**Day-01 Task**

**1. Write a blog on Difference between HTTP1.1 vs HTTP2**

**Introduction:**

In the ever-evolving landscape of the internet, protocols play a major role in determining how data is transmitted between clients and servers. One of the most fundamental protocols is **Hypertext Transfer Protocol (HTTP)**, which facilitates the transfer of information on the World Wide Web. Over time, HTTP has undergone significant advancements to keep up with the growing demands of modern web applications. Two major versions, HTTP/1.1 and HTTP/2, stand out as significant milestones in this evolutionary journey. In this blog, we delve into the key differences between HTTP/1.1 and HTTP/2, exploring how they have revolutionized web communication.

**The Foundation of Modern Web (HTTP/1.1)**

Introduced in 1997, laid the groundwork for the modern web as we know it today. While revolutionary at its inception, HTTP/1.1 eventually started showing signs of limitations as web technologies advanced. One of the major drawbacks of HTTP/1.1 is its lack of efficiency in handling multiple simultaneous requests. Each request-response cycle in HTTP/1.1 operates sequentially, leading to a phenomenon known as the "head-of-line blocking," where subsequent requests must wait for earlier requests to complete, hindering overall performance.

**A Paradigm Shift in Web Communication (HTTP/2)**

Recognizing the need for a more efficient protocol, HTTP/2 was developed and standardized in 2015. Built upon the foundation of SPDY, an experimental protocol developed by Google, HTTP/2 introduces several groundbreaking features aimed at improving web performance and efficiency.

**Multiplexing:** Parallelism Unleashed

One of the most significant enhancements in HTTP/2 is multiplexing, which allows multiple requests and responses to be sent and received in parallel over a single TCP connection. Unlike HTTP/1.1, where each request had to wait for its turn, HTTP/2 enables interleaving of frames from different streams, thereby eliminating head-of-line blocking and significantly reducing latency.

**Header Compression:** Trim the Excess

In HTTP/1.1, headers are sent with each request and response, often resulting in redundant data transmission. HTTP/2 addresses this inefficiency through header compression, using a technique called HPACK. By compressing headers, HTTP/2 reduces overhead and conserves bandwidth, leading to faster and more efficient communication between clients and servers.

**Server Push:** Anticipating Needs

Another noteworthy feature of HTTP/2 is server push, which allows servers to proactively send resources to the client's cache before they are requested. This pre-emptive approach can significantly reduce latency by eliminating the need for additional round trips to fetch dependent resources, such as stylesheets, scripts, and images.

**Performance Comparison: HTTP/1.1 vs. HTTP/2**

The performance improvements brought by HTTP/2 are substantial and have been well-documented through various benchmarks and case studies. Studies have shown that websites adopting HTTP/2 experience faster page load times, reduced latency, and improved overall user experience compared to those relying on HTTP/1.1. The benefits are particularly pronounced for websites with numerous assets and resources, such as images, scripts, and stylesheets.

**Adoption and Compatibility**

Despite its numerous advantages, the adoption of HTTP/2 was initially hindered by compatibility issues and concerns over migration complexities. However, with widespread support from major web servers, browsers, and content delivery networks (CDNs), HTTP/2 adoption has gained momentum over the years. Today, most modern web browsers and servers support HTTP/2 by default, making it increasingly accessible to web developers and users alike.

**Conclusion**

In conclusion, the transition from HTTP/1.1 to HTTP/2 represents a significant milestone in the evolution of web communication. With its groundbreaking features such as multiplexing, header compression, and server push, HTTP/2 addresses many of the performance bottlenecks inherent in its predecessor. While HTTP/1.1 laid the foundation for the modern web, HTTP/2 paves the way for faster, more efficient, and more responsive web experiences. As the internet continues to evolve, protocols like HTTP/2 will play a crucial role in shaping the future of digital communication.

1. Write a blog about objects and its internal representation in Javascript

**2.Write a blog about objects and its internal representation in JavaScript**

**Introduction**

In the world of JavaScript, objects are fundamental. They form the building blocks of most applications, serving as containers for data in key-value pairs. Yet, understanding how objects are internally represented can be a significant advantage for developers. In this blog, we'll delve into the internals of JavaScript objects, shedding light on their representation and behaviour.

At its core, a JavaScript object is a collection of properties. These properties consist of keys and corresponding values. The keys are strings (or Symbols) that serve as unique identifiers, while the values can be of any data type: primitive or reference.

**Example:**

const person = {

name: 'Ram',

age: 25,

city: 'Bangalore’

};

In this example, **person** is an object with three properties: **name**, **age**, and **city**, each with its associated value.

**Object Representation**

JavaScript engines like V8 (used in Chrome and Node.js) implement objects using various data structures internally. One common representation is a dictionary-like structure, where properties are stored as key-value pairs. This allows for fast property access, insertion, and deletion.

In some cases, engines optimize object representation based on usage patterns. For instance, if an object has only numeric keys and behaves like an array, the engine may internally optimize its representation to behave more like an array.

**Primitive Values vs. Reference Values**

**Primitive Values:**

These are immutable and stored directly in memory. Examples include strings, numbers, Booleans, null, and undefined.

**Example:** let num = 10;

**Reference Values:**

Objects and functions are reference values. Instead of containing the actual data, variables holding reference values store a pointer to the memory location where the data is stored.

**Example**: let obj = { key: 'value' };

**Object Identity and Equality**

In JavaScript, two distinct objects with identical properties are not considered equal due to their different memory addresses. Consider the following:

const obj1 = { key: 'value' };

const obj2 = { key: 'value' };

console.log(obj1 === obj2); // Output: false

Despite having the same properties, **obj1** and **obj2** are distinct objects in memory, hence the === comparison yields **false**.

**Mutable Nature of Objects**

Unlike primitive values, objects in JavaScript are mutable, meaning their properties can be altered even after creation.

const person = { name: 'Ram', age: 25 };

// Modifying property

person.age = 28;

// Adding new property

person.city = 'Bangalore';

// Deleting property

delete person.age;

**Conclusion**

Understanding the internal representation of objects in JavaScript can empower developers to write more efficient and effective code. By grasping the distinction between primitive and reference values, as well as the mutability of objects, developers can navigate the intricacies of JavaScript more adeptly. So next time you're working with objects in JavaScript, remember to consider not only their surface-level properties but also their underlying representation in memory.