**TASK1: SEE THE BLOG ON THE DIFFERENCES BETWEEN HTTP1.1 vs HTTP2**

horizontal line

# Placeholder image

# Introduction

**In the realm of web communication, Hypertext Transfer Protocol (HTTP) is the backbone that enables data exchange between clients and servers. With the increasing demand for faster and more efficient web browsing experiences, HTTP has evolved over time. Two significant versions of HTTP that have played a crucial role in this evolution are HTTP/1.1 and HTTP/2. In this blog, we will explore the key differences between these two protocols and how HTTP/2 revolutionized web performance.**

**HTTP/1.1: The Foundation of Modern Web**

**HTTP/1.1, introduced in 1999, laid the groundwork for modern web communication. It is a text-based protocol that follows a request-response model, where a client sends a request to a server and waits for a response. Here are some notable features and limitations of HTTP/1.1:**

**MULTIPLEXING: In HTTP/1.1, each request/response connection requires a separate TCP connection, resulting in latency and overhead. This limitation hampers the performance of web applications that make multiple requests.**

**HEADER COMPRESSION: HTTP/1.1 does not compress headers, leading to increased bandwidth consumption. This becomes particularly problematic when dealing with large web pages with numerous assets.**

**SERVER PUSH: Server push is not supported in HTTP/1.1. Instead, the client has to explicitly request each asset required to render a web page.**

**HTTP/2: A Leap Towards Efficiency**

**HTTP/2, released in 2015, was designed to address the limitations of HTTP/1.1 and provide a more efficient protocol for modern web applications. It introduced several key features that significantly improved web performance:**

**MULTIPLEXING: One of the most significant advancements in HTTP/2 is multiplexing. It allows multiple requests and responses to be sent and received over a single TCP connection simultaneously. This eliminates the need for multiple connections, reduces latency, and improves overall throughput.**

**BINARY PROTOCOL: Unlike HTTP/1.1, which uses plain text, HTTP/2 employs a binary protocol. This change improves parsing efficiency and enables more compact representation of data.**

**HEADER COMPRESSION: HTTP/2 utilizes header compression using the HPACK algorithm. It reduces the overhead of headers by compressing them, resulting in reduced bandwidth consumption and faster page load times.**

**SERVER PUSH: HTTP/2 introduces server push, which enables the server to proactively push resources to the client without waiting for explicit requests. This eliminates the need for additional round trips, allowing faster rendering of web pages.**

**STREAM PRIORITIZATION: HTTP/2 supports stream prioritization, allowing the client to assign relative weights and dependencies to different resources. This enables the browser to render critical content first, leading to improved user experience.**

**Conclution**

**HTTP/2 represents a significant improvement over its predecessor, HTTP/1.1. The introduction of multiplexing, binary protocol, header compression, server push, and stream prioritization has revolutionized web performance. These enhancements have resulted in faster page load times, reduced latency, improved bandwidth utilization, and ultimately, a better browsing experience for users.**

**As the adoption of HTTP/2 continues to grow, it is important for web developers and server administrators to embrace this protocol to unlock its full potential. However, it's worth noting that HTTP/2 is not a silver bullet, and optimizing web applications to leverage its features requires careful consideration and implementation.**

**In conclusion, HTTP/2 has ushered in a new era of web communication, enabling faster, more efficient, and responsive websites. As technology continues to evolve, it's essential to stay updated with the latest advancements in web protocols to deliver optimal user experiences.**

**TASK2: SEE THE BLOG ABOUT OBJECTS AND IT’S INTERNAL REPRESENTATION IN JAVASCRIPT**

**Introduction:**

**JavaScript, as a versatile programming language, provides developers with various data structures to handle complex tasks. One such fundamental data structure is objects. Objects in JavaScript are powerful and flexible, allowing you to store and manipulate structured data. In this blog post, we will delve into the internal representation of objects in JavaScript, exploring their properties, methods, and underlying concepts.**

**Understanding Objects in JavaScript:**

**In JavaScript, an object is an unordered collection of key-value pairs, where each key acts as an identifier for its corresponding value. These values can be of any data type, including primitive types (such as numbers, strings, and booleans) and even other objects. The key-value pairs are commonly referred to as object properties.**

**Creating Objects:**

**There are multiple ways to create objects in JavaScript. The most straightforward method is using object literals, which involves enclosing key-value pairs within curly braces ({}) as shown below:**

**javascript**

**Copy code**

**const person = {**

**name: 'John Doe',**

**age: 25,**

**email: 'johndoe@example.com'**

**};**

**Internal Representation of Objects:**

**Internally, JavaScript objects are implemented using a mechanism known as a hash table (or hash map). A hash table is a data structure that allows for efficient key-value pair storage and retrieval. When an object is created, JavaScript automatically creates a hash table behind the scenes to manage its properties.**

**Hash tables employ a hashing function to compute an index for each key-value pair. This index is used to store and retrieve the associated value efficiently. The hashing function ensures that each key generates a unique index within the hash table, minimizing the chances of collisions (when two keys produce the same index).**

**Object Properties:**

**JavaScript objects can have two types of properties: own properties and prototype properties.**

**Own Properties:**

**Own properties are defined directly on the object itself. In the earlier example, name, age, and email are own properties of the person object. These properties are accessed using dot notation (object.property) or bracket notation (object['property']).**

**Prototype Properties:**

**Objects in JavaScript can also inherit properties from a prototype object. A prototype is essentially a blueprint or template for creating objects. If a property is not found in an object's own properties, JavaScript will look for it in the object's prototype chain. This allows for efficient memory usage, as multiple objects can share common properties through inheritance.**

**Working with Objects:**

**JavaScript provides several built-in methods and operators to work with objects effectively. Here are a few commonly used ones:**

**Accessing and Modifying Properties:**

**You can access and modify object properties using dot notation or bracket notation, as mentioned earlier. For example:**

**javascript**

**Copy code**

**console.log(person.name); // Output: 'John Doe'**

**person.age = 26;**

**console.log(person.age); // Output: 26**

**Adding and Removing Properties:**

**You can add new properties to an object dynamically by simply assigning a value to a new key. To remove a property, you can use the delete operator. For example:**

**javascript**

**Copy code**

**person.city = 'New York';**

**delete person.email;**

**Iterating over Properties:**

**To iterate over an object's properties, you can use the for...in loop or Object.keys() method. For example:**

**javascript**

**Copy code**

**for (let key in person) {**

**console.log(`${key}: ${person[key]}`);**

**}**

**const keys = Object.keys(person);**

**console.log(keys); // Output: ['name', 'age', 'city']**

**Conclusion:**

**Understanding the internal representation of objects in JavaScript is crucial for effective object manipulation and data management. JavaScript objects, implemented using hash tables, provide a flexible**