**The browser's main functionality**:

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.

**The browser's high level structure:**

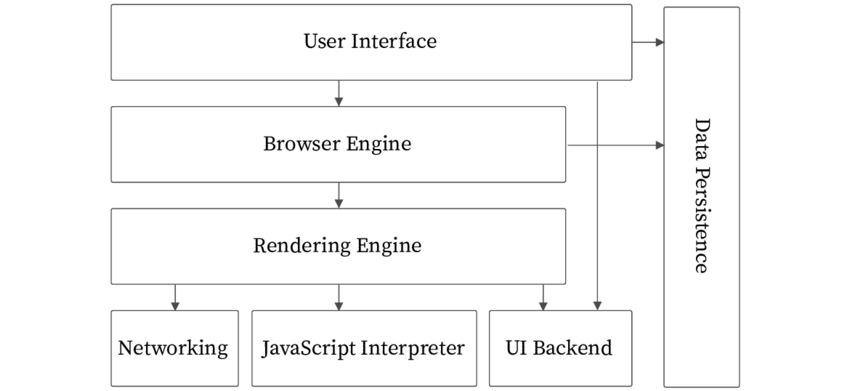
****

Fig.: Browser’s High Level Components

The browser's main components are:

1. *The User Interface:* The user interface is the space where User interacts with the browser. It includes the address bar, back and next buttons, home button, refresh and stop, bookmark option, etc. Every other part, except the window where requested web page is displayed, comes under it.
2. *The Browser Engine:* The browser engine works as a bridge between the User interface and the rendering engine. According to the inputs from various user interfaces, it queries and manipulates the rendering engine.
3. *The Rendering Engine:* The rendering engine, as the name suggests is responsible for rendering the requested web page on the browser screen. The rendering engine interprets the HTML, XML documents and images that are formatted using CSS and generates the layout that is displayed in the User Interface. However, using plugins or extensions, it can display other types data also. Different browsers uses different rendering engines:
   * Internet Explorer: Trident
   * Firefox & other Mozilla browsers: Gecko
   * Chrome & Opera 15+: Blink
   * Chrome (iPhone) & Safari: Webkit
4. *Networking:* Component of the browser which retrieves the URLs using the common internet protocols of HTTP or FTP. The networking component handles all aspects of Internet communication and security. The network component may implement a cache of retrieved documents in order to reduce network traffic.
5. *JavaScript Interpreter:* It is the component of the browser which interprets and executes the JavaScript code embedded in a website. The interpreted results are sent to the rendering engine for display. If the script is external then first the resource is fetched from the network. Parser keeps on hold until the script is executed.
6. *UI Backend:* UI backend is used for drawing basic widgets like combo boxes and windows. This backend exposes a generic interface that is not platform specific. It underneath uses operating system user interface methods.
7. *Data Persistence/Storage:* This is a persistence layer. Browsers support storage mechanisms such as localStorage, IndexedDB, WebSQL and Filesystem. It is a small database created on the local drive of the computer where the browser is installed. It manages user data such as cache, cookies, bookmarks and preferences.

**Rendering engine:**

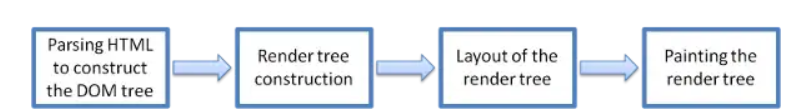
****

Fig.: Rendering engine basic flow.

The rendering engine parses the chunks of HTML document and convert the elements to DOM nodes in a tree called the “content tree” or the “DOM tree”. It also parses both the external CSS files as well in style elements.

While the DOM tree is being constructed, the browser constructs another tree, the render tree. This tree is of visual elements in the order in which they will be displayed. It is the visual representation of the document. The purpose of this tree is to enable painting the contents in their correct order. Firefox calls the elements in the render tree “frames”. WebKit uses the term renderer or render object.

After the construction of the render tree, it goes through a “layout process” of the render tree. When the renderer is created and added to the tree, it does not have a position and size. The process of calculating these values is called layout or reflow. This means giving each node the exact coordinates where it should appear on the screen. The position of the root renderer is 0,0 and its dimensions are the viewport–the visible part of the browser window. All renderers have a “layout” or “reflow” method, each renderer invokes the layout method of its children that need layout.

The next stage is painting. In the painting stage, the render tree is traversed and the renderer’s “paint()” method is called to display content on the screen. Painting uses the UI backend layer. The rendering engine always tries to display the contents on the screen as soon as possible for better user experience. It does not wait for the HTML parsing to complete before starting to build and layout the render tree. It parses and displays the content it has received from the network, while rest of the contents stills keeps coming from the network.

**The order of processing scripts and style sheets:**

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.

*Speculative parsing:*

Both WebKit and Firefox do this optimization. While executing scripts, another thread parses the rest of the document and finds out what other resources need to be loaded from the network and loads them. In this way, resources can be loaded on parallel connections and overall speed is improved. Note: the speculative parser only parses references to external resources like external scripts, style sheets and images: it doesn't modify the DOM tree - that is left to the main parser.

*Style sheets:*

Style sheets on the other hand have a different model. Conceptually it seems that since style sheets don't change the DOM tree, there is no reason to wait for them and stop the document parsing. There is an issue, though, of scripts asking for style information during the document parsing stage. If the style is not loaded and parsed yet, the script will get wrong answers and apparently this caused lots of problems. It seems to be an edge case but is quite common. Firefox blocks all scripts when there is a style sheet that is still being loaded and parsed. WebKit blocks scripts only when they try to access certain style properties that may be affected by unloaded style sheets.

**Steps for what happens when we enter a URL :**

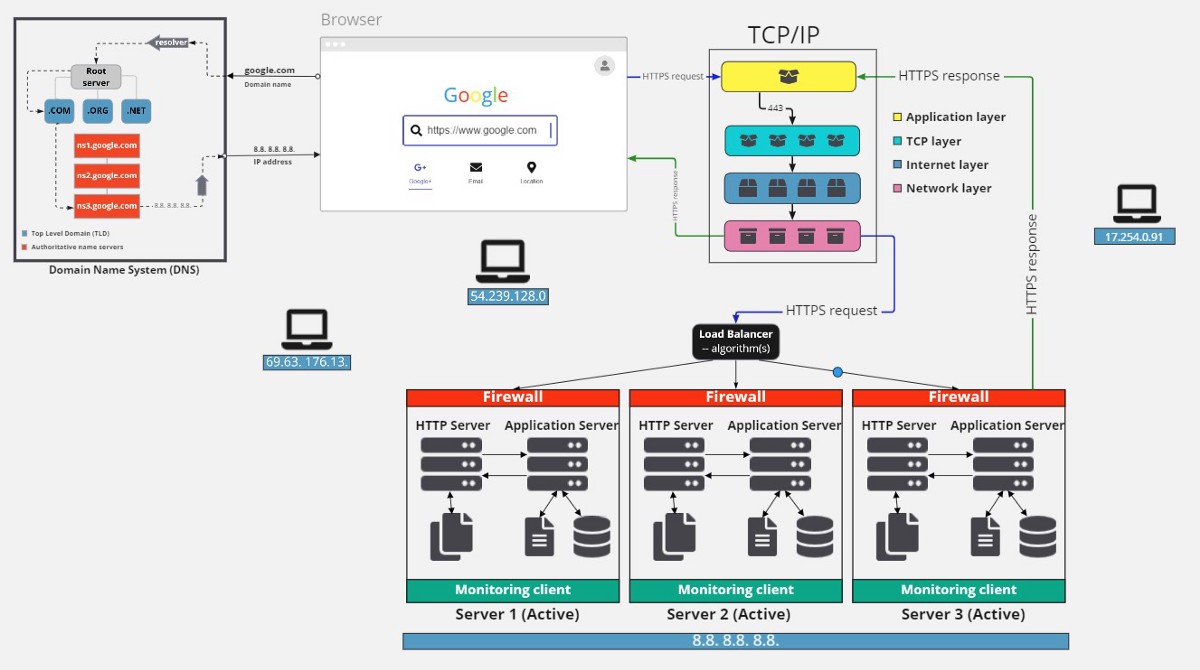
****

Fig.: HowBrowser fetch data over web.

1. Browser checks cache for DNS entry to find the corresponding IP address of website. It looks for following cache. If not found in one, then continues checking to the next until found.

* Browser Cache
* Operating Systems Cache
* Router Cache
* ISP Cache Browser

1. If not found in cache, ISP’s (Internet Service Provider) DNS server initiates a DNS query to find IP address of server that hosts the domain name. The requests are sent using small data packets that contain information content of request and IP address it is destined for.
2. Browser initiates a TCP (Transfer Control Protocol) connection with the server using synchronize(SYN) and acknowledge(ACK) messages.
3. Browser sends an HTTP request to the web server. GET or POST request.
4. Server on the host computer handles that request and sends back a response. It assembles a response in some format like JSON, XML and HTML.
5. Server sends out an HTTP response along with the status of response.
6. Browser displays HTML content
7. Finally Done.