EVOLUTION OF HTTP:

HTTP/0.9: *(One-Line Protocol)*

- It is a very simple protocol.
- It has only GET method in it.
- Only Html files were transmitted.
- Request is a single line and response consists only the file.
    - Request: GET/ path to resource
    - Response: File.
- There were no status codes or error codes, thus when a problem arises the Html file was sent back with the description of the problem.

HTTP/1.0: *(Building Extensibility)*

- Version information and Headers that contained metadata were added along with the request.
- Status codes, status messages and also Headers were added along with the response.
- Files other than Html were able to be transmitted.
- Request:

  GET/path to resource Version(e.g.: HTTP/1.0)
  Header with meta-data.
- Response:

  Status Code Status Message (e.g.: 200 OK)
  Headers with Meta-data
  File

HTTP/1.1: *(Standardized Protocol)*

- First Standardized protocol and was used for 15years.
- Connection can be re-used saving time in reopening the document.
- Pipelining is introduced, such that the subsequent requests could be sent without receiving the response for the previous request. Therefore, latency gets reduced
- Chunked responses were supported also additional cache control mechanisms were implemented.
- Content negotiation stuffs like language, encoding etc., were introduces for the client and server to agree on common grounds.
- Host Header was formulated to host different domains using same IP address.
- Request:

  GET/ path to resource/ Version.
  Host Header.
  Header with meta-data including content negotiation .
- Response:

  Status Code Status Message (e.g.: 200 OK).
  Headers with Meta-data including content negotiation.
  File.

HTTP 2.0: *(Greater performance protocol)*

- It is a Binary protocol and cannot be read or created manually, however improved optimization techniques can be employed.
- It is a multiplexed protocol thereby parallel connections can be established over the same connection.
- Headers are compressed saving the duplication of data.
- It populates data in client cache in advance, using server push mechanism

Post-HTTP/2 :

Alt-Svc header allows smart caching mechanism in Content Delivery Networks.

Client-Hints allow the communication between client or browser with the servers about the requirements or constraints.

Security cookies (cookie with security related prefixes) makes sure that cannot be altered.

DIFFERENCE BETWEEN HTTP/1.1 and HTTP/2.0:

HTTP/1.1	HTTP/2.0
Text based Protocol- can **be manually read and created**	Binary Protocol- **cannot** be manually read and created
Connections need requests sent in the correct order.	Connections need requests sent in the **any order.(Order unspecific)**
Conventional Headers are used.	Compresses header, removing duplicates.
Conventional mode of messages.	Encapsulated HTTP messages into **Frames**
Pipelining **and persistent connections: the TCP connection was** partially controlled **using the** Connection header**.**	**Multiplexing** messages **over a single connection**, helping keep the connection warm and more efficient.

DIFFRENCE BETWEEN BROWSER JS & NODE.JS

BROWSER JAVASCRIPT	NODE.JS
Its is used in Frontend for Client-side applications.	It is used in Backend for Server-side applications
It is sandboxed and have access limited to the browser.	It has full system access including access to file system like any other application .
It has objects like Window(draws out window), Document(render anything on a page) and Location(page specific-depends on URL).	These objects are missing as they are browser specific properties.
These objects are not available in browsers as they are server and system specific.	It has objects like Global(required in server-side works) and Require(used in modules in app).

They are GUI **dependent**	They are GUI **independent** or **Headless**
Not mandatory to keep everything in a module	Mandatory to modularize(Keep everything in a module) everything.
Runs in any JS engine including Spidermonkey,V8, Chakra or Nitro and can use any browsers like Chrome, Safari, Firefox, or Edge.	Runs only in V8 JS engine and is Chrome specific.

ABSTRACT WORKING OF JAVASCRIPT ENGINE-V8:

When the JavaScript file comes into V8 engine, it is first fed into the Parser.

This parser breaks the entire code into tokens based on lexical analysis. The tokens broken are called as in structures known as ASTs or Abstract Syntax Tree. This helps in understanding the meaning of the entire JavaScript code.

The tokens are fed into Interpreter called as Ignition and they are translated into byte code.

In general, there are compilers and Interpreters. The Interpreters translates line by line into Byte code and thus they are quicker in action. Whilst the Compiler scans the full code and then translates them into machine code and thus, it is more optimized.

The Profiler is like a monitor, it takes notes on how to optimize the code when the interpreter is converting line by line. When it finds some piece of code that can be optimized, it sends it to the compiler.

The compiler comes up with the optimized code and is known as Turbofan in V8 engine.

This optimized code will replace the bye code and thus, it will run faster.

WHAT HAPPENS WHEN YOU TYPE URL IN THE ADDRESS BAR?

The URL is typed in a web-browser.

The DNS converts the host address into IP address through a process called as Forward-Lookup where it searches the host address in a hierarchical list of servers.

In addition to this, it also adds up port number depending on the protocol. (http – 80 and https - 443)

It then initiates a TCP connection using 3 way handshake method(sync, sync ack and ack) and sends the HTTP request to the server where GET is the default method that jumps in, and the method searches for index.html file by default.

Also, when the connection is established, the Destination port is set as 80 and Source port is set with a random number.

Once the request has reached the server, the server sees the port 80 and identifies the HTTP & TCP protocol and sources its response as a HTTP response.

After this, the port numbers get exchanged and now the source port is 80 and destination port is the random number.

When the Browser receives the response, it identifies the port no 80 as the self-originated request. Thus, it accepts the response, and the information reaches the browser.