Day 01

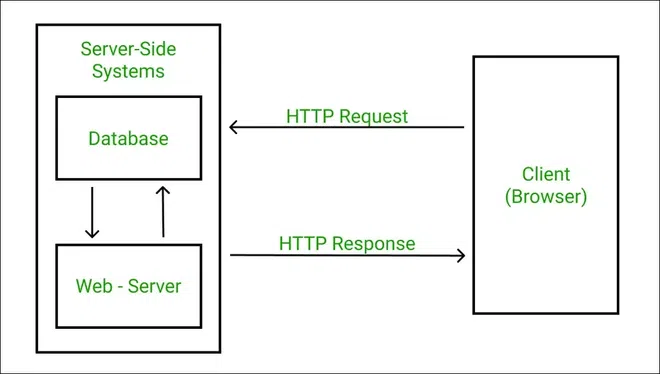
Date: 16/11/2023

Activity:

1. Write a blog on the Difference between HTTP1.1 and HTTP2
2. Write a blog about objects and its internal representation in JavaScript
3. Write a blog on the Difference between HTTP1.1 and HTTP2

**Hypertext Transfer Protocol:**

HTTP provides a standard between a web browser and a web server to establish communication. It is a set of rules for transferring data from one computer to another. Data such as text, images, and other multimedia files are shared on the World Wide Web



**HTTP/1.1:** Introduced in 1997, HTTP/1.1 is the foundational protocol for the World Wide Web, relying on a simple request-response model, but it can lead to latency issues due to multiple connections for parallel downloads.

**HTTP/2:** Introduced in 2015, HTTP/2 is a major revision aimed at addressing performance bottlenecks. It features multiplexing, allowing multiple requests and responses simultaneously over a single connection, leading to faster and more efficient web communication.

|  |  |  |
| --- | --- | --- |
| Description | HTTP1.1 | HTTP2 |
| Binary vs. Text | Uses a text-based protocol, readable for humans but less efficient for machines. | Adopts a binary protocol, improving machine efficiency and speeding up data transfer. |
| Connection Handling | It uses multiple connections for data transfer, leading to increased latency. | Consolidates data into a single connection, reducing latency and improving speed. |
| Multiplexing | Processes one request at a time on a connection. | Handles multiple requests simultaneously, making data transfer more efficient |
| Server Push | It doesn't allow servers to push data to the client without a specific request. | Supports server push, enabling proactive data transmission from the server. |
| Prioritization | Treats all requests equally, without explicit prioritization. | Prioritizes requests, ensuring more important resources are processed first. |
| Resource  Loading | Requires careful optimization of resource bundling. | Efficiently handles multiple resource requests without the need for complex bundling. |
| Backward Compatibility | Widely supported, but becoming outdated. | Growing in adoption, with increasing support from major websites and browsers. |
| Security | Lacks built-in encryption. | Encourages the use of secure connections, enhancing overall data security. |
| Adoption Rate | Still widely used. | Gaining popularity rapidly due to its performance improvements. |

HTTP/2 is like a streamlined and more efficient version of HTTP/1, designed to make the web faster and more responsive for users.

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

**Objects in JavaScript**

JavaScript is an object-based language, and objects are a fundamental data type in the language. Unlike primitive data types such as numbers and strings, objects allow developers to structure their code in a more complex and meaningful way. Objects in JavaScript are collections of key-value pairs, where keys are strings or symbols, and values can be of any data type, including other objects.

// Example of a simple JavaScript object

const car = {

make: 'Toyota',

model: 'Camry',

year: 2022,

isAutomatic: true,

};

In JavaScript, almost "everything" is an object.

* Booleans can be objects (if defined with the new keyword)
* Numbers can be objects (if defined with the new keyword)
* Strings can be objects (if defined with the new keyword)
* Dates are always objects
* Maths are always objects
* Regular expressions are always objects
* Arrays are always objects
* Functions are always objects
* Objects are always objects

All JavaScript values, except primitives, are objects.

JavaScript Objects are Mutable

Objects are mutable: They are addressed by reference, not by value.

If a person is an object, the following statement will not create a copy of the person:

const x = person; // Will not create a copy of a person.

**The internal representation of objects :**

in JavaScript involves a combination of properties and hidden classes within the underlying JavaScript engine. Let's simplify the concept:

**Properties:**

Objects in JavaScript consist of properties, which are essentially key-value pairs.

These properties can hold data of various types, including other objects.

const person = {

name: 'John Doe',

age: 30,

occupation: 'Developer', };

**Hidden Classes:**

JavaScript engines use a hidden class system to optimize object access and manipulation.

When an object is created, the engine assigns it a hidden class based on its structure.

Objects with the same structure share the same hidden class, enabling performance optimizations.

// Objects with similar structures share the same hidden class

const person1 = { name: 'John', age: 25 };

const person2 = { name: 'Jane', age: 30 };

The engine optimizes access to properties based on the hidden class, resulting in faster property access.

**Creating a JavaScript Object**

In JavaScript, there are several ways to declare and create objects. Each method has its own use case and syntax. Here are some common ways to declare objects in JavaScript:

**Object Literal:**

The most straightforward way to create an object is by using the object literal notation. This involves encapsulating key-value pairs inside curly braces {}.

const person = {

name: 'Thiru',

age: 24,

occupation: 'FSD Developer'

};

**Using new Keyword:**

The following example also creates a new JavaScript object with four properties:

function Person(first, last, age, eye) {

this.firstName = first;

this.lastName = last;

this.age = age;

this.eyeColor = eye;

}

// Create a Person object

const myFather = new Person("Prakash", "V S", 22, "Black");

**Object.create():**

The **Object.create()** method allows you to create a new object with the specified prototype object. This is useful for prototypal inheritance.

const person = new Object();  
person.firstName = "John";  
person.lastName = "Doe";  
person.age = 50;  
person.eyeColor = "blue";

**Constructor Function:**

You can create objects using constructor functions. This approach is particularly useful when you need to create multiple instances of similar objects.

function Person(name, age, occupation) {

this.name = name;

this.age = age;

this.occupation = occupation;

}

const person = new Person('John Doe', 30, 'Developer');

**Class Syntax (ES6 and later):**

With the introduction of ECMAScript 2015 (ES6), JavaScript supports class syntax, providing a more structured way to create objects and work with inheritance.

class Person {

constructor(name, age, occupation) {

this.name = name;

this.age = age;

this.occupation = occupation;

}

greet() {

console.log(`Hello, my name is ${this.name}`);

}

}

const person = new Person('John Doe', 30, 'Developer');

--X--- END---X--