

# Unit - I

## Introduction

### 1.1. Web Basics:

#### Internet:

The internet is a globally connected network system facilitating worldwide communication and access to data resources through a vast collection of private, public, business, academic and government networks.

The Internet is the global system of interconnected computer networks that uses the Internet protocol suite (TCP/IP) to communicate between networks and devices.

The terms internet and World Wide Web are often used interchangeably, but they are not exactly the same thing; *the internet refers to the global communication system, including hardware and infrastructure, while the web is one of the services communicated over the internet.*

#### Intranet:

An intranet is a private network contained within an enterprise that is used to securely share company information and computing resources among employees.

### **How do intranets, the internet and extranets differ?**

The internet, intranets and extranets are different types of networks with some similarities and overlapping aspects.

#### **Internet**

The internet works on a public network that anyone can access. There are no limits on who can access the internet, other than users must have access to a computing device that's connected to the internet. The public internet can have unlimited users at any one time, but it is more vulnerable to attackers than an intranet.

#### **Intranet**

An intranet works on a private network of computers. Only authorized people and systems can access it. They also must connect to the intranet via the required LAN or VPN. An intranet typically can host a specific number of users.

## Extranet

An extranet is an intranet that grants access to those outside of an organization to certain information and applications. Third parties such as customers, vendors and partners are given access to certain parts of the organization's intranet.

## What Does World Wide Web (WWW) Mean?

The World Wide Web (WWW) is a network of online content that is formatted in HTML and accessed via HTTP. The term refers to all the interlinked HTML pages that can be accessed over the Internet.

The World Wide Web is most often referred to simply as "the Web."

Tim Berners-Lee, a British scientist, invented the World Wide Web (WWW) in 1989, while working at CERN. The web was originally conceived and developed to meet the demand for automated information-sharing between scientists in universities and institutes around the world.

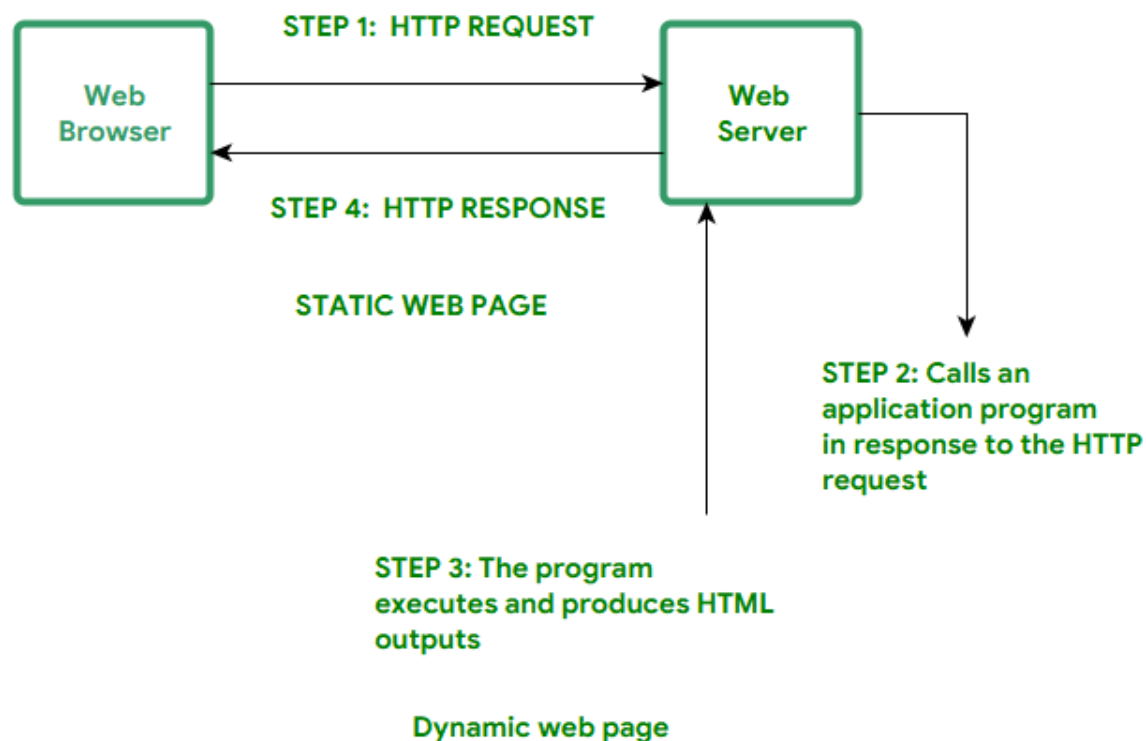
### Static and dynamic web pages:

*In the context of website creation, static means something that doesn't change, while dynamic signals something that does. A **static webpage** remains the same or fixed, in terms of the content it displays. A **dynamic webpage** is the opposite, its content changes according to the location of the user, or based on actions a user has made on the page before.*

**Static Web pages:** Static Web pages are very simple. It is written in languages such as HTML, JavaScript, CSS, etc. For static web pages when a server receives a request for a web page, then the server sends the response to the client without doing any additional process. And these web pages are seen through a web browser. In [static web pages](#), Pages will remain the same until someone changes it manually.



**Dynamic Web Pages:** Dynamic Web Pages are written in languages such as CGI, AJAX, ASP, ASP.NET, etc. In dynamic web pages, the Content of pages is different for different visitors. It takes more time to load than the static web page. [Dynamic web pages](#) are used where the information is changed frequently, for example, stock prices, weather information, etc.



### **What is the difference between web server and web client?**

A **web client** is an application that communicates with a **web server**, using Hypertext Transfer Protocol (HTTP). The basic objective of the web server is to store, process and deliver web pages to the users. Web browser is an example of a web client. Web servers speak the HTTP protocol, so they are often called HTTP servers. These HTTP servers store the Internet's data and provide the data when it is requested by HTTP clients.

### **Web Application Architecture:**

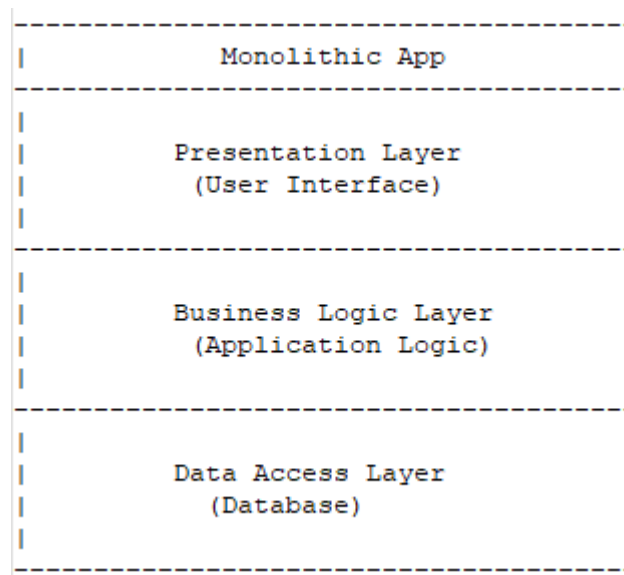
A web application is a software application that is accessed through a web browser over a network, typically the internet. It is designed to provide specific

functionalities and services to users, allowing them to interact with the application and perform tasks online.

*Web application architecture refers to the design and structure of a web application based on business requirement or use cases will follow any one of the application architecture types namely Single-tier Architecture(Monolithic Architecture), Two-tier Architecture (Client-Server Architecture), Three-tier Architecture, N-tier Architecture.*

➤ Single-tier Architecture(Monolithic Architecture):

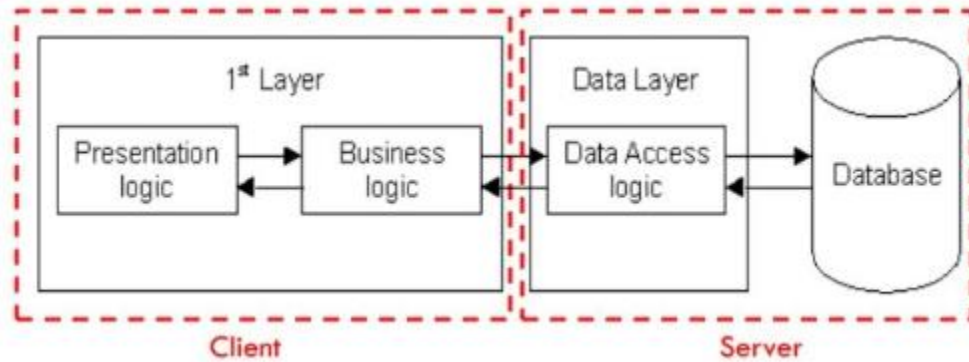
- In a single-tier architecture, the entire application runs on a single machine or server.
- The user interface, application logic, and data storage are all combined into a single unit.
- This architecture is typically used for small-scale applications with limited complexity.



➤ Two-tier Architecture (Client-Server Architecture):

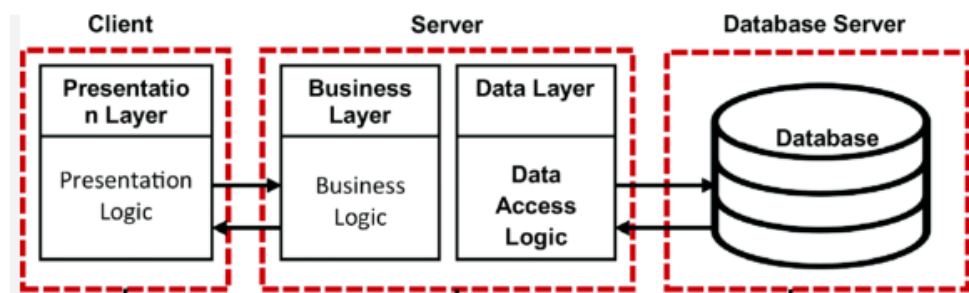
- In a two-tier architecture, the application is divided into two main tiers: client and server.
- The client tier is responsible for the user interface and user interactions.

- The server tier handles the application logic, data processing, and storage.
- Communication between the client and server occurs directly.



➤ Three-tier Architecture:

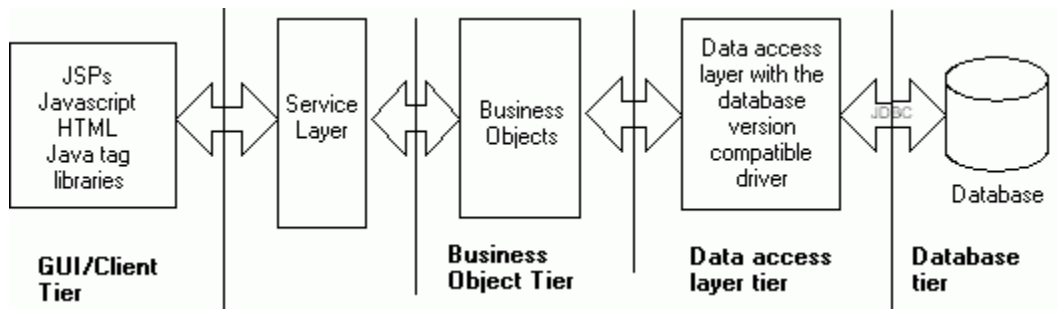
- The three-tier architecture adds an additional layer between the client and server tiers, known as the middle or application tier.
- The client tier handles the user interface.
- The application tier contains the application logic, processing user requests, and managing data.
- The data tier is responsible for data storage and retrieval, typically using a database.
- This architecture provides improved scalability, maintainability, and separation of concerns.



➤ N-tier Architecture:

- N-tier architecture extends the three-tier architecture by adding more layers or tiers as needed.

- Each additional tier represents a distinct set of responsibilities or services.
- Additional tiers can include caching layers, load balancers, message queues, micro services, etc.
- This architecture allows for better scalability, flexibility, and modularity, but also increases complexity.



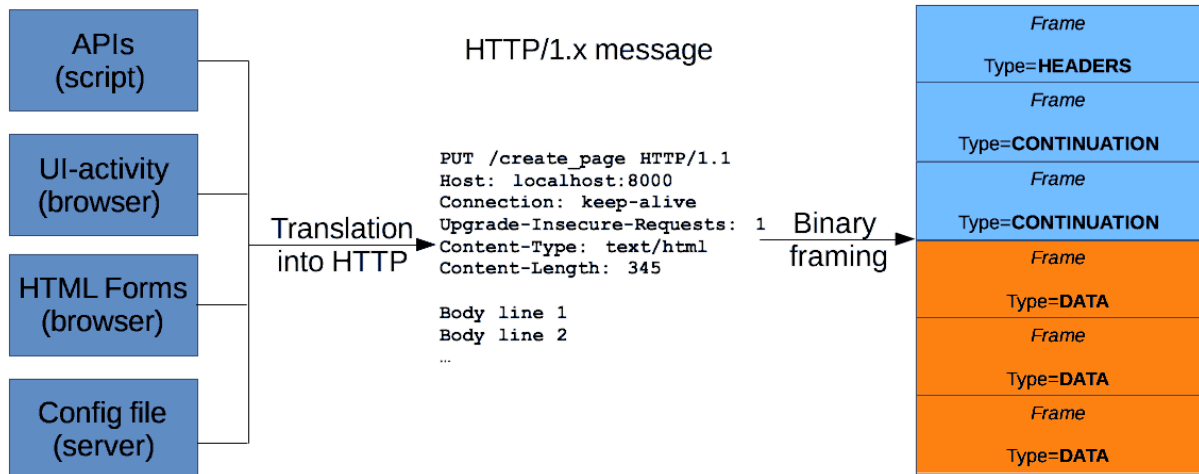
## Http Request and Response:

### HTTP Messages

HTTP messages are how data is exchanged between a server and a client. There are two types of messages: *requests* sent by the client to trigger an action on the server, and *responses*, the answer from the server.

Web developers, or webmasters, rarely craft these textual HTTP messages themselves: software, a Web browser, proxy, or Web server, perform this action. They provide HTTP messages through config files (for proxies or servers), APIs (for browsers), or other interfaces.

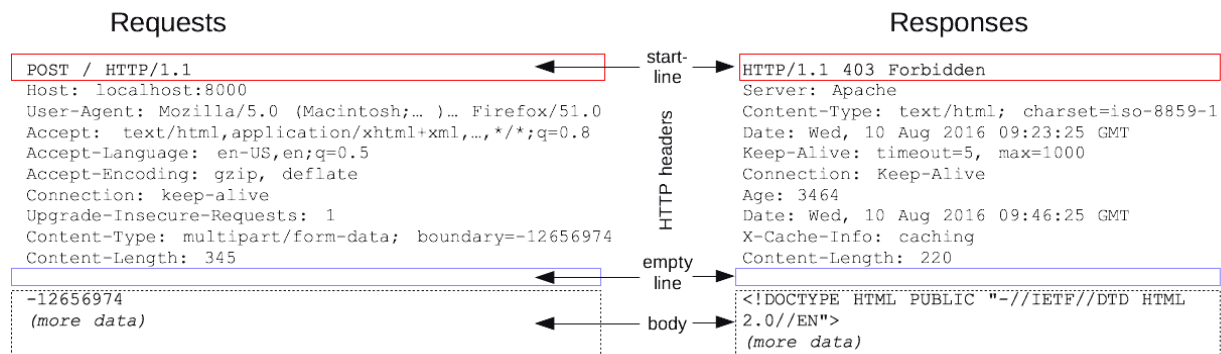
## Activity initiation



HTTP requests, and responses, share similar structure and are composed of:

1. A *start-line* describing the requests to be implemented, or its status of whether successful or a failure. This is always a single line.
2. An optional set of *HTTP headers* specifying the request, or describing the body included in the message.
3. A blank line indicating all meta-information for the request has been sent.
4. An optional *body* containing data associated with the request (like content of an HTML form), or the document associated with a response. The presence of the body and its size is specified by the start-line and HTTP headers.

The start-line and HTTP headers of the HTTP message are collectively known as the *head* of the requests, whereas its payload is known as the *body*.



## HTTP Requests

### Request line

Note: The start-line is called the "request-line" in requests.

HTTP requests are messages sent by the client to initiate an action on the server. Their *request-line* contain three elements:

1. An *HTTP method*, a verb (like [GET](#), [PUT](#) or [POST](#)) or a noun (like [HEAD](#) or [OPTIONS](#)), that describes the action to be performed. For example, GET indicates that a resource should be fetched or POST means that data is pushed to the server (creating or modifying a resource, or generating a temporary document to send back).
2. The *request target*, usually a [URL](#), or the absolute path of the protocol, port, and domain are usually characterized by the request context. The format of this request target varies between different HTTP methods. It can be
  - An absolute path, ultimately followed by a '?' and query string. This is the most common form, known as the *origin form*, and is used with GET, POST, HEAD, and OPTIONS methods.
    - POST / HTTP/1.1
    - GET /background.png HTTP/1.0
    - HEAD /test.html?query=alibaba HTTP/1.1
    - OPTIONS /anypage.html HTTP/1.0
  - A complete URL, known as the *absolute form*, is mostly used with GET when connected to a proxy. GET https://developer.mozilla.org/en-US/docs/Web/HTTP/Messages HTTP/1.1
  - The authority component of a URL, consisting of the domain name and optionally the port (prefixed by a ':'), is called the *authority form*. It is only used with CONNECT when setting up an HTTP tunnel. CONNECT developer.mozilla.org:80 HTTP/1.1
  - The *asterisk form*, a simple asterisk (\*) is used with OPTIONS, representing the server as a whole. OPTIONS \* HTTP/1.1



3. The *HTTP version*, which defines the structure of the remaining message, acting as an indicator of the expected version to use for the response.

## Headers

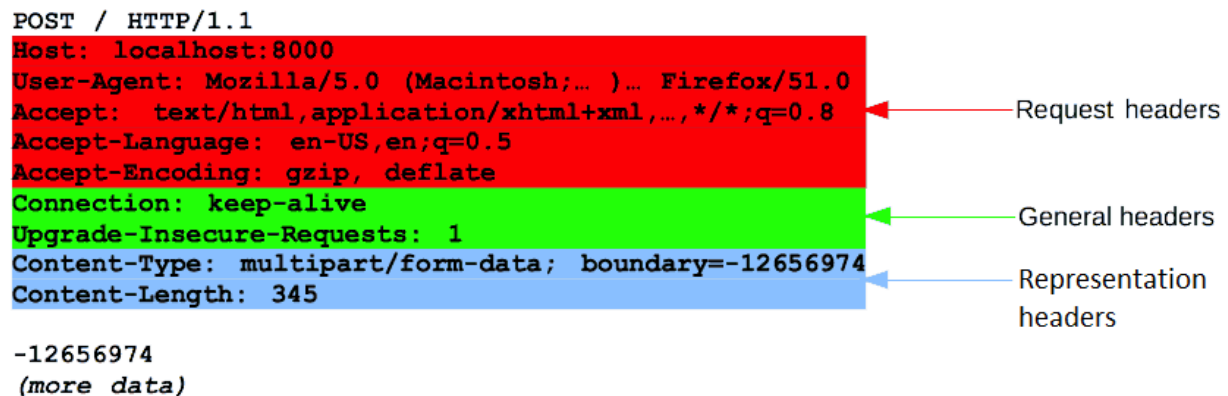
[HTTP headers](#) from a request follow the same basic structure of an HTTP header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the header. The whole header, including the value, consists of one single line, which can be quite long.

Many different headers can appear in requests. They can be divided in several groups:

- [General headers](#), like [Via](#), apply to the message as a whole.
- [Request headers](#), like [User-Agent](#) or [Accept](#), modify the request by specifying it further (like [Accept-Language](#)), by giving context (like [Referer](#)), or by conditionally restricting it (like [If-None-Match](#)).
- [Representation headers](#) like [Content-Type](#) that describe the original format of the message data and any encoding applied (only present if the message has a body).

```
POST / HTTP/1.1
Host: localhost:8000
User-Agent: Mozilla/5.0 (Macintosh;... )... Firefox/51.0
Accept: text/html,application/xhtml+xml,..., */*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Upgrade-Insecure-Requests: 1
Content-Type: multipart/form-data; boundary=-12656974
Content-Length: 345

-12656974
(more data)
```



The diagram illustrates the categorization of HTTP headers in a request. The headers are grouped into three categories, each highlighted with a different background color and labeled with an arrow:

- Request headers (Red background):** Includes `Host`, `User-Agent`, `Accept`, `Accept-Language`, and `Accept-Encoding`.
- General headers (Green background):** Includes `Connection` and `Upgrade-Insecure-Requests`.
- Representation headers (Blue background):** Includes `Content-Type` and `Content-Length`.

The body of the request is shown below the headers, starting with the boundary string `-12656974` and followed by `(more data)`.

## Body

The final part of the request is its body. Not all requests have one: requests fetching resources like GET or HEAD usually don't need a body. Requests that send data to the server to create a resource, such as PUT or POST requests, typically require a body with the data used to fulfill the request (for instance, HTML form data).

Bodies can be broadly divided into two categories:

- Single-resource bodies, consisting of one single file, defined by the two headers: [Content-Type](#) and [Content-Length](#).
- [Multiple-resource bodies](#), consisting of a multipart body, each containing a different bit of information. This is typically associated with [HTML Forms](#).

## **HTTP Responses**

### **Status line**

Note: The start-line is called the "status line" in responses.

The start line of an HTTP response, called the *status line*, contains the following information:

1. The *protocol version*, usually HTTP/1.1, but can also be HTTP/1.0.
2. A [status code](#), indicating success or failure of the request. Common status codes are [200](#), [404](#), or [302](#).
3. A *status text*. A brief, purely informational, textual description of the status code to help a human understand the HTTP message.

A typical status line looks like: HTTP/1.1 404 Not Found.

### **Headers**

[HTTP headers](#) for responses follow the same structure as any other header: a case-insensitive string followed by a colon (':') and a value whose structure depends upon the type of the header. The whole header, including its value, presents as a single line.

Many different headers can appear in responses. These can be divided into several groups:

- [General headers](#), like [Via](#), apply to the whole message.
- [Response headers](#), like [Vary](#) and [Accept-Ranges](#), give additional information about the server which doesn't fit in the status line.
- [Representation headers](#) like [Content-Type](#) that describe the original format of the message data and any encoding applied (only present if the message has a body).

```

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: Keep-Alive
Content-Encoding: gzip
Content-Type: text/html; charset=utf-8
Date: Wed, 10 Aug 2016 13:17:18 GMT
Etag: "d9b3b803e9a0dc6f22e2f20a3e90f69c41f6b71b"
Keep-Alive: timeout=5, max=999
Last-Modified: Wed, 10 Aug 2016 05:38:31 GMT
Server: Apache
Set-Cookie: csrftoken=.....
Transfer-Encoding: chunked
Vary: Cookie, Accept-Encoding
X-Frame-Options: DENY

```

(body)

## Body

The last part of a response is the body. Not all responses have one: responses with a status code that sufficiently answers the request without the need for corresponding payload (like [201 Created](#) or [204 No Content](#)) usually don't.

Bodies can be broadly divided into three categories:

- Single-resource bodies, consisting of a single file of known length, defined by the two headers: [Content-Type](#) and [Content-Length](#).
- Single-resource bodies, consisting of a single file of unknown length, encoded by chunks with [Transfer-Encoding](#) set to chunked.
- [Multiple-resource bodies](#), consisting of a multipart body, each containing a different section of information. These are relatively rare.

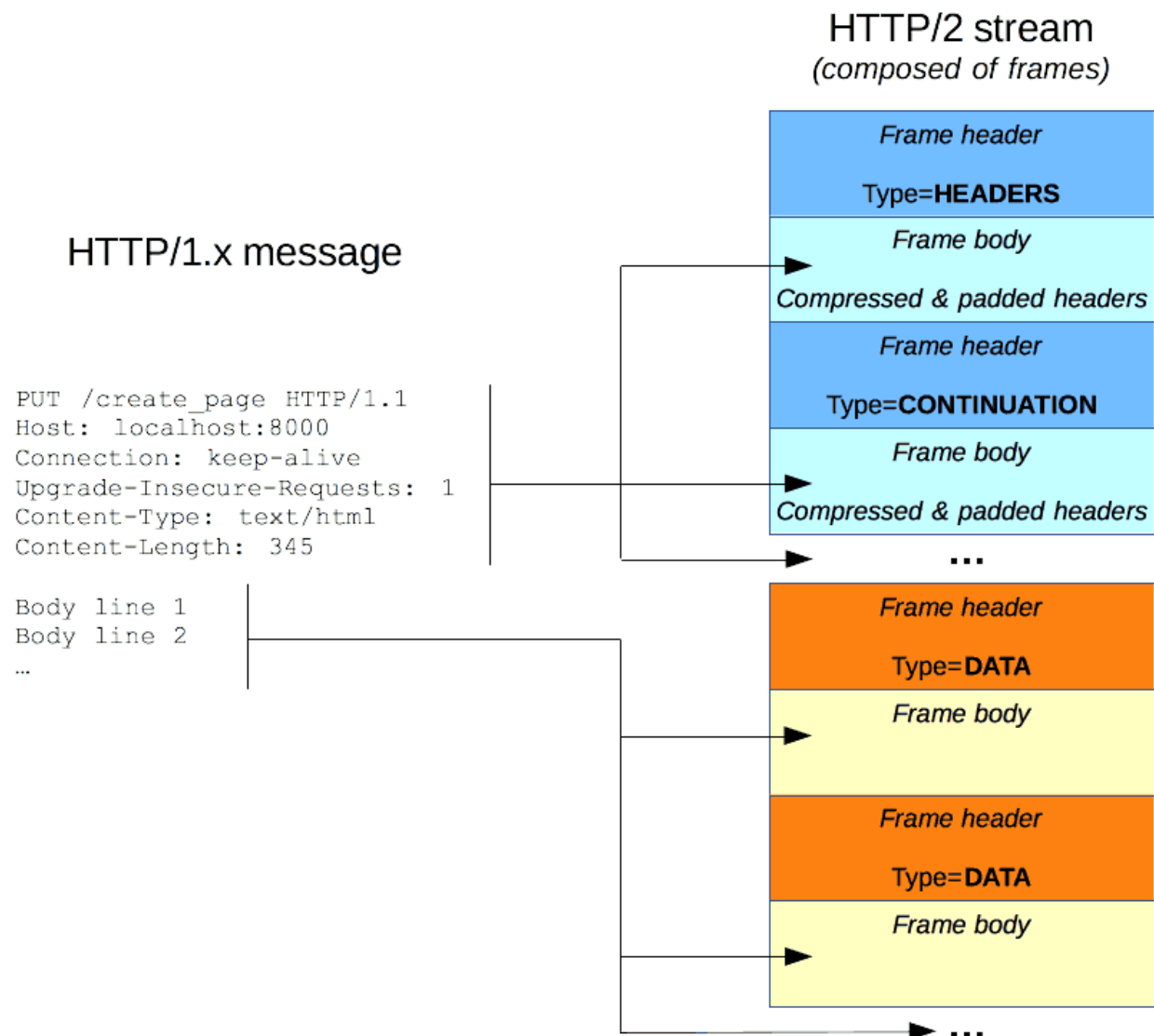
## HTTP/2 Frames

HTTP/1.x messages have a few drawbacks for performance:

- Headers, unlike bodies, are uncompressed.
- Headers are often very similar from one message to the next one, yet still repeated across connections.
- Although HTTP/1.1 has [pipelining](#), it's not activated by default in most browsers, and doesn't allow for multiplexing (i.e. sending requests concurrently). Several connections need opening on the same server to send

requests concurrently; and warm TCP connections are more efficient than cold ones.

HTTP/2 introduces an extra step: it divides HTTP/1.x messages into frames which are embedded in a stream. Data and header frames are separated, which allows header compression. Several streams can be combined together, a process called *multiplexing*, allowing more efficient use of underlying TCP connections.



HTTP frames are now transparent to Web developers. This is an additional step in HTTP/2, between HTTP/1.1 messages and the underlying transport protocol. No changes are needed in the APIs used by Web developers to utilize HTTP frames; when available in both the browser and the server, HTTP/2 is switched on and used.

## 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. Each valid URL points to a unique resource.

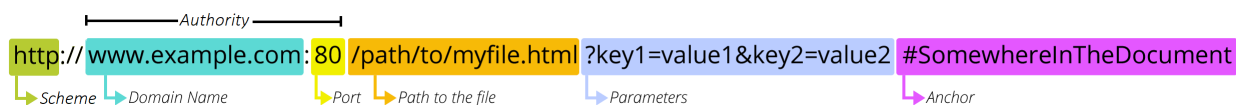
Examples of URLs:

`https://developer.mozilla.org`

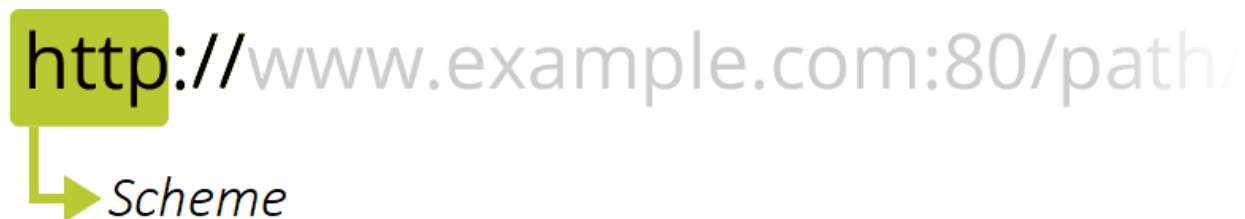
`https://developer.mozilla.org/en-US/docs/Learn/`

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

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



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 resource*

`/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`

→ *Parameters*

`?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

## lue2 #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.

### [Absolute URLs vs. relative URLs](#)(Self Study)

### **Client-Side vs. Server-Side Scripting:**

In web development, there are two primary methods of processing and delivering data: client-side and server-side.

**Client-side scripting** refers to the code that runs on the user's computer or device in a web browser. Client-side scripts are written in languages such as JavaScript, which is commonly used to manipulate and modify the user interface and interact with server-side scripts.

In contrast, **server-side scripting** refers to code that runs on the server, which receives client requests, processes them, and returns a response to the client. Server-side scripts are typically written in languages like PHP, Python, Ruby, or Java.

*The primary difference between client-side and server-side scripting is the location where the code runs. Client-side scripts are executed on the user's computer, while server-side scripts are executed on the server.*

**Client-side scripting** is used to improve the user experience by providing dynamic and interactive web pages. Some common examples of client-side scripting include form validation, AJAX, and dynamic page updates.

**Server-side scripting** is used to perform tasks that require data processing, database access, and user authentication. Server-side scripts can interact with the file system, databases, and other system resources that are not available to client-side scripts.

## **Difference between Web 1.0, Web 2.0, and Web 3.0**

Web1.0, Web2.0, and Web3.0 represent distinct stages in the evolution of the internet. Here are the key differences between these three versions:

### **Web 1.0:**

Web1.0, also known as the “Read-Only Web” or the “Static Web,” refers to the early days of the internet when it primarily served as an information repository. Key characteristics of Web1.0 include:

1. **One-Way Communication:** Web1.0 was mainly a one-way communication platform, where website owners provided content, and users consumed that content passively.
2. **Limited User Interaction:** Users had limited ability to interact with websites beyond basic browsing and clicking on links.
3. **Static Websites:** Websites in Web1.0 were predominantly static, with fixed content that rarely changed. Updates required manual editing of HTML code.
4. **Lack of Social Interaction:** Social networking and user-generated content were virtually non-existent in Web1.0. Websites focused on delivering information rather than facilitating user collaboration or participation.
5. **Absence of Dynamic Content:** Web1.0 lacked dynamic content, personalized experiences, and real-time updates.

### **Web 2.0:**

Web2.0, often referred to as the “Read-Write Web” or the “Social Web,” marks a significant shift in the internet’s evolution, emphasizing user-generated content and social interaction. Key characteristics of Web2.0 include:

1. **User-Generated Content:** Web2.0 platforms empowered users to create, share, and modify content, resulting in a significant increase in user participation and collaboration.



2. **Interactivity and Engagement:** Web2.0 enabled dynamic and interactive websites that allowed users to engage, comment, and interact with both content creators and other users.
3. **Social Media and Networking:** The rise of social media platforms like Facebook, Twitter, and YouTube defined Web2.0, enabling users to connect, share, and communicate with each other on a global scale.
4. **Rich Internet Applications (RIAs):** Web2.0 introduced the concept of RIAs, leveraging technologies like AJAX, JavaScript, and Flash to create more responsive and interactive web experiences.
5. **Personalization and Customization:** Web2.0 platforms offered personalized experiences, allowing users to customize their profiles, receive tailored content recommendations, and participate in collaborative filtering.

### **Web 3.0:**

Web3.0, also known as the “Decentralized Web” or the “Semantic Web,” represents the next phase of internet development, focused on decentralization, data ownership, and enhanced user control. Key characteristics of Web3.0 include:

1. **Decentralization and Blockchain Technology:** Web3.0 leverages decentralized networks, blockchain technology, and smart contracts to enable peer-to-peer interactions, secure transactions, and decentralized applications.
2. **User Control and Data Ownership:** Web3.0 emphasizes giving users greater control over their data, digital identities, and online interactions. It aims to shift power away from centralized entities and towards individual users.
3. **Enhanced Privacy and Security:** Web3.0 focuses on improving privacy and security by leveraging encryption, decentralized storage, and privacy-preserving technologies.
4. **Interoperability and Seamless Integration:** Web3.0 aims to establish interoperability among different blockchain networks and protocols, enabling seamless communication, data sharing, and value transfer across platforms.
5. **Intelligent Web and AI Integration:** Web3.0 envisions integrating artificial intelligence (AI) technologies, machine learning, and natural language

processing to enhance search capabilities, content curation, and user experiences.

## Simple Difference



**\*\*\*THE – END\*\*\***