***TASK -1***

1. **HTTP version history**

|  |  |
| --- | --- |
| **Year** | **HTTP Version** |
| 1991 | 0.9 |
| 1996 | 1.0 |
| 1997 | 1.1 |
| 2015 | [2.0](https://en.wikipedia.org/wiki/HTTP/2) |
| Draft (2020) | [3.0](https://en.wikipedia.org/wiki/HTTP/3) |

The term [hypertext](https://en.wikipedia.org/wiki/Hypertext) was coined by [Ted Nelson](https://en.wikipedia.org/wiki/Ted_Nelson) in 1965 in the [Xanadu Project](https://en.wikipedia.org/wiki/Xanadu_Project), which was in turn inspired by [Vannevar Bush](https://en.wikipedia.org/wiki/Vannevar_Bush" \o "Vannevar Bush)'s 1930s vision of the microfilm-based information retrieval and management "[memex](https://en.wikipedia.org/wiki/Memex" \o "Memex)" system described in his 1945 essay "[As We May Think](https://en.wikipedia.org/wiki/As_We_May_Think)". [Tim Berners-Lee](https://en.wikipedia.org/wiki/Tim_Berners-Lee) and his team at [CERN](https://en.wikipedia.org/wiki/CERN) are credited with inventing the original HTTP, along with HTML and the associated technology for a web server and a text-based web browser. Berners-Lee first proposed the "WorldWideWeb(www)" project in 1989—now known as the [World Wide Web](https://en.wikipedia.org/wiki/World_Wide_Web). The first version of the protocol had only one method, namely GET, which would request a page from a server. The response from the server was always an HTML page.

The first documented version of HTTP was [**HTTP V0.9**](https://www.w3.org/pub/WWW/Protocols/HTTP/AsImplemented.html) (1991). [Dave Raggett](https://en.wikipedia.org/wiki/Dave_Raggett) led the HTTP Working Group (HTTP WG) in 1995 and wanted to expand the protocol with extended operations, extended negotiation, richer meta-information, tied with a security protocol which became more efficient by adding additional methods and [header fields](https://en.wikipedia.org/wiki/List_of_HTTP_header_fields).  Officially introduced and recognized HTTP V1.0 in 1996.

The HTTP WG planned to publish new standards in December 1995[]](https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol#cite_note-16) and the support for pre-standard HTTP/1.1 based on the then developing (called HTTP-NG) was rapidly adopted by the major browser developers in early 1996. End-user adoption of the new browsers was rapid. In March 1996, one web hosting company reported that over 40% of browsers in use on the Internet were HTTP 1.1 compliant. That same web hosting company reported that by June 1996, 65% of all browsers accessing their servers were HTTP/1.1 compliant. The HTTP/1.1 standard as defined was officially released in January 1997. Improvements and updates to the HTTP/1.1 standard were released in June 1999.

In 2007, the [HTTP Working Group](https://httpwg.org/) was formed, in part, to revise and clarify the HTTP/1.1 specification.

1. ***Difference between browser.js and node.js***

1.Both the browser and Node.js use JavaScript as their programming language.

2.Building apps that run in the browser is a completely different thing than building a Node.js application.

3. From the perspective of a frontend developer who extensively uses JavaScript, Node.js apps bring with them a huge advantage: the comfort of programming everything - the frontend and the backend - in a single language.

4. In the browser, most of the time what you are doing is interacting with the DOM, or other Web Platform APIs like Cookies. Those do not exist in Node.js, of course. You don't have the **document, window** and all the other objects that are provided by the browser.

5. And in the browser, we don't have all the nice APIs that Node.js provides through its modules, like the filesystem access functionality.

6. You can use Babel to transform your code to be ES5-compatible before shipping it to the browser, but in Node.js, you won't need that.

7. Another difference is that Node.js uses the CommonJS module system, while in the browser we are starting to see the ES Modules standard being implemented.

8. In practice, this means that for the time being you use **require()** in Node.js and **import** in the browser.

***3. What happens when you type a URL in the address bar in the browser***

1. The browser **parses the URL** to find the protocol, host, port, and path.

2. It **forms a HTTP request** (that was most likely the protocol)

3. It look for the host to be **present in host file** of the system.

4. If present in host file it is redirected to the host listed like localhost to 127.0.0.1 which is your own machine.

5. To reach the host, it first needs to **translate**the human readable host**into an IP number**, and it does this by doing a DNS lookup on the host

6. Then a **socket needs to be opened** from the user’s computer to that IP number, on the port specified (most often port 80)

7.When a connection is open, the **HTTP request is sent** to the host

8.The host **forwards the request** to the server software (most often Apache or nginx) configured to listen on the specified port

9.The **server inspects the request** (most often only the path), and **launches the server plugin needed** to handle the request (corresponding to the server language you use, PHP, Java, .NET, Python?)

10.The plugin gets access to the full request, and starts to prepare a HTTP response.

11.To construct the response a **database**is (most likely) **accessed**. A database search is made, based on parameters in the path (or data) of the request

12.Data from the database, together with other information the plugin decides to add, is **combined into a long string** of text (probably HTML, json in case of apis).

13.The plugin **combines**that data with some meta data (in the form of HTTP headers), and **sends the HTTP response** back to the browser.

14.The browser receives the response, and **parses the HTML** (which with 95% probability is broken) in the response

15.A **DOM tree is built** out of the broken HTML

**16.New requests are made** to the server for each new resource that is found in the HTML source (typically images, style sheets, and JavaScript files). Go back to step 3 and repeat for each resource.

**17.Stylesheets are parsed**, and the rendering information in each gets attached to the matching node in the DOM tree

**18.Javascript is parsed and executed**, and DOM nodes are moved and style information is updated accordingly

The browser **renders the page** on the screen according to the DOM tree and the style information for each node

***4.*** ***Difference between http1.1 and 2.0***

To contextualize the specific changes that HTTP/2 made to HTTP/1.1, let’s first take a high-level look at the historical development and basic workings of each.

**HTTP/1.1**

Developed by Timothy Berners-Lee in 1989 as a communication standard for the World Wide Web, HTTP is a top-level application protocol that exchanges information between a client computer and a local or remote web server. In this process, a client sends a text-based request to a server by calling a *method* like GET or POST. In response, the server sends a resource like an HTML page back to the client.

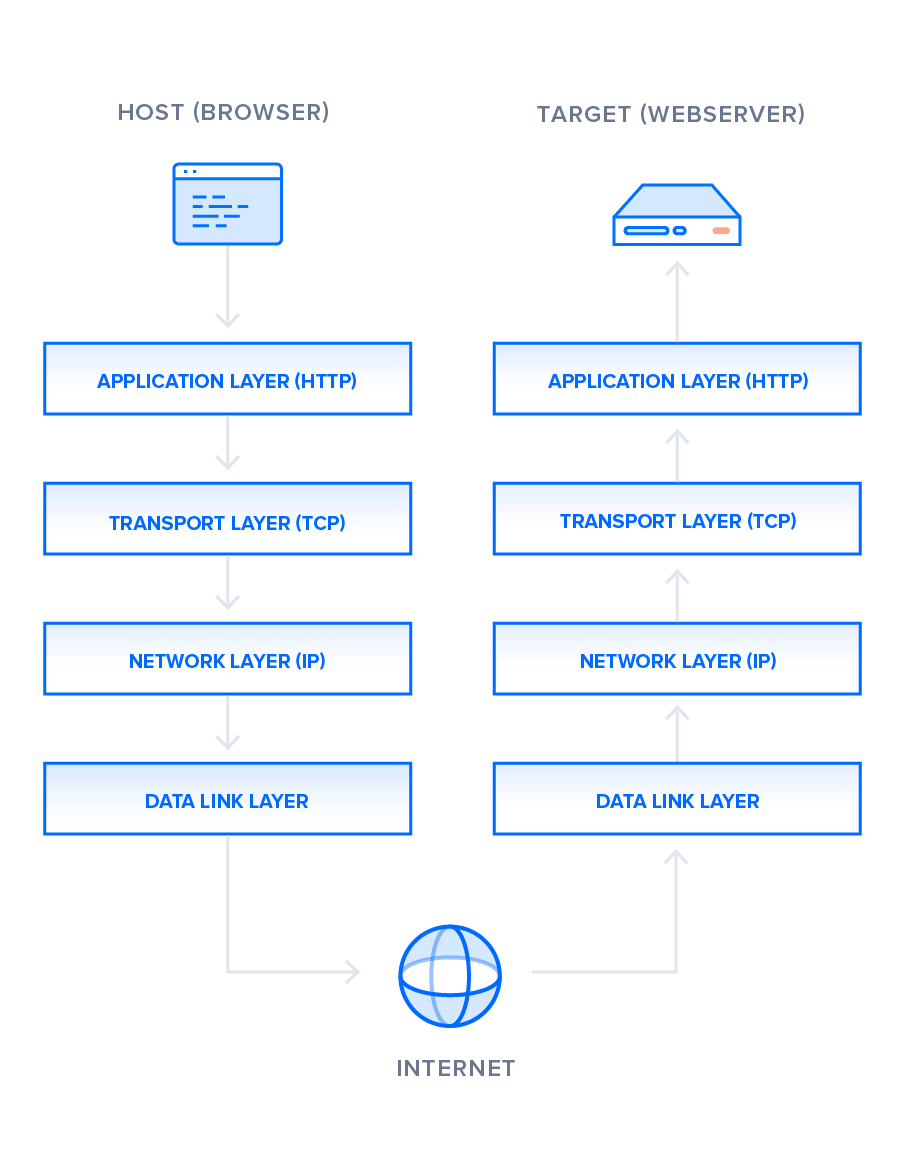
For example, let’s say you are visiting a website at the domain www.example.com. When you navigate to this URL, the web browser on your computer sends an HTTP request in the form of a text-based message, similar to the one shown here:

GET /index.html HTTP/1.1

Host: www.example.com

This request uses the GET method, which asks for data from the host server listed after Host:. In response to this request, the example.com web server returns an HTML page to the requesting client, in addition to any images, stylesheets, or other resources called for in the HTML. Note that not all of the resources are returned to the client in the first call for data. The requests and responses will go back and forth between the server and client until the web browser has received all the resources necessary to render the contents of the HTML page on your screen.

You can think of this exchange of requests and responses as a single *application layer* of the internet protocol stack, sitting on top of the *transfer layer* (usually using the Transmission Control Protocol, or TCP) and *networking layers* (using the Internet Protocol, or IP):



There is much to discuss about the lower levels of this stack, but in order to gain a high-level understanding of HTTP/2, you only need to know this abstracted layer model and where HTTP figures into it.

With this basic overview of HTTP/1.1 out of the way, we can now move on to recounting the early development of HTTP/2.

**HTTP/2**

HTTP/2 began as the SPDY protocol, developed primarily at Google with the intention of reducing web page load latency by using techniques such as compression, multiplexing, and prioritization. This protocol served as a template for HTTP/2 when the Hypertext Transfer Protocol working group httpbis of the [IETF (Internet Engineering Task Force)](https://www.ietf.org/) put the standard together, culminating in the publication of HTTP/2 in May 2015. From the beginning, many browsers supported this standardization effort, including Chrome, Opera, Internet Explorer, and Safari. Due in part to this browser support, there has been a significant adoption rate of the protocol since 2015, with especially high rates among new sites.

From a technical point of view, one of the most significant features that distinguishes HTTP/1.1 and HTTP/2 is the binary framing layer, which can be thought of as a part of the application layer in the internet protocol stack. As opposed to HTTP/1.1, which keeps all requests and responses in plain text format, HTTP/2 uses the binary framing layer to encapsulate all messages in binary format, while still maintaining HTTP semantics, such as verbs, methods, and headers. An application level API would still create messages in the conventional HTTP formats, but the underlying layer would then convert these messages into binary. This ensures that web applications created before HTTP/2 can continue functioning as normal when interacting with the new protocol.

The conversion of messages into binary allows HTTP/2 to try new approaches to data delivery not available in HTTP/1.1, a contrast that is at the root of the practical differences between the two protocols. The next section will take a look at the delivery model of HTTP/1.1, followed by what new models are made possible by HTTP/2.