

Contents

Introduction to the Internet.....	2
How the Internet Works: An Overview	2
Basic Concepts and Terminology	3
Understanding IP Addresses and Domain Names	5
Introduction to HTTP and HTTPS	5
What is HTTP?	5
What is in an HTTP request?	6
What is an HTTP method?	6
What are HTTP request headers?	6
What is in an HTTP response?	6
What's an HTTP status code?.....	7
What are HTTP response headers?.....	7
What is in an HTTP response body?	7
What is a Domain Name?	8
Structure of domain names	8
Who owns a domain name?	8
What is a URL?	9
Basics: anatomy of a URL	9
Scheme.....	9
Authority	10
Path to resource.....	10
Parameters	10
Anchor.....	10
How to use URLs	11
What is DNS?.....	12
How does DNS work?.....	12
What are the steps in a DNS lookup?	13
Resources	14

Introduction to the Internet

Before we learn what the Internet is, we need to understand what a Network is. A network is a group of computers or other devices which are connected to each other. For example, you at your home might have a network of computers and devices. Your friend living next door might have a similar network of devices. Their neighbor might have a similar network of devices. All these networks when connected together form the internet.

The internet was developed in the late 1960s by the United States Department of Defense as a means of creating a decentralized communication network that could withstand a nuclear attack. Over the years, it has evolved into a complex and sophisticated network that spans the globe.

Today, the internet is an essential part of modern life, used by billions of people around the world to access information, communicate with friends and family, conduct business, and much more. As a developer, it is essential to have a solid understanding of how the internet works and the various technologies and protocols that underpin it.

How the Internet Works: An Overview

At a high level, the internet works by connecting devices and computer systems together using a set of standardized protocols. These protocols define how information is exchanged between devices and ensure that data is transmitted reliably and securely.

The core of the internet is a global network of interconnected routers, which are responsible for directing traffic between different devices and systems. When you send data over the internet, it is broken up into small packets that are sent from your device to a router. The router examines the packet and forwards it to the next router in the path towards its destination. This process continues until the packet reaches its final destination.

To ensure that packets are sent and received correctly, the internet uses a variety of protocols, including the Internet Protocol (IP) and the Transmission Control Protocol (TCP). IP is responsible for routing packets to their correct destination, while TCP ensures that packets are transmitted reliably and in the correct order.

In addition to these core protocols, there are a wide range of other technologies and protocols that are used to enable communication and data exchange over the internet, including the Domain Name System (DNS), the Hypertext Transfer Protocol (HTTP), and the Secure Sockets Layer/Transport Layer Security (SSL/TLS) protocol. As a developer, it is important to have a solid understanding of how these different technologies and protocols work together to enable communication and data exchange over the internet.

Basic Concepts and Terminology

- **Packet:** A small unit of data that is transmitted over the internet.
- **Router:** A device that directs packets of data between different networks.
- **IP Address:** A unique identifier assigned to each device on a network, used to route data to the correct destination.
- **Domain Name:** A human-readable name that is used to identify a website, such as google.com.
- **DNS:** The Domain Name System is responsible for translating domain names into IP addresses.
- **HTTP:** The Hypertext Transfer Protocol is used to transfer data between a client (such as a web browser) and a server (such as a website).
- **HTTPS:** An encrypted version of HTTP that is used to provide secure communication between a client and server.
- **SSL/TLS:** The Secure Sockets Layer and Transport Layer Security protocols are used to provide secure communication over the internet.
- **Ports:** Ports are used to identify the application or service running on a device. Each application or service is assigned a unique port number, allowing data to be sent to the correct destination.
- **Sockets:** A socket is a combination of an IP address and a port number, representing a specific endpoint for communication. Sockets are used to establish connections between devices and transfer data between applications.
- **Connections:** A connection is established between two sockets when two devices want to communicate with each other. During the connection establishment process, the devices negotiate various parameters such as the maximum segment size and window size, which determine how data will be transmitted over the connection.
- **Data transfer:** Once a connection is established, data can be transferred between the applications running on each device. Data is typically transmitted in segments, with each segment containing a sequence number and other metadata to ensure reliable delivery.
- **Web page:** A document that can be displayed in a web browser. These are also often called just "pages". Such documents are written in the HTML language (which we look at in more detail later on).
- **Website:** A collection of web pages grouped together into a single resource, with links connecting them together. Often called a "site".
- **Web server:** A computer that hosts a website on the Internet.
- **Web service:** A software that responds to requests over the Internet to perform a function or provide data. A web service is typically backed by a web server, and may provide web pages for users to interact with. Many websites are also web services, though some websites (such as MDN) consist of static content only. Examples of web services would be something that resizes images, provides a weather report, or handles user login.

- **Search engine:** A web service that helps you find other web pages, such as Google, Bing, Yahoo, or DuckDuckGo. Search engines are normally accessed through a web browser (for example, you can perform search engine searches directly in the address bar of Firefox, Chrome, etc.) or through a web page (for example, bing.com or duckduckgo.com).

Understanding IP Addresses and Domain Names

IP addresses and domain names are both important concepts to understand when working with the internet.

An IP address is a unique identifier assigned to each device on a network. It's used to route data to the correct destination, ensuring that information is sent to the intended recipient. IP addresses are typically represented as a series of four numbers separated by periods, such as "192.168.1.1".

Domain names, on the other hand, are human-readable names used to identify websites and other internet resources. They're typically composed of two or more parts, separated by periods. For example, "google.com" is a domain name. Domain names are translated into IP addresses using the Domain Name System (DNS).

DNS is a critical part of the internet infrastructure, responsible for translating domain names into IP addresses. When you enter a domain name into your web browser, your computer sends a DNS query to a DNS server, which returns the corresponding IP address. Your computer then uses that IP address to connect to the website or other resource you've requested.

Introduction to HTTP and HTTPS

HTTP (Hypertext Transfer Protocol) and **HTTPS** (HTTP Secure) are two of the most commonly used protocols in internet-based applications and services.

HTTP is the protocol used to transfer data between a client (such as a web browser) and a server (such as a website). When you visit a website, your web browser sends an HTTP request to the server, asking for the webpage or other resource you've requested. The server then sends an HTTP response back to the client, containing the requested data.

HTTPS is a more secure version of HTTP, which encrypts the data being transmitted between the client and server using SSL/TLS (Secure Sockets Layer/Transport Layer Security) encryption. This provides an additional layer of security, helping to protect sensitive information such as login credentials, payment information, and other personal data.

When you visit a website that uses HTTPS, your web browser will display a padlock icon in the address bar, indicating that the connection is secure. You may also see the letters "https" at the beginning of the website address, rather than "http".

What is HTTP?

The Hypertext Transfer Protocol (HTTP) is the foundation of the World Wide Web, and is used to load webpages using hypertext links. HTTP is an [application layer](#) protocol designed to transfer information between networked devices and runs on top of other layers of the network [protocol](#) stack. A typical flow over HTTP involves a client machine making a request to a server, which then sends a response message.

What is in an HTTP request?

An HTTP request is the way Internet communications platforms such as web browsers ask for the information they need to load a website.

Each HTTP request made across the Internet carries with it a series of encoded data that carries different types of information. A typical HTTP request contains:

1. HTTP version type
2. a URL
3. an HTTP method
4. HTTP request headers
5. Optional HTTP body.

What is an HTTP method?

An HTTP method, sometimes referred to as an HTTP verb, indicates the action that the HTTP request expects from the queried server. For example, two of the most common HTTP methods are 'GET' and 'POST'; a 'GET' request expects information back in return (usually in the form of a website), while a 'POST' request typically indicates that the client is submitting information to the web server (such as form information, e.g. a submitted username and password).

What are HTTP request headers?

HTTP headers contain text information stored in key-value pairs, and they are included in every HTTP request (and response, more on that later). These headers communicate core information, such as what browser the client is using and what data is being requested.

What is in an HTTP response?

An HTTP response is what web clients (often browsers) receive from an Internet server in answer to an HTTP request. These responses communicate valuable information based on what was asked for in the HTTP request.

A typical HTTP response contains:

1. an HTTP status code
2. HTTP response headers
3. optional HTTP body

Let's break these down:

What's an HTTP status code?

HTTP status codes are 3-digit codes most often used to indicate whether an HTTP request has been successfully completed. Status codes are broken into the following 5 blocks:

1. 1xx Informational
2. 2xx Success
3. 3xx Redirection
4. 4xx Client Error
5. 5xx Server Error

The “xx” refers to different numbers between 00 and 99.

Status codes starting with the number ‘2’ indicate a success. For example, after a client requests a webpage, the most commonly seen responses have a status code of ‘200 OK’, indicating that the request was properly completed.

If the response starts with a ‘4’ or a ‘5’ that means there was an error and the webpage will not be displayed. A status code that begins with a ‘4’ indicates a client-side error (it is very common to encounter a ‘404 NOT FOUND’ status code when making a typo in a URL). A status code beginning in ‘5’ means something went wrong on the server side. Status codes can also begin with a ‘1’ or a ‘3’, which indicate an informational response and a redirect, respectively.

Full explanation of the status code:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Reference/Status>

What are HTTP response headers?

Much like an HTTP request, an HTTP response comes with headers that convey important information such as the language and format of the data being sent in the response body.

What is in an HTTP response body?

Successful HTTP responses to ‘GET’ requests generally have a body which contains the requested information. In most web requests, this is HTML data that a web browser will translate into a webpage.

For further and deeper study:

<https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides>

What is a Domain Name?

Domain names are a key part of the Internet infrastructure. They provide a human-readable address for any web server available on the Internet.

Any Internet-connected computer can be reached through a public [IP Address](#), either an IPv4 address (e.g., 192.0.2.172) or an IPv6 address (e.g., 2001:db8:8b73:0000:0000:8a2e:0370:1337).

Computers can handle such addresses easily, but people have a hard time finding out who is running the server or what service the website offers. IP addresses are hard to remember and might change over time.

To solve all those problems we use human-readable addresses called domain names.

Structure of domain names

A domain name has a simple structure made of several parts (it might be one part only, two, three...), separated by dots and **read from right to left**:



Each of those parts provides specific information about the whole domain name.

Who owns a domain name?

You cannot "buy a domain name". This is so that unused domain names eventually become available to be used again by someone else. If every domain name was bought, the web would quickly fill up with unused domain names that were locked and couldn't be used by anyone.

Instead, you pay for the right to use a domain name for one or more years. You can renew your right, and your renewal has priority over other people's applications. But you never own the domain name.

Companies called registrars use domain name registries to keep track of technical and administrative information connecting you to your domain name.

What is a URL?

A **URL** (Uniform Resource Locator) is the address of a unique resource on the internet. It is one of the key mechanisms used by [browsers](#) to retrieve published resources, such as HTML pages, CSS documents, images, and so on.

In theory, each valid URL points to a unique resource. In practice, there are some exceptions, the most common being a URL pointing to a resource that no longer exists or that has moved. As the resource represented by the URL and the URL itself are handled by the Web server, it is up to the owner of the web server to carefully manage that resource and its associated URL.

Basics: anatomy of a URL

Here are some examples of URLs:

`https://developer.mozilla.org`

`https://developer.mozilla.org/en-US/docs/Learn_web_development/`

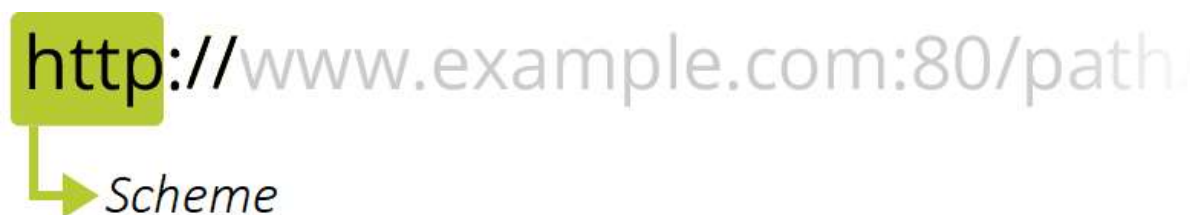
`https://developer.mozilla.org/en-US/search?q=URL`

Any of those URLs can be typed into your browser's address bar to tell it to load the associated resource, which in all three cases is a Web page.

A URL is composed of different parts, some mandatory and others optional. The most important parts are highlighted on the URL below (details are provided in the following sections):



Scheme



The first part of the URL is the *scheme*, which indicates the protocol that the browser must use to request the resource (a protocol is a set method for exchanging or transferring data around a computer network). Usually for websites the protocol is HTTPS or HTTP (its unsecured version). Addressing web pages requires one of these two, but browsers also know how to handle other schemes such as `mailto:` (to open a mail client), so don't be surprised if you see other protocols.

Authority

tp://**www.example.com:80**/path/to/my



Next follows the *authority*, which is separated from the scheme by the character pattern `://`. If present the authority includes both the *domain* (e.g., `www.example.com`) and the *port* (`80`), separated by a colon:

- The domain indicates which Web server is being requested. Usually this is a [domain name](#), but an [IP address](#) may also be used (but this is rare as it is much less convenient).
- The port indicates the technical "gate" used to access the resources on the web server. It is usually omitted if the web server uses the standard ports of the HTTP protocol (80 for HTTP and 443 for HTTPS) to grant access to its resources. Otherwise it is mandatory.

Path to resource

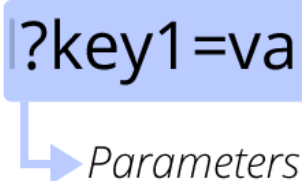
n:80/**/path/to/myfile.html**?key1=value1



`/path/to/myfile.html` is the path to the resource on the Web server. In the early days of the Web, a path like this represented a physical file location on the Web server. Nowadays, it is mostly an abstraction handled by Web servers without any physical reality.

Parameters

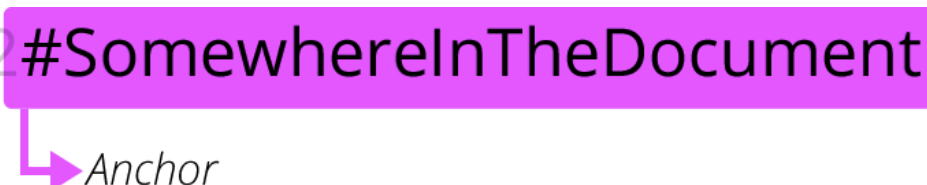
html?**?key1=value1&key2=value2**#Some



`?key1=value1&key2=value2` are extra parameters provided to the Web server. Those parameters are a list of key/value pairs separated with the `&` symbol. The Web server can use those parameters to do extra stuff before returning the resource. Each Web server has its own rules regarding parameters, and the only reliable way to know if a specific Web server is handling parameters is by asking the Web server owner.

Anchor

ue2/**#SomewhereInTheDocument**



#SomewhereInTheDocument is an anchor to another part of the resource itself. An anchor represents a sort of "bookmark" inside the resource, giving the browser the directions to show the content located at that "bookmarked" spot. On an HTML document, for example, the browser will scroll to the point where the anchor is defined; on a video or audio document, the browser will try to go to the time the anchor represents. It is worth noting that the part after the #, also known as the **fragment identifier**, is never sent to the server with the request.

How to use URLs

Any URL can be typed right inside the browser's address bar to get to the resource behind it. But this is only the tip of the iceberg!

The [HTML](#) language (see [Structuring content with HTML](#)) makes extensive use of URLs:

- to create links to other documents with the [<a>](#) element;
- to link a document with its related resources through various elements such as [<link>](#) or [<script>](#);
- to display media such as images (with the [](#) element), videos (with the [<video>](#) element), sounds and music (with the [<audio>](#) element), etc.;
- to display other HTML documents with the [<iframe>](#) element.

What is DNS?

The Domain Name System (DNS) is the phonebook of the Internet. Humans access information online through domain names, like nytimes.com or espn.com. Web browsers interact through Internet Protocol (IP) addresses. DNS translates domain names to IP addresses so browsers can load Internet resources.

Each device connected to the Internet has a unique IP address which other machines use to find the device. DNS servers eliminate the need for humans to memorize IP addresses such as 192.168.1.1 (in IPv4), or more complex newer alphanumeric IP addresses such as 2400:cb00:2048:1::c629:d7a2 (in IPv6).

How does DNS work?

The process of DNS resolution involves converting a hostname (such as www.example.com) into a computer-friendly IP address (such as 192.168.1.1). An IP address is given to each device on the Internet, and that address is necessary to find the appropriate Internet device - like a street address is used to find a particular home. When a user wants to load a webpage, a translation must occur between what a user types into their web browser (example.com) and the machine-friendly address necessary to locate the example.com webpage.

In order to understand the process behind the DNS resolution, it's important to learn about the different hardware components a DNS query must pass between. For the web browser, the DNS lookup occurs "behind the scenes" and requires no interaction from the user's computer apart from the initial request.

There are 4 DNS servers involved in loading a webpage:

DNS recursor - The recursor can be thought of as a librarian who is asked to go find a particular book somewhere in a library. The DNS recursor is a server designed to receive queries from client machines through applications such as web browsers. Typically the recursor is then responsible for making additional requests in order to satisfy the client's DNS query.

Root nameserver - The root server is the first step in translating (resolving) human readable host names into IP addresses. It can be thought of like an index in a library that points to different racks of books - typically it serves as a reference to other more specific locations.

TLD nameserver - The top level domain server (TLD) can be thought of as a specific rack of books in a library. This nameserver is the next step in the search for a specific IP address, and it hosts the last portion of a hostname (In example.com, the TLD server is "com").

Authoritative nameserver - This final nameserver can be thought of as a dictionary on a rack of books, in which a specific name can be translated into its definition. The authoritative nameserver is the last stop in the nameserver query. If the authoritative name server has access to the requested record, it will return the IP address for the requested hostname back to the DNS Recursor (the librarian) that made the initial request.

What are the steps in a DNS lookup?

For most situations, DNS is concerned with a domain name being translated into the appropriate IP address. To learn how this process works, it helps to follow the path of a DNS lookup as it travels from a web browser, through the DNS lookup process, and back again. Let's take a look at the steps.

Note: Often DNS lookup information will be cached either locally inside the querying computer or remotely in the DNS infrastructure. There are typically 8 steps in a DNS lookup. When DNS information is cached, steps are skipped from the DNS lookup process which makes it quicker. The example below outlines all 8 steps when nothing is cached.

The 8 steps in a DNS lookup:

1. A user types 'example.com' into a web browser and the query travels into the Internet and is received by a DNS recursive resolver.
2. The resolver then queries a DNS root nameserver (.).
3. The root server then responds to the resolver with the address of a Top Level Domain (TLD) DNS server (such as .com or .net), which stores the information for its domains. When searching for example.com, our request is pointed toward the .com TLD.
4. The resolver then makes a request to the .com TLD.
5. The TLD server then responds with the IP address of the domain's nameserver, example.com.
6. Lastly, the recursive resolver sends a query to the domain's nameserver.
7. The IP address for example.com is then returned to the resolver from the nameserver.
8. The DNS resolver then responds to the web browser with the IP address of the domain requested initially.

Once the 8 steps of the DNS lookup have returned the IP address for example.com, the browser is able to make the request for the web page:

9. The browser makes a [HTTP](#) request to the IP address.
10. The server at that IP returns the webpage to be rendered in the browser (step 10).

Resources

www.cloudflare.com

roadmap.sh

developer.mozilla.org