**When a user enters an URL in the browser, how does the browser fetch the desired result ? Explain this with the below in mind and Demonstrate this by drawing a diagram for the same.**

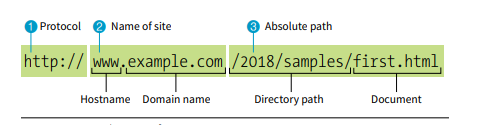
The Web is just one of the ways information can be shared over the internet. It allows documents to be linked to one another via hyper-text link— thus forming a hurge “web” of connected information. The web uses protocol called HTTP (HyperText Transfer Protocol)

(Standardized ways to transfer data and documents over the network are known as protocol.)

Every computer or device connected to the internet is assigned to a unique numeric IP address (“IP” stands for “Internet Protocol”). For example, as I write this, the computer that hosts example.com has the IP address 199.23.345.23. All those numbers can be dizzying, so fortunately, the domain Name System (DNS) was developed to allow us to refer to that server by its domain name, “example.com”, as well. DNS(Domain Name System) is a database that maintains the name of the website (URL) and the particular IP address it links to. Every single URL on the internet has a unique IP address assigned to it.The ID address is accessible to computer software, while the domain name is more accessible to humans. Matching the domain name with their respective IP address is the job of a separate DNS server. If you think of an IP address as a telephone number, the DNS server would be the phone book.

**The Parts of a URL**

Every page and resource on the web has its own special address called a URL, which stands for Uniform Resource Locator. It’s nearly impossible to get through a day without seeing a URL ) plastered on the side of a bus, printed on a business card, or broadcast on a television commercial. Web addresses are fully integrated into modern vernacular.



HTTP protocol to connect to a web server on the internet called “www.example.com” and to request the document first.html, located in the samples directory, which is in the 2018 directory

Many addresses do not include a filename, but simply point to a directory, like these:

http://www.example.com/

When a server receives a request for a directory name rather than a specific file, it looks in that directory for a default document, typically named first.html. So when someone types the previous URLs into his browser, what he’ll actually see is this:

http://www.example.com/index.html

The name of the default file (also referred to as the index file) may vary, and depends on how the server is configured. In these examples, it is named index. html, but some servers use the filename default.htm. If your site uses server side programming to generate pages, the index file might be named index.php or Default.aspx. Just check with your server administrator or the tech support department at your hosting service to make sure you give your default file the proper name

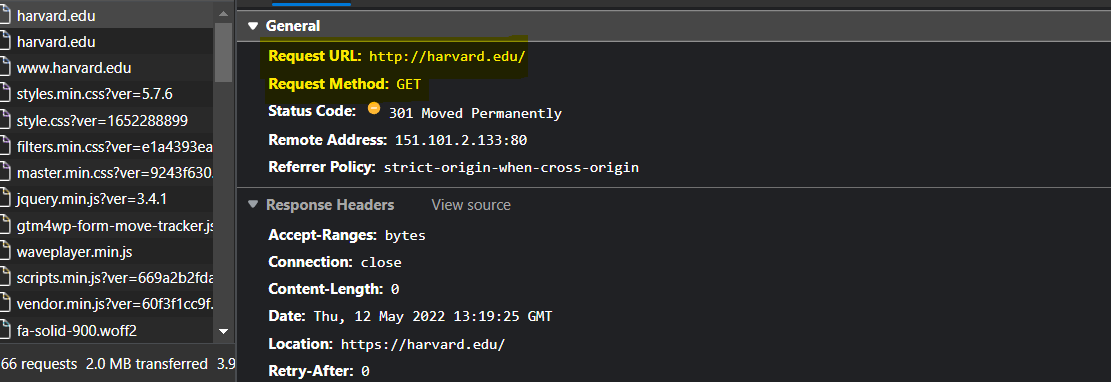
**What is the main functionality of the browser?**

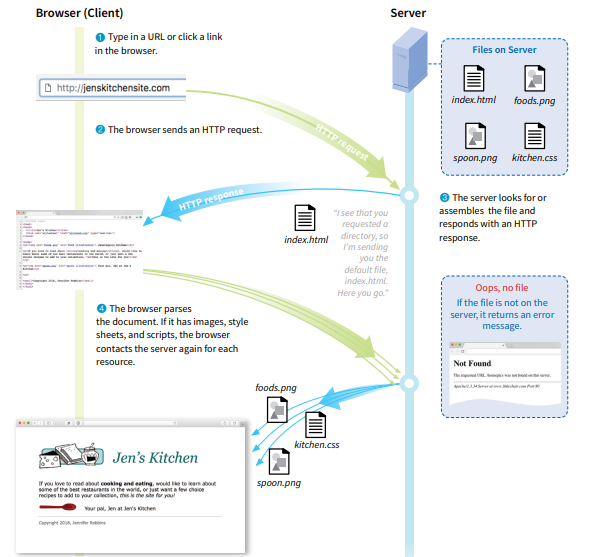
Web browser function starts with a user entering the desired URL(Uniform Resource Locator) into the address bar of the browser. For example https://www.googledotcom. Depending on the URL prefix, the web browser interprets the URL and displays the content to the user. The URL prefix provides the protocol used to access the location. HTTP:// and HTTPS:// are the most commonly used URL prefixes.

Once the resource has been located and interpreted the browser will display the content to the user. The browser application converts content such as HTML, CSS, and images to an interactive file in a process called rendering. Browsers can interpret and display content such as video, images, text, hyperlinks, and XML files.

**The browser sends an HTTP request to the webserver.**

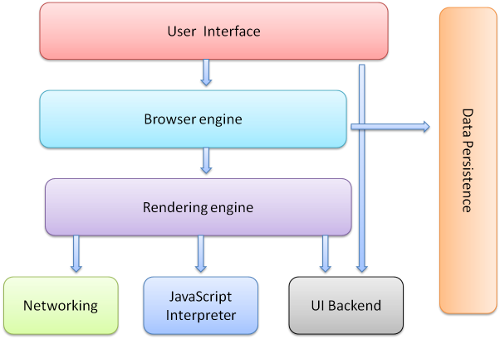
Once the TCP connection is established, it is time to start transferring data! The browser will send a GET request asking for maps.google.com web page. If you’re entering credentials or submitting a form, this could be a POST request. This request will also contain additional information such as browser identification (*User-Agent* header), types of requests that it will accept (*Accept* header), and connection headers asking it to keep the TCP connection alive for additional requests. It will also pass information taken from cookies the browser has in store for this domain.



 **High Level Components of a browser.**

Primary components of a browser are

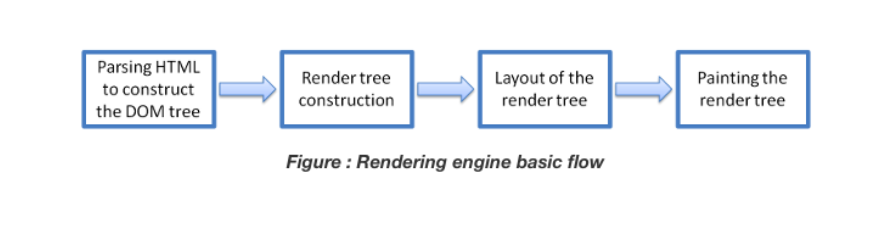
1. **User Interface** – This consists of forward and back buttons, bookmarks, address bar etc. along with the window that displays the requested page.
2. **Browser engine** – It commands action between the rendering engine and the user interface.
3. **Rendering engine** – The main function of the rendering engine is to display the content that is requested. For example, if an HTML content is requested, the engine parses CSS and HTML and when the content is parsed, it is displayed on the screen.
4. **Networking:** This component is responsible for managing network calls using standard protocols like HTTP or FTP. It also looks after security issues associated with internet communication.
5. **JavaScript Interpreter:** As the name suggests, it is responsible for parsing and executing the JavaScript code embedded in a website. Once the interpreted results are generated, they are forwarded to the rendering engine for displaying on the user interface.
6. **UI Backend:** This component uses the user interface methods of the underlying operating system. It is mainly used for drawing basic widgets (windows and combo boxes).
7. **Data Storage/Persistence:** It is a persistent layer. A web browser needs to store various types of data locally, for example, cookies. As a result, browsers must be compatible with data storage mechanisms such as WebSQL, IndexedDB, FileSystem, etc.



**Rendering engine and its use.**

Once a user requests a particular document, the rendering engine starts fetching the content of the requested document. This is done via the networking layer. The rendering engine starts receiving the content of that specific document in chunks of 8 KBs from the networking layer. After this, the basic flow of the rendering engine begins.

The four basic steps include:



The four basic Steps include.

1. The requested HTML page is parsed in chunks, including the external CSS files and in style elements, by the rendering engine. The HTML elements are then converted into DOM nodes to form a “**content tree**” or “**DOM tree.**”
2. Simultaneously, the browser also creates a **render tree**. This tree includes both the styling information as well as the visual instructions that define the order in which the elements will be displayed. The render tree ensures that the content is displayed in the desired order.
3. Further, the render tree goes through the **layout process**. When a render tree is created, the position or size values are not assigned. The entire process of calculating values for evaluating the desired position is called a layout process. In this process, every node is assigned the exact coordinates. This ensures that every node appears at an accurate position on the screen.
4. The final step is to paint the screen, wherein the render tree is traversed, and the renderer’s **paint()** method is invoked, which paints each node on the screen using the UI backend layer.

**Parsers:**

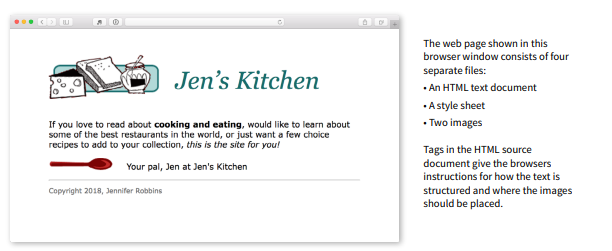
Parsing means analyzing and converting a program into an internal format that a runtime environment can run, for example, the JavaScript engine inside browsers.

The browser parses HTML into a DOM tree, HTML parsing involves **tokenization** and **tree construction**. If the document is well-formed, parsing it is straightforward and faster.

When the HTML parser finds non-blocking resources, such as an image, the browser will request those and continue parsing but <script> tags--particularly those without an async or defer attribute—block rendering, and pauses parsing of HTML.

CSS Parsing

When the browser encounters CSS styles, it parses the text into the CSS Object Model (CSSOM), a data structure. It is then used for styling layouts and painting. The browser then creates a render tree from both these structures to be able to paint the content to the screen.



**Script Processor**:

When a user enters an URL in the browser, how does the browser fetch the desired

result ? Explain this with the below in mind and Demonstrate this by drawing a diagram

for the same

**Tree Construction:**

The first step is processing the HTML and building DOM(Document Object Model). The DOM tree describes the content of the document. The <html> element is the first tag and root node of the document tree. The tree reflects the relationship and hierarchies between different tags. Tags nested within other tags are child nodes. The greater the number of DOM nodes, the longer it takes to construct the DOM tree.

**Preload Scanner:**

While the browser builds the DOM tree, this process occupies the main thread. As this happens, the preload scanner will parse through the content available and also request high-priority resources like CSS, JavaScript, and web fonts. It will retrieve the resources in the background so that by the time the main HTML parser reaches the requested asset, they may already be in flight, or have been downloaded. The optimizations the preload scanner provides reduce blockages.

**Building the CSSOM:**

The second step is processing CSS and building a CSSOM tree. The CSS object model is similar to DOM. The DOM and CSSOM are both trees. They are independent data structures. The browser converts the CSS rules into a map of styles it can understand and work with. The browser goes through each rule set in the CSS, creating a tree of nodes with a parent, child, and sibling relationships based on the CSS selectors

Building the CSSOM is very, very fast and is not displayed in a unique color in current developer tools. In terms of web performance optimization, there is lower hanging fruit, as the total time to create the CSSOM is generally less than the time it takes for one DNS lookup.

**Java script Compilation:**

While the CSS is being parsed and the CSSOM created, other assets, including JavaScript files, are downloading (thanks to the preload scanner). JavaScript is interpreted, compiled, parsed, and executed. The scripts are parsed into abstract syntax trees. Some browser engines take the Abstract Syntax Tree and pass it into an interpreter, outputting bytecode which is executed on the main thread. This is known as JavaScript compilation

**Building the Accessibility Tree:**

While the CSS is being parsed and the CSSOM created, other assets, including JavaScript files, are downloading (thanks to the preload scanner). JavaScript is interpreted, compiled, parsed, and executed. The scripts are parsed into abstract syntax trees. Some browser engines take the Abstract Syntax Tree and pass it into an interpreter, outputting bytecode which is executed on the main thread. This is known as JavaScript compilation,

Until the AOM is built, the content is not accessible to screen readers.

**Layout:**

The fourth step in the critical rendering path is running the layout on the render tree to compute the geometry of each node. The layout is the process by which the width, height, and location of all the nodes in the render tree are determined, plus the determination of the size and position of each object on the page. *Reflow* is any subsequent size and position determination of any part of the page or the entire document.

Once the render tree is built, the layout commences. The render tree identified which nodes are displayed (even if invisible) along with their computed styles, but not the dimensions or location of each node. To determine the exact size and location of each object, the browser starts at the root of the render tree and traverses it.

On the web page, almost everything is in a box. Different devices and different desktop preferences mean an unlimited number of different viewport sizes. In this phase, considering the viewport size, the browser determines what the dimensions of all the different boxes are going to be on the screen. Taking the size of the viewport as its base, the layout generally starts with the body, laying out the dimensions of all the body's descendants, with each element's box model properties.

The first time the size and position of nodes are determined is called *layout*. Subsequent recalculations of node size and locations are called *reflows*.

**Painting:**

The last step is to paint individual nodes on the screen, the first occurrence of which is called first meaningful paint. In the painting or rasterization phase, the browser converts each box calculated in the layout phase to actual pixels on the screen. Painting involves drawing every visual part of an element to the screen, including text, colors, borders, shadows, and replaced elements like buttons and images. The browser needs to do this super quickly.

Painting can break the elements in the layout tree into layers. Promoting content into layers on the GPU (instead of the main thread on the CPU) improves paint and repaint performance. Some specific properties and elements instantiate a layer, including <video> and <canvas>, and any element which has the CSS properties of opacity, a 3D transform, will-change, and a few others. These nodes will be painted onto their layer, along with their descendants, unless a descendant necessitates its layer for one (or more) of the above reasons.

Layers do improve performance but are expensive when it comes to memory management, so should not be overused as part of web performance optimization strategies