Day 1 Assignment

Question 1 : Write a blog on Difference between HTTP1.1 vs HTTP2

Answer :

**Protocol Type**

HTTP/1.1: It is a text-based protocol, which means that requests and responses are encoded in human-readable format. This design choice makes HTTP/1.1 easier to debug and understand for developers but less efficient for parsing and network transmission.

HTTP/2: Utilizes a binary protocol, framing HTTP messages in a way that is not directly human-readable but significantly more efficient for computers to parse and generate. This binary format reduces overhead and improves performance, particularly for complex web applications.

**Connection Management**

HTTP/1.1: Each request requires a separate TCP connection or uses multiple connections for concurrent requests, leading to increased latency due to the overhead of establishing connections and the limitations on the number of concurrent connections a browser can make to a server.

HTTP/2: Introduces multiplexing over a single TCP connection, allowing multiple requests and responses to be in flight simultaneously. This significantly reduces the overhead associated with TCP connections and eliminates the need for multiple connections to improve loading times.

**Compression**

HTTP/1.1: Headers, which can contain a significant amount of repetitive text, are sent uncompressed. This adds unnecessary overhead to each request and response, particularly for web applications that make numerous API calls.

HTTP/2: Implements HPACK compression for headers, significantly reducing the size of request and response headers. Since headers often contain a lot of redundant information, this compression can lead to substantial bandwidth savings.

**Multiplexing and Stream Prioritization**

HTTP/1.1: Does not support multiplexing. Requests are processed sequentially, which can lead to head-of-line blocking, where the request for a large or slow-loading resource delays subsequent requests.

HTTP/2: Supports multiplexing, allowing multiple requests and responses to be interwoven on the same connection. It also introduces stream prioritization, giving clients the ability to suggest priorities for resource loading, which helps browsers load critical resources first to improve user experience.

**Server Push**

HTTP/1.1: There is no mechanism for a server to proactively send resources to a client before they are explicitly requested, potentially delaying the loading of resources until the client realizes they are needed.

HTTP/2: Features server push, where servers can preemptively send resources to a client they anticipate the client will request soon after. This can reduce the round-trip time (RTT) necessary for loading certain resources, though it requires careful management to avoid sending unneeded data.

**Security**

HTTP/1.1: Security via TLS is optional and not inherently part of the protocol.

HTTP/2: While encryption is not technically required by the HTTP/2 specification, in practice, the vast majority of HTTP/2 traffic is encrypted using TLS. This is largely due to browser vendors choosing to only support HTTP/2 over TLS, enhancing the overall security of web communications.

**Performance Implications**

HTTP/1.1: Faces challenges with head-of-line blocking and requires multiple connections for parallelism, which increases latency and decreases page load performance, especially on high-latency networks.

HTTP/2: Designed to address these performance bottlenecks through features like multiplexing, header compression, and server push, leading to more efficient use of network resources and faster page loads.

The evolution from HTTP/1.1 to HTTP/2 represents a significant step forward in the efficiency and performance of web communications. By addressing the limitations of HTTP/1.1, HTTP/2 enables a more responsive, secure, and user-friendly web. As the web continues to grow in complexity and user expectations for speed and security increase, the adoption of HTTP/2 and further advancements in web protocols will play a crucial role in shaping the future of the internet.

Question 2 : Write a blog about objects and its internal representation in Javascript

Answer :

In JavaScript, an object is a standalone entity, with properties and type. Think of it as a container for values in a key:value format. These values can be properties and functions (methods of the object). Objects in JavaScript are dynamic, meaning they can be modified after their creation, allowing for the addition or removal of properties.

**Creating Objects**

Objects can be created in JavaScript using object literals, the new Object() syntax, or using constructor functions and the class syntax introduced in ES6.



**Internal Representation of Objects**

Internally, JavaScript engines (like V8 used in Chrome and Node.js) represent objects in a way that optimizes for performance. While the exact representation can vary between different JavaScript engines, some common strategies can be found in many of them.

**Hidden Classes**

Contrary to languages with class-based objects, JavaScript objects are shapeless when they are created. To optimize access to object properties, JavaScript engines use a concept called "hidden classes" (also known as "shapes" in some engines). Hidden classes are internal constructs that represent the layout of an object; they map the object's properties to their values.

When a new object is created, it starts with no properties, associated with a generic hidden class. As properties are added to the object, the JavaScript engine transitions the object to new hidden classes that represent the object’s current shape. This allows property access to be as fast as accessing array indexes.

**Inline Caching**

Inline caching is a performance optimization used by JavaScript engines to speed up property access. When a property of an object is accessed, the engine keeps track of the hidden class of that object. If the same property is accessed again and the object has not changed its hidden class, the engine can bypass some of the lookup steps, significantly speeding up property access.

**Property Storage**

For objects with a small number of properties, engines often store property values directly in the object structure (in-place storage). However, for objects with a large number of properties or properties that are frequently added and removed, engines may use a more flexible but less efficient storage model, such as hash tables or similar structures.