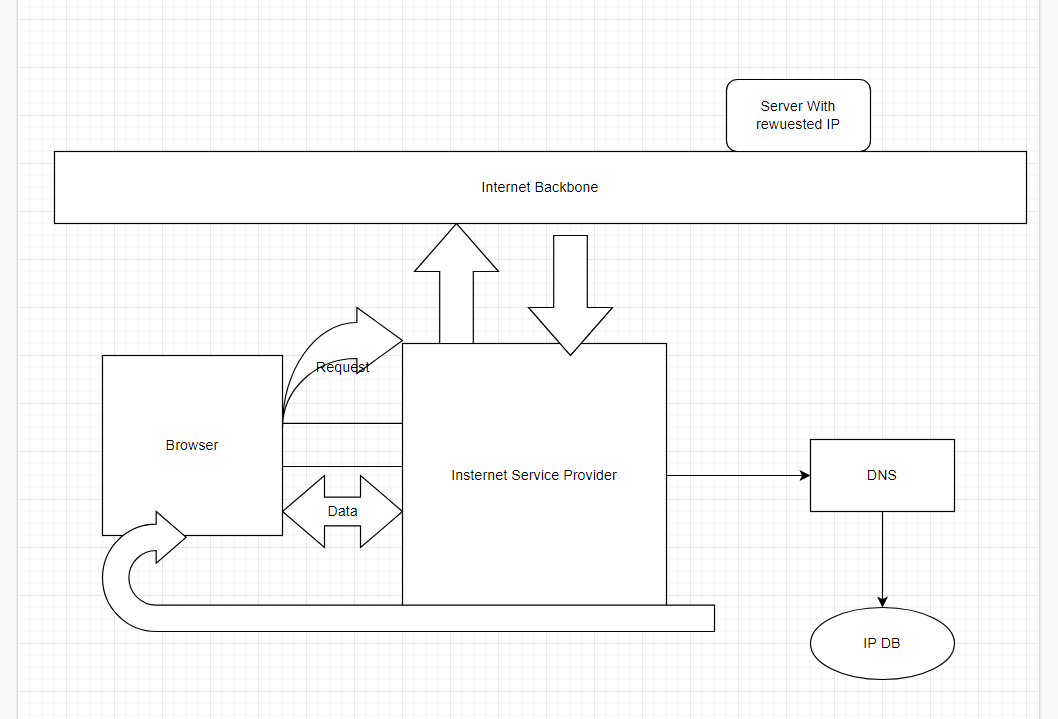
PESTO WEEK -1 Assignment

1)What happens when an url is entered in the browser?

Ans:



The Internet works with request/response mechanism.

1)Browser sends Http Request to via URL

2) The ISP(Internet Service Provider) Connects your browser to the internet where all the magic happens.

3)The URL request is Tranfered from browser to ISP to DNS where the ip address of the Server from which response is to be fetched is returned.

4)ISP goes to the Server IP fetches the required data and sends it back to browser.

5)Browser renders the response to the UI.

DETAILED EXPLANATION

The Steps taken by the browser

* look up the location of the server hosting the website
* make the connection to the server
* send a request to get the specific page
* handle the response from the server and
* render the page so you, the viewer, can interact with the website.

## **Websites, servers, IP addresses**

Websites are collections of files, often HTML, CSS, Javascript, and images, that tell your browser how to display the site, images, and data. They need to be accessible to anyone from anywhere at anytime, so hosting them on your computer at home isn’t be scalable or reliable. A powerful external computer connected to the Internet, called a server, stores these files.

When you point your browser at a URL like https://jennapederson.dev/hello-world, your browser has to figure out which server on the Internet is hosting the site. It does this by looking up the domain, jennapederson.dev, to find the address.

Each device on the Internet — servers, cell phones, your smart refrigerator — all have a unique address called an IP address. An IP address contains four numbered parts:

203.0.113.0

But numbers like this are hard to remember! That’s where domain names come in. pesto.com is much easier to remember than 203.0.113.0, right? Imagine having to remember all the phone numbers of your contacts without having the Contacts app on your phone. Your Contacts app lets you look up phone numbers by name.

We do the same on the Internet. The domain name system, or DNS, is like the Contacts app on our phone. DNS helps our browser (and us) find servers on the Internet. We can do a DNS lookup to find the IP address of the server based on the domain name, pesto.com.

## **The process**

### **1. You type https://pesto.com/hello-world in your browser and press Enter**

Let’s break down the parts of this URL you typed in to get here.

https://pesto.com/hello-world

#### **Scheme**

https:// is the scheme. HTTPS stands for Hypertext Transfer Protocol Secure. This scheme tells the browser to make a connection to the server using Transport Layer Security, or TLS. TLS is an encryption protocol to secure communications over the Internet. With HTTPS, the data exchanged between your browser and the server, like passwords or credit card info, is encrypted. You may have also seen ftp://, mailto://, or file://. These are other protocols that browsers know how to handle.

#### **Domain**

Pesto.com is the domain name of the site. It is the memorable address and points to a specific server’s IP.After DNS lookup a IP address for this domain is returned

#### **Path**

Sometimes there is an additional path to the resource in the URL. For example, for this URL,https://pesto,com/the-path-to/hello-world, the-path-to is the path on the server to the requested resource, hello-world. You can think of this like the directory structure of files and other directories on your computer. It’s a way to organize your resources, whether they are static HTML, CSS, Javascript, or image files, or dynamically generated content. Common examples of paths are blog, posts, tags, or images, each grouping different resources.

#### **Resource**

When you typed this URL into your browser, hello-world is the name of the resource on the website you want to view. Sometimes you’ll see this with a file extension like .html which indicates this is a static file on the server with HTML content. Without an extension, like this URL, it usually indicates the server generated this content. For instance, a news site would show you customized, up to date, and local news content, which it can only do when it knows who you are or where the request came from.

### **2. Browser looks up IP address for the domain**

After you’ve typed the URL into your browser and pressed enter, the browser needs to figure out which server on the Internet to connect to. To do that, it needs to look up the IP address of the server hosting the website using the domain you typed in. It does this using a DNS lookup

Because DNS is complex and has to be blazingly fast, DNS data is cached at different layers between your browser and at various places across the Internet. Your browser checks its own cache, the operating system cache, a local network cache at your router, and a DNS server cache on your corporate network or at your internet service provider (ISP). If the browser cannot find the IP address at any of those cache layers, the DNS server on your corporate network or at your ISP does a recursive DNS lookup. A recursive DNS lookup asks multiple DNS servers around the Internet, which in turn ask more DNS servers for the DNS record until it is found.

Once the browser gets the DNS record with the IP address, it’s time for it to find the server on the Internet and establish a connection.

### **3. Browser initiates TCP connection with the server**

Using the public Internet routing infrastructure, packets from a client browser request get routed through the router, the ISP, through an internet exchange to switch ISPs or networks, all using transmission control protocol, more commonly known as TCP, to find the server with the IP address to connect to. But this is a very roundabout way to get there and it’s not efficient.

Instead, many sites use a content delivery network, or CDN, to cache static and dynamic content closer to the browser.

A CDN is a globally distributed network of caching servers that improve the performance of your site or app (the origin) by bringing the content closer to your users

Once the browser finds the server on the Internet, it establishes a TCP connection with the server and if HTTPS is being used, a TLS handshake takes place to secure the communication

### **4. Browser sends the HTTP request to the server**

Now that the browser has a connection to the server, it follows the rules of communication for the HTTP(s) protocol. It starts with the browser sending an HTTP request to the server to request the contents of the page. The HTTP request contains a request line, headers (or metadata about the request), and a body. The request line contains information that the server can use to determine what the client (in this case, your browser) wants to do.

The request line contains the following:

* a request method, which is one of GET, POST, PUT, PATCH, DELETE, or a handful of other HTTP verbs
* the path, pointing to the requested resources
* the HTTP version to communicate with

The request line for the URL request looks like this:

GET /hello-world HTTP/1.1

The request line tells the server that you want to GET resource at /hello-world and to communicate with HTTP/1.1.

Remember that HTTP verbs express the semantic intent of your request. You could also use the POST, PUT, or PATCH methods to send data to the server for storage, either to create new data or update existing data at the request path. The DELETE method would delete the resource at the given path. However, it’s important to know that servers don’t have to support all verbs. A server could respond with a 200 OK status to a DELETE method and not do anything with the resource.

The next part of the request is the request headers. Headers pass extra information along from the client that help route the request, indicate what type of client is making the request (the user agent).

Headers are key-value pairs like this:

Host: pesto.com

User-Agent: curl/7.64.1

Accept: \*/\*

The last part of the request is the body. The body is (usually) empty for a GET request like ours. For a request that manipulates resources, like POST, PUT, or PATCH, the body will contain the data from the client to create or update.

The request body can be in different formats and the server understands the format based on a request header, Content-Type.

### **5. Server processes request and sends back a response**

The server takes the request and based on the info in the request line, headers, and body, decides how to process the request. For the request, GET /hello-world/ HTTP/1.1, the server gets the content at this path, constructs the response and sends it back to the client. The response contains the following:

* a status line, telling the client the status of the request
* response headers, telling the browser how to handle the response
* the requested resource at that path, either content like HTML, CSS, Javascript, or image files, or data

The browser considers a status code in the 200s to be successful. The response was 200, so the browser knows to render the response.

Resources can be static files, either text (i.e index.html) or non-text data (i.e. logo.img). Browsers aren’t always requesting static files, though. Often, these are dynamic resources generated at the time of the request and there is no file associated with the resource. In fact, in this request, that’s exactly what is happening. There is no file hello-world, but the server still knows how to process the request. The server generates a dynamic resource, building the HTML from fragments or templates and combining it with dynamic data to send back in the response, as text, so the browser can render the page.

### **6. Browser renders the content**

Once the browser has received the response from the server, it inspects the response headers for information on how to render the resource. The Content-Type header above tells the browser it received an HTML resource in the response body. Lucky for us, the browser knows what to do with HTML!

The first GET request returns HTML, the structure of the page. But if you inspect the HTML of the page (or any web page) with your browser’s dev tools, you’ll see it references other Javascript, CSS, image resources and requests additional data in order to render the web page as designed.

As the browser is parsing and rendering the HTML, it is making additional requests to get Javascript, CSS, images, and data. It can do much of this in parallel.

Summary

1. You type a URL in your browser and press Enter
2. Browser looks up IP address for the domain
3. Browser initiates TCP connection with the server
4. Browser sends the HTTP request to the server
5. Server processes request and sends back a response
6. Browser renders the content.

A) What is the main functionality of the browser?

The main function of a browser is to present the web resource you choose, by requesting it from the server and displaying it in the browser window. The resource is usually an HTML document, but may also be a PDF, image, or some other type of content. The location of the resource is specified by the user using a URI (Uniform Resource Identifier).

The way the browser interprets and displays HTML files is specified in the HTML and CSS specifications. These specifications are maintained by the W3C (World Wide Web Consortium) organization, which is the standards organization for the web. For years browsers conformed to only a part of the specifications and developed their own extensions. That caused serious compatibility issues for web authors. Today most of the browsers more or less conform to the specifications.

Browser user interfaces have a lot in common with each other. Among the common user interface elements are:

Address bar for inserting a URI

Back and forward buttons

Bookmarking options

Refresh and stop buttons for refreshing or stopping the loading of current documents

Home button that takes you to your home page

B)HIGH LEVEL COMPONENTS OF BROWSER

The browser's HIGH LEVEL COMPONENTS components are:

The user interface: this includes the address bar, back/forward button, bookmarking menu, etc. Every part of the browser display except the window where you see the requested page.

**The browser engine**: Assembles and arranges actions between the UI and the rendering engine.It provides methods to initiate the loading of a URL and other actions like (reload, back and forward).

**The rendering engine**: responsible for displaying requested content. For example if the requested content is HTML, the rendering engine parses HTML and CSS, and displays the parsed content on the screen.

**Networking**:

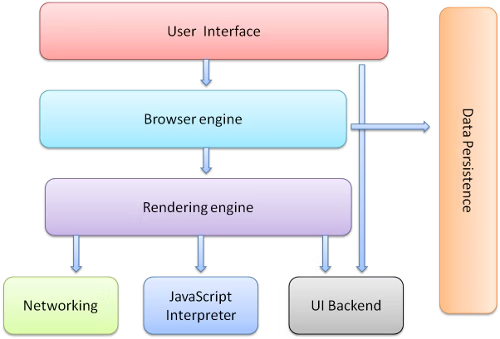
1. Networking handle all aspects of Internet Communication and handles URLs to use HTTP, FTP.

2. Implements a cache of retrieve documents to minimize network traffic.

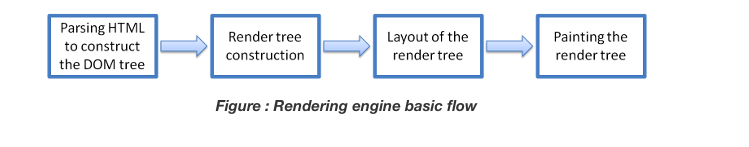
**UI backend**: used for drawing basic widgets like select box,input box, check boxes,combo boxes and windows. This backend exposes a generic interface that is not platform specific.

**JavaScript interpreter**: Used to parse and execute JavaScript code.

**Data storage**:This is a persistence layer. The browser may need to save all sorts of data locally, such as cookies.



**C)Rendering Engine and its uses**



It’s able to render the content of given URL in browser screen and interprets the HTML, XML and CSS. It is single threaded. By default, It displays data according to your specified content type (MIME). For Example HTML, Images, XML, CSS, JSON, PDF etc.

Key operation of Rendering engine is HTML Parser. Each browser use various engines like Chrome and Opera uses Blink, Firefox uses Gecko, IE Edge uses EdgeHTML, Internet Explorer uses Trident, Apple Safari uses WebKit.

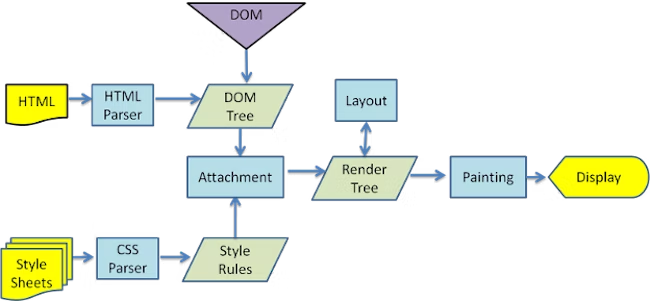
Process

1. Parsing HTML document by HTML Parser convert elements to Node and create Content Tree.

2. Parsing Styles code / document by CSS Parser and create Render Tree.

3. Render Tree goes through Layout Process. Element’s Node get position coordinates.

4. Render Tree will be traversed and each node will be painted using the UI Back-end Layer.



**D)PARSERS(HTML,CSS)**

Parsing a document means translating it to a structure the code can use. The result of parsing is usually a tree of nodes that represent the structure of the document. This is called a parse tree or a syntax tree.

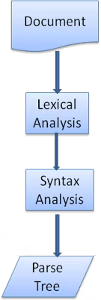
Parsing is based on the syntax rules the document obeys: the language or format it was written in. Every format you can parse must have deterministic grammar consisting of vocabulary and syntax rules. It is called a context free grammar.

Parser - Lexer combination #

Parsing can be separated into two sub processes: lexical analysis and syntax analysis.

Lexical analysis is the process of breaking the input into tokens. Tokens are the language vocabulary: the collection of valid building blocks. In human language it will consist of all the words that appear in the dictionary for that language.

Syntax analysis is the applying of the language syntax rules.



**HTML Parser**

The job of the HTML parser is to parse the HTML markup into a parse tree.

The HTML grammar definition

The vocabulary and syntax of HTML are defined in specifications created by the W3C organization.

Not a context free grammar

As we have seen in the parsing introduction, grammar syntax can be defined formally using formats like BNF.

There is a formal format for defining HTML - DTD (Document Type Definition) - but it is not a context free grammar.

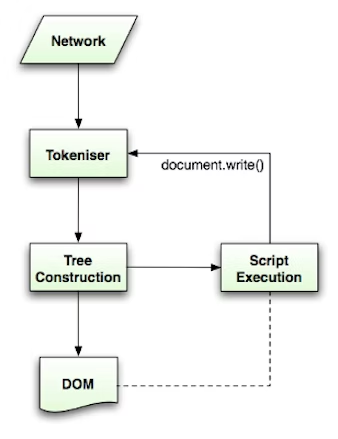
As we saw in the previous sections, HTML cannot be parsed using the regular top down or bottom up parsers.

The reasons are:

The forgiving nature of the language.

The fact that browsers have traditional error tolerance to support well known cases of invalid HTML.

The parsing process is reptativet. For other languages, the source doesn't change during parsing, but in HTML, dynamic code (such as script elements containing document.write() calls) can add extra tokens, so the parsing process actually modifies the input.



**CSS PARSING**

Well, unlike HTML, CSS is aThis means a ruleset is a selector or optionally a number of selectors separated by a comma and spaces (S stands for white space). A ruleset contains curly braces and inside them a declaration or optionally a number of declarations separated by a semicolon. "declaration" and "selector" will be defined in the following BNF definitions. context free grammar and can be parsed using the types of parsers.

**ruleset**

**: selector [ ',' S\* selector ]\***

**'{' S\* declaration [ ';' S\* declaration ]\* '}' S\***

**;**

**E)SCRIPT PROCESSORS:(NOT SURE ON THIS ONE)**

At this stage the browser will mark the document as interactive and start parsing scripts that are in "deferred" mode: those that should be executed after the document is parsed. The document state will be then set to "complete" and a "load" event will be fired.

**F)TREE CONSTRUCTION**

The CSSOM and DOM trees are combined into a render tree, which is then used to compute the layout of each visible element and serves as an input to the paint process that renders the pixels to screen.

we built the DOM and the CSSOM trees based on the HTML and CSS input.

However, both of these are independent objects that capture different aspects of the document: one describes the content, and the other describes the style rules that need to be applied to the document.

To construct the render tree, the browser roughly does the following:

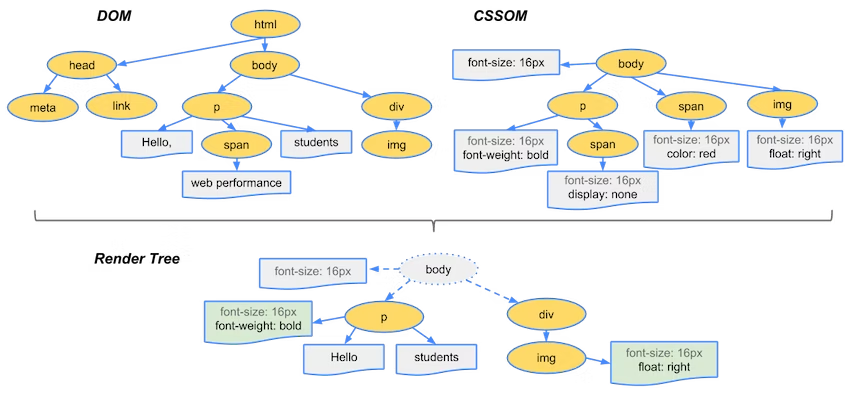
1)Starting at the root of the DOM tree, traverse each visible node.

2)Some nodes are not visible (for example, script tags, meta tags, and so on), and are omitted since they are not reflected in the rendered output.

Some nodes are hidden via CSS and are also omitted from the render tree; for example, the span node---in the example above---is missing from the render tree because we have an explicit rule that sets the "display: none" property on it.

For each visible node, find the appropriate matching CSSOM rules and apply them.

3)Emit visible nodes with content and their computed styles



**G) ORDER OF SCRIPT PROCESSING**

The model of the web is synchronous. Authors expect scripts to be parsed and executed immediately when the parser reaches a <script> tag. The parsing of the document halts until the script has been executed. If the script is external then the resource must first be fetched from the network - this is also done synchronously, and parsing halts until the resource is fetched. This was the model for many years and is also specified in HTML4 and 5 specifications. Authors can add the "defer" attribute to a script, in which case it will not halt document parsing and will execute after the document is parsed. HTML5 adds an option to mark the script as asynchronous so it will be parsed and executed by a different thread.

**H) LAYOUT AND PAINTING**

**LAYOUT**

The final output is a render tree that contains both the content and style information of all the visible content on the screen is proceed to the "layout" stage**.**

we've calculated which nodes should be visible and their computed styles, but we have not calculated their exact position and size within the viewport of the device---that's the "layout" stage.

To figure out the exact size and position of each object on the page, the browser begins at the root of the render tree and traverses it.

The output of the layout process is a "box model," which precisely captures the exact position and size of each element within the viewport: all of the relative measurements are converted to absolute pixels on the screen.

**PAINTING**

It is the last stage of Renderig.

Finally, now that we know which nodes are visible, and their computed styles and geometry, we can pass this information to the final stage, which converts each node in the render tree to actual pixels on the screen. This step is often referred to as "painting".

When layout is complete, the browser issues "Paint Setup" and "Paint" events, which convert the render tree to pixels on the screen.