email, and access to other shared company resources.

1.3- Web API:

An Application Programming Interface (or API) is a way for two web pages and/or pieces of software to communicate with each other. An API works as a middleman, taking the request from one piece of software, and then replying with the appropriate response from the other. One example of an API you may be familiar with is the Create publish and share function. Using an API, we send a request to allow us to post on your social media account such as Twitter. The Twitter API then responds by posting a status update on your social media account. The Create API opens up the possibility to streamline a number of tasks in the day-to-day running of your business by allowing for the easy creation of Apps. For instance, an App could be set up using the API to automatically transfer your shop orders into your accounting software.



As a Developer, you have the option of creating Apps for yourself or client and making these publicly available to Create customers as well. To learn more, and to read our Developer Documentation, see our API Information for Developers guide. As a customer of Create, you can take advantage of publicly created Apps built using our API or work with a developer to create your own. Please see our API Information for Customers guide for more information.

A server-side web API is a programmatic interface consisting of one or more publicly exposed endpoints to a defined request response message system, typically expressed in JSON or XML, which is exposed via the web most commonly by means of an HTTP-based web server. Mashups are web applications which combine the use of multiple server-side web APIs. Webhooks are server-side web APIs that take input as a Uniform Resource Identifier (URI) that is designed to be used like a remote named pipe or a type of callback such that the server acts as a client to dereference the provided URI and trigger an event on another server which handles this event thus providing a type of peer-to-peer IPC.

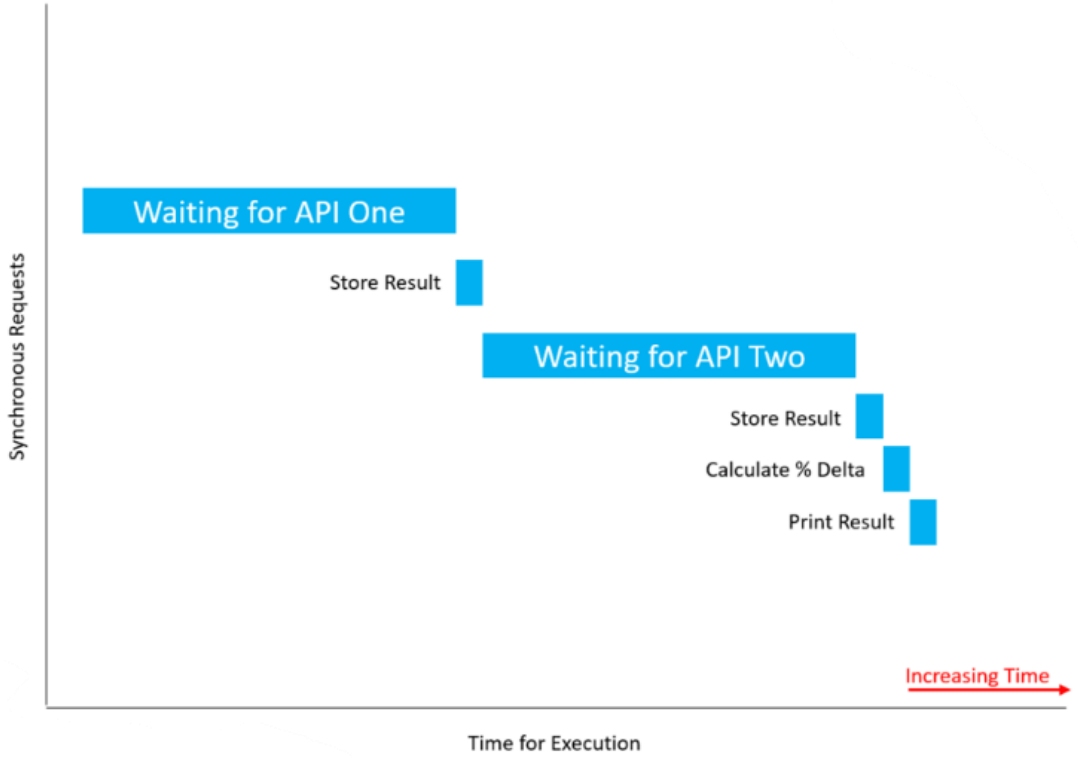
a web service is any piece of software that makes itself available over the Internet and standardizes its communication via XML encoding. A client invokes a web service by sending a request (usually in the form of an XML message), and the service sends back an XML response. Web services invoke communication over a network, with HTTP as the most common means of connectivity between the two systems. For many, web services are synonymous with SOA (Services Oriented Architecture) and primarily rely on standards such as XML-RPC and SOAP (Simple Object Access Protocol). One of the primary criticisms of the web services approach is the degree to which the client and server are coupled to one another; an inherent issue whenever a client is remotely calling a procedure ("RPC") on a distant system.

In contrast, a typical Web API specifies how software components should interact with each other using the web's protocol (HTTP) as the go-between. The client doesn't need to know what procedure to call on the server. Instead, it uses a set of commands (called "verbs") that are built into HTTP and when the command arrives on the other end, it's up to the receiving system to know what to do with it. For example, the HTTP verb that's typically used to retrieve data is "GET". When HTTP is used to abstract systems from one another, the systems are considered to be more loosely coupled (when compared to web services) and therefore the entire system is considered less brittle. Another advantage of web APIs (often referred to as RESTful APIs) is flexibility. The client system (usually called the "consumer") and the serving system (the "provider") are so independent of one another, that they can each use different languages (Java, Python, Ruby, etc.) for their part of the overall implementation. Additionally, the data payloads can be of multiple types such as JSON or XML. RESTful APIs most typically use the web's communication protocol (again, HTTP), but are not limited in the same way a web service is. For example, COAP, an HTTP-like protocol that's common to the Internet of Things, is also considered to be RESTful.

1.4- What is NodeJS

Node is only an environment, or runtime, within which to run normal JavaScript (with minor differences) outside of the browser. We can use it to build desktop applications (with frameworks like Electron), write web or app servers, and more.

Suppose we are making a database call to retrieve properties about a user. That call is going to take time, and if the request is “blocking”, then that means it will block the execution of our program until the call is complete. In this case, we made a “synchronous” request since it ended up blocking the thread. So, a synchronous operation blocks a process or thread until that operation is complete, leaving the thread in a “wait state”. An asynchronous operation, on the other hand, is non-blocking. It permits execution of the thread to proceed regardless of the time it takes for the operation to complete or the result it completes with, and no part of the thread falls into a wait state at any point. Let’s look at another example of a synchronous call that blocks a thread. Suppose we are building an application that compares the results of two Weather APIs to find their percent difference in temperature. In a blocking manner, we make a call to Weather API One and wait for the result. Once we get a result, we call Weather API Two and wait for its result. Don’t worry at this point if you are not familiar with APIs. We’ll be covering them in an upcoming section. For now, just think of an API as the medium through which two computers may communicate with one another.



1.5- APIs & JSON:

It costs money for companies to develop, maintain, and secure that aforementioned infrastructure, and so it is common for corporations to sell you access to their API. This is done with that is known as an API key, a unique alphanumeric identifier associating you, the developer, with the API. Every time you ask the API to send you data, you pass along your API key. The server can then authenticate you and keep track of how many API calls you are making, and you will be charged appropriately. The API key also permits Rate-Limiting or API Call Throttling (a method of throttling the number of API calls in a certain timeframe as to not overwhelm the server, preventing DOS attacks — Denial of Service). Most companies, however, will provide a free quota, giving you, as an example, 25,000 free API calls a day before charging you.

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40</age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
      {
            "employees" :  [
            {
                  "name" : "James Kirk",
                  "age" : 40,
            },
            {
                  "name" : "Jean-Luc Picard",
                  "age" : 45,
            },
            {
                  "name" : "Wesley Crusher",
                  "age" : 27,
            }
                              ]
      }
}
```

Up to this point, we have established that an API is a method by which two computer programs can communicate with each other. If a server is storing data, such as a website, and your browser makes a request to download the code for that site, that was the API in action. Let us look at a more tangible example, and then we'll look at a more real-world, technical one. Suppose you are eating out at a restaurant for dinner. You are equivalent to the client, sitting at the table, and the chef in the back is equivalent to the server. Since you will never directly talk to the chef, there

is no way for him/her to receive your request (for what order you would like to make) or for him/her to provide you with your meal once you order it. We need someone in the middle. In this case, it's the waiter, analogous to the API. The API provides a medium with which you (the client) may talk to the server (the chef), as well as a set of rules for how that communication should be made (the menu — one meal is allowed two sides, etc.)

Now, how do you actually talk to the API (the waiter)? You might speak English, but the chef might speak Spanish. Is the waiter expected to know both languages to translate? What if a third person comes in who only speaks Mandarin? What then? Well, all clients and servers have to agree to speak a common language, and in computer programming, that language is JSON, pronounced JAY-sun, and it stands for JavaScript Object Notation. At this point, we don't quite know what JSON looks like. It's not a computer programming language, it's just, well, a language, like English or Spanish, that everyone (everyone being computers) understands on a guaranteed basis. It's guaranteed because it's a standard, notably RFC 8259, the JavaScript Object Notation (JSON) Data Interchange Format by the Internet Engineering Task Force (IETF). Even without formal knowledge of what JSON actually is and what it looks like (we'll see in an upcoming Document in this series), we can go ahead introduce a technical example operating on the Internet today that employs APIs and JSON. APIs and JSON are not just something you can choose to use, it's not equivalent to one out of a thousand JavaScript frameworks you can pick to do the same thing. It is THE standard for data exchange on the web.

Suppose you are building a travel website that compares prices for aircraft, rental car, and hotel ticket prices. Let us walk through, step-by-step, on a high level, how we would build such an application. Of course, we need our User Interface, the front-end, but that is out of scope for this Document. We want to provide our users with the lowest price booking method. Well, that means we need to somehow attain all possible booking prices, and then compare all of the elements in that set (perhaps we store them in an array) to find the smallest element (known as the infimum in mathematics.) How will we get this data? Well, suppose all of the booking sites have a database full of prices. Those sites will provide an API, which exposes the data in those databases for use by you. You will call each API for each site to attain all possible booking prices, store them in your own array, find the lowest or minimum element of that array, and then provide the price and booking link to your user. We'll ask the API to query its database for the price in JSON, and it will respond with said price in JSON to us. We can then use, or parse, that accordingly. We have to parse it because APIs will return JSON as a string, not the actual JavaScript data type of JSON. This might not make sense now, and that's okay. We'll be covering it more in a future Document.

1.6- HTTP Requests:

HTTP is the HyperText Transfer Protocol. It is the underlying protocol that determines how messages are formatted as they are transmitted and received across the web. Let's think about what happens when, for example, you attempt to load the home page of Smashing Magazine in your web browser. You type the website URL (Uniform Resource Locator) in the URL bar, where the DNS server (Domain Name Server, out of scope for this Document) resolves the URL into the appropriate IP Address. The browser makes a request, called a GET Request, to the Web Server to, well, GET the underlying HTML behind the site. The Web Server will respond with a message such as "OK", and then will go ahead and send the HTML down to the browser where it will be parsed and rendered accordingly.

There are a few things to note here. First, the GET Request, and then the "OK" response. Suppose you have a specific database, and you want to write an API to expose that database to your users. Suppose the database contains books the user wants to read (as it will in a future Document in this series). Then there are four fundamental operations your user may want to perform on this database, that is, CREATE a record, READ a record, UPDATE a record, or DELETE a record, known collectively as CRUD operations. Let's look at the Read operation for a moment. Without incorrectly assimilating or conflating the notion of a web server and a database, that Read operation is very similar to your web browser attempting to get the site from the server, just as to read a record is to get the record from the database. This is known as an HTTP Request. You are making a request to some server somewhere to get some data, and, as such, the request is appropriately named "GET", capitalization being a standard way to denote such requests.

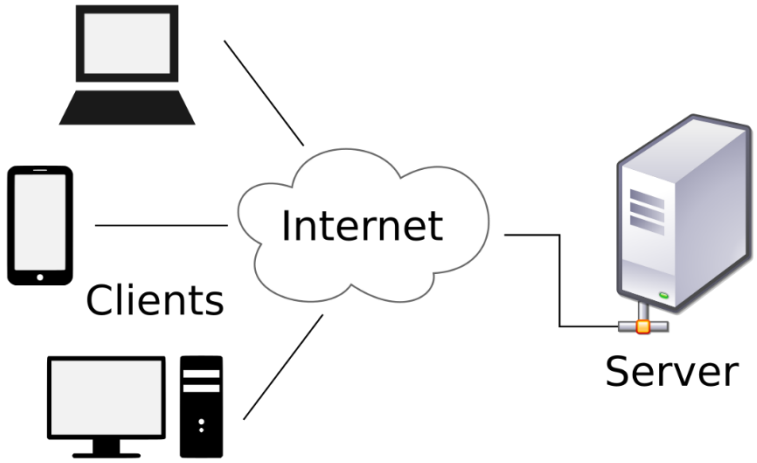
What about the Create portion of CRUD? Well, when talking about HTTP Requests, that is known as a POST request. Just as you might post a message on a social media platform, you might also post a new record to a database. CRUD's Update allows us to use either a PUT or PATCH Request in order to update a resource. HTTP's PUT will either create a new record or will update/replace the old one. An API generally works by making HTTP requests to specific routes in a URL. Suppose we are making an API to talk to a DB containing a user's booklist. Then we might be able to view those books at the URL (.../books). A POST requests to (.../books) will create a new book with whatever properties you define (think id, title, ISBN, author, publishing data, etc.) at the (.../books) route. It doesn't matter what the underlying data structure is that stores all the books at (.../books) right now. We just care that the API exposes that endpoint (accessed through the route) to manipulate data. The prior sentence was key: A POST request creates a new book at the ...books/ route. The difference between PUT and POST, then, is that PUT will create a new book (as with POST) if no such book exists, or, it will replace an existing book if the book already exists within that aforementioned data structure.

Suppose each book has the following properties: id, title, ISBN, author, hasRead (Boolean). Then to add a new book, as seen earlier, we would make a POST request to (.../books). If we wanted to completely update or replace a book, we would make a PUT request to (.../books/id) where (id) is the ID of the book we want to replace. While PUT completely replaces an existing book, PATCH updates something having to do with a specific book, perhaps modifying the hasRead Boolean property we defined above — so we'd make a PATCH request to …/books/id sending along the new data. It can be difficult to see the meaning of this right now, for thus far, we've established everything in theory but haven't seen any tangible code that actually makes an HTTP request.

There is one last fundamental CRUD operation and it's called Delete. As you would expect, the name of such an HTTP Request is "DELETE", and it works much the same as PATCH, requiring the book's ID be provided in a route. We have learned thus far, then, that routes are specific URLs to which you make an HTTP Request, and that endpoints are functions the API provides, doing something to the data it exposes. That is, the endpoint is a programming language function located on the other end of the route, and it performs whatever HTTP Request you specified. We also learned that there exist such terms as POST, GET, PUT, PATCH, DELETE, and more (known as HTTP verbs) that actually specify what requests you are making to the API. Like JSON, these HTTP Request Methods are Internet standards as defined by the Internet Engineering Task Force (IETF), most notably, RFC 7231, Section Four: Request Methods, and RFC 5789, Section Two: Patch Method, where RFC is an acronym for Request for Comments.

1.7- Client vs Server

In computing terminology, both “client” and “server” refer to computers that are used for different purposes. A client is a small computer that accesses a server through a network. For example, in an organization, an employee logs in to the client machine to access the files and applications running on a server machine. This two-tier architecture is also known as client-server architecture which mainly focuses on the division of labor in an organization. A server machine is a large-capacity computer that can store a wide variety of files such as application and data files. There are various types of servers, such as; application server, file server, web server, database server, print server, proxy server, game server, standalone server, etc. A client can be classified into fat, thin, and hybrid. A fat client supports both local storage and local processing. A thin client is a less powerful machine with minimum hardware installed. It usually utilizes the resources of a host machine and relies on the server to perform any data processing. The primary job of a thin client is just to graphically display the images provided by an application server. A hybrid client processes locally but relies on the server for data storage.



Some application servers may require users to log in from their client machines in order to access specific applications utilizing the client-server architecture. The client machines can not only access the applications and data files, but they can also use the processor of the server to perform certain tasks without having to add any additional hardware resources to the client machine. The client computer usually contains more end-user software than the server computer. A server usually contains more operating system components. Multiple users can log into a server at the same time. A client machine is simple and inexpensive whereas a server machine is more powerful and expensive. The main difference between a client machine and a server machine is in its performance. The client machines are considered optimal for applications which require speedy start-up times. A server machine is considered optimal for applications where the emphasis is more on performance.

Ch2: FIELD OF WORK:

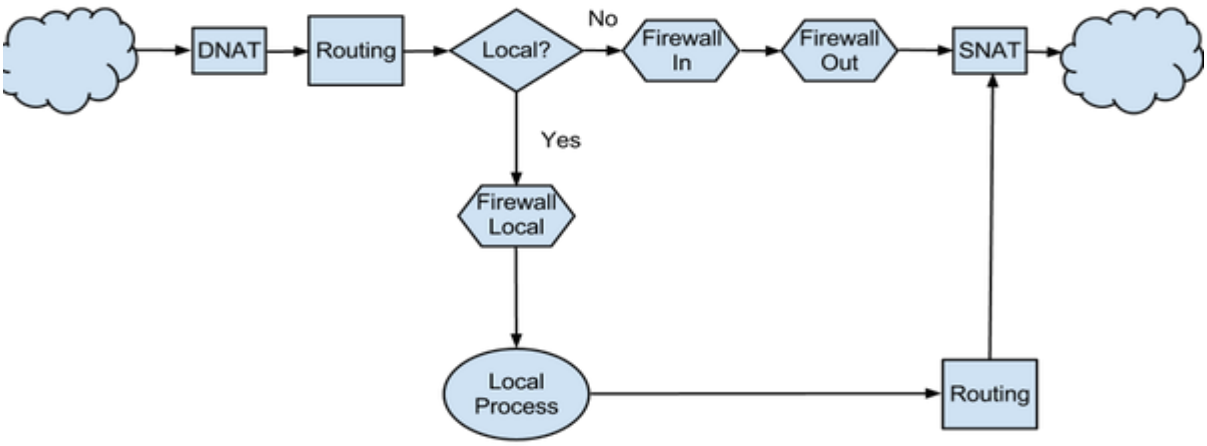
This project discusses how to control network with a simple app which has a friendly user interface to deal with. NCP offers a network control system, voice stream, text messaging and file transfer. In this section we will talk about the field of work of this project and who is the end user and how it affects the flow of the work in companies that uses this system, in other hand we will talk about the benefits of NCP system and how such a system can solve a problem.

2.1- End User:

End user of this app is a company that wants to control all computer from admin mobile easily without a technical IT employee, parents that want to control internet access of their children and schools that wants to control library PCs or laboratories PCs.

2.2- NCP Network Control feature:

With NCP network control module admin can permit or block user connectivity to the internet via a **netsh** command line. This command mainly controls all networking interfaces of the OS via specific switches and with this command we can add role to turn on or of the internet access for the user.

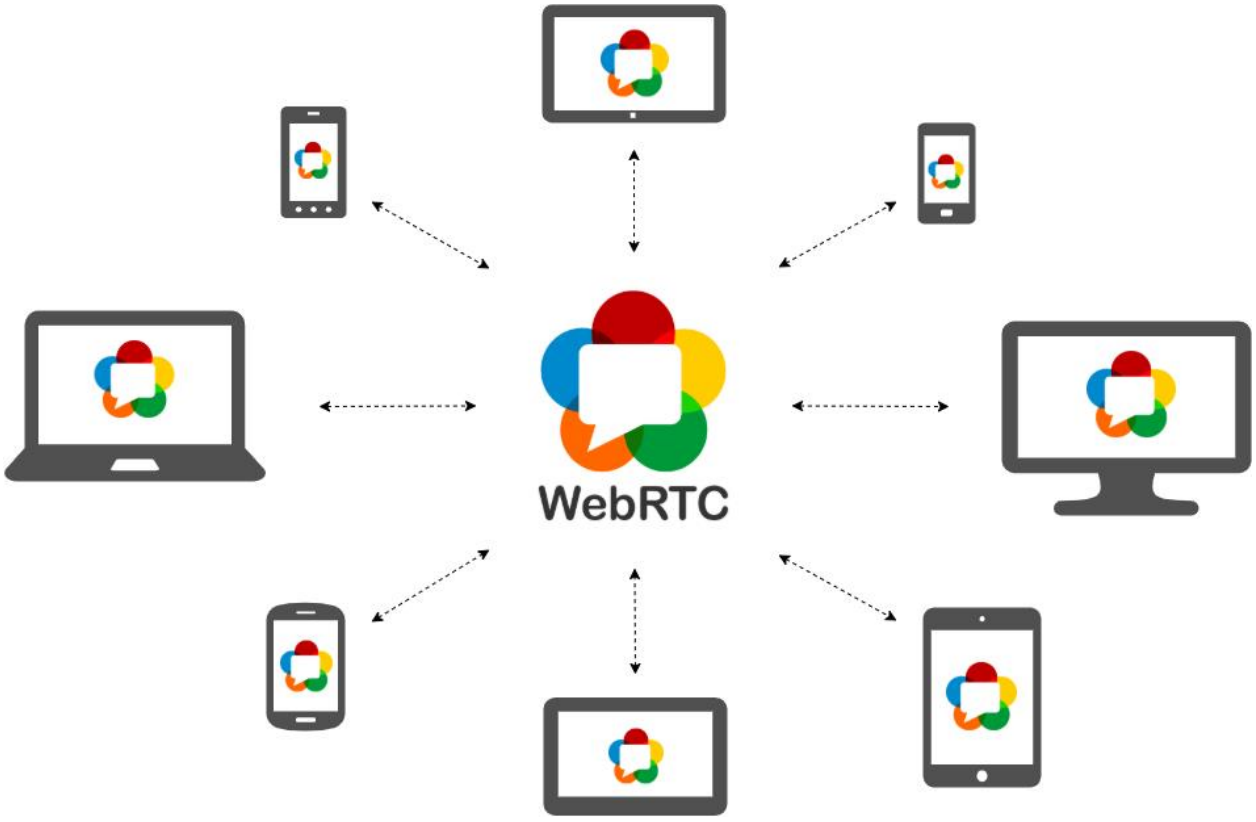


When a role added to firewall to block internet access to the internet the firewall acts as a filer for this app requests to the internet and strictly blocks it from the internet. This feature is valid via two buttons (allow, reject) and a label of current status of this device. The program simply searches for all browsers in the OS and change their roles of connectivity to the internet via firewall of the OS. You can control each device separately to be allowed or prevented from internet access. When you hit allow button NCP searches for all browsers installed in the system and then add an allowed role to the fire wall for out requests.

```
`netsh advfirewall firewall add rule name="${browser.name}" dir=out action=block profile=any program="${browser.path}"`
```

2.3- Voice streaming:

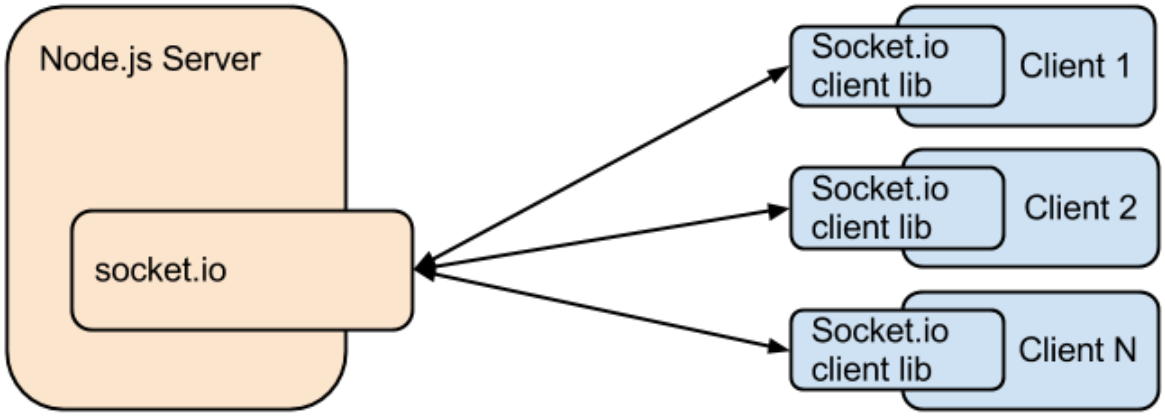
For voice communication we use a LANMIC protocol that depends on a WEBRTC technology. With WebRTC, you can add real-time communication capabilities to your application that works on top of an open standard. It supports video, voice, and generic data to be sent between peers, allowing developers to build powerful voice- and video-communication solutions. The technology is available on all modern browsers as well as on native clients for all major platforms. The technologies behind WebRTC are implemented as an open web standard and available as regular JavaScript APIs in all major browsers. For native clients, like Android and iOS applications, a library is available that provides the same functionality. The WebRTC project is open-source and supported by Apple, Google, Microsoft and Mozilla, amongst others. This page is maintained by the Google WebRTC team.



There are many different use-cases for WebRTC, from basic web apps that uses the camera or microphone, to more advanced video-calling applications and screen sharing. We have gathered a number of code samples to better illustrate how the technology works and what you can use it for. A WebRTC application will usually go through a common application flow. Accessing the media devices, opening peer connections, discovering peers, and start streaming. We recommend that new developers read through our introduction to WebRTC before they start developing.

2.4- Text Messages:

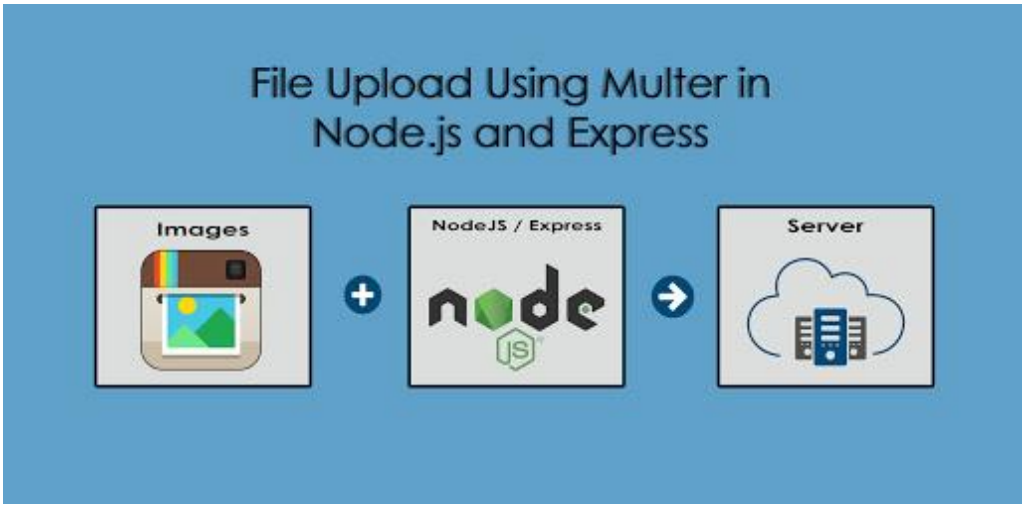
Chat module is based on the socket.IO module to send and receive a real time message. Socket.IO is a JavaScript library for real-time web applications. It enables real-time, bi-directional communication between web clients and servers. It has two parts: a client-side library that runs in the browser, and a server-side library for Node.js. Both components have a nearly identical API. Like Node.js, it is event-driven. Socket.IO primarily uses the WebSocket protocol with polling as a fallback option, while providing the same interface. Although it can be used as simply a wrapper for WebSocket, it provides many more features, including broadcasting to multiple sockets, storing data associated with each client, and asynchronous I/O.



Socket.IO provides the ability to implement real-time analytics, binary streaming, instant messaging, and document collaboration. Notable users include Microsoft Office, Yammer, and Zendesk. Socket.IO handles the connection transparently. It will automatically upgrade to WebSocket if possible. This requires the programmer to only have Socket.IO knowledge. Socket.IO is not a WebSocket library with fallback options to other real-time protocols. It is a custom real-time transport protocol implementation on top of other real-time protocols. A Socket.IO implementing server cannot connect to a non-Socket.IO WebSocket client. A Socket.IO implementing client cannot talk to a non-Socket.IO WebSocket or Long Polling Comet server. Socket.IO requires using the Socket.IO libraries on both client and server side.

2.5- **File upload**:

File uploading is based on Multer module that handles files uploaded in a multipart form. Multer is Express middleware. Middleware is a piece of software that connects different applications or software components. In Express, middleware processes and transforms incoming requests to the server. In our case, Multer acts as a helper when uploading files. Multer adds a body object and a file or files object to the request object. The body object contains the values of the text fields of the form, the file or files object contains the files uploaded via the form. Multer accepts an options object, the most basic of which is the **dest** property, which tells Multer where to upload the files. In case you omit the options object, the files will be kept in memory and never written to disk.



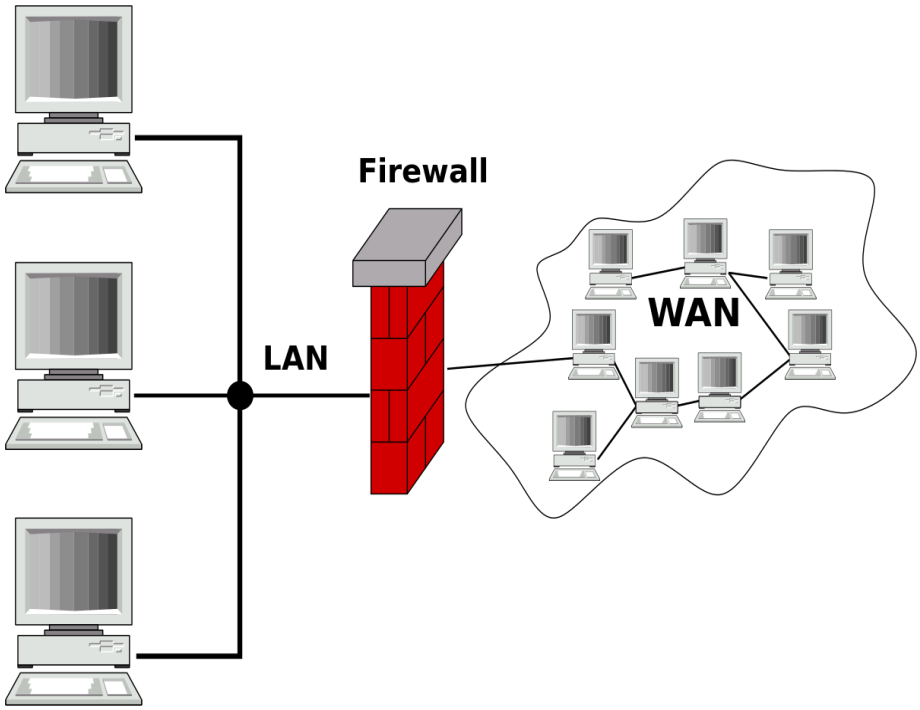
By default, Multer will rename the files so as to avoid naming conflicts. The renaming function can be customized according to your needs. There are two options available, destination and filename. They are both functions that determine where the file should be stored. destination is used to determine within which folder the uploaded files should be stored. This can also be given as a string (e.g. '/tmp/uploads'). If no destination is given, the operating system's default directory for temporary files is used. When encountering an error, Multer will delegate the error to Express. You can display a nice error page using the standard express way. If you want to catch errors specifically from Multer, you can call the middleware function by yourself. Also, if you want to catch only the Multer errors, you can use the MulterError class that is attached to the Multer object itself.

Ch3: MAIN WORK:

In this Section we will talk about the actual work with details but first we will talk about some notations that will be used in this section later lice firewall, IP address, ping, MVC, and ionic framework.

3.1- Firewall:

Broadly speaking, a computer firewall is a software program that prevents unauthorized access to or from a private network. Firewalls are tools that can be used to enhance the security of computers connected to a network, such as LAN or the Internet. They are an integral part of a comprehensive security framework for your network. A firewall absolutely isolates your computer from the Internet using a "wall of code" that inspects each individual "packet" of data as it arrives at either side of the firewall — inbound to or outbound from your computer — to determine whether it should be allowed to pass or be blocked. Firewalls have the ability to further enhance security by enabling granular control over what types of system functions and processes have access to networking resources. These firewalls can use various types of signatures and host conditions to allow or deny traffic. Although they sound complex, firewalls are relatively easy to install, setup and operate.



Most people think that a firewall is a of device that is installed on the network, and it controls the traffic that passes through the network segment. However, you can have a host-based firewalls. This can be executed on the systems themselves, such as with ICF (Internet Connection Firewall). Basically, the work of both the firewalls is the same: to stop intrusion and provide a strong method of access control policy. In simple definition, firewalls are nothing but a system that safeguards your computer; access control policy enforcement points.

It is important to understand why we need a firewall and how it helps us in the world of secure computing. We need to understand the goals of information security because it helps us to understand how a firewall may address those needs. In the age of high-speed Internet Access, you electronically connect your computer to a broad network over which, unless you have installed a personal firewall, you have limited control and from which you have limited protection. Until recently, unless you worked for an organization that provided high-speed internet access.

Like anything, the high-speed connection has its own drawbacks. Ironically, the very feature that makes a high-speed connection attractive is also the reason that makes it vulnerable. In a way, connecting to the internet via high-speed connection is like leaving the front door of your house open and unlocked. This is because high-speed Internet connections have the following features:

- A constant IP - Make it easy for an intruder who has discovered your computer on the internet to find you again and again.
- High-Speed Access - Means that the intruder can work much faster when trying to break into your computer.
- Always active connection - means that your computer is vulnerable every time when it is connected to the internet.

A Personal firewall is important when

- You surf the internet at home using an 'always on' broadband connection
- You connect to the internet via a public WIFI network in a park, cafe or airport
- You run a home network which needs to be kept isolated from the internet
- You wish to be kept informed when any program on your computer attempts to connect to the internet
- Most Personal Firewalls are highly configurable so you can easily create security policies to suit your individual needs

3.2- IP Address:

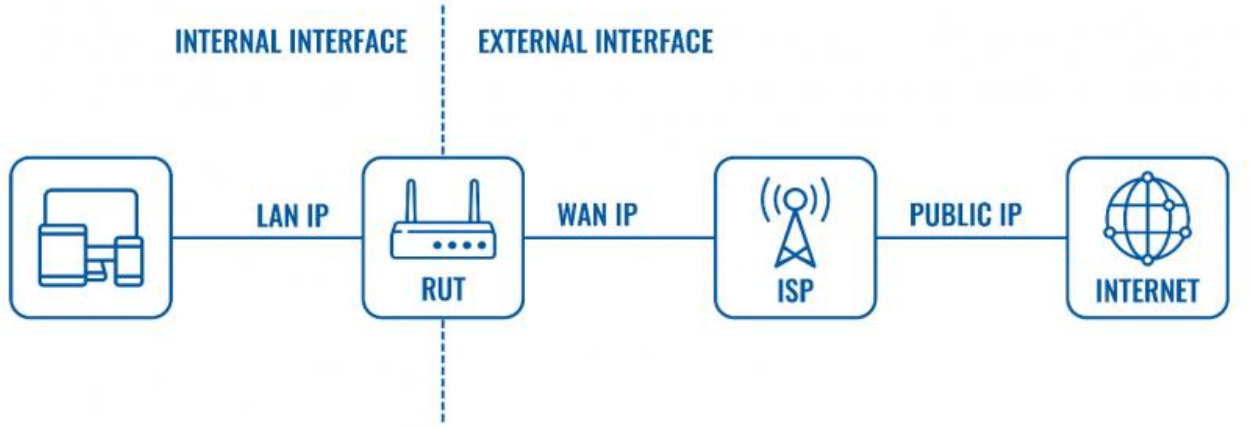
IP (Internet Protocol) Address is an address of your network hardware. It helps in connecting your computer to other devices on your network and all over the world. An IP Address is made up of numbers or characters.

An example of an IP address would be: 506.457.14.512

All devices that are connected to an internet connection have a unique IP address which means there's a need of billions of IP addresses. This requirement is fulfilled by the new IP version IPv6. There are two IP versions: IPv4 and IPv6. IPv4 is the older version which has a space of over 4 billion IP addresses. However, the new IPv6 version can provide up to trillions of IP addresses to fulfill the need of all internet users and devices. The IPv4 version used to configure IP addresses in numerical value (numbers) which may conflict with other IP addresses. That's why IPv6 adopted the hexadecimal method to provide unique IP addresses to billions of users in the world.

Example of an IPv6 IP address would be: 4ggr:1925:5656:7:600:t4tt:tc54:98vt

There are a few types of IP addresses like private IP addresses, public IP addresses, static IP addresses and dynamic IP addresses. Let's talk about these different types of IP addresses one by one:



Private IP Address

A private IP address is the address of your device connected on the home or business network. If you have a few different devices connected to one ISP (Internet Service Provider), then all your devices will have a unique private IP address. This IP address cannot be accessed from devices outside your home or business network.

For example: 192.168.1.1

You can find out the private IP address of your device using a few techniques. If you are a Windows user, then simply go to the command prompt and enter the command ipconfig. If you're a mac user, then you need to enter the following command ifconfig in your Terminal app

If you are using the internet on a mobile phone, then you can go to your WIFI settings to find out the IP address. iOS users can find the IP address by clicking on the 'i' button next to the network they are connected to. Android users can click on the network name in their WIFI settings, and it will show the IP address.

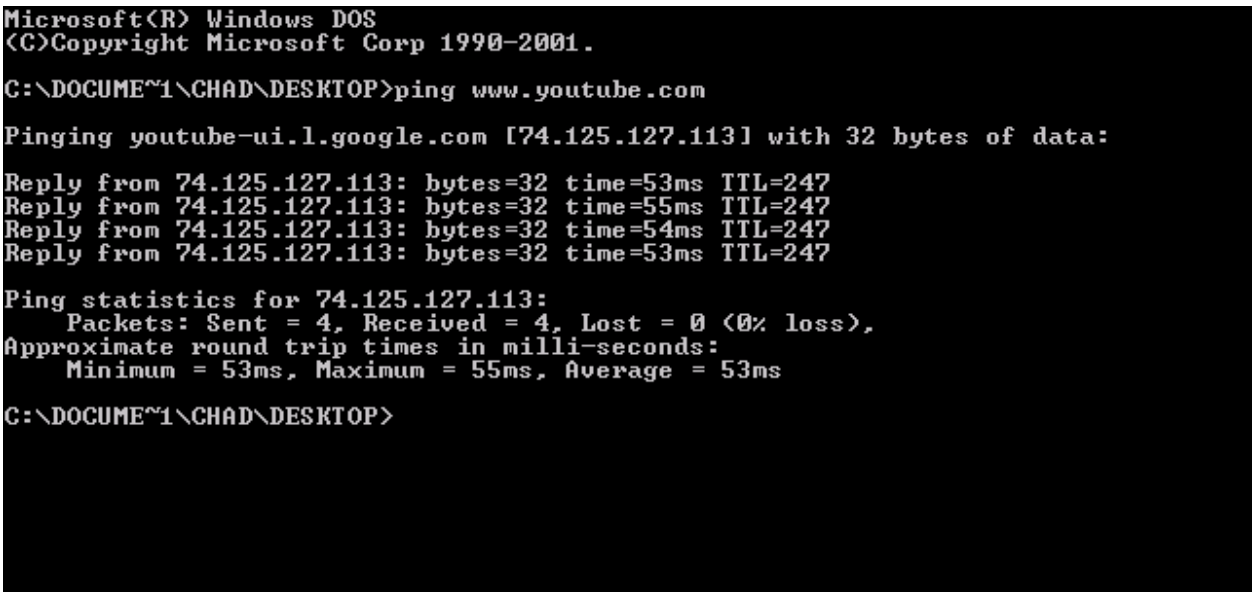
Public IP Address

Your public IP address is the main IP address to which your home or business network is connected. This IP address connects you to the world, and it's unique for all users. To find out your public IP address, simply go to SupportAlly site in your browser, and it will display the public IP, and other browser information.

All private and public IP addresses can be either static or dynamic. IP addresses that you configure manually and fix them to the network of your device are called static IP addresses. Static IP addresses cannot change automatically. The dynamic IP address configures automatically and assign an IP to your network when you set up the router with internet. This distribution of IP addresses is managed by Dynamic Host Configuration Protocol (DHCP). DHCP can be your internet router that assigns an IP address to your network in your home or business environment.

3.3- Ping:

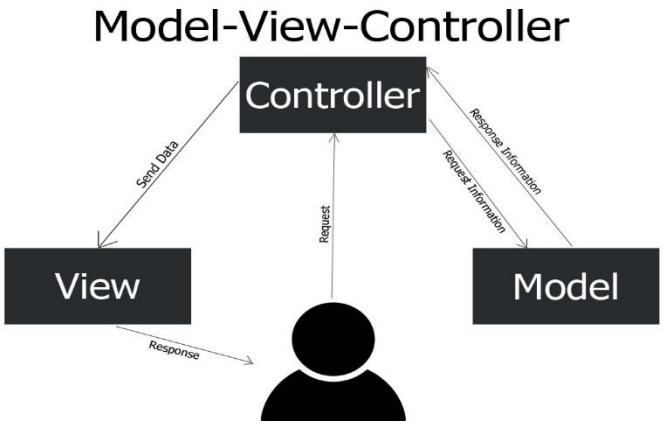
Ping is a computer network administration software utility used to test the reachability of a host on an Internet Protocol (IP) network. It is available for virtually all operating systems that have networking capability, including most embedded network administration software. Ping measures the round-trip time for messages sent from the originating host to a destination computer that are echoed back to the source. The name comes from active sonar terminology that sends a pulse of sound and listens for the echo to detect objects under water.



Ping operates by sending Internet Control Message Protocol (ICMP) echo request packets to the target host and waiting for an ICMP echo reply. The program reports errors, packet loss, and a statistical summary of the results, typically including the minimum, maximum, the mean round-trip times, and standard deviation of the mean. The command-line options of the ping utility and its output vary between the numerous implementations. Options may include the size of the payload, count of tests, limits for the number of network hops (TTL) that probes traverse, interval between the requests and time to wait for a response. Many systems provide a companion utility ping6, for testing on Internet Protocol version 6 (IPv6) networks, which implement ICMPv6.

3.4- MVC:

Model–view–controller (usually known as MVC) is a software design pattern commonly used for developing user interfaces which divides the related program logic into three interconnected elements. This is done to separate internal representations of information from the way's information is presented to and accepted from the user. This kind of pattern is used for designing the layout of the page.



Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

3.5- IONIC framework:

Ionic is a complete open-source SDK for hybrid mobile app development created by Max Lynch, Ben Sperry, and Adam Bradley of Drifty Co. in 2013. The original version was released in 2013 and built on top of AngularJS and Apache Cordova. However, the latest release was re-built as a set of Web Components, allowing the user to choose any user interface framework, such as Angular, React or Vue.js. It also allows the use of Ionic components with no user interface framework at all. Ionic provides tools and services for developing hybrid mobile, desktop, and Progressive Web Apps based on modern web development technologies and practices, using Web technologies like CSS, HTML5, and Sass. In particular, mobile apps can be built with these Web technologies and then distributed through native app stores to be installed on devices by utilizing Cordova or Capacitor.

Ionic uses Cordova and, more recently, Capacitor plugins to gain access to host operating systems features such as Camera, GPS, Flashlight, etc. Users can build their apps, and they can then be customized for Android, iOS, Windows, Desktop (with Electron), or modern browsers. Ionic allows app building and deployment by wrapping around the build tool Cordova or Capacitor with a simplified 'ionic' command line tool.

Using Web Components, Ionic provides custom components and methods for interacting with them. One such component, virtual scroll, allows users to scroll through a list of thousands of items without any performance hits. Another component, tabs, creates a tabbed interface with support for native-style navigation and history state management. Besides the SDK, Ionic also provides services that developers can use to enable features, such as code deploys, automated builds. Ionic also provides its own IDE known as Ionic Studio.

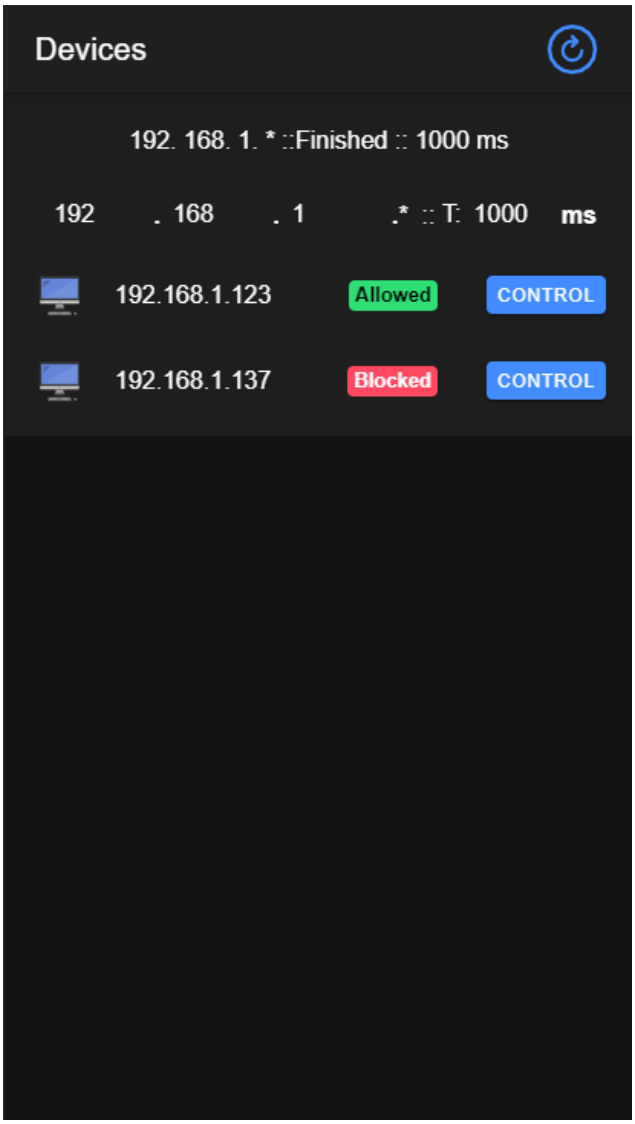
3.6- Network controller APK:

Network controller app is based on ionic framework and it is a cross platform app that can either work on web, android or IOS. This version was built and compiled to run on android only so we will discuss each component of this app code.

The application is consisted of main component named IPS component which contain all other components and is the responsible for getting live Ips of the connected devices to the system, Ip component which contains the control components for each IP device, messenger components which handle messages and chatting page and voice chatting components which is responsible for voice streaming from mobile to the client device. Now we will discuss each function of the code in details

IPS COMPONENT

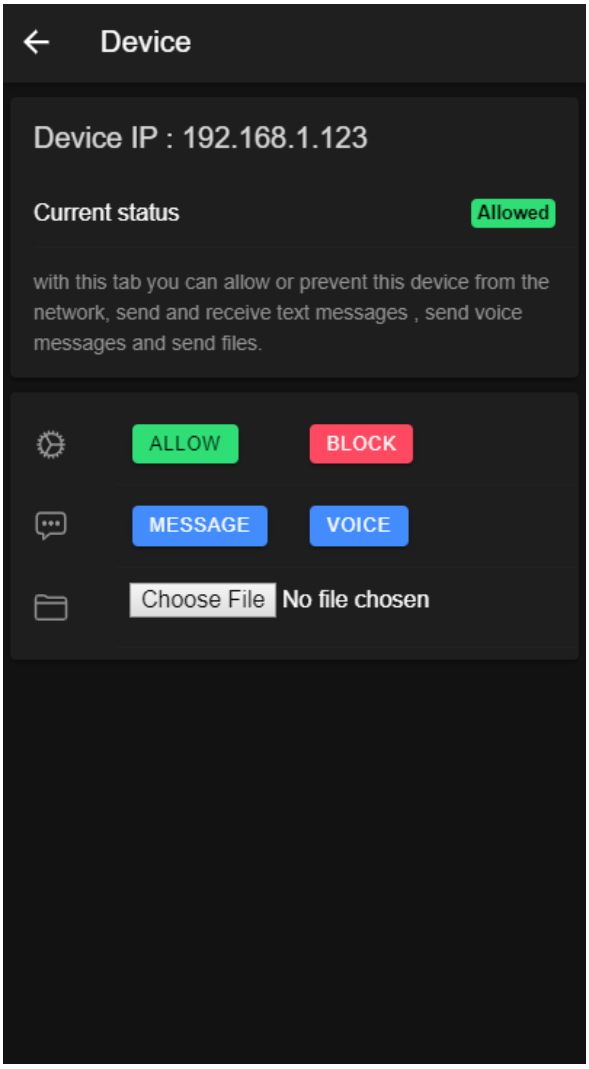
In this component valid Ips are shown and their internet status. First you should set your network Getway then you can get live Ips of connected devices in this example the default Getway is 192.168.1.1 and have a request latency 1000ms which is the time for request to time out.



To get live Ips the app simply sends an http request to all connected devices of the network on port 5050/check and resolve the response If there is a response and reject error if there is no one as an indication of that this device isn't connected or IP is unlocated

IP COMPONENT:

Is the responsible for controlling device it contains a status section to get current data like ip and status of the internet access in this ip and a control section which contain all controllers like internet controller, message chatting controller, voice stream controller, upload file controller.



ALLOW BUTTON:

This button sends a request to the device to allow internet connection and get the current status response to check that the request was processed successfully.

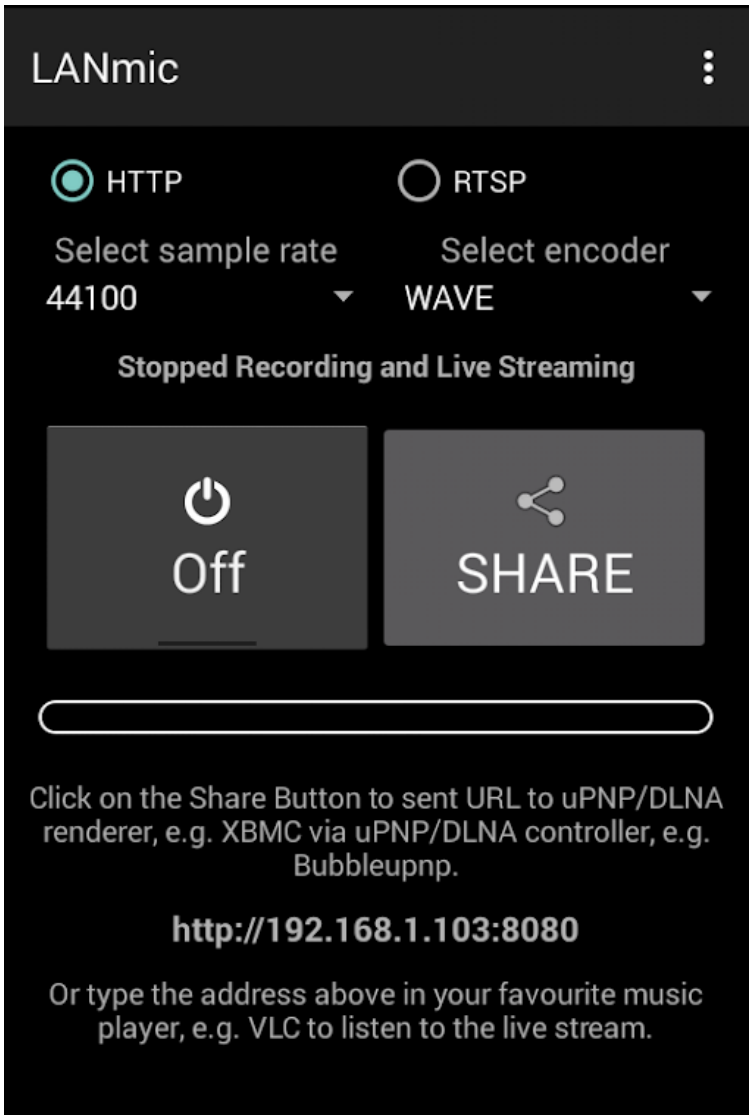
The http request shown below shows the request URI and its directory.

```
public sendRequest(url: string, action: string): Observable<any> {
    return this.http.get(`${url}/${action}`);
}
```

```
sendAction(action: string) {
    this.control.sendRequest(this.ip.url, action).subscribe(
        (response) => {
            if (response.status === "Allowed") this.status = true;
            else this.status = false;
            console.log(response);
        },
        (err) => {
            console.error(err);
        }
    );
```

VOICE STREAM BUTTON:

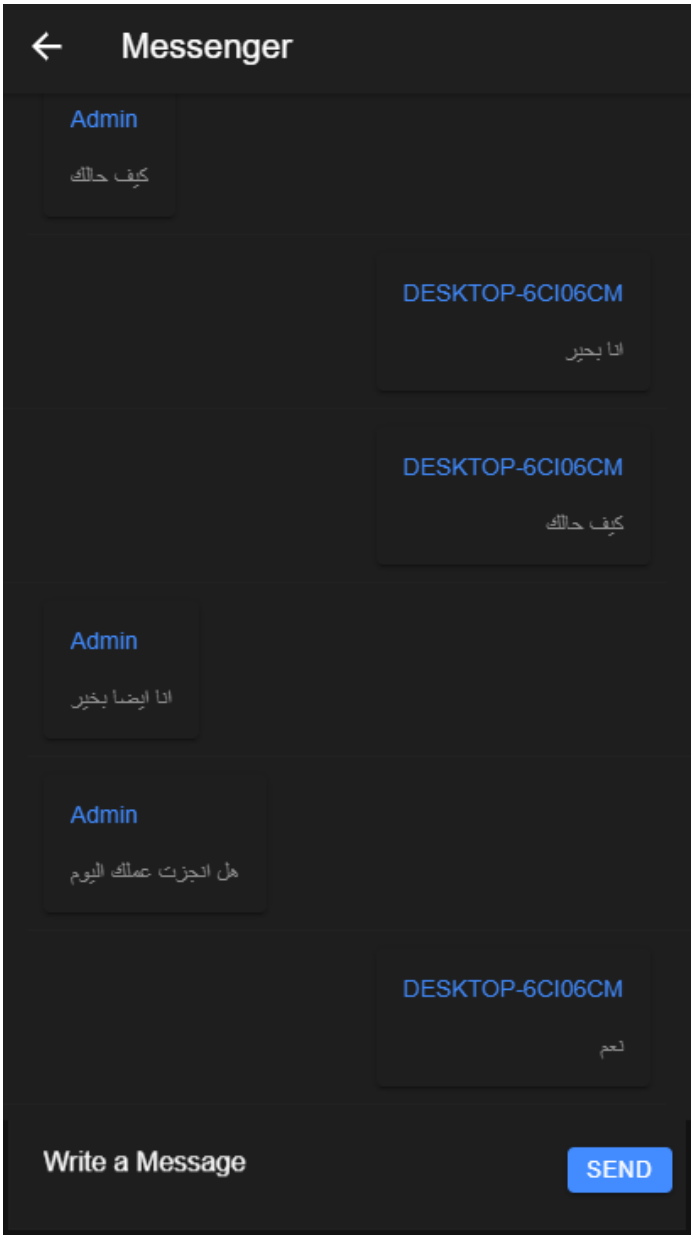
This button opens an external app called LANMIC which starts a voice stream to the selected client. When you click on it will automatically send a request to client to open a new tab listening to this app in the mobile that streams voice. For this purpose, we have a module to call external app in our code




```
async openApp() {
    var ret = await App.canOpenUrl({ url: "com.portable.lanmic" });
    var retx = await App.openUrl({ url: "com.portable.lanmic" });
    this.control.sendRequest(this.ip.url, "voice").subscribe(
        (response) => {
            console.log(response);
        },
        (err) => {
            console.error(err);
        }
    );
    console.log("Open url response: ", ret);
}
```

MESSAGE BUTTON:

Message button routs app to a new chatting page to send and receive real-time messages to and from the client. For this real time application, we used socket.io module that discussed in last chapter.

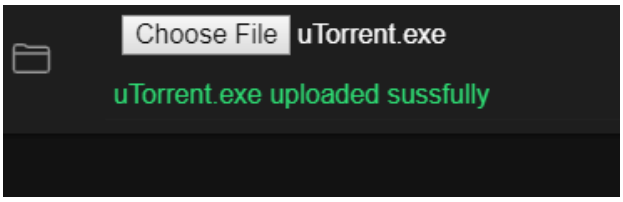


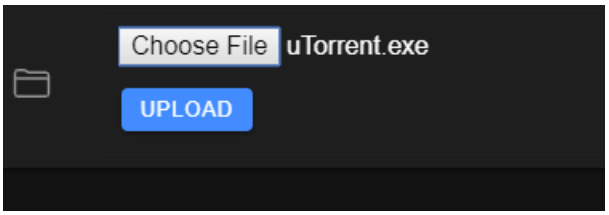

```
setSocket(ip: IP) {
    this.socket = io(ip.url);
}
getSocket() {
    return this.socket;
}
sendMessage(message: string) {
    this.socket.emit("send", message);
}
```

```
sendMessage() {
    if (this.message) {
        console.log(this.message);
        this.chat.sendMessage(this.message);
        this.messages.push({ sender: "Admin", message: this.message });
        this.message = "";
    }
    setTimeout(this.chat.getLastElement, 500);
}
```

UPLOAD FILE BUTTON:

This button simply opens a file browser to get file from mobile and send it to the client. When you select a file, its name appears under the select file button and an upload button appear to upload file. After uploading the file, you get a response of the process under the selected file button and upload button disappears.






```
upload() {
    const fd = new FormData();
    fd.append("file", this.file[0], this.file[0].name);
    this.control.postFiles(this.ip.url, fd).subscribe(
        (user) => {
            this.isUploaded = this.file[0].name + " uploaded sussfully";
            this.file = null;
        },
        (err) => {
            console.error(err);
        }
    );
}
```

3.7- NCP app:

This is the server-side application that control every thing in the device and all features of the system, this app is based on express server running on NodeJS http server and built with MVC architecture. It contains 3 models, 3 routers and 3 controllers for (chat, internet-access, file) these controllers control every process of the system. In this section we will talk about each part of this app in details.

INTERNET-ACCESS:

This controller controls the internet access via some commands that add or delete role to the firewall to allow or bloc browsers from the internet, first of all there is a model that gets all data about all installed browsers , then a module to get the required status and apply it via a CLI command with netsh tool.

```
exports.controlInternet = (state) => {
    let output = [],
        counter = 0;
    return new Promise((resolve, reject) => {
        detect(function (err, browsers) {
            if (err) {
                reject(err);
                throw err;
            }
            console.log(browsers);
            for (let browser of browsers) {
                cmdRun(state ? deleteRule(browser) : addRule(browser))
                    .then((out) => {
                        output.push({ browser: browser.name, out: out, status: "success" });
                        counter++;
                        if (counter >= browsers.length) resolve(output);
                    })
                    .catch((err) => {
                        console.log(err);
                        output.push({ browser: browser.name, out: err, status: "error" });
                        counter++;
                        if (counter >= browsers.length) resolve(output);
                    });
            }
        });
    });
};
```

FILE:

This router gets the file and save it in a folder called uploads in the desktop of the clients. First there is a module to get the path of desktop then it simply set the destination of the file to the current desktop/upload directory.

```
router.post(
    "/file",
    multer({
        storage: multer.diskStorage({
            destination: (req, file, cb) => {
                internetModel
                    .cmdRun("echo %USERPROFILE%\\Desktop\\uploads")
                    .then((res) => {
                        cb(null, res.trim());
                        console.log(res);
                    })
                    .catch((err) => console.log(err));
            },
            filename: (req, file, cb) => {
                cb(null, Date.now() + file.originalname);
            },
        }),
    }).single("file"),
    fileController.postFile
);
```

```
exports.readStatus = () => {
    return new Promise((resolve, reject) => {
        fs.readFile(statusPath, (err, data) => {
            if (err) {
                reject(err);
            } else {
                data = JSON.parse(data);
                resolve(data);
            }
        });
    });
};
```

```
exports.changeStatus = (status) => {
    return new Promise((resolve, reject) => {
        fs.writeFile(statusPath, JSON.stringify(status), (err) => {
            if (err) {
                reject(err);
            } else {
                resolve();
            }
        });
    });
};
```

CHAT:

Chat controller have two routers one for client to be able to send and receive messages and one for admin to be able to send and receive messages. It works on socket.io module to provide a real-time messaging between two peers.

```
exports.postChatMessages = (req, res, next) => {
    if (req.body.message) {
        internetModel
            .getHostName()
            .then((name) => {
                let message = {
                    message: req.body.message,
                    sender: name,
                };
                require("../server").clientSocket.emit("newMessage", message);
                chatModel.postChatMessage(message).then(() => {
                    res.redirect("/chat");
                });
            })
            .catch((err) => console.log(err));
    } else {
        res.redirect("/chat");
    }
};
```

```
exports.postAdminChatMessages = (req, res, next) => {
    if (req.body.message) {
        chatModel
            .postChatMessage({
                message: req.body.message,
                from: "Admin",
            })
            .then(() => {
                res.redirect("/chat/admin");
            })
            .catch((err) => console.log(err));
    } else {
        res.redirect("/chat/admin");
    }
};
```

```
exports.getAdminChatMessages = (req, res, next) => {
    chatModel
        .getChatMessages()
        .then((messages) => {
            res.json(messages);
        })
        .catch((err) => console.log(err));
};
```

Ch4: FUTURE WORK:

We intend to add some features to this app like:

- Voice chatting between two sides.
  - To make the admin listen to the client as well.

- Remote desktop view and control.
  - To make the admin able to vie desktop of the client and control it from mobile with full access.

- Firewall issue resolve for user installed firewall like Mcafee.
  - There is an issue with installed firewalls that they fully control the fire wall so app can't do its job correctly with internet access control.

- Windows home support without any need to enable gpedit.msc.
  - This app currently depends on the administrator tools supported with OS but in-home these tools need to be installed by the user in order to make the app work correctly.

- File collecting from the client to admin.
  - We need to provide a feature to admin to be able to get files from client as well as it sends it to get the full duplex file sharing technique.

- WEBRTC full communication protocols for video calls.
  - In order to get live video with the admin if there is a problem and need to be solved or there is an online meeting for employees.

- Screen recorder.
  - In order to record the client screen to be able to review what he really do with the work time.

- Specific links block.
  - It will be more sense to block specific URLs in network access not all the internet to provide connectivity to company site at least.

REFRENCES:

- https://www.computerhope.com/netsh.htm
- https://docs.microsoft.com/en-us/previous-versions/tn-archive/bb490939(v=technet.10)
- https://doi.org/10.1145%2F253769.253802
- http://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf
- http://www.cs.unm.edu/~treport/tr/02-12/firewall.pdf
- https://techcrunch.com/2014/03/10/drifty-makers-of-the-ionic-mobile-framework-raise-1-million/
- https://blog.ionicframework.com/introducing-ionic-4-ionic-for-everyone/
- https://venturebeat.com/2012/01/24/why-walmart-is-using-node-js/
- https://www.cnet.com/news/netscape-opens-intranet-attack/
- http://www.infoworld.com/article/2855057/application-development/why-iojs-decided-to-fork-nodejs.html
- https://tools.ietf.org/html/rfc760
- https://tools.ietf.org/html/rfc1883
- https://tools.ietf.org/html/rfc8200
- https://web.archive.org/web/20090323032904/https://www.artima.com/articles/dci_vision.html
- http://blog.iandavis.com/2008/12/what-are-the-benefits-of-mvc/