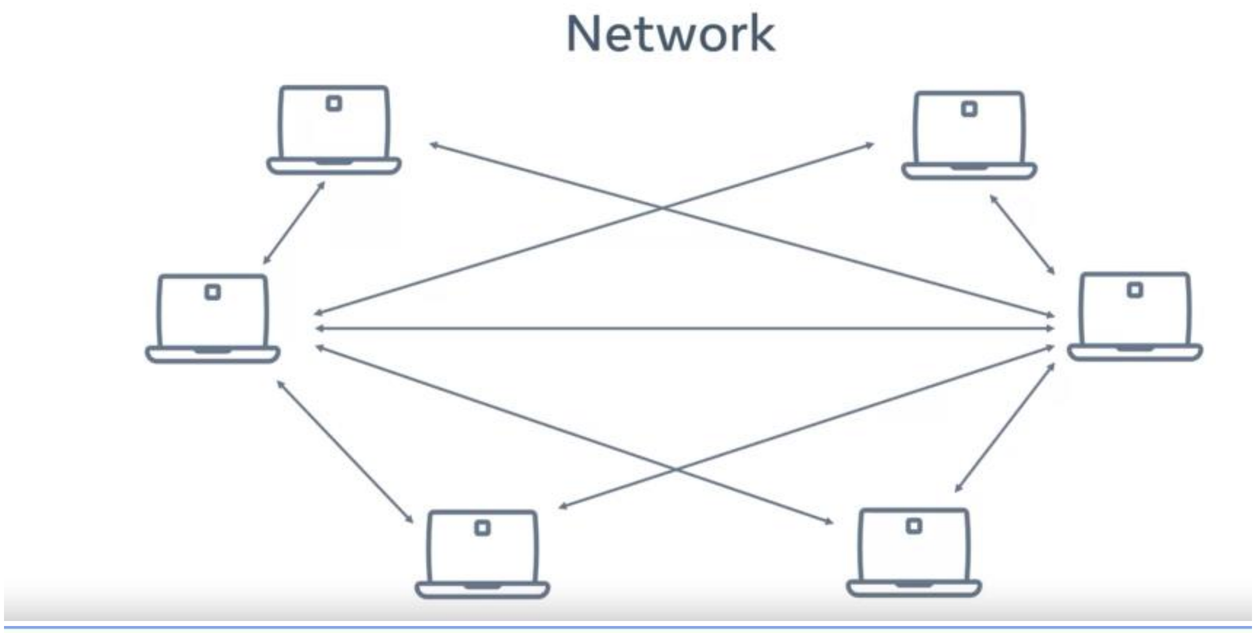
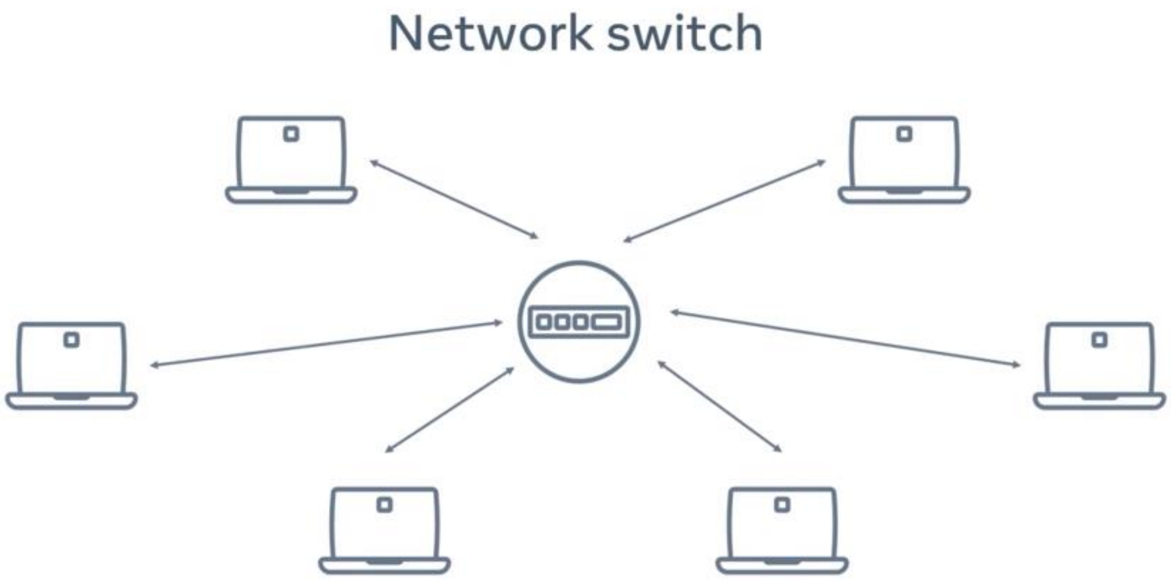
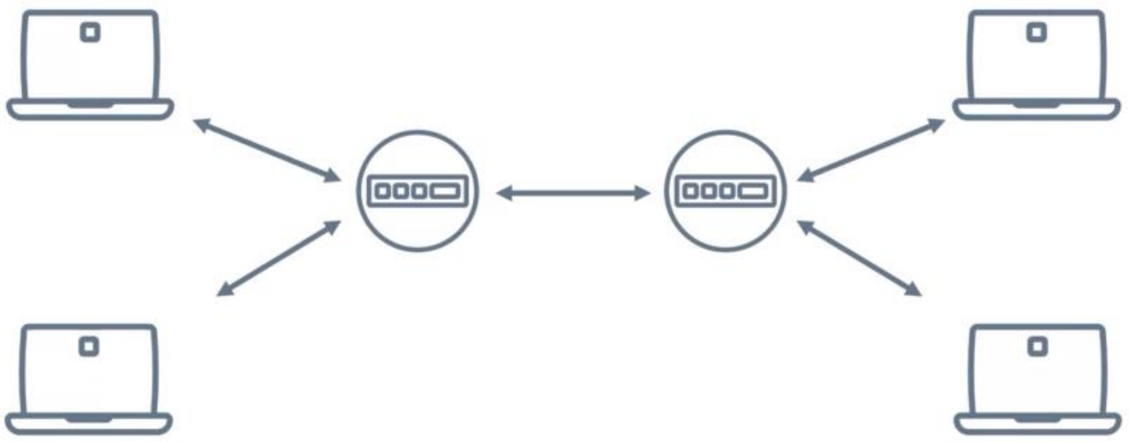
How the internet works

Did you ever find yourself searching for something online? How long does it take to get a response from your search? You search for something and you get a response back in no time. That's really something, isn't it? But did you ever think about how it all happens? In this video, I will guide you through the basics of how the Internet works. Let's start at the beginning. You open your favorite app on your device and you're instantly connected to the world. This is all made possible because two devices connect and communicate via a wired or wireless connection, forming something called a network. You can connect multiple devices to this network. But this becomes very complicated very quickly,
as each device needs to connect to every other device to communicate effectively. This problem is solved by something called a network switch that connects multiple devices and allows them to communicate with each other. The network switch can connect to other network switches, and now two networks can connect. These network switches then connect to more network switches until you have something called an interconnected network. This interconnected network has another name that you might be familiar with. It's called the Internet. When we use websites or video streaming services on the Internet, these are provided by computers called servers. Our devices are called clients. This is known as the client-server model. Internet connects the entire world. Have you ever had a video call with someone on another continent? That video data travel through large undersea cables connecting the world's networks. These cables can transfer huge volumes of data per second. There are a lot more technical details that go into making the Internet possible. But this is the main idea. Hopefully, this gave you the big picture of how the Internet works.

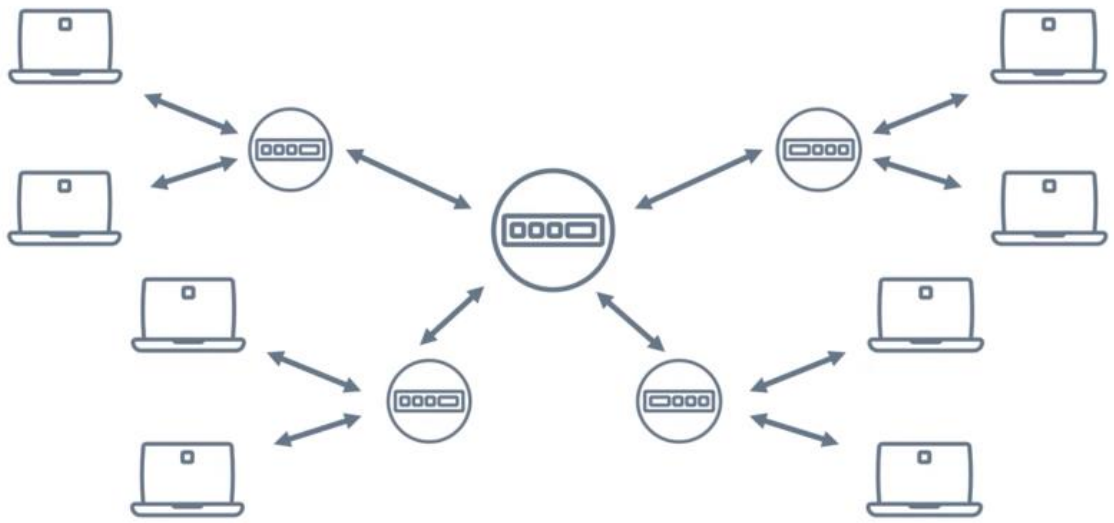

Network

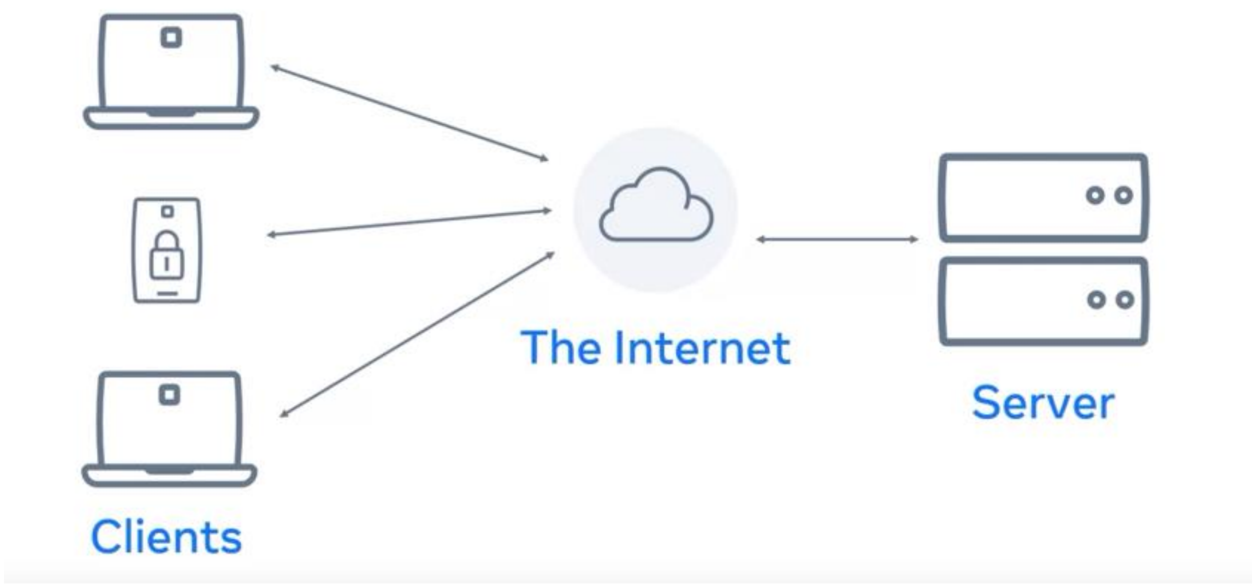

Network switch

Network switch



Interconnected network

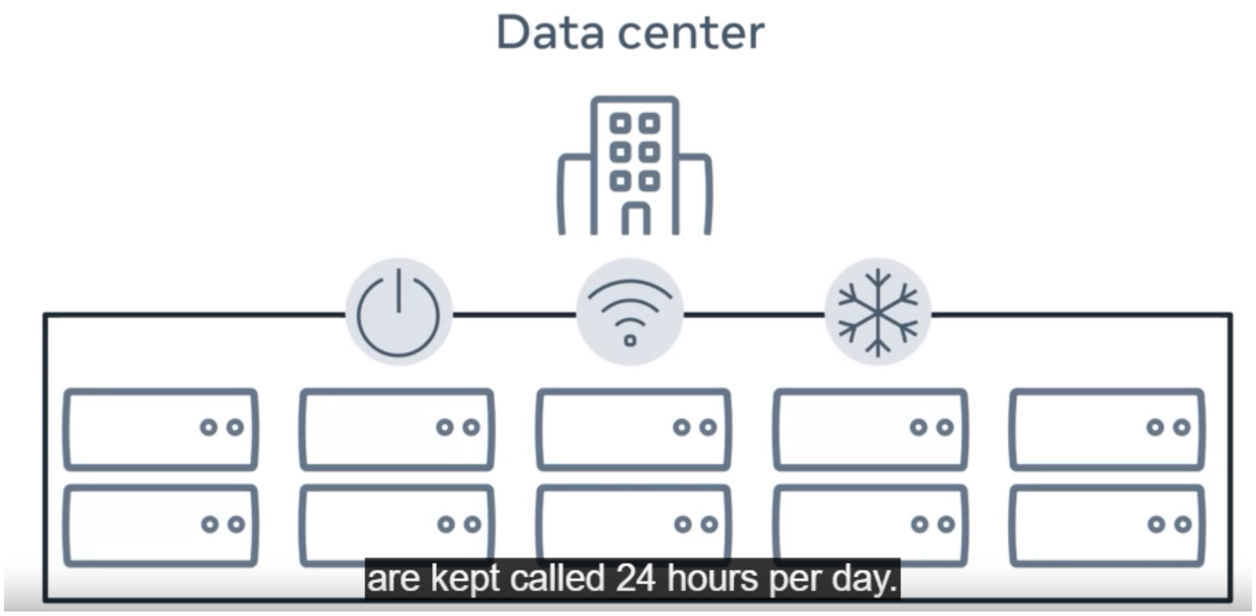


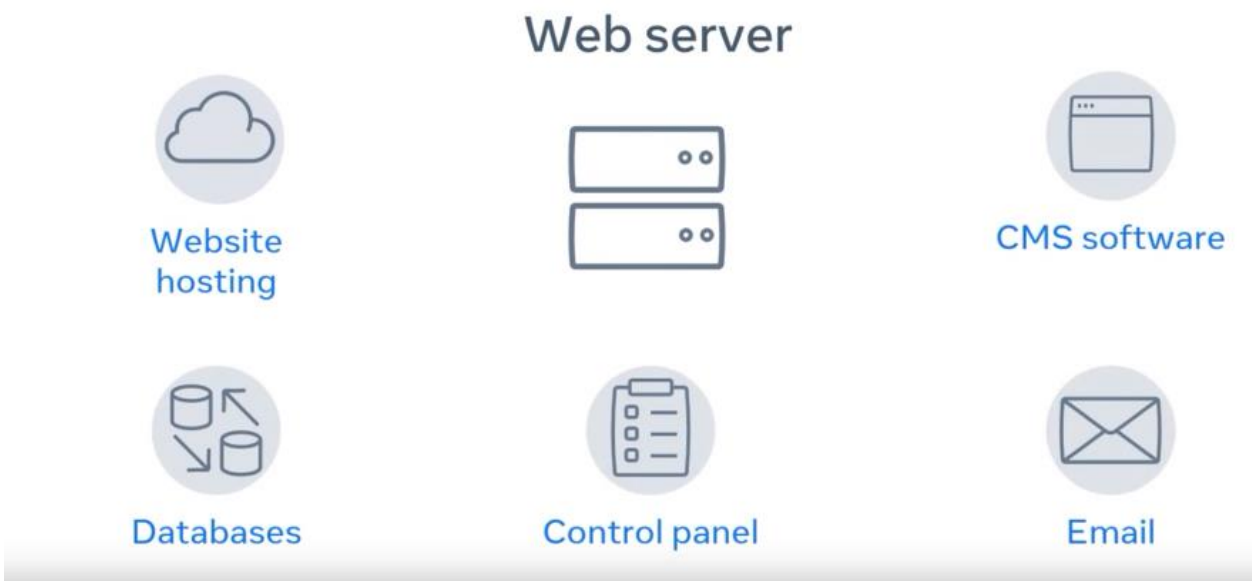


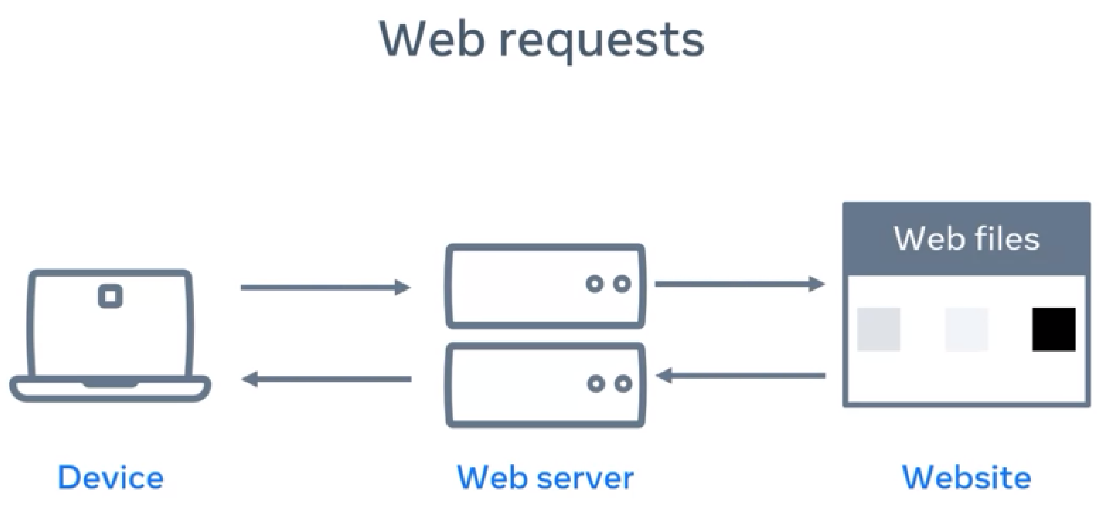
What is a web server and how does it work?

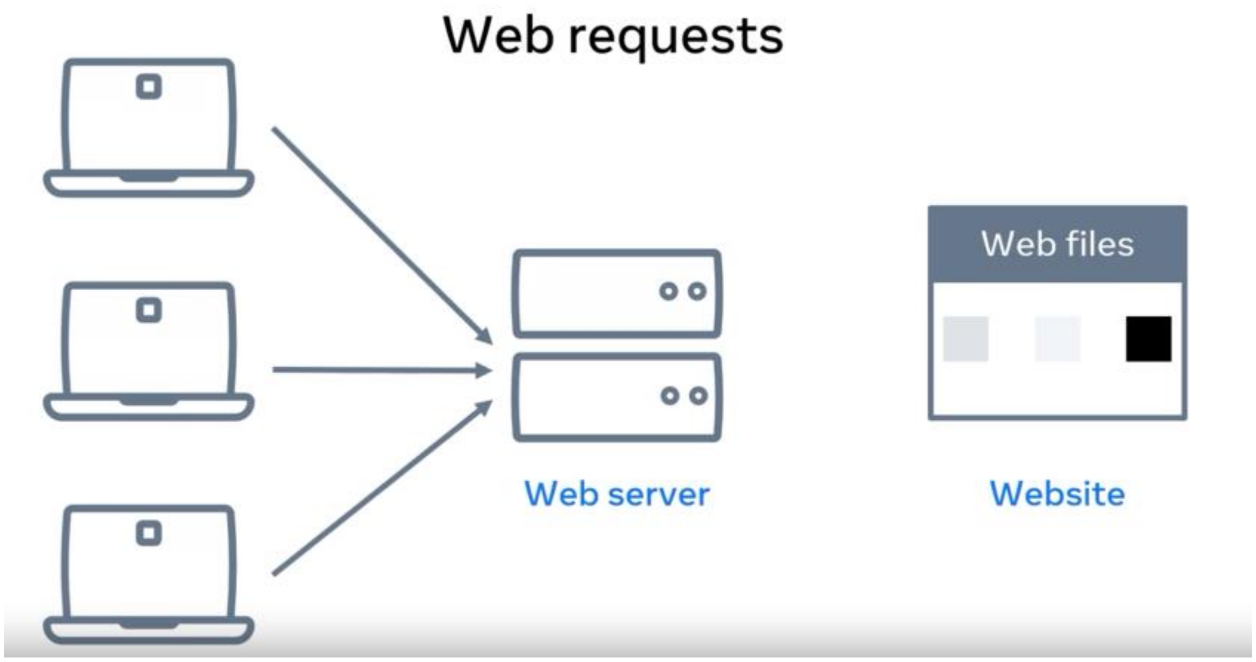
A server is a computer that runs applications and
services ranging from websites to instant messaging.
It's called a server because it provides a service to another computer and
its user also known as the client.
Is typically stored in something called a data center with hundreds or thousands of other servers, all running different services connected to the internet.
There are many different systems in data centers to ensure that servers have continuous power, continuous internet connection and
are kept called 24 hours per day.
Did you know that there are data centers located all around the world.
Many websites use these to allow you to access your content quickly from the data center nearest to you.
The data center servers are built based on the service purpose.
For example, if the server is only to be used for
storing images, it might have a lot of hard drive space.

Whereas a server computing complex calculations might need a fast processor and a lot of memory.
This is usually referred to as a server hardware.
The physical components of a server.
Once the server hardware is installed in the data center,
a piece of code can now run.
The code that runs on the hardware is commonly known as software.
One way I like to remember this is to think of hardware as something you can physically touch and
is difficult to change as you need to physically replace components.
Software is soft or easy to change.
You just replace the code running on the server.
The surface is a server can run can vary but
in this video you will learn about a type of service software known as a web server.
A web server has many functions which includes website storage and administration, data storage, security and managing email.
Another primary function is to handle something known as a web request.
When you open a browser on your device and type the name of the website,
it's the job of the web server to send you back to that website's content.
This process is known as the request response cycle and
you will learn more about it later.
Web servers are also designed to respond to thousands of requests from clients per second.
In this video, you learned about web servers and
how they consist of hardware and software.
You also learned that one of the primary roles of a web server is to respond to web requests from the client.


Data center
are kept called 24 hours per day.


Web server
Website hosting
CMS software
Databases
Control panel
Email





Web hosting

websites and files are stored on web servers located in datacenters.
But what if you wanted to create your own website? Do you really need your own datacenter with specialized hardware and software? Thankfully, the answer is no. Developers can launch websites to the Internet using something known as web hosting. Web hosting is a service where you place your website and files on

the hosting companies web server. You're essentially renting the space in return for stable and secure storage. You don't need to be accompanied to use a web host. Individuals can rent space too. First, let me share with you some of the different hosting options available. These can include **shared hosting, virtual private hosting, dedicated hosting, and Cloud hosting**. Let's explore each of these hosting types in a little more detail now. The cheapest form of web hosting is known as shared hosting.
You pay for a location on a web server containing many web hosting accounts with shared hosting. This means that you also share the service processing power, memory, and bandwidth with other websites that might slow your performance. This option is best for a small website with a small number of visitors. Many developers also use this as a low-cost sandbox environment to practice deploying or hosting their personal websites. Some companies offer free shared hosting, but with limitations and often have advertisements embedded in the webpages. Sites with more considerable demands use virtual private surface or VPS. A VPS is a virtual server with dedicated CPU, memory, and bandwidth resources. It will be running on a hardware server with other VPS instances but as the resources are fixed per VPS instance, your website is unlikely to be impacted by the performance of other VPS instances. A VPS instance will be more expensive than shared hosting. The next option up is to use dedicated hosting. This will be a hardware server that is dedicated to you only. All hardware, CPU, memory, and bandwidth resources are yours to use. Generally, this option is more expensive than a VPS hosting. The last type of web hosting is something you may have heard of. Cloud hosting and the Cloud has grown in popularity over the last decade and is often mentioned in various news and services you use. With Cloud hosting, your website is run in something called a Cloud environment, which spans across multiple physical and virtual servers. If a physical or virtual server fails, your website will run on a different server and stay online.
The main advantage of Cloud hosting is that you can use as many resources as you need without hardware limitations. However, you pay based on resource use. For example, if you transfer a file from the Cloud to a web browser, you'll pay for the bandwidth used for that transfer at a fractional cent cost per megabyte. While this can quickly become more expensive, is allows websites and web applications to scale their costs as popularity grows. This is how many of the major web applications operate. In this video, you learned about web hosting and the different hosting options available to individuals and companies. Soon you will build your very first website. Are you excited to get it hosted so you can share it

with the world? For more information on web hosting and Services, please see the additional reading at the end of this lesson.

What is a Web Server? (NGINX)

https://www.nginx.com/resources/glossary/web-server/

What is a Web Browser? (Mozilla)

https://www.mozilla.org/en-US/firefox/browsers/what-is-a-browser/

Who invented the Internet? And why? (Kurzgesagt)

https://youtu.be/21eFwbb48sE

What is Cloud Computing? (Amazon)

https://youtu.be/mxT233EdY5c

Browser Engines (Wikipedia)

https://en.wikipedia.org/wiki/Browser_engine

Introduction to Internet Protocols

Email is a common communication method that we all know about.
But before existed the alternative was to send mail to one another through the postal system.
There is a surprising parallel between the postal system and
how the computer sends and receives data across the internet every day.
Let's compare how they both work.
Data sent across the internet is quite an important part of our
everyday lives and
it wouldn't be possible without Internet Protocol addresses or IP addresses.
A useful way to think of IP addresses is that they function much like
addresses in a postal system that make it possible for
packets of information to be delivered to you.
And like with the postal system things can go wrong.
But let's first go over how things work.

Before we think about how they can go wrong in this video you will learn what
IP addresses are and explore how computers send data across the internet.
You probably learned how computers connect to each other to form networks and
how these networks connect to each other to form the internet.
When you send data between computers across the internet,
a common way of understanding that data is needed by the computers and
networks that the data travels across.
What makes that possible is the Internet Protocol.
Version four and version six are currently the two most widely used
standards of internet protocol.
Think of the old fashioned postal system again when you send a letter to a friend
you need their address otherwise they won't receive your letter.
Computers work in a similar way.
Every computer on a network is assigned an IP address.
In protocol version four an IP address contains four octet.
It's separated by periods or dots.
For example 192.0.2.235.
In protocol version six.
An IP address contains eight groups of hexadecimal digits separated by a colon.
For example
4527:0a00:1567:0200:ff00:0042:8329.
Play video starting at :2:11 and follow transcript2:11
When you send data across a network,
you send the data as a series of messages called IP packets.
Also known as data grams at a high level IP packets
contain a header and a payload or the data.
Think of that old fashioned postal system again, when you send a letter.
You not only include the recipient's address but
also your own address in case a return location is needed.
IP packets are the same.
They include the destination IP address and source IP address.
These addresses are in the header along with some additional information to help
deliver the packet.
And the payload contains the data of the packet and
some of the other protocols which will cover in a moment.
Earlier I mentioned that things can go wrong with the postal system.
When sending multiple letters to a friend it's possible they may arrive out

of order.
It's possible that a package will get damaged or
if you're really unlucky a letter could get lost.
These issues can happen to IP packets too they can arrive out of order,
become damaged or corrupted to in transit or be dropped or lost during transit.
To solve these problems,
the payload part of the packets contains other protocols too.
You can think of them as another message inside the payload of the IP packet.
The two most common protocols are the Transmission Control Protocol
referred to as TCP and the User Datagram Protocol, also known as UDP.
TCP can solve all three of the previously mentioned issues but
at the cost of a small delay when sending the data.
This protocol is used for sending the data that must arrive correctly and
in order such as a text or image files.
UDP solves the corrupt packet issue but
packets can still arrive out of order or not arrive at all.
This protocol is used for sending data that can tolerate some data loss such
as voice calls or live video streaming.
Both of these protocols contain payloads that contain further protocols inside
of them and there you have it.
The internet uses internet protocols much like an old fashioned postal system.
These protocols can help to make sure that data
arrives in order does not become corrupted or lost or dropped during transit.
Now you're able to explain how IP addresses work and
how computers send data across the internet.

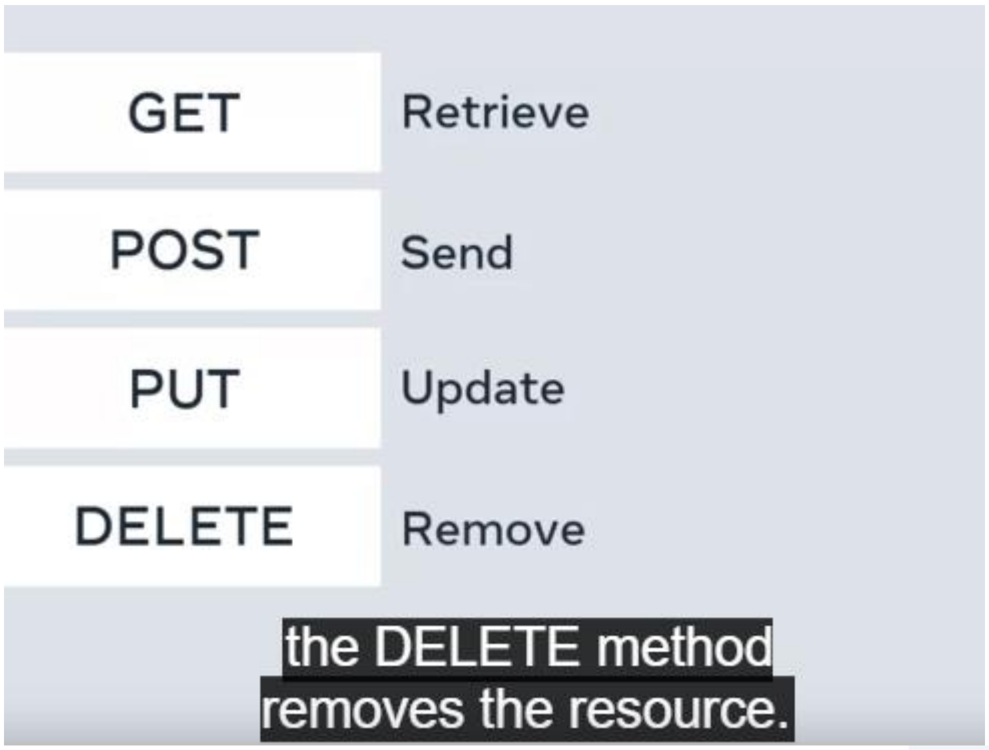
Introduction to HTTP

Have you ever noticed the lock icon beside the URL in your web browser? This means the secure version of HTTP is being used. HTTP is a core operational protocol of the world wide web. It is what enables your web browser to communicate with a web server that hosts a website. HTTP is the communication protocol you use whenever you browse the web. HTTP stands for Hypertext Transfer Protocol is a protocol used for transferring web resources such as HTML documents, images, styles, and other files. HTTP is a request response based protocol. A web browser or client sends an HTTP request to a server, and the webserver sends the HTTP response back to the browser. Next, let's start exploring the makeup of an HTTP request. An HTTP requests consists of a method,

path, version, and headers. The HTTP method describes the type of action that the client was to perform. The primary or the most commonly used HTTP methods are GET, POST, PUT and DELETE. The GET method is used to retrieve information from the given server. The POST request is used to send data to the server.
The PUT method updates whatever currently exist on the website with something else and the DELETE method removes the resource. The path is the representation of where the resource is stored on the webserver. For example, if you requested an image at https://example.com/index.html, the path would be /index.html. There are multiple versions of the HTTP protocol. I won't explore these right now, but I want you to be aware that Version 1.1 and 2.0 are the most used. Finally, there are the headers. Headers contain additional information about the request, and the client that is making the request. For certain requests methods, the requests will also contain a body of content that the client is sending. Now, let's cover some details about the makeup of an HTTP response. HTTP responses follow a format similar to the request format. Following the header, the response will optionally contain a message body consisting of the response contents such as the HTML document, the image file, and so forth. HTTP status codes indicate if the HTTP requests successfully completed. The code values are in the range of a 100-599 and a grouped by purpose. The status message is a text representation of the status code. During your web browsing, have you ever encountered pages that display 404 error not found or 500 errors? Server is not responding? These are HTTP status codes. I want to briefly explain to you about the status codes and their grouping. There are five groups of status codes. They're grouped by the first digit of the error number. Informational is grouped from 100-199. Successful responses are grouped from 200-299. Redirection message are from 300-399. Client error responses ranged from 400-499 and server error responses are from 500- 599. Information responses are provisional responses sent by the server. These responses are interim before the actual response. The most common inflammation response is 100 Continue, which indicates that the web client should continue to request or ignore the response if the request is already finished. Successful responses indicate that the request was successfully processed by the web server, with the most common success response being 200 Ok. You're receiving these responses every day when you receive content successfully from a website. The meaning of Ok depends on the HTTP method. If the method is GET, it means that the resource is found and is

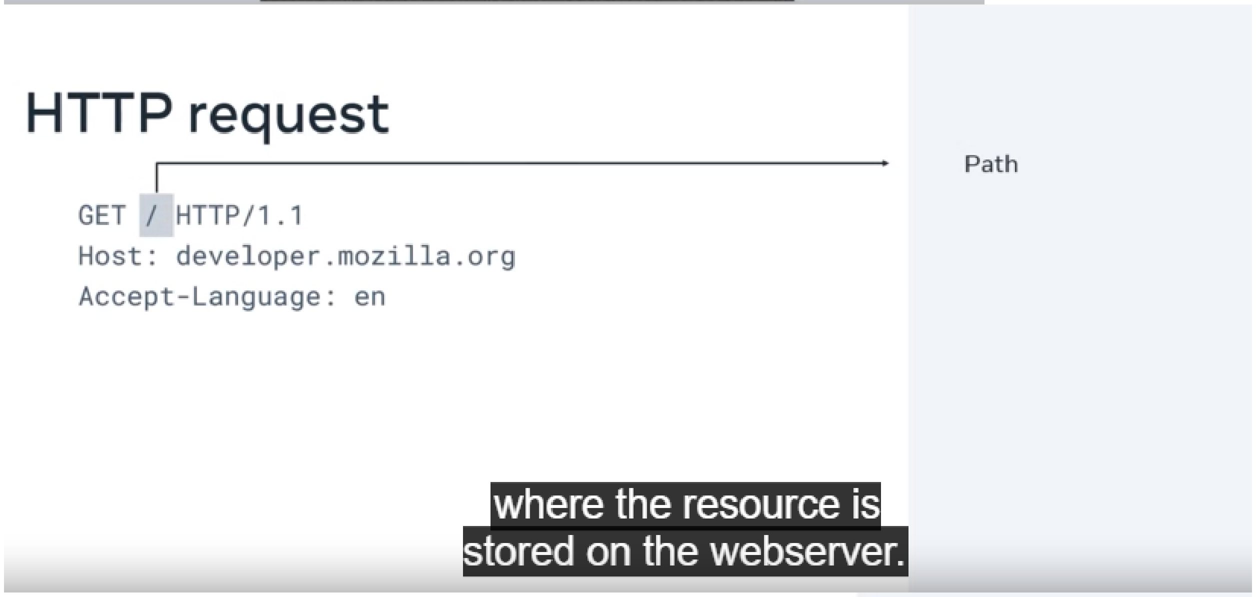
included in the body of the HTTP response. If it's POST, it means that the resource was successfully transmitted to the webserver. If it's PUT, the resource was successfully transmitted to the webserver. Finally, if the method is DELETE, it means the resource was deleted. Redirection responses indicate to the web client that the requested resource has been moved to a different path. The most common response codes used are 301 Moved Permanently and 302 Found.
The difference between the redirection messages 301 and 302 is that 302 indicates a temporary redirection. The resource has been temporarily moved. When web browsers receive these responses, they will automatically submit the request for the resource at the new path. Client error responses indicate that the requests contained bad syntax or content and cannot be processed by the webserver. The most common codes used are, 400 is used where the web browser or client submitted bad data to the webserver. 401 is used to indicate that the user must log into an account before the request can be processed. 403 is used to indicate the request was valid, but that the webserver is refusing to process it. This is often used to indicate that a user does not have sufficient permissions to execute an action in a web application. 404 is used to indicate that the request resource was not found on the webserver.
Server error responses indicate that a failure occurred on the webserver while trying to process the request. The most common code used is 500 Internal Server Error, which is a generic error status indicating that the server fail to process the request. Now, have you ever bought something online and needed to enter your credit card information? You wouldn't want someone else to get this information from the HTTP request. This is where HTTPS is involved. HTTPS is the secure version of HTTP. It is used for secure communication between two computers so that nobody else can see the information being sent and received. It does this by using something called encryption. We won't cover encryption right now.
Like an HTTP, the requests and responses still behave in the same way and have the same content. The big difference is before the content is sent, it is turned into a secret code. Only the other computer can turn the secret code back into its original content. If someone else was to look at the code, it wouldn't be understandable. You use HTTPS every day. This is the lock icon you see beside the URL in your web browser. Before I finish, I want to leave you with a brief summary of HTTP. Firstly, it is a protocol used by web clients and web servers. It works to transfer web resources such as HTML files and as the foundation of any data exchanges on the web. Also, remember that by using HTTPS,

we send the information securely. Requests are sent by the client, usually a web browser, and the server replies with responses which may be the return of an image or an HTML page. HTTP requests have a syntax that includes method, path, versions, and headers. HTTP responses follow a similar format to the request. An HTTP status codes indicate whether the HTTP requests successfully completed. The status code is a three-digit number that corresponds with groups representing different types of results. Now you know how the HTTP protocol request and response cycle works. You know about its methods and the elements that make up an HTTP request. Good job.



GET	Retrieve
POST	Send
PUT	Update
DELETE	Remove

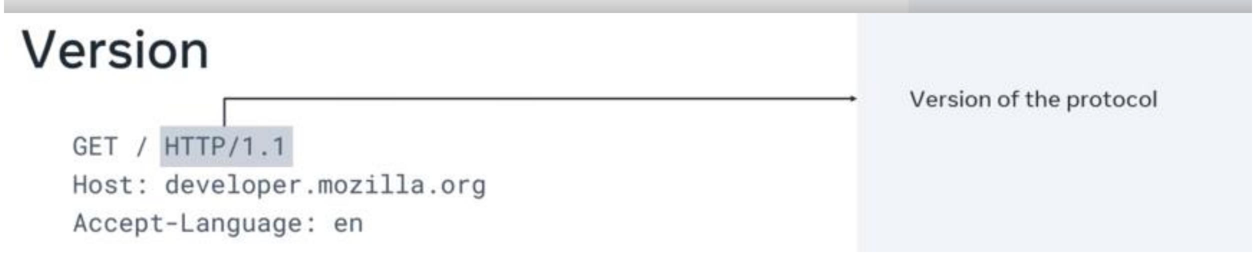
the DELETE method
removes the resource.



HTTP request

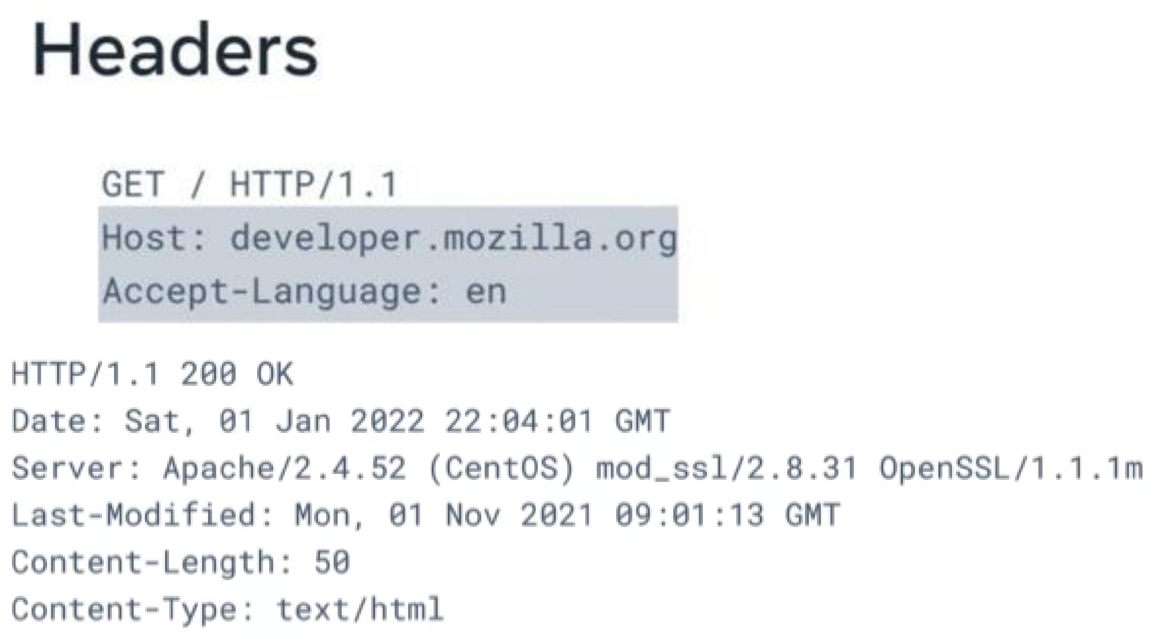
```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: en
```

where the resource is
stored on the webserver.



Version

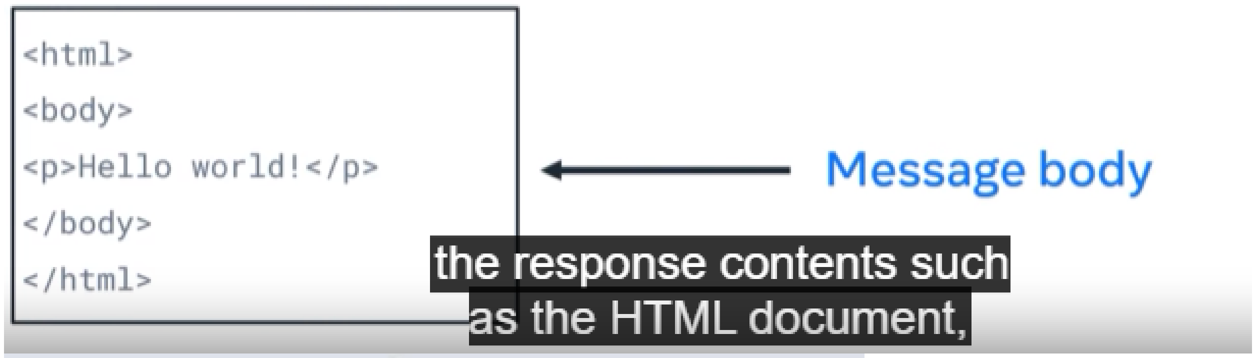
```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: en
```



Headers

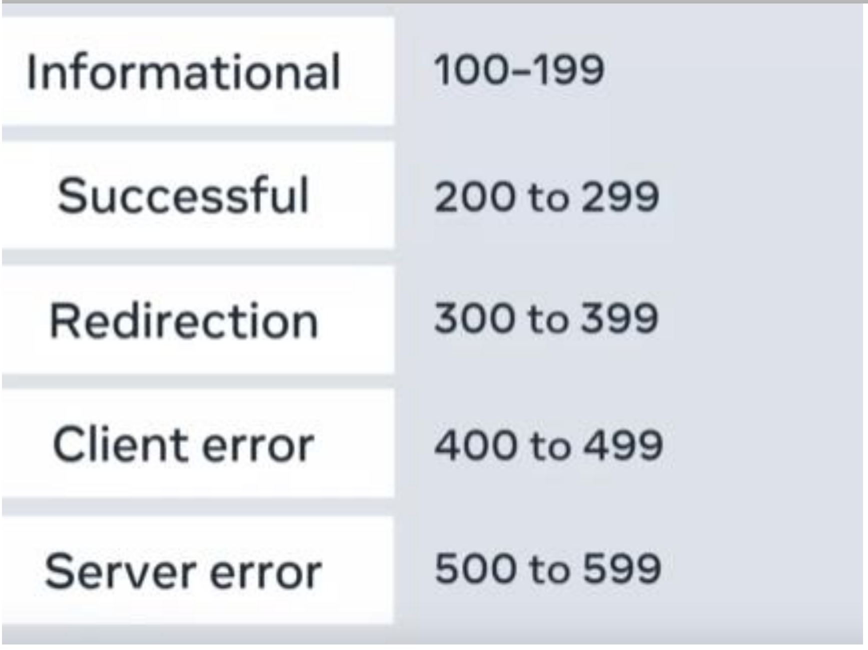
```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: en
```

```
HTTP/1.1 200 OK
Date: Sat, 01 Jan 2022 22:04:01 GMT
Server: Apache/2.4.52 (CentOS) mod_ssl/2.8.31 OpenSSL/1.1.1m
Last-Modified: Mon, 01 Nov 2021 09:01:13 GMT
Content-Length: 50
Content-Type: text/html
```



```
<html>
<body>
<p>Hello world!</p>
</body>
</html>
```

the response contents such
as the HTML document,



Informational	100–199
Successful	200 to 299
Redirection	300 to 399
Client error	400 to 499
Server error	500 to 599

HTTP examples

This reading explores the contents of HTTP requests and responses in more depth.

Request Line

Every HTTP request begins with the request line.

This consists of the HTTP method, the requested resource and the HTTP protocol version.

```
GET /home.html HTTP/1.1
```

In this example, `GET` is the HTTP method, `/home.html` is the resource requested and HTTP 1.1 is the protocol used.

HTTP Methods

HTTP methods indicate the action that the client wishes to perform on the web server resource.

Common HTTP methods are:

HTTP Method	Description
GET	The client requests a resource on the web server.
POST	The client submits data to a resource on the web server.
PUT	The client replaces a resource on the web server.
DELETE	The client deletes a resource on the web server.

HTTP Request Headers

After the request line, the HTTP headers are followed by a line break.

There are various possibilities when including an HTTP header in the HTTP request. A header is a case-insensitive name followed by a `:` and then followed by a value.

Common headers are:

```
1. Host: example.com
2. User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:50.0)
   Gecko/20100101 Firefox/50.0
3. Accept: */*
4. Accept-Language: en
5. Content-type: text/json
```

- The **`Host`** header specifies the host of the server and indicates where the resource is requested from.
- The **`User-Agent`** header informs the web server of the application that is making the request. It often includes the operating system (Windows, Mac, Linux), version and application vendor.
- The **`Accept`** header informs the web server what type of content the client will accept as the response.
- The **`Accept-Language`** header indicates the language and optionally the locale that the client prefers.
- The **`Content-type`** header indicates the type of content being transmitted in the request body.

HTTP Request Body

HTTP requests can optionally include a request body. A request body is often included when using the HTTP POST and PUT methods to transmit data.

```
POST /users HTTP/1.1
Host: example.com
{
"key1":"value1",
"key2":"value2",
  "array1":["value3","value4"]
}
```

```
PUT /users/1 HTTP/1.1
Host: example.com
Content-type: text/json
{"key1":"value1"}
```

HTTP Responses

When the web server is finished processing the HTTP request, it will send back an HTTP response.

The first line of the response is the status line. This line shows the client if the request was successful or if an error occurred.

```
HTTP/1.1 200 OK
```

The line begins with the HTTP protocol version, followed by the status code and a reason phrase. The reason phrase is a textual representation of the status code.

HTTP Status Codes

The first digit of an HTTP status code indicates the category of the response: Information, Successful, Redirection, Client Error or Server Error.

The common status codes you'll encounter for each category are:

1XX Informational

Status Code	Reason Phrase	Description
100	Continue	The server received the request headers and should continue send the request body.
101	Switching Protocols	The client has requested the server to switch protocols and th server has agreed to do so.

2XX Successful

Status Code	Reason Phrase	Description
200	OK	Standard response returned by the server to indicate it successfu processed the request.
201	Created	The server successfully processed the request and a resource w created.
202	Accepted	The server accepted the request for processing but the processi not yet been completed.
204	No Content	The server successfully processed the request but is not returnin content.

3XX Redirection

Status Code	Reason Phrase	Description
301	Moved Permanently	This request and all future requests should be sent to the r location.
302	Found	This request should be sent to the returned location.

4XX Client Error

Status Code	Reason Phrase	Description
400	Bad Request	The server cannot process the request due to a client error, e.g., ir request or transmitted data is too large.
401	Unauthorized	The client making the request is unauthorized and should authenti
403	Forbidden	The request was valid but the server is refusing to process it. This usually returned due to the client having insufficient permissions fo website, e.g., requesting an administrator action but the user is not administrator.
404	Not Found	The server did not find the requested resource.
405	Method Not Allowed	The web server does not support the HTTP method used.

5XX Server Error

Status Code	Reason Phrase	Description
500	Internal Server Error	A generic error status code given when an unexpected error or condition occurred while processing the request.
502	Bad Gateway	The web server received an invalid response from the Applicati Server.
503	Service Unavailable	The web server cannot process the request.

HTTP Response Headers

Following the status line, there are optional HTTP response headers followed by a line break.

Similar to the request headers, there are many possible HTTP headers that can be included in the HTTP response.

Common response headers are:

```
 Date: Fri, 11 Feb 2022 15:00:00 GMT+2
Server: Apache/2.2.14 (Linux)
Content-Length: 84
Content-Type: text/html
```

- The `Date` header specifies the date and time the HTTP response was generated.
- The `Server` header describes the web server software used to generate the response.

- The `Content-Length` header describes the length of the response.
- The `Content-Type` header describes the media type of the resource returned (e.g. HTML document, image, video).

HTTP Response Body

Following the HTTP response headers is the HTTP response body. This is the main content of the HTTP response.

This can contain images, video, HTML documents and other media types.

```
HTTP/1.1 200 OK
Date: Fri, 11 Feb 2022 15:00:00 GMT+2
Server: Apache/2.2.14 (Linux)
Content-Length: 84
Content-Type: text/html

<html>
  <head><title>Test</title></head>
  <body>Test HTML page.</body>
</html>
```

Other Internet Protocols

Hypertext Transfer Protocols (HTTP) are used on top of Transmission Control Protocol (TCP) to transfer webpages and other content from websites. This reading explores other protocols commonly used on the Internet.

Dynamic Host Configuration Protocol (DHCP)

You've learned that computers need IP addresses to communicate with each other. When your computer connects to a network, the Dynamic Host Configuration Protocol or DHCP as it is commonly known, is used to assign your computer an IP address. Your computer communicates over User Datagram Protocol (UDP) using the protocol with a type of server called a DHCP server. The server keeps track of computers on the network and their IP addresses. It will assign your computer an IP address and respond over the protocol to let it know which IP address to use. Once your computer has an IP address, it can communicate with other computers on the network.

Domain Name System Protocol (DNS)

Your computer needs a way to know with which IP address to communicate when you visit a website in your web browser, for example, `meta.com`. The Domain Name System Protocol, commonly known as DNS, provides this function. Your computer then checks with the DNS server associated with the domain name and then returns the correct IP address.

Internet Message Access Protocol (IMAP)

Do you check your emails on your mobile or tablet device? Or maybe you use an email application on your computer? Your device needs a way to download emails and manage your mailbox on the server storing your emails. This is the purpose of the Internet Message Access Protocol or IMAP.

Simple Mail Transfer Protocol (SMTP)

Now that your emails are on your device, you need a way to send emails. The Simple Mail Transfer Protocol, or SMTP, is used. It allows email clients to submit emails for sending via an SMTP server. You can also use it to receive emails from an email client, but IMAP is more commonly used.

Post Office Protocol (POP)

The Post Office Protocol (POP) is an older protocol used to download emails to an email client. The main difference in using POP instead of IMAP is that POP will delete the emails on the server once they have been downloaded to your local device. Although it is no longer commonly used in email clients, developers often use it to implement email automation as it is a more straightforward protocol than IMAP.

File Transfer Protocol (FTP)

When running your websites and web applications on the Internet, you'll need a way to transfer the files from your local computer to the server they'll run on. The standard protocol used for this is the File Transfer Protocol or FTP. FTP allows you to list, send, receive and delete files on a server. Your server must run an FTP Server and you will need an FTP Client on your local machine. You'll learn more about these in a later course.

Secure Shell Protocol (SSH)

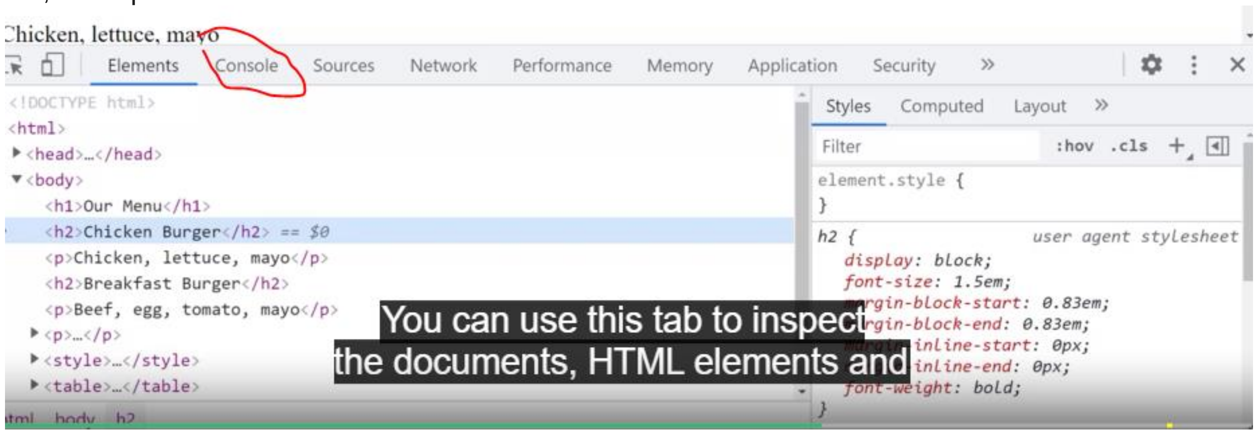
When you start working with servers, you'll also need a way to log in and interact with the computer remotely. The most common method of doing this is using the Secure Shell Protocol, commonly referred to as SSH. Using an SSH client allows you to connect to an SSH server running on a server to perform commands on the remote computer. All data sent over SSH is encrypted. This means that third parties cannot understand the data transmitted. Only the sending and receiving computers can understand the data.

SSH File Transfer Protocol (SFTP)

The data is transmitted insecurely when using the File Transfer Protocol. This means that third parties may understand the data that you are sending. This is not right if you transmit company files such as software and databases. To solve this, the SSH File Transfer Protocol, alternatively called the Secure File Transfer Protocol, can be used to transfer files over the SSH protocol. This ensures that the data is transmitted securely. Most FTP clients also support the SFTP protocol.

Web site and web application

The easy way to remember this is that a website is more informative and a web application is more interactive.
1st, let's open the console tab.



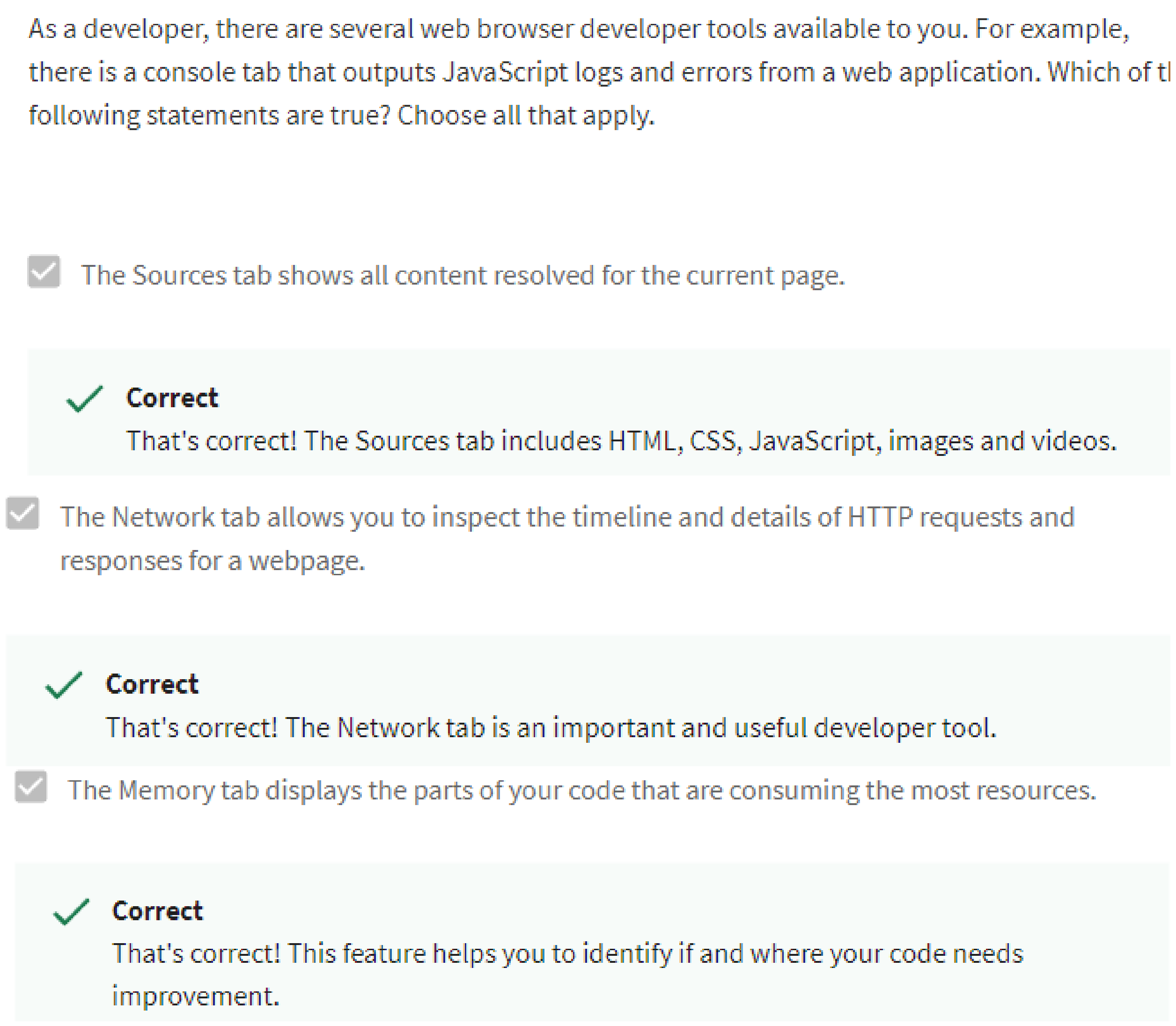
```
<!DOCTYPE html>
<html>
 ▶<head>…</head>
 ▼<body>
    <h1>Our Menu</h1>
    <h2>Chicken Burger</h2> == $0
    <p>Chicken, lettuce, mayo</p>
    <h2>Breakfast Burger</h2>
    <p>Beef, egg, tomato, mayo</p>
   ▶<p>…</p>
   ▶<style>…</style>
   ▶<table>…</table>
```

```
element.style {
}
h2 {                       user agent stylesheet
   display: block;
   font-size: 1.5em;
   margin-block-start: 0.83em;
   margin-block-end: 0.83em;
   margin-inline-start: 0px;
   margin-inline-end: 0px;
   font-weight: bold;
}
```

You can use this tab to inspect
the documents, HTML elements and

This tab outputs, javascript logs and errors from your web application.
The sources tab shows all content resolved for the current page.
It includes HTML, CSS, Javascript, images and videos.
With the network tab, you can inspect the timeline and
details of http requests and responses for the web page.
The performance tab shows what the web browser is doing over time.
It is useful if your web application is running slow because you can pinpoint
the functions that are taking the most time.
The memory tab displays the parts of your code that are consuming the most resources.
Finally, let's check the most used tab the elements tab.
You can use this tab to inspect the documents, HTML elements and
their properties.
For example, when I hover over an element in the elements tab,
it highlights that element in the browser pane.
On the right side of the panel, there are tabs for
inspecting the details of the elements further.

This panel shows us what CSS is applied to an element and
the result of the element displayed in the browser.
We will explore these details in a later lesson.
For now, you just need to know that if you click on an HTML element,
then the information for that element will appear in the tab.
Now, I'm going to demonstrate a fun trick.
If you double click the HTML, you can edit it in the web browser.
For example, if I select the Our Menu body element, then I can change the first item
in the menu from chicken Burger, to Turkey Burger.



As a developer, there are several web browser developer tools available to you. For example, there is a console tab that outputs JavaScript logs and errors from a web application. Which of tl
following statements are true? Choose all that apply.

[✓] The Sources tab shows all content resolved for the current page.

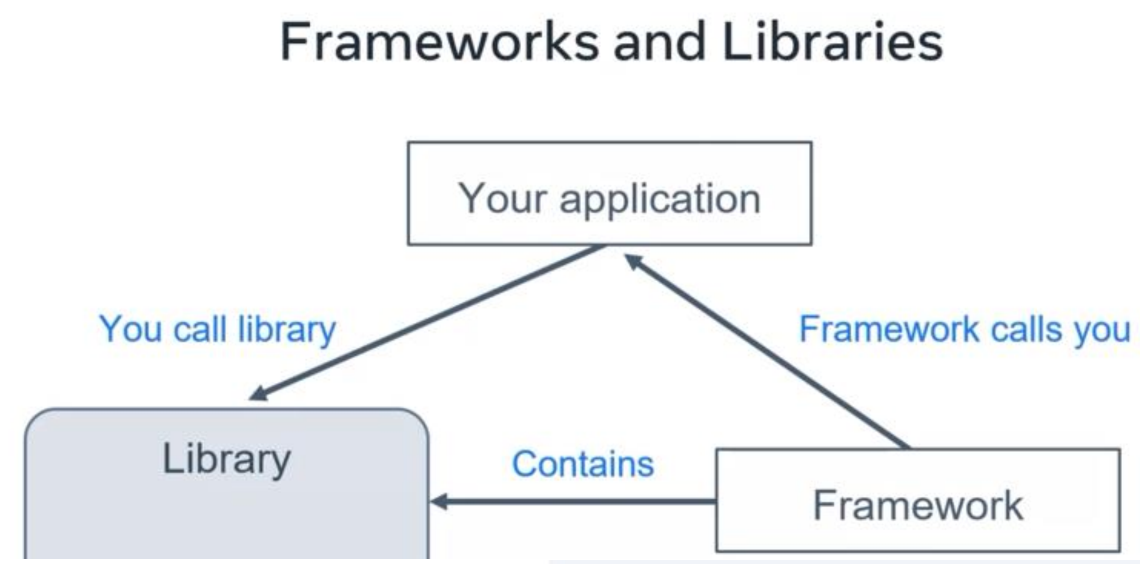
✓ **Correct**
That's correct! The Sources tab includes HTML, CSS, JavaScript, images and videos.

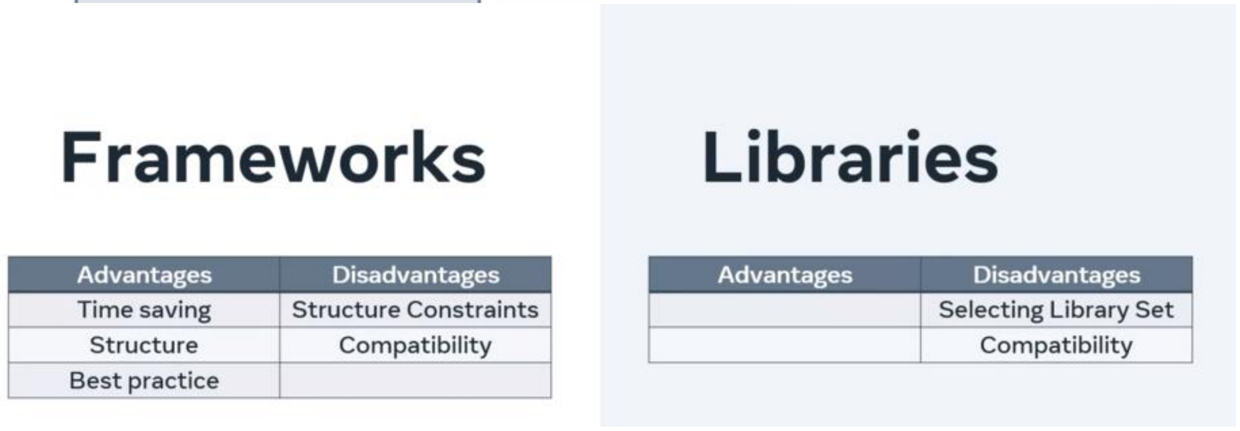
[✓] The Network tab allows you to inspect the timeline and details of HTTP requests and responses for a webpage.

✓ **Correct**
That's correct! The Network tab is an important and useful developer tool.

[✓] The Memory tab displays the parts of your code that are consuming the most resources.

✓ **Correct**
That's correct! This feature helps you to identify if and where your code needs improvement.





Frameworks

Advantages	Disadvantages
Time saving	Structure Constraints
Structure	Compatibility
Best practice	

Libraries

Advantages	Disadvantages
	Selecting Library Set
	Compatibility

You are developing solutions but you need
to save some time and build faster.
What if some of your build problems
have already been solved for you?
Well it's true someone has already figured out
many key development processes and they're contained in
frameworks and libraries that are used
every day in software development,
so what exactly are frameworks and libraries?
Let's say you are not a developer,
but instead you work as a carpenter.
You make chairs and sell them online.
As a carpenter you don't design
a new hammer for every chair you make.

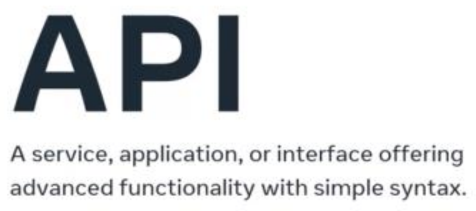
It makes much more sense to use an existing hammer,
but of course you are a developer.
As such it's important for you to
know that to speed up development,
developers use already developed frameworks
and libraries in their application development.
These might be open source,
meaning that the source code is
freely-available for anyone to modify and build from.
There are thousands of open source libraries and
frameworks available or there might be proprietary,
ones that are licensed or developed internally.
Many developers use the terms framework
and library interchangeably,
so what's the difference between them?
Libraries are reusable pieces of
code that can be used by your application.
They are purpose-built to
provide a specific functionality.
To give a more technical example,
you're building a small e-commerce website.
When a user wants to register they
need to provide their email address.
Email addresses while easy to
read can be complicated to validate.
In fact email addresses are
defined across several technical specifications.
That's a lot of reading just to validate an email.
Even if you do read through all the specification,
there's no point in spending hours
or even days implementing their standards
because you have access to
so many readily-available libraries
to validate email addresses.
It is for specific functionality like
validating an email address that libraries are useful.
A developers simply uses the library

to access the functionality they require,
as a result they can have more time to
continue focusing on
the development of their application.
Frameworks on the other hand provide
a structure for developers to build with.
Consider this in the context of our carpenter analogy.
As a carpenter you create a lot of different chairs,
therefore there would be
a blueprint for each chair to speed up building them.
You can decide the type of wood to use,
but the dimensions and
style of the chair are always the same.
Frameworks act as a structure where the developer
provides their own code
that the framework interacts with.
For example, there are
many frameworks for developing web applications.
These frameworks handle functionality
that is common to all web applications
such as receiving HTTP requests
and sending HTTP responses.
The developer then adds their own code that
implements the functionality of the web application.
For instance with the e-commerce website example,
a framework would handle receiving HTTP requests.
The developer would implement
code that processes the request and returns
a response from which the framework
would send a response over HTTP.
Now let's compare how the frameworks relate to libraries.
Most frameworks use many libraries.
The libraries that the framework uses
can be used for your application.
If you wish, your application
can also use other libraries.
You also need to consider when to use

a framework and when to use a library.
Frameworks are considered opinionated
and libraries are considered unopinionated.
This is defined as the degree of freedom
available to the developer to
choose how to code a feature.
The opinionatedness will vary between frameworks,
but by definition they will always
be more opinionated than a library.
The benefit of this is that they can
replace libraries as needed.
For example when new technologies become
available frameworks to find
the libraries flow and control of an application,
whereas with the libraries
those are left to the developer to decide.
As with everything there are
advantages and disadvantages to both.
Frameworks are a great way to reduce
development time and to enforce
a structure on how code is written.
They have many best practices already in place and
contain most of what is needed
to develop an application,
however, sometimes you may find that the way you need to
code something doesn't fit
into the structure of the framework.
Other times you may find that some of
the libraries the framework uses may
conflict with a library that you are required to
use and cause compatibility issues.
If an application is built without a framework,
the developer will need to decide on the set of libraries
they wish to use to achieve
the functionality they must deliver.
They will also need to take care that
the selected libraries can work together.

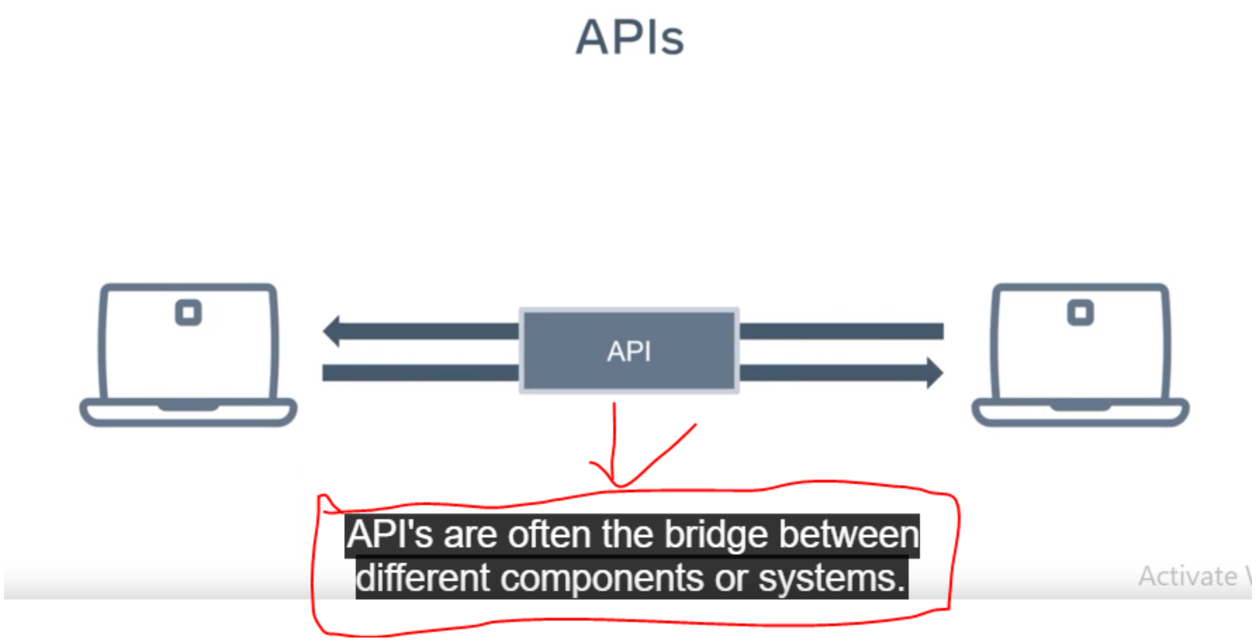
The upside to this is that they can
replace libraries as needed.
For example, if a new better library is released,
the developer can replace the usage of the old library.
This is much easier than replacing a framework.
Frameworks and libraries give you the opportunity
to reuse existing web app functions.
This can result in faster development, fewer errors,
and more time for you to spend on
the essential features of your application.
Instead of reinventing the wheel,
you can use frameworks and libraries that are designed
specifically to help your web app development processes.

<u>API & web service</u>



API

A service, application, or interface offering advanced functionality with simple syntax.



API's are often the bridge between different components or systems.

Every day you access information on your phone, like reading the news, purchasing
goods and services or communicating with friends over social media.
But how is all this information transferred behind the scenes?
Your favorite websites and apps.
Probably use API's and as a web developer, you'll discover that API's developer friendly, easily accessible and a very valuable and useful development tool.
A PI is the acronym for application programming interface.
An API is a set of functions and procedures for
creating applications that access the features or data of an operating system, application or other service.
If this still sounds a bit vague, just remember that the term API,
is intentionally open too many applications and use cases.
As a web developer, a lot of the day to day job involves working with API's.
Some common API's that web developers work with include Browser,
API REST API and Sensor-Based API.
Over the next few minutes, you'll explore each of these API types and review a few specific examples.
To begin with, here's a brief outline of how a piecewise functions.
In Software development,
API's are often the bridge between different components or systems.
This earns them names like gateway or middleware.
The term is used widely to represent many different tools and systems.
Let's consider some examples of different API use cases.
One common type of API, is Browser or Web APIs,
which are built into the browser itself.
They extend the functionality of the browser by adding new services and are designed to simplify complex functions and provide easy syntax for building advanced features.
A good example, is the DOM API.
The DOM API turns the html document into a tree of nodes that are represented as JavaScript objects.
Another example, is the geolocation API that returns coordinates of where the browser is located.
There are also other API's available for **fetching** data known as **Fetch API drawing, graphics or Canvas API keeping history or history API.**
And client side storage also known as Web Storage API.

Another critical type of API for web development is the RESTful or REST API.
This kind of API provides data for popular web and mobile apps.
These are also called web servers.
Let's explore REST in a bit more detail.
REST or representational state transfer,
is a set of principles that help build highly efficient API's.
One of the main responsibilities of these kinds of API's is sending and receiving data to and from a centralized database.
We can query our own REST API or third party ones.
One last type of API,
that you might encounter as a web developer is a Sensor-Based API.
This is what the internet of things also known as IOT is based on.
These are actual physical senses that are interconnected with each other.
The sensors can communicate through API and track and respond to physical data.
Some examples are Philips hue, smart lights and node bots.
That's a lot of API to think about.
Fortunately, for web developers the most common data API is a RESTtful API which
as you've learned is a web server that provides responses to requests.
These API web servers are designed to provide the data backbone for
a web client like a web page or mobile app.
This means that these API's must be able to accomplish things like getting data or get, creating data.
Also referred to as post updating data or put and deleting data or delete.
API issues, REST principles and
good design practices to create discoverable interfaces.
This helps us get the exact response expected.
But exactly how do they work?
Here's a closer description of their activity.
These API's use endpoints to specify how different resources can be accessed.
The endpoint is built into the URL when accessing the API.
Once the endpoint is hit, the API performs whatever service side
processing is needed to build the response.
Two common forms of response are full web page is a data form based on JavaScript called Jason.
In this video, you explored some API's and as a web developer,
you will frequently work with many different types of API's.

You will often use API's to extend the abilities of systems or
to act as a bridge between different components.

HTTP Overview (Mozilla)

https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Introduction to Networking by Dr.Charles R Severance

https://www.amazon.com/Introduction-Networking-How-Internet-Works/dp/1511654945/

Chrome Developer Tools Overview (Google)

https://developer.chrome.com/docs/devtools/overview/

Firefox Developer Tools User Docs (Mozilla)

https://firefox-source-docs.mozilla.org/devtools-user/index.html

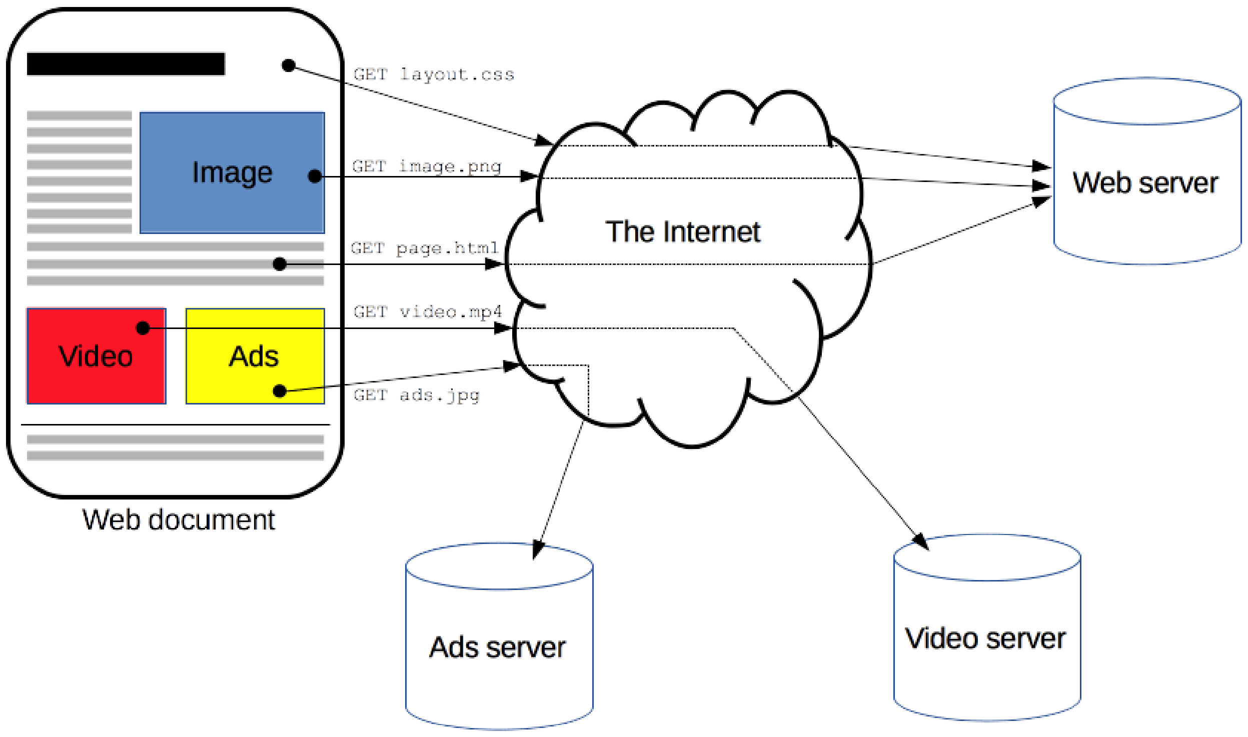
Getting Started with Visual Studio Code (Microsoft)

https://code.visualstudio.com/docs

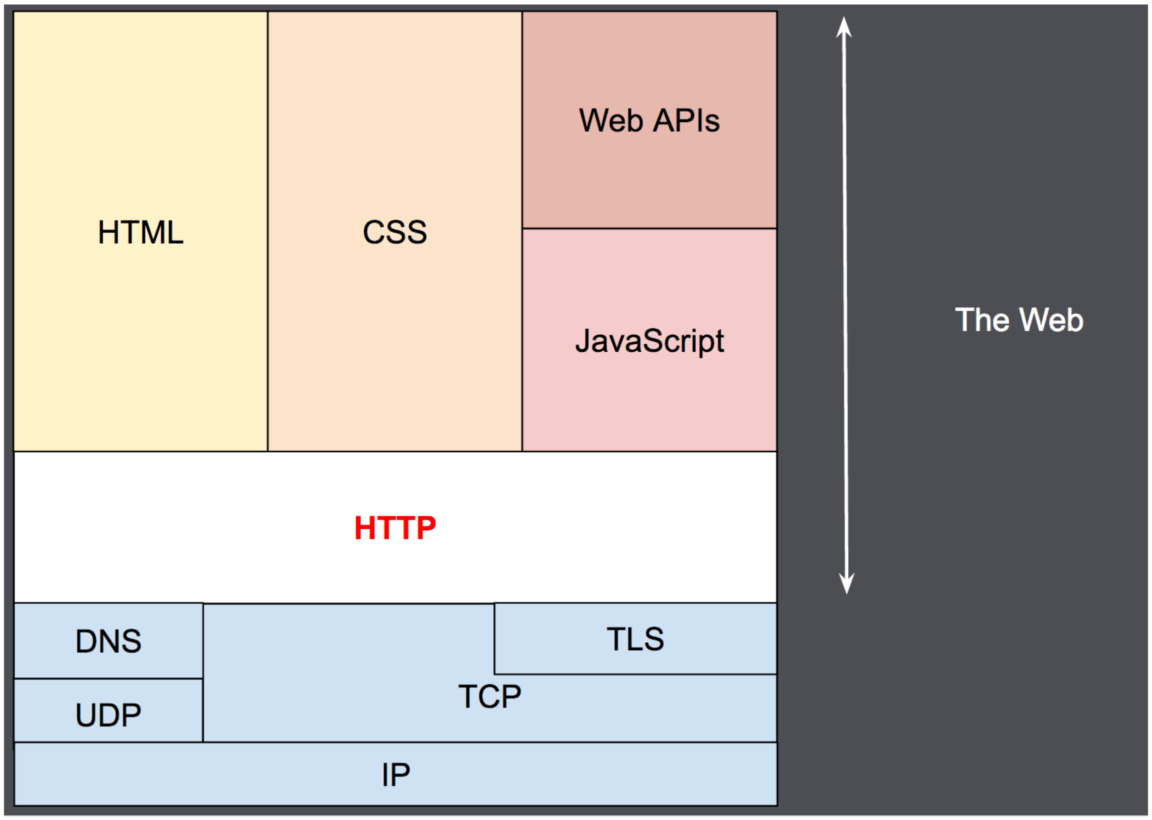
{

An overview of HTTP

HTTP is a protocol for fetching resources such as HTML documents. It is the foundation of any data exchange on the Web and it is a client-server protocol, which means requests are initiated by the recipient, usually the Web browser. A complete document is reconstructed from the different sub-documents fetched, for instance, text, layout description, images, videos, scripts, and more.



Clients and servers communicate by exchanging individual messages (as opposed to a stream of data). The messages sent by the client, usually a Web browser, are called *requests* and the messages sent by the server as an answer are called *responses*.

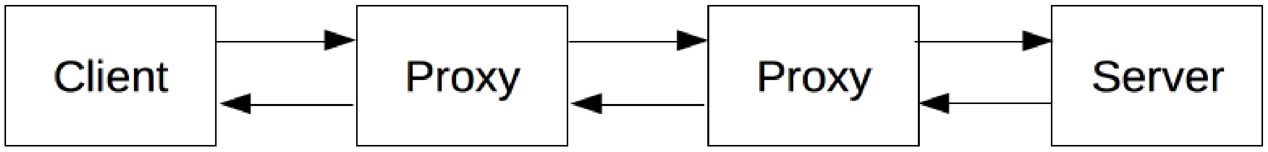


Designed in the early 1990s, HTTP is an extensible protocol which has evolved over time. It is an application layer protocol that is sent over TCP, or over a TLS-encrypted TCP connection, though any reliable transport protocol could theoretically be used. Due to its extensibility, it is used to not only fetch hypertext documents, but also images and videos or to post content to servers, like with HTML form results. HTTP can also be used to fetch parts of documents to update Web pages on demand.

Components of HTTP-based systems

HTTP is a client-server protocol: requests are sent by one entity, the user-agent (or a proxy on behalf of it). Most of the time the user-agent is a Web browser, but it can be anything, for example, a robot that crawls the Web to populate and maintain a search engine index.

Each individual request is sent to a server, which handles it and provides an answer called the *response*. Between the client and the server there are numerous entities, collectively called proxies, which perform different operations and act as gateways or caches, for example.



In reality, there are more computers between a browser and the server handling the request: there are routers, modems, and more. Thanks to the layered design of the Web, these are hidden in the network and transport layers. HTTP is on top, at the application layer. Although important for diagnosing network problems, the underlying layers are mostly irrelevant to the description of HTTP.

Client: the user-agent

The *user-agent* is any tool that acts on behalf of the user. This role is primarily performed by the Web browser, but it may also be performed by programs used by engineers and Web developers to debug their applications.

The browser is **always** the entity initiating the request. It is never the server (though some mechanisms have been added over the years to simulate server-initiated messages).

To display a Web page, the browser sends an original request to fetch the HTML document that represents the page. It then parses this file, making additional requests corresponding to execution scripts, layout information (CSS) to display, and sub-resources contained within the page (usually images and videos). The Web browser then combines these resources to present the complete document, the Web page. Scripts executed by the browser can fetch more resources in later phases and the browser updates the Web page accordingly.

A Web page is a hypertext document. This means some parts of the displayed content are links, which can be activated (usually by a click of the mouse) to fetch a new Web page, allowing the user to direct their user-agent and navigate through the Web. The browser translates these directions into HTTP requests, and further interprets the HTTP responses to present the user with a clear response.

The Web server

On the opposite side of the communication channel is the server, which *serves* the document as requested by the client. A server appears as only a single machine virtually; but it may actually be a collection of servers sharing the load (load balancing), or a complex piece of software interrogating other computers (like cache, a DB server, or e-commerce servers), totally or partially generating the document on demand.

A server is not necessarily a single machine, but several server software instances can be hosted on the same machine. With HTTP/1.1 and the Host header, they may even share the same IP address.

Proxies

Between the Web browser and the server, numerous computers and machines relay the HTTP messages. Due to the layered structure of the Web stack, most of these operate at the transport, network or physical levels, becoming transparent at the HTTP layer and potentially having a significant impact on performance. Those operating at the application layers are generally called **proxies**. These can be transparent, forwarding on the requests they receive without altering them in any way, or non-transparent, in which case they will change the request in some way before passing it along to the server. Proxies may perform numerous functions:

- caching (the cache can be public or private, like the browser cache)
- filtering (like an antivirus scan or parental controls)
- load balancing (to allow multiple servers to serve different requests)
- authentication (to control access to different resources)
- logging (allowing the storage of historical information)

Basic aspects of HTTP

HTTP is simple

HTTP is generally designed to be simple and human readable, even with the added complexity introduced in HTTP/2 by encapsulating HTTP messages into frames. HTTP messages can be read and understood by humans, providing easier testing for developers, and reduced complexity for newcomers.

HTTP is extensible

Introduced in HTTP/1.0, HTTP headers make this protocol easy to extend and experiment with. New functionality can even be introduced by a simple agreement between a client and a server about a new header's semantics.

HTTP is stateless, but not sessionless

HTTP is stateless: there is no link between two requests being successively carried out on the same connection. This immediately has the prospect of being problematic for users attempting to interact with certain pages coherently, for example, using e-commerce shopping baskets. But while the core of HTTP itself is stateless, HTTP cookies allow the use of stateful sessions. Using header extensibility, HTTP Cookies are added to the workflow, allowing session creation on each HTTP request to share the same context, or the same state.

HTTP and connections

A connection is controlled at the transport layer, and therefore fundamentally out of scope for HTTP. HTTP doesn't require the underlying transport protocol to be connection-based; it only requires it to be *reliable*, or not lose messages (at minimum, presenting an error in such cases). Among the two most common transport protocols on the Internet, TCP is reliable and UDP isn't. HTTP therefore relies on the TCP standard, which is connection-based.

Before a client and server can exchange an HTTP request/response pair, they must establish a TCP connection, a process which requires several round-trips. The default behavior of HTTP/1.0 is to open a separate TCP connection for each HTTP request/response pair. This is less efficient than sharing a single TCP connection when multiple requests are sent in close succession.

In order to mitigate this flaw, HTTP/1.1 introduced *pipelining* (which proved difficult to implement) and *persistent connections*: the underlying TCP connection can be partially controlled using the Connection header. HTTP/2 went a step further by multiplexing messages over a single connection, helping keep the connection warm and more efficient.

Experiments are in progress to design a better transport protocol more suited to HTTP. For example, Google is experimenting with QUIC which builds on UDP to provide a more reliable and efficient transport protocol.

What can be controlled by HTTP

This extensible nature of HTTP has, over time, allowed for more control and functionality of the Web. Cache and authentication methods were functions handled early in HTTP history. The ability to relax the *origin constraint*, by contrast, was only added in the 2010s.

Here is a list of common features controllable with HTTP:

- *Caching*: How documents are cached can be controlled by HTTP. The server can instruct proxies and clients about what to cache and for how long. The client can instruct intermediate cache proxies to ignore the stored document.
- *Relaxing the origin constraint*: To prevent snooping and other privacy invasions, Web browsers enforce strict separation between Web sites. Only pages from the **same origin** can access all the information of a Web page. Though such a constraint is a burden to the server, HTTP headers can relax this strict separation on the server side, allowing a document to become a patchwork of information sourced from different domains; there could even be security-related reasons to do so.
- *Authentication*: Some pages may be protected so that only specific users can access them. Basic authentication may be provided by HTTP, either using the WWW-Authenticate and similar headers, or by setting a specific session using HTTP cookies.
- *Proxy and tunneling*: Servers or clients are often located on intranets and hide their true IP address from other computers. HTTP requests then go through proxies to cross this network barrier. Not all proxies are HTTP proxies. The SOCKS protocol, for example, operates at a lower level. Other protocols, like ftp, can be handled by these proxies.
- *Sessions*: Using HTTP cookies allows you to link requests with the state of the server. This creates sessions, despite basic HTTP being a state-less protocol. This is useful not only for e-commerce shopping baskets, but also for any site allowing user configuration of the output.

HTTP flow

When a client wants to communicate with a server, either the final server or an intermediate proxy, it performs the following steps:

1. Open a TCP connection: The TCP connection is used to send a request, or several, and receive an answer. The client may open a new connection, reuse an existing connection, or open several TCP connections to the servers.

2. Send an HTTP message: HTTP messages (before HTTP/2) are human-readable. With HTTP/2, these simple messages are encapsulated in frames, making them impossible to read directly, but the principle remains the same. For example:
3. GET / HTTP/1.1
4. Host: developer.mozilla.org
5. Accept-Language: fr

   Copy to Clipboard

6. Read the response sent by the server, such as:
7. HTTP/1.1 200 OK
8. Date: Sat, 09 Oct 2010 14:28:02 GMT
9. Server: Apache
10. Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
11. ETag: "51142bc1-7449-479b075b2891b"
12. Accept-Ranges: bytes
13. Content-Length: 29769
14. Content-Type: text/html
15.
16. <!DOCTYPE html>… (here come the 29769 bytes of the requested web page)

   Copy to Clipboard

17. Close or reuse the connection for further requests.

If HTTP pipelining is activated, several requests can be sent without waiting for the first response to be fully received. HTTP pipelining has proven difficult to implement in existing networks, where old pieces of software coexist with modern versions. HTTP pipelining has been superseded in HTTP/2 with more robust multiplexing requests within a frame.

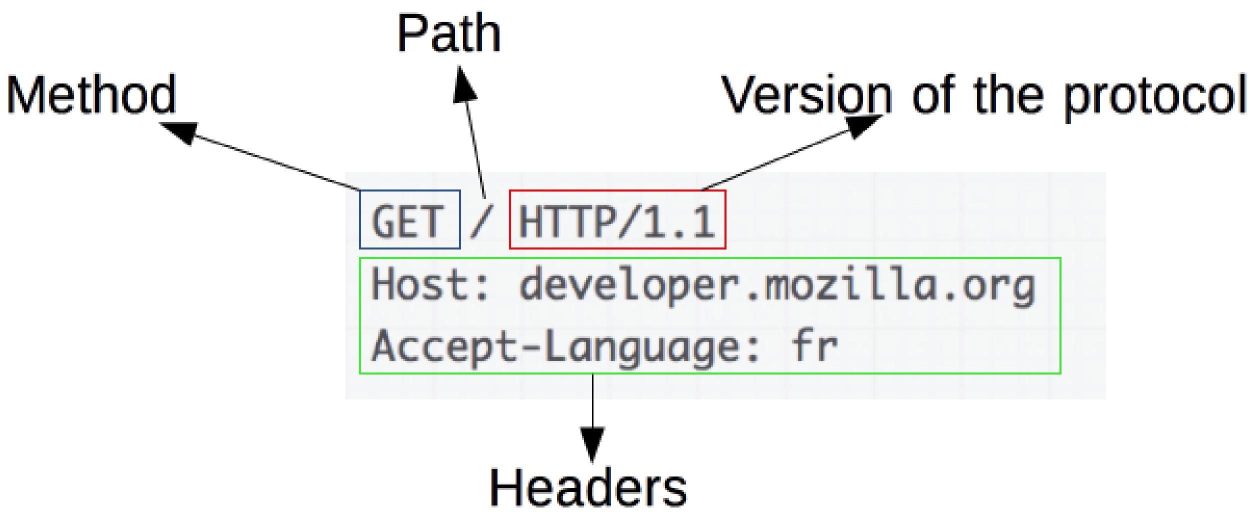
HTTP Messages

HTTP messages, as defined in HTTP/1.1 and earlier, are human-readable. In HTTP/2, these messages are embedded into a binary structure, a *frame*, allowing optimizations like compression of headers and multiplexing. Even if only part of the original HTTP message is sent in this version of HTTP, the semantics of each message is unchanged and the client reconstitutes (virtually) the original HTTP/1.1 request. It is therefore useful to comprehend HTTP/2 messages in the HTTP/1.1 format.

There are two types of HTTP messages, requests and responses, each with its own format.

Requests

An example HTTP request:



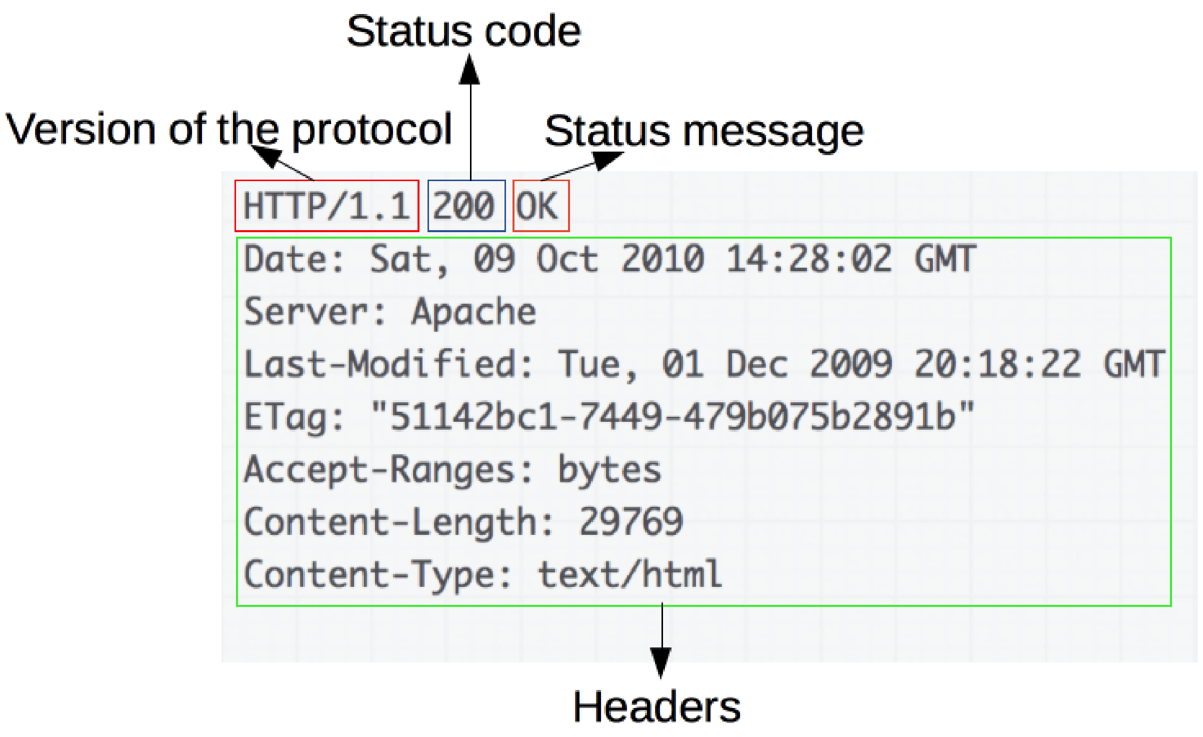
```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Requests consist of the following elements:

- An HTTP method, usually a verb like GET, POST, or a noun like OPTIONS or HEAD that defines the operation the client wants to perform. Typically, a client wants to fetch a resource (using GET) or post the value of an HTML form (using POST), though more operations may be needed in other cases.
- The path of the resource to fetch; the URL of the resource stripped from elements that are obvious from the context, for example without the protocol (http://), the domain (here, developer.mozilla.org), or the TCP port (here, 80).
- The version of the HTTP protocol.
- Optional headers that convey additional information for the servers.
- A body, for some methods like POST, similar to those in responses, which contain the resource sent.

Responses

An example response:



```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```

Responses consist of the following elements:

- The version of the HTTP protocol they follow.
- A status code, indicating if the request was successful or not, and why.
- A status message, a non-authoritative short description of the status code.
- HTTP headers, like those for requests.
- Optionally, a body containing the fetched resource.

APIs based on HTTP

The most commonly used API based on HTTP is the XMLHttpRequest API, which can be used to exchange data between a user agent and a server. The modern Fetch API provides the same features with a more powerful and flexible feature set.

Another API, server-sent events, is a one-way service that allows a server to send events to the client, using HTTP as a transport mechanism. Using the EventSource interface, the client opens a connection and establishes event handlers. The client browser automatically converts the messages that arrive on the HTTP stream into appropriate Event objects. Then it delivers them to the event handlers that have been registered for the events' type if known, or to the onmessage event handler if no type-specific event handler was established.

Conclusion

HTTP is an extensible protocol that is easy to use. The client-server structure, combined with the ability to add headers, allows HTTP to advance along with the extended capabilities of the Web.

Though HTTP/2 adds some complexity by embedding HTTP messages in frames to improve performance, the basic structure of messages has stayed the same since HTTP/1.0. Session flow remains simple, allowing it to be investigated and debugged with a simple

}

Additional Resources

Learn more Here is a list of resources that may be helpful as you continue your learning journey.

HTML Elements Reference (Mozilla)

https://developer.mozilla.org/en-US/docs/Web/HTML/Element

The Form Element (Mozilla)

https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form

What is the Document Object Model? (W3C)

https://www.w3.org/TR/WD-DOM/introduction.html

ARIA in HTML (W3C via Github)

https://w3c.github.io/html-aria/

ARIA Authoring Practices (W3C)

https://www.w3.org/TR/wai-aria-practices-1.2/

Additional resources

Learn more Here is a list of resources that may be helpful as you continue your learning journey.

CSS Reference (Mozilla)

https://developer.mozilla.org/en-US/docs/Web/CSS/Reference

HTML and CSS: Design and build websites by Jon Duckett

https://www.amazon.com/HTML-CSS-Design-Build-Websites/dp/1118008189/

CSS Definitive Guide by Eric Meyer

https://www.amazon.com/CSS-Definitive-Guide-Visual-Presentation/dp/1449393195/